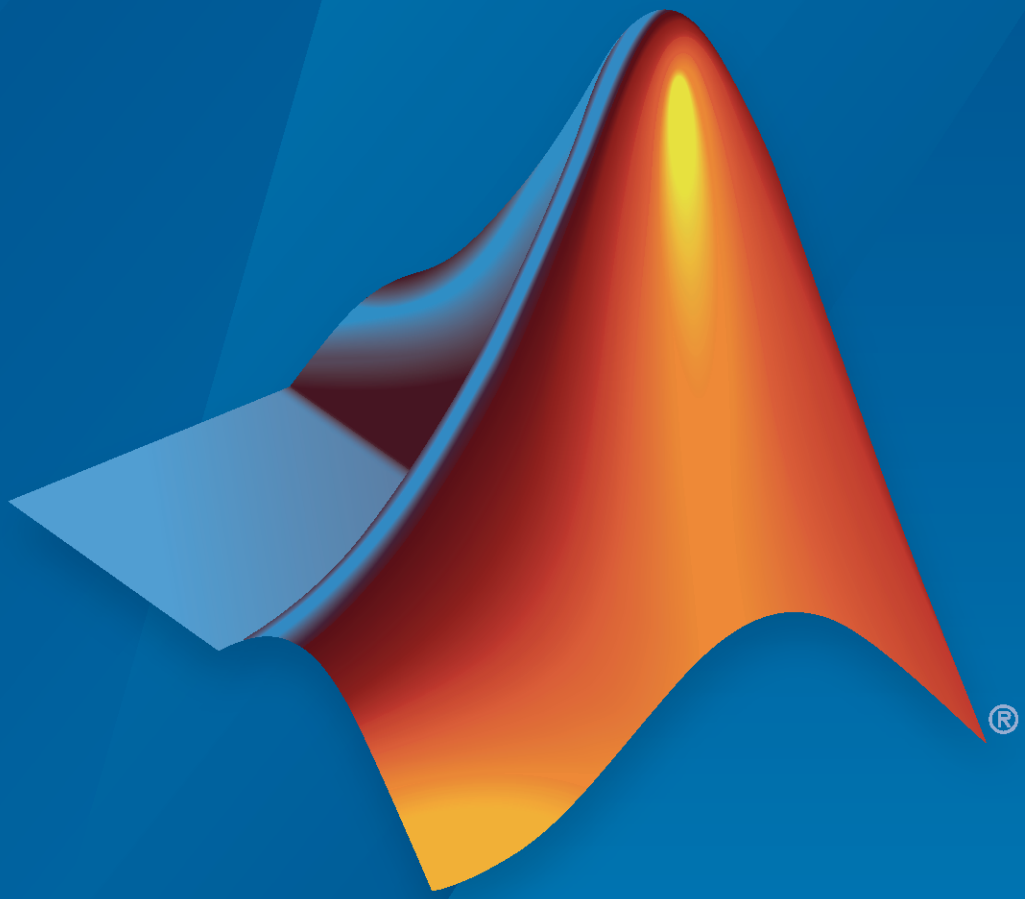
# LTE Toolbox™

Reference



MATLAB®

R2023a

MathWorks®

# How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

Technical support: `www.mathworks.com/support/contact_us`

Phone: 508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

*LTE Toolbox™ Reference*



**Revision History**

| | | |
|---|---|---|
| September 2013 | Online only | Revised for Version 1.0 (Release 2013b) |
| March 2014 | Online only | Revised for Version 1.1 (Release 2014a) |
| October 2014 | Online only | Revised for Version 1.2 (Release 2014b) |
| March 2015 | Online only | Revised for Version 2.0 (Release 2015a) |
| September 2015 | Online only | Revised for Version 2.1 (Release 2015b) |
| March 2016 | Online only | Revised for Version 2.2 (Release 2016a) |
| September 2016 | Online only | Revised for Version 2.3 (Release 2016b) |
| March 2017 | Online only | Revised for Version 2.4 (Release 2017a) |
| September 2017 | Online only | Revised for Version 2.5 (Release 2017b) |
| March 2018 | Online only | Revised for Version 2.6 (Release 2018a) |
| September 2018 | Online only | Revised for Version 3.0 (Release 2018b) |
| March 2019 | Online only | Revised for Version 3.1 (Release 2019a) |
| September 2019 | Online only | Revised for Version 3.2 (Release 2019b) |
| March 2020 | Online only | Revised for Version 3.3 (Release 2020a) |
| September 2020 | Online only | Revised for Version 3.4 (Release 2020b) |
| March 2021 | Online only | Revised for Version 3.5 (Release 2021a) |
| September 2021 | Online only | Revised for Version 3.6 (Release 2021b) |
| March 2022 | Online only | Revised for Version 3.7 (Release 2022a) |
| September 2022 | Online only | Revised for Version 3.8 (Release 2022b) |
| March 2023 | Online only | Revised for Version 3.9 (Release 2023a) |

# Contents

# Blocks

# Waveform From Wireless Waveform Generator App

Wireless waveform source exported to Simulink


**Libraries:**
None

## Description

The Waveform From Wireless Waveform Generator App block is generated using the **Wireless Waveform Generator** app. You can use the generated block as a wireless waveform source in a Simulink® model.

**Note** The actual block name and output waveform depend on the waveform that you configure in the app before generating the block.

For an overview of the waveform types that you can export to Simulink using the LTE Toolbox software, see the **LTE Waveform Generator** app.

To generate a block:

1. On the app toolstrip, in the **Waveform Type** section, click the waveform that you want to configure and export to Simulink.
2. Set the parameters of the selected waveform.
3. On the app toolstrip, in the **Export** section, click **Export** and select **Export to Simulink**.

The **Code** tab of the Mask Editor window contains the MATLAB® code that the block executes to output the configured waveform. To access read-only block parameters and waveform configuration parameters, use the `UserData` common block property, which is a structure with these fields.

- `WaveformConfig` — Waveform configuration parameters
- `WaveformLength` — Waveform length
- `Fs` — Waveform sample rate

For more information on how to use the generated block, see “Generate Wireless Waveform in Simulink Using App-Generated Block”.

## Limitations

With the exception of blocks that are generated for 5G NR waveforms, blocks that are generated using random user-defined signal data for the waveform do not support rapid accelerator mode. To enable rapid accelerator mode in these blocks when you set the **Bit-source** app parameter to `User-defined`, use pseudo-noise (PN) data as the data source.

## Ports

### Output

**wf** — Time-domain wireless waveform
complex matrix

Time-domain wireless waveform, returned as a complex matrix. The number of matrix columns corresponds to the number of transmit antennas. The waveform type you select in the app determines the output waveform type. To access waveform configuration parameters, use the `WaveformConfig` structure field of the `UserData` common block property.

Data Types: `double`

## Parameters

### Read-Only Waveform Parameters

The block automatically updates these parameters based on the waveform configuration in the **Code** tab.

**Waveform sample rate (Fs)** — Waveform sample rate
numeric scalar

This parameter is read-only.

To access this parameter, use the `Fs` structure field of the `UserData` common block property. Units of the `Fs` structure field are in Hz.

**Waveform length** — Waveform length
positive integer

This parameter is read-only.

To access this parameter, use the `WaveformLength` structure field of the `UserData` common block property. Units of the `WaveformLength` structure field are in samples.

### Simulation Parameters

These parameters control how the block outputs the waveform during simulation.

**Samples per frame** — Samples per frame
1 (default) | positive integer

This parameter specifies the number of samples to buffer into each output frame.

**Form output after final data value by** — Output values after last waveform sample
`Cyclic repetition` (default) | `Setting to zero`

This parameter specifies the output values after the block has output all available waveform samples.

- When you select `Cyclic Repetition`, the block repeats the waveform from the beginning after reaching the last sample in the waveform.
- When you select `Setting To Zero`, the block generates zero-valued outputs for the duration of the simulation after generating the last frame of the waveform.

# Version History

**Introduced in R2021b**

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

### Apps

**LTE Waveform Generator**

# Functions

# displayChannel

Visualize and explore 3-D MIMO fading channel model characteristics

## Syntax

```
fig = displayChannel(lte3D)
fig = displayChannel(lte3D,Name,Value)
```

## Description

`fig = displayChannel(lte3D)` displays geometric and electromagnetic characteristics of the specified 3-D multiple-input/multiple-output (MIMO) channel model at the transmitter and receiver ends. The visualization includes the position, polarization, and directivity radiation pattern of the antenna elements, cluster paths directions, and average path gains. Because all antenna elements are equal, the visualization shows the radiation pattern of the first antenna element only and displays the cluster paths directions centered also at the first antenna element. By adding customized data tips to the visualization windows, you can explore antenna element, element pattern, and cluster paths characteristics. The function also returns an array of figure objects that correspond to the displayed visualization windows.

`fig = displayChannel(lte3D,Name,Value)` specifies visualization options of the displayed channel characteristics by using one or more name-value pair arguments. For example, `'LinkEnd','Tx'` specifies visualization at the transmitter end only.

## Examples

### Visualize Channel Characteristics

This example shows how to visualize 3-D channel characteristics and explore channel information about the antenna element, element pattern, and cluster paths.

Define the channel configuration by using an `lte3DChannel` System object. Specify the delay profile as CDL-D.

```
lte3D = lte3DChannel.makeCDL('CDL-D');
```

Configure the transmit array size as a vector of the form $\left[M\,N\,P\,M_g\,N_g\right]$ = [4 3 2 1 2], which specifies two rectangular panels ($M_g$ = 1 and $N_g$ = 2) of a 4-by-3 antenna array ($M$ = 4 and $N$ = 3) and two polarizations ($P$ = 2). The total number of polarized elements in the array is $M \times N \times P \times M_g \times N_g$ = 48.

```
txSize = [4 3 2 1 2];
lte3D.TransmitAntennaArray.Size = txSize;
```

Configure the vertical and horizontal element spacing and the vertical and horizontal panel spacing, in wavelength, as a vector of the form $[\lambda_v\,\lambda_h\,\mathrm{dg}_v\,\mathrm{dg}_h]$. Because panel spacing is measured from the center of the panels, to avoid panel overlapping, set $\mathrm{dg}_h$ to a value greater than one wavelength. To

ensure uniform antenna element spacing across vertically and horizontally separated panels, configure panel spacings as $dg_v = \lambda_v \times M$ and $dg_h = \lambda_h \times N$, respectively.

```
lambda_v = 0.5;
lambda_h = 0.5;
dg_v = lambda_v*txSize(1); % lambda_v * M
dg_h = lambda_h*txSize(2); % lambda_h * N
lte3D.TransmitAntennaArray.ElementSpacing = [lambda_v lambda_h dg_v dg_h];
```

Configure the mechanical orientation of the array as $\left[\alpha\ \beta\ \gamma\right]^T = [0\ 15\ 0]^T$, which specifies 0 degrees bearing, 15 degrees downtilt, and 0 degrees slant.

```
lte3D.TransmitAntennaArray.Orientation = [0 15 0]';
```

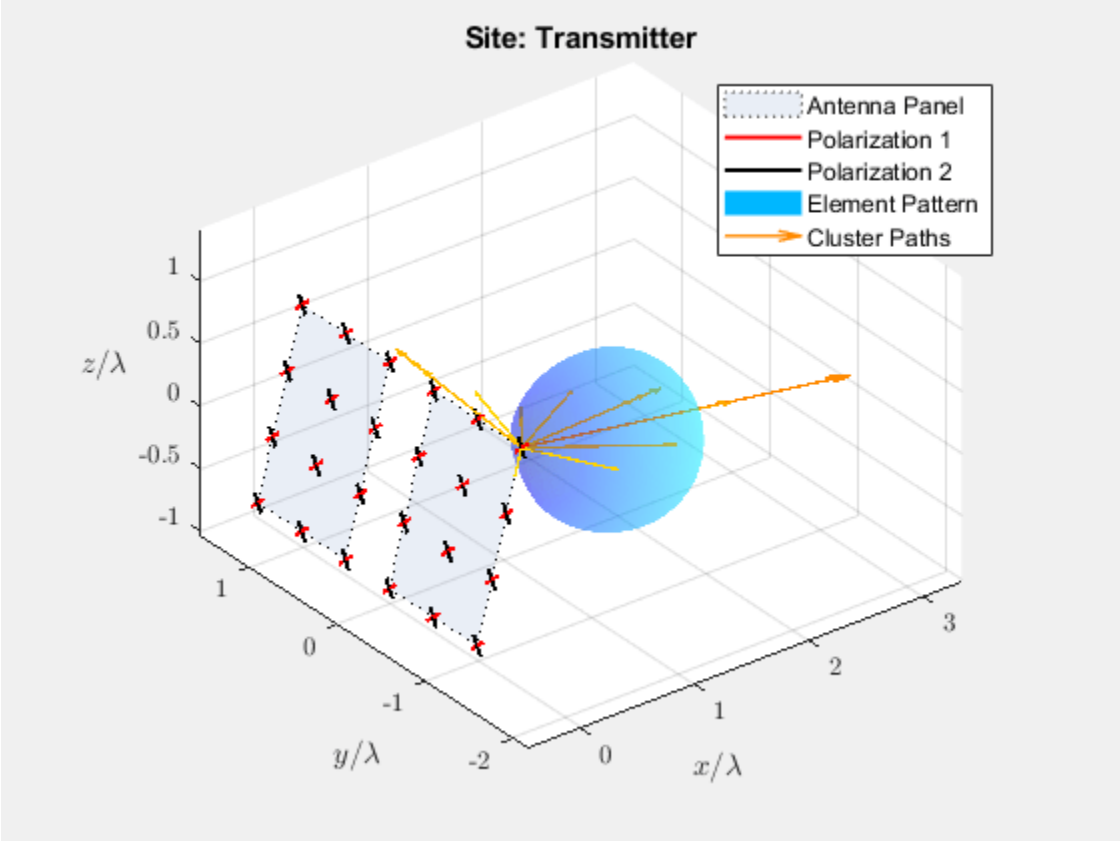
For an overview of all transmit antenna array properties, see the `TransmitAntennaArray` property of the `lte3DChannel` System object.

Display the channel characteristics at the transmitter end.

```
figTx = displayChannel(lte3D,'LinkEnd','Tx');
```

The generated figure supports customized data tips. Add data tips in the current figure by enabling the data cursor mode.

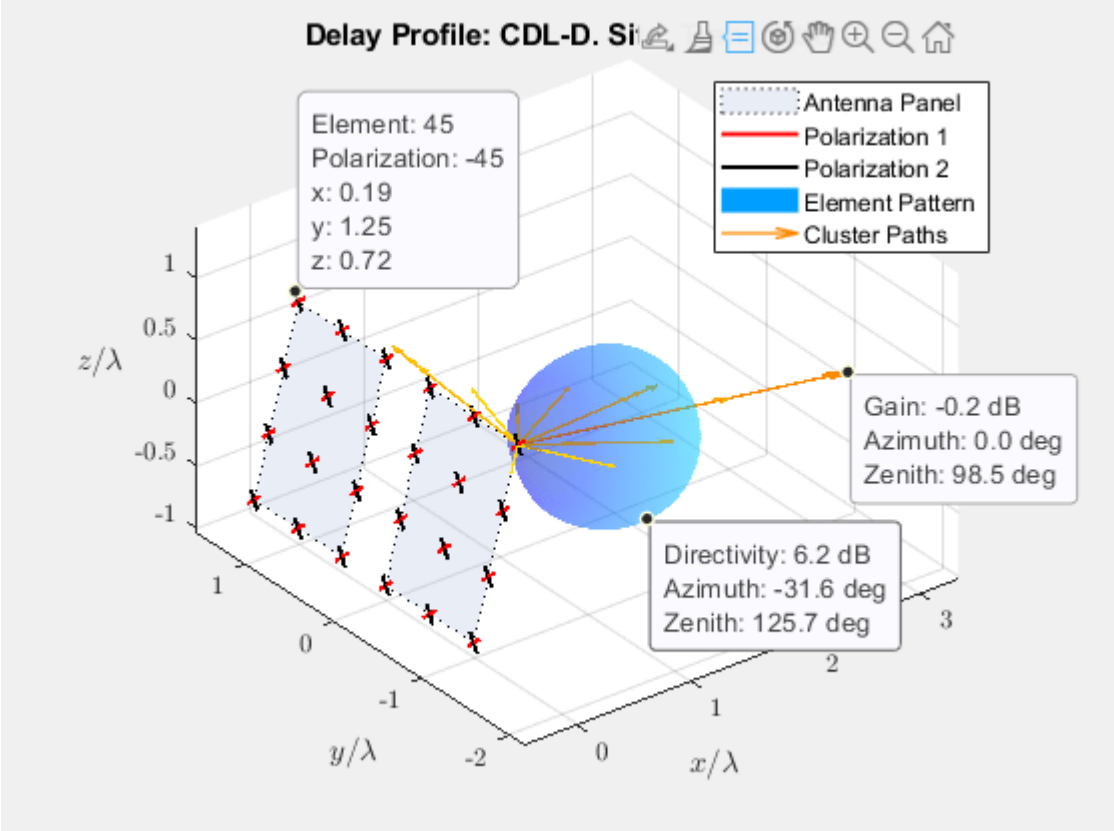
```
datacursormode on;
```



With data cursor mode enabled, explore channel characteristics by adding data tips. To create a data tip, click a data point. To create multiple data tips, press the **Shift** key while clicking the data points.

For example, this figure shows data tips added to the antenna element, element pattern, and cluster paths at the transmitter end.

- Antenna element data tips include information about the position, polarization angle, and element number of each antenna element. The element numbers indicate the order in which the channel model maps input signals column-wise to antenna elements. For more details, see the `TransmitAntennaArray.Size` property of the `lte3DChannel` System object.
- Element pattern data tips include the directivity corresponding to any azimuth and zenith angles.
- Cluster path data tips include the average path gain and azimuth and zenith angles of the cluster path.

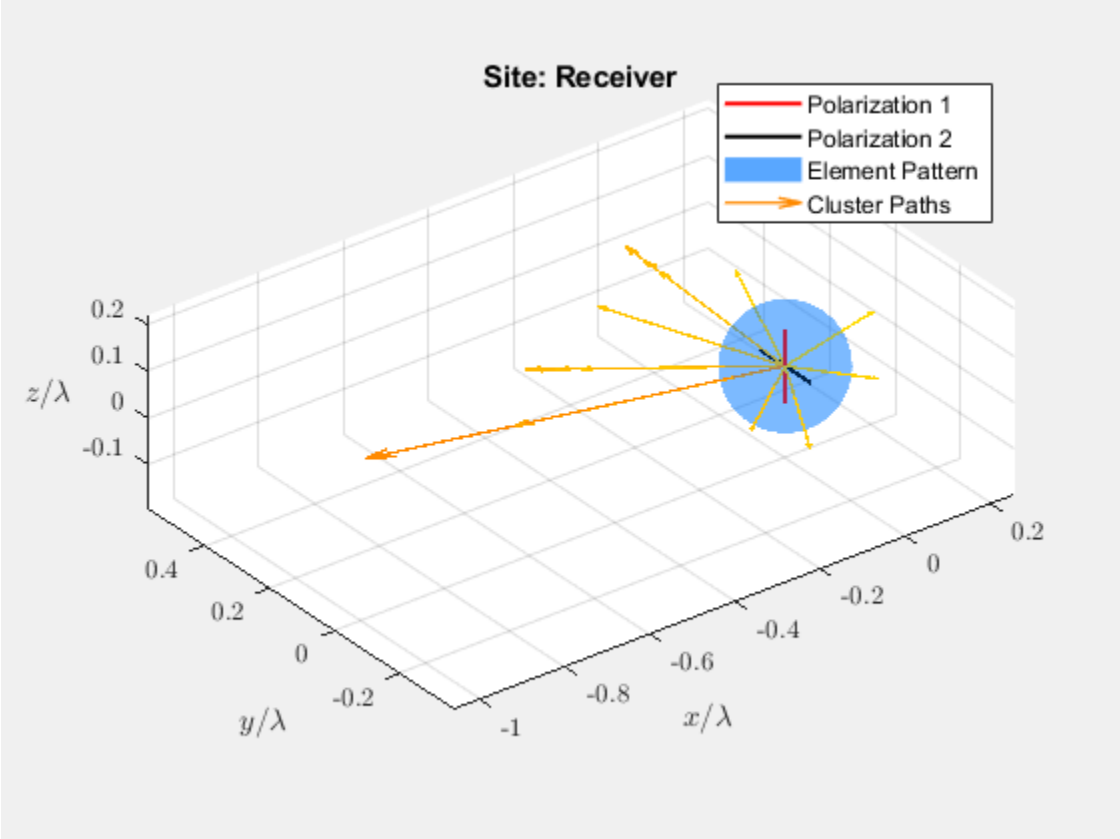



Visualize and explore channel characteristics at the receiver end. To customize the receive antenna array, use the `ReceiveAntennaArray` property of the `lte3DChannel` System object. Then, display the channel characteristics at the receiver end by calling the `displayChannel` function with the `'LinkEnd','Rx'` name-value pair argument.

```
figRx = displayChannel(lte3D,'LinkEnd','Rx');
```

Explore channel information about the antenna element, element pattern, and cluster paths at the receiver end by enabling data cursor mode for the current figure.

```
datacursormode on;
```




## Input Arguments

### `lte3D` — 3-D channel model

`lte3DChannel` System object™

3-D channel model, specified as an `lte3DChannel` System object. This object implements the link-level MIMO fading channel specified in section 7.3 of TR 36.867 [1] with the optional cluster delay line (CDL) profile from section 7.7.1 of TR 38.901 [2].

#### Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

*Before R2021a, use commas to separate each name and value, and enclose* `Name` *in quotes.*

Example: `'LinkEnd','Tx'` specifies visualization for the transmitter end only.

### `LinkEnd` — Link-level channel end

`'Both'` (default) | `'Tx'` | `'Rx'`

Link-level channel end, specified as the comma-separated pair consisting of `'LinkEnd'` and one of these values.

- `'Both'` — Display channel characteristics at both ends: the transmitter and receiver ends.
- `'Tx'` — Display channel characteristics only at the transmitter end.
- `'Rx'` — Display channel characteristics only at the receiver end.

Data Types: `char` | `string`

**`Polarization` — Polarization angle of antenna elements**
`'on'` (default) | `'off'`

Polarization angle of antenna elements, specified as the comma-separated pair consisting of `'Polarization'` and `'on'` or `'off'`. To display the polarization angle of the antenna elements, specify this input as `'on'`.

Data Types: `char` | `string`

**`ElementPattern` — Directivity radiation pattern of antenna elements**
`'on'` (default) | `'off'`

Directivity radiation pattern of antenna, specified as the comma-separated pair consisting of `'ElementPattern'` and `'on'` or `'off'`. To display the directivity radiation pattern of the antenna elements, specify this input as `'on'`.

**Note** In the specified channel model, `lte3D`, the antenna element pattern is the same for all antenna elements. To orient the array with respect to the cluster paths, the function displays the element pattern centered at the first element of the array.

Data Types: `char` | `string`

**`ClusterPaths` — Direction and average gain of cluster paths**
`'on'` (default) | `'off'`

Direction and average gain of cluster paths, specified as the comma-separated pair consisting of `'ClusterPaths'` and `'on'` or `'off'`. To display the direction and average gain of the cluster paths, specify this input as `'on'`.

**Note** In the specified channel model, `lte3D`, the cluster path directions are the same for all antenna elements. To orient the array with respect to the cluster paths, the function displays the path directions centered at the first element of the array.

Data Types: `char` | `string`

## Output Arguments

**`fig` — Visualization windows**
1-by-2 array of `figure` objects

Visualization windows, returned as a 1-by-2 array of `figure` objects.

# Version History

**Introduced in R2020b**

## References

[1] 3GPP TR 36.873. “Study on 3D channel model for LTE.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. URL: https://www.3gpp.org.

[2] 3GPP TR 38.901. “Study on channel model for frequencies from 0.5 to 100 GHz.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

**Functions**

`getPathFilters` | `info`

**Objects**

`lte3DChannel`

# lteACKDecode

HARQ-ACK channel decoding

## Syntax

```
out = lteACKDecode(chs,in)
```

## Description

`out = lteACKDecode(chs,in)` performs block decoding on soft input data `in`, assumed to be encoded using the procedure defined for HARQ-ACK in Section 5.2.2.6 of [1], for PUSCH channel transmission configuration `chs`. The decoded output, `out`, is a vector of length `OACK`, the number of uncoded HARQ-ACK bits transmitted.

**Note** If `NBundled` is 0, TDD ACK-NACK descrambling is disabled.

Multiple codewords can be parameterized by two different forms of the `chs` structure. Each codeword can be defined by separate elements of a 1-by-2 structure array, or the codeword parameters can be combined together in the fields of a single scalar, or 1-by-1, structure. Any scalar field values apply to both codewords and a scalar `chs.NLayers` is the total number. See “UL-SCH Parameterization” for further details.

The block decoding is performed separately on each soft input data codeword using a maximum likelihood (ML) approach, assuming that `in` has been demodulated and equalized to best restore the originally transmitted values.

The HARQ-ACK decoder performs different type of block decoding depending upon the number of uncoded HARQ-ACK bits to be recovered (`OACK`). For `OACK` less than 3 bits, the decoder assumes the bits are encoded using the procedure defined in TS 36.212 [1], Section 5.2.2.6.

For decoding between 3 and 11 HARQ-ACK bits, the decoder assumes the bits are block encoded using the procedure defined in TS 36.212 [1], Section 5.2.2.6.4. For greater than 11 bits, the decoder performs the inverse procedure described in TS 36.212 [1], Section 5.2.2.6.5.

## Examples

### Decode HARQ-ACK Channel

Show the block decoding of 3 coded HARQ-ACK information bits.

Create input and initialize channel structure. Encode bits and turn logical bits into soft data compatible with log-likelihood ratio check. Use `pskmod` with an initial phase offset of $\pi$ to align mapping with LTE codebook.

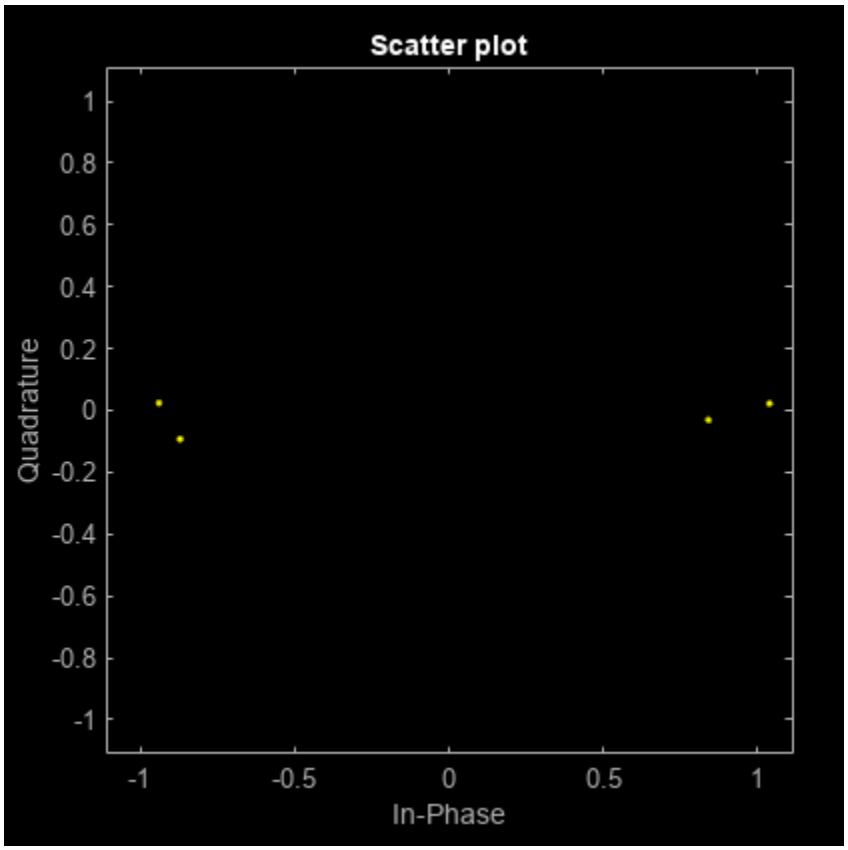
Perform HARQ-ACK bit encoding and modulation.

```
in = [1;0;1];
chs = struct('Modulation','QPSK','QdACK',2,'OACK',length(in));
```

```
encodedBits = lteACKEncode(chs,in);
encodedBits = pskmod(double(encodedBits),2,pi());
```

Pass transmitted encoded bits through an AWGN channel with a 20 dB signal-to-noise ratio. Show a `scatterplot` of the noisy received HARQ-ACK softbits.

```
rxBits = awgn(encodedBits,20);
scatterplot(rxBits)
```




Decode the received softbits. Compare the decoded bits with the input bits to show the bits have been recovered with no error.

```
decodedBits = lteACKDecode(chs,rxBits)
```

```
decodedBits = 3x1 logical array

   1
   0
   1
```

```
isequal(in,decodedBits)
```

```
ans = logical
   1
```

## Input Arguments

### `chs` — PUSCH-specific channel transmission configuration

scalar structure | structure array

PUSCH-specific channel transmission configuration, specified as a structure or a structure array, which contains the following parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`Modulation`** | Required | `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'` | Modulation type, specified as a character vector, cell array of character vectors, or string array. If blocks, each cell is associated with a transport block. |
| **`OACK`** | Optional | nonnegative scalar integer, 0 (default) | Number of uncoded HARQ-ACK bits.<br><br>The HARQ-ACK decoder performs different type of block decoding depending upon the number of uncoded HARQ-ACK bits to be recovered (`OACK`). For `OACK` less than 3 bits, the decoder assumes the bits are encoded using the procedure defined in TS 36.212 [1], Section 5.2.2.6. For decoding between 3 and 11 HARQ-ACK bits, the decoder assumes the bits are block encoded using the procedure defined in TS 36.212 [1], Section 5.2.2.6.4. For greater than 11 bits, the decoder performs the inverse procedure described in TS 36.212 [1], Section 5.2.2.6.5. |
| **`NLayers`** | Optional | 1 (default), 2, 3, 4 | Number of transmission layers. |
| **`NBundled`** | Optional | 0 (default), 1, ..., 9 | TDD HARQ-ACK bundling scrambling sequence index. When set to 0, the function disables the TDD HARQ-ACK bundling scrambling. Therefore, it is off by default. |

### `in` — Soft input data

numeric vector

Soft input data, specified as a numeric vector. The input data is assumed to be encoded using the procedure defined for HARQ-ACK in TS 36.212 [1], Section 5.2.2.6.

## Output Arguments

### `out` — Decoded HARQ-ACK channel

numeric column vector

Decoded HARQ-ACK channel output, returned as an `OACK`-by-1 column vector.

Data Types: `logical`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteACKEncode` | `lteRIDecode` | `lteCQIDecode` | `lteULSCHDeinterleave` | `lteULSCHDecode` | `lteUCIDecode`

# lteACKEncode

HARQ-ACK channel encoding

## Syntax

```
out = lteACKEncode(chs,in)
```

## Description

`out = lteACKEncode(chs,in)` returns the coded HARQ-ACK information bits after performing block coding defined for HARQ-ACK in TS 36.212 [1], Section 5.2.2.6 . The input argument, `in`, is a vector or cell array containing up to 20 HARQ-ACK information bits. The output argument, `out`, is the encoded bits in the same form.

Multiple codewords can be parameterized by two different forms of the `chs` structure. Each codeword can be defined by separate elements of a 1-by-2 structure array, or the codeword parameters can be combined together in the fields of a single scalar, or 1-by-1, structure. Any scalar field values apply to both codewords and a scalar `NLayers` is the total number. See “UL-SCH Parameterization” for further details.

Since the HARQ-ACK bits are carried on all defined codewords, a single input results in a cell array of encoded outputs if multiple codewords are parameterized. This allows for easy integration with the other toolbox functions.

The HARQ-ACK coder performs different types of block coding depending upon the number of HARQ-ACK bits in vector `in`. If `in` consists of one element, it uses TS 36.212 [1], Table 5.2.2.6-1. If `in` consists of two elements, it uses TS 36.212 [1], Table 5.2.2.6-2 [1] for encoding. The placeholder bits, *x* and *y* in the referenced tables, are represented by –1 and –2, respectively.

Similarly, for between 3 and 11 bits, the HARQ-ACK encoding is performed as described in TS 36.212 [1], Section 5.2.2.6.4. For bits greater than 11, the encoding is performed as described in TS 36.212 [1], Section 5.2.2.6.5.

## Examples

### Encode HARQ-ACK Channel with one codeword

Encode a HARQ-ACK information bit for one codeword with 16QAM modulation.

```
ackbit = 1;
chs.Modulation = '16QAM';
chs.QdACK = 1;
out1 = lteACKEncode(chs,ackbit)
```

```
out1 = 4x1 int8 column vector

    1
   -2
   -1
```

```
    -1
```

### Encode HARQ-ACK Channel with two codewords

Encode a HARQ-ACK information bit for two codewords with differing modulation schemes.

```
ackbit = 1;
chs.Modulation = {'16QAM' '64QAM'};
chs.NLayers = 2;
chs.QdACK = 1;
out2 = lteACKEncode(chs,ackbit)
```

```
out2=1×2 cell array
    {4x1 int8}    {6x1 int8}
```

## Input Arguments

### `chs` — PUSCH-specific channel transmission configuration

scalar structure | structure array

PUSCH-specific channel transmission configuration, specified as a structure or a structure array, which contains the following parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`QdACK`** | Required | nonnegative scalar integer | Number of coded HARQ-ACK symbols for ACK or NACK ($Q'\_ACK$) |
| **`Modulation`** | Required | `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'` | Modulation type, specified as a character vector, cell array of character vectors, or string array. If blocks, each cell is associated with a transport block. |
| **`NLayers`** | Optional | 1 (default), 2, 3, 4 | Number of transmission layers, total or per codeword |
| **`NBundled`** | Optional | 0 (default), 1, ..., 9 | TDD HARQ-ACK bundling scrambling sequence index. When set to 0, the function disables the TDD HARQ-ACK bundling scrambling. Therefore, it is off by default. |

### `in` — HARQ-ACK information bits

logical vector of length 1 to 20 | cell array of logical vectors

HARQ-ACK information bits, specified as a logical vector or a cell array of logical vectors. Each vector can have a length of up to 20 information bits.

Data Types: `logical` | `double` | `cell`

## Output Arguments

### `out` — Encoded HARQ-ACK information bits

integer column vector | cell array of integer column vectors

Encoded HARQ-ACK information bits, returned as either an integer column vector or a cell array of integer column vectors. The encoded bits are in the same form as the input bits. Therefore, if the PUSCH-specific parameter structure, `chs`, defines multiple codewords, `out` is a cell array.

Data Types: `int8` | `cell`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteACKDecode` | `lteULSCHInterleave` | `lteRIEncode` | `lteCQIEncode` | `lteUCIEncode` | `lteULSCH`

# lteBCH

Broadcast channel

## Syntax

```
codeblk = lteBCH(enb,trblk)
codeblk = lteBCH(trblk,outlen,cellrefp)
```

## Description

`codeblk = lteBCH(enb,trblk)` returns a vector of BCH transport channel coded bits. The encoding process includes CRC calculation and attachment, convolutional encoding, and rate matching as defined in TS 36.212 [1], Section 5.3.1.

The rate matching internal to the coding results in many repetitions of the coded block. This repetition is deliberate so that part of a received block can be successfully decoded in isolation. Typically, the receiver can recover the BCH bits from the reception of just one frame (¼ of the transmitted block), rather than waiting 40 ms (four frames) for the full block to be received.

`codeblk = lteBCH(trblk,outlen,cellrefp)` returns the vector rate-matched to the output length `outlen`. The argument `cellrefp` controls the CRC port mask.

## Examples

### Encode BCH Information Bits

Generate the BCH coded vector of length 1920, corresponding to normal cyclic prefix.

```
enb = struct('CellRefP',1,'CyclicPrefix','Normal');
bchCoded = lteBCH(enb,ones(24,1));
bchCodedSize = size(bchCoded)
```

```
bchCodedSize = 1×2

        1920           1
```

## Input Arguments

### enb — eNodeB cell-wide settings

scalar structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`CellRefP`** | Required | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **`CyclicPrefix`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |

**`trblk` — Transport block**
numeric vector

Transport block, specified as a numeric vector of length 24 bits. This argument represents the transport block delivered to the BCH every 40 ms.

**`outlen` — Output length**
numeric scalar

Output length, specified as a numeric scalar.

Data Types: `double`

**`cellrefp` — Number of cell-specific reference signal (CRS) antenna ports**
`0` | `1` | `2` | `4`

Number of cell-specific reference signal (CRS) antenna ports, specified as `0`, `1`, `2`, or `4`. To turn the CRC port mask off, set `cellrefp` to `0`.

## Output Arguments

**`codeblk` — BCH transport channel coded bits**
numeric column vector

BCH transport channel coded bits, returned as an integer column vector with 1920 bits for normal cyclic prefix or 1728 bits for extended cyclic prefix.

Data Types: `int8`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: http://www.3gpp.org.

## See Also

`lteBCHDecode` | `lteMIB` | `ltePBCH`

# lteBCHDecode

Broadcast channel decoding

## Syntax

```
[trblk,cellrefp] = lteBCHDecode(enb,softbits)
```

## Description

`[trblk,cellrefp] = lteBCHDecode(enb,softbits)` returns a vector, `trblk`, of the decoded information bits (24 bits). `cellrefp` is the number of cell-specific reference signal antenna ports detected in the CRC mask for given input, `softbits`, and the structure, `enb`. This function performs the inverse of the Broadcast Channel (BCH) processing described in TS 36.212 [1], Section 5.3.1.

## Examples

### Decode BCH-Encoded Block

Perform BCH coding of one transport block, and BCH decoding of part (one quarter) of the encoded block. In a practical system, this approach would be used to attempt BCH decoding on the one quarter part of the encoded block that is transmitted in the first subframe of each frame.

Create cell-wide configuration structure, initialized to RMC R.4. Perform BCH coding of one transport block.

```
enb = lteRMCDL('R.4');
bchCoded = lteBCH(enb,ones(24,1));
```

Perform BCH decoding of one quarter of the transport block.

```
out = bchCoded(1:length(bchCoded)/4);
[bchDecoded,cellRefP] = lteBCHDecode(enb,out);
bchDecoded(1:10)
```

```
ans = 10x1 int8 column vector

   1
   1
   1
   1
   1
   1
   1
   1
   1
   1
```

## Input Arguments

### `enb` — eNodeB cell-wide settings

scalar structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| `CyclicPrefix` | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |

### `softbits` — Soft bits to decode

numeric vector

Soft bits to decode, specified as a numeric vector. This vector can have any length.

The transport block size, 24, is relatively small when compared to the number of coded bits sent in the BCH transmission, 1920 or 1728. For this reason, the rate matching internal to the BCH coding results in many repetitions of the coded block. This decoder allows the input argument `softbits` to be of any length because successful decoding of coded BCH blocks is often possible using a fraction of the full coded block length.

## Output Arguments

### `trblk` — Decoded information bits

integer-valued column vector

Decoded information bits, returned as an integer-valued column vector of length 24.

Data Types: `int8`

### `cellrefp` — Number of CRS antenna ports

`0` | `1` | `2` | `4`

Number of cell-specific reference signal (CRS) antenna ports detected, returned as `0`, `1`, `2`, or `4`. A value of `0` indicates that the function detects a cyclic redundancy check (CRC) error during the decoding process.

Data Types: `uint32`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteBCH` | `ltePBCHDecode`

# lteCFI

Control format indicator block encoding

## Syntax

```
cw = lteCFI(enb)
```

## Description

`cw = lteCFI(enb)` returns a 32-element vector, `cw`, that represents the rate 1/16 block encoding of the control format indicator (CFI) value defined in the `CFI` field of the `enb` structure.

The value for CFI can be 1, 2, or 3. This value indicates the time span, in OFDM symbols, of the DCI PDCCH transmission (the control region) in that downlink subframe. For bandwidths in which `NDLRB` is greater than 10 RB, the span of the DCI in OFDM symbols is the same as the actual CFI value. If `NDLRB` is less than or equal to 10 RB, the span is *CFI*+1 symbols.

## Examples

### Encode CFI Value

Generate the 32-element vector that represents block encoding of a CFI value of 2.

```
cw = lteCFI(struct('CFI',2));
cw(1:10)
```

```
ans = 10x1 int8 column vector

   1
   0
   1
   1
   0
   1
   1
   0
   1
   1
```

## Input Arguments

### enb — eNodeB cell-wide settings

scalar structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`CFI`** | Required | 1, 2, or 3<br>Scalar or if the CFI varies per subframe, a vector of length 10 (corresponding to a frame). | Control format indicator (`CFI`) value. In TDD mode, `CFI` varies per subframe for the RMCs (`'R.0'`, `'R.5'`, `'R.6'`, `'R.6-27RB'`, `'R.12-9RB'`)<br><br>The value for CFI can be 1, 2, or 3. This value indicates the time span, in OFDM symbols, of the DCI PDCCH transmission (the control region) in that downlink subframe. For bandwidths in which `NDLRB` is greater than 10 RB, the span of the DCI in OFDM symbols is the same as the actual CFI value. If `NDLRB` is less than or equal to 10 RB, the span is *CFI*+1 symbols. |

## Output Arguments

### `cw` — CFI codeword

integer column vector

CFI codeword, returned as an integer column vector of length 32. This vector represents the 1/16 block encoding of the CFI value defined in structure `enb`.

Data Types: `int8`

# Version History

**Introduced in R2014a**

## See Also

`lteCFIDecode` | `ltePCFICH`

# lteCFIDecode

Control format indicator block decoding

## Syntax

```
cfi = lteCFIDecode(ibits)
```

## Description

`cfi = lteCFIDecode(ibits)` performs the block decoding on soft input data `ibits`, assumed to be encoded using procedure defined in TS 36.212 [1], Section 5.3.4.1. The output, `cfi`, is a scalar representing the control format indicator (CFI) value resulted after performing block decoding on input data. Strictly speaking, `ibits` should be a vector 32 bits long, as per encoded `cfi`. See the `lteCFI` function reference for details. However, this function can decode any size segment of encoded data.

The value for CFI can be 1, 2, or 3. This value indicates the time span, in OFDM symbols, of the DCI PDCCH transmission (the control region) in that downlink subframe. For bandwidths in which `NDLRB` is greater than 10 RB, the span of the DCI in OFDM symbols is the same as the actual CFI value. If `NDLRB` is less than or equal to 10 RB, the span is *CFI*+1 symbols.

## Examples

### Decode CFI Block

Decode a noisy 32-element vector that represents the block encoding of the control format indicator (CFI) value.

```
cw = double(lteCFI(struct('CFI',2)));
noisycw = cw + 0.4*randn(length(cw),1);
cfi = lteCFIDecode(noisycw)
```

```
cfi = int32
    2
```

## Input Arguments

### `ibits` — Soft input data

numeric vector

Soft input data, specified as a numeric vector of length 32. This input data is assumed to be encoded using the procedure defined in TS 36.212 [1], Section 5.3.4.1.

## Output Arguments

### `cfi` — Control format indicator value

1 | 2 | 3

Control format indicator value, returned as a positive scalar integer. This integer represents the CFI value resulting from performing block decoding on a vector of soft input data, `ibits`.

The value for CFI can be 1, 2, or 3. This value indicates the time span, in OFDM symbols, of the DCI PDCCH transmission (the control region) in that downlink subframe. For bandwidths in which `NDLRB` is greater than 10 RB, the span of the DCI in OFDM symbols is the same as the actual CFI value. If `NDLRB` is less than or equal to 10 RB, the span is *CFI*+1 symbols.

Data Types: `int32`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteCFI` | `ltePCFICHDecode`

# lteCQIDecode

Channel quality information channel decoding

## Syntax

```
out = lteCQIDecode(chs,in)
```

## Description

`out = lteCQIDecode(chs,in)` performs the decoding on soft input data, `in`, assumed to be encoded using the procedure defined for channel quality information (CQI) in TS 36.212, Sections 5.2.2.6 and 5.2.2.6.4 [1] for given channel transmission configuration, `chs`. The decoded output, `out`, is a vector of length `OCQI`, the number of uncoded CQI bits transmitted.

Multiple codewords can be parameterized by two different forms of the `chs` structure. Each codeword can be defined by separate elements of a 1-by-2 structure array, or the codeword parameters can be combined together in the fields of a single scalar, or 1-by-1, structure. Any scalar field values apply to both codewords and a scalar `NLayers` is the total number. See “UL-SCH Parameterization” for further details.

The block decoding is performed separately on each soft input data using a maximum likelihood (ML) approach, which assumes that `in` has been demodulated and equalized to best restore the original transmitted values. The length of CQI bits defines the decoding process.

If the number of CQI bits, `OCQI`, is less than or equal to 11, a block decoding is performed to invert the coding procedure defined in TS 36.212, Section 5.2.2.6.4 [1]. If `OCQI` is greater than 11, the CQI bits are recovered by performing rate matching to `OCQI`, tail-biting Viterbi decoding, and 8-bit CRC decoding.

## Examples

### Decode CQI bits

Decode encoded CQI bits.

Create input stream and initialize channel settings structures for encoding and decoding. Encode CQI bits and turn logical bits into soft data. Decode the CQI bits.

```
cqi = [0; 1; 0; 1; 0; 1];

chsEnc.Modulation = 'QPSK';
chsEnc.QdCQI = 16;
chsEnc.NLayers = 1;

chsDec.NLayers = 1;
chsDec.OCQI = 6;

enc = lteCQIEncode(chsEnc,cqi);
enc = double(enc)-0.5;

rxCqi = lteCQIDecode(chsDec,enc)
```

```
rxCqi = 6x1 logical array

   0
   1
   0
   1
   0
   1
```

## Input Arguments

### `chs` — Channel-specific transmission configuration

scalar structure | structure array

Channel-specific transmission configuration, specified as a structure that can contain the following parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`OCQI`** | Optional | nonnegative scalar integer, 0 (default) | Number of uncoded channel quality information (CQI) bits |
| **`NLayers`** | Optional | 1 (default), 2, 3, 4 | Number of transmission layers. |

### `in` — Encoded soft input data

numeric vector

Encoded soft input data, specified as a numeric vector.

## Output Arguments

### `out` — Decoded output

logical column vector

Decoded output, returned as a logical column vector of length `OCQI`.

Data Types: `logical`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteCQIEncode` | `lteACKDecode` | `lteRIDecode` | `lteUCIDecode` | `lteULSCHDecode`

# lteCQIEncode

Channel quality information channel encoding

## Syntax

```
out = lteCQIEncode(chs,in)
```

## Description

`out = lteCQIEncode(chs,in)` returns the encoded channel quality information (CQI) bits after performing channel coding defined for CQI in TS 36.212 [1], Sections 5.2.2.6 and 5.2.2.6.4. `in` should be a vector or cell array containing the CQI bits and `out` is the encoded bits in the same form. `out` is also cell array if the PUSCH-specific parameter structure, `chs`, defines multiple codewords.

Multiple codewords can be parameterized by two different forms of the `chs` structure. Each codeword can be defined by separate elements of a 1-by-2 structure array, or the codeword parameters can be combined together in the fields of a single scalar, or 1-by-1, structure. Any scalar field values apply to both codewords and a scalar `NLayers` is the total number. See “UL-SCH Parameterization” for further details.

While the CQI information bits are carried on one codeword only, a single input still results in a cell array of encoded outputs if multiple codewords are parameterized. In this case, the `QdCQI` field should contain a 0 in the position of the unused codeword. This allows for easy integration with the other toolbox functions.

The CQI coder uses two different coding schemes depending upon the number of CQI bits to be coded. If the number of CQI bits are less than or equal to 11, the channel coding of the CQI bits is performed according to TS 36.212 [1], Section 5.2.2.6.4. For CQI bits greater than 11, the coding process includes 8-bit CRC attachment, tail-biting convolutional coding and rate matching to the output length deduced from parameters `QdCQI` and `Modulation`.

## Examples

### Encode CQI Bits for One Codeword

Generate the coded CQI bits for a single codeword.

Create input stream and initialize channel settings structure. Encode CQI bits.

```
in = [0; 1; 0; 1; 0; 1];
chs1.Modulation = '16QAM';
chs1.QdCQI = 4;
chs1.NLayers = 2;

codedCqi1 = lteCQIEncode(chs1,in)
```

```
codedCqi1 = 32x1 int8 column vector

   1
   1
```

```
     1
     1
     0
     1
     0
     1
     0
     1
```

### Encode CQI Bits for Second of Two Codewords

Generate the coded CQI bits for two codewords with CQI on the second codeword.

Create input stream and initialize channel settings structure. Encode CQI bits. In this case the CQI is on the second codeword. The output is a cell array where the first cell is empty.

```
in = [0; 1; 0; 1; 0; 1];
chs2.Modulation = {'16QAM' '16QAM'};
chs2.QdCQI = [0 4];
chs2.NLayers = 2;

codedCqi2 = lteCQIEncode(chs2,in)
```

```
codedCqi2=1×2 cell array
    {0x1 int8}    {16x1 int8}
```

## Input Arguments

### `chs` — Channel-specific transmission configuration

scalar structure | structure array

Channel-specific transmission configuration, specified as a structure that can contain the following parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`QdCQI`** | Required | nonnegative scalar integer | Number of coded channel quality information (CQI) symbols (*Q'_CQI*) |
| **`Modulation`** | Required | `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'` | Modulation type, specified as a character vector, cell array of character vectors, or string array. If blocks, each cell is associated with a transport block. |
| **`NLayers`** | Optional | 1 (default), 2, 3, 4 | Number of transmission layers. |

### `in` — CQI input bits

numeric vector | cell array of numeric vectors

CQI input bits, specified as a numeric vector or a cell array of numeric vectors.

## Output Arguments

**`out` — Encoded CQI output bits**
integer vector | cell array of integer vectors

Encoded CQI output bits, returned as an integer vector or a cell array of integer vectors. This argument contains the coded CQI bits after performing channel coding.

Data Types: `int8` | `cell`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: http://www.3gpp.org.

## See Also

`lteCQIDecode` | `lteACKEncode` | `lteRIEncode` | `lteUCIEncode` | `lteULSCH`

# lteCQISelect

PDSCH channel quality indication calculation

## Syntax

```
[cqi,sinrs] = lteCQISelect(enb,chs,hest,noiseest)
```

## Description

`[cqi,sinrs] = lteCQISelect(enb,chs,hest,noiseest)` calculates PDSCH CQI (Channel Quality Indication) for cell-wide configuration `enb`, channel configuration `chs`, channel estimate `hest`, and receiver noise variance `noiseest`. For more information, see “CQI Selection” on page 2-35.

## Examples

### Calculate CQI

An empty resource grid for RMC R.13 is populated with cell-specific reference signals symbols. The signal is filtered through the channel, demodulated and the corresponding channel is estimated along with an estimate of noise power spectral density on the reference signal subcarriers. The estimates are used for CQI calculation.

Populate an empty resource grid for RMC R.13 with cell-specific reference signal symbols and modulate the waveform. Add noise to `txWaveform`. Configure an EPA fading channel and filter the signal through this channel.

```
enb = lteRMCDL('R.13');
reGrid = lteResourceGrid(enb);
reGrid(lteCellRSIndices(enb)) = lteCellRS(enb);
[txWaveform,info] = lteOFDMModulate(enb,reGrid);

noise = 0.5*complex(randn(size(txWaveform)),randn(size(txWaveform)));
txWaveform_nz = txWaveform + noise;

chcfg.SamplingRate = info.SamplingRate;
chcfg.DelayProfile = 'EPA';
chcfg.NRxAnts = 4;
chcfg.DopplerFreq = 5;
chcfg.MIMOCorrelation = 'Low';
chcfg.InitTime = 0;
chcfg.Seed = 1;
rxWaveform = lteFadingChannel(chcfg,txWaveform_nz);
```

Demodulate the received signal. Perform downlink channel estimate and noise power spectral density estimation on the demodulated signal. Use estimates of channel and noise power spectral density for CQI calculation.

```
rxSubframe = lteOFDMDemodulate(enb,rxWaveform);
```

```
cec.FreqWindow = 1;
cec.TimeWindow = 15;
cec.InterpType = 'cubic';
cec.PilotAverage = 'UserDefined';
cec.InterpWinSize = 1;
cec.InterpWindow = 'Centered';
[hest, noiseEst] = lteDLChannelEstimate(enb,cec,rxSubframe);

cqi = lteCQISelect(enb,enb.PDSCH,hest,noiseEst)

cqi = 5
```

## Input Arguments

### enb — eNodeB cell-wide settings

structure

eNodeB cell-wide settings, specified as a structure. The structure contains the following parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NDLRB** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **NCellID** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **CellRefP** | Required | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **CyclicPrefix** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **DuplexMode** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| The following parameters apply when `DuplexMode` is set to `'TDD'`. | | | |
| **TDDConfig** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **SSC** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |
| The following parameters apply when `DuplexMode` is set to `'TDD'` or `chs.TxScheme` is set to `'Port7-14'` | | | |
| **NSubframe** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| The following parameters apply when `chs.TxScheme` is set to `'Port7-14'` | | | |
| **CSIRefP** | Required | 1, 2, 4, 8 | Array of number of CSI-RS antenna ports |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **CSIRSConfig** | Required | Scalar integer | Array CSI-RS configuration indices. See TS 36.211, Table 6.10.5.2-1. |
| **CSIRSperiod** | Optional | `'On'` (default), `'Off'`, `Icsi-rs` (0,...,154), `[Tcsi-rs Dcsi-rs]`. You can also specify values in a cell array of configurations for each resource. | CSI-RS subframe configurations for one or more CSI-RS resources. Multiple CSI-RS resources can be configured from a single common subframe configuration or from a cell array of configurations for each resource. |
| **NFrame** | Optional | 0 (default), nonnegative scalar integer | Frame number |

**`chs` — Channel-specific transmission configuration**
structure | structure array

Channel-specific transmission configuration, specified as a structure or structure array. The structure contains the following parameter fields:

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NLayers** | Required | Integer from 1 to 8 | Number of transmission layers. |
| **CSIMode** | Required | `'PUCCH 1-0'`, `'PUCCH 1-1'`, `'PUSCH 1-2'`, `'PUSCH 3-0'`, `'PUSCH 3-1'` | CSI reporting mode |

<table>
<tr><th>Parameter Field</th><th>Required or Optional</th><th>Values</th><th>Description</th></tr>
<tr><td>TxScheme</td><td>Required</td><td>'Port0', 'TxDiversity', 'CDD', 'SpatialMux', 'MultiUser', 'Port7-8', 'P 'Port7-14'.</td><td>PDSCH transmission scheme, specified as one of the following options.
<table>
<tr><th>Transmission scheme</th><th>Description</th></tr>
<tr><td>'Port0'</td><td>Single antenna port, port 0</td></tr>
<tr><td>'TxDiversity'</td><td>Transmit diversity</td></tr>
<tr><td>'CDD'</td><td>Large delay cyclic delay diversity scheme</td></tr>
<tr><td>'SpatialMux'</td><td>Closed loop spatial multiplexing</td></tr>
<tr><td>'MultiUser'</td><td>Multi-user MIMO</td></tr>
<tr><td>'Port5'</td><td>Single-antenna port, port 5</td></tr>
<tr><td>'Port7-8'</td><td>Single-antenna port, port 7, when NLayers = 1. Dual layer transmission, ports 7 and 8, when NLayers = 2.</td></tr>
<tr><td>'Port8'</td><td>Single-antenna port, port 8</td></tr>
<tr><td>'Port7-14'</td><td>Up to eight layer transmission, ports 7–14</td></tr>
</table></td></tr>
<tr><td>SINRs90pc</td><td>Optional</td><td>15 element vector, or function handle</td><td>A vector of 15 SINR values or a function handle to a function of the form f(enb, chs) which returns a vector of 15 SINR values, one for each CQI index 1, ..., 15. These correspond to the lowest SINR for which the throughput of the PDSCH in the CQI/CSI reference resource, for the given configuration and CQI index, is at least 90%. Default is to internally select SINRs based on configuration given in enb and chs, assuming perfect channel estimation and either MMSE equalization or transmit diversity decoding (as appropriate for the transmission scheme) at the receiver.</td></tr>
<tr><td colspan="4">The following parameter applies for 'SpatialMux', 'MultiUser', 'Port5', 'Port7-8', 'Port8', 'Port7-14' transmission schemes.</td></tr>
</table>

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **PMISet** | Required | Integer vector with element values from 0 to 15. | A vector of Precoder Matrix Indications. The vector may contain either a single value (corresponding to single PMI mode) or multiple values (corresponding to multiple or subband PMI mode). For the `'Port7-14'` transmission scheme with eight CSI-RS ports or for CSI reporting with the alternative codebook for four antennas, an additional first value indicates the wideband codebook index, `i1`, and subsequent values indicate the subband codebook indices, `i2`, or the wideband codebook index, `i2`. Valid value range depends on `CellRefP`, `CSIRefP`, `NLayers`, `TxScheme`, and `AltCodebook4Tx`. For more information about setting PMI parameters, see `ltePMIInfo`. |
| The following parameter applies for `'Port7-14'` transmission scheme with `CSIRefP` equal to 4, or for `'Port7-8'` or `'Port8'` transmission scheme with `CellRefP` equal to 4. | | | |
| **AltCodebook 4Tx** | Required | `'Off'` (default), `'On'` | If set to `'On'`, enables the alternative codebook for CSI reporting with four antennas defined in TS 36.213, Tables 7.2.4-0A to 7.2.4-0D. The default is `'Off'`. (*alternativeCodeBookEnabledFor4TX-r12*) |
| Additionally, one of the following fields must be included. see note 1 | | | |
| **NCodewords** | Required | 1, 2 | Number of codewords |
| **Modulation** | Required | `'QPSK'`, `'16QAM'`, `'64QAM'`, `'256QAM'`, `'1024QAM'` | Modulation type, specified as a character vector, cell array of character vectors, or string array. If blocks, each cell is associated with a transport block. |
| note 1 – Specify the number of codewords directly in the `NCodewords` field. Alternatively, if the `Modulation` field is provided, the number of codewords is established from the number of modulation formats. This value lets you establish the correct number of codewords using the channel transmission configuration structure, `chs`, as provided to `ltePDSCH` function on the transmit side. If present, the `NCodewords` field takes precedence. | | | |

### `hest` — Channel estimate

multidimensional array

Channel estimate, specified as a *K*-by-*L*-by-*NRxAnts*-by-*P* array, where:

- *K* is the number of subcarriers.
- *L* is the number of OFDM symbols.
- *NRxAnts* is the number of receive antennas.
- *P* is the number of transmit antennas.

Data Types: `double`

**`noiseest` — Receiver noise variance**
numeric scalar

Receiver noise variance, specified as a numeric scalar. `noiseest` is an estimate of the received noise power spectral density.

Data Types: `double`

## Output Arguments

**`cqi` — Channel quality information**
column vector

Channel quality information, returned as a column vector containing a channel quality information report. Report contents depend on the CSI reporting mode.

| Report Mode | Reporting Contents |
|---|---|
| Single codeword: | |
| **`'PUCCH 1-0'`** | A single wideband CQI index |
| **`'PUSCH 3-0'`** | A single wideband CQI index, followed by a subband differential CQI offset level for each subband. |
| Two codewords: | |
| **`'PUCCH 1-1'`** | A single wideband CQI index for codeword 0, followed by a spatial differential CQI offset level for codeword 1. |
| **`'PUSCH 1-2'`** | A single wideband CQI index for codeword 0, followed by a single wideband CQI index for codeword 1. |
| **`'PUSCH 3-1'`** | A single wideband CQI index for codeword 0, followed by a subband differential CQI offset level for each subband for codeword 0, followed by a single wideband CQI index for codeword 1, followed by a subband differential CQI offset level for each subband for codeword 1. |

**Note** CSI reporting modes, are separated into the modes that support one or two codewords, as described by the standard. The CQI select function derives these code words from `chs.NCodewords` or `chs.Modulation`.

**`sinrs` — signal-to-interference plus noise ratios**
matrix

Signal-to-interference plus noise ratios, in dB, returned as a matrix. Each column of the matrix represents a single codeword. If subband CQI reporting is configured, the SINR for the wideband CQI is in the first row, followed by the `sinrs` for the subband CQIs in subsequent rows. `sinrs` is an optional output.

## More About

### CQI Selection

The function performs the CQI selection by first obtaining SINR (Signal to Interference and Noise Ratio) estimates for a given configuration from `ltePMISelect`. Then the function performs a lookup between those SINR estimates and the CQI index. The lookup tables are precomputed and stored in this function. CQI selection is conditioned on the rank indicated by `chs.NLayers`, except for the `'TxDiversity'` transmission scheme which has a rank of 1. On PUCCH, CQI selection corresponds to Report Type 2 (for reporting Mode1-1) or Report Type 4 (for reporting Mode 1-0). On PUSCH, the reporting is Mode 1-2, Mode 3-0, or Mode 3-1.

A CQI Index is a scalar (0,...,15), indicating the selected value of the CQI index. The CQI index is defined as per TS 36.213. The highest CQI index is selected when a single PDSCH transport block with a modulation scheme and transport block size of CQI index, and occupying a group of downlink physical resource blocks termed the CSI reference resource, can be received with a transport block error probability not exceeding 0.1. If a CQI index of 1 does not satisfy this condition, then the returned CQI index is 0. The CQI reference resource is defined in TS 36.213, Section 7.2.3. The relationship between CQI indices, modulation scheme, and code rate (from which transport block size is derived) is described in TS 36.213, Tables 7.2.3-1 and 7.2.3-2.

A subband differential CQI offset level is the difference between a subband CQI index and the corresponding wideband CQI index.

A spatial differential CQI offset level is the difference between the wideband CQI index for codeword 0 and the wideband CQI index for codeword 1.

Within the 3GPP standard, CQI offsets are reported as *CQI values*. These values are nonnegative integers corresponding to single CQI offset levels or ranges of CQI offset levels (see TS 36.213, Tables 7.2-2 and 7.2.1-2). The CQI offset levels reported here are either the single CQI offset level corresponding to the CQI value reported or the boundary value of the CQI offset level range corresponding to the CQI value reported. For example, a calculated spatial differential CQI offset level of –6 would be reported per the standard as a spatial differential CQI value of 4. This function will return a spatial differential offset level of –4 because the calculated differential CQI offset level exceeds this boundary value, meaning –6 < –4 (see TS 36.213, Table 7.2-2).

For transmission schemes using UE-specific beamforming (`'Port 5'`, `'Port 7-8'`, `'Port 8'`, `'Port7-14'`), the performance depends on the beamforming used. For UE-specific beamforming, the appropriate value of `chs.SINRs90pc` field is provided. If this field is not provided, for single antenna ports, the function uses default `SINRs90pc` values.

# Version History

**Introduced in R2014b**

## References

[1] 3GPP TS 36.213. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteRISelect` | `ltePMISelect`

# lteCRCDecode

Cyclic redundancy check decoding and removal

## Syntax

```
[blk,err] = lteCRCDecode(blkcrc,poly)
[blk,err] = lteCRCDecode(blkcrc,poly,mask)
```

## Description

`[blk,err] = lteCRCDecode(blkcrc,poly)` checks the input data vector for a CRC error assuming the vector comprises a block of data with the associated CRC bits attached. The data part of the input is returned in vector `blk`. The logical difference (XOR) between the attached CRC and the CRC recalculated across the data part of the input is returned in `uint32` scalar `err`. If `err` is not equal to 0, either an error has occurred or the input CRC has been masked. A logical mask can also be applied directly to `err`. See TS 36.212 [1], Section 5.1.1 for the associated polynomials.

`[blk,err] = lteCRCDecode(blkcrc,poly,mask)` checks the input data vector for a CRC error XOR-ing with the scalar `mask` parameter before it is returned in `err`. The `mask` value is applied to the CRC bits with the most significant bit (MSB) first and the least significant bit (LSB) last.

## Examples

### Check Data Vector for CRC Error

Check the effect of CRC decoding a block of data with and without a mask.

CRC encode attaching a masked `'24A'`-type CRC to an all-ones vector of length 100.

```
rnti = 8;
blkcrc = lteCRCEncode(ones(100,1),'24A',rnti);
```

CRC decode with the data block without using a mask.

```
[blk1,err1] = lteCRCDecode(blkcrc,'24A');
err1
```

```
err1 = uint32
    8
```

The logical difference between the original CRC and recalculated CRC equals the CRC mask. Since the CRC was been masked, decoding without specifying the mask, returned `err1` = 8, which is the value of `rnti`.

CRC decode using the RNTI as a mask.

```
[blk2,err2] = lteCRCDecode(blkcrc,'24A',rnti);
err2
```

```
err2 = uint32
    0
```

The returned output, `err2`, is 0 because the original mask, `rnti`, is XORed with itself.

## Input Arguments

### `blkcrc` — CRC input data bit vector

numeric column vector

CRC input data bit vector, specified as a numeric column vector. The function checks the input bit vector for a CRC error assuming that the data consists of a block of data with CRC bits attached.

### `poly` — CRC polynomial

`'8'` | `'16'` | `'24A'` | `'24B'`

CRC polynomial, specified as `'8'`, `'16'`, `'24A'`, or `'24B'`. See TS 36.212 [1], Section 5.1.1 for the associated polynomials.

### `mask` — XOR mask

scalar integer

XOR mask, specified as a scalar integer. The CRC difference is XOR-ed with `mask` before `err` is returned.

Data Types: `double`

## Output Arguments

### `blk` — Data bit vector

column vector

Data bit vector, returned as a column vector. `blk` is the data-only part of the input `blkcrc`.

Data Types: `int8`

### `err` — Logical difference

integer

Logical difference, returned as an integer. `err` is the logical difference between the CRC and CRC recalculated across the data part of the input.

Data Types: `uint32`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteCRCEncode` | `lteCodeBlockDesegment` | `lteConvolutionalDecode`

# lteCRCEncode

Cyclic redundancy check calculation and appending

## Syntax

```
blkcrc = lteCRCEncode(blk,poly)
blkcrc = lteCRCEncode(blk,poly,mask)
```

## Description

`blkcrc = lteCRCEncode(blk,poly)` calculates a cyclic redundancy check (CRC) for the input data vector and returns a copy of the vector with the CRC attached. To support the correct processing of filler bits, negative input bit values are interpreted as logical 0 for the purposes of the CRC calculation. A value of –1 is used to represent filler bits. `lteCRCEncode` calculates the CRC defined by `poly` for the input bit vector `blk` and returns a copy of the input with the CRC appended in vector `blkcrc`. Valid options for the CRC polynomial are `'8'`, `'16'`, `'24A'`, or `'24B'`. See TS 36.212 [1], Section 5.1.1 for the associated polynomials.

`blkcrc = lteCRCEncode(blk,poly,mask)` XOR masks the appended CRC bits with the integral value of `mask`. The `mask` value is applied to the CRC bits with the most significant bit (MSB) first and the least significant bit (LSB) last.

## Examples

### Calculate and Append CRC

Calculate and append the CRC associated with an all zero vector, which is also zero.

```
crc1 = lteCRCEncode(zeros(100,1),'24A');
crc1(1:10)
```

```
ans = 10x1 int8 column vector

   0
   0
   0
   0
   0
   0
   0
   0
   0
   0
```

The result is an all-zeros vector of length 124.

### Calculate and Append CRC with MSB First

Mask the CRC bits in an MSB-first order.

Set the XOR mask to 1 to make the appended CRC bits XOR masked from the most significant to least significant bit.

```
mask = 1;
crc2 = lteCRCEncode(zeros(100,1),'24A',mask);
crc2(end-10:end)
```

```
ans = 11x1 int8 column vector

   0
   0
   0
   0
   0
   0
   0
   0
   0
   0
      ⋮
```

The result is all zeros, except for a single one in last element position.

## Input Arguments

### `blk` — Data bit vector

numeric column vector

Data bit vector, specified as a numeric column vector.

### `poly` — CRC polynomial

`'8'` | `'16'` | `'24A'` | `'24B'`

CRC polynomial, specified as `'8'`, `'16'`, `'24A'`, or `'24B'`. See TS 36.212 [1], Section 5.1.1 for the associated polynomials.

### `mask` — XOR mask

integer

XOR mask, specified as an integer. The appended CRC bits are XOR masked from the most significant to least significant bit.

## Output Arguments

### `blkcrc` — Bit vector with CRC

column vector

Bit vector with CRC, returned as a column vector.

Data Types: `int8`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: http://www.3gpp.org.

## See Also

`lteCRCDecode` | `lteCodeBlockSegment` | `lteConvolutionalEncode`

# lteCSICodebook

Codebook for channel state information reporting

## Syntax

```
out = lteCSICodebook(nu,p,idx)
out = lteCSICodebook(nu,p,idx,table)
out = lteCSICodebook(nu,p,i1,i2)
```

## Description

`out = lteCSICodebook(nu,p,idx)` returns the precoding matrix associated with channel state information (CSI) reporting as defined in TS 36.213 [1], Section 7.2.4 given the number of layers, `nu`, the number of antennas, `p`, and the codebook index, `idx`. For more information, see "CSI Codebook Reporting" on page 2-45 and `ltePMIInfo`.

`out = lteCSICodebook(nu,p,idx,table)` where `table` specifies the codebook selection table. For more information, see "CSI Codebook Reporting" on page 2-45.

`out = lteCSICodebook(nu,p,i1,i2)` where `i1` and `i2` specify the first and second codebook indices, respectively. This signature was only intended for `p` = 8. This signature may be removed in a future release, instead use `out` = lteCSICodebook(`nu`,`p`,`idx`) with `idx` = [`i1` `i2`].

## Examples

### Create Codebook Entry for CSI Reporting

This example creates a codebook entry for CSI reporting with 2 layers, 4 antennas, and a codebook index of 3.

```
lteCSICodebook(2,4,3)
```

```
ans = 4×2 complex

   0.3536 + 0.0000i   0.0000 + 0.3536i
   0.0000 - 0.3536i   0.3536 + 0.0000i
  -0.3536 + 0.0000i   0.0000 + 0.3536i
   0.0000 + 0.3536i   0.3536 + 0.0000i
```

### Create Alternate Codebook Entry for CSI Reporting

Create an alternative codebook entry for CSI reporting with three layers, and four antennas, using codebook indices provided.

```
lteCSICodebook(3,4,[0 7],'AltCodeBook4Tx')
```

```
ans = 4×3 complex
```

```
   0.2887 + 0.0000i   0.0000 + 0.2887i  -0.2041 + 0.2041i
   0.2041 - 0.2041i  -0.2041 - 0.2041i   0.0000 - 0.2887i
   0.0000 - 0.2887i   0.2887 + 0.0000i  -0.2041 - 0.2041i
  -0.2041 - 0.2041i  -0.2041 + 0.2041i   0.2887 + 0.0000i
```

The codebook entry [i1 i2] = [0 7] from TS 36.213, Table 7.2.4-0C is used.

## Input Arguments

### `nu` — Number of transmission layers

1,...,8 | positive scalar integer

Number of transmission layers, specified as an integer from 1 to 8.

### `p` — Number of transmission antennas

1 | 2 | 4 | 8 | positive scalar integer

Number of transmission antennas, specified as 1, 2, 4, or 8.

### `idx` — Codebook index

0,...,15 | scalar integer | vector with two integers

Codebook index, specified as an integer or vector of two integers from 0 to 15.

- If `p` = 8, `idx` should be a pair of indices [`i1` `i2`].
- If `p` = 4 and `table` = `'AltCodebook4Tx'`, `idx` should be a pair of indices [`i1` `i2`].
- If `p` = 4 and `table` = `'StdCodebook4Tx'`, `idx` should be a single index or a pair with `i1` set to zero.
- If `p` = 1 or `p` = 2, `idx` should be a single index or a pair with `i1` set to zero.

For more information, see “CSI Codebook Reporting” on page 2-45.

Example: [0 3] indicates the codebook indices [`i1` `i2`].

### `i1` — First codebook index

0 (default),...,15 | scalar integer

First codebook index, specified as an integer from 0 to 15. For more information, see “CSI Codebook Reporting” on page 2-45.

### `i2` — Second codebook index

0,...,15 | scalar integer

Second codebook index, specified as an integer from 0 to 15. For more information, see “CSI Codebook Reporting” on page 2-45.

### `table` — Codebook selection table

‘StdCodebook4Tx’ (default) | ‘AltCodebook4Tx’ | optional

Codebook selection table for four transmission antennas, specified as `'StdCodebook4Tx'` or `'AltCodebook4Tx'`. `table` is optional and only applicable when `p` = 4. For more information, see “CSI Codebook Reporting” on page 2-45.

Data Types: `char` | `string`

## Output Arguments

### `out` — Precoding matrix associated with CSI reporting

complex-valued numeric matrix

Precoding matrix associated with CSI reporting, returned as a complex-valued numeric p-by-nu matrix, where p is the number of transmission antennas, and nu is the number of transmission layers. nu must always be less than or equal to p. For more information, see “CSI Codebook Reporting” on page 2-45.

Data Types: `double`
Complex Number Support: Yes

## More About

### CSI Codebook Reporting

A UE reports the precoding matrix indicator (PMI) according to the feedback modes as described in TS 36.213 [1], Section 7.2.4. `lteCSICodebook` returns the precoding matrix for CSI reporting as a p-by-nu matrix, where p is the number of transmission antennas (CSI-RS or CRS ports) and nu is the number of transmission layers (PDSCH transmission layers). nu must always be less than or equal to p. Inputs to the function are nu, p, `idx` indicating the codebook indices, and optionally `table` indicating the codebook selection table.

| Reference Signal | Codebook | Number of layers, nu | p | Codebook indices, idx (See Note 1) | | table | Transmission scheme |
|---|---|---|---|---|---|---|---|
| | | | | i1 | i2 | | |
| CSIRefP | TS 36.213, Table 7.2.4-1 | 1 | 8 | 0–15 | 0–15 | n/a | 'Port7-14' |
| | TS 36.213, Table 7.2.4-2 | 2 | | | | | |
| | TS 36.213, Table 7.2.4-3 | 3 | | 0–3 | | | |
| | TS 36.213, Table 7.2.4-4 | 4 | | | 0–7 | | |
| | TS 36.213, Table 7.2.4-5 | 5 | | | 0 | | |
| | TS 36.213, Table 7.2.4-6 | 6 | | | | | |

| Reference Signal | Codebook | Number of layers, nu | p | Codebook indices, idx (See Note 1) | | table | Transmission scheme |
|---|---|---|---|---|---|---|---|
| | | | | i1 | i2 | | |
| | TS 36.213, Table 7.2.4-7 | 7 | | | | | |
| | TS 36.213, Table 7.2.4-8 | 8 | | 0 | | | |
| CellRefP or CSIRefP (See Note 2) | TS 36.213, Table 7.2.4-0A | 1 | 4 | 0–15 | 0–15 | 'AltCodebook4Tx' | 'Port7-8' or 'Port7-14' (See Note 2) |
| | TS 36.213, Table 7.2.4-0B | 2 | | | | | |
| | TS 36.213, Table 7.2.4-0C | 3 | | 0 | | | |
| | TS 36.213, Table 7.2.4-0D | 4 | | | | | |
| | TS 36.211, Table 6.3.4.2.3-2 | 1–4 | 4 | 0 | 0–15 | 'StdCodebook4Tx' | |
| | TS 36.211, Table 6.3.4.2.3-1 | 1 | 2 | | 0–3 | n/a | |
| | | 2 | | | 0–2 | | |
| | Precoding matrix = 1 | 1 | 1 | | 0 | | any single antenna |

**Note**

**1** Preferred format for codebook indices, idx, is a two element vector, [i1 i2]. A scalar format is accepted when the only available setting for i1 is zero. For this case, use the applicable scalar input range shown for i2.

**2** CellRefP for the 'Port7-8' or CSIRefP for the 'Port7-14'

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.213. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePMISelect` | `ltePMIInfo` | `ltePDSCH` | `ltePDSCHDecode` | `lteDLPrecode`

# lteCSIRS

Channel state information reference signal

## Syntax

```
sym = lteCSIRS(enb)
sym = lteCSIRS(enb,opts)
```

## Description

`sym = lteCSIRS(enb)` returns the channel state information reference signal (CSI-RS) symbols for transmission in a single subframe on up to eight antenna ports ($p$ = 15,...,22). See “lteCSIRS Processing” on page 2-51.

`sym = lteCSIRS(enb,opts)` formats the returned symbols using options specified by `opts`.

## Examples

### Create CSI-RS Symbols and Combine with Resource Grid

Generate CSI-RS symbols and combine them with a 10 MHz, release 8, port 0 PDSCH subframe resource grid.

Initialize a reference channel structure. Create a 10 MHz, release 8, port 0 PDSCH configuration parameter structure. Set subframe number to 1, number of CSI-RS antenna ports to 8, CSI-RS configuration to 0, and CSIRSPeriod to 6.

```
rmc = lteRMCDL('R.2','FDD',1);
rmc.NSubframe = 1;
rmc.CSIRefP = 8;
rmc.CSIRSConfig = 0;
rmc.CSIRSPeriod = 6;
```

The 8 antenna ports are ports 15 to 22. The setting for `CSIRSPeriod` is `Icsi-rs`, which equals [ `Tcsi-rs` `Dcsi-rs`]=[10 1].

Create a 3-D resource grid to contain the subframes for all eight CSI-RS ports.

```
rgrid = lteResourceGrid(rmc,rmc.CSIRefP);
```

Write the release 8 port 0 transmission into the first plane of the resource grid.

```
[wave,rgrid(:,:,1)] = lteRMCDLTool(rmc,[1,0,0,1]);
```

Create the CSI-RS symbols for ports 15 to 22. Overwrite all ports included in the port 0 transmission with the actual CSI-RS and unused RE.

```
rgrid(lteCSIRSIndices(rmc,'rs+unused')) = lteCSIRS(rmc,'rs+unused');
```

## Input Arguments

### enb — eNodeB cell-wide settings

structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NDLRB** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **NCellID** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **NSubframe** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **NFrame** | Optional | 0 (default), nonnegative scalar integer | Frame number |
| `CellRefP` is only used when the *`Indexing format`* option for indexing generation is `'rs+unused'` | | | |
| **CellRefP** | Optional | 1 (default), 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **CyclicPrefix** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **DuplexMode** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| The following parameters apply when `DuplexMode` is set to `'TDD'`. | | | |
| **TDDConfig** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **SSC** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |
| **CSIRSPeriod** | Optional | `'On'` (default), `'Off'`, `Icsi-rs` (0,...,154), `[Tcsi-rs Dcsi-rs]`. You can also specify values in a cell array of configurations for each resource.<br><br>See note. | CSI-RS subframe configurations for one or more CSI-RS resources. Multiple CSI-RS resources can be configured from a single common subframe configuration or from a cell array of configurations for each resource. |
| The following CSI-RS resource parameters apply only when `CSIRSPeriod` sets one, or more CSI-RS subframe configurations to any value other than `'Off'`. Each parameter length must be equal to the number of CSI-RS resources required. | | | |
| **CSIRSConfig** | Required | Nonnegative scalar integer | Array CSI-RS configuration indices. See TS 36.211, Table 6.10.5.2-1. |
| **CSIRefP** | Required | 1 (default), 2, 4, 8 | Array of number of CSI-RS antenna ports |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NCSIID** | Optional | Nonnegative scalar integer | CSI-RS scrambling identity. If this field is not present, then `NCellID` is used as the identity. |
| **ZeroPowerCSIRSPeriod** | Optional | `'Off'` (default), `'On'`, `Icsi-rs` (0,...,154), `[Tcsi-rs Dcsi-rs]`. You can also specify values in a cell array of configurations for each resource.<br><br>See note. | Zero power CSI-RS subframe configurations for one or more zero power CSI-RS resource configuration index lists. Multiple zero power CSI-RS resource lists can be configured from a single common subframe configuration or from a cell array of configurations for each resource list. |
| The following zero power CSI-RS resource parameter is only required if one, or more of the above zero power subframe configurations are set to any value other than `'Off'`. | | | |
| **ZeroPowerCSIRSConfig** | Required | 16-bit bitmap character vector or string scalar (truncated if not 16 bits or `'0'` MSB extended), or a numeric list of CSI-RS configuration indices. You can also specify values in a cell array of configurations for each resource. | Zero power CSI-RS resource configuration index lists (TS 36.211 Section 6.10.5.2). Specify each list as a 16-bit bitmap character vector or string scalar (if less than 16 bits, then `'0'` MSB extended), or as a numeric list of CSI-RS configuration indices from TS 36.211 Table 6.10.5.2-1 in the `'4'` CSI reference signal column. Multiple lists can be defined using a cell array of individual lists.<br><br>[1] |

Note:

**1** `CSIRSPeriod` and `ZeroPowerCSIRSPeriod` parameters control the downlink subframes in which the different CSI-RS resources are present. Valid settings include:

- always `'On'`
- always `'Off'`
- scalar subframe configuration index `Icsi-rs` from 0 through 154
- explicit subframe periodicity and offset pair `[Tcsi-rs Dcsi-rs]`

The subframes containing CSI-RS are located with `NSubframe` and the optional `NFrame` parameters. `NSubframe` can be greater than 10; thus `NSubframe` = 11 is equivalent to setting `NSubframe` to 1 and `NFrame` to 1.

For more information, see TS 36.211 [1], Section 6.10.5.3.

**`opts` — Symbol generation options**
character vector | cell array of character vectors | string array

Symbol generation options, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string):

| Option | Values | Description |
|---|---|---|
| Symbol style | `'ind'` (default), `'mat'` | Style for returning CSI-RS symbols, specified as one of the following options.<br>• `'ind'` — returns the CSI-RS symbols as a column vector (default)<br>• `'mat'` — returns the CSI-RS symbols as a matrix, where each column contains symbols for an individual port and CSI-RS configuration. To form a matrix, a column can contain duplicate entries. |
| Symbol format | `'rsonly'` (default), `'rs+unused'` | Format for the returned symbols, specified as one of the following options.<br>• `'rsonly'` — returns only defined CSI-RS symbols (default), both zero and non-zero<br>• `'rs+unused'` — also includes zeros for the resource element (RE) locations that should be unused because they are reserved for CSI-RS on another port. |

**Note** Returned symbols specify the CSI-RS resource values within an N-by-M-by-antennas array. The number of antennas is `max(CSIRefP)` or if zero power CSI-RS are also defined number of antennas is `max(max(CSIRefP),4)`. For the `'rs+unused'` option, the number of antennas used to define the empty REs (either because they are zero power or they are unused in another port) is `max(max(CSIRefP),CellRefP)`.

Example: `'ind rsonly'`, `"ind rsonly"`, `{'ind','rsonly'}`, or `["ind","rsonly"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `sym` — CSI-RS symbols

column vector (default) | matrix

CSI-RS symbols for transmission in a single subframe on up to eight antenna ports, returned as a column vector or matrix of concatenated CSI-RS symbol sequences for each of the `enb.CSIRefP` ports based on the cell-wide parameter settings. The length of `sym` is the number of resource elements. See “lteCSIRS Processing” on page 2-51.

Data Types: `double`
Complex Number Support: Yes

## More About

### lteCSIRS Processing

The `lteCSIRS` function supports the creation of multiple non-zero power CSI-RS resources and zero power CSI-RS.

By default the output symbols are returned as a column vector and are ordered as they should be mapped into the resource elements along with lteCSIRSIndices. If, according to the CSI-RS resource subframe configurations and duplex mode, there are no CSI-RS scheduled in the subframe, then the

output is empty. Optionally the returned symbols can also include zeros representing the resource elements which should be unused since they are reserved for CSI-RS symbols in one or more of the other ports. On assignment into a populated subframe grid, these zeros create empty resource elements for both Release 8, and Release 10 and 11 compatibility. When multiple non-zero power resources and zero power CSI-RS are output, the zero power CSI-RS symbols are first in the concatenated output, followed by the symbols for the ordered set of CSI-RS resources.

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteCSIRSIndices` | `lteCellRS` | `lteDMRS` | `lteEPDCCHDMRS` | `ltePRS` | `ltePRBS`

# lteCSIRSIndices

CSI-RS resource element indices

## Syntax

```
ind = lteCSIRSIndices(enb)
ind = lteCSIRSIndices(enb,opts)
```

## Description

`ind = lteCSIRSIndices(enb)` returns the indices of the channel state information reference signal (CSI-RS) resource elements (RE) for the specified subframe. See “lteCSIRSIndices Processing” on page 2-57.

`ind = lteCSIRSIndices(enb,opts)` formats the returned indices using options specified by `opts`.

## Examples

### Generate Column Vector of CSI-RS RE Indices

Generate a column vector of CSI-RS resource element linear indices for ports 15 to 22 of a 10 MHz downlink subframe 0 resource grid.

Create a 10 MHz, downlink, subframe 0 configuration parameter structure. Set the number of antenna ports to 8, the CSI-RS configuration to 0, and `Icsi-rs` to 5.

```
rmc = lteRMCDL('R.2');
rmc.CSIRefP = 8;
rmc.CSIRSConfig = 0;
rmc.CSIRSPeriod = 5;
```

The 8 antenna ports are ports 15 to 22. The variable `Icsi-rs` = 5 is equivalent to a `[Tcsi-rs Dcsi-rs]` setting of `[10 0]`.

```
csirs1 = lteCSIRSIndices(rmc);
csirs1(1:5)
```

```
ans = 5x1 uint32 column vector

   3010
   3022
   3034
   3046
   3058
```

### Generate Matrix of CSI-RS RE Indices

This example shows how to generate a matrix of CSI-RS RE linear indices for ports 15 to 22 of a 10 MHz downlink subframe 0 resource grid.

Create a 10 MHz, downlink, subframe 0 configuration parameter structure. Set the number of antenna ports to 8, the CSI-RS configuration to 0, and `Icsi-rs` to 5.

```
rmc = lteRMCDL('R.2');
rmc.CSIRefP = 8;
rmc.CSIRSConfig = 0;
rmc.CSIRSPeriod = 5;
```

Generate a matrix of linear indices with eight columns.

```
csirs2 = lteCSIRSIndices(rmc,'mat');
size(csirs2)
```

```
ans = 1×2

    88     8
```

### Generate Used and Unused CSI-RS RE Indices

This example shows how to generate both used and unused CSI-RS RE linear indices for ports 15 to 22 of a 10 MHz downlink subframe 0 resource grid.

Create a 10 MHz, downlink, subframe 0 configuration parameter structure. Set the number of antenna ports to 8, the CSI-RS configuration to 0, and `Icsi-rs` to 5.

```
rmc = lteRMCDL('R.2');
rmc.CSIRefP = 8;
rmc.CSIRSConfig = 0;
rmc.CSIRSPeriod = 5;
```

The 8 antenna ports are ports 15 to 22. The variable `Icsi-rs` = 5 is equivalent to a `[Tcsi-rs Dcsi-rs]` setting of `[10 0]`.

Generate both used and unused CSI-RS RE in all ports.

```
csirs3 = lteCSIRSIndices(rmc,'rs+unused');
csirs3(1:5)
```

```
ans = 5x1 uint32 column vector

   3010
   3022
   3034
   3046
   3058
```

## Input Arguments

### enb — eNodeB cell-wide settings

structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NDLRB** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **NSubframe** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **NFrame** | Optional | 0 (default), nonnegative scalar integer | Frame number |
| `CellRefP` is only used when the *`Indexing format`* option for indexing generation is `'rs+unused'` | | | |
| **CellRefP** | Optional | 1 (default), 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **CyclicPrefix** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **DuplexMode** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| The following apply when `DuplexMode` is set to `'TDD'`. | | | |
| **TDDConfig** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **SSC** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |
| **CSIRSPeriod** | Optional | `'On'` (default), `'Off'`, `Icsi-rs` (0,...,154), `[Tcsi-rs Dcsi-rs]`. You can also specify values in a cell array of configurations for each resource.<br><br>See note. | CSI-RS subframe configurations for one or more CSI-RS resources. Multiple CSI-RS resources can be configured from a single common subframe configuration or from a cell array of configurations for each resource. |
| The following CSI-RS resource parameters apply only when `CSIRSPeriod` sets one, or more CSI-RS subframe configurations to any value other than `'Off'`. Each parameter length must be equal to the number of CSI-RS resources required. | | | |
| **CSIRSConfig** | Required | Nonnegative scalar integer | Array CSI-RS configuration indices. See TS 36.211, Table 6.10.5.2-1. |
| **CSIRefP** | Required | 1 (default), 2, 4, 8 | Array of number of CSI-RS antenna ports |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **ZeroPowerCSIRSPeriod** | Optional | 'Off' (default), 'On', Icsi-rs (0,...,154), [Tcsi-rs Dcsi-rs]. You can also specify values in a cell array of configurations for each resource.<br><br>See note. | Zero power CSI-RS subframe configurations for one or more zero power CSI-RS resource configuration index lists. Multiple zero power CSI-RS resource lists can be configured from a single common subframe configuration or from a cell array of configurations for each resource list. |
| The following zero power CSI-RS resource parameter is required only if one, or more of the other zero power subframe configurations are set to any value other than 'Off'. | | | |
| **ZeroPowerCSIRSConfig** | Required | 16-bit bitmap character vector or string scalar (truncated if not 16 bits or '0' MSB extended), or a numeric list of CSI-RS configuration indices. You can also specify values in a cell array of configurations for each resource. | Zero power CSI-RS resource configuration index lists (TS 36.211 Section 6.10.5.2). Specify each list as a 16-bit bitmap character vector or string scalar (if less than 16 bits, then '0' MSB extended), or as a numeric list of CSI-RS configuration indices from TS 36.211 Table 6.10.5.2-1 in the '4' CSI reference signal column. Multiple lists can be defined using a cell array of individual lists. |

Note:

**1** The CSIRSPeriod and ZeroPowerCSIRSPeriod parameters control the downlink subframes in which the different CSI-RS resources are present. Valid settings include:

- Always 'On'
- Always 'Off'
- Scalar subframe configuration index Icsi-rs from 0 through 154
- Explicit subframe periodicity and offset pair [Tcsi-rs Dcsi-rs]

To locate the subframes containing CSI-RS, use the NSubframe parameter and the optional NFrame parameter. NSubframe can be greater than 10. Thus NSubframe = 11 is equivalent to setting NSubframe to 1 and NFrame to 1.

For more information, see TS 36.211 [1], Section 6.10.5.3.

**opts — Index generation options**

character vector | cell array of character vectors | string array

Index generation options, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for opts when specified as a character vector include (use double quotes for string):

| Option | Values | Description |
|---|---|---|
| Indexing style | `'ind'` (default), `'mat'`, `'sub'` | Style for the returned indices, specified as one of the following options.<br>• `'ind'` — returns the indices as an $N_{RE}$-by-1 vector (default)<br>• `'mat'` — returns the indices as a matrix. If not precoded, each column contains indices for an individual layer/port. If precoded, each column contains symbols for a transmit antenna. To form a matrix, a column can contain duplicate entries.<br>• `'sub'` — returns the indices as an $N_{RE}$-by-3 matrix. in `[subcarrier, symbol, antenna]` subscript row style.<br>$N_{RE}$ is the number of resource elements. |
| Index base | `'1based'` (default), `'0based'` | Base value of the returned indices. Specify `'1based'` to generate indices where the first value is 1. Specify `'0based'` to generate indices where the first value is 0. |
| Indexing format | `'rsonly'` (default), `'rs+unused'` | Format for the returned locations, specified as one of the following options.<br>• `'rsonly'` — returns only defined CSI-RS locations (default), both zero and non-zero<br>• `'rs+unused'` — also includes zeros for the resource element (RE) locations that should be unused because they are reserved for CSI-RS on another port. |

**Note** Returned indices specify the CSI-RS resource values within an N-by-M-by-antennas array. Where the number of antennas is `max(CSIRefP)` or if zero power CSI-RS are also defined number of antennas is `max(max(CSIRefP),4)`. In the case of the 'rs+unused' option, the number of antennas used to define the empty REs (either because they are zero power or they are unused in another port) is `max(max(CSIRefP),CellRefP)`.

Example: `'ind rsonly'`, `"ind rsonly"`, `{'ind','rsonly'}`, or `["ind","rsonly"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `ind` — Channel state information reference signal (CSI-RS) indices

column vector (default) | matrix

Channel state information reference signal (CSI-RS) indices, returned as a vector or matrix. See “lteCSIRSIndices Processing” on page 2-57.

Data Types: `uint32`

## More About

### lteCSIRSIndices Processing

The `lteCSIRSInidices` function supports the creation of multiple non-zero power resources and zero power CSI-RS.

By default the output indices as re returned as a column vector in one-based linear indexing form, that can directly index elements in an *N*-by-*M*-by-*max(CSIRefP)* array. These indices represent the

subframe grid across *max(CSIRefP)* antenna ports *(p = 15,...,22)*. Other index representations can also be created as well as whether the output includes the RE that should be empty in a specific port because of CSI-RS transmissions in another port. These indices are ordered as the complex CSI-RS symbols should be mapped and do not include any elements allocated to PBCH, PSS, and SSS. You can define the CSI-RS subframe configuration schedule as required for the CSI-RS resources. If the subframe contains no CSI-RS, then an empty vector is returned. When multiple non-zero power and zero power CSI-RS are returned, the indices for the zero power CSI-RS appear first in the concatenated output, followed by the indices for the ordered set of CSI-RS resources.

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteCSIRS` | `lteCellRSIndices` | `lteDMRSIndices` | `ltePRSIndices`

# lteCellRS

Cell-specific reference signal

## Syntax

```
sym = lteCellRS(enb)
sym = lteCellRS(enb,ports)
```

## Description

`sym = lteCellRS(enb)` returns cell-specific reference signal symbols for cell-wide settings in the `enb` structure. `sym` is a complex-valued column vector containing cell-specific reference signal symbols. Unlike other physical channels and signals, the symbols for multiple antennas are concatenated into a single column rather than returned in a matrix with a column for each antenna. The reason for this behavior is that the number of symbols varies across the antenna ports.

`sym = lteCellRS(enb,ports)` returns cell-specific reference signal symbols for antenna ports specified in the vector, `ports` (0,1,2,3), and cell-wide settings structure, `enb`. In this case, `CellRefP` is ignored if present in `enb` and `ports` is used instead.

## Examples

### Find Length of Cell-Specific Reference Signals

This example shows different numbers of cell-specific reference signal symbols transmitted at antenna port 0 and 2.

Initialize cell wide parameter structure, `enb`, to RMC R.6

```
enb = lteRMCDL('R.6');
```

Observe the number of cell-specific reference symbols on port 0

```
cellRefPort0 = length(lteCellRS(enb,0))
```

```
cellRefPort0 = 200
```

Observe the number of cell-specific reference symbols on port 2

```
cellRefPort2 = length(lteCellRS(enb,2))
```

```
cellRefPort2 = 100
```

## Input Arguments

### enb — eNodeB cell-wide settings

structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NDLRB** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **NCellID** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **NSubframe** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **CellRefP** | Required | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **CyclicPrefix** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **DuplexMode** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| The following parameters are dependent upon the condition that `DuplexMode` is set to `'TDD'`. | | | |
| **TDDConfig** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **SSC** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |

**`ports` — Antenna ports**
`0` (default) | 1 | 2 | 3 | numeric vector | optional

Antenna ports, specified as a numeric vector whose elements must be (0, 1, 2, 3).

## Output Arguments

**`sym` — Cell-specific reference signal symbols**
complex-valued numeric column vector

Cell-specific reference signal symbols, returned as a complex-valued numeric column vector. This argument contains cell-specific reference signal symbols for the specified cell-wide settings, `enb`, and optional number of antenna ports, `ports`.

Data Types: `double`

# Version History

**Introduced in R2014a**

## See Also

`lteCellRSIndices` | `lteDMRS` | `lteEPDCCHDMRS` | `ltePRS` | `lteCSIRS` | `ltePRBS`

# lteCellRSIndices

CRS resource element indices

## Syntax

```
ind = lteCellRSIndices(enb)
ind = lteCellRSIndices(enb,opts)
ind = lteCellRSIndices(enb,ports)
ind = lteCellRSIndices(enb,ports,opts)
```

## Description

`ind = lteCellRSIndices(enb)` returns a column vector of resource element (RE) indices for the cell-specific reference signal (RS), given the cell-wide settings in the `enb` structure. By default, the indices are returned in 1-based linear indexing form that can directly index elements of a 3-D array representing the subframe resource grid for all antenna ports. These indices are ordered as the reference signal modulation symbols should be mapped. Unlike other physical channels and signals, the indices for multiple antennas are concatenated into a single column rather than returned in a matrix with a column for each antenna. The indices for each antenna are concatenated because the number of indices varies across the antenna ports.

`ind = lteCellRSIndices(enb,opts)` returns RE indices in a format specified by `opts`.

`ind = lteCellRSIndices(enb,ports)` returns RE indices for antenna ports specified in the vector `ports`. In this case, the `CellRefP` field of `enb` is ignored, and `ports` is used instead.

`ind = lteCellRSIndices(enb,ports,opts)` returns RE indices for the specified antenna ports and formatting options.

## Examples

### Generate Cell-Specific Reference Signal RE Indices

Generate zero-based cell-specific reference signal (CRS) resource element (RE) indices in subscript form for antenna port 2.

```
enb = lteRMCDL('R.0');
enb.NCellID = 10;
ind = lteCellRSIndices(enb,2,{'0based','sub'});
ind(1:4,:)
```

```
ans = 4x3 uint32 matrix

    4    1    2
   10    1    2
   16    1    2
   22    1    2
```

In this case, each row of the generated matrix has three columns, which represent subcarrier, symbol, and antenna port, respectively.

## Input Arguments

### `enb` — eNodeB cell-wide settings

structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NDLRB`** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **`NCellID`** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **`CellRefP`** | Required | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **`CyclicPrefix`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **`DuplexMode`** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| The following parameters are dependent upon the condition that `DuplexMode` is set to `'TDD'`. | | | |
| **`Nsubframe`** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **`TDDConfig`** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **`SSC`** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |

### `ports` — Antenna ports

numeric vector

Antenna ports, specified as a numeric vector whose elements must be (0, 1, 2, 3).

### `opts` — Index generation options

character vector | cell array of character vectors | string array

Index generation options, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string):

| Option | Values | Description |
|---|---|---|
| Indexing style | `'ind'` (default), `'sub'` | Style for the returned indices, specified as one of the following options.<br>• `'ind'` — returns the indices in linear index form as a column vector (default)<br>• `'sub'` — returns the indices in `[subcarrier, symbol, antenna]` subscript row style. The number of rows in the output, `ind`, is the number of resource elements ($N_{RE}$). Thus, `ind` is an $N_{RE}$-by-3 matrix. |
| Index base | `'1based'` (default), `'0based'` | Base value of the returned indices. Specify `'1based'` to generate indices where the first value is 1. Specify `'0based'` to generate indices where the first value is 0. |

Example: `'ind 0based'`, `"ind 0based"`, `{'ind','0based'}`, `["ind","0based"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `ind` — Cell-specific reference signal RE indices

column vector | numeric matrix

Cell-specific reference signal RE indices, returned as a column vector. Optionally, can be returned as an NRE-by-3 matrix.

Data Types: `uint32`

# Version History

**Introduced in R2014a**

## See Also

`lteCellRS` | `lteCSIRSIndices` | `lteDMRSIndices` | `ltePRSIndices`

# lteCellSearch

Cell identity search using PSS and SSS

## Syntax

```
[cellid,offset,peak] = lteCellSearch(enb,waveform)
[cellid,offset,peak] = lteCellSearch(enb,waveform,alg)
[cellid,offset,peak] = lteCellSearch(enb,waveform,cellids)
```

## Description

`[cellid,offset,peak] = lteCellSearch(enb,waveform)` returns the cell identity carried by the PSS and SSS signals in the input waveform, the timing offset to the start of the first frame of the waveform, and the peak correlation magnitude. The cell-wide settings structure, `enb`, defines the link configuration.

`[cellid,offset,peak] = lteCellSearch(enb,waveform,alg)` takes an additional input structure, `alg`, which provides control over the cell search. The input structure, `alg`, contains optional fields to define the SSS detection method, the maximum number of cells to detect, and which cell identities to search.

`[cellid,offset,peak] = lteCellSearch(enb,waveform,cellids)` uses an additional input to constrain the cell search to the list of cell identities specified by in `cellids`.

---

**Note** This syntax will be removed in a future release. Instead use the syntax [`cellid`,`offset`,`peak`] = lteCellSearch(`enb`,`waveform`,`alg`) and set `alg.CellIDs = cellids`.

---

## Examples

### Find Cell Identity

Search for the cell identity (in this case 171) of an `R.12` RMC waveform.

Initialize reference channel configuration, `rmc`. Perform cell search on the waveform produced using this configuration.

```
rmc = lteRMCDL('R.12');
rmc.NCellID = 171;
rmc.TotSubframes = 1;

cellID = lteCellSearch(rmc,lteRMCDLTool(rmc,[1;0;0;1]))
```

```
cellID = 171
```

## Input Arguments

### `enb` — eNodeB cell-wide settings

structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NDLRB`** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **`CyclicPrefix`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **`Duplexmode`** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |

### `waveform` — Time-domain waveform

numeric matrix

Time-domain waveform, specified as a numeric matrix of size *T*-by-*P*. Where *T* is the number of time-domain samples and *P* is the number of receive antennas. The sampling rate of the time domain waveform must be the same as used in the `lteOFDMModulate` function for the specified number of resource blocks `enb.NDLRB`. The number of time domain samples, *T*, must be sufficient to provide at least one subframe for FDD (or 2 for TDD since in TDD mode PSS and SSS lie in adjacent subframes). For the cell search to succeed, the waveform provided must contain the PSS and SSS signals.

**Note** `enb.NDLRB` is only required to specify the sampling rate of `waveform`.

Data Types: `double`
Complex Number Support: Yes

### `alg` — Cell search algorithm control

structure

Cell search algorithm control, specified as a structure. `alg` accepts these fields defining optional cell search algorithm settings.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`SSSDetection`** | Optional | `'PreFFT'` (default), `'PostFFT'`. | SSS detection method. [a] |
| **`MaxCellCount`** | Optional | Nonnegative scalar integer. (1, ..., 504), default 1. | The number of cell identities to detect. [b] |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| `CellIDs` | Optional | Vector of nonnegative integers, default vector (0:503). | A vector containing the cell identities to use for the cell search. [c] |

a `'PostFFT'` SSS detection operates in the frequency domain. For `'PostFFT'`:

- OFDM demodulation is performed using the timing estimate from PSS detection,
- the demodulated SSS resource elements are correlated with possible SSS sequences to find the cell identity group,
- and the peak correlation magnitude is the sum of the peak correlation magnitudes from time-domain PSS detection and frequency-domain SSS detection.

b When `alg.MaxCellCount > 1`, the returned `cellid`, `offset`, and `peak` are vectors, with each element corresponding to one cell.

c If `alg.CellIDs` is absent, the output vectors are sorted by decreasing correlation peak magnitude, that is, decreasing `peak` value. If `alg.CellIDs` is present and `alg.MaxCellCount = numel(alg.CellIDs)`, the output vectors are in the same order as the cell identities in `alg.CellIDs`. Sorting the peaks enables monitoring of the `peak` output for a predetermined set of cells.

### `cellids` — Cell identities

nonnegative scalar integer | vector of nonnegative integers

Cell identities to be used in the cell search, specified as a nonnegative scalar integer or vector of nonnegative integers.

**Note** `cellids` and the syntax it is associated with will be removed in a future release. Instead use `alg.CellIDs` and the recommended alternate syntax.

Data Types: `double`

## Output Arguments

### `cellid` — Cell identity

nonnegative scalar integer | vector of nonnegative integers

Physical layer cell identity, returned as a nonnegative scalar integer or vector of nonnegative integers. `cellid` indicates the detected cell identity. The returned `cellid` is a vector when `alg.MaxCellCount > 1` and more than one cell is detected.

The overall physical layer cell identity is `cellid = (3*`$N_{id1}$`) + `$N_{id2}$. PSS conveys the second cell identity number (`N`$_{id2}$`, (0,1,2))` within a cell identity group and is established via time-domain correlation using the `lteDLFrameOffset` function. SSS conveys the first cell identity number (`N`$_{id1}$`, (0,...,167))` and is established in a similar fashion.

Data Types: `double`

### `offset` — Timing offset

nonnegative scalar integer | vector of nonnegative integers

Timing offset, returned as a nonnegative scalar integer or vector of nonnegative integers. `offset` indicates the number of samples from the start of the input waveform to the position in that waveform where the first frame begins. The timing offset is calculated by correlating with the detected PSS and SSS. The returned `offset` is a vector when `alg.MaxCellCount > 1` and more than one cell is detected.

Data Types: `double`

**peak — Peak magnitude**
numeric scalar | vector of numeric values

Peak magnitude of the correlation, returned as a numeric scalar or vector of numeric values, used for cell detection. The returned `peak` is a vector when `alg.MaxCellCount > 1` and more than one cell is detected. The peak correlation magnitude is the sum of the peak correlation magnitudes from PSS and SSS detection. A complete correlation output is available as the output argument, `corr`, from `lteDLFrameOffset`.

# Version History

**Introduced in R2014a**

## See Also

`lteDLFrameOffset` | `lteFrequencyOffset` | `lteFrequencyCorrect` | `lteOFDMDemodulate`

# lteCodeBlockDesegment

Code block desegmentation and CRC decoding

## Syntax

```
[blk,err] = lteCodeBlockDesegment(cbs,blklen)
[blk,err] = lteCodeBlockDesegment(cbs)
```

## Description

`[blk,err] = lteCodeBlockDesegment(cbs,blklen)` concatenates the input code block vectors contained in `cbs` into an output vector, `blk`, of length `blklen`. `blklen` is also used to validate the dimensions of the data in `cbs` and to calculate the amount of filler to be removed. If `cbs` is a cell array containing more than one vector, each vector is assumed to have a type-24B CRC attached. This CRC is decoded and stripped from each code block before output concatenation and the CRC error result is placed in the associated element of vector `err`. The length of `err` is the number of code blocks. If `cbs` is a single vector or a cell array containing a single vector, no CRC decoding or stripping is performed and `err` is empty. In all cases, the number of filler bits stripped from the beginning of the (first) code block is calculated from `blklen`. `lteCodeBlockDesegment` performs the inverse of the code block segmentation and CRC appending (see `lteCodeBlockSegment`).

`[blk,err] = lteCodeBlockDesegment(cbs)` no leading filler bits are stripped from the output.

## Examples

### Desegment Code Block

Perform code block desegmentation and discover when segmentation occurs.

Code block segmentation occurs if the input length is greater than 6144. The input vector of length 6145 is segmented by `lteCodeBlockSegment` into two vectors of length 3072 and 3136.

```
cbs = lteCodeBlockSegment(ones(6145,1));
```

Next, perform desegmentation and CRC removal.

```
[blk,err] = lteCodeBlockDesegment(cbs);
size(blk)
```

```
ans = 1×2

        6160           1
```

```
err
```

```
err = 1x2 int8 row vector

   0   0
```

The first output, `blk`, is a column vector of length 6160. The second output, `err`, is a column vector of zero values.

## Input Arguments

### `cbs` — Code block segments

column vector | cell array

Code block segments, specified as a column vector or cell array of column vectors. If `cbs` is a cell array containing more than one vector, each vector is assumed to have a type-24B CRC attached. This CRC is decoded and stripped from each code block before output concatenation and the CRC error result is placed in the associated element of vector `err`. The length of `err` is the number of code blocks. If `cbs` is a single vector or a cell array containing a single vector, no CRC decoding or stripping is performed and `err` is empty. In all cases, the number of filler bits stripped from the beginning of the (first) code block is calculated from `blklen`.

### `blklen` — Block length

nonnegative integer

Block length, specified as a nonnegative integer.

## Output Arguments

### `blk` — Output data block

column vector

Output data block, returned as a column vector. The input code blocks are segmented into a single output data block, `blk`, removing any filler and type-24B CRC bits.

Data Types: `int8`

### `err` — Code block CRC decoding errors

column vector | nonnegative integer

Code block CRC decoding errors, returned as a nonnegative integer. The length of `err` is equal to the number of code blocks. If `cbs` is a cell array containing multiple vector elements, `lteCodeBlockDesegment` assumes that each vector has a type-24B CRC attached. The CRC is decoded and stripped from each code block before output concatenation and the CRC error result is placed in the associated element of `err`. If `cbs` is a single vector or a cell array containing a single vector, no CRC decoding or stripping is performed and `err` is empty.

Data Types: `int8`

# Version History

**Introduced in R2014a**

## See Also

`lteCodeBlockSegment` | `lteTurboDecode` | `lteCRCDecode` | `lteDLSCHDecode` | `lteULSCHDecode`

# lteCodeBlockSegment

Code block segmentation and CRC attachment

## Syntax

```
cbs = lteCodeBlockSegment(blk)
```

## Description

`cbs = lteCodeBlockSegment(blk)` splits the input data bit vector `blk` into a cell array `cbs` of code block segments, with filler bits and type-24B CRC appended as appropriate, according to the rules of TS 36.212 [1], Section 5.1.2. Code block segmentation occurs in transport blocks, after initial type-24A CRC appending, for turbo encoded transport channels, including DL-SCH, UL-SCH, PCH, and MCH.

The segmentation and padding operation ensures that code blocks entering the turbo coder are no larger than 6144 in length and are all legal turbo code blocks sizes. The LTE turbo coder only supports a finite set of code block sizes. If the input block length is greater than 6144, the input block is split into a cell array of smaller code blocks where each individual block also has a type-24B CRC appended to it. The NULL filler bits, represented by –1 at the output, are prepended to the first code block so that all blocks in the set have acceptable lengths. If the input block length is less than or equal to 6144, no segmentation occurs and no CRC is appended, but the single output code block may have NULL filler bits prepended. The latter case still results in a cell array output containing a single vector.

## Examples

### Segment Code Block

Perform code block segmentation, providing two vectors with different lengths.

Code block segmentation occurs if the input length is greater than 6144. Provide a vector of length 6144.

```
cbs1 = lteCodeBlockSegment(ones(6144,1))
```

```
cbs1 = 1x1 cell array
    {6144x1 int8}
```

No segmentation occurs.

Provide a vector of length 6145.

```
cbs2 = lteCodeBlockSegment(ones(6145,1))
```

```
cbs2=1×2 cell array
    {3072x1 int8}    {3136x1 int8}
```

Segmentation occurs for input length greater than 6144.

## Input Arguments

### `blk` — Data bit vector

column vector

Data bit vector, specified as a column vector.

## Output Arguments

### `cbs` — Code block segments

cell array of integer column vectors

Code block segments, returned as a cell array with `int8` column vector elements. If the input block length is less than or equal to 6144, `cbs` is a cell array containing a single column vector. If the input block length is greater than 6144, `cbs` is a cell array of multiple column vectors.

Data Types: `cell`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: http://www.3gpp.org.

## See Also

`lteCodeBlockDesegment` | `lteCRCEncode` | `lteTurboEncode` | `lteDLSCHInfo` | `lteDLSCH`

# lteConvolutionalDecode

Convolutional decoding

## Syntax

```
out = lteConvolutionalDecode(softBits)
```

## Description

`out = lteConvolutionalDecode(softBits)` returns `out`, data recovered by convolutionally decoding `softBits`, a vector of soft bits.

The decoder uses a soft input wrap-around Viterbi algorithm without any quantization. The algorithm creates training data to append to the start and end of the packet by cyclically extending the packet. The traceback decoding length is 42.

## Examples

### Perform Convolutional Decoding

Generate random bits and convolutionally encode them.

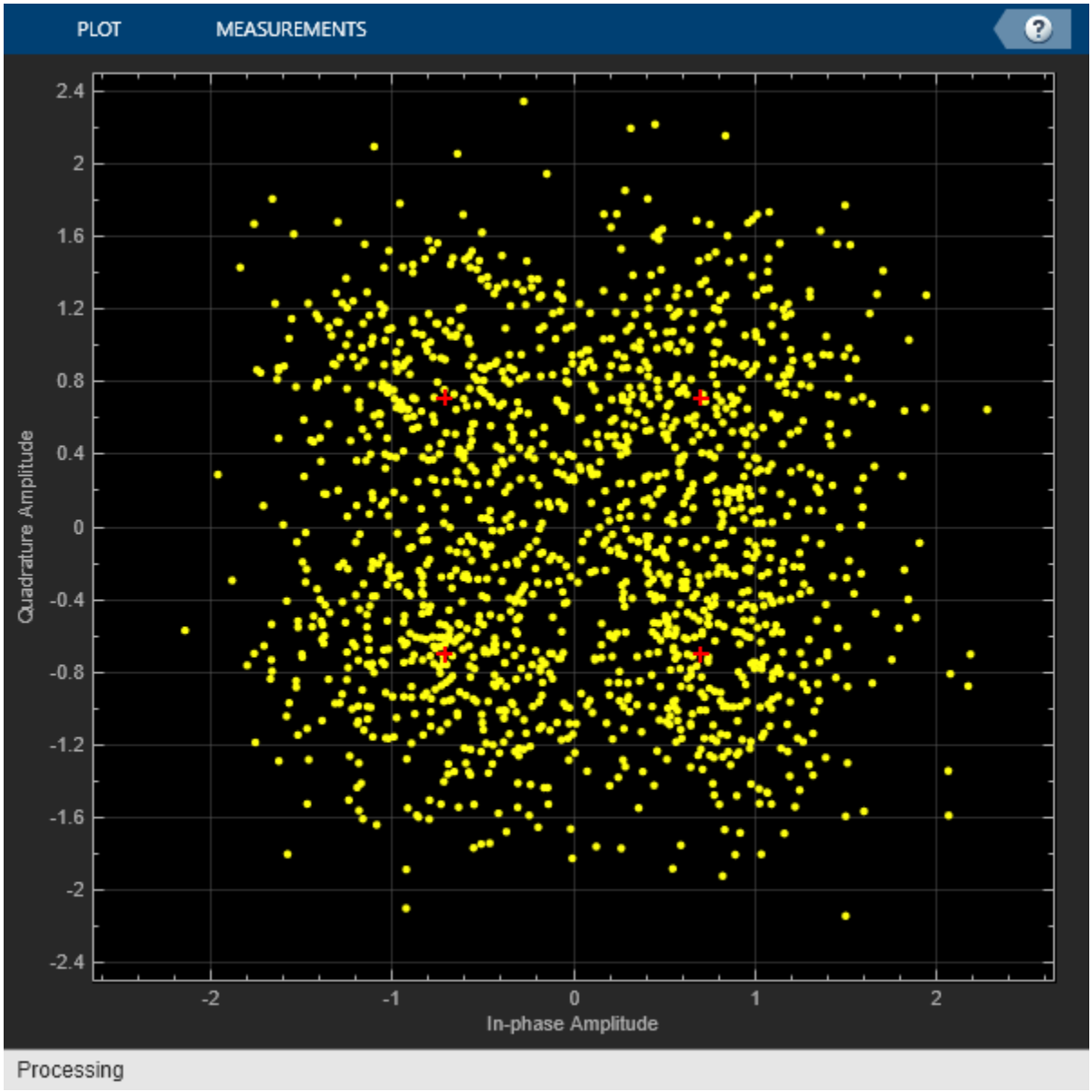
```
txBits = randi([0 1],1000,1);
codedData = lteConvolutionalEncode(txBits);
```

QPSK modulate the coded bits and add noise to the received symbols.

```
txSym = lteSymbolModulate(codedData,'QPSK');
noise = 0.5*complex(randn(size(txSym)),randn(size(txSym)));
rxSym = txSym + noise;
```

Show `rxSymbols` constellation, setting `txSymbols` as the reference constellation.

```
xylimits = [-2.5 2.5];
cdScope = comm.ConstellationDiagram('ReferenceConstellation',txSym,'XLimits',xylimits ,'YLimits'
cdScope(rxSym)
```




Demodulate the noisy symbols to obtain soft bits, convolutionally decode the soft bits, and display the number of erroneous bits.

```
softBits = lteSymbolDemodulate(rxSym,'QPSK','Soft');
out = lteConvolutionalDecode(softBits);
disp(sum(out ~= int8(txBits)))
```

```
     0
```

## Input Arguments

### `softBits` — Soft bit data

column vector

Soft bit data, specified as a column vector. The function assumes that input data has been encoded by a tail-biting convolutional code with constraint length 7, coding rate 1/3, and octal polynomials *G0* = 133, *G1* = 171 and *G2* = 165. The function also assumes that the input data vector is structured as three encoded parity streams concatenated block-wise in the form `[D0 D1 D2]`, where `D0`, `D1`, and `D2` are the separate parity streams resulting from the original encoding with individual polynomials *G0*, *G1*, and *G2*.

Data Types: `double`

## Output Arguments

**`out` — Convolutionally decoded data**
column vector

Convolutionally decoded data, returned as a column vector. The length of this vector is 1/3 of the length of the `softBits` input.

Data Types: `int8`

# Version History

**Introduced in R2014a**

## See Also

`lteConvolutionalEncode` | `lteTurboDecode` | `lteRateRecoverConvolutional` | `lteCRCDecode` | `lteBCHDecode` | `lteCQIDecode` | `lteDCIDecode`

# lteConvolutionalEncode

Convolutional encoding

## Syntax

```
output = lteConvolutionalEncode(input)
```

## Description

`output = lteConvolutionalEncode(input)` returns the result of convolutionally encoding the input data vector `input`. The convolutional code has constraint length 7 and is tail biting with coding rate 1/3 and octal polynomials *G0*=133, *G1*=171 and *G2*=165. Because the code is tail-biting, `output` is three times the length of the input. The three encoded parity streams are concatenated block-wise to form the encoded output that is, `out = [D0 D1 D2]` where `D0`, `D1`, and `D2` are the separate vectors resulting from encoding the input `input` with the individual polynomials G0, G1, and G2.

## Examples

### Perform Convolutional Encoding

Perform convolutional encoding and compare the length of the input vector to the length of the output vector.

Perform convolutional encoding of a vector of length 100.

```
coded = lteConvolutionalEncode(ones(100,1));
size(coded)
```

```
ans = 1×2

   300     1
```

The resulting output is a coded vector of length 300, which is three times the length of the input vector, as expected.

## Input Arguments

### `input` — Input data vector

column vector

Input data vector, specified as a column vector.

## Output Arguments

### `output` — Convolutionally encoded data

column vector

Convolutionally encoded data, returned as a column vector. Because the code is tail biting, `output` is three times the length of the input. The three encoded parity streams are concatenated block-wise to form the encoded output that is, `out = [D0 D1 D2]` where `D0`,`D1`, and `D2` are the separate vectors resulting from encoding the input `input` with the individual octal polynomials *G0*=133, *G1*=171, and *G2*=165.

Data Types: `int8`

# Version History

**Introduced in R2014a**

## See Also

`lteConvolutionalDecode` | `lteTurboEncode` | `lteCRCEncode` | `lteRateMatchConvolutional` | `lteBCH` | `lteDCIEncode`

# lteDCI

Downlink control information format structures and bit payloads

## Syntax

```
dciout = lteDCI(enb,dciin)
[dciout,bitsout] = lteDCI(enb,dciin)
[ ___ ] = lteDCI(enb,dciin,opts)
[ ___ ] = lteDCI(enb,chs,dciin,opts)
[ ___ ] = lteDCI(enb,bitsin,opts)
[ ___ ] = lteDCI(enb,chs,bitsin,opts)

[ ___ ] = lteDCI(istr,opts)
```

## Description

`dciout = lteDCI(enb,dciin)` returns the `dciout` structure containing a downlink control information (DCI) message given input structures containing the cell-wide settings and the DCI format setting. With this syntax, the messages created have the minimum possible sizes for the cell configuration (link bandwidths, frame structure, and so on).

This function creates and manipulates DCI messages for the formats defined in TS 36.212 [2], Section 5.3.3. Later releases of the LTE standard may add UE-specific bit fields to a format. By default, any UE-specific bit fields added after a format is first released, appear in the output but are inactive. Uses for `lteDCI` include creation of a default DCI message, blind decoding of DCI format types, and determining the sizes of the bit fields.

For information on link bandwidth assignment, see “Specifying Number of Resource Blocks” on page 2-101.

`[dciout,bitsout] = lteDCI(enb,dciin)` also returns a vector, `bitsout`, representing the set of message fields mapped to the information bit payload (including any zero padding).

`[ ___ ] = lteDCI(enb,dciin,opts)` formats the returned structure through the options specified by `opts`.

This syntax supports output options from prior syntaxes.

`[ ___ ] = lteDCI(enb,chs,dciin,opts)` permits formats to be extended with additional bit fields on a per-UE basis using the UE-specific channel configuration structure, `chs`.

`[ ___ ] = lteDCI(enb,bitsin,opts)` uses `bitsin` to initialize all the message fields. `bitsin` is treated as the DCI information bit payload and directly maps to `bitsout`, (`bitsout == bitsin`). By default the format is deduced directly from the length of `bitsin`. Therefore, the length of `bitsin` must be one of the valid format sizes for the given cell-wide parameters, `enb`. For more information, see `lteDCIInfo`.

When multiple formats have the same payload size, the first matching format is selected. The function checks formats 0 and 1A first, favoring the more likely common search space. If no match is found, the remaining formats are searched in alphanumerical order. To override the blind format matching in this syntax, add an explicit `enb.DCIFormat` field.

`[ ___ ] = lteDCI(enb,chs,bitsin,opts)` permits formats to be extended with additional bit fields on a per-UE basis using the UE-specific channel configuration structure, `chs`. The DCI payload sizes for the combination of cell-wide and UE-specific parameters define the set of valid `bitsin` lengths. For more information, see `lteDCIInfo`.

As with the previous syntax, the format type is deduced from the length of `bitsin`. To override the blind format matching in this syntax, add an explicit `chs.DCIFormat` field.

`[ ___ ] = lteDCI(istr,opts)` accepts an input structure, `istr`. The fields described in the structures `enb` and `dciin` must be present as part of `istr`. In this syntax, `dciout`, also carries forward the `NDLRB` and `DCIFormat` fields supplied in `istr`.

This syntax is not recommended and will be removed in a future release. Instead, use one of the previous syntaxes that separates the parameters into different input structures.

## Examples

### Create DCI

Create a format 1A DCI message structure with the distributed VRB allocation type. The allocation message fields are contained in the `dci1A.Allocation` substructure. When the format 1A `AllocationType` field is properly initialized at the input to the function, the appropriate set of fields is output. For format 1A, setting `AllocationType` to 1 gives a distributed allocation and 0 gives a localized allocation.

```
enb = struct('NDLRB',50,'CellRefP',1,'DuplexMode','FDD');
dciin = struct('DCIFormat','Format1A','AllocationType',1);
dci1A = lteDCI(enb,dciin)
```

```
dci1A = struct with fields:
           DCIFormat: 'Format1A'
                 CIF: 0
      AllocationType: 1
          Allocation: [1x1 struct]
           ModCoding: 0
              HARQNo: 0
             NewData: 0
                  RV: 0
            TPCPUCCH: 0
            TDDIndex: 0
          SRSRequest: 0
    HARQACKResOffset: 0
```

```
allocfields = dci1A.Allocation
```

```
allocfields = struct with fields:
    RIV: 0
    Gap: 0
```

The field values of this structure can be set and passed back through the function. Output the information bits with the new values.

```
dci1A.RV = 1;
dci1A.Allocation.RIV = 6;
dci1Aupdated = lteDCI(enb,dci1A)

dci1Aupdated = struct with fields:
           DCIFormat: 'Format1A'
                 CIF: 0
      AllocationType: 1
          Allocation: [1x1 struct]
           ModCoding: 0
              HARQNo: 0
             NewData: 0
                  RV: 1
            TPCPUCCH: 0
            TDDIndex: 0
          SRSRequest: 0
    HARQACKResOffset: 0


allocfields = dci1Aupdated.Allocation

allocfields = struct with fields:
    RIV: 6
    Gap: 0
```

### Create Format 1 TDD DCI Message

Create a format 1 DCI message structure with the resource allocation type 1 and TDD modulation scheme. Set `AllocationType` to 1, and output the set of allocation fields. `AllocationType` is the resource allocation header bit for format 1. Also initialize the `ModCoding` field at the input. All noninitialized fields default to 0.

```
enb.NDLRB = 50;
enb.CellRefP = 1;
enb.DuplexMode = 'TDD';

dciin.DCIFormat = 'Format1';
dciin.AllocationType = 1;
dciin.ModCoding = 7;

dci1 = lteDCI(enb,dciin)

dci1 = struct with fields:
           DCIFormat: 'Format1'
                 CIF: 0
      AllocationType: 1
          Allocation: [1x1 struct]
           ModCoding: 7
              HARQNo: 0
             NewData: 0
                  RV: 0
            TPCPUCCH: 0
            TDDIndex: 0
```

```
    HARQACKResOffset: 0
```

```
allocfields = dci1.Allocation
```

```
allocfields = struct with fields:
      Bitmap: '00000000000000'
    RBSubset: 0
       Shift: 0
```

For the specified configuration, the `Allocation` substructure includes the character vector bit field, `Bitmap`, plus `RBSubset` and `Shift` fields.

### Create DCI Bit Message

Create a format 1A DCI message structure and output the `bitsout` message. Modify the DCI message and observe the change.

Create cell-wide settings and DCI message settings structures. For the DCI message, assign format 1A and allocation type 0. Generate the DCI message. View the DCI message structure and bits output.

```
enb = struct('NDLRB',25,'CellRefP',1,'DuplexMode','FDD');
dciin = struct('DCIFormat','Format1A','AllocationType',0);

[dciout,bitsout] = lteDCI(enb,dciin);

dciout
```

```
dciout = struct with fields:
           DCIFormat: 'Format1A'
                 CIF: 0
      AllocationType: 0
          Allocation: [1x1 struct]
           ModCoding: 0
              HARQNo: 0
             NewData: 0
                  RV: 0
            TPCPUCCH: 0
            TDDIndex: 0
          SRSRequest: 0
    HARQACKResOffset: 0
```

```
bitsout'
```

```
ans = 1x25 int8 row vector

   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

The first bit in `bitsout` is a 1 for DCI message format 1A. The second bit is 0 for `AllocationType` = 0.

Modify the allocation type to 1. Regenerate the DCI message. View the DCI message structure and bits output.

```
dciin = struct('DCIFormat','Format1A','AllocationType',1);

[dciout,bitsout] = lteDCI(enb,dciin);

dciout
```

```
dciout = struct with fields:
           DCIFormat: 'Format1A'
                 CIF: 0
      AllocationType: 1
          Allocation: [1x1 struct]
           ModCoding: 0
              HARQNo: 0
             NewData: 0
                  RV: 0
            TPCPUCCH: 0
            TDDIndex: 0
          SRSRequest: 0
    HARQACKResOffset: 0
```

```
bitsout'
```

```
ans = 1x25 int8 row vector

   1   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

Note the `AllocationType` and the second bit of `bitsout` both changed from 0 to 1.

Modify the DCI message format to 0. Regenerate the DCI message. View the DCI message structure and bits output.

```
dciin = struct('DCIFormat','Format0','AllocationType',1);

[dciout,bitsout] = lteDCI(enb,dciin);

dciout
```

```
dciout = struct with fields:
          DCIFormat: 'Format0'
                CIF: 0
         Allocation: [1x1 struct]
          ModCoding: 0
            NewData: 0
                TPC: 0
         CShiftDMRS: 0
           TDDIndex: 0
         CSIRequest: 0
         SRSRequest: 0
    AllocationType: 1
```

```
bitsout'
```

```
ans = 1x25 int8 row vector

   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0
```

The first bit in `bitsout` change from 1 to 0. Because the message formats 0 and 1A have the same length, the first bit in `bitsout` is used to distinguish these formats. For all other formats, the message length is used to distinguish the format types. For format 0, the setting for `AllocationType` is specified by bit number 24.

### Optional DCI Message Views

Create a format 1 DCI message structure and supply the optional `'fieldsizes'` and `'excludeunusedfields'` inputs. By default, the output structure contains all possible fields for the input format. Not all fields are active for the given input parameters. Specifically, some might not be present in the payload bits. To see the number of bits associated with each field, use the optional `'fieldsizes'` input. The `'fieldsizes'` option also adds the `'Padding'` field to the output indicating the number of padding bits.

```
enb.NDLRB = 50;
enb.CellRefP = 1;
enb.DuplexMode = 'TDD';

dciin.DCIFormat = 'Format1';
dciin.AllocationType = 1;
dciin.ModCoding = 7;

opts = {'fieldsizes'}
```

```
opts = 1x1 cell array
    {'fieldsizes'}
```

```
dci1 = lteDCI(enb,dciin,opts)
```

```
dci1 = struct with fields:
           DCIFormat: 'Format1'
                 CIF: 0
      AllocationType: 1
          Allocation: [1x1 struct]
           ModCoding: 5
              HARQNo: 4
             NewData: 1
                  RV: 2
            TPCPUCCH: 2
            TDDIndex: 2
    HARQACKResOffset: 0
             Padding: 0
```

```
allocfields = dci1.Allocation
```

```
allocfields = struct with fields:
      Bitmap: 14
    RBSubset: 2
       Shift: 1
```

View the output to see the sizes for all DCI message fields.

Remove unused (0 bit) fields from the output structure by using the `'excludeunusedfields'` option.

```
opts = {'fieldsizes','excludeunusedfields'}
```

```
opts = 1x2 cell
    {'fieldsizes'}    {'excludeunusedfields'}
```

```
dci1 = lteDCI(enb,dciin,opts)
```

```
dci1 = struct with fields:
         DCIFormat: 'Format1'
    AllocationType: 1
        Allocation: [1x1 struct]
         ModCoding: 5
            HARQNo: 4
           NewData: 1
                RV: 2
          TPCPUCCH: 2
          TDDIndex: 2
```

```
allocfields = dci1.Allocation
```

```
allocfields = struct with fields:
      Bitmap: 14
    RBSubset: 2
       Shift: 1
```

The output fields with bit length equal to zero bits no longer appear in the output.

### Blindly Recover Modified Format 1A DCI Message

Create a format 1A DCI message structure with the distributed VRB allocation type. The `Allocation` substructure contains the allocation message fields. To specify a distributed allocation, set the format 1A `AllocationType` field to 1. To specify a localized allocation, set the `AllocationType` field to 0.

```
enb.NDLRB = 50;
enb.CellRefP = 1;
enb.DuplexMode = 'FDD';
dciin.DCIFormat = 'Format1A';
dciin.AllocationType = 1;
[dci1A,bits] = lteDCI(enb,dciin);
disp(dci1A)
```

```
           DCIFormat: 'Format1A'
                 CIF: 0
      AllocationType: 1
          Allocation: [1x1 struct]
           ModCoding: 0
              HARQNo: 0
             NewData: 0
                  RV: 0
```

```
            TPCPUCCH: 0
            TDDIndex: 0
          SRSRequest: 0
    HARQACKResOffset: 0

disp(dci1A.Allocation)

    RIV: 0
    Gap: 0
```

Adjust the `RV` and `RIV` field values of `dci1A`. Call the `lteDCI` function again to update the information bits with the new values. View the updated message fields by blindly recovering them directly from the output DCI message bits.

```
dci1A.RV = 1;
dci1A.Allocation.RIV = 6;
[~,bitsUpdated] = lteDCI(enb,dci1A);
dci1Arec = lteDCI(enb,bitsUpdated);
disp(dci1Arec)

           DCIFormat: 'Format1A'
                 CIF: 0
      AllocationType: 1
          Allocation: [1x1 struct]
           ModCoding: 0
              HARQNo: 0
             NewData: 0
                  RV: 1
            TPCPUCCH: 0
            TDDIndex: 0
          SRSRequest: 0
    HARQACKResOffset: 0

disp(dci1Arec.Allocation)

    RIV: 6
    Gap: 0
```

### Create DCI Message Using UE-Specific Control

Use an additional UE-specific input parameter structure to control UE-specific DCI fields. Create a message to be sent on the EPDCCH that is intended for a UE configured with the carrier indicator field, `CIF`.

Initialize cell-wide structure `enb`, DCI format structure `dciin`, UE-specific structure `chs`, and output options structure `opts`.

```
enb.NDLRB = 50;
enb.CellRefP = 1;
enb.DuplexMode = 'TDD';

dciin.DCIFormat = 'Format1';
dciin.AllocationType = 1;
dciin.ModCoding = 7;

chs.ControlChannelType = 'EPDCCH';
```

```
chs.EnableCarrierIndication = 'On';
chs.EnableSRSRequest = 'Off';
chs.EnableMultipleCSIRequest = 'Off';

opts = {'fieldsizes','excludeunusedfields'}
```

```
opts = 1x2 cell
    {'fieldsizes'}    {'excludeunusedfields'}
```

Create and view the DCI message.

```
dci1 = lteDCI(enb,chs,dciin,opts)
```

```
dci1 = struct with fields:
           DCIFormat: 'Format1'
                 CIF: 3
      AllocationType: 1
          Allocation: [1x1 struct]
           ModCoding: 5
              HARQNo: 4
             NewData: 1
                  RV: 2
            TPCPUCCH: 2
            TDDIndex: 2
    HARQACKResOffset: 2
```

```
allocfields = dci1.Allocation
```

```
allocfields = struct with fields:
      Bitmap: 14
    RBSubset: 2
       Shift: 1
```

Based on the UE-specific settings in `chs`, the output includes the three bit `CIF` field and the two bit `HARQACKResOffset` field. If these fields were present in `dciin`, their values would be mapped into the appropriate positions in the information bits at the output.

### Create DCI Message Using UE-Specific Control And Bit Stream

Use an additional UE-specific input parameter structure to control UE-specific DCI fields. Create a message to be sent on the EPDCCH that is intended for a UE configured with the carrier indicator field, `CIF`.

Initialize cell-wide structure `enb`, UE-specific structure `chs`, and output options structure `opts`.

```
enb.NDLRB = 50;
enb.CellRefP = 1;
enb.DuplexMode = 'TDD';

chs.DCIFormat = 'Format1B';
chs.ControlChannelType = 'EPDCCH';
chs.EnableCarrierIndication = 'On';
chs.EnableSRSRequest = 'Off';
```

```
chs.EnableMultipleCSIRequest = 'Off';
chs.NTxAnts = 1;

opts = {'fieldsizes','excludeunusedfields'};
```

Based on the UE-specific settings in `chs`, the DCI message length is extended to include fields `CIF` (3 bits) and `HARQACKResOffset` (2 bits). Using `lteDCIInfo` and `chs` to determine the correct input bitstream length, create `bitsin`.

```
info = lteDCIInfo(enb,chs);

bitsin = zeros(getfield(info,chs.DCIFormat),1);
```

Create a new DCI message using cell-wide settings, UE-specific Control and `bitsin`.

```
[dciout,bitsout] = lteDCI(enb,chs,bitsin,opts);
dciout
```

```
dciout = struct with fields:
           DCIFormat: 'Format1B'
                 CIF: 3
      AllocationType: 1
          Allocation: [1x1 struct]
           ModCoding: 5
              HARQNo: 4
             NewData: 1
                  RV: 2
            TPCPUCCH: 2
            TDDIndex: 2
                TPMI: 2
                 PMI: 1
    HARQACKResOffset: 2
```

## Input Arguments

### enb — eNodeB cell-wide settings

structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NDLRB** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **NULRB** | Required | Scalar integer from 6 to 110 | Number of uplink resource blocks. ($N_{RB}^{UL}$) |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **DCIFormat** | Required (see syntax descriptions for applicability) | 'Format0', 'Format1', 'Format1A', 'Format1B', 'Format1C', 'Format1D', 'Format2', 'Format2A', 'Format2B', 'Format2C', 'Format2D', 'Format3', 'Format3A', 'Format4', 'Format5', 'Format5A' | Downlink control information (DCI) format |
| **CellRefP** | Optional | 1 (default), 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **DuplexMode** | Optional | 'FDD' (default), 'TDD' | Duplexing mode, specified as either:<br>• 'FDD' for Frequency Division Duplex<br>• 'TDD' for Time Division Duplex |

### dciin — DCI settings

structure

DCI settings, specified as a structure that can contain these fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **DCIFormat** | Required, except when bitsin is input | 'Format0', 'Format1', 'Format1A', 'Format1B', 'Format1C', 'Format1D', 'Format2', 'Format2A', 'Format2B', 'Format2C', 'Format2D', 'Format3', 'Format3A', 'Format4', 'Format5', 'Format5A' | Downlink control information (DCI) format |

Any format-specific fields can be initialized by adding them to dciin. See dciout for specific fields output for each DCIFormat.

### opts — Formatting options for output DCI structure

character vector | cell array of character vectors | string array

Formatting options for output DCI structure, specified as a character vector, cell array of character vectors, or a string array. You can specify a format for the *Field content* and *Fields to include*. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for opts when specified as a character vector include (use double quotes for string):

| Category | Options | Description |
|---|---|---|
| *Field content* | 'fieldvalues' (default) | Set the fields to zero or to their input values. |

| Category | Options | Description |
|---|---|---|
| | `'fieldsizes'` | Sets the field values to their bit sizes and adds the `Padding` field to `dciout`. `Padding` indicates the number of padding bits appended. |
| *Fields to include* | `'includeallfields' (default)` | `dciout` includes all possible fields for the requested DCI format. |
| | `'excludeunusedfields'` | `dciout` excludes fields that have zero length for the given parameter set. |

Example: `'fieldsizes excludeunusedfields'`, `"fieldsizes excludeunusedfields"`, `{'fieldsizes','excludeunusedfields'}`, or `["fieldsizes","excludeunusedfields"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

**`chs` — User-equipment-related channel configuration**
structure

User-equipment-related (UE-related) channel configuration, specified as a structure containing these UE-specific fields.

**Note** All fields in `chs` are optional. The presence of these optional fields depends on:

- Whether the transmission of DCI message is in a PDCCH using common search space mapping or in an EPDCCH.
- The release-specific features configured at the destination UE.

These additional UE-specific bit fields are off by default.

**`DCIFormat` — DCI format name**
`'Format0'` | `'Format1'` | `'Format1A'` | `'Format1B'` | `'Format1C'` | `'Format1D'` | `'Format2'` | `'Format2A'` | `'Format2B'` | `'Format2C'` | `'Format2D'` | `'Format3'` | `'Format3A'` | `'Format4'` | `'Format5''Format5A'`

DCI format name, specified as a character vector or string scalar. For string scalar, use double quotes. See syntax descriptions for applicability.

Data Types: `char` | `string`

**`ChannelControlType` — Physical control channel type**
`'PDCCH'` (default) | `'EPDCCH'` | optional

Physical control channel type used to carry DCI formats, specified as `'PDCCH'` or `'EPDCCH'`. The setting for `ChannelControlType` affects the presence of the HARQ-ACK resource offset field and message padding.

Data Types: `char` | `string`

**`SearchSpace` — Search space mapping**
`'UESpecific'` (default) | `'Common'` | optional

Search space mapping for DCI formats 0/1A/1C, specified as `'UESpecific'` or `'Common'`. This field is only applicable for PDCCH. None of the additional fields can be present when formats 0 or 1A are mapped into the PDCCH common search space.

Data Types: `char` | `string`

**`EnableCarrierIndication` — Option to enable carrier indication**
`'Off'` (default) | `'On'` | optional

Option to enable carrier indication field (CIF) in the UE configuration, specified as `'Off'` or `'On'`. By default, `EnableCarrierIndication` is disabled. When `EnableCarrierIndication` is enabled (`'On'`), the CIF is present in the UE-specific configuration.

Data Types: `char` | `string`

**`EnableSRSRequest` — Option to enable SRS request**
`'Off'` (default) | `'On'` | optional

Option to enable SRS request in the UE configuration, specified as `'Off'` or `'On'`. By default, `EnableSRSRequest` is disabled. When `EnableSRSRequest` is enabled (`'On'`), the SRS request field is present in UE-specific formats 0/1A for FDD or TDD and formats 2B/2C/2D for TDD.

Data Types: `char` | `string`

**`EnableMultipleCSIRequest` — Option to enable multiple CSI requests**
`'Off'` (default) | `'On'` | optional

Option to enable multiple CSI requests in the UE configuration, specified as `'Off'` or `'On'`. By default, `EnableMultipleCSIRequest` is disabled. When `EnableMultipleCSIRequest` is enabled (`'On'`), the UE is configured to process multiple channel state information (CSI) requests from cells. Enabling multiple CSI requests affects the length of the CSI request field in UE-specific formats 0 and 4.

Data Types: `char` | `string`

**`NTxAnts` — Number of UE transmission antennas**
1 (default) | 2 | 4 | optional

Number of UE transmission antennas, specified as 1, 2, or 4. The number of UE transmission antennas affects the length of the precoding information field in DCI format 4.

Data Types: `double`

**`PSSCHNSubchannels` — Number of sub-channels in V2X PSSCH pool**
1 (default) | integer scalar from 2 to 110 | optional

Number of sub-channels in V2X PSSCH pool, specified as an integer scalar from 1 to 110. It affects the length of RIV in format 5A

Data Types: `double`

Data Types: `struct`

**`bitsin` — Input bits**
vector

Input bits, specified as a column vector. `bitsin` is treated as the DCI information bit payload, that is, `bitsout == bitsin`. The length of `bitsin` must be one of the valid sizes for the format type and

number of resource blocks. For information on link bandwidth assignment, see “Specifying Number of Resource Blocks” on page 2-101. For information on valid sizes, see `lteDCIInfo`.

When `bitsin` is specified, the structure `dciin` does not require the `DCIFormat` field. If the `DCIFormat` field is not present, `lteDCI` attempts to decode the format from the length of the payload vector `bitsin`.

Data Types: `double`

**`istr` — Input structure**
structure

Input structure, specified as a structure that includes all the fields described in the structures `enb` and `dciin`.

Use of the `istr` input syntax is not recommended and will be removed in a future release. Instead, use one of the previous syntaxes that separates the parameters into different input structures.

## Output Arguments

**`dciout` — DCI message structure**
structure

DCI message structure, returned as a structure whose fields match the associated DCI format contents.

The field names associated with `dciout` depend on the DCI format field in `dciin`. By default, all values are set to zero. However, if any of the DCI fields are already present in the input `dciin`, their values are carried forward into `dciout`. The input field values appear in the associated bit positions in `bitsout`. Carrying the values forward allows for easy initialization of DCI field values, particularly the resource allocation type, which affects the fields used by the format. `dciout` also carries forward the `NDLRB` and `DCIFormat` fields supplied in `dciin`.

This table presents the fields associated with each DCI format as defined in TS 36.212 [2], Section 5.3.3.

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| `'Format0'` | `DCIFormat` | - | `'Format0'` |
| | `CIF` | 0 or 3 bits | Carrier indicator field |
| | `FreqHopping` | 1 bit | PUSCH frequency hopping flag |
| | `Allocation` | Varies | Resource block assignment/allocation |
| | `ModCoding` | 5 bits | Modulation, coding scheme, and redundancy version |
| | `NewData` | 1 bit | New data indicator |
| | `TPC` | 2 bits | PUSCH TPC command |
| | `CShiftDMRS` | 3 bits | Cyclic shift for DM RS |

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| | `TDDIndex` | 2 bits | For TDD config 0, this field is the Uplink Index.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | `CSIRequest` | 1, 2, or 3 bits | CSI request |
| | `SRSRequest` | 0 or 1 bit | SRS request. This field can only be present in DCI formats scheduling PUSCH which are mapped onto the UE specific search space given by the C-RNTI |
| | `AllocationType` | 1 bit | Resource allocation type, only present if $N_{RB}^{UL} \leq N_{RB}^{DL}$. |
| `'Format1'` | `DCIFormat` | - | `'Format1'` |
| | `CIF` | 0 or 3 bits | Carrier indicator field |
| | `AllocationType` | 1 bit | Resource allocation header: type 0, type 1. If downlink bandwidth is ≤10 PRBs there is no resource allocation header and resource allocation type 0 is assumed. |
| | `Allocation` | Varies | Resource block assignment/allocation |
| | `ModCoding` | 5 bits | Modulation and coding scheme |
| | `HARQNo` | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |
| | `NewData` | 1 bit | New data indicator |
| | `RV` | 2 bits | Redundancy version |
| | `TPCPUCCH` | 2 bits | PUCCH TPC command |
| | `TDDIndex` | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | `HARQACKResOffset` | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| `'Format1A'` | `DCIFormat` | - | `'Format1A'` |
| | `CIF` | 0 or 3 bits | Carrier indicator field |
| | `AllocationType` | 1 bit | VRB assignment flag: 0 (localized), 1 (distributed) |

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| | `Allocation` | Varies | Resource block assignment/allocation |
| | `ModCoding` | 5 bits | Modulation and coding scheme |
| | `HARQNo` | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |
| | `NewData` | 1 bit | New data indicator |
| | `RV` | 2 bits | Redundancy version |
| | `TPCPUCCH` | 2 bits | PUCCH TPC command |
| | `TDDIndex` | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | `SRSRequest` | 0 or 1 bit | SRS request. This field can only be present in DCI formats scheduling PUSCH which are mapped onto the UE specific search space given by the C-RNTI |
| | `HARQACKResOffset` | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| `'Format1B'` | `DCIFormat` | - | `'Format1B'` |
| | `CIF` | 0 or 3 bits | Carrier indicator field |
| | `AllocationType` | 1 bit | VRB assignment flag: 0 (localized), 1 (distributed) |
| | `Allocation` | Varies | Resource block assignment/allocation |
| | `ModCoding` | 5 bits | Modulation and coding scheme |
| | `HARQNo` | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |
| | `NewData` | 1 bit | New data indicator |
| | `RV` | 2 bits | Redundancy version |
| | `TPCPUCCH` | 2 bits | PUCCH TPC command |
| | `TDDIndex` | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| | TPMI | 2 bits for two antennas<br><br>4 bits for four antennas | PMI information |
| | PMI | 1 bit | PMI confirmation |
| | HARQACKResOffset | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| 'Format1C' | DCIFormat | - | 'Format1C' |
| | Allocation | Varies | Resource block assignment/allocation |
| | ModCoding | 5 bits | Modulation and coding scheme |
| 'Format1D' | DCIFormat | - | 'Format1D' |
| | CIF | 0 or 3 bits | Carrier indicator field |
| | AllocationType | 1 bit | VRB assignment flag: 0 (localized), 1 (distributed) |
| | Allocation | Varies | Resource block assignment/allocation |
| | ModCoding | 5 bits | Modulation and coding scheme |
| | HARQNo | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |
| | NewData | 1 bit | New data indicator |
| | RV | 2 bits | Redundancy version |
| | TPCPUCCH | 2 bits | PUCCH TPC command |
| | TDDIndex | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | TPMI | 2 bits for two antennas<br><br>4 bits for four antennas | Precoding TPMI information |
| | DlPowerOffset | 1 bit | Downlink power offset |
| | HARQACKResOffset | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| 'Format2' | DCIFormat | - | 'Format2' |
| | CIF | 0 or 3 bits | Carrier indicator field |

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| | `AllocationType` | 1 bit | Resource allocation header: type 0, type 1. If downlink bandwidth is ≤10 PRBs there is no resource allocation header and resource allocation type 0 is assumed. |
| | `Allocation` | Varies | Resource block assignment/allocation |
| | `TPCPUCCH` | 2 bits | PUCCH TPC command |
| | `TDDIndex` | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | `HARQNo` | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |
| | `SwapFlag` | 1 bit | Transport block to codeword swap flag |
| | `ModCoding1` | 5 bits | Modulation and coding scheme for transport block 1 |
| | `NewData1` | 1 bit | New data indicator for transport block 1 |
| | `RV1` | 2 bits | Redundancy version for transport block 1 |
| | `ModCoding2` | 5 bits | Modulation and coding scheme for transport block 2 |
| | `NewData2` | 1 bit | New data indicator for transport block 2 |
| | `RV2` | 2 bits | Redundancy version for transport block 2 |
| | `PrecodingInfo` | 3 bits for two antennas<br><br>6 bits for four antennas | Precoding information |
| | `HARQACKResOffset` | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| `'Format2A'` | `DCIFormat` | `-` | `'Format2A'` |
| | `CIF` | 0 or 3 bits | Carrier indicator field |
| | `AllocationType` | 1 bit | Resource allocation header: type 0, type 1. If downlink bandwidth is ≤10 PRBs there is no resource allocation header and resource allocation type 0 is assumed. |
| | `Allocation` | Varies | Resource block assignment/allocation |

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| | TPCPUCCH | 2 bits | PUCCH TPC command |
| | TDDIndex | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | HARQNo | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |
| | SwapFlag | 1 bit | Transport block to codeword swap flag |
| | ModCoding1 | 5 bits | Modulation and coding scheme for transport block 1 |
| | NewData1 | 1 bit | New data indicator for transport block 1 |
| | RV1 | 2 bits | Redundancy version for transport block 1 |
| | ModCoding2 | 5 bits | Modulation and coding scheme for transport block 2 |
| | NewData2 | 1 bit | New data indicator for transport block 2 |
| | RV2 | 2 bits | Redundancy version for transport block 2 |
| | PrecodingInfo | 0 bits for two antennas<br><br>2 bits for four antennas | Precoding information |
| | HARQACKResOffset | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| 'Format2B' | DCIFormat | - | 'Format2B' |
| | CIF | 0 or 3 bits | Carrier indicator field |
| | AllocationType | 1 bit | Resource allocation header: type 0, type 1. If downlink bandwidth is ≤10 PRBs there is no resource allocation header and resource allocation type 0 is assumed. |
| | Allocation | Varies | Resource block assignment/allocation |
| | TPCPUCCH | 2 bits | PUCCH TPC command |
| | TDDIndex | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| | HARQNo | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |
| | ScramblingId | 1 bit | Scrambling identity |
| | ModCoding1 | 5 bits | Modulation and coding scheme for transport block 1 |
| | NewData1 | 1 bit | New data indicator for transport block 1 |
| | RV1 | 2 bits | Redundancy version for transport block 1 |
| | ModCoding2 | 5 bits | Modulation and coding scheme for transport block 2 |
| | NewData2 | 1 bit | New data indicator for transport block 2 |
| | RV2 | 2 bits | Redundancy version for transport block 2 |
| | HARQACKResOffset | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| 'Format2C' | DCIFormat | - | 'Format2C' |
| | CIF | 0 or 3 bits | Carrier indicator field |
| | AllocationType | 1 bit | Resource allocation header: type 0, type 1. If downlink bandwidth is ≤10 PRBs there is no resource allocation header and resource allocation type 0 is assumed. |
| | Allocation | Varies | Resource block assignment/allocation |
| | TPCPUCCH | 2 bits | PUCCH TPC command |
| | TDDIndex | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | HARQNo | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |
| | TxIndication | 3 bits | Antenna ports, scrambling identity, and number of layers indicator |
| | SRSRequest | Varies | SRS request. Only present for TDD. |
| | ModCoding1 | 5 bits | Modulation and coding scheme for transport block 1 |
| | NewData1 | 1 bit | New data indicator for transport block 1 |

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| | RV1 | 2 bits | Redundancy version for transport block 1 |
| | ModCoding2 | 5 bits | Modulation and coding scheme for transport block 2 |
| | NewData2 | 1 bit | New data indicator for transport block 2 |
| | RV2 | 2 bits | Redundancy version for transport block 2 |
| | HARQACKResOffset | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| 'Format2D' | DCIFormat | - | 'Format2D' |
| | CIF | 0 or 3 bits | Carrier indicator field |
| | AllocationType | 1 bit | Resource allocation header: type 0, type 1. If downlink bandwidth is ≤10 PRBs there is no resource allocation header and resource allocation type 0 is assumed. |
| | Allocation | Varies | Resource block assignment/allocation |
| | TPCPUCCH | 2 bits | PUCCH TPC command |
| | TDDIndex | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | HARQNo | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |
| | TxIndication | 3 bits | Antenna ports, scrambling identity, and number of layers indicator |
| | SRSRequest | Varies | SRS request. Only present for TDD. |
| | ModCoding1 | 5 bits | Modulation and coding scheme for transport block 1 |
| | NewData1 | 1 bit | New data indicator for transport block 1 |
| | RV1 | 2 bits | Redundancy version for transport block 1 |
| | ModCoding2 | 5 bits | Modulation and coding scheme for transport block 2 |
| | NewData2 | 1 bit | New data indicator for transport block 2 |
| | RV2 | 2 bits | Redundancy version for transport block 2 |

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| | REMappingAndQCL | 2 bits | PDSCH RE Mapping and Quasi-Co-Location Indicator |
| | HARQACKResOffset | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| 'Format3' | DCIFormat | - | 'Format3' |
| | TPCCommands | Varies | TPC commands for PUCCH and PUSCH |
| 'Format3A' | DCIFormat | - | 'Format3A' |
| | TPCCommands | Varies | TPC commands for PUCCH and PUSCH |
| 'Format4' | DCIFormat | - | 'Format4' |
| | CIF | 0 or 3 bits | Carrier indicator field |
| | Allocation | Varies | Resource block assignment/allocation |
| | TPC | 2 bits | PUSCH TPC command |
| | CShiftDMRS | 3 bits | Cyclic shift for DM-RS |
| | TDDIndex | 2 bits | For TDD config 0, this field is Uplink Index.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | CSIReq | Varies | CSI request |
| | SRSRequest | 2 bits | SRS request |
| | AllocationType | 1 bit | Resource allocation header type 0 or type 1. |
| | ModCoding | 5 bits | Modulation, coding scheme, and redundancy version |
| | NewData | 1 bit | New data indicator |
| | ModCoding1 | 5 bits | Modulation and coding scheme for transport block 1 |
| | NewData1 | 1 bit | New data indicator for transport block 1 |
| | ModCoding2 | 5 bits | Modulation and coding scheme for transport block 2 |
| | NewData2 | 1 bit | New data indicator for transport block 2 |
| | PrecodingInfo | 3 bits for two antennas<br><br>6 bits for four antennas | Precoding information |
| 'Format5' | DCIFormat | - | 'Format5' |
| | PSCCHResource | 6 bits | Resource for PSCCH |

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| | `TPC` | 1 bit | TPC command for PSCCH and PSSCH |
| | `FreqHopping` | 1 bit | Frequency hopping flag |
| | `Allocation` | Varies | Resource block assignment and hopping resource allocation |
| | `TimeResourcePattern` | 7 bits | Time resource pattern |
| `'Format5A'` | `DCIFormat` | - | `'Format5A'` |
| | `CIF` | 3 bits | Carrier indicator |
| | `FirstSubchannelIdx` | $\left\lceil \log_2\left(N_{\text{subchannel}}^{\text{SL}}\right) \right\rceil$ | Lowest index of the subchannel allocation to the initial transmission |
| | `RIV` | from 0 to 13 bits, $\left\lceil \log_2 \left( \frac{N_{\text{subchannel}}^{\text{SL}} \times \left(N_{\text{subchannel}}^{\text{SL}} + 1\right)}{2} \right) \right\rceil$ | Resource indication value |
| | `TimeGap` | 4 bits | Time gap between initial transmission and retransmission |
| | `SLIndex` | 2 bits | SL SPS configuration index |

The `DCIFormat` field indicates the DCI format. All other fields are represented by an integer which is converted to a set of binary message bits for each individual field.

The `ModCoding` fields in the table correspond to the variable $I_{\text{MCS}}$ defined in TS 36.213 [3], Section 7.1.7, Table 7.1.7.1-1. This field expects to be assigned a decimal number. The call to `lteDCI` serializes `ModCoding` into a 5-bit field value. For example, the `ModCoding` field for 64QAM modulation ($Q_{\text{m}}$) and transport block index ($I_{\text{TBS}}$) 15 is assigned 17 (a decimal number).

The fields included in the `Allocation` structure vary based on the format type as outlined in these tables. All fields take a character vector of zeros and ones with the appropriate bit length.

| **Resource allocation type 0 on page 2-101** | | | |
|---|---|---|---|
| **DCI Formats** | **`Allocation` Fields** | **Size (bits)** | **Description** |
| `'Format1'`<br>`'Format2'`<br>`'Format2A'`<br>`'Format2B'` | `Bitmap` | Varies | Bitmap value in terms of RBG, specified as a character vector |

| Resource allocation type 1 on page 2-101 | | | |
|---|---|---|---|
| **DCI Formats** | **`Allocation` Fields** | **Size (bits)** | **Description** |
| `'Format1'`<br>`'Format2'`<br>`'Format2A'`<br>`'Format2B'` | `Bitmap` | Varies | Bitmap value in terms of RBG, specified as a character vector |
| | `RBSubset` | 2 bits | Selected resource blocks subset indicator |
| | `Shift` | 1 bit | Shift of the resource allocation span indicator |

| Resource allocation type 2 (localized) on page 2-102 | | | |
|---|---|---|---|
| **DCI Formats** | **`Allocation` Fields** | **Size (bits)** | **Description** |
| `'Format1A'`<br>`'Format1B'`<br>`'Format1C'`<br>`'Format1D'` | `RIV` | Varies | Resource indication value |

| Resource allocation type 2 (distributed) on page 2-102 | | | |
|---|---|---|---|
| **DCI Formats** | **`Allocation` Fields** | **Size (bits)** | **Description** |
| `'Format1A'`<br>`'Format1B'`<br>`'Format1C'`<br>`'Format1D'` | `RIV` | Varies | Resource indication value |
| | `Gap` | 1 bit | Gap value: 0 (gap1), 1 (gap2) |

| Uplink Nonhopping allocation on page 2-102 | | | |
|---|---|---|---|
| **DCI Formats** | **`Allocation` Fields** | **Size (bits)** | **Description** |
| `'Format0'`<br>`'Format5'` | `RIV` | Varies | Resource indication value |

| Uplink Hopping allocation on page 2-102 | | | |
|---|---|---|---|
| **DCI Formats** | **`Allocation` Fields** | **Size (bits)** | **Description** |
| `'Format0'`<br>`'Format5'` | `RIV` | Varies | Resource indication value |
| | `HoppingBits` | Varies | When the number of hopping bits is 1, `HoppingBits` value can be 0 or 1.<br>When the number of hopping bits is 2, `HoppingBits` value can be 00, 01, 10, or 11.<br>See TS 36.213 [3], Table 8.4-2. |

**`bitsout` — DCI message in bit payload form**
bit vector

DCI message in bit payload form, returned as a column vector. `bitsout` represents the set of message fields mapped to the information bit payload (including any zero-padding).

## More About

### Specifying Number of Resource Blocks

The number of resource blocks specifies the uplink and downlink bandwidth. The LTE Toolbox implementation assumes symmetric link bandwidth unless you specifically assign different values to `NULRB` and `NDLRB`. If the number of resource blocks is initialized in only one link direction, then the initialized number of resource blocks (`NULRB` or `NDLRB`) is used for both uplink and downlink. When this mapping is used, no warning is displayed. An error occurs if `NULRB` and `NDLRB` are both undefined.

## Algorithms

### Resource allocation type 0

In type 0 resource allocation, a bitmap represents a resource block group (RBG) allocated to a UE. `P` gives the RBG size, which can be deduced from TS 36.213 [3], Table 7.1.6.1-1 for a given system bandwidth. The number of bits in the `Bitmap` field is equal to $\lceil NDLRB/P \rceil$. Each bit in the `Bitmap` selects a small contiguous group whose size depends on the bandwidth (RBG: 1,...,4). The maximum resource block (RB) coverage of any type 0 allocation is the entire bandwidth, that is, a type 0 allocation with all the bits in bitmap set to `'1'` is equivalent to the entire bandwidth.

**Example** 50 RB bandwidth

The number of bits in `Bitmap` are 17. Each bit in the 17-bit bitmap selects a group of three RB (apart from the last group, which only contains two RB for this bandwidth). Each bit is associated with a group of RBs with the same color.


### Resource allocation type 1

In type 1 resource allocation, a bitmap indicates physical resource blocks inside a selected resource block group subset $p$, where $0 \le p < P$. The maximum resource block (RB) coverage of any type 1 allocation is a subset of entire bandwidth. A type 1 allocation, even with all the bits in the `Bitmap` set to `'1'`, does not span the entire bandwidth. Each bit in the bitmap selects a single RB from "islands" of small contiguous groups whose size (RBG) and separation depend on the total bandwidth. This grouping provides the provision of selecting a single RB without turning on any other RB.

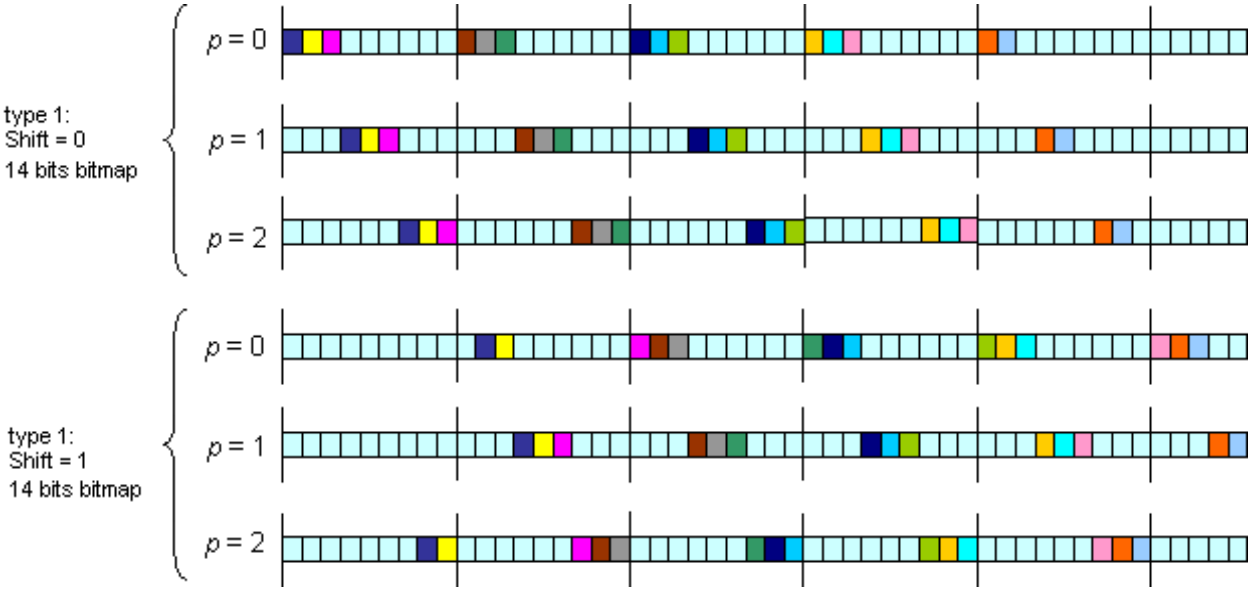
In type 1, the resource block assignment signaling is split into three field parts:

1. `RBSubset` — Represents the selected resource block group subset
2. `Shift` — Indicates whether to apply an offset when interpreting the bitmap
3. `Bitmap` — Contains the bitmap that indicates to the UE the specific physical resource block within the resource block group subset.

In comparison to type 0, the bitmap size for type 1 is always short by $\lceil \log_2(P) \rceil + 1$ bits, where $P$ is defined as in resource allocation type 0.

**Example** 50 RB bandwidth

The number of bits in `Bitmap` are 14 (3 bits short as compared to type 0, due to `RBSubset` and `Shift` parameters). Each bit in the 14-bit bitmap selects an individual RB inside a selected subset. The figure shows all the bits in `Bitmap` set to `'1'` for different subsets and offset values.




### Resource allocation type 2

In type 2 resource allocation, physical resource blocks are not directly allocated. Instead, virtual resource blocks are allocated, which are then mapped onto physical resource blocks. Type 2 allocation supports both localized and distributed virtual resource block allocation, differentiated by one-bit flag. The starting point of the virtual resource block and the length in terms of the contiguously allocated virtual resource blocks can be derived from Resource Indication Value (RIV) signaled within the DCI.

**Example** 50 RB bandwidth

A UE is allocated a bandwidth of 25 resource blocks ($L_{CRBs}$=25), starting from resource block 10 ($RB_{start}$=10) in the frequency domain. To calculate the RIV value, refer to the formula given in TS 36.213 [3], Section 7.1.6.3, which yields RIV = 1210. Using this RIV, which is signaled in the DCI, the UE can unambiguously derive the starting resource block and the number of allocated resource blocks from RIV again.

### Uplink Nonhopping Resource Allocation

For uplink nonhopping resource allocation, the rules for type 2 localized resource allocation apply for deriving the resource allocation from the RIV value.

### Uplink Hopping Resource Allocation

When `FreqHopping` is set to 1, uplink hopping resource allocation is available. For uplink hopping resource allocation, two types of hopping are used: Type 1 PUSCH Hopping and Type 2 PUSCH Hopping. Do not confuse these types with downlink resource allocation types 1 and 2 described earlier. Type 1 PUSCH Hopping is calculated using the RIV value and parameters signaled by higher

layers. Type 2 PUSCH Hopping is calculated using a predefined pattern, which is a function of the subframe and frame number, as defined in TS 36.211 [1], Section 5.3.4. The fundamental set of resource blocks used as part of the hopping is calculated via the rules for type 2 localized resource allocation from the RIV value, except either 1 or 2 (depending on system bandwidth) hopping bits have been deducted from the resource allocation bitmap. These hopping bits specify whether Type 1 or Type 2 PUSCH Hopping is used, and for the case of 2 bits, variations of the position of the Type 1 hopping in the frequency domain. The definition of the hopping bits can be found in TS 36.213 [3], Table 8.4-2. Bandwidth dependency for the number of hopping bits allocated follows the following rule:

- If the system BW is `NULRB<=49`, the number of hopping bits is 1, and `HoppingBits` can be 0 or 1.
- If the system BW is `NULRB>49`, the number of hopping bits is 2, and `HoppingBits` can be 00, 01, 10, or 11.

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[3] 3GPP TS 36.213. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteDCIEncode` | `lteDCIDecode` | `lteDCIResourceAllocation` | `lteDCIInfo` | `lteSCI`

# lteDCIDecode

Downlink control information decoding

## Syntax

```
[dcibits,crc_rnti] = lteDCIDecode(dcilen,softbits)
[dcibits,crc_rnti] = lteDCIDecode(enb,softbits)
[dcibits,crc_rnti] = lteDCIDecode(enb,chs,softbits)
```

## Description

`[dcibits,crc_rnti] = lteDCIDecode(dcilen,softbits)` recovers a downlink control information (DCI) message, given the DCI vector length, `dcilen`, and a `softbits` input vector. For more information, see “DCI Message Decoding” on page 2-109.

`[dcibits,crc_rnti] = lteDCIDecode(enb,softbits)` uses the cell-wide configuration structure, `enb`. With this syntax, the DCI message length is deduced from `enb.DCIFormat` and cell-wide settings in `enb`.

`[dcibits,crc_rnti] = lteDCIDecode(enb,chs,softbits)` uses the UE-specific channel configuration structure, `chs`. With this syntax, the DCI message length is deduced from `chs.DCIFormat`, the cell-wide configuration in `enb`, and the UE-specific channel configuration in `chs`.

## Examples

### DCI Decoding

Perform DCI decoding of a sample codeword and return the decoded vector, `decodedDCIbits`, of the length defined for the DCI Format 1 message.

```
enb.NDLRB = 50;
enb.CellRefP = 1;
enb.DuplexMode = 'FDD';

dciInfo = lteDCIInfo(enb);
dcilen = dciInfo.Format1
```

```
dcilen = uint64
    31
```

```
ue.PDCCHFormat = 1;
ue.RNTI = 10;
ue.NDLRB = 50;

dciBits = zeros(dcilen,1);
cw = lteDCIEncode(ue,dciBits);

[decodedDCIbits,crcRNTI] = lteDCIDecode(dcilen,cw);
```

```
decodedDCIbitslen = size(decodedDCIbits)

decodedDCIbitslen = 1×2

    31     1


crcRNTI

crcRNTI = uint32
    10
```

The `decodedDCIbits` length matches the value of `dcilen`. The `crcRNTI` output has a value of 10, corresponding to the RNTI values used in CRC masking.

### DCI Decoding with ENB Structure

Perform DCI decoding of a sample codeword and return the decoded vector, `decodedDCIbits`, of the length defined for the DCI Format 1 message. The `lteDCIDecode` function uses fields in `enb` to determine DCI length.

```
enb.NDLRB = 25;
enb.CellRefP = 1;
enb.DuplexMode = 'FDD';

dciInfo = lteDCIInfo(enb);
dcilen = dciInfo.Format1

dcilen = uint64
    27


ue.PDCCHFormat = 1;
ue.RNTI = 7;
ue.NDLRB = 25;

dciBits = zeros(dcilen,1);
cw = lteDCIEncode(ue,dciBits);
```

Define the `enb` configuration structure for recovery of the DCI message and RNTI. Perform DCI decoding using `enb`.

```
enb.DCIFormat = 'Format1';

[decodedDCIbits,crcRNTI] = lteDCIDecode(enb,cw);

decodedDCIbitslen = size(decodedDCIbits)

decodedDCIbitslen = 1×2

    27     1


crcRNTI
```

```
crcRNTI = uint32
    7
```

The `decodedDCIbits` length matches the value of `dcilen`. The `crcRNTI` value recovered corresponds to and matches `ue.RNTI`, which is the RNTI value used in the CRC masking.

### DCI Decoding Using Channel Structure

Perform DCI decoding of a sample codeword and return the decoded vector, `decodedDCIbits`, of the length defined for the DCI Format 2A message.

```
enb.NDLRB = 25;
enb.CellRefP = 1;
enb.DuplexMode = 'FDD';

dciInfo = lteDCIInfo(enb);
dcilen = dciInfo.Format2A

dcilen = uint64
    36


ue.PDCCHFormat = 2;
ue.RNTI = 5;
ue.NDLRB = 25;

dciBits = zeros(dcilen,1);
cw = lteDCIEncode(ue,dciBits);
```

Define the ue-specific configuration structure, `chs`, for recovery of the DCI message and RNTI. Perform DCI decoding using `enb` and `chs`.

```
chs.DCIFormat = 'Format2A';
chs.ControlChannelType  = 'PDCCH';
chs.EnableCarrierIndication  = 'Off';

[decodedDCIbits,crcRNTI] = lteDCIDecode(enb,chs,cw);

decodedDCIbitslen = size(decodedDCIbits)

decodedDCIbitslen = 1×2

    36     1


crcRNTI

crcRNTI = uint32
    5
```

The `decodedDCIbits` length matches the value of `dcilen`. The `crcRNTI` value recovered corresponds to and matches `ue.RNTI`, which is the RNTI value used in the CRC masking.

## Input Arguments

### `dcilen` — Length of recovered DCI message vector

integer

Length of recovered DCI message vector, specified as a positive integer.

Data Types: `double`

### `softbits` — Floating-point soft bits

vector

Floating-point soft bits, specified as a column vector.

Data Types: `double` | `int8`

### `enb` — eNodeB cell-wide settings

structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NDLRB`** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **`NULRB`** | Required | Scalar integer from 6 to 110 | Number of uplink resource blocks. ($N_{RB}^{UL}$) |
| **`DCIFormat`** | Required (see syntax descriptions for applicability) | `'Format0'`, `'Format1'`, `'Format1A'`, `'Format1B'`, `'Format1C'`, `'Format1D'`, `'Format2'`, `'Format2A'`, `'Format2B'`, `'Format2C'`, `'Format2D'`, `'Format3'`, `'Format3A'`, `'Format4'`, `'Format5'`, `'Format5A'` | Downlink control information (DCI) format |
| **`CellRefP`** | Optional | 1 (default), 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **`DuplexMode`** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |

### `chs` — User-equipment-related channel configuration

structure

User-equipment-related (UE-related) channel configuration, specified as a structure containing these UE-specific fields.

**Note** All fields in `chs` are optional. The presence of these optional fields depends on:

- Whether the transmission of DCI message is in a PDCCH using common search space mapping or in an EPDCCH.
- The release-specific features configured at the destination UE.

These additional UE-specific bit fields are off by default.

**`DCIFormat` — DCI format name**
`'Format0'` | `'Format1'` | `'Format1A'` | `'Format1B'` | `'Format1C'` | `'Format1D'` | `'Format2'` | `'Format2A'` | `'Format2B'` | `'Format2C'` | `'Format2D'` | `'Format3'` | `'Format3A'` | `'Format4'` | `'Format5''Format5A'`

DCI format name, specified as a character vector or string scalar. For string scalar, use double quotes. See syntax descriptions for applicability.

Data Types: `char` | `string`

**`ChannelControlType` — Physical control channel type**
`'PDCCH'` (default) | `'EPDCCH'` | optional

Physical control channel type used to carry DCI formats, specified as `'PDCCH'` or `'EPDCCH'`. The setting for `ChannelControlType` affects the presence of the HARQ-ACK resource offset field and message padding.

Data Types: `char` | `string`

**`SearchSpace` — Search space mapping**
`'UESpecific'` (default) | `'Common'` | optional

Search space mapping for DCI formats 0/1A/1C, specified as `'UESpecific'` or `'Common'`. This field is only applicable for PDCCH. None of the additional fields can be present when formats 0 or 1A are mapped into the PDCCH common search space.

Data Types: `char` | `string`

**`EnableCarrierIndication` — Option to enable carrier indication**
`'Off'` (default) | `'On'` | optional

Option to enable carrier indication field (CIF) in the UE configuration, specified as `'Off'` or `'On'`. By default, `EnableCarrierIndication` is disabled. When `EnableCarrierIndication` is enabled (`'On'`), the CIF is present in the UE-specific configuration.

Data Types: `char` | `string`

**`EnableSRSRequest` — Option to enable SRS request**
`'Off'` (default) | `'On'` | optional

Option to enable SRS request in the UE configuration, specified as `'Off'` or `'On'`. By default, `EnableSRSRequest` is disabled. When `EnableSRSRequest` is enabled (`'On'`), the SRS request field is present in UE-specific formats 0/1A for FDD or TDD and formats 2B/2C/2D for TDD.

Data Types: `char` | `string`

**`EnableMultipleCSIRequest` — Option to enable multiple CSI requests**
`'Off'` (default) | `'On'` | optional

Option to enable multiple CSI requests in the UE configuration, specified as `'Off'` or `'On'`. By default, `EnableMultipleCSIRequest` is disabled. When `EnableMultipleCSIRequest` is enabled

('On'), the UE is configured to process multiple channel state information (CSI) requests from cells. Enabling multiple CSI requests affects the length of the CSI request field in UE-specific formats 0 and 4.

Data Types: char | string

**NTxAnts — Number of UE transmission antennas**
1 (default) | 2 | 4 | optional

Number of UE transmission antennas, specified as 1, 2, or 4. The number of UE transmission antennas affects the length of the precoding information field in DCI format 4.

Data Types: double

**PSSCHNSubchannels — Number of sub-channels in V2X PSSCH pool**
1 (default) | integer scalar from 2 to 110 | optional

Number of sub-channels in V2X PSSCH pool, specified as an integer scalar from 1 to 110. It affects the length of RIV in format 5A

Data Types: double

Data Types: struct

## Output Arguments

**dcibits — Recovered DCI message bit vector**
vector

Recovered DCI message bit vector, returned as a column vector. The length of dcibits is defined though structure enb in terms of the DCI message format and the bandwidth.

Data Types: int8

**crc_rnti — 16-bit integer result of the CRC decoder**
vector

16-bit integer result of the CRC decoder, returned as a column vector. crc_rnti is equivalent to the RNTI value that would need to mask (XOR) the CRC for no CRC error.

Data Types: uint32

## More About

### DCI Message Decoding

Downlink control information (DCI) message decoding performs the inverse DCI processing operation as specified in TS 36.212 [1], Section 5.3.3. Specifically, lteDCIDecode performs rate recovery, and Viterbi and CRC decoding to recover the DCI message bit vector (dcibits) from an input vector of received soft bits previously coded by the DCI processing. lteDCIDecode also returns the 16-bit integer result of the CRC decoder, crc_rnti, which is equivalent to the RNTI value that would need to mask (XOR) the CRC for no CRC error. Using the RNTI, recovered with no CRC errors, enables the system to match the recovered DCI message with a specific ue.

The length of the DCI information payload to be recovered can be specified

- Directly by `dcilen`
- Determined using the fields in `enb` that specify the DCI message format (`DCIFormat`) and bandwidth (either `NDLRB` or `NULRB`).

For information on link bandwidth assignment, see “Specifying Number of Resource Blocks” on page 2-110.

#### Specifying Number of Resource Blocks

The number of resource blocks specifies the uplink and downlink bandwidth. The LTE Toolbox implementation assumes symmetric link bandwidth unless you specifically assign different values to `NULRB` and `NDLRB`. If the number of resource blocks is initialized in only one link direction, then the initialized number of resource blocks (`NULRB` or `NDLRB`) is used for both uplink and downlink. When this mapping is used, no warning is displayed. An error occurs if `NULRB` and `NDLRB` are both undefined.

## Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteDCIEncode` | `lteDCI` | `lteDCIResourceAllocation` | `lteDCIInfo` | `ltePDCCHDecode`

# lteDCIInfo

Downlink control information message information

## Syntax

```
info = lteDCIInfo(enb)
info = lteDCIInfo(enb,chs)
```

## Description

`info = lteDCIInfo(enb)` returns a structure indicating the minimum possible payload sizes, including zero-padding bits when necessary, for all downlink control information (DCI) message formats, given the cell-wide configuration in `enb`. This syntax uses default values for the UE-specific channel configuration.

To access the individual bit field sizes for each separate format, use the related function `lteDCI`.

TS 36.212 [1], Section 5.3.3.1, along with the referenced procedures of TS 36.213 [2], specify the rules defining the relationship between the bit field sizes and message padding per format, and the cell-wide and UE-specific parameters.

For information on link bandwidth assignment, see “Specifying Number of Resource Blocks” on page 2-117.

`info = lteDCIInfo(enb,chs)` permits formats to be extended with additional bit fields on a per-UE basis, using the UE-specific channel configuration, `chs`.

## Examples

### Get DCI Message Information

Find the minimum payload sizes of all DCI message formats for NDLRB = 50 (10 MHz symmetric link bandwidth), FDD duplexing, and PDCCH transmission.

```
enb = struct('NDLRB',50,'DuplexMode','FDD','CellRefP',1);
lteDCIInfo(enb)
```

```
ans = struct with fields:
     Format0: 27
     Format1: 31
    Format1A: 27
    Format1B: 28
    Format1C: 13
    Format1D: 28
     Format2: 43
    Format2A: 41
    Format2B: 41
    Format2C: 42
    Format2D: 45
     Format3: 27
```

```
    Format3A: 27
     Format4: 36
     Format5: 27
    Format5A: 27
```

### Get DCI Message Using UE-Specific Configuration

Using the UE-specific channel structure, `chs`, extend the DCI formats to include optional fields dependent on the target UE protocol configuration.

Show the minimum DCI message size per format for `NDLRB` = 50 (10 MHz symmetric link bandwidth), FDD duplexing, and PDCCH transmission.

```
enb = struct('NDLRB',50,'DuplexMode','FDD','CellRefP',1);
dciout = lteDCIInfo(enb)
```

```
dciout = struct with fields:
     Format0: 27
     Format1: 31
    Format1A: 27
    Format1B: 28
    Format1C: 13
    Format1D: 28
     Format2: 43
    Format2A: 41
    Format2B: 41
    Format2C: 42
    Format2D: 45
     Format3: 27
    Format3A: 27
     Format4: 36
     Format5: 27
    Format5A: 27
```

Default settings for the UE-specific channel structure, `chs`, are:

```
chs.ControlChannelType = 'PDCCH';
chs.SearchSpace = 'UESpecific';
chs.EnableCarrierIndication = 'Off';
chs.EnableMultipleCSIRequest = 'Off';
chs.EnableSRSRequest = 'Off';
chs.NTxAnts = 1;
```

Enable carrier indication, and show the sizes per format when the DCI message is configured to include the UE-specific 3 bit carrier indicator field (CIF).

```
chs.EnableCarrierIndication = 'On';

dciout = lteDCIInfo(enb,chs)
```

```
dciout = struct with fields:
     Format0: 29
     Format1: 34
    Format1A: 29
```

```
    Format1B: 31
    Format1C: 13
    Format1D: 31
     Format2: 46
    Format2A: 43
    Format2B: 43
    Format2C: 45
    Format2D: 47
     Format3: 27
    Format3A: 27
     Format4: 39
     Format5: 29
    Format5A: 29
```

The sizes have not changed for formats 1C/3/3A, because the CIF does not apply to them. Also, because of the padding rules, the original lengths for some of the formats increased by less than 3 bits. These lengths are for formats mapped into the UE-specific search space, not formats 3/3A.

Change the UE-specific parameter to map the CIF into the PDCCH common search space.

```
chs.SearchSpace = 'Common';
dciout = lteDCIInfo(enb,chs)
```

```
dciout = struct with fields:
     Format0: 27
     Format1: 34
    Format1A: 27
    Format1B: 31
    Format1C: 13
    Format1D: 31
     Format2: 46
    Format2A: 43
    Format2B: 43
    Format2C: 45
    Format2D: 47
     Format3: 27
    Format3A: 27
     Format4: 39
     Format5: 27
    Format5A: 27
```

When the DCI message is configured for PDCCH-common search space, the format 0/1A length returns to its original size.

As specified in TS 36.212, with regard to search space, these points apply:

- Only formats 0/1A/1C can be mapped into either the PDCCH common or UE-specific search spaces.
- Formats 3/3A can be mapped into the common search space only.
- All other formats are mapped into UE-specific spaces only.
- There is no common search space for the EPDCCH.

## Input Arguments

### enb — DCI message format and bandwidth

structure

DCI message format and bandwidth, specified as a structure that can contain the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NDLRB** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$)<br><br>For information on link bandwidth assignment, see “Specifying Number of Resource Blocks” on page 2-117. |
| **NULRB** | Required | Scalar integer from 6 to 110 | Number of uplink resource blocks. ($N_{RB}^{UL}$)<br><br>For information on link bandwidth assignment, see “Specifying Number of Resource Blocks” on page 2-117. |
| **CellRefP** | Optional | 1 (default), 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **DuplexMode** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |

### chs — User Equipment (UE) related channel configuration

structure

User-equipment-related (UE-related) channel configuration, specified as a structure containing these UE-specific fields.

**Note** All fields in `chs` are optional. The presence of these optional fields depends on:

- Whether the transmission of DCI message is in a PDCCH using common search space mapping or in an EDPCCH.
- The release-specific features configured at the destination UE.

These additional UE-specific bit fields are off by default.

### EnableCarrierIndication — Option to enable carrier indication

`'Off'` (default) | `'On'` | optional

Option to enable carrier indication field (CIF) in the UE configuration, specified as `'Off'` or `'On'`. By default, `EnableCarrierIndication` is disabled. When `EnableCarrierIndication` is enabled (`'On'`), the CIF is present in the UE-specific configuration.

Data Types: `char` | `string`

**`ChannelControlType` — Physical control channel type**
`'PDCCH'` (default) | `'EPDCCH'` | optional

Physical control channel type used to carry DCI formats, specified as `'PDCCH'` or `'EPDCCH'`. The setting for `ChannelControlType` affects the presence of the HARQ-ACK resource offset field and message padding.

Data Types: `char` | `string`

**`SearchSpace` — Search space mapping**
`'UESpecific'` (default) | `'Common'` | optional

Search space mapping for DCI formats 0/1A/1C, specified as `'UESpecific'` or `'Common'`. This field is only applicable for PDCCH. None of the additional fields can be present when formats 0 or 1A are mapped into the PDCCH common search space.

Data Types: `char` | `string`

**`EnableSRSRequest` — Option to enable SRS request**
`'Off'` (default) | `'On'` | optional

Option to enable SRS request in the UE configuration, specified as `'Off'` or `'On'`. By default, `EnableSRSRequest` is disabled. When `EnableSRSRequest` is enabled (`'On'`), the SRS request field is present in UE-specific formats 0/1A for FDD or TDD and formats 2B/2C/2D for TDD.

Data Types: `char` | `string`

**`EnableMultipleCSIRequest` — Option to enable multiple CSI requests**
`'Off'` (default) | `'On'` | optional

Option to enable multiple CSI requests in the UE configuration, specified as `'Off'` or `'On'`. By default, `EnableMultipleCSIRequest` is disabled. When `EnableMultipleCSIRequest` is enabled (`'On'`), the UE is configured to process multiple channel state information (CSI) requests from cells. Enabling multiple CSI requests affects the length of the CSI request field in UE-specific formats 0 and 4.

Data Types: `char` | `string`

**`NTxAnts` — Number of UE transmission antennas**
1 (default) | 2 | 4 | optional

Number of UE transmission antennas, specified as 1, 2, or 4. The number of UE transmission antennas affects the length of the precoding information field in DCI format 4.

Data Types: `double`

**`PSSCHNSubchannels` — Number of sub-channels in V2X PSSCH pool**
1 (default) | integer scalar from 2 to 110 | optional

Number of sub-channels in V2X PSSCH pool, specified as an integer scalar from 1 to 110. It affects the length of RIV in format 5A

Data Types: `double`

Data Types: `struct`

## Output Arguments

### `info` — Payload sizes for all DCI message formats

structure

Payload sizes for all DCI message formats, returned as a structure with the following parameter fields.

| Parameter Field | Description | Values | Data Type |
|---|---|---|---|
| **`Format0`** | `Format0` payload size. `Format0` is the DCI format used for the scheduling of PUSCH. | Integer | `uint64` |
| **`Format1`** | `Format1` payload size. `Format1` is the DCI format used for the scheduling of one PDSCH codeword. | Integer | `uint64` |
| **`Format1A`** | `Format1A` payload size. `Format1A` is the DCI format used for the compact scheduling of one PDSCH codeword and random access procedure. | Integer | `uint64` |
| **`Format1B`** | `Format1B` payload size. `Format1B` is the DCI format used for the compact scheduling of one PDSCH codeword with precoding information. | Integer | `uint64` |
| **`Format1C`** | `Format1C` payload size. `Format1C` is the DCI format used for very compact scheduling of one PDSCH codeword. | Integer | `uint64` |
| **`Format1D`** | `Format1D` payload size. `Format1D` is the DCI format used for the compact scheduling of one PDSCH codeword with precoding and power offset information. | Integer | `uint64` |
| **`Format2`** | `Format2` payload size. `Format2` is the DCI format used for the scheduling of two PDSCH codewords with precoding information for closed-loop spatial multiplexing. | Integer | `uint64` |
| **`Format2A`** | `Format2A` payload size. `Format2A` is the DCI format used for the scheduling of two PDSCH codewords with precoding information for open-loop spatial multiplexing. | Integer | `uint64` |
| **`Format2B`** | `Format2B` payload size. `Format2B` is the DCI format used for the scheduling of dual-layer transmission, for antenna ports 7 and 8. | Integer | `uint64` |
| **`Format2C`** | `Format2C` payload size. `Format2C` is the DCI format used for the scheduling of up to eight-layer transmission, for antenna ports 7–14, using TM9. | Integer | `uint64` |
| **`Format2D`** | `Format2D` payload size. `Format2D` is the DCI format used for the scheduling of up to eight-layer transmission, for antenna ports 7–14, using TM10. | Integer | `uint64` |

| Parameter Field | Description | Values | Data Type |
|---|---|---|---|
| **`Format3`** | `Format3` payload size. `Format3` is the DCI format used for the transmission of transmit power control (TPC) commands for PUCCH and PUSCH with **2-bit** power adjustments. | Integer | `uint64` |
| **`Format3A`** | `Format3A` payload size. `Format3A` is the DCI format used for the transmission of transmit power control (TPC) commands for PUCCH and PUSCH with **1-bit** power adjustments. | Integer | `uint64` |
| **`Format4`** | `Format4` payload size. `Format4` is the DCI format used for the scheduling of PUSCH with multi-antenna port transmission mode. | Integer | `uint64` |
| **`Format5`** | `Format5` payload size. `Format5` is the DCI format used for the scheduling of PSCCH, and also contains several SCI format 0 fields used for the scheduling of PSSCH. | Integer | `uint64` |
| `Format5A` | `Format5A` payload size. `Format5A` is the DCI format used for the scheduling of V2X PSSCH. | Integer | `uint64` |

According to the rules defined in TS 36.212 [1], Section 5.3.3, the payload size of DCI `Format0` and `Format1A` should always be the same and either format will be appended with padding bits, if necessary, to fulfill this condition.

None of the DCI format payload sizes should equal the ambiguous sizes defined in TS 36.212 [1], Table 5.3.3.1.2-1. If necessary, padding bits are added to the DCI format payload. When transmitting DCI messages using PDCCH, the ambiguous format payload sizes are 12, 14, 16, 20, 24, 26, 32, 40, 44, and 56.

## More About

### Specifying Number of Resource Blocks

The number of resource blocks specifies the uplink and downlink bandwidth. The LTE Toolbox implementation assumes symmetric link bandwidth unless you specifically assign different values to `NULRB` and `NDLRB`. If the number of resource blocks is initialized in only one link direction, then the initialized number of resource blocks (`NULRB` or `NDLRB`) is used for both uplink and downlink. When this mapping is used, no warning is displayed. An error occurs if `NULRB` and `NDLRB` are both undefined.

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.213. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteDCI` | `lteDCIEncode` | `lteDCIDecode` | `lteDCIResourceAllocation` | `lteSCIInfo`

# lteDCIEncode

Downlink control information encoding

## Syntax

```
cw = lteDCIEncode(ue,dcibits)
cw = lteDCIEncode(ue,dcibits,outlen)
```

## Description

`cw = lteDCIEncode(ue,dcibits)` returns the vector resulting from downlink control information (DCI) processing of the input bit vector, `dcibits`, given the field settings in the structure, `ue`.

As described in TS 36.212 [1], Section 5.3.3, DCI processing involves CRC attachment with `ue.RNTI` masking of the CRC, convolutional coding, and rate matching to the capacity of the PDCCH format.

`cw = lteDCIEncode(ue,dcibits,outlen)` rate matches the output to `outlen`. For this syntax, `ue.PDCCHFormat` is ignored if present. The ability to request arbitrary output length makes this syntax useful for golden reference comparisons. Use this syntax for DCI encoding of PDCCH or EPDCCH transmissions.

## Examples

### Encode DCI with Zero RNTI

Perform DCI processing on an all-zero input. This processing results in an all-zero output when you set `RNTI` to 0.

Generate a `dcibits` input vector with zeros for a DCI format 1 message. `enb` is defined with 50 downlink RBs, 1 cell-specific reference signal antenna port, and FDD duplex mode.

```
enb = struct('NDLRB',50,'CellRefP',1,'DuplexMode','FDD');
dciInfo = lteDCIInfo(enb);
dcibits = zeros(dciInfo.Format1,1);
```

Define a `ue` parameter structure with PDCCH format 1 and RNTI set to 0.

```
ue = struct('PDCCHFormat',1,'RNTI',0);
```

Encode the DCI bits.

```
cw = lteDCIEncode(ue,dcibits);
cw(1:5)
```

```
ans = 5x1 int8 column vector

   0
   0
   0
   0
```

```
   0
```

For PDCCH format 1, the output vector length is 144. For this example, the output is an all-zero vector because the DCI bits were 0 and `RNTI` was set to 0.

### Encode DCI with Unity RNTI

Perform DCI processing on an all-zero input with `RNTI` set to 1. This processing results in a nonzero output when you set `RNTI` to 1.

Generate a `dcibits` input vector with zeros for a DCI format 1 message. `enb` is defined with 50 downlink RBs, 1 cell-specific reference signal antenna port, and FDD duplex mode.

```
enb = struct('NDLRB',50,'CellRefP',1,'DuplexMode','FDD');
dciInfo = lteDCIInfo(enb);
dcibits = zeros(dciInfo.Format1,1);
```

Define a `ue` parameter structure with PDCCH format 1 and RNTI set to 1.

```
ue = struct('PDCCHFormat',1,'RNTI',1);
```

Encode the DCI bits.

```
cw = lteDCIEncode(ue,dcibits);
cw(1:10)
```

```
ans = 10x1 int8 column vector

   0
   0
   0
   0
   0
   0
   0
   1
   0
   0
```

For PDCCH format 1, the output vector length is 144. For this example, with `RNTI` set to 1, the output vector is not all-zeros.

### Encode DCI Rate Matching Output Length

Perform DCI processing on an all-zero input. Set the output length to 100 bits.

Define `enb` with 50 downlink RBs, 1 cell-specific reference signal antenna port, and FDD duplex mode. Use `lteDCIInfo` to determine DCI message lengths for the defined configuration. Generate a `dcibits` input vector with zeros for a format 1 DCI message.

```
enb = struct('NDLRB',50,'CellRefP',1,'DuplexMode','FDD');
dciInfo = lteDCIInfo(enb);
dcibits = zeros(dciInfo.Format1,1);
```

Define a ue parameter structure with PDCCH format 1 and RNTI set to 0.

```
ue = struct('PDCCHFormat',1,'RNTI',0);
```

Encode the DCI bits.

```
cw1 = lteDCIEncode(ue,dcibits);
size(cw1)
```

```
ans = 1×2

   144     1
```

Encode the DCI bits again, setting the output length to 100 bits.

```
cw2 = lteDCIEncode(ue,dcibits,100);
size(cw2)
```

```
ans = 1×2

   100     1
```

The output vector length for cw2 is 100, rather than the encoded PDCCH format 1 length of 144 bits in cw1, as expected for the configuration.

### Encode DCI Message on EPDCCH

Use the DCI encoding function, `lteDCIEncode`, to code a DCI for transmission on EPDCCH. The required size is output by the `lteEPDCCHIndices` function and defined by `info.EPDCCHG`.

Specify the cell-wide settings in parameter structure enb.

```
enb.NDLRB = 6;
enb.NSubframe = 0;
enb.NCellID = 0;
enb.CellRefP = 1;
enb.CyclicPrefix = 'Normal';
enb.DuplexMode = 'FDD';
enb.NFrame = 0;
enb.CSIRSPeriod = 'Off';
enb.ZeroPowerCSIRSPeriod = 'Off';
```

Specify the channel transmission configuration in parameter structure chs.

```
chs.ControlChannelType = 'EPDCCH';
chs.SearchSpace = 'UESpecific';
chs.EnableCarrierIndication = 'Off';
chs.EnableMultipleCSIRequest = 'Off';
chs.EnableSRSRequest = 'Off';
chs.NTxAnts = 1;
```

```
chs.EPDCCHECCE = [2 3];
chs.EPDCCHType = 'Localized';
chs.EPDCCHPRBSet = 4:5;
chs.EPDCCHStart = 2;
chs.EPDCCHNID = 0;
chs.PDCCHFormat = 1;
chs.RNTI = 1;
dciInfo = lteDCIInfo(enb,chs);
dciin = zeros(dciInfo.Format1A,1);
```

Determine the EPDCCH data bit capacity, output by `lteEPDCCHIndices` in `info.EPDCCHG`.

```
[ind,info] = lteEPDCCHIndices(enb,chs);
info
```

```
info = struct with fields:
         EPDCCHG: 114
        EPDCCHGd: 57
         nEPDCCH: 114
           NECCE: 8
     NECCEPerPRB: 4
    NEREGPerECCE: 4
     EPDCCHPorts: 4
```

Encode the DCI bits.

```
cw1 = lteDCIEncode(chs,dciin);
size(cw1)
```

```
ans = 1×2

   144     1
```

Encode the DCI bits again, setting the output length to `info.EPDCCHG` bits.

```
cw2 = lteDCIEncode(chs,dciin,info.EPDCCHG);
size(cw2)
```

```
ans = 1×2

   114     1
```

The output vector length for `cw2` is 114, rather than the encoded format 1A length of 144 bits in `cw1`, as expected for the configuration.

## Input Arguments

### ue — Parameter structure for DCI processing

structure

Parameter structure for DCI processing, specified as a structure that must have these fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`PDCCHFormat`** | Required | 0, 1, 2, 3 | PDCCH format |
| **`RNTI`** | Required | 0 (default), scalar integer | Radio network temporary identifier (RNTI) value (16 bits) |

**`dcibits` — DCI message bit vector**
vector

DCI message bit vector, specified as a column vector. `dcibits` are the DCI processing input bits to be transmitted on a single PDCCH.

Data Types: `double` | `int8`

**`outlen` — Output vector length**
optional | nonnegative scalar integer

Output vector length, specified as a nonnegative scalar integer.

## Output Arguments

**`cw` — Output vector**
vector

Output vector resulting from DCI processing, returned as a column vector. `cw` is the result of DCI processing the input vector, `dcibits`. Depending on the function syntax used, the length of `cw` is either:

- $72*2^{\texttt{ue.PDCCUFormat}}$ elements, where $2^{\texttt{ue.PDCCUFormat}}$ represents the number of control channel elements (CCE) and one CCE is 72 bits.
- Rate matched to `outlen`.

Data Types: `int8`

# Version History
**Introduced in R2013b**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also
`lteDCI` | `lteDCIDecode` | `lteDCIResourceAllocation` | `lteDCIInfo` | `ltePDCCH` | `lteEPDCCH`

# lteDCIResourceAllocation

DCI message physical resource blocks allocation

## Syntax

```
[prbset, nrbg, rbgsize] = lteDCIResourceAllocation(enbue,dcistr)
[prbset, nrbg, rbgsize] = lteDCIResourceAllocation(dcistr)
```

## Description

`[prbset, nrbg, rbgsize] = lteDCIResourceAllocation(enbue,dcistr)` returns a matrix containing the zero-based physical resource block (PRB) indices `prbset`, the number of resource block groups `nrbg`, and the resource block group size `rbgsize`, for the specified DCI message settings structure `enbue` and DCI message structure `dcistr`.

TS 36.213 [1] specifies resource allocation types used for downlink, uplink and sidelink. For more information, see “Resource Allocation Types” on page 2-139.

If you specify DCI Format 0, Format 4, or Format 5 in `dcistr.DCIFormat`, the function sets the system bandwidth based on the number of uplink resource blocks, `enbue.NULRB`. If you do not specify `enbue.NULRB`, the function sets the system bandwidth based on the number of downlink resource blocks, `enbue.NDLRB`. For all other formats, the function first checks `enbue.NDLRB` for the number of resource blocks. For more information, see “Specifying Number of Resource Blocks” on page 2-140.

`[prbset, nrbg, rbgsize] = lteDCIResourceAllocation(dcistr)` returns outputs `prbset`, `nrbg`, and `rbgsize` as above, except the fields described in structure `enbue` must be present as part of `dcistr`.

Calling `lteDCIResourceAllocation` specifying the `dcistr` structure as the only input argument is not recommended because this signature will be removed in a future release.

## Examples

### Get Allocated PRB Indices for DCI Message

Allocate DCI resource and shows the allocation of the DCI resources.

Create a DCI message structure with a system bandwidth of 50 resource blocks and DCI Format 1A.

```
enb = struct('NDLRB',50);
dciStr = lteDCI(enb,struct('DCIFormat','Format1A','AllocationType',1));
```

Return allocated physical resource block indices, the number of resource block groups, and the resource block group size.

```
[prbSet, nrBg, rbgSize] = lteDCIResourceAllocation(enb,dciStr)
```

```
prbSet = 1x2 uint64 row vector
```

```
    0   27


nrBg = int32
    17

rbgSize = int32
    3
```

### Display Uplink PRB Allocation Type 1

Display the PRB allocations associated with the sequence of subframes in a frame for DCI Format 0 and uplink resource allocation type 1.

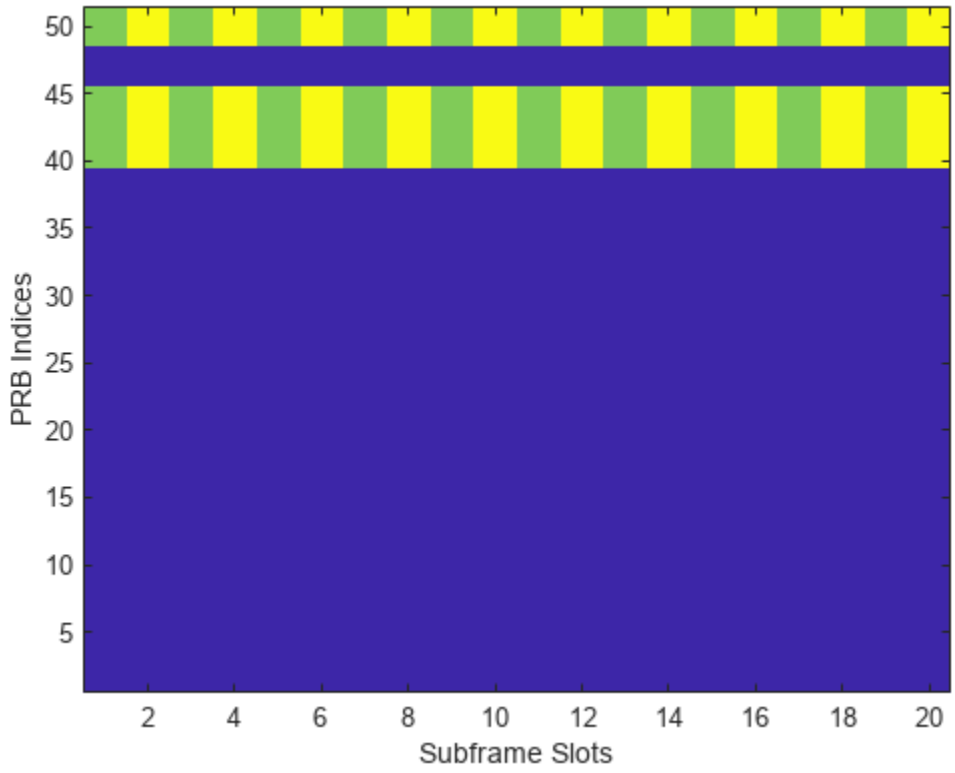
Configure a type 1 uplink resource allocation (multi-cluster). TS 36.213, Section 8.1.2 describes the resource indication value (RIV) determination.

```
enbue = struct('NDLRB',50);
dcistr = lteDCI(enbue,struct('DCIFormat','Format0','AllocationType',1));
dcistr.Allocation.RIV = 1;
```

Display an image of the PRBs used in each slot of each subframe in a frame.

- Create a `subframeslots` matrix full of zeros. There are 20 slots per frame, specifically two slots per subframe and ten subframes per frame.
- Loop through assigning a PRB set of indices for each subframe. Also assign a value in `subframeslots` for each occupied PRB index.

```
subframeslots = zeros(enbue.NDLRB,20);
for i = 0:9
    enbue.NSubframe = i;
    prbSet = lteDCIResourceAllocation(enbue,dcistr);
    prbSet = repmat(prbSet,1,2/size(prbSet,2));
    for s = 1:2
        subframeslots(prbSet(:,s)+1,2*i+s) = 20+s*20;
    end
end
imagesc(subframeslots);
axis xy;
xlabel('Subframe Slots');
ylabel('PRB Indices');
```




Observe from the image that the same set of PRB indices is used in each slot.

### Display Uplink Hopping PRB Allocation

Display the PRB allocations associated with the sequence of subframes in a frame for an uplink resource allocation with hopping.

Configure a type 1 uplink resource allocation that has type 0 hopping and slot and subframe hopping.

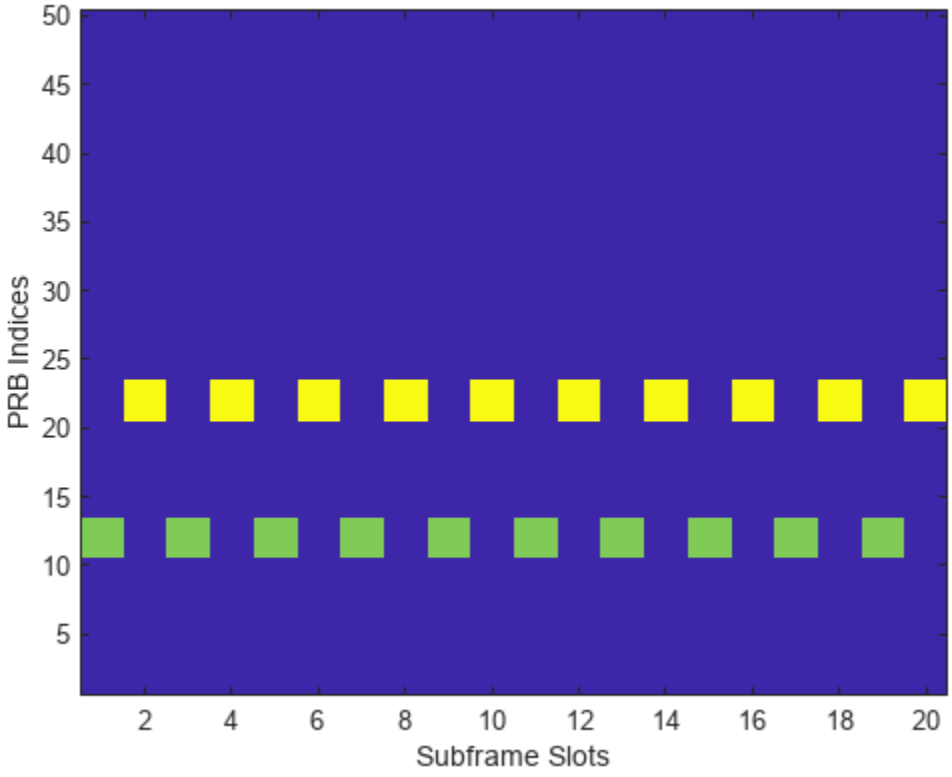
```
enbue = struct('NDLRB',50,'NCellID',0);
dcistr = lteDCI(enbue,struct('DCIFormat','Format0','AllocationType',0,...
    'FreqHopping',1));
dcistr.Allocation.HoppingBits = 0;
dcistr.Allocation.RIV = 110;
enbue.PUSCHHopping = 'InterAndIntra';
enbue.MacTxNumber = 0;
enbue.NSubbands = 1;
enbue.PUSCHHoppingOffset = 10;
```

Display an image of the PRBs used in each slot of each subframe in a frame.

- Create a `subframeslots` matrix full of zeros. There are 20 slots per frame, specifically two slots per subframe and ten subframes per frame.

- Loop through assigning a PRB set of indices for each subframe. Also assign a value in `subframeslots` for each occupied PRB index.

```
subframeslots = zeros(enbue.NDLRB,20);
for i = 0:9
    enbue.NSubframe = i;
    prbSet = lteDCIResourceAllocation(enbue,dcistr);
    prbSet = repmat(prbSet,1,2/size(prbSet,2));
    for s = 1:2
        subframeslots(prbSet(:,s)+1,2*i+s) = 20+s*20;
    end
end
imagesc(subframeslots)
axis xy
xlabel('Subframe Slots')
ylabel('PRB Indices')
```




Observe from the image that the occupied PRB indices hops in odd and even slots.

## Input Arguments

### enbue — DCI message settings

structure

DCI message settings, specified as a structure. `enbue` can contain the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NDLRB** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **CellRefP** | Optional | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **DuplexMode** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as one of the following:<br>• `'FDD'` — Frequency division duplex (default)<br>• `'TDD'` — Time division duplex |
| The following parameters apply when `dcistr.DCIFormat = 'Format0' or 'Format4'` | | | |
| **NULRB** | Required | Scalar integer from 6 to 110 | Number of uplink resource blocks. ($N_{RB}^{UL}$) |
| The following parameters apply when `dcistr.FreqHopping` = 1 | | | |
| **NCellID** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **NSubframe** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **NFrame** | Required | 0 (default), nonnegative scalar integer | Frame number |
| **PUSCHHopping** | Optional | `'Inter'` (default), `'InterAndIntra'` | Uplink subframe hopping mode |
| **MacTxNumber** | Optional | Scalar integer from 0 (default) to 27 | Number of the current MAC (re-)transmission, *CURRENT_TX_NB* |
| **NSubbands** | Optional | 1 (default), 2, 3, or 4 | Number of subbands. |
| **PUSCHHoppingOffset** | Optional | Scalar integer from 0 (default) to 98 | PUSCH hopping offset |
| The following parameters apply for DCI Format 5 (`dcistr = 'Format5'`). | | | |
| **NULRB** | Required | Scalar integer from 6 to 110 | Number of uplink resource blocks. ($N_{RB}^{UL}$) |
| The following parameters apply for DCI Format 5 (`dcistr = 'Format5'`) with frequency hopping (`dcistr.FreqHopping = 1`). | | | |
| **NSubframePSSCH** | Required | Integer | Subframe number in PSSCH subframe pool |
| **PSSCHHoppingParameter** | Optional | Integer from 0 (default) to 510. All values ≥ 504 are set to 510. | PSSCH hopping parameter |
| **NSubbands** | Optional | 1, 2, or 4 | Number of subbands |
| **PSSCHHoppingOffset** | Optional | Integer from 0 (default) to 110 | PSSCH hopping offset |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`PRBPool`** | Optional | Integer vector | PSSCH resource block pool (sidelink transmission mode 2). A vector of zero-based indices giving the PRBs in the pool. If absent or empty then the pool is assumed to be the full transmission bandwidth |
| The following parameters apply for DCI Format 5A (`dcistr = 'Format5A'`) . | | | |
| *`PSSCHNSubchannels`* | Optional | Integer from 1 (default) to 110 | Number of sub-channels in the V2X PSSCH resource pool |
| *`PSSCHSubchannelSize`* | Optional | Integer from 1 to 110. Default value is 4. | Number of PRB in each sub-channel |
| *`PSSCHSubchannelPRBStart`* | Optional | Integer from 0 (default) to 109 | First PRB index associated with first sub-channel of the resource pool |
| *`PSSCHAdjacency`* | Optional | `'On'` (default), `'Off'` | Whether PSCCH and PSSCH are transmitted in adjacent PRB |

Data Types: `struct`

**`dcistr` — DCI message structure**
structure

DCI message structure, returned as a structure whose fields match those of the associated DCI format.

The field names associated with `dcistr` are dependent on the DCI format. The format is expected to be one of the formats generated by `lteDCI`. The LTE standard defines resource allocations types for downlink, uplink, and sidelink. For more information, see “Resource Allocation Types” on page 2-139

The following table details the DCI formats and accompanying `dcistr` parameter fields.

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| `'Format0'` | `DCIFormat` | - | `'Format0'` |
| | `CIF` | 0 or 3 bits | Carrier indicator field |
| | `FreqHopping` | 1 bit | PUSCH frequency hopping flag |
| | `Allocation` | Varies | Resource block assignment/allocation |
| | `ModCoding` | 5 bits | Modulation, coding scheme, and redundancy version |
| | `NewData` | 1 bit | New data indicator |
| | `TPC` | 2 bits | PUSCH TPC command |
| | `CShiftDMRS` | 3 bits | Cyclic shift for DM RS |

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| | `TDDIndex` | 2 bits | For TDD config 0, this field is the Uplink Index.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | `CSIRequest` | 1, 2, or 3 bits | CSI request |
| | `SRSRequest` | 0 or 1 bit | SRS request. This field can only be present in DCI formats scheduling PUSCH which are mapped onto the UE specific search space given by the C-RNTI |
| | `AllocationType` | 1 bit | Resource allocation type, only present if $N_{RB}^{UL} \leq N_{RB}^{DL}$. |
| `'Format1'` | `DCIFormat` | - | `'Format1'` |
| | `CIF` | 0 or 3 bits | Carrier indicator field |
| | `AllocationType` | 1 bit | Resource allocation header: type 0, type 1. If downlink bandwidth is ≤10 PRBs there is no resource allocation header and resource allocation type 0 is assumed. |
| | `Allocation` | Varies | Resource block assignment/allocation |
| | `ModCoding` | 5 bits | Modulation and coding scheme |
| | `HARQNo` | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |
| | `NewData` | 1 bit | New data indicator |
| | `RV` | 2 bits | Redundancy version |
| | `TPCPUCCH` | 2 bits | PUCCH TPC command |
| | `TDDIndex` | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | `HARQACKResOffset` | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| `'Format1A'` | `DCIFormat` | - | `'Format1A'` |
| | `CIF` | 0 or 3 bits | Carrier indicator field |
| | `AllocationType` | 1 bit | VRB assignment flag: 0 (localized), 1 (distributed) |

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| | `Allocation` | Varies | Resource block assignment/allocation |
| | `ModCoding` | 5 bits | Modulation and coding scheme |
| | `HARQNo` | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |
| | `NewData` | 1 bit | New data indicator |
| | `RV` | 2 bits | Redundancy version |
| | `TPCPUCCH` | 2 bits | PUCCH TPC command |
| | `TDDIndex` | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | `SRSRequest` | 0 or 1 bit | SRS request. This field can only be present in DCI formats scheduling PUSCH which are mapped onto the UE specific search space given by the C-RNTI |
| | `HARQACKResOffset` | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| `'Format1B'` | `DCIFormat` | - | `'Format1B'` |
| | `CIF` | 0 or 3 bits | Carrier indicator field |
| | `AllocationType` | 1 bit | VRB assignment flag: 0 (localized), 1 (distributed) |
| | `Allocation` | Varies | Resource block assignment/allocation |
| | `ModCoding` | 5 bits | Modulation and coding scheme |
| | `HARQNo` | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |
| | `NewData` | 1 bit | New data indicator |
| | `RV` | 2 bits | Redundancy version |
| | `TPCPUCCH` | 2 bits | PUCCH TPC command |
| | `TDDIndex` | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| | TPMI | 2 bits for two antennas<br><br>4 bits for four antennas | PMI information |
| | PMI | 1 bit | PMI confirmation |
| | HARQACKResOffset | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| 'Format1C' | DCIFormat | - | 'Format1C' |
| | Allocation | Varies | Resource block assignment/allocation |
| | ModCoding | 5 bits | Modulation and coding scheme |
| 'Format1D' | DCIFormat | - | 'Format1D' |
| | CIF | 0 or 3 bits | Carrier indicator field |
| | AllocationType | 1 bit | VRB assignment flag: 0 (localized), 1 (distributed) |
| | Allocation | Varies | Resource block assignment/allocation |
| | ModCoding | 5 bits | Modulation and coding scheme |
| | HARQNo | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |
| | NewData | 1 bit | New data indicator |
| | RV | 2 bits | Redundancy version |
| | TPCPUCCH | 2 bits | PUCCH TPC command |
| | TDDIndex | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | TPMI | 2 bits for two antennas<br><br>4 bits for four antennas | Precoding TPMI information |
| | DlPowerOffset | 1 bit | Downlink power offset |
| | HARQACKResOffset | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| 'Format2' | DCIFormat | - | 'Format2' |
| | CIF | 0 or 3 bits | Carrier indicator field |

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| | `AllocationType` | 1 bit | Resource allocation header: type 0, type 1. If downlink bandwidth is ≤10 PRBs there is no resource allocation header and resource allocation type 0 is assumed. |
| | `Allocation` | Varies | Resource block assignment/allocation |
| | `TPCPUCCH` | 2 bits | PUCCH TPC command |
| | `TDDIndex` | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | `HARQNo` | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |
| | `SwapFlag` | 1 bit | Transport block to codeword swap flag |
| | `ModCoding1` | 5 bits | Modulation and coding scheme for transport block 1 |
| | `NewData1` | 1 bit | New data indicator for transport block 1 |
| | `RV1` | 2 bits | Redundancy version for transport block 1 |
| | `ModCoding2` | 5 bits | Modulation and coding scheme for transport block 2 |
| | `NewData2` | 1 bit | New data indicator for transport block 2 |
| | `RV2` | 2 bits | Redundancy version for transport block 2 |
| | `PrecodingInfo` | 3 bits for two antennas<br><br>6 bits for four antennas | Precoding information |
| | `HARQACKResOffset` | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| `'Format2A'` | `DCIFormat` | `-` | `'Format2A'` |
| | `CIF` | 0 or 3 bits | Carrier indicator field |
| | `AllocationType` | 1 bit | Resource allocation header: type 0, type 1. If downlink bandwidth is ≤10 PRBs there is no resource allocation header and resource allocation type 0 is assumed. |
| | `Allocation` | Varies | Resource block assignment/allocation |

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| | TPCPUCCH | 2 bits | PUCCH TPC command |
| | TDDIndex | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | HARQNo | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |
| | SwapFlag | 1 bit | Transport block to codeword swap flag |
| | ModCoding1 | 5 bits | Modulation and coding scheme for transport block 1 |
| | NewData1 | 1 bit | New data indicator for transport block 1 |
| | RV1 | 2 bits | Redundancy version for transport block 1 |
| | ModCoding2 | 5 bits | Modulation and coding scheme for transport block 2 |
| | NewData2 | 1 bit | New data indicator for transport block 2 |
| | RV2 | 2 bits | Redundancy version for transport block 2 |
| | PrecodingInfo | 0 bits for two antennas<br><br>2 bits for four antennas | Precoding information |
| | HARQACKResOffset | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| 'Format2B' | DCIFormat | - | 'Format2B' |
| | CIF | 0 or 3 bits | Carrier indicator field |
| | AllocationType | 1 bit | Resource allocation header: type 0, type 1. If downlink bandwidth is ≤10 PRBs there is no resource allocation header and resource allocation type 0 is assumed. |
| | Allocation | Varies | Resource block assignment/allocation |
| | TPCPUCCH | 2 bits | PUCCH TPC command |
| | TDDIndex | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| | `HARQNo` | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |
| | `ScramblingId` | 1 bit | Scrambling identity |
| | `ModCoding1` | 5 bits | Modulation and coding scheme for transport block 1 |
| | `NewData1` | 1 bit | New data indicator for transport block 1 |
| | `RV1` | 2 bits | Redundancy version for transport block 1 |
| | `ModCoding2` | 5 bits | Modulation and coding scheme for transport block 2 |
| | `NewData2` | 1 bit | New data indicator for transport block 2 |
| | `RV2` | 2 bits | Redundancy version for transport block 2 |
| | `HARQACKResOffset` | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| `'Format2C'` | `DCIFormat` | - | `'Format2C'` |
| | `CIF` | 0 or 3 bits | Carrier indicator field |
| | `AllocationType` | 1 bit | Resource allocation header: type 0, type 1. If downlink bandwidth is ≤10 PRBs there is no resource allocation header and resource allocation type 0 is assumed. |
| | `Allocation` | Varies | Resource block assignment/allocation |
| | `TPCPUCCH` | 2 bits | PUCCH TPC command |
| | `TDDIndex` | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | `HARQNo` | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |
| | `TxIndication` | 3 bits | Antenna ports, scrambling identity, and number of layers indicator |
| | `SRSRequest` | Varies | SRS request. Only present for TDD. |
| | `ModCoding1` | 5 bits | Modulation and coding scheme for transport block 1 |
| | `NewData1` | 1 bit | New data indicator for transport block 1 |

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| | RV1 | 2 bits | Redundancy version for transport block 1 |
| | ModCoding2 | 5 bits | Modulation and coding scheme for transport block 2 |
| | NewData2 | 1 bit | New data indicator for transport block 2 |
| | RV2 | 2 bits | Redundancy version for transport block 2 |
| | HARQACKResOffset | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| 'Format2D' | DCIFormat | - | 'Format2D' |
| | CIF | 0 or 3 bits | Carrier indicator field |
| | AllocationType | 1 bit | Resource allocation header: type 0, type 1. If downlink bandwidth is ≤10 PRBs there is no resource allocation header and resource allocation type 0 is assumed. |
| | Allocation | Varies | Resource block assignment/allocation |
| | TPCPUCCH | 2 bits | PUCCH TPC command |
| | TDDIndex | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | HARQNo | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |
| | TxIndication | 3 bits | Antenna ports, scrambling identity, and number of layers indicator |
| | SRSRequest | Varies | SRS request. Only present for TDD. |
| | ModCoding1 | 5 bits | Modulation and coding scheme for transport block 1 |
| | NewData1 | 1 bit | New data indicator for transport block 1 |
| | RV1 | 2 bits | Redundancy version for transport block 1 |
| | ModCoding2 | 5 bits | Modulation and coding scheme for transport block 2 |
| | NewData2 | 1 bit | New data indicator for transport block 2 |
| | RV2 | 2 bits | Redundancy version for transport block 2 |

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| | `REMappingAndQCL` | 2 bits | PDSCH RE Mapping and Quasi-Co-Location Indicator |
| | `HARQACKResOffset` | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| `'Format3'` | `DCIFormat` | - | `'Format3'` |
| | `TPCCommands` | Varies | TPC commands for PUCCH and PUSCH |
| `'Format3A'` | `DCIFormat` | - | `'Format3A'` |
| | `TPCCommands` | Varies | TPC commands for PUCCH and PUSCH |
| `'Format4'` | `DCIFormat` | - | `'Format4'` |
| | `CIF` | 0 or 3 bits | Carrier indicator field |
| | `Allocation` | Varies | Resource block assignment/allocation |
| | `TPC` | 2 bits | PUSCH TPC command |
| | `CShiftDMRS` | 3 bits | Cyclic shift for DM-RS |
| | `TDDIndex` | 2 bits | For TDD config 0, this field is Uplink Index.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | `CSIReq` | Varies | CSI request |
| | `SRSRequest` | 2 bits | SRS request |
| | `AllocationType` | 1 bit | Resource allocation header type 0 or type 1. |
| | `ModCoding` | 5 bits | Modulation, coding scheme, and redundancy version |
| | `NewData` | 1 bit | New data indicator |
| | `ModCoding1` | 5 bits | Modulation and coding scheme for transport block 1 |
| | `NewData1` | 1 bit | New data indicator for transport block 1 |
| | `ModCoding2` | 5 bits | Modulation and coding scheme for transport block 2 |
| | `NewData2` | 1 bit | New data indicator for transport block 2 |
| | `PrecodingInfo` | 3 bits for two antennas<br><br>6 bits for four antennas | Precoding information |
| `'Format5'` | `DCIFormat` | - | `'Format5'` |
| | `PSCCHResource` | 6 bits | Resource for PSCCH |

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| | `TPC` | 1 bit | TPC command for PSCCH and PSSCH |
| | `FreqHopping` | 1 bit | Frequency hopping flag |
| | `Allocation` | Varies | Resource block assignment and hopping resource allocation |
| | `TimeResourcePattern` | 7 bits | Time resource pattern |
| `'Format5A'` | `DCIFormat` | - | `'Format5A'` |
| | `CIF` | 3 bits | Carrier indicator |
| | `FirstSubchannelIdx` | $\left\lceil \log_2\left(N_{\text{subchannel}}^{\text{SL}}\right) \right\rceil$ | Lowest index of the subchannel allocation to the initial transmission |
| | `RIV` | from 0 to 13 bits, $\left\lceil \log_2 \left( \frac{N_{\text{subchannel}}^{\text{SL}} \times \left(N_{\text{subchannel}}^{\text{SL}} + 1\right)}{2} \right) \right\rceil$ | Resource indication value |
| | `TimeGap` | 4 bits | Time gap between initial transmission and retransmission |
| | `SLIndex` | 2 bits | SL SPS configuration index |

Data Types: `struct`

## Output Arguments

### `prbset` — Physical resource block indices

nonnegative integer column-vector | nonnegative integer column-matrix

Physical resource block indices, returned as a nonnegative integer column-vector or *N*-by-2 matrix of zero-based indices. The returned `prbset` will be a single column vector or a two-column matrix depending on whether the allocation type defines a different set of PRB indices in the first and second slots of the subframe.

Data Types: `uint64`

### `nrbg` — Number of resource block groups in the allocation

integer

Number of resource block groups in the allocation, returned as an integer.

Data Types: `int32`

### `rbgsize` — Resource block group size

integer

Number of resource blocks in a group, returned as an integer.

Data Types: `int32`

## More About

### Resource Allocation Types

The LTE standard specifies resource allocation types used for downlink, uplink and sidelink. For a detailed description of the resource allocation types, see `lteDCI`.

- For downlink, the LTE standard specifies three resource allocation types: type 0, 1, and 2. In terms of the DCI formats, formats 1, 2, 2A, 2B, 2C , and 2D can use either resource allocation type 0 or type 1, with the choice signalled by `dcistr.AllocationType=0` and `dcistr.AllocationType=1` respectively. DCI formats 1A, 1B, 1C, and 1D use resource allocation type 2, which can be configured to be localized or distributed across resource blocks, signalled by `dcistr.AllocationType=0` and `dcistr.AllocationType=1` respectively.
- For uplink allocations (signaled in DCI format 0 messages), the allocation type is either hopping or non-hopping, signalled by `dcistr.FreqHopping=1` and `dcistr.FreqHopping=0`, respectively.
  - For hopping allocations, there are two types of hopping: type 1 PUSCH hopping and type 2 PUSCH hopping (frequency hopping with a predefined pattern). The hopping type is signalled by `dcistr.Allocation.HoppingBits` as described in TS 36.213 [1], Table 8.4-2.
  - For non-hopping uplink allocations, there are two types of resource allocation: type 0 and type 1. These are signalled by `dcistr.AllocationType=0` and `dcistr.AllocationType=1` respectively. In case of DCI format 0 and uplink resource allocation type 1, the concatenation of the frequency hopping flag field (`dcistr.FreqHopping`) and the resource block assignment and hopping resource allocation field provides the resource allocation field (`dcistr.Allocation`). Type 0 allocations create a single contiguous set of PRB, whereas type 1 can create two contiguous PRB sets. The DCI format 4 messages can only signal non-hopping resource allocations type 0 and type 1.
- For D2D sidelink PSSCH (signaled by DCI format 5 messages), allocations are the same as uplink PUSCH allocation type 0, both non-hopping and hopping, but with a different set of additional parameters required in the hopping case. Details for sidelink transmission mode 1 and mode 2 are specified in TS 36.213 [1], Sections 14.1.1.2 and 14.1.1.4 respectively.
- Sidelink V2X PSSCH allocations (signalled by DCI format 5A messages when in sidelink transmission mode 3) create a single contiguous set of PRB using one or more sub-channels. See TS 36.213 [1], sections 14.1.1.4A and 14.1.1.4B for sidelink transmission mode 3 and mode 4 respectively. TM2 and TM4 use autonomous scheduling and therefore do not employ DCI messages from the eNodeB to deliver the transmission grants.

All allocations define a single set of PRB for both slots in a subframe (`prbset` is a column vector) except for the distributed resource allocation type 2 and uplink hopping allocations, where different PRB sets are used across the slot pair.

The allocation type may also define a minimum unit of resource block allocation, which is defined by the resource block group size, `rbgsize`. This specifies the number of resource blocks in a group. There are `nrbg` resource block groups in the allocation.

### Specifying Number of Resource Blocks

The number of resource blocks specifies the uplink and downlink bandwidth. The LTE Toolbox implementation assumes symmetric link bandwidth unless you specifically assign different values to `NULRB` and `NDLRB`. If the number of resource blocks is initialized in only one link direction, then the initialized number of resource blocks (`NULRB` or `NDLRB`) is used for both uplink and downlink. When this mapping is used, no warning is displayed. An error occurs if `NULRB` and `NDLRB` are both undefined.

## Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.213. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteDCI` | `ltePDCCH` | `lteDLSCH` | `lteEPDCCH` | `lteSCIResourceAllocation`

# lteDLChannelEstimate

Downlink channel estimation

## Syntax

```
[hest,noiseEst] = lteDLChannelEstimate(enb,rxgrid)
[hest,noiseEst] = lteDLChannelEstimate(enb,cec,rxgrid)
[hest,noiseEst] = lteDLChannelEstimate(enb,pdsch,cec,rxgrid)
[hest,noiseEst] = lteDLChannelEstimate(enb,epdcch,cec,rxgrid)
```

## Description

`[hest,noiseEst] = lteDLChannelEstimate(enb,rxgrid)` returns `hest`, the estimated channel response between each transmit and receive antenna for the input cell-wide settings `enb` and the resource grid `rxgrid`. The function also returns `noiseEst`, an estimate of the noise power spectral density on the reference signal subcarriers. For more information, see “Channel Estimation Processing” on page 2-159.

Use this syntax to estimate the channel in an LTE configuration by using the method described in Annex E of [1] and Annex F of [2].

`[hest,noiseEst] = lteDLChannelEstimate(enb,cec,rxgrid)` specifies the channel estimation method and parameters in the channel estimator configuration structure `cec`. The value that you specify for the `Reference` field in `cec` determines whether the function estimates the channel for an LTE or NB-IoT configuration.

`[hest,noiseEst] = lteDLChannelEstimate(enb,pdsch,cec,rxgrid)` performs physical downlink shared channel (PDSCH) estimation for `pdsch`, the PDSCH transmission configuration.

`[hest,noiseEst] = lteDLChannelEstimate(enb,epdcch,cec,rxgrid)` performs enhanced physical downlink control channel (EPDCCH) estimation for `epdcch`, the EPDCCH transmission configuration.

## Examples

### Estimate Downlink Channel Characteristics

Estimate the channel for an RMC R.12 (four-antenna transmit diversity) waveform.

Initialize a cell-wide configuration structure for transmission of RMC R.12.

```
rc = 'R.12';
enb = lteRMCDL(rc);
```

Initialize a channel estimation configuration. The averaging window size is configured in terms of resource elements (REs), time, and frequency. Use cubic interpolation with an averaging window of 1-by-1 REs. No noise estimate or averaging is required because no noise is not present in this example. You can therefore set the frequency window and time window size to one.

```
cec.FreqWindow = 1;
cec.TimeWindow = 1;
cec.InterpType = 'cubic';
cec.PilotAverage = 'UserDefined';
cec.InterpWinSize = 3;
cec.InterpWindow = 'Causal';
```

Generate a transmit waveform for the specified cell-wide settings by using the `lteRMCDLTool` function.

```
txWaveform = lteRMCDLTool(enb,[1;0;0;1]);
```

Model the propagation channel by combining all transmit antennas into one receive antenna.

```
rxWaveform = sum(txWaveform,2);
```

Perform OFDM demodulation.

```
rxGrid = lteOFDMDemodulate(enb,rxWaveform);
```

Estimate the channel characteristics, displaying the size of the returned array. Confirm that the noise power spectral density estimate is zero.

```
[hest,noiseEst] = lteDLChannelEstimate(enb,cec,rxGrid);
disp(size(hest))
```

```
    72   140     1     4
```

```
disp(noiseEst)
```

```
     0
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure. The fields that you specify in `enb` depend on whether the function performs channel estimation for an LTE or NB-IoT configuration.[1]

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **NDLRB** | Required for LTE configuration | Integer in the interval [6, 110] | Number of downlink resource blocks | This field applies only when you specify the `Reference` field of the `cec` input to a value other than `'NRS'`. | `double` |

1 The value to which you set the `Reference` field of the `cec` input determines whether the function performs channel estimation for an LTE or NB-IoT configuration.

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **`CellRefP`** | Required for LTE configuration | `1`, `2`, `4` | Number of cell-specific reference signal (CRS) antenna ports | This field applies only when you specify the `Reference` field of the `cec` input to a value other than `'NRS'`. | `double` |
| **`NCellID`** | Required for LTE configuration | Integer in the interval [0, 503] | Physical layer cell identity (PCI) | This field applies only when you specify the `Reference` field of the `cec` input to a value other than `'NRS'`. | `double` |
| **`NSubframe`** | Required | Nonnegative integer | Subframe number | Not applicable | `double` |
| **`CyclicPrefix`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length | Not applicable | `char`, `string` |
| **`DuplexMode`** | Optional | `'FDD'` (default), `'TDD'` | Duplex mode, specified as `'FDD'` for frequency division duplex or `'TDD'` for time division duplex. | Not applicable | `char`, `string` |
| **`TDDConfig`** | Optional | `1` (default), integer in the interval [0, 6] | Uplink-downlink configuration; for more information, see Section 4.2 of [3]. | This field applies only when you specify the `DuplexMode` field as `'TDD'`. | `double` |
| **`SSC`** | Optional | `0` (default), integer in the interval [0, 9] | Special subframe configuration; for more information, see Section 4.2 of [3]. | This field applies only when you specify the `DuplexMode` field as `'TDD'`. | `double` |
| **`CSIRefP`** | Required when you specify the `Reference` field of the `cec` input as `'CSIRS'`. | `1`, `2`, `4`, `8` | Number of channel-state information reference signal (CSI-RS) antenna ports | This field applies only when you specify the `Reference` field of the `cec` input as `'CSIRS'`. | `double` |

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **CSIRSConfig** | Required when you specify the `Reference` field of the `cec` input as `'CSIRS'`. | Integer in the interval [0, 31] | CSI-RS configuration index; for more information, see Table 6.10.5.2-1 in [3]. | This field applies only when you specify the `Reference` field of the `cec` input as `'CSIRS'`. | `double` |
| **CSIRSPeriod** | Optional | `'On'` (default), `'off'`, integer in the interval [0, 154], 1-by-2 vector of integers | CSI-RS subframe configuration, specified as one of these values:<br>• `'On'`<br>• `'Off'`<br>• An integer in the interval [0, 154] corresponding to the value of $I_{\text{CSI-RS}}$ in Table 6.10.5.3-1 of [3]<br>• A 1-by-2 vector of integers in the form [$T_{\text{CSI-RS}}$ $\Delta_{\text{CSI-RS}}$], where<br>  • $T_{\text{CSI-RS}}$ is the CSI-RS periodicity<br>  • $\Delta_{\text{CSI-RS}}$ is the CSI-RS subframe offset<br>For more information, see Table 6.10.5.3-1 of [3]. | This field applies only when you specify the `Reference` field of the `cec` input as `'CSIRS'`. | `double`, `char`, `string` |
| **NNCellID** | Required for NB-IoT configuration | Integer in the interval [0, 503] | Narrowband PCI | This field applies only when you specify the `Reference` field of the `cec` input as `'NRS'`. | `double` |

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **NBRefP** | Required for NB-IoT configuration | 1, 2 | Number of narrowband reference signal (NRS) antenna ports | This field applies only when you specify the `Reference` field of the `cec` input as `'NRS'`. | `double` |

Data Types: `struct`

### `rxgrid` — Received resource element grid

complex-valued 3-D array

Received resource element grid, specified as a complex-valued array of size $N_{SC}$-by-$N_{Sym}$-by-$N_R$, where:

- $N_{SC}$ is the number of subcarriers
- $N_{Sym} = N_{SF} \times N_{SymPerSF}$ is the number of OFDM symbols, where:
  - $N_{SF}$ is the total number of subframes

    **Note** To adhere to the estimation method defined in [1] and [2], $N_{SF}$ must be `10`.
  - $N_{SymPerSF}$ is the number of OFDM symbols per subframe
    - For normal cyclic prefix, each subframe contains 14 OFDM symbols.
    - For extended cyclic prefix, each subframe contains 12 OFDM symbols.
- $N_R$ is the number of receive antennas

### `cec` — Channel estimation configuration

structure

Channel estimation configuration, specified as a structure containing these fields.

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **PilotAverage** | Required | `'TestEVM'`, `'UserDefined'` | Type of pilot averaging[a] | The `'TestEVM'` value applies only when you specify the `Reference` field as a value other than `'NRS'`. | `char`, `string` |
| **FreqWindow** | Required | Positive integer | Size of window for frequency averaging, in resource elements | Not applicable | `double` |

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **`TimeWindow`** | Required | Positive integer | Size of window for time averaging, in resource elements | Not applicable | `double` |
| **`InterpType`** | Required | `'nearest'`, `'linear'`, `'natural'`, `'cubic'`, `'v4'`, `'none'` | Type of interpolation between pilot symbols, specified as one of these values:<br>• `'nearest'` – Nearest neighbor interpolation<br>• `'linear'` – Linear interpolation<br>• `'natural'` – Natural neighbor interpolation<br>• `'cubic'` – Cubic interpolation<br>• `'v4'` – MATLAB 4 `griddata` method<br>• `'none'` – No interpolation[b]<br>For more information, see the `griddata` function. | Not applicable | `char`, `string` |
| **`InterpWindow`** | Required | `'Causal'`, `'Non-causal'`, `'Centred'`, `'Centered'` | Interpolation type; the values `'Centred'` and `'Centered'` are equivalent. For more information, see “Noise Reduction and Interpolation” on page 2-159. | Not applicable | `char`, `string` |

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **`InterpWinSize`** | Required | Positive scalar | Interpolation window size, in number of subframes | If you specify the `InterpWindow` field as `'Centred'` or `'Centered'`, you cannot specify this field as an even integer. | `double` |

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **`Reference`** | Optional | `'DMRS'` (Default), `'CSIRS'`, `'CellRS'`, `'EPDCCHDMRS'`, `'NRS'` | Reference signals for channel estimation, specified as one of these values:<br>• `'DMRS'` – Perform PDSCH estimation by using the demodulation reference signals (DM-RSs)<br>• `'CSIRS'` – Perform PDSCH estimation by using the channel-state information reference signals (CSI-RSs)[c]<br>• `'CellRS'` – Perform channel estimation by using the cell-specific reference signals (CRSs)<br>• `'EPDCCHDMRS'` – Perform EPDCCH estimation by using the DM-RSs<br>• `'NRS'` – Perform channel estimation for an NB-IoT configuration | This field applies only when you specify one of these configurations:<br>• PDSCH channel estimation with the `TxScheme` field of the `pdsch` argument specified as one of these values: `'Port5'`, `'Port7-8'`, `'Port8'`, `'Port7-14'`<br>• EPDCCH channel estimation<br>• NB-IoT channel estimation | `char`, `string` |

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| | | | by using the NRSs. | | |

a If you specify this field as `'TestEVM'`, the function ignores any other fields you specify in `cec`. The function performs pilot averaging according to the method set out in Annex E of [1] and Annex F of [2]. This method is for transmitter error vector magnitude (EVM) testing and is not supported for NB-IoT configurations.

When you specify this field as `'UserDefined'`, the function performs pilot averaging with a rectangular kernel of size `FreqWindow`-by-`TimeWindow`. The function also performs a two-dimensional filtering operation on the pilots. The pilots near the edge of the resource grid either have no neighbors or a limited number of neighbors through the creation of virtual pilots. Consequently, these pilots are not averaged in the same way as pilots away from the edge of the resource grid.

b When you specify this field as `'none'`, the function performs no interpolation between pilot symbols and does not create virtual pilots. The `hest` output contains channel estimates in the locations of transmitted reference symbols for each receive antenna, and all other elements of `hest` are `0`. The function still performs pilot symbol averaging in accordance with the values you specify for the `FreqWindow` and `TimeWindow` fields.

c CSI-RS-based channel estimation is strictly only valid within the standard for the transmission scheme corresponding to the `'Port7-14'` value of the `TxScheme` field of the `pdsch` argument. For more information, see Section 6.10.5.3 of [3].

### `pdsch` — PDSCH transmission configuration

structure

PDSCH transmission configuration, specified as a structure containing these fields.

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **TxScheme** | Required | 'Port0', 'TxDiversity', 'CDD', 'SpatialMux', 'MultiUser', 'Port5', 'Port7-8', 'Port8', 'Port7-14' | PDSCH transmission scheme, specified as one of these values:<br>• 'Port0' – Single-antenna port, port 0<br>• 'TxDiversity' – Transmit diversity<br>• 'CDD' – Large-delay cyclic delay diversity (CDD) scheme<br>• 'SpatialMux' – Closed-loop spatial multiplexing<br>• 'MultiUser' – Multiuser multiple-input/multiple-output (MIMO)<br>• 'Port5' – Single-antenna port, port 5<br>• 'Port7-8' – Single-antenna, port 7 when the NLayers field is 1; dual-layer transmission, ports 7 and 8 when the | Not applicable | char, string |

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| | | | `NLayers` field is 2<br>• `'Port8'` – Single-antenna port, port 8<br>• `'Port7-14'` – Up to eight-layer transmission, ports 7–14 | | |

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **PRBSet** | Required | Column vector of integers, two-column matrix of integers, cell array | Physical resource block (PRB) indices, in zero-based form, corresponding to the slot-wise resource allocations for the PDSCH. Specify this field as one of:<br>• A column vector of integers, for which the resource allocation is the same in both slots of the subframe<br>• A two-column matrix, in which you can specify PRBs for each slot in a subframe<br>• A cell array of length 10, corresponding to a frame if the allocated PRBs vary across subframes<br>This field varies per subframe for these reference measurement channels (RMCs): `'R.25'` (TDD), | Not applicable | `single`, `double`, `cell` |

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| | | | `'R.26'` (TDD), `'R.27'` (TDD), `'R.43'` (FDD), `'R.44'`, `'R.45'`, `'R.48'`, `'R.50'`, and `'R.51'`. | | |
| **`RNTI`** | Required | Nonnegative integer | Radio network temporary identifier (RNTI) value | Not applicable | `double` |
| **`NLayers`** | Required | Integer in the interval [1, 8] | Number of transmission layers | This field applies only when you specify the `TxScheme` field as one of these values: `'Port5'`, `'Port7-8'`, `'Port8'`, `'Port7-14'`. | `double` |

You can initialize a special case by specifying:

- The `TxScheme` field of `pdsch` as `'Port7-8'`, `'Port8'`, or `'Port7-14'`
- The `PilotAverage` field of `cec` as `'UserDefined'`
- The `TimeWindow` field of `cec` as `2` or `4`
- The `FreqWindow` field of `cec` as `1`.

The function uses a window of two or four pilots in time to average the pilot estimates. For this configuration, averaging is always applied across two or four pilots, regardless of their separation in OFDM symbols. Averaging is required for the UE-RS and CSI-RS ports because they occupy the same time/frequency locations, using different orthogonal covers for the receiver to differentiate them.

- For the CSI-RS with any number of configured CSI-RS antenna ports, the pilot REs occur in one pair per subframe. The CSI-RS pilot RE pairs are averaged with the `TimeWindow` field of `cec` set to `2`, resulting in one channel estimate per subframe.
- For the UE-RS with the `NLayers` field of `pdsch` specified as `1`, `2`, `3`, or `4`, the pilot REs occur in pairs repeated in each slot. The UE-RS pilot RE pairs are averaged with the `TimeWindow` field of `cec` set to `2`, resulting in two estimates per subframe, one for each slot.

For the UE-RS with the `NLayers` field of `pdsch` specified as `5`, `6`, `7`, or `8`, the pairs are distinct between the slots of the subframe. The pairs are averaged with the `TimeWindow` field of `cec` set to `4`, resulting in one estimate per subframe. In these cases, `rxgrid` must contain only one subframe because only a single subframe can be estimated.

Data Types: `struct`

**epdcch — EPDCCH transmission configuration**
structure

EPDCCH transmission configuration, specified as a structure containing these fields.

| Name | Required or Optional | Values | Description | Data Types |
|---|---|---|---|---|
| **EPDCCHType** | Required | `'Localized'`, `'Distributed'` | EPDCCH transmission type. As indicated in Table 6.8A.5-1 of [3], the function performs channel estimation according to the value you specify for this field.<br>• When you specify this field as `'Localized'`, the function performs channel estimation in one of these sets of antenna ports: {107, 108, 109, 110}, {107, 109}, or {107, 108}. The antenna ports used depend on the cell configuration.<br>• When you specify this field as `'Distributed'`, the function performs channel estimation in the pair of EPDCCH antenna ports used for EPDCCH transmission. When you specify the `CyclicPrefix` field of the `enb` input as | `char`, `string` |

| Name | Required or Optional | Values | Description | Data Types |
|---|---|---|---|---|
| | | | `'Normal'`, the function uses antenna ports 107 and 109. When you specify the `CyclicPrefix` field of the `enb` input as `'Extended'`, the function uses antenna ports 107 and 108.<br>• In other EPDCCH antenna ports, the channel estimate is zero. | |

| Name | Required or Optional | Values | Description | Data Types |
|---|---|---|---|---|
| **`EPDCCHPRBSet`** | Required | Vector of integers | EPDCCH PRB pair indices, in zero-based form. The length of this field must be a power of two. If no transmission is required, specify this field as an empty vector.<br><br>The function returns only a channel estimate for the PRB pairs that you specify in this field, but performs estimation for all EPDCCH candidate locations within those pairs. In other PRBs, the function interpolates the channel estimate according to the interpolation type that you specify in the `InterpType` field of the `cec` input. | `double` |
| **`EPCCHNID`** | Required | Nonnegative integer | EPDCCH scrambler initialization parameter. This field represents the parameter $n_{\mathrm{ID},m}^{\mathrm{EPDCCH}}$ in the definition of the initial state of the scrambling sequence generator, given in Section 6.8A.2 of [3]. | `double` |

**Note** Specifying the `PilotAverage`, `TimeWindow`, and `FreqWindow` fields of the `cec` input as `'UserDefined'`, `2`, and `1`, respectively, initializes a special case. The function performs the "despreading" pilot averaging behavior described in the note for the `TxScheme` field of the `pdsch` input. This behavior results because the EPDCCH DMRS and PDSCH DMRS RE have the same arrangement and employ the same use of orthogonal cover codes.

#### Dependencies

This argument applies only when you specify the `Reference` field of the `cec` input as `'EPDCCHDMRS'`.

Data Types: `struct`

## Output Arguments

### `hest` — Estimated channel between transmit and receive antennas

complex-valued 4-D array

Estimated channel between transmit and receive antennas, returned as a complex-valued 4-D array. The fourth dimension of `hest` varies based on the reference signal option you specify in the `Reference` field of the `cec` argument and the `TxScheme` field of the `pdsch` input.

| Value of `Reference` Field of `cec` | Output Array Dimensions | RS-Specific Dimension | Transmission Scheme |
|---|---|---|---|
| `'DMRS'` | $N_{SC}$-by-$N_{Sym}$-by-$N_R$-by-`NLayers` | `NLayers` is the number of transmission layers. | `'Port5'`, `'Port7-8'`, `'Port8'`, and `'Port7-14'` |
| `'CSIRS'` | $N_{SC}$-by-$N_{Sym}$-by-$N_R$-by-`CSIRefP` | `CSIRefP` is the number of CSI-RS antenna ports. | `'Port5'`, `'Port7-8'`, `'Port8'`, and `'Port7-14'` |
| `'CellRS'` | $N_{SC}$-by-$N_{Sym}$-by-$N_R$-by-`CellRefP` | `CellRefP` is the number of cell-specific reference signal antenna ports. | `'SpatialMux'`, `'Port0'`, `'TxDiversity'`, `'CDD'`, `'MultiUser'`, `'Port5'`, `'Port7-8'`, `'Port8'`, `'Port7-14'` |
| `'EPDCCHDMRS'` | $N_{SC}$-by-$N_{Sym}$-by-$N_R$-by-4 | Estimate across all four possible EPDCCH ports (107–110), which ensures consistency with the indexing used by the `lteEPDCCHDMRSIndices` and `lteEPDCCHIndices` functions | Not applicable |
| `'NRS'` | $N_{SC}$-by-$N_{Sym}$-by-$N_R$-by-`NBRefP` | `NBRefP` is the number of NRS antenna ports. | Not applicable |

| Value of `Reference` Field of `cec` | Output Array Dimensions | RS-Specific Dimension | Transmission Scheme |
|---|---|---|---|
| Output array dimensions:<br>• $N_{SC}$ is the number of subcarriers.<br>• $N_{Sym}$ is the number of OFDM symbols.<br>• $N_R$ is the number of receive antennas. | | | |

Data Types: `double`

**`noiseEst` — Noise power spectral density estimate**
real-valued scalar

Noise power spectral density estimate on reference signal subcarriers, returned as a real-valued scalar. The function computes `noiseest` by using the reference signals.

Data Types: `double`

## Algorithms

### Channel Estimation Processing

The steps associated with channel estimation processing are:

1. Extract the reference signals, or pilot symbols, for a transmit-receive antenna pair from the received grid. Use the reference signals to calculate the least-squares estimates of the channel response at the pilot symbol positions within a received grid.

   The function obtains the least-squares estimates of the reference signals by dividing the received pilot symbols by their expected value. Any system noise affects the least-squares estimates. Remove or reduce the noise to achieve a reasonable estimation of the channel at pilot symbol locations. For more information, see “Noise Reduction and Interpolation” on page 2-159.
2. Average the least-squares estimates to reduce any unwanted noise from the pilot symbols.
3. Interpolate the cleaned pilot symbol estimates into an estimate of the channel for the entire number of subframes passed into the function.

### Noise Reduction and Interpolation

To minimize the effects of noise on the pilot symbol estimates, the function averages the least-squares estimates through an averaging window. This method ensures a substantial reduction in the level of noise found on the pilot symbols. The two pilot symbol averaging methods, which also define the interpolation method performed to obtain the channel estimate, are `'TestEVM'` and `'UserDefined'`.

- `'TestEVM'` — Follows the method described in Annex F.3.4 of [2]. The function performs time averaging across each pilot symbol carrying subcarrier, resulting in a column vector containing the time averaged estimates of the channel. The function then performs frequency averaging by using a moving window with a maximum size of 19. The function uses linear interpolation to estimate the values between the pilot symbols. The function replicates the estimated vector and uses it as the entire channel estimate.

**Note** For `'TestEVM'`, there are no user-defined parameters. Estimation behaves as described in [2].

The algorithm differs from the implementation described in [2] due to the number of subframes across which time-averaging is performed. In [2], the method requires 10 subframes. The `lteDLChannelEstimate` function performs time averaging across the total number of subframes contained in the `rxgrid` input.

- `'UserDefined'` — Uses an averaging window that you define. The averaging window size is in resource elements. Any pilot symbols located within the window are used to average the value of the pilot symbol found at the center of the window. The function uses the averaged pilot symbol estimates to perform a 2-D interpolation across a window of subframes. The location of pilot symbols within the subframe is not ideally suited to interpolation. To account for this issue, the function creates virtual pilots and places them outside the area of the current subframe. This approach allows for complete and accurate interpolation. The `InterpWindow` field defines the causal nature of the available data. Valid settings for `InterpWindow` are `'Causal'`, `'Non-causal'`, `'Centred'`, or `'Centered'`.

  The value that you specify for `InterpWindow` depends upon the data that you use for interpolation.

  - `'Causal'` – Use past data.
  - `'Non-causal'` – Use future data, the opposite of `'Causal'`. Relying on only future data is commonly referred to as an anti-causal method of interpolation.
  - `'Centered'` or `'Centred'` – Use a combination of past, present, and future data.

# Version History

**Introduced in R2013b**

## References

[1] 3GPP TS 36.104. “Base Station (BS) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*.

[2] 3GPP TS 36.141. “Base Station (BS) conformance testing.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*.

[3] 3GPP TS 36.211. “Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*.

## See Also

`lteOFDMDemodulate` | `lteEqualizeMIMO` | `lteEqualizeMMSE` | `lteEqualizeZF` | `lteDLPerfectChannelEstimate` | `griddata`

**Topics**

“Channel Estimation”

# lteDLConformanceTestTool

Opens the LTE Throughput Analyzer app for performing downlink PDSCH demodulation conformance tests

## Syntax

```
lteDLConformanceTestTool
```

## Description

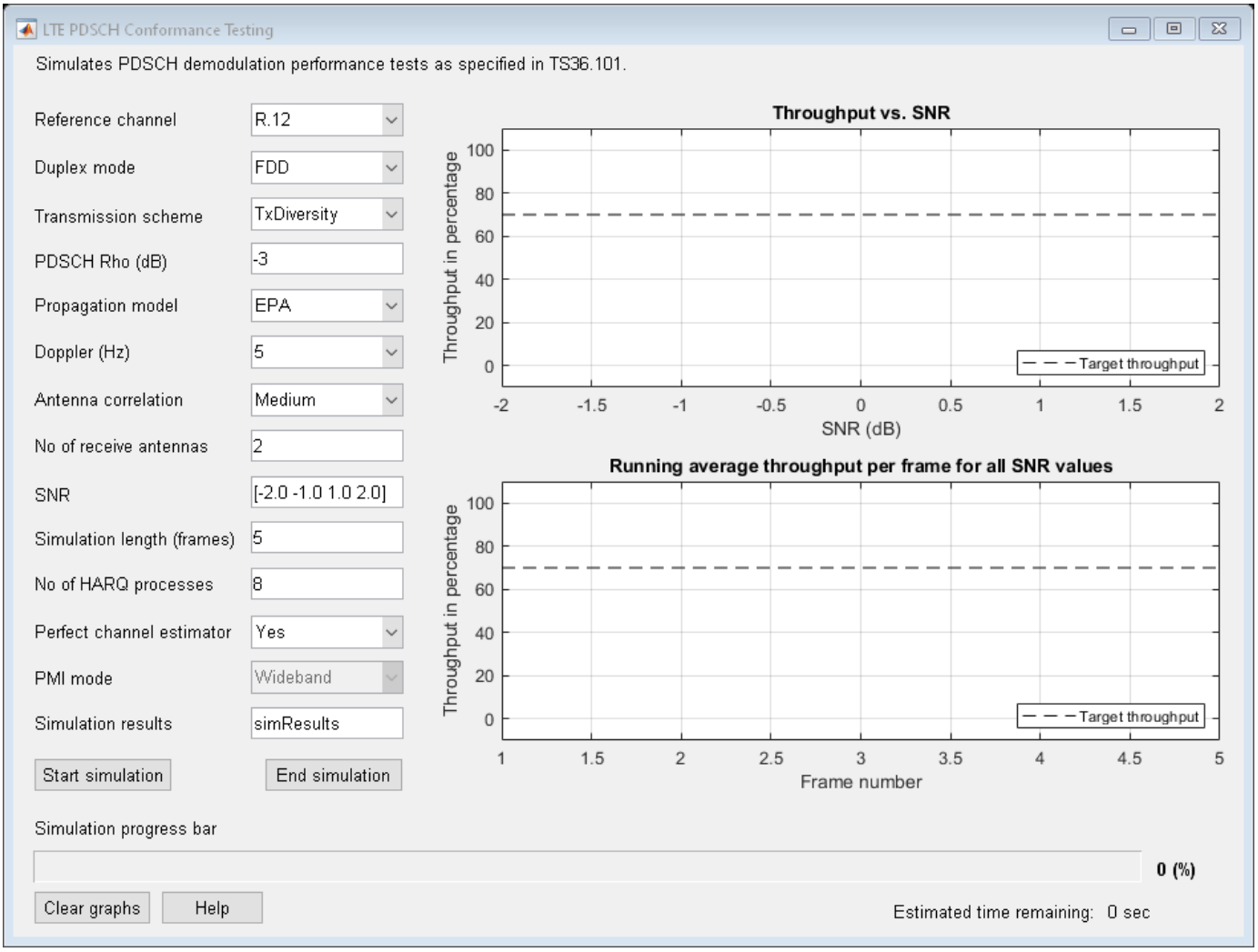
`lteDLConformanceTestTool` opens the **LTE Throughput Analyzer** app for performing downlink PDSCH demodulation conformance tests as defined in TS 36.101 [1].

The throughput performance graphs update dynamically during the simulation run and provides an early understanding system behavior for a given setup. For more information, see **LTE Throughput Analyzer**.

## Examples

### Open Throughput Analyzer App

```
lteDLConformanceTestTool
```




The LTE PDSCH Conformance Testing user interface opens.

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

**Apps**
**LTE Throughput Analyzer**

**Functions**
`lteRMCDLTool` | `lteRMCULTool` | `lteTestModel`

**Topics**
“Analyze Throughput for PDSCH Demodulation Performance Test”

# lteDLDeprecode

Downlink deprecoding onto transmission layers

## Syntax

```
out = lteDLDeprecode(in,nu,txscheme,codebook)
out = lteDLDeprecode(enb,chs,in)
```

## Description

`out = lteDLDeprecode(in,nu,txscheme,codebook)` returns a symbol matrix by performing deprecoding using matrix pseudo-inversion to undo processing described in TS 36.211 [1], Section 6.3.4. The overall operation of the deprecoder is to transpose what is defined in the specification.

`out = lteDLDeprecode(enb,chs,in)` performs deprecoding of the precoded symbol matrix, `in`, according to cell-wide settings `enb` and `chs` (channel transmission configurations).

## Examples

### Perform Deprecoding on Identity Matrix

Deprecode a precoded identity matrix having codebook index 1 for three layers and four antennas.

```
in  = lteDLPrecode(eye(3),4,'SpatialMux',1);
out = lteDLDeprecode(in,3,'SpatialMux',1)
```

```
out = 3×3 complex

   1.0000 + 0.0000i   0.0000 - 0.0000i  -0.0000 + 0.0000i
   0.0000 - 0.0000i   1.0000 + 0.0000i   0.0000 + 0.0000i
  -0.0000 + 0.0000i   0.0000 - 0.0000i   1.0000 + 0.0000i
```

## Input Arguments

### `in` — Precoded input symbols

numeric matrix

Precoded input symbols, specified as numeric matrix. The size of the matrix is *N*-by-*P*, where *P* is the number of transmission antennas and *N* is the number of symbols per antenna. Generate the matrix by extracting a PDSCH using `ltePDSCHIndices` function on a received resource array. You can perform a similar extraction using the index generator for any other downlink channel that utilizes precoding.

### `nu` — Number of layers

integer from 1 to 8

Number of layers, specified as an integer from 1 to 8. The maximum number of layers depends on the transmission scheme, `txscheme`.

Data Types: `double`

**`txscheme` — Transmission scheme**
`'Port0'` | `'TxDiversity'` | `'CDD'` | `'SpatialMux'` | `'MultiUser'` | `'Port5'` | `'Port7-8'` | `'Port8'` | `'Port7-14'`

PDSCH transmission scheme, specified as one of the following options.

| Transmission scheme | Description |
|---|---|
| `'Port0'` | Single antenna port, port 0 |
| `'TxDiversity'` | Transmit diversity |
| `'CDD'` | Large delay cyclic delay diversity scheme |
| `'SpatialMux'` | Closed loop spatial multiplexing |
| `'MultiUser'` | Multi-user MIMO |
| `'Port5'` | Single-antenna port, port 5 |
| `'Port7-8'` | Single-antenna port, port 7, when `NLayers` = 1. Dual layer transmission, ports 7 and 8, when `NLayers` = 2. |
| `'Port8'` | Single-antenna port, port 8 |
| `'Port7-14'` | Up to eight layer transmission, ports 7–14 |

Data Types: `char` | `single`

**`codebook` — Codebook index**
integer from 0 to 15

Codebook index to select the precoding matrix, specified as an integer from 0 to 15. This input is ignored for the `'Port0'`, `'TxDiversity'`, and `'CDD'` transmission schemes. Find the precoding matrix corresponding to a particular codebook index in TS 36.211 [1], Section 6.3.4. In the case of `'TxDiversity'` and `nu=1`, the function falls back to single port processing.

Data Types: `double`

**`enb` — eNodeB cell-wide settings**
structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields:

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| When `chs.TxScheme` is set to `'TxDiversity'`, `'CDD'`, `'SpatialMux'`, or `'MultiUser'`, these parameters are applicable: | | | |
| **`CellRefP`** | Required | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| When `chs.TxScheme` is set to `'SpatialMux'`, or `'MultiUser'` and `chs.PMISet` is present, these parameters are applicable:. | | | |
| **`NCellID`** | Required | Integer from 0 to 503 | Physical layer cell identity |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NSubframe** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **NDLRB** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **CFI** | Required | 1, 2, or 3<br>Scalar or if the CFI varies per subframe, a vector of length 10 (corresponding to a frame). | Control format indicator (`CFI`) value. In TDD mode, `CFI` varies per subframe for the RMCs (`'R.0'`, `'R.5'`, `'R.6'`, `'R.6-27RB'`, `'R.12-9RB'`) |
| **CyclicPrefix** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **DuplexMode** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| When `DuplexMode` is set to `'TDD'`, these parameters are applicable: | | | |
| **TDDConfig** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **SSC** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |

Data Types: `struct`

**`chs` — Channel-specific transmission configuration**
structure

Channel-specific transmission configuration, specified as a structure that can contain the following parameter fields:

<table>
<tr><th>Parameter Field</th><th>Required or Optional</th><th>Values</th><th>Description</th></tr>
<tr><td>TxScheme</td><td>Required</td><td>'Port0', 'TxDiversity', 'CDD', 'SpatialMux', 'MultiUser', 'Port5', 'Port7-8', 'Port8', 'Port7-14'.</td><td>PDSCH transmission scheme, specified as one of the following options.
<table>
<tr><th>Transmission scheme</th><th>Description</th></tr>
<tr><td>'Port0'</td><td>Single antenna port, port 0</td></tr>
<tr><td>'TxDiversity'</td><td>Transmit diversity</td></tr>
<tr><td>'CDD'</td><td>Large delay cyclic delay diversity scheme</td></tr>
<tr><td>'SpatialMux'</td><td>Closed loop spatial multiplexing</td></tr>
<tr><td>'MultiUser'</td><td>Multi-user MIMO</td></tr>
<tr><td>'Port5'</td><td>Single-antenna port, port 5</td></tr>
<tr><td>'Port7-8'</td><td>Single-antenna port, port 7, when NLayers = 1. Dual layer transmission, ports 7 and 8, when NLayers = 2.</td></tr>
<tr><td>'Port8'</td><td>Single-antenna port, port 8</td></tr>
<tr><td>'Port7-14'</td><td>Up to eight layer transmission, ports 7–14</td></tr>
</table></td></tr>
<tr><td>NLayers</td><td>Required</td><td>Integer from 1 to 8</td><td>Number of transmission layers.</td></tr>
<tr><td colspan="4">The following parameters are applicable when TxScheme is set to 'SpatialMux' or 'MultiUser'. Include either CodebookIdx field or both PMISet and PRBSet fields. For more information, see Algorithms on page 2-169.</td></tr>
<tr><td>CodebookIdx</td><td>Required</td><td>Integer from 0 to 15</td><td>Codebook index used during precoding</td></tr>
</table>

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`PMISet`** | Required | Integer vector with element values from 0 to 15. | Precoder matrix indication (PMI) set. It can contain either a single value, corresponding to single PMI mode, or multiple values, corresponding to multiple or subband PMI mode. The number of values depends on CellRefP, transmission layers and TxScheme. For more information about setting PMI parameters, see `ltePMIInfo`. |
| **`PRBSet`** | Required | Integer column vector or two-column matrix | Zero-based physical resource block (PRB) indices corresponding to the slot wise resource allocations for this PDSCH. `PRBSet` can be assigned as:<br>• a column vector, the resource allocation is the same in both slots of the subframe,<br>• a two-column matrix, this parameter specifies different PRBs for each slot in a subframe,<br>• a cell array of length 10 (corresponding to a frame, if the allocated physical resource blocks vary across subframes).<br>PRBSet varies per subframe for the RMCs `'R.25'`(TDD), `'R.26'`(TDD), `'R.27'`(TDD), `'R.43'`(FDD), `'R.44'`, `'R.45'`, `'R.48'`, `'R.50'`, and `'R.51'`. |
| The fields `PMISet` and `PRBSet` are used to determine the frequency-domain position occupied by each precoded symbol in `out`. This step is performed to apply the correct subband precoder when multiple PMI mode is used. Alternatively, you can provide the `CodebookIdx` parameter field. `CodebookIdx` is a scalar specifying the codebook index to use across the entire bandwidth. Therefore, the `CodebookIdx` field does not support subband precoding. The relationship between PMI values and codebook index is given in TS 36.213 [2], Section 7.2.4. | | | |

Data Types: `struct`

## Output Arguments

### `out` — Deprecoded downlink output

matrix

Deprecoded downlink output, returned as $N_{SYM}$-by-$v$ matrix, containing $v$ layers, with $N_{SYM}N_{SYM}$ symbols in each layer. The symbols for layers and antennas lie in columns rather than in rows.

Data Types: `double`

## Algorithms

For transmission schemes `'CDD'`, `'SpatialMux'`, and `'MultiUser'`, and degenerately `'Port0'`,

- Precoding involves multiplying a $P$-by-$v$ precoding matrix, $F$, by a $v$-by-$N_{SYM}$ matrix, representing $N_{SYM}$ symbols on each of $v$ transmission layers. This multiplication yields a $P$-by-$N_{SYM}$ matrix, representing $N_{SYM}$ precoded symbols on each of $P$ antenna ports. Depending on the transmission scheme, the precoding matrix can be composed of multiple matrices multiplied together. But the size of the product, $F$, is always $P$-by-$v$.

For the `'TxDiversity'` transmission scheme,

- A $P^2$-by-$2v$ precoding matrix, $F$, is multiplied by a $2v$-by-$N_{SYM}$ matrix, formed by splitting the real and imaginary components of a $v$-by-$N_{SYM}$ matrix of symbols on layers. This multiplication yields a $P^2$-by-$N_{SYM}$ matrix of precoded symbols, which is then reshaped into a $P$-by-$PN_{SYM}$ matrix for transmission. Since $v$ is $P$ for the `'TxDiversity'` transmission scheme, $F$ is of size $P^2$-by-$2P$, rather than $P^2$-by-$2v$.

When $v$ is $P$ in `'CDD'`, `'SpatialMux'`, and `'MultiUser'` transmission schemes, and when $P$ and $v$ are 2 in the `'TxDiversity'` transmission scheme,

- The precoding matrix, $F$, is square. Its size is $2P$-by-$2P$ for the transmit diversity scheme and $P$-by-$P$ otherwise. In this case, the deprecoder takes the matrix inversion of the precoding matrix to yield the deprecoding matrix $F^{-1}$. The matrix inversion is computed using LU decomposition with partial pivoting (row exchange):

  1. Perform LU decomposition $P_xF = LU$.
  2. Solve $LY = I$ using forward substitution.
  3. Solve $UX = Y$ using back substitution.
  4. $F^{-1} = XP_x$.

The degenerate case of the `'Port0'` transmission scheme falls into this category, with $P = v = 1$.

For the `'CDD'`, `'SpatialMux'`, and `'MultiUser'` transmission schemes,

- The deprecoding is then performed by multiplying $F^{-1}$ by the transpose of the input `symbols` (`symbols` is size $N_{SYM}$-by-$P$, so the transpose is a $P$-by-$N_{SYM}$ matrix). This multiplication recovers the $v$-by-$N_{SYM}$ (equals $P$-by-$N_{SYM}$) matrix of transmission layers.

For the `'TxDiversity'` transmission scheme,

- The deprecoding is performed, multiplying $F^{-1}$ by the transpose of the input `symbols` (`symbols` is size $PN_{SYM}$-by-$P$, so the transpose is a $P$-by-$PN_{SYM}$ matrix), having first been reshaped into a $2P$-by-$N_{SYM}$ matrix. This multiplication yields a $2v$-by-$N_{SYM}$, matrix which is then split into two $v$-by-$N_{SYM}$ matrices. To recover the $v$-by-$N_{SYM}$ matrix of transmission layers multiply the second matrix by $j$ and add the two matrices together (thus recombining real and imaginary parts).

For the other cases, specifically `'CDD'`, `'SpatialMux'`, and `'MultiUser'` transmission schemes with $v \neq P$ and the `'TxDiversity'` transmission scheme with $P = 4$,

- The precoding matrix $F$ is not square. Instead, the matrix is rectangular with size $P$-by-$v$, except in the case of `'TxDiversity'` transmission scheme with $P = 4$, where it is of size $P^2$-by-$(2P = 16)$-by-8. The number of rows is always greater than the number of columns in the matrix $F$ is size $m$-by-$n$ with $m > n$.
- In this case, the deprecoder takes the matrix pseudo-inversion of the precoding matrix to yield the deprecoding matrix $F^+$. The matrix pseudo-inversion is computed as follows.

  1. Perform LU decomposition $P_x F = LU$.
  2. Remove the last $m - n$ rows of $U$ to give $\bar{U}$.
  3. Remove the last $m - n$ columns of $L$ to give $\bar{L}$.
  4. $X = \bar{U}^H\left(\bar{U}\bar{U}^H\right)^{-1}\left(\bar{L}^H\bar{L}\right)^{-1}\bar{L}^H$ (the matrix inversions are carried out as in the previous steps).
  5. $F^+ = XP_x$

The application of the deprecoding matrix $F^+$ is the same process as described for deprecoding the square matrix case with $F^+$ in place of $F^{-1}$.

This method of pseudo-inversion is based on*Linear Algebra and Its Application* [3], Chapter 3.4, Equation (56).

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.213. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[3] Strang, Gilbert. *Linear Algebra and Its Application*. Academic Press, 1980. 2nd Edition.

## See Also

`lteDLPrecode` | `lteLayerDemap`

# lteDLFrameOffset

Downlink frame timing estimate

## Syntax

```
offset=lteDLFrameOffset(enb,waveform)
[offset,corr]=lteDLFrameOffset(enb,waveform)
[offset,corr]=lteDLFrameOffset(enb,waveform,corrcfg)
[offset,corr]=lteDLFrameOffset(enb,waveform,'TestEVM')
```

## Description

`offset=lteDLFrameOffset(enb,waveform)` returns the timing offset, in samples, between the start of the input `waveform` and the start of the first frame. `offset` is measured using the reference signals defined in the LTE standard.

`lteDLFrameOffset` performs synchronization using the PSS and SSS for the time-domain waveform, given cell-wide settings structure, `enb`. Note that this function does not perform PSS/SSS cell identity search. The cell identity must be provided in `enb`. The function `lteCellSearch` can be used to perform cell identity search.

`[offset,corr]=lteDLFrameOffset(enb,waveform)` also returns a complex matrix, `corr`, of the same dimensions as the input `waveform`.

`[offset,corr]=lteDLFrameOffset(enb,waveform,corrcfg)` provides control over which reference signals are used for timing estimation, as specified in the input structure, `corrcfg`.

`[offset,corr]=lteDLFrameOffset(enb,waveform,'TestEVM')`, provides the input `'TestEVM'` to stipulate alignment of the correlation configuration with TS 36.104, Annex E [1].

## Examples

### Synchronize and Demodulate Test Model Output

Synchronization and demodulation of Test Model output which has been delayed by five samples.

Initialize cell-wide parameters structure. Generate waveform for test model 1.1 with 5MHz bandwidth. A five sample delay is achieved by inserting five zeros at the beginning of the waveform. Compute and display the offset. Perform demodulation of the waveform accounting for the offset delay by adjusting waveform start index.

```
enb = lteTestModel('1.1','5MHz');
tx = [0; 0; 0; 0; 0; lteTestModelTool(enb)];

offset = lteDLFrameOffset(enb,tx)
```

```
offset = 5
```

```
rxGrid = lteOFDMDemodulate(enb,tx(1+offset:end));
```

## Input Arguments

### enb — Cell-wide settings

scalar structure

Cell-wide settings, specified as a structure. `enb` can contain these fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NDLRB`** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **`CyclicPrefix`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **`NCellID`** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **`DuplexMode`** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| The following parameters are only required for `CellRS = 'On'` or `'OmitEdgeRBs'`. See `corrcfg`. | | | |
| **`CellRefP`** | Required | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| The following parameters are only required when `DuplexMode = 'TDD'` and `CellRS = 'On'` or `'OmitEdgeRBs'`. See `corrcfg`. | | | |
| **`TDDConfig`** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **`SSC`** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |

Data Types: `struct`

### waveform — Time-domain waveform

numeric matrix

Time-domain waveform, specified as a *T*-by-*P* numeric matrix, where *T* is the number of time-domain samples and *P* is the number of receive antennas. `waveform` should be at least one subframe long and contain the PSS and SSS signals. Use `lteOFDMModulate` or one of the channel model functions (`lteFadingChannel`, `lteHSTChannel`, or `lteMovingChannel`) to generate this matrix.

Data Types: `double` | `single`

### corrcfg — Control reference signals used for timing estimation

scalar structure

Control reference signals used for timing estimation, specified as a structure containing any or all of these fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **PSS** | Optional | `'On'` (default), `'Off'` | Primary synchronization signal (PSS) correlation mode |
| **SSS** | Optional | `'On'` (default), `'Off'` | Secondary synchronization signal (SSS) correlation mode |
| **CellRS** | Optional | `'Off'` (default), `'OmitEdgeRBs'`, `'On'` | Cell-specific reference signal (CRS) correlation mode |

For the `corrcfg` fields, `lteDLFrameOffset` uses the reference signals, (`PSS`, `SSS`, or `CellRS`) as configured by initializing particular reference signal correlation mode(s) to `'On'`. For `CellRS`, using the mode setting, `'OmitEdgeRBs'`, instead of `'On'`, removes the uppermost and lowermost resource block of reference signals from the correlation. The `'OmitEdgeRBs'` method is specified for EVM testing in TS 36.104, Annex E [1]. Omitting band edge RBs removes potential transmit filtering nonlinear phase response and the resulting influence on group delay response for the overall band.

Data Types: `struct`

**`'TestEVM'` — Test EVM setting**
`'TestEVM'`

Test EVM setting, specified as `'TestEVM'`. As defined in TS 36.104 [1], Annex E, sets correlation with:

- `PSS` to `'On'`,
- `SSS` to `'Off'`, and
- `CellRS` to `'OmitEdgeRBs'`.

Data Types: `char` | `string`

## Output Arguments

**`offset` — Timing offset from the start of the input waveform to the start of the first frame**
numeric scalar

Timing offset from the start of the input waveform to the start of the first frame, returned as a numeric scalar. It indicates the number of samples from the start of `waveform`, to the position in `waveform` where the first frame begins. `offset` is computed by extracting the timing of the peak of the correlation between `waveform` and the internally generated time-domain reference waveforms containing PSS and SSS signals. The correlation is performed separately for each antenna. `lteDLFrameOffset` uses the antenna with the earliest correlation peak and a correlation peak magnitude at least 50% of the maximum across the antennas to compute `offset`.

Data Types: `double`

**`corr` — Signal used to extract timing offset**
complex numeric matrix

Signal used to extract the timing offset, returned as a complex numeric matrix of the same size as `waveform`. Each column of `corr` is the correlation for each column (antenna) of `waveform`.

Data Types: `double`
Complex Number Support: Yes

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.104. “Base Station (BS) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*.

## See Also

`lteNBDLFrameOffset` | `lteFadingChannel` | `lteMovingChannel` | `lteHSTChannel` | `lteOFDMDemodulate` | `lteCellSearch` | `lteFrequencyCorrect` | `lteFrequencyOffset`

# lteDLPerfectChannelEstimate

Downlink perfect channel estimation

## Syntax

```
hest = lteDLPerfectChannelEstimate(enb,propchan)
hest = lteDLPerfectChannelEstimate(enb,propchan,timefreqoffset)
hest = lteDLPerfectChannelEstimate(enb,propchan,timefreqoffset,ntxants)
```

## Description

`hest = lteDLPerfectChannelEstimate(enb,propchan)` performs perfect channel estimation for a system configuration given structures containing the cell-wide settings, and the propagation channel configuration. The perfect channel estimates are only produced for channel models created using `lteFadingChannel` or `lteHSTChannel`.

This function provides a perfect MIMO channel estimate after OFDM modulation. Perfect channel estimation is achieved by setting the channel with the desired configuration and sending a set of known symbols through it for each transmit antenna in turn.

`hest = lteDLPerfectChannelEstimate(enb,propchan,timefreqoffset)` specifies the timing and frequency offsets. This parameter allows `hest` to be the precise channel that results when the receiver is precisely synchronized.

`hest = lteDLPerfectChannelEstimate(enb,propchan,timefreqoffset,ntxants)` specifies the number of transmit antenna planes.

---

**Note** This syntax is provided to allow modeling of greater than four transmit antenna planes. For this syntax, the `enb.CellRefP` field, is not required and, if included, is not used to define the number of antenna planes.

---

## Examples

### Perform Perfect DL Channel Estimation

Perform perfect channel estimation for a given propagation channel configuration in the downlink.

Initialize eNodeB and propagation channel configuration structures.

```
enb.NDLRB = 6;
enb.CyclicPrefix = 'Normal';
enb.CellRefP = 4;
enb.TotSubframes = 1;

chs.Seed = 1;
chs.DelayProfile = 'EPA';
chs.NRxAnts = 2;
chs.DopplerFreq = 5.0;
```

```
chs.MIMOCorrelation = 'Low';
chs.InitPhase = 'Random';
chs.InitTime = 0.0;
chs.ModelType = 'GMEDS';
chs.NTerms = 16;
chs.NormalizeTxAnts = 'On';
chs.NormalizePathGains = 'On';
```

Compute the downlink channel estimate and display the dimension of the output channel estimate.

```
H = lteDLPerfectChannelEstimate(enb,chs);
sizeH = size(H)
```

```
sizeH = 1×4

    72    14     2     4
```

### Perfect DL Channel Estimation on a Time Offset Waveform

Perform perfect channel estimation on a time offset waveform that has passed through a fading channel.

#### Configuration initialization

- Initialize cell-wide configuration to R.12 (TxDiversity, 6 RB, CellRefP=4, normal cyclic prefix).
- Initialize propagation channel configuration.

```
enb = lteRMCDL('R.1','FDD',1);
enb.TotSubframes = 1;

chan.Seed = 1;
chan.DelayProfile = 'EPA';
chan.NRxAnts = 1;
chan.DopplerFreq = 5.0;
chan.MIMOCorrelation = 'Low';
chan.InitPhase = 'Random';
chan.InitTime = 0.0;
chan.ModelType = 'GMEDS';
chan.NTerms = 16;
chan.NormalizeTxAnts = 'On';
chan.NormalizePathGains = 'On';
```

#### Waveform processing

- Create waveform and add samples for channel delay.
- Pass through a fading channel, generating time-domain receiver samples.

```
[txwave,txgrid,rmcCfg] = lteRMCDLTool(enb,[1;0;0;1]);
txwave = [txwave; zeros(25,enb.CellRefP)];
chan.SamplingRate = rmcCfg.SamplingRate;
rxwave = lteFadingChannel(chan,txwave);
```

### Determine timing offset

- Use `lteDLFrameOffset` to estimate time offset.
- Account for the timing offset in the received waveform.

```
toffset = lteDLFrameOffset(enb,rxwave)
```

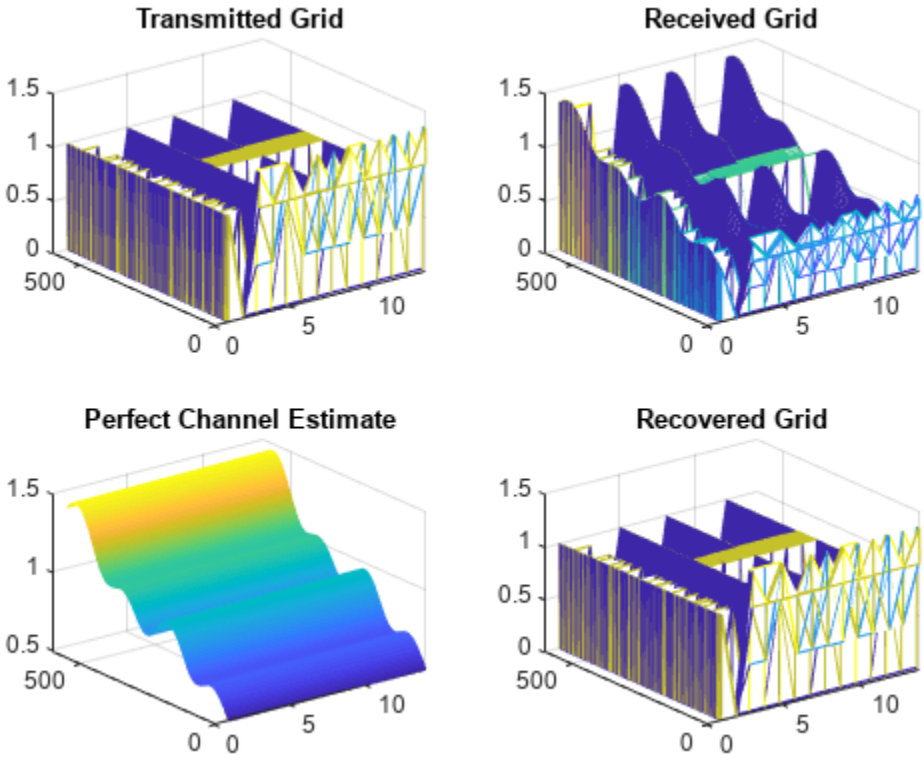
```
toffset = 7
```

```
rxwave = rxwave(1+toffset:end,:);
```

### Demodulation and perfect channel estimation

- Demodulate `rxwave` to generate frequency-domain receiver data in `rxgrid`.
- Equalize with perfect channel estimate using time offset.
- Plot resource element grids to show impact of fading channel on the transmitted signal and recovery of the signal using the perfect channel estimate.

```
rxgrid = lteOFDMDemodulate(enb,rxwave);
hest = lteDLPerfectChannelEstimate(enb,chan,[toffset,0]);
sizeH = size(hest);
recoveredgrid = rxgrid./hest;

subplot(2,2,1)
mesh(abs(txgrid(:,:,1,1)))
title('Transmitted Grid');
subplot(2,2,2)
mesh(abs(rxgrid(:,:,1,1)))
title('Received Grid');
subplot(2,2,3)
mesh(abs(hest(:,:,1,1)))
title('Perfect Channel Estimate');
subplot(2,2,4)
mesh(abs(recoveredgrid(:,:,1,1)))
title('Recovered Grid');
```




Comparing the transmitted grid to the recovered grid shows equalization of the received grid with the perfect channel estimate recovers the transmission.

### Perform HST Model Perfect DL Channel Estimation

Perform perfect channel estimation for a high speed train (HST) propagation channel configuration in the downlink. Include time and frequency offsets in the channel estimation computation.

#### Configuration initialization

Initialize configuration structures for eNodeB and HST propagation channel.

```
enb.NDLRB = 6;
enb.NCellID = 1;
enb.CyclicPrefix = 'Normal';
enb.CellRefP = 1;
enb.TotSubframes = 1;

hst.NRxAnts = 2;
hst.Ds = 100;
hst.Dmin = 500;
hst.Velocity = 200;
hst.DopplerFreq = 5.0;
hst.InitTime = 0.0;
hst.ModelType = 'GMEDS';
hst.NormalizeTxAnts = 'On';
```

### Waveform processing

- Create waveform and add samples for channel delay.
- Pass through an HST channel, generating time-domain receiver samples.

```
[txwave,txgrid,rmcCfg] = lteRMCDLTool(enb,[1;0;0;1]);
txwave = [txwave; zeros(25,enb.CellRefP)];
hst.SamplingRate = rmcCfg.SamplingRate;
rxwave = lteHSTChannel(hst,txwave);
```

### Determine timing and frequency offsets

- Use `lteDLFrameOffset` to estimate time offset.
- Account for the timing offset in the received waveform.
- Use `lteFrequencyOffset` to estimate frequency offset.

```
toffset = lteDLFrameOffset(enb,rxwave)
```

```
toffset = 7
```

```
rxwave = rxwave(1+toffset:end,:);
foffset = lteFrequencyOffset(enb,rxwave)
```

```
foffset = 0.4953
```

### Demodulation and perfect channel estimation

- Demodulate `rxwave` to generate frequency-domain receiver data in `rxgrid`.
- Equalize with perfect channel estimate using time and frequency offsets.

```
rxgrid = lteOFDMDemodulate(enb,rxwave);
hest = lteDLPerfectChannelEstimate(enb,hst,[toffset,foffset]);
sizeH = size(hest)
```

```
sizeH = 1×3

    72    14     2
```

```
recoveredgrid = rxgrid./hest;
```

### Perform Perfect DL Channel Estimation for Eight Antenna Planes

Perform perfect channel estimation for eight transmit antenna planes for a given propagation channel configuration in the downlink.

Initialize eNodeB and propagation channel configuration structures. Define a local variable for the number of transmit antenna planes.

```
enb.NDLRB = 6;
enb.CyclicPrefix = 'Normal';
enb.TotSubframes = 1;

chs.Seed = 1;
chs.DelayProfile = 'EPA';
```

```
chs.NRxAnts = 2;
chs.DopplerFreq = 5.0;
chs.MIMOCorrelation = 'Low';
chs.InitPhase = 'Random';
chs.InitTime = 0.0;
chs.ModelType = 'GMEDS';
chs.NTerms = 16;
chs.NormalizeTxAnts = 'On';
chs.NormalizePathGains = 'On';

txAntPlanes = 8;
```

Compute the downlink channel estimate and display the dimension of the output channel estimate.

```
chest = lteDLPerfectChannelEstimate(enb,chs,[0 0],txAntPlanes);
sizeH = size(chest)
```

```
sizeH = 1×4

    72    14     2     8
```

The dimensionality of `chest` indicates two receive and eight transmit antenna planes are included in the channel estimate.

## Input Arguments

### enb — Cell-wide settings

scalar structure

Cell-wide settings, specified as a structure with the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NDLRB`** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **`CyclicPrefix`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **`CellRefP`** | Required | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **`TotSubframes`** | Optional | Nonnegative scalar integer<br><br>1 (default) | Total number of subframes to generate |

Data Types: `struct`

### propchan — Propagation channel configuration

structure

Propagation channel configuration, specified as a structure that can contain these parameter fields. `propchan` must contain the fields required to parameterize the channel model for a fading channel (`lteFadingChannel`) or a high-speed train channel (`lteHSTChannel`).

**Note** Before execution of the channel itself, `lteDLPerfectChannelEstimate` sets `SamplingRate` internally to the sampling rate of the time domain waveform passed to `lteFadingChannel` or `lteHSTChannel` for channel filtering. Therefore, the `propchan` structure does not require the `SamplingRate` field. If one is included, it is not used.

`propchan` structure fields to be included for fading channel model case:

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NRxAnts`** | Required | Positive scalar integer | Number of receive antennas |
| **`MIMOCorrelation`** | Required | `'Low'`, `'Medium'`, `'UplinkMedium'`, `'High'`, `'Custom'` | Correlation between UE and eNodeB antennas<br>• `'Low'` correlation is equivalent to no correlation between antennas.<br>• `'Medium'` correlation level is applicable to tests defined in TS 36.101 [1].<br>• `'UplinkMedium'` correlation level is applicable to tests defined in TS 36.104 [2]. |
| **`NormalizeTxAnts`** | Optional | `'On'` (default), `'Off'` | Transmit antenna number normalization.<br>• `'On'`, this function normalizes the model output by `1/sqrt(`$N_{TX}$`)`, where $N_{TX}$ is the number of transmit antennas. Normalization by the number of transmit antennas ensures that the output power per receive antenna is unaffected by the number of transmit antennas.<br>• `'Off'`, normalization is not performed. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **DelayProfile** | Required | `'EPA'`, `'EVA'`, `'ETU'`, `'Custom'`, `'Off'` | Delay profile model. For more information, see “Propagation Channel Models”.<br><br>Setting `DelayProfile` to `'Off'` switches off fading completely and implements a static MIMO channel model. In this case, the antenna geometry corresponds to `propchan.MIMOCorrelation`, `propchan.NRxAnts`, and the number of transmit antennas. The temporal part of the model for each link between transmit and receive antennas consists of a single path with zero delay and constant unit gain. |
| The following fields are applicable when `DelayProfile` is set to a value other than `'Off'`. | | | |
| **DopplerFreq** | Required | Scalar | Maximum Doppler frequency, in Hz. |
| **InitTime** | Required | Scalar | Fading process time offset, in seconds. |
| **NTerms** | Optional | 16 (default)<br><br>scalar power of 2 | Number of oscillators used in fading path modeling. |
| **ModelType** | Optional | `'GMEDS'` (default), `'Dent'` | Rayleigh fading model type.<br><br>• `'GMEDS'`, the Rayleigh fading is modeled using the Generalized Method of Exact Doppler Spread (GMEDS), as described in [4].<br>• `'Dent'`, the Rayleigh fading is modeled using the modified Jakes fading model described in [3].<br><br>**Note** `ModelType = 'Dent'` is not recommended. Use `ModelType = 'GMEDS'` instead. |
| **NormalizePathGains** | Optional | `'On'` (default), `'Off'` | Model output normalization.<br><br>• `'On'`, the model output is normalized such that the average power is unity.<br>• `'Off'`, the average output power is the sum of the powers of the taps of the delay profile. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **InitPhase** | Optional | 'Random' (default), scalar (in Radians), or $N$-by-$L$-by-$N_{TX}$-by-$N_{RX}$ array | Phase initialization for the sinusoidal components of the model.<br>• 'Random', sets the phases randomly initialized according to Seed.<br>• A scalar, assumed to be in radians, is used to initialize the phases of all components.<br>• An $N$-by-$L$-by-$N_{TX}$-by-$N_{RX}$ array is used to explicitly initialize the phase in radians of each component. In this case, $N$ is the number of phase initialization values per path, $L$ is the number of paths, $N_{TX}$ is the number of transmit antennas, and $N_{RX}$ is the number of receive antennas. (NRxAnts)<br>**Note**<br>• When ModelType is set to 'GMEDS', $N = 2 \times$ NTerms.<br>• When ModelType is set to 'Dent', $N =$ NTerms. |
| The following field is applicable when DelayProfile is set to a value other than 'Off' and InitPhase is set to 'Random'. | | | |
| **Seed** | Required | Scalar | Random number generator seed. To use a random seed, set Seed to zero.<br>**Note** MathWorks® recommends using Seed values from 0 to $2^{31} - 1 - (K(K - 1)/2)$, where $K = N_{TX} \times N_{RX}$, the product of the number of transmit and receive antennas. Seed values outside of this range are not guaranteed to give distinct results. |
| The following fields are applicable when DelayProfile is set to 'Custom'. | | | |
| **AveragePathGaindB** | Required | Vector | Average gains of the discrete paths, expressed in dB. |
| **PathDelays** | Required | Vector | Delays of the discrete paths, expressed in seconds. This vector must have the same size as AveragePathGaindB. |
| The following fields are applicable when MIMOCorrelation is set to 'Custom'. | | | |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **TxCorrelationMatrix** | Required | Matrix | Correlation between each of the transmit antennas, specified as a $N_{TX}$-by-$N_{TX}$ complex matrix. |
| **RxCorrelationMatrix** | Required | Matrix | Correlation between each of the receive antennas, specified as a complex matrix of size $N_{RX}$-by-$N_{RX}$. |

`propchan` structure fields to be included for the high-speed train channel model case:

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NRxAnts** | Required | Positive scalar integer | Number of receive antennas |
| **Ds** | Required | Scalar | Train-to-eNodeB double initial distance, in meters.<br>`Ds`/2 is initial distance between train and eNodeB, in meters |
| **Dmin** | Required | Scalar | eNodeB to railway track distance, in meters |
| **Velocity** | Required | Scalar | Train velocity, in kilometers per hour |
| **DopplerFreq** | Required | Scalar | Maximum Doppler frequency, in Hz. |
| **InitTime** | Required | Scalar | Doppler shift timing offset, in seconds |
| **NormalizeTxAnts** | Optional | `'On'` (default), `'Off'` | Transmit antenna number normalization.<br>• `'On'`, `lteHSTChannel` normalizes the model output by `1/sqrt(`$N_{TX}$`)`, where $N_{TX}$ is the number of transmit antennas. Normalization by the number of transmit antennas ensures that the output power per receive antenna is unaffected by the number of transmit antennas.<br>• `'Off'`, normalization is not performed. |

Data Types: `struct`

**`timefreqoffset` — Timing and frequency offset**
[0, 0] (default) | two element row vector, [`toffset`, `foffset`] | nonnegative scalar, `toffset` | optional

Timing and frequency offset, specified as a nonnegative scalar providing `toffset` or two element row vector providing [`toffset`, `foffset`].

**`toffset` — Timing offset**
0 (default) | nonnegative scalar | optional

Timing offset in samples from the start of the output of the channel to the OFDM demodulation starting point, specified as a nonnegative scalar. The timing offset accounts for delay introduced during propagation, which is useful to obtain the perfect estimate of the channel seen by a synchronized receiver. Use `lteDLFrameOffset` to derive `toffset`.

**`foffset` — Frequency offset**
0 (default) | scalar | optional

Frequency offset in Hertz of the time-domain waveform, specified as a scalar. Use `lteFrequencyOffset` to derive `foffset`.

Example: `[3 100]` indicates a time offset of three samples and a frequency offset of 100 Hz.

Data Types: `double`

**`ntxants` — Number of transmit antenna planes**
1 (default) | nonnegative integer | optional

Number of transmit antenna planes, specified as a nonnegative integer.

## Output Arguments

**`hest` — Perfect channel estimate**
4-D array

Perfect channel estimate, returned as an $N_{SC}$-by-$N_{SYM}$-by-$N_{RX}$-by-$N_{TX}$ array.

- $N_{SC}$ is the number of subcarriers.
- $N_{SYM}$ is the number of OFDM symbols.
- $N_{RX}$ is the number of receive antennas as specified by `propchan.NRxAnts`.
- $N_{TX}$ is the number of transmit antenna planes, specified either by the input `ntxants` or by `enb.CellRefP`. If `ntxants` is provided as an input, the `enb.CellRefP` field is not required and, if included, is not used.

Data Types: `double`

# Version History

**Introduced in R2013b**

## References

[1] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.104. “Evolved Universal Terrestrial Radio Access (E-UTRA); Base Station (BS) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[3] Dent, P., G. E. Bottomley, and T. Croft. “Jakes Fading Model Revisited.” *Electronics Letters*. Vol. 29, 1993, Number 13, pp. 1162–1163.

[4] Pätzold, Matthias, Cheng-Xiang Wang, and Bjørn Olav Hogstad. “Two New Sum-of-Sinusoids-Based Methods for the Efficient Generation of Multiple Uncorrelated Rayleigh Fading Waveforms.” *IEEE Transactions on Wireless Communications*. Vol. 8, 2009, Number 6, pp. 3122–3131.

## See Also

`lteDLChannelEstimate` | `lteOFDMDemodulate` | `lteEqualizeMMSE` | `lteEqualizeZF` | `lteFadingChannel` | `lteULPerfectChannelEstimate`

# lteDLPrecode

Downlink precoding of transmission layers

## Syntax

```
out = lteDLPrecode(in,ntxants,txscheme,codebook)
out = lteDLPrecode(enb,chs,in)
```

## Description

`out = lteDLPrecode(in,ntxants,txscheme,codebook)` performs precoding according to TS 36.211 [1], Section 6.3.4. The `out` matrix returned is identical to the matrix returned by `ltePDSCH` for the same set of parameters. The overall operation of the precoder is the transpose of the matrix defined in the specification. The symbols for layers and antennas lie in columns rather than rows.

This function performs precoding of the matrix of layers, `in`, onto *P* antennas, using the transmission scheme specified by `txscheme`. For transmission scheme precoding dependencies, see “Algorithms” on page 2-191.

`out = lteDLPrecode(enb,chs,in)` precodes the matrix of layers, `in`, according to cell-wide settings `enb` and channel transmission configurations `chs`.

## Examples

### Perform Downlink Precoding on Identity Matrix

Perform downlink precoding using an identity matrix as input.

By precoding an identity matrix, you can gain access to the precoding matrices. Obtain the precoding matrix having codebook index 1 for three layers and four antennas.

```
out = lteDLPrecode(eye(3),4,'SpatialMux',1).'
```

```
out = 4×3 complex

   0.2887 + 0.0000i   0.0000 - 0.2887i  -0.2887 + 0.0000i
   0.0000 + 0.2887i   0.2887 + 0.0000i   0.0000 + 0.2887i
  -0.2887 + 0.0000i   0.0000 - 0.2887i   0.2887 + 0.0000i
   0.0000 - 0.2887i   0.2887 + 0.0000i   0.0000 - 0.2887i
```

## Input Arguments

### `in` — Input layers

matrix

Input layers, specified as an $N_{SYM}$-by-$\nu$ matrix, consisting of the $N_{SYM}$ modulation symbols for transmission on $\nu$ layers. Generate this matrix using `lteLayerMap`.

Data Types: `double`
Complex Number Support: Yes

**`ntxants` — Number of antennas**
positive integer

Number of antennas, specified as a positive integer.

Data Types: `double`

**`txscheme` — Transmission scheme**
`'Port0'` | `'TxDiversity'` | `'CDD'` | `'SpatialMux'` | `'MultiUser'` | `'Port5'` | `'Port7-8'` | `'Port8'` | `'Port7-14'`

PDSCH transmission scheme, specified as one of the following options.

| Transmission scheme | Description |
|---|---|
| `'Port0'` | Single antenna port, port 0 |
| `'TxDiversity'` | Transmit diversity |
| `'CDD'` | Large delay cyclic delay diversity scheme |
| `'SpatialMux'` | Closed loop spatial multiplexing |
| `'MultiUser'` | Multi-user MIMO |
| `'Port5'` | Single-antenna port, port 5 |
| `'Port7-8'` | Single-antenna port, port 7, when `NLayers` = 1. Dual layer transmission, ports 7 and 8, when `NLayers` = 2. |
| `'Port8'` | Single-antenna port, port 8 |
| `'Port7-14'` | Up to eight layer transmission, ports 7–14 |

Data Types: `char` | `string`

**`codebook` — Codebook index**
integer from 0 to 15

Codebook index to select the precoding matrix, specified as an integer from 0 to 15. This input is ignored for the `'Port0'`, `'TxDiversity'`, and `'CDD'` transmission schemes. Find the precoding matrix corresponding to a particular codebook index in the TS 36.211 [1], Section 6.3.4. Since codebook is a scalar, the syntax that includes this parameter does not support subband precoding or multiple PMI mode. In the case of `'TxDiversity'` and `P=1`, the function falls back to single port processing.

Data Types: `double`

**`enb` — eNodeB cell-wide settings**
structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields:

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| When `chs.TxScheme` is set to `'TxDiversity'`, `'CDD'`, `'SpatialMux'`, or `'MultiUser'`, these parameters are applicable: | | | |
| **`CellRefP`** | Required | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| When `chs.TxScheme` is set to `'SpatialMux'`, or `'MultiUser'` and `chs.PMISet` is present, these parameters are applicable:. | | | |
| **`NCellID`** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **`NSubframe`** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **`NDLRB`** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **`CFI`** | Required | 1, 2, or 3<br>Scalar or if the CFI varies per subframe, a vector of length 10 (corresponding to a frame). | Control format indicator (`CFI`) value. In TDD mode, `CFI` varies per subframe for the RMCs (`'R.0'`, `'R.5'`, `'R.6'`, `'R.6-27RB'`, `'R.12-9RB'`) |
| **`CyclicPrefix`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **`DuplexMode`** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| When `DuplexMode` is set to `'TDD'`, these parameters are applicable: | | | |
| **`TDDConfig`** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **`SSC`** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |

Data Types: `struct`

### `chs` — Channel-specific transmission configuration

structure | structure array

Channel specific transmission configuration, specified as a structure that can contain the following parameter fields.

<table>
<tr><th>Parameter Field</th><th>Required or Optional</th><th>Values</th><th>Description</th></tr>
<tr><td>TxScheme</td><td>Optional</td><td>'Port0' (default), 'TxDiversity', 'CDD', 'SpatialMux', 'MultiUser', 'Port7-8', 'P… 'Port7-14'.</td><td>PDSCH transmission scheme, specified as one of the following options.
<table>
<tr><th>Transmission scheme</th><th>Description</th></tr>
<tr><td>'Port0'</td><td>Single antenna port, port 0</td></tr>
<tr><td>'TxDiversity'</td><td>Transmit diversity</td></tr>
<tr><td>'CDD'</td><td>Large delay cyclic delay diversity scheme</td></tr>
<tr><td>'SpatialMux'</td><td>Closed loop spatial multiplexing</td></tr>
<tr><td>'MultiUser'</td><td>Multi-user MIMO</td></tr>
<tr><td>'Port5'</td><td>Single-antenna port, port 5</td></tr>
<tr><td>'Port7-8'</td><td>Single-antenna port, port 7, when NLayers = 1. Dual layer transmission, ports 7 and 8, when NLayers = 2.</td></tr>
<tr><td>'Port8'</td><td>Single-antenna port, port 8</td></tr>
<tr><td>'Port7-14'</td><td>Up to eight layer transmission, ports 7–14</td></tr>
</table></td></tr>
<tr><td colspan="4">When chs.TxScheme is set to 'SpatialMux' or 'MultiUser', these parameters are applicable, include either Codebookidx or both PMISet and PRBSet:</td></tr>
<tr><td>Codebookidx</td><td>Optional</td><td>Integer from 0 to 15</td><td>Codebook index used during precoding</td></tr>
<tr><td>PMISet</td><td>Optional</td><td>Integer vector with element values from 0 to 15.</td><td>Precoder matrix indication (PMI) set. It can contain either a single value, corresponding to single PMI mode, or multiple values, corresponding to multiple or subband PMI mode. The number of values depends on CellRefP, transmission layers and TxScheme. For more information about setting PMI parameters, see ltePMIInfo.</td></tr>
</table>

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **PRBSet** | Optional | Integer column vector or two-column matrix | Zero-based physical resource block (PRB) indices corresponding to the slot wise resource allocations for this PDSCH. `PRBSet` can be assigned as:<br>• a column vector, the resource allocation is the same in both slots of the subframe,<br>• a two-column matrix, this parameter specifies different PRBs for each slot in a subframe,<br>• a cell array of length 10 (corresponding to a frame, if the allocated physical resource blocks vary across subframes).<br>PRBSet varies per subframe for the RMCs `'R.25'`(TDD), `'R.26'`(TDD), `'R.27'`(TDD), `'R.43'`(FDD), `'R.44'`, `'R.45'`, `'R.48'`, `'R.50'`, and `'R.51'`. |
| The fields `PMISet` and `PRBSet` determine the frequency-domain position that each precoded symbol in `out` occupies to apply the correct subband precoder when multiple PMI mode is being used. Alternatively, you can provide `CodebookIdx` field. `CodebookIdx` is a scalar specifying the codebook index to use across the entire bandwidth. Therefore, the `CodebookIdx` field does not support subband precoding. TS 36.213 [2], Section 7.2.4 specifies the relationship between PMI values and codebook indices. | | | |

Data Types: `struct`

## Output Arguments

### `out` — Precoded downlink output

matrix

Precoded downlink output, returned as an $N_{SYM}$-by-$P$ matrix. $N_{SYM}$ is the number of symbols per antenna, and $P$ is the number of transmission antennas. The symbols for layers and antennas lie in columns rather than rows.

Data Types: `double`

## Algorithms

For transmission schemes `'CDD'`, `'SpatialMux'`, and `'MultiUser'`, and degenerately `'Port0'`,

- Precoding involves multiplying a $P$-by-$v$ precoding matrix, denoted as $F$, by a $v$-by-$N_{SYM}$ matrix, representing $N_{SYM}$ symbols on each of $v$ transmission layers, to yield a $P$-by-$N_{SYM}$ matrix, consisting of $N_{SYM}$ precoded symbols on each of $P$ antenna ports. Depending on the transmission scheme, the precoding matrix can be composed of multiple matrices multiplied together, but the size of the product, $F$, is always $P$-by-$v$.

For the `'TxDiversity'` transmission scheme,

- A $P^2$-by-$2v$ precoding matrix, $F$, is multiplied by a $2v$-by-$N_{SYM}$ matrix, formed by splitting the real and imaginary components of a $v$-by-$N_{SYM}$ matrix of symbols on layers, to yield a $P^2$-by-$N_{SYM}$ matrix of precoded symbols, which is then reshaped into a $P$-by-$PN_{SYM}$ matrix for transmission. As $v = P$ for the `'TxDiversity'` transmission scheme, we can consider $F$ be of size $P^2$-by-$2P$ rather than $P^2$-by-$2v$.

For the other cases, specifically `'CDD'`, `'SpatialMux'`, and `'MultiUser'` transmission schemes with $v \neq P$, and the `'TxDiversity'` transmission scheme with $P = 4$,

- The precoding matrix $F$ is not square; it is rectangular with size $P$-by-$v$ except for the `'TxDiversity'` transmission scheme with $P = 4$ where it is of size $P^2$-by-($2P$= 16)-by-8. The number of rows is always greater than the number of columns that is, the matrix $F$ is size $m$-by-$n$ with $m$-by-$n$.

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.213. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteDLDeprecode` | `lteLayerMap`

# lteDLResourceGrid

Downlink subframe resource array

## Syntax

```
grid = lteDLResourceGrid(enb)
grid = lteDLResourceGrid(enb,p)
```

## Description

`grid = lteDLResourceGrid(enb)` returns an empty resource array generated from the cell-wide-specific settings structure `enb`. For more information on the resource grid and the multidimensional array used to represent the resource elements for one subframe across all configured antenna ports, see “Represent Resource Grids”.

`grid = lteDLResourceGrid(enb,p)` accepts an additional input, `p`, which directly specifies the number of antenna planes in the array. In this syntax, `CellRefP` is not required as a structure field of `enb`.

## Examples

### Create Empty Resource Array

Create an empty resource array representing the resource elements for 10MHz bandwidth, one subframe, and two antennas.

```
rgrid = lteDLResourceGrid(struct('NDLRB',50,'CellRefP',2));
size(rgrid)
```

```
ans = 1×3

   600    14     2
```

### Create DL Subframe Resource Array Using Optional Antenna Plane Input

Create an empty resource array that represents the downlink resource elements for 5 MHz bandwidth, one subframe, extended cyclic prefix, and four antenna ports.

```
cfg = struct('NDLRB',25,'CyclicPrefix','Extended');
p = 4;
griddl = lteDLResourceGrid(cfg,p);
size(griddl)
```

```
ans = 1×3

   300    12     4
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure having the following fields.

### NDLRB — Number of downlink resource blocks

scalar integer from 6 to 110

Number of downlink resource blocks, specified as a scalar integer from 6 to 110.

Data Types: `double`

### CyclicPrefix — Cyclic prefix length

`'Normal'` (default) | optional | `'Extended'`

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

### CellRefP — Number of cell-specific reference signal antenna ports

1 | 2 | 4

Number of cell-specific reference signal antenna ports, specified as 1, 2, or 4.

Data Types: `double`

Data Types: `struct`

### p — Number of antenna planes

positive scalar integer

Number of antenna planes, specified as a positive scalar integer.

Data Types: `double`

## Output Arguments

### grid — Empty downlink resource grid

3-D numeric array

Empty downlink resource grid, returned as a 3-D numeric array. This array is used to represent the resource elements for one subframe across all configured antenna ports. It has dimensions of:

- When the function has a single input argument, `enb`, an *N*-by-*M*-by-`CellRefP` array is returned. *N* is the number of subcarriers (12×`NDLRB`). *M* is the number of OFDM symbols in a subframe (14 for normal cyclic prefix and 12 for extended cyclic prefix). `CellRefP` is the number of transmit antenna ports.
- When the function has two input arguments, `enb` and `p`, an *N*-by-*M*-by-`p` array is returned. `p` is the number of antenna planes.

Data Types: `double`

# Version History

**Introduced in R2014a**

## See Also

`lteDLResourceGridSize` | `lteResourceGrid` | `lteResourceGridSize` | `lteULResourceGrid` | `lteULResourceGridSize` | `lteOFDMModulate`

# lteDLResourceGridSize

Size of downlink subframe resource array

## Syntax

```
d = lteDLResourceGridSize(enb)
d = lteDLResourceGridSize(enb,p)
```

## Description

`d = lteDLResourceGridSize(enb)` returns a three-element row vector of dimension lengths for the resource array generated from the cell-wide settings structure `enb`. For more information on the resource grid and the multidimensional array used to represent the resource elements for one subframe across all configured antenna ports, see “Represent Resource Grids”.

`d = lteDLResourceGridSize(enb,p)` returns a three-element row vector, where `p` directly specifies the number of antenna planes in the array. In this syntax, `CellRefP` is not required as a structure field of `enb`.

## Examples

### Determine Downlink Subframe Resource Array Size

Determine the size of a downlink subframe resource array.

Determine the dimensions of a downlink subframe resource array, using cell-wide settings, `enb`. Then, use the returned vector directly to create a resource grid as a multidimensional array.

```
enb = struct('NDLRB',50,'CellRefP',2,'CyclicPrefix','Normal');
rgrid = zeros(lteDLResourceGridSize(enb));
size(rgrid)
```

```
ans = 1×3

   600    14     2
```

The same result can be obtained by calling the `lteDLResourceGrid` function.

### Get Downlink Subframe Resource Array Size Using Optional Antenna Plane Input

Get the downlink subframe resource array size from an downlink configuration structure using the antenna plane input. Then, use the returned vector to directly create a MATLAB™ array.

```
cfgdl = struct('NDLRB',50,'CyclicPrefix','Normal');
p = 2;
griddl = zeros(lteDLResourceGridSize(cfgdl,p));
size(griddl)
```

```
ans = 1×3

   600    14     2
```

The output grid, `griddl`, is a resource array. This resource array size could be obtained in a similar manner using the `lteResourceGridSize` function.

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure having the following fields.

### NDLRB — Number of downlink resource blocks

scalar integer from 6 to 110

Number of downlink resource blocks, specified as a scalar integer from 6 to 110. Standard bandwidth values are 6, 15, 25, 50, 75, and 100.

Data Types: `double`

### CellRefP — Number of cell-specific reference signal antenna ports

1 | 2 | 4

Number of cell-specific reference signal antenna ports, specified as 1, 2, or 4.

Data Types: `double`

### CyclicPrefix — Cyclic prefix length

`'Normal'` (default) | optional | `'Extended'`

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char`

Data Types: `struct`

### p — Number of antenna planes

positive scalar integer

Number of antenna planes, specified as a positive scalar integer. This argument directly specifies the number of antenna planes in the array.

Data Types: `double`

## Output Arguments

### d — Downlink resource grid dimensions

numeric 1-by-3 row vector

Downlink resource grid dimensions, returned as a numeric 1-by-3 row vector. When the function has a single argument, `d` is `[N M CellRefP]`. `N` is the number of subcarriers (12×`NDLRB`). `M` is the number of OFDM symbols in a subframe, 14 for normal cyclic prefix and 12 for extended cyclic prefix.

`CellRefP` is the number of transmit antenna ports. When the number of antenna planes, `p`, is specified as the second input argument, then `d` is `[N M p]` and the input field `CellRefP` of `enb` is not required.

Data Types: `double`

## Version History

**Introduced in R2014a**

## See Also

`lteDLResourceGrid` | `lteResourceGridSize` | `lteULResourceGridSize`

# lteDLSCH

Downlink shared channel

## Syntax

```
[cwout,chinfo] = lteDLSCH(enb,chs,outlen,trblkin)
```

## Description

`[cwout,chinfo] = lteDLSCH(enb,chs,outlen,trblkin)` applies the complete DL-SCH transport channel coding chain to the input data, `trblkin`, and returns the codewords in `cwout`. The encoding process includes type-24A CRC calculation, code block segmentation and type-24B CRC attachment, if any, turbo encoding, rate matching with RV, and code block concatenation. Additional information about the encoding process is returned in the fields of structure `chinfo`. For the case of spatial multiplexing schemes transmitting two codewords, `lteDLSCH` processes a single transport block or pairs of blocks, contained in a cell array. The data type for `cwout` matches the input, `trblkin`. Thus, if `trblkin` is a cell array containing one or two transport blocks, `cwout` is a cell array of one or two codewords. If `trblkin` is a vector of information bits, `cwout` is a vector also. Define pairs of modulation schemes and RV indicators in the appropriate parameter fields to encode a pair of transport blocks.

## Examples

### Generate DL-SCH Codewords

Generate the DL-SCH codeword as defined by TS36.101 RMC R.7 for FDD duplexing mode

Initialize the `rmc` structure and generate transport block data. Generate the DL-SCH codewords and view the first ten.

```
rmc = lteRMCDL('R.7');
data = randi([0,1],rmc.PDSCH.TrBlkSizes(1),1);

codeWord = lteDLSCH(rmc,rmc.PDSCH,rmc.PDSCH.CodedTrBlkSizes(1),data);
codeWord(1:10)
```

```
ans = 10x1 int8 column vector

   1
   0
   0
   1
   1
   1
   0
   0
   0
   0
```

## Input Arguments

### enb — eNodeB cell-wide settings

structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| If `chs.NSoftBits` is defined include: | | | |
| **`DuplexMode`** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| When `DuplexMode` is set to `'TDD'` include: | | | |
| **`TDDConfig`** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| When `chs.TxScheme` is set to `'TxDiversity'` include: | | | |
| **`CellRefP`** | Optional | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |

### chs — Channel configuration

structure

Channel configuration, specified as a structure. It defines aspects of the PDSCH onto which the codewords are mapped. It also defines the DL-SCH soft buffer size and redundancy versions of the generated codewords.

`chs` can contain the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`Modulation`** | Required | `'QPSK'`, `'16QAM'`, `'64QAM'`, `'256QAM'`, `'1024QAM'` | Modulation type, specified as a character vector, cell array of character vectors, or string array. If blocks, each cell is associated with a transport block. |
| **`NLayers`** | Required | Integer from 1 to 8 | Total number of transmission layers associated with the transport block or blocks. |

<table>
<tr><th>Parameter Field</th><th>Required or Optional</th><th>Values</th><th>Description</th></tr>
<tr><td>TxScheme</td><td>Optional</td><td>'Port0' (default), 'TxDiversity', 'CDD', 'SpatialMux', 'MultiUser', 'Port7-8', 'Port7-14'.</td><td>PDSCH transmission scheme, specified as one of the following options.
<table>
<tr><th>Transmission scheme</th><th>Description</th></tr>
<tr><td>'Port0'</td><td>Single antenna port, port 0</td></tr>
<tr><td>'TxDiversity'</td><td>Transmit diversity</td></tr>
<tr><td>'CDD'</td><td>Large delay cyclic delay diversity scheme</td></tr>
<tr><td>'SpatialMux'</td><td>Closed loop spatial multiplexing</td></tr>
<tr><td>'MultiUser'</td><td>Multi-user MIMO</td></tr>
<tr><td>'Port5'</td><td>Single-antenna port, port 5</td></tr>
<tr><td>'Port7-8'</td><td>Single-antenna port, port 7, when NLayers = 1. Dual layer transmission, ports 7 and 8, when NLayers = 2.</td></tr>
<tr><td>'Port8'</td><td>Single-antenna port, port 8</td></tr>
<tr><td>'Port7-14'</td><td>Up to eight layer transmission, ports 7–14</td></tr>
</table></td></tr>
<tr><td>RV</td><td>Required</td><td>Integer vector (0,1,2,3). A one or two column matrix (for one or two codewords).</td><td>Specifies the redundancy version for one or two codewords used in the initial subframe number, NSubframe. This parameter field is only for informational purposes and is read-only.</td></tr>
<tr><td>NSoftbits</td><td>Optional</td><td>Nonnegative scalar integer (default 0)</td><td>Total number of soft buffer bits. The default setting of 0 signifies that there is no buffer limit.</td></tr>
</table>

### `outlen` — Codeword length

numeric vector of one or two elements

Codeword length, specified as a numeric vector of one or two elements. This vector defines the codeword lengths to which the input transport blocks should be rate matched. It represents the PDSCH capacity for the associated codeword. Therefore, it also represents the lengths of the vectors in `cwout`.

### `trblkin` — Transport block information bits to be encoded

numeric vector | cell array of one or two numeric vectors

Transport block information bits to be encoded, specified as a numeric vector or a cell array of numeric vectors. `trblkin` is an input parameter containing the transport block information bits to be encoded. If it is a cell array, all rate matching calculations assume that the pair is transmitting on a single PDSCH, distributed across the total number of layers defined in `chs`, as per TS 36.211 [2]. The lowest order information bit of `trblkin` maps to the most significant bit of the transport block, as defined in TS 36.321 [3], Section 6.1.1 .

## Output Arguments

### `cwout` — DL-SCH encoded codewords

numeric column vector | cell array of one or two numeric column vectors

DL-SCH encoded codewords, returned as a numeric column vector or a cell array of one or two numeric column vectors. It reflects the data type and size of the input data, `trblkin`.

Data Types: `int8` | `cell`

### `chinfo` — Additional information about encoding process

structure array | optional

Additional information about encoding process, returned as a structure array. It contains parameter fields related to code block segmentation and rate matching. If two transport blocks are encoded, `chinfo` is a structure array of two elements, with one element for each block. The code block segmentation fields in this structure can also be created independently using the `lteDLSCHInfo` function.

`chinfo` contains the following fields.

| Parameter Field | Description | Values |
|---|---|---|
| **`C`** | Total number of code blocks | Nonnegative scalar integer |
| **`Km`** | Lower code block size ($K-$) | Nonnegative scalar integer |
| **`Cm`** | Number of code blocks of size `Km` ($C-$) | Nonnegative scalar integer |
| **`Kp`** | Upper code block size ($K+$) | Nonnegative scalar integer |
| **`Cp`** | Number of code blocks of size `Kp` ($C+$) | Nonnegative scalar integer |
| **`F`** | Number of filler bits in first block | Nonnegative scalar integer |
| **`L`** | Number of segment cyclic redundancy check (CRC) bits | Nonnegative scalar integer |
| **`Bout`** | Total number of bits in all segments | Nonnegative scalar integer |
| **`NLayers`** | Number of transmission layers. | Nonnegative scalar integer |
| **`NL`** | Number of layers used in rate matching calculation | Nonnegative scalar integer |
| **`Qm`** | Bits per symbol variable used in rate matching calculation | Nonnegative scalar integer |
| **`NIR`** | Number of soft bits associated with transport block. Soft buffer size for entire input transport block | Nonnegative scalar integer |
| **`RV`** | RV value associated with one codeword<br><br>Included if RV is present at the input. | Nonnegative scalar integer |

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[3] 3GPP TS 36.321. “Evolved Universal Terrestrial Radio Access (E-UTRA); Medium Access Control (MAC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteDLSCHDecode` | `lteDLSCHInfo` | `ltePDSCH`

# lteDLSCHDecode

Downlink shared channel decoding

## Syntax

```
[trblkout,blkcrc,stateout] = lteDLSCHDecode(enb,chs,trblklen,cwin,statein)
```

## Description

`[trblkout,blkcrc,stateout] = lteDLSCHDecode(enb,chs,trblklen,cwin,statein)` returns the information bits, `trblkout`, decoded from the input soft LLR codeword data, `cwin`. The DL-SCH decoder includes rate recovery, turbo decoding, block concatenation, and CRC calculations. The function also returns the type-24A transport block CRC decoding result in `blkcrc` and the HARQ process decoding state in `stateout`. The initial HARQ process state can be provided as the optional `statein` parameter. The function is capable of processing both a single codeword or pairs of codewords, contained in a cell array, for the case of spatial multiplexing schemes transmitting two codewords. The type of the return variable, `trblkout`, is the same as the input, `cwin`. If `cwin` is a cell array containing one or two codewords, `trblkout` is a cell array of one or two transport blocks. If `cwin` is a vector of soft data, `trblkout` is a vector also. If you are decoding a pair of codewords, you must provide pairs of modulation schemes and RV indicators in the appropriate parameter fields.

`enb` is an input parameter structure that may include optional fields defining the duplex mode. Since the duplex mode defaults to `'FDD'`, if the `'DuplexMode'` field is absent, `enb` can be an empty structure.

`chs` is an input parameter structure defining aspects of the PDSCH onto which the codewords are mapped and the DL-SCH soft buffer size and redundancy versions of the received codewords.

`trblklen` is an input vector, one or two elements in length, defining the transport block lengths to which the input code blocks are rate recovered and decoded.

`cwin` is an input parameter containing the floating point soft LLR data of the codewords to be decoded. It is either a single vector or a cell array containing one or two vectors. If it is a cell array, all rate matching calculations assume that the pair is transmitting on a single PDSCH, distributed across the total number of layers defined in `chs`, as per TS 36.211 [1].

`statein` is an optional input structure array, empty or one or two elements, which can input the current decoder buffer state for each transport block in an active HARQ process. If `statein` is not an empty array and it contains a non-empty field, `CBSBuffers`, this field should contain a cell array of vectors representing the LLR soft buffer states for the set of code blocks at the input to the turbo decoder, after explicit rate recovery. The updated buffer states after decoding are returned in the `CBSBuffers` field in the output parameter, `stateout`. The `statein` array would normally be generated and recycled from the `stateout` of previous calls to `lteDLSCHDecode` as part of a sequence of HARQ transmissions.

`trblkout` is the output parameter containing the decoded information bits. It is either a single vector or a cell array containing one or two vectors, depending on the class and dimensionality of `cwin`.

`blkcrc` is an output array, one or two elements, containing the result of the type-24A transport block CRC decoding for the transport blocks.

`stateout`, the final output parameter, is a one- or two-element structure array containing the internal state of each transport block decoder. The `stateout` array is normally reapplied via the `statein` variable of subsequent `lteDLSCHDecode` function calls as part of a sequence of HARQ retransmissions.

## Examples

### Generate and Decode DL-SCH Transmissions

This example generates and decodes 2 transmissions, one with RV set to 0 and one with RV set to 1, as part of a single codeword HARQ process for RMC R.7.

Set subframe number. Get the definition of RMC R.7. Generate transport block data. Apply DL-SCH transport channel coding chain to `trBlkData`. Create a codeword with RV = 0. Turn logical bits into 'LLR' data

```
nsf = 1;

rmc = lteRMCDL('R.7');

trBlkSize = rmc.PDSCH.TrBlkSizes(nsf);
codedTrBlkSize = rmc.PDSCH.CodedTrBlkSizes(nsf);
trBlkData = randi([0,1],trBlkSize,1);

rmc.PDSCH.RV = 0;
cw = lteDLSCH(rmc,rmc.PDSCH,codedTrBlkSize,trBlkData);

cw(cw == 0) = -1;
```

Initialize the decoder states for the first HARQ transmission. The returned `decState` contains the decoder buffer state for each transport block for an active HARQ process with RV = 1

```
decState = [];
[rxTrBlk,~,decState] = lteDLSCHDecode(rmc,rmc.PDSCH,trBlkSize,cw,decState);
```

Create a second retransmitted codeword. Turn logical bits into 'LLR' data. Use the previous transmission decoder buffer state, `decState`, as part of the sequence of active HARQ transmissions

```
rmc.PDSCH.RV = 1;
cw = lteDLSCH(rmc,rmc.PDSCH,codedTrBlkSize,trBlkData);

cw(cw == 0) = -1;
rxTrBlk = lteDLSCHDecode(rmc,rmc.PDSCH,trBlkSize,cw,decState);

size(rxTrBlk)
```

```
ans = 1×2

       28336           1
```

```
rxTrBlk(1:10)
```

```
ans = 10x1 int8 column vector

   1
   1
   0
   1
   1
   0
   0
   1
   1
   1
```

## Input Arguments

### enb — Cell-wide settings

scalar structure

Cell-wide settings, specified as a structure with the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| If `chs.NSoftBits` is defined include: | | | |
| **`DuplexMode`** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex<br>Because the duplex mode defaults to `'FDD'`, if this field is absent, `enb` can be an empty structure. |
| When `DuplexMode` is set to `'TDD'` include: | | | |
| **`TDDConfig`** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration<br>Only required for `'TDD'` duplex mode. |

Data Types: `struct`

### chs — Channel configuration

structure

Channel configuration, specified as a structure having the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`Modulation`** | Required | `'QPSK'`, `'16QAM'`, `'64QAM'`, `'256QAM'`, `'1024QAM'` | Modulation type associated with each transport block, specified as a character vector, cell array of character vectors for 2 blocks, or string array. |

<table>
<tr><th>Parameter Field</th><th>Required or Optional</th><th>Values</th><th>Description</th></tr>
<tr><td>NLayers</td><td>Required</td><td>1, 2, 3, 4</td><td>Total number of transmission layers associated with the transport block or blocks.</td></tr>
<tr><td>TxScheme</td><td>Optional</td><td>'Port0' (default), 'TxDiversity', 'CDD', 'SpatialMu 'MultiUser 'Port5', 'Port7-8', 'Port8', 'Port7-14'</td><td>PDSCH transmission scheme, specified as one of the following options.
<table>
<tr><th>Transmission scheme</th><th>Description</th></tr>
<tr><td>'Port0'</td><td>Single antenna port, port 0</td></tr>
<tr><td>'TxDiversity'</td><td>Transmit diversity</td></tr>
<tr><td>'CDD'</td><td>Large delay cyclic delay diversity scheme</td></tr>
<tr><td>'SpatialMux'</td><td>Closed loop spatial multiplexing</td></tr>
<tr><td>'MultiUser'</td><td>Multi-user MIMO</td></tr>
<tr><td>'Port5'</td><td>Single-antenna port, port 5</td></tr>
<tr><td>'Port7-8'</td><td>Single-antenna port, port 7, when NLayers = 1. Dual layer transmission, ports 7 and 8, when NLayers = 2.</td></tr>
<tr><td>'Port8'</td><td>Single-antenna port, port 8</td></tr>
<tr><td>'Port7-14'</td><td>Up to eight layer transmission, ports 7–14</td></tr>
</table></td></tr>
<tr><td>RV</td><td>Required</td><td>0, 1, 2, 3<br><br>2-element numeric vector</td><td>Redundancy version indicator, specified as a numeric vector of 1 or 2 values. Possible values are 0, 1, 2, or 3.</td></tr>
<tr><td>NSoftbits</td><td>Optional</td><td>Nonnegative scalar integer (default 0)</td><td>Total number of soft buffer bits. The default setting of 0 signifies that there is no buffer limit.<br><br>If NSoftbits is absent, no limit is placed on the number of soft bits.</td></tr>
<tr><td>NTurboDecIts</td><td>Optional</td><td>5 (default)<br><br>Integer from 1 to 30</td><td>Number of turbo decoder iteration cycles</td></tr>
</table>

Data Types: `struct`

**`trblklen` — Transport block lengths**
one- or two-element numeric vector

Transport block lengths, specified as a one- or two-element numeric vector. It defines the transport block lengths to which the input code blocks should be rate-recovered and decoded.

Data Types: `double`

### `cwin` — Soft LLR codeword data

numeric vector | cell array of one or two numeric vectors

Soft LLR data of the codewords to be decoded, specified as either a numeric vector or a cell array containing one or two vectors.

Data Types: `double`

### `statein` — Initial HARQ process state

optional | structure array

Initial HARQ process state, specified as a structure array. Optional. This structure array, which can be empty or contain one or two elements, can input the current decoder buffer state for each transport block in an active HARQ process.

Data Types: `struct`

## Output Arguments

### `trblkout` — Decoded information bits

numeric vector | cell array of one or two numeric vectors

Decoded information bits, returned as a numeric vector or a cell array of one or two numeric vectors. `trblkout` reflects the data type and size of `cwin`.

Data Types: `int8` | `cell`

### `blkcrc` — Type-24A transport block CRC decoding result

logical vector of one or two elements

Type-24A transport block CRC decoding result, returned as a logical vector of one or two elements.

Data Types: `logical`

### `stateout` — HARQ process decoding state

structure array of one or two elements

HARQ process decoding state, returned as a structure array of one or two elements. It contains the internal state of each transport block in the following fields.

| Parameter Field | Values | Description |
|---|---|---|
| **`CBSBuffers`** | Cell array of vectors | Cell array of vectors representing the LLR soft buffer states for the set of code blocks associated with a single transport block. The buffers are positioned at the input to the turbo decoder, after explicit rate recovery. |
| **`CBSCRC`** | Logical vector | Array of type-24B code block set CRC decoding results |

| Parameter Field | Values | Description |
|---|---|---|
| **BLKCRC** | Logical scalar | Type-24A transport block CRC decoding error |

Data Types: `struct`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteDLSCH` | `lteDLSCHInfo` | `ltePDSCHDecode`

# lteDLSCHInfo

DL-SCH segmentation information

## Syntax

```
info = lteDLSCHInfo(blklen)
info = lteDLSCHInfo(enb,chs,blklen)
```

## Description

`info = lteDLSCHInfo(blklen)` returns a structure containing the Downlink Shared Channel (DL-SCH) code block segmentation information for the given transport block length.

`info = lteDLSCHInfo(enb,chs,blklen)` returns a structure containing the DL-SCH code block segmentation information for the given eNodeB cell-wide settings structure, channel configuration structure, and transport block length.

## Examples

### Display DL-SCH Segmentation Information

Show the sizing information before turbo coding for an input transport block of length 132. The info structure fields shows that there are 4 filler bits and the total size of the one segment after CRC addition is 160.

```
lteDLSCHInfo(132)
```

```
ans = struct with fields:
       C: 1
      Km: 0
      Cm: 0
      Kp: 160
      Cp: 1
       F: 4
       L: 0
    Bout: 160
```

### Display DL-SCH Transport Channel Information for RMC R.11

Show the DL-SCH transport channel sizing information for an R.11 RMC.

```
rmc = lteRMCDL('R.11');
lteDLSCHInfo(rmc,rmc.PDSCH,rmc.PDSCH.TrBlkSizes(1))
```

```
ans = struct with fields:
          C: 3
         Km: 4288
```

```
      Cm: 0
      Kp: 4352
      Cp: 3
       F: 0
       L: 24
    Bout: 13056
NLayers: 2
      NL: 2
      Qm: 4
     NIR: 0
      RV: 0
```

## Input Arguments

### `enb` — eNodeB cell-wide settings

structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`DuplexMode`** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| When `DuplexMode` is set to `'TDD'` include: | | | |
| **`TDDConfig`** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| When `chs.TxScheme` is set to `'TxDiversity'` include: | | | |
| **`CellRefP`** | Optional | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |

### `chs` — Channel configuration

structure

Channel configuration, specified as a structure. It defines aspects of the PDSCH onto which the codewords are mapped. It also defines the DL-SCH soft buffer size and redundancy versions of the generated codewords.

`chs` can contain the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`Modulation`** | Required | `'QPSK'`, `'16QAM'`, `'64QAM'`, `'256QAM'`, `'1024QAM'` | Modulation type, specified as a character vector, cell array of character vectors, or string array. If blocks, each cell is associated with a transport block. |
| **`NLayers`** | Required | Integer from 1 to 8 | Total number of transmission layers associated with the transport block or blocks. |

<table>
<tr><th>Parameter Field</th><th>Required or Optional</th><th>Values</th><th>Description</th></tr>
<tr><td>TxScheme</td><td>Required</td><td>'Port0', 'TxDiversity', 'CDD', 'SpatialMux', 'MultiUser', 'Port7-8', 'Port7-14'.</td><td>PDSCH transmission scheme, specified as one of the following options.
<table>
<tr><th>Transmission scheme</th><th>Description</th></tr>
<tr><td>'Port0'</td><td>Single antenna port, port 0</td></tr>
<tr><td>'TxDiversity'</td><td>Transmit diversity</td></tr>
<tr><td>'CDD'</td><td>Large delay cyclic delay diversity scheme</td></tr>
<tr><td>'SpatialMux'</td><td>Closed loop spatial multiplexing</td></tr>
<tr><td>'MultiUser'</td><td>Multi-user MIMO</td></tr>
<tr><td>'Port5'</td><td>Single-antenna port, port 5</td></tr>
<tr><td>'Port7-8'</td><td>Single-antenna port, port 7, when NLayers = 1. Dual layer transmission, ports 7 and 8, when NLayers = 2.</td></tr>
<tr><td>'Port8'</td><td>Single-antenna port, port 8</td></tr>
<tr><td>'Port7-14'</td><td>Up to eight layer transmission, ports 7–14</td></tr>
</table></td></tr>
<tr><td>RV</td><td>Required</td><td>Integer vector (0,1,2,3). A one or two column matrix (for one or two codewords).</td><td>Specifies the redundancy version for one or two codewords used in the initial subframe number, NSubframe. This parameter field is only for informational purposes and is read-only.</td></tr>
<tr><td>NSoftbits</td><td>Optional</td><td>Nonnegative scalar integer (default 0)</td><td>Total number of soft buffer bits. The default setting of 0 signifies that there is no buffer limit.</td></tr>
</table>

#### `blklen` — Transport block length

positive scalar integer | two-element positive integer vector

Transport block length, specified as a positive integer or a two-element positive integer vector. A two-element vector defines the length of transport blocks for two codewords.

Data Types: `double`

## Output Arguments

#### `info` — DL-SCH code block segmentation information

structure array

DL-SCH code block segmentation information, returned as a structure array including the following fields.

| Parameter Field | Description | Values |
|---|---|---|
| **C** | Total number of code blocks | Nonnegative scalar integer |

| Parameter Field | Description | Values |
|---|---|---|
| **Km** | Lower code block size ($K-$) | Nonnegative scalar integer |
| **Cm** | Number of code blocks of size `Km` ($C-$) | Nonnegative scalar integer |
| **Kp** | Upper code block size ($K+$) | Nonnegative scalar integer |
| **Cp** | Number of code blocks of size `Kp` ($C+$) | Nonnegative scalar integer |
| **F** | Number of filler bits in first block | Nonnegative scalar integer |
| **L** | Number of segment cyclic redundancy check (CRC) bits | Nonnegative scalar integer |
| **Bout** | Total number of bits in all segments | Nonnegative scalar integer |
| When syntax includes `enb` and `chs` inputs, output `info` also includes these fields: | | |
| **NLayers** | Number of layers associated with one codeword | Nonnegative scalar integer |
| **NL** | Number of layers used in rate matching calculation | Nonnegative scalar integer |
| **Qm** | Bits per symbol variable used in rate matching calculation | Nonnegative scalar integer |
| **NIR** | Number of soft bits associated with transport block. Soft buffer size for entire input transport block | Nonnegative scalar integer |
| **RV** | RV value associated with one codeword<br><br>Included if RV is present at the input. | Nonnegative scalar integer |

## Version History

**Introduced in R2014a**

## See Also

`lteDLSCH` | `lteDLSCHDecode`

# lteDMRS

UE-specific demodulation reference signals

## Syntax

```
sym = lteDMRS(enb,chs)
sym = lteDMRS(enb,chs,opts)
```

## Description

`sym = lteDMRS(enb,chs)` returns the downlink UE-specific demodulation reference signal (DM-RS) symbols for transmission in a single subframe, given structures containing the cell-wide settings, and the PDSCH configuration settings. For more information, see “DM-RS Associated with PDSCH” on page 2-219.

`sym = lteDMRS(enb,chs,opts)` allows control of the format of the returned symbols with the options cell array, `opts`.

## Examples

### Map PDSCH DM-RS Symbols to Grid

Map DM-RS symbols for 4 layers onto an 8 antenna grid.

Initialize cell-wide settings for RMC `'R.1'` (10 MHz bandwidth, 1 RB allocation) and change to Release 10 transmission (`'Port7-14'`). Use `enb.PDSCH` for the channel configuration structure input. Generate and map DM-RS without clearing the REs that should not be mapped because of the DM-RS on other ports.

```
enb = lteRMCDL('R.1');
enb.PDSCH.TxScheme = 'Port7-14';
enb.PDSCH.NLayers = 4;
ntxants = 8;
enb.PDSCH.W = lteCSICodebook(enb.PDSCH.NLayers,ntxants,[0 0]).';

subframe = ones(lteResourceGridSize(enb,ntxants));
enb.PDSCH.NTxAnts = size(enb.PDSCH.W,2);
dmrsInd = lteDMRSIndices(enb,enb.PDSCH);
dmrs = lteDMRS(enb,enb.PDSCH);
subframe(dmrsInd) = dmrs;
```

View the size of the output symbols, indices, and the Release 10 transmission `subframe`.

```
size(dmrs)
```

```
ans = 1×2

   192     1
```

```
size(dmrsInd)
```

```
ans = 1×2

   192     1
```

```
size(subframe)
```

```
ans = 1×3

   600    14     8
```

### Map Non-Precoded DM-RS Symbols to Grid

Map non-precoded DM-RS symbols onto an 4 layer grid, and clear the REs which should not be used because of the DM-RS of other ports.

Initialize cellwide settings for RMC `'R.1'` (10 MHz bandwidth, 1 RB allocation) and change to Release 10 transmission (`'Port7-14'`). Generate and map DM-RS clearing the REs that should not be used because of the DM-RS on other ports.

```
enb = lteRMCDL('R.1');
enb.PDSCH.TxScheme = 'Port7-14';
enb.PDSCH.NLayers = 4;

subframe = ones(lteResourceGridSize(enb,enb.PDSCH.NLayers));
dmrsInd = lteDMRSIndices(enb,enb.PDSCH,'rs+unused');
dmrs = lteDMRS(enb,enb.PDSCH,'rs+unused');
subframe(dmrsInd) = dmrs;

size(dmrs)
```

```
ans = 1×2

    96     1
```

```
size(dmrsInd)
```

```
ans = 1×2

    96     1
```

```
size(subframe)
```

```
ans = 1×3

   600    14     4
```

## Input Arguments

### enb — eNodeB cell-wide settings

structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NDLRB** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **NCellID** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **NSubframe** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **CyclicPrefix** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **DuplexMode** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| The following parameters are dependent upon the condition that `DuplexMode` is set to `'TDD'`. | | | |
| **TDDConfig** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **SSC** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |

**`chs` — PDSCH-specific channel transmission configuration**
structure

PDSCH-specific channel transmission configuration, specified as a structure that can contain these parameter fields.

<table>
<tr><th>Parameter Field</th><th>Required or Optional</th><th>Values</th><th>Description</th></tr>
<tr><td>PRBSet</td><td>Required</td><td>Integer column vector or two-column matrix</td><td>Zero-based physical resource block (PRB) indices corresponding to the slot wise resource allocations for this PDSCH. PRBSet can be assigned as:<ul><li>a column vector, the resource allocation is the same in both slots of the subframe,</li><li>a two-column matrix, this parameter specifies different PRBs for each slot in a subframe,</li><li>a cell array of length 10 (corresponding to a frame, if the allocated physical resource blocks vary across subframes).</li></ul>PRBSet varies per subframe for the RMCs 'R.25'(TDD), 'R.26'(TDD), 'R.27'(TDD), 'R.43'(FDD), 'R.44', 'R.45', 'R.48', 'R.50', and 'R.51'.</td></tr>
<tr><td>TxScheme</td><td>Optional</td><td>'Port5' (default), 'Port7-8', 'Port8', 'Port7-14'</td><td>DM-RS-specific transmission scheme, specified as one of the following options.<table><tr><th>Transmission scheme</th><th>Description</th></tr><tr><td>'Port5'</td><td>Rel-8 single-antenna port, port 5</td></tr><tr><td>'Port7-8'</td><td>Rel-9 single-antenna port, port 7 if NLayers is 1. Rel-9 dual-layer transmission, ports 7 and 8 if NLayers is 2.</td></tr><tr><td>'Port8'</td><td>Rel-9 single-antenna port, port 8</td></tr><tr><td>'Port7-14'</td><td>Rel-10 up to 8 layer transmission, ports 7–14 if NLayers a value from 1 to 8.</td></tr></table></td></tr>
<tr><td>NLayers</td><td>Optional</td><td>1 (default), 2, 3, 4, 5, 6, 7, 8</td><td>Number of transmission layers.</td></tr>
<tr><td>W</td><td>Optional</td><td>Numeric matrix, [] (default)</td><td>NLayers-by-P precoding matrix for the wideband UE-specific beamforming of the DM-RS. P is the number of transmit antennas. An empty matrix, [], signifies no precoding.</td></tr>
<tr><td colspan="4">The following parameter is applicable when TxScheme is set to 'Port7-8', 'Port8', or 'Port7-14'.</td></tr>
</table>

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NSCID** | Optional | 0 (default), 1 | Scrambling identity (*ID*) |
| The following parameter is applicable when `TxScheme` is set to `'Port5'`. | | | |
| **RNTI** | Required | 0 (default), scalar integer | Radio network temporary identifier (RNTI) value (16 bits) |

### `opts` — Symbol generation options

character vector | cell array of character vectors | string array

Symbol generation options, specified as a character vector, cell array of character vectors, or string array. Values for `opts` when specified as a character vector include (use double quotes for string):

| Option | Values | Description |
|---|---|---|
| Symbol style | `'ind'` (default), `'mat'` | Style for returning DM-RS symbols, specified as one of the following options.<br>• `'ind'` — returns the DM-RS symbols as an $N_{RE}$-by-1 vector (default)<br>• `'mat'` — returns the DM-RS symbols as a matrix. To form a matrix, a column may contain duplicate entries. In this style, each column contains symbols for —<br>  • an individual port or layer, if symbols are not precoded,<br>  • or the projected layers per transmit antenna if symbols are precoded.<br>$N_{RE}$ is the number of resource elements. |
| Symbol format | `'rsonly'` (default), `'rs+unused'` | Format for returning DM-RS symbols, specified as one of the following options.<br>• `'rsonly'` — returns only active DM-RS symbols (default)<br>• `'rs+unused'` — returns include zeros for the RE locations that should be unused because of DM-RS transmission on another port or layer. This format is equivalent to precoding with `W` set to `eye(NLayers)`. |

Example: `{'ind','rs+unused'}`, returns the DM-RS symbols as a column vector that includes zeros for the RE locations that should be unused because of DM-RS transmission on another port or layer.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `sym` — DM-RS symbol sequences

$N_{RE}$-by-1 numeric column vector (default) | numeric matrix

DM-RS symbol sequences, returned as an $N_{RE}$-by-1 numeric column vector, or a numeric matrix. $N_{RE}$ is the number of resource elements. The `opts` input offers alternative output styles or formats.

`sym` contains the non-precoded or precoded DM-RS symbol sequences concatenated for all the layers, or the transmit antennas for the transmission scheme. The symbols are always ordered as they should be mapped using `lteDMRSIndices` into an *M*-by-*N*-by-*P* array representing the subframe grid across

either the non-precoded PDSCH layers or precoded transmit antennas. *M* is the number of subcarriers, *N* is the number of symbols, and *P* is the number of layers, or antennas.

Since precoding projects the DM-RS in each PDSCH layer onto all `NTxAnts` transmit antennas, the output contains the concatenation of all DM-RS across all layers, which are then duplicated in all `chs.NTxAnts` planes of the 3-D grid.

- The output is returned empty unless `chs.TxScheme` is set to one of the schemes related to DM-RS, specifically `'Port5'`, `'Port7-8'`, `'Port8'`, or `'Port7-14'`.
- If the `chs.TxScheme` is single port, `chs.NLayers` = 1 implicitly.
- The output does not include any elements allocated to PBCH, PSS, and SSS. If the subframe contains no DM-RS, an empty vector is returned.
- If the precoding matrix, field `chs.`*`W`*, is not present or is empty, the output is returned containing only the concatenated non-precoded DM-RS symbols for the `NLayers` ports.
- Otherwise, the output, `sym`, contains all DM-RS symbol values after they are precoded using the `NLayers`-by-`NTxAnts` beamforming matrix, *W*, onto `NTxAnts` transmit antennas. The symbols are ordered by:
  - The concatenation of DM-RS symbols per layer/port if not precoded
  - The projected layers per transmit antenna if precoded.

For more information, see “DM-RS Associated with PDSCH” on page 2-219.

Data Types: `double`
Complex Number Support: Yes

## More About

### DM-RS Associated with PDSCH

As specified in TS 36.211, Section 6.10.3, UE-specific demodulation reference signal (DM-RS) associated with the physical downlink shared channel (PDSCH):

- are transmitted in a single subframe on antenna ports *p*=5, *p*=7, *p*=8, or *p*=7, 8,..., (`NLayers`+6).
- are present and are a valid reference for PDSCH demodulation only if the PDSCH transmission is associated with the corresponding antenna port according to TS 36.213, Section 7.1.
- are transmitted only on the physical resource blocks upon which the corresponding PDSCH is mapped.

These DM-RS are for use with Release 8, 9, and 10 non-codebook-based PDSCH transmission schemes.

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.213. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteDMRSIndices` | `lteCellRS` | `lteEPDCCHDMRS` | `ltePRS` | `lteCSIRS` | `ltePDSCH` | `ltePRBS`

# lteDMRSIndices

UE-specific DM-RS resource element indices

## Syntax

```
ind = lteDMRSIndices(enb,chs)
ind = lteDMRSIndices(enb,chs,opts)
```

## Description

`ind = lteDMRSIndices(enb,chs)` returns the indices of the downlink UE-specific demodulation reference signal (DM-RS) resource elements (RE) in a subframe, given structures containing the cell-wide settings, and the PDSCH settings. For more information, see “DM-RS Associated with PDSCH” on page 2-225.

`ind = lteDMRSIndices(enb,chs,opts)` formats the returned indices using options specified by `opts`.

## Examples

### Generate PDSCH DM-RS Indices

Generate DM-RS Resource Element (RE) indices in default format for RMC R.28.

Initialize cell-wide parameters to RMC R.28 using the `lteRMCDL` function. Use `enb.PDSCH` for the channel configuration structure. Generate the RE indices.

```
enb = lteRMCDL('R.28');
ind = lteDMRSIndices(enb,enb.PDSCH);
```

View first four rows of index column vector

```
size(ind)
```

```
ans = 1×2

    24     1
```

```
ind(1:4)
```

```
ans = 4x1 uint32 column vector

   1801
   1805
   1809
   3603
```

### Generate Zero-Based DM-RS Indices

Generate 0-based Resource Element indices in subscript form for RMC R.28. The resultant matrix has three columns, with each row representing [subcarrier, symbol, antenna port].

Initialize cell-wide parameters to RMC R.28 using the `lteRMCDL` function.

```
enb = lteRMCDL('R.28');
```

Generate zero-based RE indices in subscript form.

```
ind = lteDMRSIndices(enb,enb.PDSCH,{'0based','sub'});
```

View first four rows of index matrix.

```
size(ind)
```

```
ans = 1×2

    24     3
```

```
ind(1:4,:)
```

```
ans = 4x3 uint32 matrix

   0   3   0
   4   3   0
   8   3   0
   2   6   0
```

## Input Arguments

### enb — eNodeB cell-wide settings

structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NDLRB** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **NSubframe** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **CyclicPrefix** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **DuplexMode** | Optional | 'FDD' (default), 'TDD' | Duplexing mode, specified as either:<br>• 'FDD' for Frequency Division Duplex<br>• 'TDD' for Time Division Duplex |
| The following parameters are dependent upon the condition that DuplexMode is set to 'TDD'. | | | |
| **TDDConfig** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **SSC** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |
| The following parameter is only applicable when TxScheme is set to 'Port5'. | | | |
| **NCellID** | Required | Integer from 0 to 503 | Physical layer cell identity |

**chs — PDSCH-specific channel transmission configuration**
structure

PDSCH-specific channel transmission configuration, specified as a structure that can contain these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **PRBSet** | Required | Integer column vector or two-column matrix | Zero-based physical resource block (PRB) indices corresponding to the slot wise resource allocations for this PDSCH. PRBSet can be assigned as:<br>• a column vector, the resource allocation is the same in both slots of the subframe,<br>• a two-column matrix, this parameter specifies different PRBs for each slot in a subframe,<br>• a cell array of length 10 (corresponding to a frame, if the allocated physical resource blocks vary across subframes).<br>PRBSet varies per subframe for the RMCs 'R.25'(TDD), 'R.26'(TDD), 'R.27'(TDD), 'R.43'(FDD), 'R.44', 'R.45', 'R.48', 'R.50', and 'R.51'. |

<table>
<tr><th>Parameter Field</th><th>Required or Optional</th><th>Values</th><th>Description</th></tr>
<tr><td>TxScheme</td><td>Optional</td><td>'Port5' (default), 'Port7-8', 'Port8', 'Port7-14'</td><td>DM-RS-specific transmission scheme, specified as one of the following options.
<table>
<tr><th>Transmission scheme</th><th>Description</th></tr>
<tr><td>'Port5'</td><td>Rel-8 single-antenna port, port 5</td></tr>
<tr><td>'Port7-8'</td><td>Rel-9 single-antenna port, port 7 if NLayers is 1. Rel-9 dual-layer transmission, ports 7 and 8 if NLayers is 2.</td></tr>
<tr><td>'Port8'</td><td>Rel-9 single-antenna port, port 8</td></tr>
<tr><td>'Port7-14'</td><td>Rel-10 up to 8 layer transmission, ports 7–14 if NLayers a value from 1 to 8.</td></tr>
</table></td></tr>
<tr><td>NLayers</td><td>Optional</td><td>1 (default), 2, 3, 4, 5, 6, 7, 8</td><td>Number of transmission layers.</td></tr>
<tr><td>NTxAnts</td><td>Optional</td><td>0 (default), nonnegative integer</td><td>Number of transmission antenna ports. This argument is present only for UE-specific demodulation reference symbols.</td></tr>
</table>

### opts — Index generation options

character vector | cell array of character vectors | string array

Index generation options, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for opts when specified as a character vector include (use double quotes for string):

| Option | Values | Description |
|---|---|---|
| Indexing style | 'ind' (default), 'mat', 'sub' | Style for the returned indices, specified as one of the following options.<br>• 'ind' — returns the indices as an $N_{RE}$-by-1 vector (default)<br>• 'mat' — returns the indices as a matrix. If not precoded, each column contains indices for an individual layer/port. If precoded, each column contains symbols for a transmit antenna. To form a matrix, a column can contain duplicate entries.<br>• 'sub' — returns the indices as an $N_{RE}$-by-3 matrix. in [subcarrier, symbol, antenna] subscript row style.<br>$N_{RE}$ is the number of resource elements. |
| Index base | '1based' (default), '0based' | Base value of the returned indices. Specify '1based' to generate indices where the first value is 1. Specify '0based' to generate indices where the first value is 0. |

| Option | Values | Description |
|---|---|---|
| Indexing format | `'rsonly'` (default), `'rs+unused'` | Format for the returned indices, specified as one of the following options.<br>• `'rsonly'` — returns only active DM-RS symbols (default)<br>• `'rs+unused'` — also includes zeros for the resource element (RE) locations that should be unused because of DM-RS transmission on another port or layer. This format is equivalent to precoding with `NTxAnts` set to `NLayers`. |

Example: `'ind 1based rs+unused'`, `"ind 1based rs+unused"`, `{"ind","1based","rs+unused"]`or `{'ind','1based','rs+unused'}` specify the same formatting options.

Example: `'ind rsonly'`, `"ind rsonly"`, `{'ind','rsonly'}`, or `["ind","rsonly"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `ind` — DM-RS resource element indices

linear indexing $N_{RE}$-by-1 column vector (default) | linear indexing matrix | numeric 3-column matrix

DM-RS resource element indices, returned as a linear indexing $N_{RE}$-by-1 column vector, a linear indexing matrix, or a numeric 3-column matrix. The `opts` input offers alternative output styles or formats.

`ind` can directly index the DM-RS elements in an *M*-by-*N*-by-*P* array representing the subframe grid across either the non-precoded PDSCH layers, or precoded transmit antennas. *M* is the number of subcarriers, *N* is the number of symbols, and *P* is the number of layers, or antennas.

For more information, see “DM-RS Associated with PDSCH” on page 2-225.

Data Types: `uint32`

## More About

### DM-RS Associated with PDSCH

As specified in TS 36.211, Section 6.10.3, UE-specific demodulation reference signal (DM-RS) associated with the physical downlink shared channel (PDSCH):

- are transmitted in a single subframe on antenna ports *p*=5, *p*=7, *p*=8, or *p*=7, 8,..., (`NLayers`+6).
- are present and are a valid reference for PDSCH demodulation only if the PDSCH transmission is associated with the corresponding antenna port according to TS 36.213, Section 7.1.
- are transmitted only on the physical resource blocks upon which the corresponding PDSCH is mapped.

These DM-RS are for use with Release 8, 9, and 10 non-codebook-based PDSCH transmission schemes.

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.213. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteDMRS` | `lteCellRSIndices` | `lteCSIRSIndices` | `lteEPDCCHDMRSIndices` | `ltePRSIndices` | `lteSRSIndices`

# lteDuplexingInfo

Duplexing information

## Syntax

```
info = lteDuplexingInfo(enb)
```

## Description

`info = lteDuplexingInfo(enb)` returns a structure, `info`, providing information on the duplexing arrangement. For more information, see “Duplex Mode Configuration” on page 2-229.

## Examples

### Get TDD Downlink Frame Duplexing Information

Get the number of downlink OFDM symbols in each subframe for a TDD (configuration 0) frame.

A Configuration 0 TDD frame is organized as follows:

- Only subframes 0, 1, 5, and 6 will contain a non-zero number of DL OFDM symbols.
- Subframe 0 and 5 are designated for DL.
- Subframes 1 and 6 are special subframes.
- Subframes 2, 3, 4, 7, 8, and 9 are designated for UL.

Initialize a cell-wide configuration structure for RMC R.0 and a Configuration 0 TDD frame.

```
enb = lteRMCDL('R.0');
enb.DuplexMode = 'TDD';
enb.SSC = 0;
enb.TDDConfig = 0;
```

Loop through all subframes in a frame.

```
for n = 0:9
    enb.NSubframe = n;
    duplexInfo = lteDuplexingInfo(enb);
    fprintf('DL symbols in subframe %d: %d\n',n,duplexInfo.NSymbolsDL)
end
```

```
DL symbols in subframe 0: 14
DL symbols in subframe 1: 3
DL symbols in subframe 2: 0
DL symbols in subframe 3: 0
DL symbols in subframe 4: 0
DL symbols in subframe 5: 14
DL symbols in subframe 6: 3
DL symbols in subframe 7: 0
DL symbols in subframe 8: 0
DL symbols in subframe 9: 0
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure containing these fields. For more information, see “Duplex Mode Configuration” on page 2-229.

### CyclicPrefix — Cyclic prefix length in downlink

`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length in downlink, specified as `'Normal'` or `'Extended'`.

Data Types: `char`

### CyclicPrefixUL — Cyclic prefix length in uplink

`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length in uplink, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

### DuplexMode — Duplexing mode

`'FDD'` (default) | `'TDD'` | optional

Duplexing mode, specified as `'FDD'` or `'TDD'`.

Data Types: `char` | `string`

### TDDConfig — Uplink or downlink configuration

0 (default) | integer from 0 to 6 | optional

Uplink or downlink configuration, specified as an integer from 0 to 6. Required only if `DuplexMode` is set to `'TDD'`.

Data Types: `double`

### SSC — Special subframe configuration

0 (default) | integer from 0 to 9 | optional

Special subframe configuration, specified as an integer from 0 to 9. Required only if `DuplexMode` is set to `'TDD'`.

Data Types: `double`

### NSubframe — Subframe number

nonnegative integer

Subframe number, specified as a nonnegative integer. Required only if `DuplexMode` is set to `'TDD'`.

Data Types: `double`

Data Types: `struct`

## Output Arguments

### `info` — Duplexing information

structure

Duplexing information, returned as a structure containing the following fields.

### `NSymbols` — Total number of symbols in subframe

nonnegative integer

Total number of symbols in subframe, returned as a nonnegative integer.

### `SubframeType` — Type of subframe

`'Downlink'` | `'Uplink'` | `'Special'`

Type of subframe, returned as `'Downlink'`, `'Uplink'`, or `'Special'`.

### `NSymbolsDL` — Number of symbols used for transmission in downlink

nonnegative integer

Number of symbols used for transmission in downlink, returned as a nonnegative integer.

### `NSymbolsGuard` — Number of symbols in the guard period

nonnegative integer

Number of symbols in the guard period, returned as a nonnegative integer.

### `NSymbolsUL` — Number of symbols used for transmission in uplink

nonnegative integer

Number of symbols used for transmission in uplink (UL), returned as a nonnegative integer.

## More About

### Duplex Mode Configuration

For FDD duplex mode:

- If `CyclicPrefixUL` is present, the link direction is assumed to be uplink.
- If `CyclicPrefixUL` is not present, the link direction is assumed to be downlink, and cyclic prefix is set according to `CyclicPrefix`.
  - If `CyclicPrefix` is also not present, the default `'Normal'` cyclic prefix is used.

For TDD duplex mode:

- The subframe type can be *uplink*, *downlink*, or *special*. `TDDConfig` and `NSubframe` identify the subframe type as specified in TS 36.211 [1], Table 4.2-2.
  - For uplink or downlink subframes, `CyclicPrefixUL` or `CyclicPrefix`, respectively, indicate the relevant cyclic prefix setting.
  - For special subframes, the `lteDuplexingInfo` function uses `SSC` and `CyclicPrefix` to identify the special subframe configuration, as specified in TS 36.211 [1], Table 4.2-1.

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteResourceGrid` | `lteDLResourceGrid` | `lteULResourceGrid`

**Topics**

“FDD and TDD Duplexing”

# lteEPDCCH

Enhanced physical downlink control channel

## Syntax

```
sym = lteEPDCCH(enb,chs,cw)
```

## Description

`sym = lteEPDCCH(enb,chs,cw)` returns a vector `sym` of complex modulation symbols associated with a single Enhanced Physical Downlink Control Channel (EPDCCH) transmission in a subframe. The channel processing includes the stages of scrambling and QPSK modulation. The function is initialized according to the cell-wide settings, `enb`, and the channel transmission configuration, `chs`. For a given input bit vector, `cw`, the column output, `sym`, contains the QPSK symbols ready to be mapped into the resource elements indicated by `lteEPDCCHIndices`.

This function performs no precoding. If necessary, apply precoding externally.

You can obtain the EPDCCH transmission capacity from the `info` structure produced by `lteEPDCCHIndices`.

## Examples

### Generate Complex Modulated EPDCCH Symbols

Specify the cell-wide settings and channel transmission configuration in parameter structures `enb` and `chs`.

```
enb.NSubframe = 4;
chs.EPDCCHNID = 7;
```

Generate EPDCCH symbols by encoding the input `cw` into QPSK symbols.

```
cw = randi([0 1],100,1);
sym = lteEPDCCH(enb,chs,cw);
```

Display the size and the first 10 indices of `sym`. Because these are QPSK symbols, `sym` contains half as many symbols as the number of bits that can be transmitted on the EPDCCH.

```
size(sym)
```

```
ans = 1×2

    50     1
```

```
sym(1:10)
```

```
ans = 10×1 complex

  -0.7071 + 0.7071i
```

```
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
```

## Input Arguments

### `enb` — eNodeB cell-wide settings

structure

eNodeB cell-wide settings, specified as a structure. This argument must contain the following parameter field.

### `NSubframe` — Subframe number

nonnegative scalar integer

Subframe number, specified as a nonnegative scalar integer.

Data Types: `double`

Data Types: `struct`

### `chs` — Channel-specific transmission configuration

structure

Channel-specific transmission configuration, specified as a structure. This argument must contain the following parameter field.

### `EPDCCHNID` — EPDCCH scrambling sequence initialization

nonnegative scalar integer

EPDCCH nID parameter for scrambling sequence initialization, specified as a nonnegative scalar integer.

Data Types: `double`

Data Types: `struct`

### `cw` — Input bit vector

vector

Input bit vector containing the bit values of the EPDCCH codeword for modulation.

## Output Arguments

### `sym` — EPDCCH modulation symbols

complex-vector

Given an input bit vector, `cw`, the output, `sym`, is returned as a vector of complex modulation symbols associated with a single EPDCCH transmission in a subframe. `sym` contains the QPSK symbols ready to be mapped into the resource elements indicated by `lteEPDCCHIndices`.

## Version History

**Introduced in R2014b**

## See Also

`lteEPDCCHIndices` | `lteEPDCCHPRBS` | `lteDCIEncode` | `ltePDCCH`

# lteEPDCCHDMRS

EPDCCH demodulation reference signals

## Syntax

```
sym = lteEPDCCHDMRS(enb,chs)
sym = lteEPDCCHDMRS(enb,chs,opts)
```

## Description

`sym = lteEPDCCHDMRS(enb,chs)` returns the Enhanced Physical Downlink Control Channel Demodulation Reference Signal (EPDCCH DM-RS) symbols for transmission in a single subframe. By default the symbols are returned as a column vector. The order of the symbols is the same as the order that results when you use `lteEPDCCHDMRSIndices` to map them into an *N*-by-*M*-by-4 array. This array represents the resource element subframe grid across the four possible EPDCCH antenna ports (*p* = 107...110).

The symbols are parameterized in terms of a configured PRB pair set which defines:

- the overall set of possible EPDCCH candidates and
- the aggregation of one or more consecutive enhanced control channel elements (ECCE). This aggregation identifies the specific EPDCCH instance that the DM-RS is associated with.

The DM-RS symbols are created only for the specific PRB pairs and antenna ports that the corresponding EPDCCH is mapped to.

For a localized EPDCCH transmission, the EPDCCH is associated with a single antenna port from *p* = 107...110, dependent on the `chs.RNTI` and ECCEs selected. Thus, the DM-RS antenna port symbols are output only for that single port.

For a distributed transmission, the EPDCCH is mapped to two antenna ports in an alternating fashion. Therefore, the DM-RS symbols are generated for the PRBs in both ports: *p* = 107,109 for normal cyclic prefix and *p* = 107,108 for extended cyclic prefix. The output is ordered so that the symbols for the lowest antenna ports index come first. This order matches that of the DM-RS RE indices produced by `lteEPDCCHDMRSIndices`.

`sym = lteEPDCCHDMRS(enb,chs,opts)` allows control of the format of the symbols through the options specified by `opts`. You can use this syntax to return the symbols as a numeric matrix, where each column contains symbols for an active antenna port.

This function performs no precoding. If necessary, apply precoding externally.

## Examples

### Generate EPDCCH DM-RS Symbols

Specify the cell-wide settings and channel transmission configuration in parameter structures `enb` and `chs`.

```
enb.NDLRB = 6;
enb.NSubframe = 0;
chs.EPDCCHECCE = [0 7];
chs.EPDCCHType = 'Localized';
chs.EPDCCHPRBSet = 2:3;
chs.EPDCCHNID = 0;
chs.RNTI = 1;
```

Generate EPDCCH demodulation reference signal symbols.

```
sym = lteEPDCCHDMRS(enb,chs)
```

```
sym = 12×1 complex

   0.7071 - 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
   0.7071 - 0.7071i
   0.7071 - 0.7071i
   0.7071 + 0.7071i
   0.7071 - 0.7071i
   0.7071 + 0.7071i
  -0.7071 - 0.7071i
       ⋮
```

Note: The warning messages generated simply advise you that default values are available and being used for uninitialized parameters. To suppress warnings for defaulted lte parameter settings, precede code with the following command: `lteWarning('off','DefaultValue')`

### Generate DM-RS Symbols for EPDCCH Having a Distributed Transmission

Specify the cell-wide settings and channel transmission configuration in parameter structures `enb` and `chs`.

```
enb.NDLRB = 6;
enb.NSubframe = 0;
chs.EPDCCHECCE = [0,7];
chs.EPDCCHType = 'Distributed';
chs.EPDCCHPRBSet = 2:3;
chs.EPDCCHNID = 0;
chs.RNTI = 1;
```

Generate DM-RS symbols for an EPDCCH having a distributed transmission. Return the symbols as a matrix, where each column contains symbols for an active antenna.

```
sym = lteEPDCCHDMRS(enb,chs,'mat')
```

```
sym = 12×2 complex

   0.7071 - 0.7071i   0.7071 - 0.7071i
  -0.7071 - 0.7071i  -0.7071 - 0.7071i
   0.7071 + 0.7071i   0.7071 + 0.7071i
   0.7071 - 0.7071i   0.7071 - 0.7071i
```

```
 0.7071 - 0.7071i   0.7071 - 0.7071i
-0.7071 + 0.7071i  -0.7071 + 0.7071i
-0.7071 - 0.7071i  -0.7071 - 0.7071i
 0.7071 - 0.7071i   0.7071 - 0.7071i
-0.7071 - 0.7071i  -0.7071 - 0.7071i
-0.7071 - 0.7071i  -0.7071 - 0.7071i
    ⋮
```

Note: The warning messages generated simply advise you that default values are available and being used for uninitialized parameters. To suppress warnings for defaulted lte parameter settings, precede code with the following command: `lteWarning('off','DefaultValue')`

## Input Arguments

### `enb` — eNodeB cell-wide settings

structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NDLRB`** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **`CyclicPrefix`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **`NSubframe`** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **`DuplexMode`** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as one of the following:<br>• `'FDD'` — Frequency division duplex (default)<br>• `'TDD'` — Time division duplex |
| The following parameters apply when `DuplexMode` is set to `'TDD'`. | | | |
| **`TDDConfig`** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **`SSC`** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |

### `chs` — Channel-specific channel transmission configuration

structure

Channel-specific transmission configuration, specified as a structure that can contain the following parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **EPDCCHECCE** | Required | 1-element or 2-element vector specifying the 0-based ECCE index or inclusive [`begin`, `end`] ECCE index range according to the aggregation level `L` (`L = end − begin + 1`). The number of ECCEs in the candidate must be a power of 2.<br><br>If no transmission is required, leave this parameter empty. | The set of one or several consecutive ECCEs defining the EPDCCH transmission candidate in the overall EPDCCH set. |
| **EPDCCHType** | Required | `'Localized'`, `'Distributed'` | EPDCCH transmission type |
| **EPDCCHPRBSet** | Required | Vector of zero-based indices for the PRB pairs corresponding to the EPDCCH PRB set. The number of PRB pair indices must be a power of 2.<br><br>If no transmission is required, leave this parameter empty. | EPDCCH PRB pair indices |
| **EPDCCHNID** | Required | Nonnegative scalar integer | EPDCCH nID parameter for scrambling sequence initialization |
| The following parameters apply when `EPDCCHType` is set to `'Localized'`. | | | |
| **RNTI** | Required | 0 (default), scalar integer | Radio network temporary identifier (RNTI) value (16 bits) |

### `opts` — Symbol generation options

character vector | cell array of character vectors | string array

Symbol generation options, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string):

| Option | Values | Description |
|---|---|---|
| Symbol style | `'ind'` (default), `'mat'` | Style for the returned symbols, specified as one of the following:<br>• `'ind'` — returns the symbols as a column vector (default)<br>• `'mat'` — returns the symbols as a matrix in which each column contains symbols for an active antenna port from the set $p$ = 107...110 |
| Symbol format | `'rsonly'` (default), `'rs+unused'` | Format of the returned symbols.<br>• `'rsonly'` — returns only active DM-RS symbols (default)<br>• `'rs+unused'` — also returns zeros for the RE locations, which should be unused because of EPDCCH DM-RS transmission on other EPDCCH antenna ports $p$ = 107...110 that are not used by this EPDCCH transmission. |

Example: `'ind rs+unused'`, `"ind rs+unused"`, `{'ind','rs+unused'}`, or `{"ind","rs+unused"}` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `sym` — EPDCCH DM-RS symbols

numeric column vector | numeric matrix

EPDCCH demodulation reference signal symbols, returned as a column vector containing the non-precoded DM-RS symbol sequences concatenated for all active PRB pairs and antenna ports. Optionally, the function returns `sym` as a numeric matrix, where each column contains symbols for an active antenna port.

Data Types: `double`

# Version History

**Introduced in R2014b**

## See Also

`lteEPDCCHDMRSIndices` | `lteCellRS` | `lteDMRS` | `lteCSIRS` | `ltePRS` | `lteEPDCCH` | `ltePRBS`

# lteEPDCCHDMRSIndices

EPDCCH DM-RS resource element indices

## Syntax

```
ind = lteEPDCCHDMRSIndices(enb,chs)
ind = lteEPDCCHDMRSIndices(enb,chs,opts)
```

## Description

`ind = lteEPDCCHDMRSIndices(enb,chs)` returns indices of the Enhanced Physical Downlink Control Channel Demodulation Reference Signal (EPDCCH DM-RS) resource elements (RE) associated with an EPDCCH transmission candidate in a subframe. By default, `ind` is a column vector of indices in one-based linear indexing form. Use this form to directly index the EPDCCH DM-RS REs of an *N*-by-*M*-by-4 array that represents the subframe resource grid across the four possible EPDCCH antenna ports ($p$ = 107...110). You can also generate alternative index representations. The order of the indices is the same as required for the complex EPDCCH DM-RS symbols mapping. `lteEPDCCHDMRS` generates these symbols.

The indices are parameterized in terms of a configured PRB pair set which defines:

- the overall set of possible EPDCCH candidates and
- the aggregation of one or more consecutive enhanced control channel elements (ECCE). This aggregation identifies the specific EPDCCH instance that the DM-RS are associated with.

The DM-RS indices are created only for the specific PRB pairs and antenna ports that the corresponding EPDCCH is mapped to. They do not account for any external precoding operations.

For a localized EPDCCH transmission, the EPDCCH is associated with a single antenna port from $p$ = 107...110, dependent on the `RNTI` and ECCEs selected. Thus, the DM-RS antenna port indices (1...4 respectively, if one-based) are output for that single port.

For a distributed transmission, the EPDCCH is mapped to two antenna ports in an alternating fashion. Therefore, the DM-RS indices are generated for the PRBs in both ports: $p$ = 107,109 for normal cyclic prefix and $p$ = 107,108 for extended cyclic prefix. The output is ordered so that the symbols for the lowest antenna index plane come first. These indices are suitable for indexing an *N*-by-*M*-by-4 array representing the subframe resource grid across the four possible EPDCCH antenna ports ($p$ = 107...110).

This syntax returns an NRE length column vector of one-based linear indices for the DM-RS resource elements associated with a particular EPDCCH candidate. The function is initialized according to the cell-wide settings, `enb`, and the EPDCCH transmission configuration, `chs`.

`ind = lteEPDCCHDMRSIndices(enb,chs,opts)` formats the returned indices using options specified by `opts`.

## Examples

### Generate EPDCCH DM-RS Indices

Specify the cell-wide settings and channel transmission configuration in parameter structures `enb` and `chs`.

```
enb = struct('CyclicPrefix','Normal','DuplexMode','FDD');
enb.NDLRB = 6;
enb.NSubframe = 0;
chs.EPDCCHECCE = [0 7];
chs.EPDCCHType = 'Localized';
chs.EPDCCHPRBSet = 2:3;
chs.RNTI = 1;
```

Create the EPDCCH DM-RS indices for an EPDCCH having eight ECCEs.

```
ind = lteEPDCCHDMRSIndices(enb,chs)
```

```
ind = 12x1 uint32 column vector

   1898
   1903
   1908
   1910
   1915
   1920
   1970
   1975
   1980
   1982
      ⋮
```

## Input Arguments

### enb — eNodeB cell-wide settings

structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NDLRB** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **CyclicPrefix** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **NSubframe** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **DuplexMode** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as one of the following:<br>• `'FDD'` — Frequency division duplex (default)<br>• `'TDD'` — Time division duplex |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| The following parameters apply when `DuplexMode` is set to `'TDD'`. | | | |
| **`TDDConfig`** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **`SSC`** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |

### `chs` — Channel-specific transmission configuration

structure

Channel-specific transmission configuration, specified as a structure that can contain the following parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`EPDCCHECCE`** | Required | 1-element or 2-element vector specifying the 0-based ECCE index or inclusive `[begin, end]` ECCE index range according to the aggregation level `L` (`L = end − begin + 1`). The number of ECCEs in the candidate must be a power of 2.<br><br>If no transmission is required, leave this parameter empty. | The set of one or several consecutive ECCEs defining the EPDCCH transmission candidate in the overall EPDCCH set. |
| **`EPDCCHType`** | Required | `'Localized'`, `'Distributed'` | EPDCCH transmission type |
| **`EPDCCHPRBSet`** | Required | Vector of zero-based indices for the PRB pairs corresponding to the EPDCCH PRB set. The number of PRB pair indices must be a power of 2.<br><br>If no transmission is required, leave this parameter empty. | EPDCCH PRB pair indices |
| The following parameters apply when `EPDCCHType` is set to `'Localized'`. | | | |
| **`RNTI`** | Required | 0 (default), scalar integer | Radio network temporary identifier (RNTI) value (16 bits) |

### `opts` — Index generation options

character vector | cell array of character vectors | string array

Index generation options, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string):

| Option | Values | Description |
|---|---|---|
| Indexing style | `'ind'` (default), `'mat'`, `'sub'` | Style for the returned indices, specified as one of the following:<br>• `'ind'` — returns the indices in linear index form as a column vector (default)<br>• `'mat'` — returns the indices in linear index form as a matrix, where each column contains indices for an individual port.<br>• `'sub'` — returns the indices in `[subcarrier, symbol, antenna]` subscript row style. The number of rows in the output, `ind`, is the number of resource elements (NRE). Thus, `ind` is an NRE-by-3 matrix. |
| Index base | `'1based'` (default), `'0based'` | Base value of the returned indices. Specify `'1based'` to generate indices where the first value is 1. Specify `'0based'` to generate indices where the first value is 0. |
| Indexing format | `'rsonly'` (default), `'rs+unused'` | RE locations mode of the returned indices.<br>• `'rsonly'` — returns only active DM-RS locations (default)<br>• `'rs+unused'` — also includes all RE locations, which should be unused because of DM-RS transmission on other EPDCCH antenna ports $p$ = 107...110 that are not used by this EPDCCH transmission |

Example: `'ind rsonly'`, `"ind rsonly"`, `{'ind','rsonly'}`, or `["ind","rsonly"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `ind` — EPDCCH DM-RS RE indices

numeric column vector | numeric matrix

EPDCCH DM-RS resource element indices, returned by default as a numeric vector of length NRE-by-1. Optionally, for subscript-specific indexing style [`subcarrier`, `symbol`, `antenna`], `ind` is returned as an NRE-by-3 numeric matrix. NRE is the number of subframe resource elements. You can also return the indices in a linear indexing matrix, where each column contains indices for an individual antenna port. By default, the indices are returned in one-based linear indexing form, which you can use to directly index the EPDCCH DM-RS resource elements.

Data Types: `double`

# Version History

**Introduced in R2014b**

## See Also

`lteEPDCCHDMRS` | `lteEPDCCHIndices` | `lteDMRSIndices`

# lteEPDCCHDecode

Enhanced physical downlink control channel (EPDCCH) decoding

## Syntax

```
[bits,symbols] = lteEPDCCHDecode(enb,chs,sym)
[bits,symbols] = lteEPDCCHDecode(enb,chs,rxsym,hest,noiseest)
[bits,symbols] = lteEPDCCHDecode(enb,chs,rxsym,hest,noiseest,alg)

[bits,symbols] = lteEPDCCHDecode(enb,chs,grid)
[bits,symbols] = lteEPDCCHDecode(enb,chs,rxgrid,hestgrid,noiseest,alg)
```

## Description

`[bits,symbols] = lteEPDCCHDecode(enb,chs,sym)` returns softbits and received constellation of complex symbols resulting from performing the inverse of enhanced physical downlink control channel (EPDCCH) processing of a single configured EPDCCH candidate given cell-wide settings structure, EPDCCH transmission configuration structure, and EPDCCH symbols. The input symbols are assumed to contain ideal EPDCCH symbols, so no equalization is performed. The output received EPDCCH symbols are QPSK symbol demodulated and descrambled. For more EPDCCH processing information, see `lteEPDCCH` and TS 36.211[1], Section 6.8A.

When using this syntax, the input structures only require `enb.NSubframe` and `chs.EPDCCHNID`.

For more information, see “Syntax Dependent Processing” on page 2-251.

`[bits,symbols] = lteEPDCCHDecode(enb,chs,rxsym,hest,noiseest)` performs EPDCCH decoding and equalization for a single configured EPDCCH candidate given cell-wide settings structure, EPDCCH transmission configuration structure, received EPDCCH symbols `rxsym`, channel estimate `hest`, and noise estimate `noiseest`. The output received EPDCCH symbols are equalized, and QPSK symbol demodulated and descrambled.

`[bits,symbols] = lteEPDCCHDecode(enb,chs,rxsym,hest,noiseest,alg)` performs EPDCCH decoding and equalization for a single configured EPDCCH candidate and provides control over weighting the output soft bits with channel state information (CSI) calculated during the equalization stage using algorithmic configuration structure, `alg`.

`[bits,symbols] = lteEPDCCHDecode(enb,chs,grid)` performs EPDCCH decoding for all possible EPDCCH candidate locations given cell-wide settings structure, EPDCCH transmission configuration structure, and the resource element grid across all possible EPDCCH antenna ports. The resource element grid is assumed to contain ideal EPDCCH REs, so no equalization is performed. The decoding consists of extraction of all EPDCCH REs from `grid` followed by QPSK symbol demodulation. Each EPDCCH candidate is descrambled individually during EPDCCH search. For this syntax, `chs.EPDCCHECCE` and `chs.EPDCCHNID` are not required as no candidate-specific resource extraction or descrambling is performed.

`[bits,symbols] = lteEPDCCHDecode(enb,chs,rxgrid,hestgrid,noiseest,alg)` performs EPDCCH decoding and equalization for all possible EPDCCH candidate locations given cell-wide settings structure, EPDCCH transmission configuration structure, received resource element grid, channel estimate grid, noise estimate, and provides control over weighting the output soft bits with

channel state information (CSI) calculated during the equalization stage using algorithmic configuration structure, `alg`. EPDCCH RE locations extracted from `rxgrid` and `hestgrid` are equalized, then QPSK symbol demodulated. Each EPDCCH candidate is descrambled individually during EPDCCH search. For this syntax, `chs.EPDCCHECCE` and `chs.EPDCCHNID` are not required as no candidate-specific resource extraction or descrambling is performed.

## Examples

### Decode EPDCCH codeword

Modulate and then demodulate EPDCCH symbols for a codeword of random bits.

Initialize the cell-wide settings structure and the EPDCCH transmission channel configuration structure.

```
enb.NSubframe = 0;
chs.EPDCCHNID = 1;
```

Create an input codeword for EPDCCH and generate EPDCCH symbols.

```
cw = randi([0 1],108,1);
sym = lteEPDCCH(enb,chs,cw);
```

Decode the symbols and confirm the codeword was successfully recovered.

```
rxcw = lteEPDCCHDecode(enb,chs,sym);
isequal(cw,rxcw>0)
```

```
ans = logical
   1
```

### Decode EPDCCH DCI Message

Perform DCI coding to the capacity of a particular EPDCCH candidate. EPDCCH modulate the coded message and transmit it. Add EPDCCH demodulation reference signals (DMRS) and perform channel estimation. Finally, extract the EPDCCH (and corresponding channel estimate) from the resource grid. Perform EPDCCH demodulation and decode the received DCI message.

Initialize the cell-wide settings structure.

```
enb.NSubframe = 0;
enb.NDLRB = 15;
enb.CyclicPrefix = 'Extended';
enb.CellRefP = 2;
enb.NCellID = 1;
enb.CFI = 1;
```

Initialize the EPDCCH transmission channel configuration structure.

```
chs.EPDCCHNID = 1;
chs.EPDCCHPRBSet = (0:3).';
chs.EPDCCHType = 'Localized';
```

```
chs.EPDCCHFormat = 2;
chs.ControlChannelType = 'EPDCCH';
chs.DCIFormat = 'Format2D';
chs.RNTI = 11;
```

Create a set of random bits representing a DCI message and performing DCI coding to the capacity of a particular EPDCCH candidate.

```
dciInfo = lteDCIInfo(enb,chs);
dcibits = randi([0 1],dciInfo.(chs.DCIFormat),1);
candidates = lteEPDCCHSpace(enb,chs);
chs.EPDCCHECCE = candidates(1,:);
[ind,info] = lteEPDCCHIndices(enb,chs);
cw = lteDCIEncode(chs,dcibits,info.EPDCCHG);
```

Generate EPDCCH symbols and resource element grid. Populate the grid.

```
sym = lteEPDCCH(enb,chs,cw);
grid = lteDLResourceGrid(enb,4);
grid(ind) = sym;
grid(lteEPDCCHDMRSIndices(enb,chs)) = lteEPDCCHDMRS(enb,chs);
```

Generate channel estimate.

```
cec.PilotAverage = 'TestEVM';
cec.Reference = 'EPDCCHDMRS';
[hestgrid,noiseest] = lteDLChannelEstimate(enb,chs,cec,grid);
[rxsym,hest] = lteExtractResources(ind,grid,hestgrid);
```

Decode the symbols and DCI message bits. Confirm the DCI message was successfully recovered.

```
rxcw = lteEPDCCHDecode(enb,chs,rxsym,hest,noiseest);
rxdcibits = lteDCIDecode(dciInfo.(chs.DCIFormat),rxcw);
isequal(dcibits,rxdcibits>0)
```

```
ans = logical
   1
```

## Input Arguments

### enb — eNodeB cell-wide settings

structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NDLRB** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **NCellID** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **CyclicPrefix** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`CellRefP`** | Required | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **`NSubframe`** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| The following parameter is only read when `chs.EPDCCHStart` is not present. | | | |
| **`CFI`** | Required | 1, 2, or 3<br>Scalar or if the CFI varies per subframe, a vector of length 10 (corresponding to a frame). | Control format indicator (`CFI`) value. In TDD mode, `CFI` varies per subframe for the RMCs (`'R.0'`, `'R.5'`, `'R.6'`, `'R.6-27RB'`, `'R.12-9RB'`) |
| **`DuplexMode`** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as one of the following:<br>• `'FDD'` — Frequency division duplex (default)<br>• `'TDD'` — Time division duplex |
| The following parameters apply when `DuplexMode` is set to `'TDD'`. | | | |
| **`TDDConfig`** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **`SSC`** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |
| **`NFrame`** | Optional | 0 (default), nonnegative scalar integer | Frame number |
| **`CSIRSPeriod`** | Optional | `'Off'` (default), `'On'`, `Icsi-rs` (0,...,154), `[Tcsi-rs Dcsi-rs]`. You can also specify values in a cell array of configurations for each resource. | CSI-RS subframe configurations for one or more CSI-RS resources. Multiple CSI-RS resources can be configured from a single common subframe configuration or from a cell array of configurations for each resource. |
| The following CSI-RS resource parameters apply only when `CSIRSPeriod` sets one or more CSI-RS subframe configurations to any value other than `'Off'`. Each parameter length must be equal to the number of CSI-RS resources required. | | | |
| **`CSIRSConfig`** | Required | Nonnegative scalar integer | Array CSI-RS configuration indices. See TS 36.211, Table 6.10.5.2-1. |
| **`CSIRefP`** | Required | 1 (default), 2, 4, 8 | Array of number of CSI-RS antenna ports |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **ZeroPowerCSIRSPeriod** | Optional | `'Off'` (default), `'On'`, `Icsi-rs` (0,...,154), `[Tcsi-rs Dcsi-rs]`. You can also specify values in a cell array of configurations for each resource. | Zero power CSI-RS subframe configurations for one or more zero power CSI-RS resource configuration index lists. Multiple zero power CSI-RS resource lists can be configured from a single common subframe configuration or from a cell array of configurations for each resource list. |
| The following zero power CSI-RS resource parameter is only applicable if one or more of the above zero power subframe configurations are set to any value other than `'Off'`. | | | |
| **ZeroPowerCSIRSConfig** | Required | 16-bit bitmap character vector or string scalar (truncated if not 16 bits or `'0'` MSB extended), or a numeric list of CSI-RS configuration indices. You can also specify values in a cell array of configurations for each resource. | Zero power CSI-RS resource configuration index lists (TS 36.211 Section 6.10.5.2). Specify each list as a 16-bit bitmap character vector or string scalar (if less than 16 bits, then `'0'` MSB extended), or as a numeric list of CSI-RS configuration indices from TS 36.211 Table 6.10.5.2-1 in the `'4'` CSI reference signal column. Multiple lists can be defined using a cell array of individual lists. |

**`chs` — EPDCCH-specific channel transmission configuration**
structure

EPDCCH-specific channel transmission configuration, specified as a structure that can contain the following parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **EPDCCHECCE** | Required | 1- or 2- element vector specifying the zero-based ECCE index or inclusive [*begin*, *end*] ECCE index range according to the aggregation level `L`, where *L = end – begin + 1*. The number of ECCEs in the candidate must be a power of 2.<br><br>If no transmission is required, leave this parameter empty. | The set of one of several consecutive ECCEs defining the EPDCCH transmission candidate in the overall EPDCCH set. |
| **EPDCCHType** | Required | `'Localized'`, `'Distributed'` | EPDCCH transmission type |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **EPDCCHPRBSet** | Required | Vector of zero-based indices for the PRB pairs corresponding to the EPDCCH PRB set. The number of PRB pair indices must be a power of 2.<br><br>If no transmission is required, leave this parameter empty. | EPDCCH PRB pair indices |
| **EPDCCHStart** | Optional | integer from 0 to 4<br><br>If this parameter is not present, then the cell-wide CFI parameter is used for the starting symbol. | EPDCCH starting symbol |
| **EPDCCHNID** | Required | nonnegative scalar integer | EPDCCH scrambling sequence initialization |
| The following parameter applies when `EPDCCHType` is set to `'Localized'`. | | | |
| **RNTI** | Required | 0 (default), scalar integer | Radio network temporary identifier (RNTI) value (16 bits) |

**`sym` — EPDCCH modulation symbols**
complex-vector

`EPDCCH` modulation symbols associated with a single `EPDCCH` transmission in a subframe, specified as a complex vector. This input contains the QPSK symbols.

Data Types: `double`

**`rxsym` — Received EPDCCH symbols**
*EPDCCHGd*-by-*NRxAnts* matrix

Received EPDCCH symbols, specified as a *EPDCCHGd*-by-*NRxAnts* matrix. *EPDCCHGd* is the EPDCCH symbol capacity, given by the `info.EPDCCHGd` field of `lteEPDCCHIndices`. *NRxAnts* is the number of receive antennas. This matrix contains the elements of the received resource grid in the locations of the EPDCCH REs for the candidate configured via `chs.EPDCCHECCE`.

Data Types: `double`

**`hest` — Channel estimate**
*EPDCCHGd*-by-*NRxAnts*-by-4 array

Channel estimate, specified as an *EPDCCHGd*-by-*NRxAnts* array. *EPDCCHGd* is the EPDCCH symbol capacity, given by the `info.EPDCCHGd` field of `lteEPDCCHIndices`. *NRxAnts* is the number of receive antennas. The third dimension represents the 4 possible EPDCCH antenna ports (p=107...110). This array contains the elements of the channel estimate array in the locations of the EPDCCH REs for the candidate configured via `chs.EPDCCHECCE`.

Data Types: `double`

**`noiseest` — Noise estimate**
numeric scalar

Noise estimate of the noise power spectral density per RE on the received subframe, specified as a numeric scalar. The `lteDLChannelEstimate` function provides this estimate.

Data Types: `double`

**`alg` — Algorithmic configuration**
structure

Algorithmic configuration, specified as a structure. The structure must have the field:

| | | | |
|---|---|---|---|
| **`CSI`** | Optional | `'On'` (default), `'Off'` | Flag provides control over weighting the soft values that are used to determine the output values with the channel state information (CSI) calculated during the equalization process |

Data Types: `double`

**`grid` — Resource grid**
*K*-by-*L*-by-4 array

Resource grid across the four possible EPDCCH ports, specified as a *K*-by-*L*-by-4 array. *K* is the number of subcarriers, *L* is the number of OFDM symbols in one frame, and 4 is all possible EPDCCH antenna ports (p=107...110).

Data Types: `double`

**`rxgrid` — Received resource elements grid**
*K*-by-*L*-by-*NRxAnts* array

Received resource elements grid, specified as a *K*-by-*L*-by-*NRxAnts* array. *K* is the number of subcarriers, *L* is the number of OFDM symbols in one frame, and *NRxAnts* is the number of receive antennas.

Data Types: `double`

**`hestgrid` — Channel estimate grid**
*K*-by-*L*-by-*NRxAnts*-by-4 array

Channel estimate grid, specified as a *K*-by-*L*-by-*NRxAnts*-by-4 array. *K* is the number of subcarriers, *L* is the number of OFDM symbols in one frame, and *NRxAnts* is the number of receive antennas. The 4th dimension represents the 4 possible `EPDCCH` antenna ports (p=107...110).

Data Types: `double`

## Output Arguments

**`bits` — Decoded bit estimates**
column-vector | $M_{Tot}$-by-4 matrix

Decoded bit estimates for the candidate configured via `chs.EPDCCHECCE`, returned as one of the following:

- a column-vector of length *EPDCCHG* = *EDPCCHGd* × 2.
- $M_{Tot}$-by-4 matrix. $M_{Tot}$ is the total number of bits associated with EPDCCHs and 4 is all possible EPDCCH antenna ports (p=107...110). Since the `bits` output is used as input to `lteEPDCCHSearch`, where each ECCE candidate has to be descrambled individually, the `bits` output are not descrambled.

**`symbols` — Received QPSK symbols**
column-vector | ($M_{Tot}$ / 2)-by-4 matrix

Received QPSK symbols corresponding to bits in `bits`, specified as one of the following:.

- A column-vector of length *EPDCCHGd*, where *EPDCCHGd* is the EPDCCH symbol capacity, given by the `info.EPDCCHGd` field of `lteEPDCCHIndices`.
- ($M_{Tot}$ / 2)-by-4 matrix, for all `EPDCCH ECCEs` and all `4 EPDCCH` reference signal ports (p=107...110).

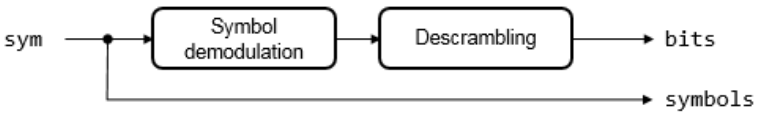
## More About

### Syntax Dependent Processing

The `lteEPDCCHDecode` function works with only one EPDCCH-PRB-Set because `lteDLChannelEstimate` works with only one EPDCCH-PRB-Set. The `lteEPDCCHDecode` function processing performed depends on which input signals are provided to the function. The figures shown here align available syntaxes with processing performed.

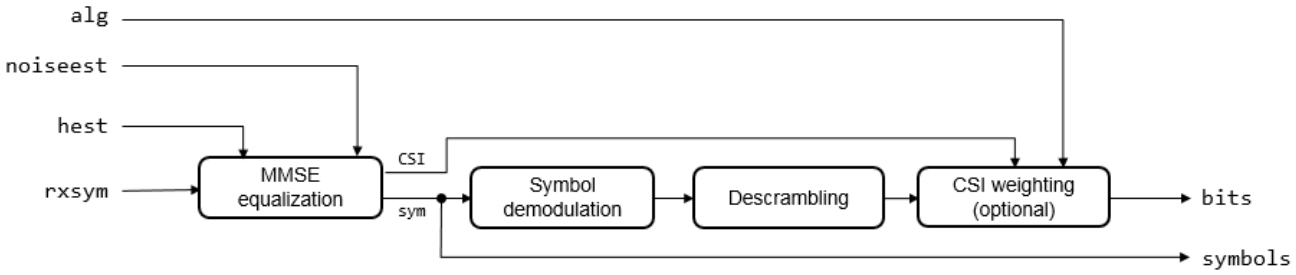
If the symbols for a single configured EPDCCH candidate are input, the function performs symbol demodulation and descrambling. The function assumes the input symbols were already equalized.

```
[bits,symbols] = lteEPDCCHDecode(enb,chs,sym)
```



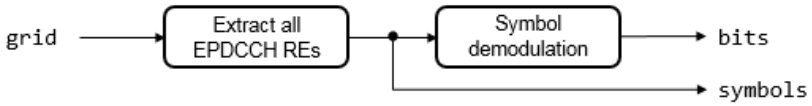

If the symbols for a single configured EPDCCH candidate are input along with channel and noise estimates, the function performs MMSE equalization, then symbol demodulation and descrambling. If the optional `alg` input is provided, CSI weighting is applied to the output bits.

```
[bits,symbols] = lteEPDCCHDecode(enb,chs,rxsym,hest,noiseest,alg)
```



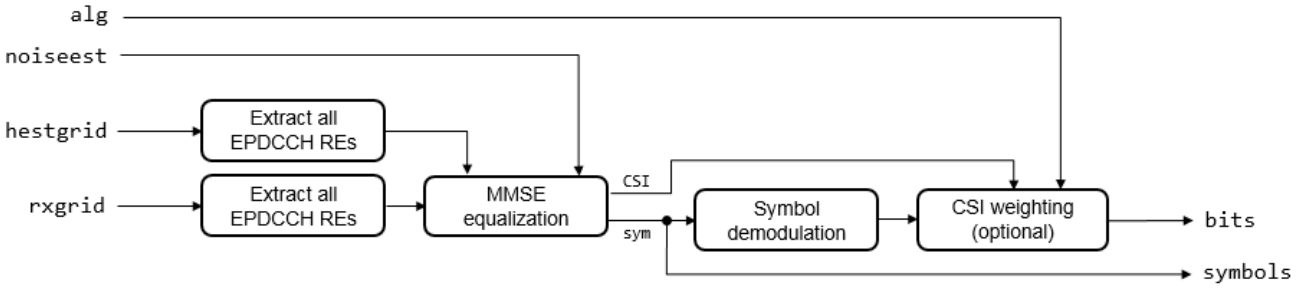

If the resource element grid across all possible EPDCCH antenna ports is input, the function extracts all EPDCCH resource elements and performs EPDCCH decoding for all possible EPDCCH candidate locations. The function assumes the input symbols were already equalized. Each EPDCCH candidate is descrambled individually during EPDCCH search.

```
[bits,symbols] = lteEPDCCHDecode(enb,chs,grid)
```




If the resource element grid is input, along with channel and noise estimates, the function extracts all EPDCCH resource elements and performs MMSE equalization, then symbol demodulation. If the optional `alg` input is provided, CSI weighting is applied to the output bits. Each EPDCCH candidate is descrambled individually during EPDCCH search.

```
[bits,symbols] = lteEPDCCHDecode(enb,chs,rxgrid,hestgrid,noiseest,alg)
```




## Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteEPDCCH` | `lteEPDCCHDMRSIndices` | `lteEPDCCHIndices` | `lteEPDCCHSearch` | `lteEPDCCHSpace` | `lteEPDCCHPRBS` | `lteDCIDecode`

# lteEPDCCHSearch

Enhanced downlink control information search

## Syntax

```
[dcistr,dcibits] = lteEPDCCHSearch(enb,chs,softbits)
```

## Description

`[dcistr,dcibits] = lteEPDCCHSearch(enb,chs,softbits)` recovers DCI message structures, and corresponding vectors of DCI message bits, after blind decoding the multiplexed EPDCCHs. The multiplexed EPDCCHs are within the received EPDCCH payload given by matrix of soft bits. This function carries out search for a single EPDCCH set. For more information, see “EPDCCH Search Processing” on page 2-265.

## Examples

### Search EPDCCH for DCI Messages

Encode a DCI message and modulate it on the EPDCCH. Perform EPDCCH decoding and then EPDCCH blind-search to recover the DCI message. For `DCI` messages sent on the EPDCCH, set the ControlChannelType to `'EPDCCH'`.

Initialize cell-wide settings structure and EPDCCH transmission channel structure.

```
enb = lteRMCDL('R.43');
chs.RNTI = 42;
chs.ControlChannelType = 'EPDCCH';
chs.EPDCCHType = 'Localized';
chs.EPDCCHPRBSet = [2 3];
chs.EPDCCHNID = 0;
chs.EPDCCHFormat = 1;
chs.DCIFormat = 'Format1A';
```

Create a DCI message. Generate EPDCCH candidates.

```
[dci,dciBits] = lteDCI(enb,chs,struct('DCIFormat',chs.DCIFormat));

candidates = lteEPDCCHSpace(enb,chs);
chs.EPDCCHECCE = candidates(1,:);
```

Generate RE grid indices and EPDCCH info structure. Encode the DCI message into a codeword for transmission. Generate EPDCCH symbols and populate resource grid.

```
[ind,info] = lteEPDCCHIndices(enb,chs);
cw = lteDCIEncode(chs,dciBits,info.EPDCCHG);

sym = lteEPDCCH(enb,chs,cw);
grid = lteDLResourceGrid(enb,4);
grid(ind) = sym;
```

Decode the EPDCCH transmission. Recover and view DCI message.

```
rxsoftbits = lteEPDCCHDecode(enb,chs,grid);

rxdci = lteEPDCCHSearch(enb,chs,rxsoftbits);
rxdci{1}
```

```
ans = struct with fields:
           DCIFormat: 'Format1A'
                 CIF: 0
      AllocationType: 0
          Allocation: [1x1 struct]
           ModCoding: 0
              HARQNo: 0
             NewData: 0
                  RV: 0
            TPCPUCCH: 0
            TDDIndex: 0
          SRSRequest: 0
    HARQACKResOffset: 0
```

Search for multiple EPDCCH sets. The first EPDCCH set is as configured above and the second is of Distributed type with 8 PRBs.

Transmit the EPDCCH DM-RS for channel estimation.

```
grid(lteEPDCCHDMRSIndices(enb,chs)) = lteEPDCCHDMRS(enb,chs);
```

Configure the channel estimation.

```
cec.PilotAverage = 'TestEVM';
cec.Reference = 'EPDCCHDMRS';
```

Configure two EPDCCH sets.

```
chs.EPDCCHTypeList = {'Localized' 'Distributed'};
chs.EPDCCHPRBSetList = {[2; 3] (8:15).'};
```

Perform the EPDCCH search for each set.

```
for p = 1:numel(chs.EPDCCHTypeList)
    chs.EPDCCHType = chs.EPDCCHTypeList{p};
    chs.EPDCCHPRBSet = chs.EPDCCHPRBSetList{p};
    [hestgrid,noiseest] = lteDLChannelEstimate(enb,chs,cec,grid);
    rxsoftbits = lteEPDCCHDecode(enb,chs,grid,hestgrid,noiseest);
    rxdci = lteEPDCCHSearch(enb,chs,rxsoftbits);
    X = ['EPDCCH set ' num2str(p)];
    disp([X ', DCI messages found: ' num2str(numel(rxdci))])
    if (~isempty(rxdci))
        rxdci{1}
    end
end
```

```
EPDCCH set 1, DCI messages found: 1

ans = struct with fields:
           DCIFormat: 'Format1A'
                 CIF: 0
```

```
      AllocationType: 0
          Allocation: [1x1 struct]
           ModCoding: 0
              HARQNo: 0
             NewData: 0
                  RV: 0
            TPCPUCCH: 0
            TDDIndex: 0
          SRSRequest: 0
    HARQACKResOffset: 0

EPDCCH set 2, DCI messages found: 0
```

A DCI message is found in EPDCCH set 1 but not in EPDCCH set 2.

## Input Arguments

### enb — eNodeB cell-wide settings

structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NDLRB** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$)<br><br>See “Specifying Number of Resource Blocks” on page 2-265. |
| **NULRB** | Required | Scalar integer from 6 to 110 | Number of uplink resource blocks. ($N_{RB}^{UL}$) |
| **NCellID** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **CyclicPrefix** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **CellRefP** | Required | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **NSubframe** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| The following parameter is applicable only when `chs.EPDCCHStart` is absent. | | | |
| **CFI** | Required | 1, 2, or 3<br>Scalar or if the CFI varies per subframe, a vector of length 10 (corresponding to a frame). | Control format indicator (`CFI`) value. In TDD mode, `CFI` varies per subframe for the RMCs (`'R.0'`, `'R.5'`, `'R.6'`, `'R.6-27RB'`, `'R.12-9RB'`) |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`DuplexMode`** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as one of the following:<br>• `'FDD'` — Frequency division duplex (default)<br>• `'TDD'` — Time division duplex |
| The following parameters apply when `DuplexMode` is set to `'TDD'`. | | | |
| **`TDDConfig`** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **`SSC`** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |
| **`NFrame`** | Optional | 0 (default), nonnegative scalar integer | Frame number |
| **`CSIRSPeriod`** | Optional | `'Off'` (default), `'On'`, `Icsi-rs` (0,...,154), `[Tcsi-rs Dcsi-rs]`. You can also specify values in a cell array of configurations for each resource. | CSI-RS subframe configurations for one or more CSI-RS resources. Multiple CSI-RS resources can be configured from a single common subframe configuration or from a cell array of configurations for each resource. |
| The following CSI-RS resource parameters apply only when `CSIRSPeriod` sets one or more CSI-RS subframe configurations to any value other than `'Off'`. Each parameter length must be equal to the number of CSI-RS resources required. | | | |
| **`CSIRSConfig`** | Required | Nonnegative scalar integer | Array CSI-RS configuration indices. See TS 36.211, Table 6.10.5.2-1. |
| **`CSIRefP`** | Required | 1 (default), 2, 4, 8 | Array of number of CSI-RS antenna ports |
| **`ZeroPowerCSIRSPeriod`** | Optional | `'Off'` (default), `'On'`, `Icsi-rs` (0,...,154), `[Tcsi-rs Dcsi-rs]`. You can also specify values in a cell array of configurations for each resource. | Zero power CSI-RS subframe configurations for one or more zero power CSI-RS resource configuration index lists. Multiple zero power CSI-RS resource lists can be configured from a single common subframe configuration or from a cell array of configurations for each resource list. |
| The following zero power CSI-RS resource parameter is applicable only if one or more of the above zero power subframe configurations are set to any value other than `'Off'`. | | | |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **ZeroPowerCSIRSConfig** | Required | 16-bit bitmap character vector or string scalar (truncated if not 16 bits or `'0'` MSB extended), or a numeric list of CSI-RS configuration indices. You can also specify values in a cell array of configurations for each resource. | Zero power CSI-RS resource configuration index lists (TS 36.211 Section 6.10.5.2). Specify each list as a 16-bit bitmap character vector or string scalar (if less than 16 bits, then `'0'` MSB extended), or as a numeric list of CSI-RS configuration indices from TS 36.211 Table 6.10.5.2-1 in the `'4'` CSI reference signal column. Multiple lists can be defined using a cell array of individual lists. |

**chs — EPDCCH-specific channel transmission configuration**
structure

EPDCCH-specific channel transmission configuration, specified as a structure that can contain the following parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **EPDCCHType** | Required | `'Localized'`, `'Distributed'` | EPDCCH transmission type |
| **EPDCCHPRBSet** | Required | Vector of zero-based indices for the PRB pairs corresponding to the EPDCCH PRB set. The number of PRB pair indices must be a power of 2. | EPDCCH PRB pair indices |
| **EPDCCHStart** | Optional | integer from 0 to 4<br><br>If this parameter is not present, then the cell-wide CFI parameter is used for the starting symbol. | EPDCCH starting symbol |
| **RNTI** | Required | 0 (default), scalar integer | Radio network temporary identifier (RNTI) value (16 bits) |
| **EPDCCHNID** | Required | nonnegative scalar integer | EPDCCH nID parameter for scrambling sequence initialization. |
| **EPDCCHPRBSetList** | Optional | cell array of one or two vectors | PRB pair indices for one or two EPDCCH sets. |
| **EPDCCHPRBTypeList** | Optional | cell array of character vector or string array | EPDCCH transmission types for one or two EPDCCH sets. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **EnableCarrierIndication** | Optional | 'Off' (default), 'On' | UE configured with carrier indication field (affects presence of CIF) |
| **EnableSRSRequest** | Optional | 'Off' (default), 'On' | UE configured for SRS request (affects presence of SRS request field in UE-specific formats 0/1A and 2B/2C/2D TDD) |
| **EnableMultipleCSIRequest** | Optional | 'Off' (default), 'On' | UE configured for multiple CSI requests (multiple cells/CSI processes) (affects length of CSI request field in UE-specific formats 0/4) |
| **NTxAnts** | Optional | 1 (default), 2, 4 | Number of UE transmission antennas (affects length of precoding information field in format 4) |

**`softbits` — Received EPDCCH payload**
*MTot*-by-4 matrix

Received EPDCCH payload containing coded Downlink Control Information (DCI), specified as a *MTot*-by-4 matrix. *MTot* is the total number of bits associated with EPDCCHs, $\frac{nEPDCCH*NECCE}{NECCEPerPRB*2}$. The matrix contains soft EPDCCH bits estimates for all EPDCCH ECCEs and all EPDCCH reference signal ports.

If `chs.EPDCCHPRBSetList` and `chs.EPDCCHTypeList` are present and each contain two elements, the creation of the EPDCCH candidate locations support two EPDCCH sets. For more information, see TS 36.213 [2], Tables 9.1.4-3a to 9.1.4-5b.

Data Types: `double`

## Output Arguments

**`dcistr` — DCI message structure**
cell array of structures

DCI message structure, returned as a cell array of structures whose fields match of the fields associated DCI format.

The field names associated with `dcistr` depend on the DCI format. The format is expected to be one of the formats generated by `lteDCI`.

The following table details the DCI formats and accompanying `dcistr` parameter fields.

| DCI Formats | DCISTRFields | Size | Description |
|---|---|---|---|
| 'Format0' | DCIFormat | — | 'Format0' |
| | FreqHopping | 1-bit | PUSCH frequency hopping flag |
| | Allocation | variable | Resource block assignment/allocation |
| | ModCoding | 5-bits | Modulation, coding scheme, and redundancy version |
| | NewData | 1-bit | New data indicator |
| | TPC | 2-bits | PUSCH TPC command |
| | CShiftDMRS | 3-bits | Cyclic shift for DM RS |
| | CQIReq | 1-bit | CQI request |
| | TDDIndex | 2-bits | For TDD config 0, this field is the Uplink Index.<br><br>For TDD Config 1-6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| 'Format1' | DCIFormat | — | 'Format1' |
| | AllocationType | 1-bit | Resource allocation header: type 0, type 1 (only if downlink bandwidth is >10 PRBs) |
| | Allocation | variable | Resource block assignment/allocation |
| | ModCoding | 5-bits | Modulation and coding scheme |
| | HARQNo | 3-bits (FDD)<br><br>4-bits (TDD) | HARQ process number |
| | NewData | 1-bit | New data indicator |
| | RV | 2-bits | Redundancy version |
| | TPCPUCCH | 2-bits | PUCCH TPC command |
| | TDDIndex | 2-bits | For TDD config 0, this field is not used. For TDD Config 1-6, this field is the Downlink Assignment Index. Not present for FDD. |
| 'Format1A' | DCIFormat | — | 'Format1A' |
| | AllocationType | 1-bit | VRB assignment flag: 0 (localized), 1 (distributed) |
| | Allocation | variable | Resource block assignment/allocation |
| | ModCoding | 5-bits | Modulation and coding scheme |
| | HARQNo | 3-bits (FDD)<br><br>4-bits (TDD) | HARQ process number |
| | NewData | 1-bit | New data indicator |
| | RV | 2-bits | Redundancy version |

| DCI Formats | DCISTRFields | Size | Description |
|---|---|---|---|
| | `TPCPUCCH` | 2-bits | PUCCH TPC command |
| | `TDDIndex` | 2-bits | For TDD config 0, this field is not used. For TDD Config 1-6, this field is the Downlink Assignment Index. Not present for FDD. |
| `'Format1B'` | `DCIFormat` | — | `'Format1B'` |
| | `AllocationType` | 1-bit | VRB assignment flag: 0 (localized), 1 (distributed) |
| | `Allocation` | variable | Resource block assignment/allocation |
| | `ModCoding` | 5-bits | Modulation and coding scheme |
| | `HARQNo` | 3-bits (FDD)<br>4-bits (TDD) | HARQ process number |
| | `NewData` | 1-bit | New data indicator |
| | `RV` | 2-bits | Redundancy version |
| | `TPCPUCCH` | 2-bits | PUCCH TPC command |
| | `TPMI` | 2-bits (2 antennas)<br>4-bits (4 antennas) | PMI information |
| | `PMI` | 1-bit | PMI confirmation |
| | `TDDIndex` | 2-bits | For TDD config 0, this field is not used. For TDD Config 1-6, this field is the Downlink Assignment Index. Not present for FDD. |
| `'Format1C'` | `DCIFormat` | — | `'Format1C'` |
| | `Allocation` | variable | Resource block assignment/allocation |
| | `ModCoding` | 5-bits | Modulation and coding scheme |
| `'Format1D'` | `DCIFormat` | — | `'Format1D'` |
| | `AllocationType` | 1-bit | VRB assignment flag: 0 (localized), 1 (distributed) |
| | `Allocation` | variable | Resource block assignment/allocation |
| | `ModCoding` | 5-bits | Modulation and coding scheme |
| | `HARQNo` | 3-bits (FDD)<br>4-bits (TDD) | HARQ process number |
| | `NewData` | 1-bit | New data indicator |
| | `RV` | 2-bits | Redundancy version |
| | `TPCPUCCH` | 2-bits | PUCCH TPC command |
| | `TPMI` | 2-bits (2 antennas)<br>4-bits (4 antennas) | Precoding TPMI information |

| DCI Formats | DCISTRFields | Size | Description |
|---|---|---|---|
| | `DlPowerOffset` | 1-bit | Downlink power offset |
| | `TDDIndex` | 2-bits | For TDD config 0, this field is not used. For TDD Config 1-6, this field is the Downlink Assignment Index. Not present for FDD. |
| `'Format2'` | `DCIFormat` | — | `'Format2'` |
| | `AllocationType` | 1-bit | Resource allocation header: type 0, type 1 (only if downlink bandwidth is >10 PRBs) |
| | `Allocation` | variable | Resource block assignment/allocation |
| | `TPCPUCCH` | 2-bits | PUCCH TPC command |
| | `HARQNo` | 3-bits (FDD)<br>4-bits (TDD) | HARQ process number |
| | `SwapFlag` | 1-bit | Transport block to codeword swap flag |
| | `ModCoding1` | 5-bits | Modulation and coding scheme for transport block 1 |
| | `NewData1` | 1-bit | New data indicator for transport block 1 |
| | `RV1` | 2-bits | Redundancy version for transport block 1 |
| | `ModCoding2` | 5-bits | Modulation and coding scheme for transport block 2 |
| | `NewData2` | 1-bit | New data indicator for transport block 2 |
| | `RV2` | 2-bits | Redundancy version for transport block 2 |
| | `PrecodingInfo` | 3-bits (2-antennas)<br>6-bits (4-antennas) | Precoding information |
| | `TDDIndex` | 2-bits | For TDD config 0, this field is not used. For TDD Config 1-6, this field is the Downlink Assignment Index. Not present for FDD. |
| `'Format2A'` | `DCIFormat` | — | `'Format2A'` |
| | `AllocationType` | 1-bit | Resource allocation header: type 0, type 1 (only if downlink bandwidth is >10 PRBs) |
| | `Allocation` | variable | Resource block assignment/allocation |
| | `TPCPUCCH` | 2-bits | PUCCH TPC command |
| | `HARQNo` | 3-bits (FDD)<br>4-bits (TDD) | HARQ process number |
| | `SwapFlag` | 1-bit | Transport block to codeword swap flag |

| DCI Formats | DCISTRFields | Size | Description |
|---|---|---|---|
| | `ModCoding1` | 5-bits | Modulation and coding scheme for transport block 1 |
| | `NewData1` | 1-bit | New data indicator for transport block 1 |
| | `RV1` | 2-bits | Redundancy version for transport block 1 |
| | `ModCoding2` | 5-bits | Modulation and coding scheme for transport block 2 |
| | `NewData2` | 1-bit | New data indicator for transport block 2 |
| | `RV2` | 2-bits | Redundancy version for transport block 2 |
| | `PrecodingInfo` | 0-bits (2 antennas)<br>2-bits (4 antennas) | Precoding information |
| | `TDDIndex` | 2-bits | For TDD config 0, this field is not used. For TDD Config 1-6, this field is the Downlink Assignment Index. Not present for FDD. |
| `'Format2B'` | `DCIFormat` | — | `'Format2B'` |
| | `AllocationType` | 1-bit | Resource allocation header: type 0, type 1 (only if downlink bandwidth is >10 PRBs) |
| | `Allocation` | variable | Resource block assignment/allocation |
| | `TPCPUCCH` | 2-bits | PUCCH TPC command |
| | `HARQNo` | 3-bits (FDD)<br>4-bits (TDD) | HARQ process number |
| | `ScramblingId` | 1-bit | Scrambling identity |
| | `ModCoding1` | 5-bits | Modulation and coding scheme for transport block 1 |
| | `NewData1` | 1-bit | New data indicator for transport block 1 |
| | `RV1` | 2-bits | Redundancy version for transport block 1 |
| | `ModCoding2` | 5-bits | Modulation and coding scheme for transport block 2 |
| | `NewData2` | 1-bit | New data indicator for transport block 2 |
| | `RV2` | 2-bits | Redundancy version for transport block 2 |
| | `TDDIndex` | 2-bits | For TDD config 0, this field is not used. For TDD Config 1-6, this field is the Downlink Assignment Index. Not present for FDD. |
| `'Format2C'` | `DCIFormat` | — | `'Format2C'` |

| DCI Formats | DCISTRFields | Size | Description |
|---|---|---|---|
| | CIF | variable | Carrier indicator |
| | AllocationType | 1-bit | Resource allocation header: type 0, type 1 (only if downlink bandwidth is >10 PRBs) |
| | Allocation | variable | Resource block assignment/allocation |
| | TPCPUCCH | 2-bits | PUCCH TPC command |
| | HARQNo | 3-bits (FDD)<br>4-bits (TDD) | HARQ process number |
| | TxIndication | 3-bits | Antenna port(s), scrambling identity, and number of layers indicator |
| | SRSRequest | variable | SRS request. Only present for TDD. |
| | ModCoding1 | 5-bits | Modulation and coding scheme for transport block 1 |
| | NewData1 | 1-bit | New data indicator for transport block 1 |
| | RV1 | 2-bits | Redundancy version for transport block 1 |
| | ModCoding2 | 5-bits | Modulation and coding scheme for transport block 2 |
| | NewData2 | 1-bit | New data indicator for transport block 2 |
| | RV2 | 2-bits | Redundancy version for transport block 2 |
| | TDDIndex | 2-bits | For TDD config 0, this field is not used.<br>For TDD Config 1-6, this field is the Downlink Assignment Index.<br>Not present for FDD. |
| 'Format3' | DCIFormat | — | 'Format3' |
| | TPCCommands | variable | TPC commands for PUCCH and PUSCH |
| 'Format3A' | DCIFormat | — | 'Format3A' |
| | TPCCommands | variable | TPC commands for PUCCH and PUSCH |
| 'Format4' | DCIFormat | — | 'Format4' |
| | CIF | variable | Carrier indicator |
| | Allocation | variable | Resource block assignment/allocation |
| | TPC | 2-bits | PUSCH TPC command |
| | CShiftDMRS | 3-bits | Cyclic shift for DMRS |
| | TDDIndex | 2-bits | For TDD config 0, this field is Uplink Index. For TDD Config 1-6, this field is the Downlink Assignment Index. Not present for FDD. |

| DCI Formats | DCISTRFields | Size | Description |
|---|---|---|---|
| | `CQIReq` | variable | CQI request |
| | `SRSRequest` | 2-bits | SRS request |
| | `AllocationType` | 1-bit | Resource allocation header: non-hopping PUSCH resource allocation type 0, type 1 |
| | `ModCoding` | 5-bits | Modulation, coding scheme and redundancy version |
| | `NewData` | 1-bit | New data indicator |
| | `ModCoding1` | 5-bits | Modulation and coding scheme for transport block 1 |
| | `NewData1` | 1-bit | New data indicator for transport block 1 |
| | `ModCoding2` | 5-bits | Modulation and coding scheme for transport block 2 |
| | `NewData2` | 1-bit | New data indicator for transport block 2 |
| | `PrecodingInfo` | 3-bits (2 antennas)<br>6-bits (4 antennas) | Precoding information |

Data Types: `struct`

**`dcibits` — Recovered DCI message bit vector**
cell array of vectors

Recovered DCI message bit vector, returned as a column vector. The length of `dcibits` is defined though structure `enb` in terms of the DCI message format and the bandwidth.

Data Types: `int8`

## More About

### EPDCCH Search Processing

EPDCCH search processing blindly decodes DCI messages based on their lengths. The lengths and order in which the DCI messages are searched for is provided by `lteDCIInfo`. For DCI messages conveyed on the EPDCCH, set `ControlChannelType` to `'EPDCCH'` when calling `lteDCIInfo`.

If one or more messages have the same length, the first message format in the list is used to decode the message. The other potential message formats are ignored. The `lteEPDCCHSearch` function does not consider transmission mode during blind search, and no DCI message format is filtered based on transmission mode. It does not search for format 3 and 3A (power adjustment commands for PUSCH and PUCCH). It also does not search for format 1C as this format is never used in the UE-specific search space. EPDCCH is never used for common search space messages. For more information on the association between transmission mode, transmission scheme, DCI format, and search space, see TS 36.213 [2], Section 7.1 and Table 7.1-5A.

### Specifying Number of Resource Blocks

The number of resource blocks specifies the uplink and downlink bandwidth. The LTE Toolbox implementation assumes symmetric link bandwidth unless you specifically assign different values to

`NULRB` and `NDLRB`. If the number of resource blocks is initialized in only one link direction, then the initialized number of resource blocks (`NULRB` or `NDLRB`) is used for both uplink and downlink. When this mapping is used, no warning is displayed. An error occurs if `NULRB` and `NDLRB` are both undefined.

# Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.213. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteEPDCCH` | `lteEPDCCHDecode` | `lteEPDCCHIndices` | `lteEPDCCHSpace` | `lteEPDCCHPRBS`

# lteEPDCCHSpace

EPDCCH search space candidates

## Syntax

```
[ind,info] = lteEPDCCHSpace(enb,chs)
```

## Description

`[ind,info] = lteEPDCCHSpace(enb,chs)` returns a matrix or cell array of EPDCCH ECCE candidate indices, and related dimensional information for the given cell-wide settings structure and EPDCCH transmission configuration structure. Depending on the configuration, the function returns a matrix of candidates for a single EPDCCH set, or a cell array containing one or two matrices of candidates for one or two EPDCCH sets.

## Examples

### EPDCCH Search Space

EPDCCH Search Space for DCI Format 2A and 1.

For a particular configuration, establish the sizes of DCI messages for format 2A and format 1. Note that for DCI messages conveyed on the EPDCCH, ControlChannelType should be set to `'EPDCCH'`.

```
enb.NDLRB = 50;
enb.CellRefP = 1;
enb.NCellID = 0;
enb.NSubframe = 0;
enb.CFI = 1;
chs.EPDCCHType = 'Localized';
chs.EPDCCHPRBSet = (0:3).';
chs.EPDCCHFormat = 1;
chs.RNTI = 7;
chs.ControlChannelType = 'EPDCCH';

dcisizes = lteDCIInfo(enb,chs);
format2Asize = dcisizes.Format2A
```

```
format2Asize = uint64
    42
```

```
format1size = dcisizes.Format1
```

```
format1size = uint64
    33
```

Create the EPDCCH search space candidates for a localized EPDCCH transmission of a DCI format 2A message.

```
chs.DCIFormat = 'Format2A';
[candidates,info] = lteEPDCCHSpace(enb,chs)
```

```
candidates = 4×2

     4     7
     8    11
    12    15
     0     3


info = struct with fields:
           nEPDCCH: 126
       NECCEPerPRB: 4
      NEREGPerECCE: 4
    NECCEPerEPDCCH: 4
        EPDCCHCase: 1
             NECCE: 16
```

Create the candidates for a DCI format 1 message for the same configuration. The DCI format 1 message is smaller than the format 2A message, resulting in a change of case number (`info.EPDCCHCase`) from 1 to 3. The aggregation level (`info.NECCEPerEPDCCH`) changes from 4 to 2, resulting in a greater number of candidates.

```
chs.DCIFormat = 'Format1';
[candidates,info] = lteEPDCCHSpace(enb,chs)

candidates = 6×2

     2     3
     4     5
     6     7
    10    11
    12    13
    14    15


info = struct with fields:
           nEPDCCH: 126
       NECCEPerPRB: 4
      NEREGPerECCE: 4
    NECCEPerEPDCCH: 2
        EPDCCHCase: 3
             NECCE: 16
```

## Input Arguments

### enb — eNodeB cell-wide settings

structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NDLRB** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **NCellID** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **CyclicPrefix** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **CellRefP** | Required | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **NSubframe** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| The following parameter applies only when `chs.EPDCCHStart` is absent. | | | |
| **CFI** | Required | 1, 2, or 3<br>Scalar or if the CFI varies per subframe, a vector of length 10 (corresponding to a frame). | Control format indicator (`CFI`) value. In TDD mode, `CFI` varies per subframe for the RMCs (`'R.0'`, `'R.5'`, `'R.6'`, `'R.6-27RB'`, `'R.12-9RB'`) |
| **DuplexMode** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| The following parameters apply when `DuplexMode` is set to `'TDD'`. | | | |
| **TDDConfig** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **SSC** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |
| **NFrame** | Optional | 0 (default), nonnegative scalar integer | Frame number |
| **CSIRSPeriod** | Optional | `'Off'` (default), `'On'`, `Icsi-rs` (0,...,154), `[Tcsi-rs Dcsi-rs]`. You can also specify values in a cell array of configurations for each resource. | CSI-RS subframe configurations for one or more CSI-RS resources. Multiple CSI-RS resources can be configured from a single common subframe configuration or from a cell array of configurations for each resource. |
| The following CSI-RS resource parameters apply only when `CSIRSPeriod` sets one or more CSI-RS subframe configurations to any value other than `'Off'`. Each parameter length must be equal to the number of CSI-RS resources required. | | | |
| **CSIRSConfig** | Required | Nonnegative scalar integer | Array CSI-RS configuration indices. See TS 36.211, Table 6.10.5.2-1. |
| **CSIRefP** | Required | 1 (default), 2, 4, 8 | Array of number of CSI-RS antenna ports |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **ZeroPowerCSIRSPeriod** | Optional | `'Off'` (default), `'On'`, `Icsi-rs` (0,...,154), `[Tcsi-rs Dcsi-rs]`. You can also specify values in a cell array of configurations for each resource. | Zero power CSI-RS subframe configurations for one or more zero power CSI-RS resource configuration index lists. Multiple zero power CSI-RS resource lists can be configured from a single common subframe configuration or from a cell array of configurations for each resource list. |
| The following zero power CSI-RS resource parameter is only applicable if one or more of the above zero power subframe configurations are set to any value other than `'Off'`. | | | |
| **ZeroPowerCSIRSConfig** | Required | 16-bit bitmap character vector or string scalar (truncated if not 16 bits or `'0'` MSB extended), or a numeric list of CSI-RS configuration indices. You can also specify values in a cell array of configurations for each resource. | Zero power CSI-RS resource configuration index lists (TS 36.211 Section 6.10.5.2). Specify each list as a 16-bit bitmap character vector or string scalar (if less than 16 bits, then `'0'` MSB extended), or as a numeric list of CSI-RS configuration indices from TS 36.211 Table 6.10.5.2-1 in the `'4'` CSI reference signal column. Multiple lists can be defined using a cell array of individual lists. |

**`chs` — EPDCCH-specific channel transmission configuration**
structure

EPDCCH-specific channel transmission configuration, specified as a structure that can contain the following parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **EPDCCHType** | Required if the EPDCCH type list parameter field is absent | `'Localized'`, `'Distributed'` | EPDCCH transmission type |
| **EPDCCHPRBSet** | Required if the EPDCCH Type list parameter field is absent | Vector of zero-based indices for the PRB pairs corresponding to the EPDCCH PRB set. The number of PRB pair indices must be a power of 2. | EPDCCH PRB pair indices |
| **EPDCCHFormat** | Required | 0, 1, 2, 3, or 4 | Number of ECCEs per EPDCCH transmission (equivalently the aggregation level L) as required by TS 36.211 Table 6.8A 1–2. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **EPDCCHStart** | Optional | 0, 1, 2, 3, or 4<br><br>If this parameter is not present, then the cell-wide CFI parameter is used for the starting symbol. | EPDCCH starting symbol |
| **RNTI** | Required | 0 (default), scalar integer | Radio network temporary identifier (RNTI) value (16 bits) |
| **DCIFormat** | Optional | `'Format0'`, `'Format1'`, `'Format1A'`, `'Format1B'`, `'Format1C'`, `'Format1D'`, `'Format2'`, `'Format2A'`, `'Format2B'`, `'Format2C'`, `'Format2D'`, `'Format3'`, `'Format3A'`, `'Format4'`, `'Format5'`, `'Format5A'` | Downlink control information (DCI) format |
| **EPDCCHPRBSetList** | Optional | cell array of one or two vectors | PRB pair indices for one or two EPDCCH sets |
| **EPDCCHTypeList** | Optional | cell array of one or two arrays | EPDCCH transmission types for one or two EPDCCH sets |

## Output Arguments

### `ind` — EPDCCH ECCE candidate indices

*M*-by-2 matrix | cell array

EPDCCH ECCE candidate indices, returned as an *M*-by-2 matrix or a cell array containing 2 *M*-by-2 matrices. *M* is the number of EPDCCH candidates monitored for the configuration provided. This variable is defined in TS 36.213 Tables 9.1.4-1a to 9.1.4-5b. Each two-element row of the matrix `ind` (or the rows of each matrix in cell array `ind`) contains the inclusive indices of a single EPDCCH candidate location.

- If `chs.EPDCCHPRBSetList` and `chs.EPDCCHTypeList` are present and either `chs.EPDCCHPRBSet` or `chs.EPDCCHType` are absent, one or two EPDCCH sets are returned in a cell array containing one or two matrices, one for each set.
- If both `chs.EPDCCHPRBSet` and `chs.EPDCCHType` are present, only the single candidate matrix which matches the PRB set size and EPDCCH type given by `chs.EPDCCHPRBSet` and `chs.EPDCCHType` is returned. This allows the number of candidates *M* to be correctly calculated for TS 36.213 Tables 9.1.4-3a to 9.1.4-5b (corresponding to two `EPDCCH` sets) while returning a single set of candidates in matrix form. This format is consistent with the parameterization other EPDCCH-related functions that take `CHS.EPDCCHPRBSet` and `CHS.EPDCCHType` as parameters and operate on a single EPDCCH set.
- If `chs.EPDCCHPRBSetList` is absent, then `chs.EPDCCHPRBSet` is required, and if `chs.EPDCCHTypeList` is absent then `chs.EPDCCHType` is required.

**`info` — Dimensional information about the `EPDCCH` search space candidates**
scalar structure

Dimensional information about the `EPDCCH` search space candidates, returned as a scalar structure containing:

| Parameter Field | Description | Values |
|---|---|---|
| **`nEPDCCH`** | Number of REs in a PRB pair configured for possible EPDCCH transmission. See TS 36.211. [1], Section 6.8A.1. | Integer |
| **`NECCEPerPRB`** | Number of ECCE per PRB pair | Integer |
| **`NEREGPerECCE`** | Number of EREG per ECCE | Integer |
| **`NECCEPerEPDCCH`** | Number of ECCES per EPDCCH transmission (equivalently the EPDCCH aggregation level L) as given by TS 36.211 [1] ,Table 6.8A.1-2 | Integer |
| **`EPDCCHCase`** | Case number (1,2,3). See TS 36.213 [2], Section 9.1.4 | Integer |
| **`NECCE`** | One or two element vector containing the number of ECCE available for transmission of EPDCCHs in the PRB pair set | Integer |

Data Types: `struct`

# Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.213. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteEPDCCH` | `lteEPDCCHDecode` | `lteEPDCCHIndices` | `lteEPDCCHSearch` | `lteEPDCCHPRBS`

# lteEPDCCHIndices

Enhanced physical downlink control channel (EPDCCH) resource element indices

## Syntax

```
[ind,info] = lteEPDCCHIndices(enb,chs)
[ind,info] = lteEPDCCHIndices(enb,chs,opts)
```

## Description

`[ind,info] = lteEPDCCHIndices(enb,chs)` returns the subframe resource element (RE) indices for the Enhanced Physical Downlink Control Channel (EPDCCH) and information related to EPDCCH indices, given the cell-wide settings structure, `enb`, and the EPDCCH transmission configuration, `chs`.

`[ind,info] = lteEPDCCHIndices(enb,chs,opts)` formats the returned indices using options specified by `opts`.

## Examples

### Generate RE Indices of Localized Transmission

This example generates RE Indices of localized transmission in default and subscripted formats.

Specify the cell-wide settings in parameter structure, `enb`.

```
enb.NDLRB = 6;
enb.NSubframe = 0;
enb.NCellID = 0;
enb.CellRefP = 1;
enb.CyclicPrefix = 'Normal';
enb.DuplexMode = 'FDD';
enb.NFrame = 0;
enb.CSIRSPeriod = 'Off';
enb.ZeroPowerCSIRSPeriod = 'Off';
```

Specify the channel transmission configuration in parameter structure, `chs`.

```
chs.EPDCCHECCE = [0 7];
chs.EPDCCHType = 'Localized';
chs.EPDCCHPRBSet = 2:3;
chs.EPDCCHStart = 2;
chs.RNTI = 1;
```

Generate 1-based linear resource element indices of a localized transmission.

```
[ind,info] = lteEPDCCHIndices(enb,chs);
size(ind)
```

```
ans = 1×2
```

```
   228     1
```

Display the size and the first 10 indices of `ind`.

```
ind(1:10)
```

```
ans = 10x1 uint32 column vector

   1177
   1178
   1179
   1180
   1181
   1182
   1183
   1184
   1185
   1186
```

Generate 1-based resource element indices in the subscript format [ `subcarrier`, `symbol`, `antenna` ].

```
[ind,info] = lteEPDCCHIndices(enb,chs,'sub');
size(ind)
```

```
ans = 1×2

   228     3
```

Display the size and the first 10 indices of `ind`.

```
ind(1:10,:)
```

```
ans = 10x3 uint32 matrix

   25    3    2
   26    3    2
   27    3    2
   28    3    2
   29    3    2
   30    3    2
   31    3    2
   32    3    2
   33    3    2
   34    3    2
```

## Input Arguments

### enb — eNodeB cell-wide settings

structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NDLRB** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **NCellID** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **CyclicPrefix** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **CellRefP** | Required | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **NSubframe** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| The following parameter is only read when `chs.EPDCCHStart` is absent. | | | |
| **CFI** | Required | 1, 2, or 3<br>Scalar or if the CFI varies per subframe, a vector of length 10 (corresponding to a frame). | Control format indicator (`CFI`) value. In TDD mode, `CFI` varies per subframe for the RMCs (`'R.0'`, `'R.5'`, `'R.6'`, `'R.6-27RB'`, `'R.12-9RB'`) |
| **DuplexMode** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as one of the following:<br>• `'FDD'` — Frequency division duplex (default)<br>• `'TDD'` — Time division duplex |
| The following parameters apply when `DuplexMode` is set to `'TDD'`. | | | |
| **TDDConfig** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **SSC** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |
| **NFrame** | Optional | 0 (default), nonnegative scalar integer | Frame number |
| **CSIRSPeriod** | Optional | `'Off'` (default), `'On'`, `Icsi-rs` (0,...,154), `[Tcsi-rs Dcsi-rs]`. You can also specify values in a cell array of configurations for each resource. | CSI-RS subframe configurations for one or more CSI-RS resources. Multiple CSI-RS resources can be configured from a single common subframe configuration or from a cell array of configurations for each resource. |
| The following CSI-RS resource parameters apply only when `CSIRSPeriod` sets one or more CSI-RS subframe configurations to any value other than `'Off'`. Each parameter length must be equal to the number of CSI-RS resources required. | | | |
| **CSIRSConfig** | Required | Nonnegative scalar integer | Array CSI-RS configuration indices. See TS 36.211, Table 6.10.5.2-1. |
| **CSIRefP** | Required | 1 (default), 2, 4, 8 | Array of number of CSI-RS antenna ports |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **ZeroPowerCSIRSPeriod** | Optional | `'Off'` (default), `'On'`, `Icsi-rs` (0,...,154), `[Tcsi-rs Dcsi-rs]`. You can also specify values in a cell array of configurations for each resource. | Zero power CSI-RS subframe configurations for one or more zero power CSI-RS resource configuration index lists. Multiple zero power CSI-RS resource lists can be configured from a single common subframe configuration or from a cell array of configurations for each resource list. |
| The following zero power CSI-RS resource parameter is only applicable if one or more of the above zero power subframe configurations are set to any value other than `'Off'`. | | | |
| **ZeroPowerCSIRSConfig** | Required | 16-bit bitmap character vector or string scalar (truncated if not 16 bits or `'0'` MSB extended), or a numeric list of CSI-RS configuration indices. You can also specify values in a cell array of configurations for each resource. | Zero power CSI-RS resource configuration index lists (TS 36.211 Section 6.10.5.2). Specify each list as a 16-bit bitmap character vector or string scalar (if less than 16 bits, then `'0'` MSB extended), or as a numeric list of CSI-RS configuration indices from TS 36.211 Table 6.10.5.2-1 in the `'4'` CSI reference signal column. Multiple lists can be defined using a cell array of individual lists. |

**`chs` — EPDCCH-specific channel transmission configuration**
structure

EPDCCH-specific channel transmission configuration, specified as a structure that can contain the following parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **EPDCCHECCE** | Required | 1- or 2- element vector specifying the zero-based ECCE index or inclusive `[begin, end]` ECCE index range according to the aggregation level `L` (`L = end − begin + 1`). The number of ECCEs in the candidate must be a power of 2.<br><br>If no transmission is required, leave this parameter empty. | The set of one of several consecutive ECCEs defining the EPDCCH transmission candidate in the overall EPDCCH set. |
| **EPDCCHType** | Required | `'Localized'`, `'Distributed'` | EPDCCH transmission type |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **EPDCCHPRBSet** | Required | Vector of zero-based indices for the PRB pairs corresponding to the EPDCCH PRB set. The number of PRB pair indices must be a power of 2.<br><br>If no transmission is required, leave this parameter empty. | EPDCCH PRB pair indices |
| **EPDCCHStart** | Optional | integer from 0 to 4<br><br>If this parameter is not present, then the cell-wide CFI parameter is used for the starting symbol. | EPDCCH starting symbol |
| The following parameters apply when `EPDCCHType` is set to `'Localized'`. | | | |
| **RNTI** | Required only for the `'Localized'` transmission type | 0 (default), scalar integer | Radio network temporary identifier (RNTI) value (16 bits) |

**`opts` — Index generation options**

character vector | cell array of character vectors | string array

Index generation options, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string):

| Option | Values | Description |
|---|---|---|
| Indexing style | `'ind'` (default), `'sub'` | Style for the returned indices, specified as one of the following options.<br>• `'ind'` — returns the indices in linear index form as a column vector (default)<br>• `'sub'` — returns the indices in `[subcarrier, symbol, antenna]` subscript row style. The number of rows in the output, `ind`, is the number of resource elements ($N_{RE}$). Thus, `ind` is an $N_{RE}$-by-3 matrix. |
| Index base | `'1based'` (default), `'0based'` | Base value of the returned indices. Specify `'1based'` to generate indices where the first value is 1. Specify `'0based'` to generate indices where the first value is 0. |

Whether in linear or subscript format style, the indices are always formed out of `[subcarrier, symbol, antenna]` subscripts. These subscripts identify the used resource elements in each subframe resource grid per antenna port.

For the EPDCCH, the antenna subscripts have the possible range 1...4 (if index is one-based), which represents antenna ports $p$ = 107...110. For a localized EPDCCH transmission, the antenna

subscripts are a single value out of 1...4, dependent on the RNTI and ECCEs selected. For a distributed EPDCCH transmission, the antenna subscripts alternate between one of two values: {1,3} ($p$ = 107,109) for normal cyclic prefix, and {1,2} ($p$ = 107,108) for extended cyclic prefix. See TS 36.211 [1], Section 6.8A.5. Use these indices to index the subframe grid directly. The grid comprises the four possible EPDCCH antenna ports ($p$ = 107...110) and is represented as an.$N_{SC}$-by-$N_{SYM}$-by-4 array.

Example: `'ind 1based'`, `"ind 1based"`, `{'ind','1based'}`, or `["ind","1based"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `ind` — Subframe EPDCCH RE indices

integer column vector | 3-column integer matrix

EPDCCH subframe resource element indices, returned by default in one-based linear indexing form as a numeric column vector of length $N_{RE}$-by-1. $N_{RE}$ is the number of subframe resource elements. Optionally, for subscript-specific indexing style [`subcarrier`, `symbol`, `antenna`], `ind` is returned as a numeric matrix of size $N_{RE}$-by-3. The grid comprises the four possible EPDCCH antenna ports ($p$ = 107,...,110) and is represented as an $N_{SC}$-by-$N_{SYM}$-by-4 array. $N_{SC}$ is the number of subcarriers, $N_{SYM}$ is the number of symbols, and 4 is the number of antenna ports.

The indices are for a single transmission instance of the EPDCCH. The order of the indices is the same as required for the complex EPDCCH symbols mapping. Generate these symbols using `lteEPDCCH`. The indices are parameterized in terms of a configured PRB pair set that defines:

- the overall set of possible EPDCCH candidates and
- the aggregation of one or more consecutive enhanced control channel elements (ECCE). This aggregation identifies the specific EPDCCH instance within the set of EPDCCH candidates.

The EPDCCH can use either localized or distributed transmission, differing in the mapping of ECCEs to REs, active PRB pairs, and antenna ports.

Data Types: `double`

### `info` — Information related to EPDCCH indices

scalar structure

Dimensional information related to EPDCCH indices, returned as a scalar structure. The structure `info` contains the following fields.

| Parameter Field | Description | Values | Data Type |
|---|---|---|---|
| **`EPDCCHG`** | EPDCCH data bit capacity | Integer | int32 |
| **`EPDCCHGd`** | EPDCCH QPSK symbol capacity | Integer | int32 |

| Parameter Field | Description | Values | Data Type |
|---|---|---|---|
| **nEPDCCH** | Number of REs in a PRB pair configured for possible EPDCCH transmission. See TS 36.211. [1], Section 6.8A.1. | Integer | int32 |
| **NECCE** | Number of ECCE available for transmission of EPDCCHs in the PRB pair set | Integer | int32 |
| **NECCEPerPRB** | Number of ECCE per PRB pair | Integer | int32 |
| **NEREGPerECCE** | Number of EREG per ECCE | Integer | int32 |
| **EPDCCHPorts** | A vector indicating the set of antenna subscripts used by REs for this transmission instance of the EPDCCH. The subscripts are one-based (default) or zero-based as specified by `opts`. | Vector of integers | int32 |

# Version History

**Introduced in R2014b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteEPDCCH` | `lteEPDCCHDMRSIndices`

# lteEPDCCHPRBS

EPDCCH pseudorandom scrambling sequence

## Syntax

```
[seq,cinit] = lteEPDCCHPRBS(enb,chs,n)
[seq,cinit] = lteEPDCCHPRBS(enb,chs,n,mapping)

[subseq,cinit] = lteEPDCCHPRBS(enb,chs,pn)
[subseq,cinit] = lteEPDCCHPRBS(enb,chs,pn,mapping)
```

## Description

`[seq,cinit] = lteEPDCCHPRBS(enb,chs,n)` returns the first `n` outputs of the Enhanced Physical Downlink Control Channel (EPDCCH) scrambling sequence in `seq`. It also returns an initialization value `cinit` for the pseudorandom binary sequence (PRBS) generator. The function is initialized according to the cell-wide settings structure, `enb`, and the channel transmission configuration structure, `chs`.

`[seq,cinit] = lteEPDCCHPRBS(enb,chs,n,mapping)` allows additional control over the format of the returned sequence, `seq`, with the input `mapping`.

`[subseq,cinit] = lteEPDCCHPRBS(enb,chs,pn)` returns a subsequence of a full PRBS sequence, specified by `pn`.

`[subseq,cinit] = lteEPDCCHPRBS(enb,chs,pn,mapping)` allows additional control over the format of the returned subsequence, `subseq`, with the input `mapping`.

## Examples

### Generate the EPDCCH Scrambling Sequence

Specify the cell-wide settings and channel transmission configuration in parameter structures `enb` and `chs`.

```
enb.NSubframe = 0;
chs.EPDCCHNID = 0;
```

Create the codeword and generate the EPDCCH scrambling sequence.

```
cw = randi([0 1],100,1);
prbs = lteEPDCCHPRBS(enb,chs,length(cw));
```

Scramble the DCI coded bits.

```
scrambled = xor(prbs,cw);
prbs(1:20)
```

```
ans = 20x1 logical array
```

```
   0
   0
   0
   0
   0
   0
   1
   0
   0
   0
      ⋮
```

Generate the EPDCCH scrambling sequence using the `'signed'` sequence format.

```
prbs = lteEPDCCHPRBS(enb,chs,length(cw),'signed');
prbs(1:20)
```

```
ans = 20×1

     1
     1
     1
     1
     1
     1
    -1
     1
     1
     1
      ⋮
```

## Input Arguments

### `enb` — eNodeB cell-wide settings

structure

eNodeB cell-wide settings, specified as a structure. This argument contains the following parameter field.

### `NSubframe` — Subframe number

nonnegative scalar integer

Subframe number, specified as a nonnegative scalar integer.

Data Types: `double`

Data Types: `struct`

### `chs` — Channel-specific transmission configuration

structure

Channel-specific transmission configuration, specified as a structure. This argument contains the following parameter field.

**EPDCCHNID — EPDCCH scrambling sequence initialization**
nonnegative scalar integer

EPDCCH nID parameter for scrambling sequence initialization, specified as a nonnegative scalar integer.

Data Types: `double`

Data Types: `struct`

**n — Number of elements in returned sequence**
numeric scalar

Number of elements in returned sequence, `seq`, specified as a numeric scalar.

Data Types: `double`

**pn — Range of elements in returned subsequence**
row vector

Range of elements in returned subsequence, `subseq`, specified as a row vector of `[p n]`. The subsequence returns `n` values of the PRBS generator, starting at position `p` (0-based).

Data Types: `double`

**mapping — Output sequence formatting**
`'binary'` (default) | `'signed'`

Output sequence formatting, specified as `'binary'` or `'signed'`. `mapping` controls the format of the returned sequence.

- `'binary'` maps `true` to 1 and `false` to 0.
- `'signed'` maps `true` to –1 and `false` to 1.

Data Types: `char` | `string`

## Output Arguments

**seq — EPDCCH pseudorandom scrambling sequence**
logical column vector | numeric column vector

EPDCCH pseudorandom scrambling sequence, returned as a logical column vector or a numeric column vector. `seq` contains the first `n` outputs of the EPDCCH scrambling sequence. If you set `mapping` to `'signed'`, the output data type is `double`. Otherwise, the output data type is `logical`.

Data Types: `logical` | `double`

**subseq — EPDCCH pseudorandom scrambling subsequence**
logical column vector | numeric column vector

EPDCCH pseudorandom scrambling subsequence, returned as a logical column vector or a numeric column vector. `subseq` contains the values of the PRBS generator specified by `pn`. If you set `mapping` to `'signed'`, the output data type is `double`. Otherwise, the output data type is `logical`.

Data Types: `logical` | `double`

**`cinit` — Initialization value for PRBS generator**
numeric scalar

Initialization value for PRBS generator, returned as a numeric scalar.

Data Types: `uint32`

# Version History

**Introduced in R2014b**

## See Also

`lteEPDCCH` | `lteEPDCCHIndices`

# lteEVM

Error vector magnitude calculation

## Syntax

```
evm = lteEVM(x,r)
evm = lteEVM(ev)
```

## Description

`evm = lteEVM(x,r)` returns a structure, `evm`, containing error vector magnitude (EVM) information for the input array, `x`, given the reference signal array, `r`. The EVM is defined using the error, or difference, between the input values, `x`, and the reference signal, `r`.

The EVM values in the `RMS` and `Peak` structure fields are linear EVM, not EVM as a percentage. To obtain EVM as a percentage, multiply the value of the `RMS` and `Peak` structure fields by 100.

`evm = lteEVM(ev)` returns a structure, `evm`, for the input array, `ev`, which is taken to be the normalized error vector given by the expression `ev=(x-r)/sqrt(mean(abs(r.^2)))`. This syntax allows for peak and RMS EVM calculation for preexisting normalized error vectors. For example, it can be used to calculate the EVM across an array of previous EVM results, by extracting and concatenating the EV fields from the array to form the `ev` input vector.

## Examples

### Measure LTE Symbol EVM

Generate a random QPSK constellation at a defined EVM level. Measure and confirm the added EVM.

Generate a stream of QPSK symbols.

```
txSym = lteSymbolModulate(randi([0,1],10000,1),'QPSK');
```

Add noise at a defined EVM level, `evmPercent`.

```
evmPercent = 14.0;
N0 = complex(randn(size(txSym)),randn(size(txSym)));
noise = N0 * (evmPercent/100)/sqrt(2);
rxSym = txSym + noise;
```

Measure and display the root mean square EVM level in percent.

```
evm = lteEVM(rxSym,txSym)
```

```
evm = struct with fields:
      EV: [5000x1 double]
    Peak: 0.4260
     RMS: 0.1382
```

```
evm.RMS*100
```

```
ans = 13.8234
```

## Input Arguments

### x — Input array

column vector | matrix | 3-D array

Input array, specified as a column vector, matrix or 3-D array.

Data Types: `double` | `single`
Complex Number Support: Yes

### r — Reference signal vector

column vector | matrix | 3-D array

Reference signal array, specified as a column vector, matrix or 3-D array.

Data Types: `double` | `single`
Complex Number Support: Yes

### ev — Normalized error array

column vector

Normalized error array, specified as a column vector, matrix or 3-D array.

Data Types: `double` | `single`
Complex Number Support: Yes

## Output Arguments

### evm — EVM information

structure

EVM information, returned as structure. `evm` contains the following fields.

### RMS — Root mean square (RMS) EVM

positive numeric scalar

Root mean square (RMS) EVM, specified as a positive numeric scalar. It is the square root of the mean of the squares of all the values of the EVM.

Data Types: `double` | `single`

### Peak — Peak EVM

positive numeric scalar

Peak EVM, returned as a positive numeric scalar. It is the largest single EVM value calculated across all input values.

Data Types: `double` | `single`

### EV — Normalized error vector

numeric column vector

Normalized error vector, returned as a numeric column vector.

Data Types: `double` | `single`
Complex Number Support: Yes

Data Types: `struct`

## Version History

**Introduced in R2014a**

## See Also

`lteSymbolDemodulate`

# lteEqualizeMIMO

MMSE-based joint downlink equalization and combining

## Syntax

```
[out,csi] = lteEqualizeMIMO(enb,chs,in,hest,noiseest)
```

## Description

`[out,csi] = lteEqualizeMIMO(enb,chs,in,hest,noiseest)` performs joint equalization and combining of the received PDSCH symbols in `in`, given cell-wide settings structure, `enb`, PDSCH configuration structure, `chs`, channel estimate, `hest`, and noise power estimate, `noiseest`. MMSE equalization is performed on the product of the channel matrix and precoding matrices. Thus, it performs MMSE equalization between transmit and receive layers and returns the result, `out`.

## Examples

### Equalize and Deprecode PDSCH Symbols

Equalize and deprecode the PDSCH symbols for RMC R.11 in a MIMO configuration. The PDSCH symbols are extracted from a transmit resource grid. An ideal (identity) channel estimate and ideal (zero) noise estimate are created. The channel and noise estimates are used to equalize and deprecode the PDSCH symbols.

Initialize cell-wide configuration structure, `enb`. Generate and populate transmit resource grid for RMC R.11.

```
rmccfg.RC = 'R.11';
ncodewords = 2;
enb = lteRMCDL(rmccfg, ncodewords);
enb.TotSubframes = 1;
[~,txGrid] = lteRMCDLTool(enb, {[1;0] [0;1]});
```

Extract the PDSCH symbols from this transmit grid.

```
[ind,indInfo] = ltePDSCHIndices(enb, enb.PDSCH, enb.PDSCH.PRBSet);
pdschSym = txGrid(ind);
```

Create an ideal, or identity, channel estimate and an ideal, or zero, noise estimate.

```
hest = permute(repmat(eye(enb.CellRefP), [1 1 indInfo.Gd]), [3 1 2]);
nest = 0.0;
```

Equalize and deprecode the PDSCH symbols, using the channel and noise estimates.

```
[out,csi] = lteEqualizeMIMO(enb, enb.PDSCH, pdschSym, hest, nest);
deprecoded = lteDLDeprecode(enb,enb.PDSCH,out);
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure with the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NDLRB** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **NCellID** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **CyclicPrefix** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **CellRefP** | Required | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **NSubframe** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **DuplexMode** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| The following parameters are dependent upon the condition that `enb.DuplexMode` is set to `'TDD'`. | | | |
| **TDDConfig** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **SSC** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |
| The following parameter fields are dependent upon the condition that `chs.TxScheme` is set to `'SpatialMux'` or `'MultiUser'`. | | | |
| **CFI** | Required | 1, 2, or 3<br>Scalar or if the CFI varies per subframe, a vector of length 10 (corresponding to a frame). | Control format indicator (`CFI`) value. In TDD mode, `CFI` varies per subframe for the RMCs (`'R.0'`, `'R.5'`, `'R.6'`, `'R.6-27RB'`, `'R.12-9RB'`) |

Data Types: `struct`

### chs — PDSCH configuration

structure

PDSCH configuration, specified as a structure with the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NLayers** | Required | Integer from 1 to 8 | Number of transmission layers (downlink modulation) |

<table>
<tr><th>Parameter Field</th><th>Required or Optional</th><th>Values</th><th>Description</th></tr>
<tr><td>RNTI</td><td>Required</td><td>0 (default), scalar integer</td><td>Radio network temporary identifier (RNTI) value (16 bits)</td></tr>
<tr><td>TxScheme</td><td>Required</td><td>'CDD', 'SpatialMux', 'MultiUser'</td><td>Transmission scheme, specified as one of the following options.
<table>
<tr><th>Transmission scheme</th><th>Description</th></tr>
<tr><td>'CDD'</td><td>Large delay cyclic delay diversity</td></tr>
<tr><td>'SpatialMux'</td><td>Closed loop spatial multiplexing</td></tr>
<tr><td>'MultiUser'</td><td>Multi-user MIMO</td></tr>
</table></td></tr>
<tr><td colspan="4">The following parameters are dependent upon the condition that TxScheme is set to 'SpatialMux' or 'MultiUser'.</td></tr>
<tr><td>PMISet</td><td>Required</td><td>Integer vector with element values from 0 to 15.</td><td>Precoder matrix indication (PMI) set. It can contain either a single value, corresponding to single PMI mode, or multiple values, corresponding to multiple or subband PMI mode. The number of values depends on CellRefP, transmission layers and TxScheme. For more information about setting PMI parameters, see ltePMIInfo.</td></tr>
<tr><td>PRBSet</td><td>Required</td><td>Integer column vector or two-column matrix</td><td>Zero-based physical resource block (PRB) indices corresponding to the slot wise resource allocations for this PDSCH. PRBSet can be assigned as:<ul><li>a column vector, the resource allocation is the same in both slots of the subframe,</li><li>a two-column matrix, this parameter specifies different PRBs for each slot in a subframe,</li><li>a cell array of length 10 (corresponding to a frame, if the allocated physical resource blocks vary across subframes).</li></ul>PRBSet varies per subframe for the RMCs 'R.25'(TDD), 'R.26'(TDD), 'R.27'(TDD), 'R.43'(FDD), 'R.44', 'R.45', 'R.48', 'R.50', and 'R.51'.</td></tr>
</table>

Data Types: struct

### in — Received PDSCH input symbols

numeric matrix

Received PDSCH input symbols, specified as a numeric matrix of size *M*-by-NRxAnts, where *M* is the number of received symbols for each of NRxAnts receive antennas.

Data Types: double
Complex Number Support: Yes

**hest — Channel estimate**
3-D numeric array

Channel estimate, specified as a 3-D numeric array of size *M*-by-NRxAnts-by-enb.CellRefP, where:

- *M* is the number of received symbols in in,
- NRxAnts is the number of receive antennas,
- enb.CellRefP is the number of cell-specific reference signal antenna ports.

Data Types: double

**noiseest — Noise power estimate**
numeric scalar

Noise power estimate, specified as a numeric scalar. This argument is an estimate of the noise power spectral density per RE on rxgrid. Such an estimate is provided by the lteDLChannelEstimate function.

Data Types: double

## Output Arguments

**out — Equalized output symbols**
numeric matrix

Equalized output symbols, returned as a numeric matrix of size *M*-by-*NU*, where

- *M* is the number of received symbols for each receive antenna
- *NU* is the number of transmit layers

Data Types: double
Complex Number Support: Yes

**csi — Soft channel state information**
numeric matrix

Soft channel state information, returned as a numeric matrix of size *M*-by-NU, the same size as out. This argument contains soft channel state information and provides an estimate, via MMSE, of the received gain for each received layer.

Data Types: double

# Version History

**Introduced in R2014a**

## See Also

lteDLChannelEstimate | ltePDSCHDecode | lteDLPrecode | lteEqualizeMMSE | lteEqualizeZF | lteEqualizeULMIMO

# lteEqualizeMMSE

MMSE equalization

## Syntax

```
[out,csi] = lteEqualizeMMSE(rxgrid,channelest,noiseest)
```

## Description

`[out,csi] = lteEqualizeMMSE(rxgrid,channelest,noiseest)` returns equalized data in multidimensional array, `out`. MMSE equalization is applied to the received data resource grid in the matrix, `rxgrid`, using the channel information in the `channelest` matrix. `noiseest` is an estimate of the received noise power spectral density.

Alternatively, the input `channelest` can be provided as a 3-D array of size *NRE*-by-`NRxAnts`-by-*P*, and the input `rxgrid` can be provided as a matrix of size *NRE*-by-`NRxAnts`. In this case, the first two dimensions have been reduced to one dimension by appropriate indexing through the frequency and time locations of the resource elements of interest, typically for a single physical channel. The outputs, `out` and `csi`, are of size (*N*×*M*)-by-*P*.

## Examples

### Equalize MMSE for RMC R.4

Equalize the received signal for RMC R.4 after channel estimation. Use the MMSE equalizer.

Create cell-wide configuration structure and generate transmit signal. Configure propagation channel.

```
enb = lteRMCDL('R.4');
[txSignal,~,info] = lteRMCDLTool(enb,[1;0;0;1]);

chcfg.DelayProfile = 'EPA';
chcfg.NRxAnts = 1;
chcfg.DopplerFreq = 70;
chcfg.MIMOCorrelation = 'Low';
chcfg.SamplingRate = info.SamplingRate;
chcfg.Seed = 1;
chcfg.InitPhase = 'Random';
chcfg.InitTime = 0;

txSignal = [txSignal; zeros(15,1)];
N = length(txSignal);
noise = 1e-3*complex(randn(N,chcfg.NRxAnts),randn(N,chcfg.NRxAnts));
rxSignal = lteFadingChannel(chcfg,txSignal)+noise;
```

Perform synchronization and OFDM demodulation.

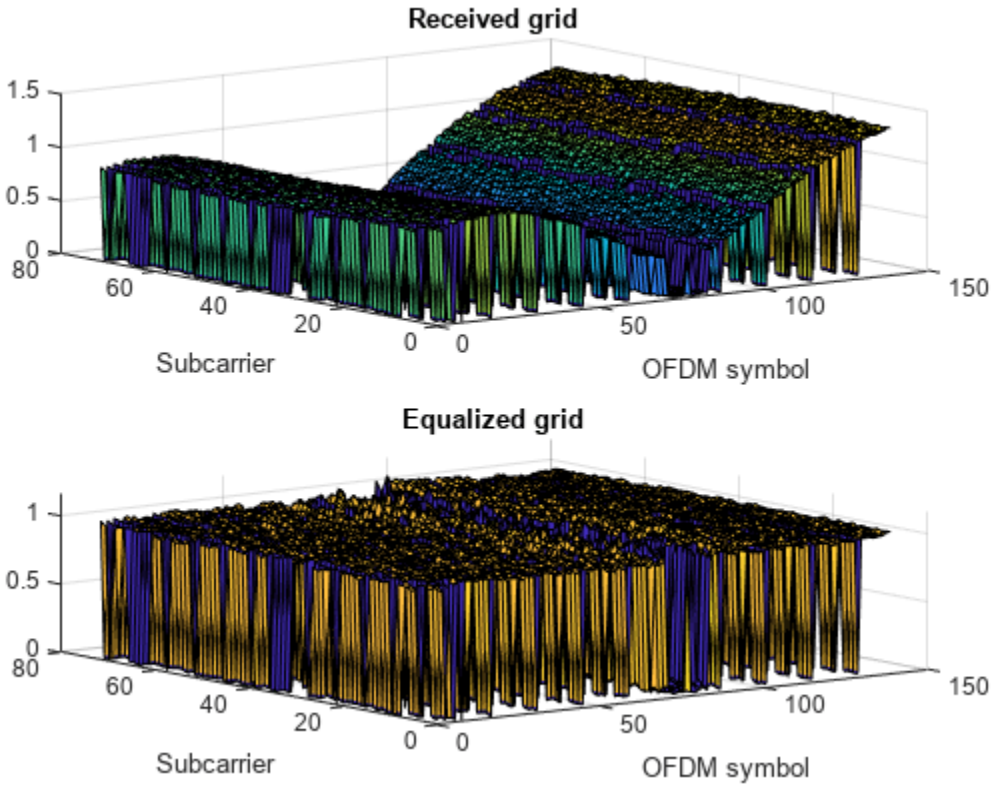
```
offset = lteDLFrameOffset(enb,rxSignal);
rxGrid = lteOFDMDemodulate(enb,rxSignal(1+offset:end,:));
```

Create channel estimation configuration structure and perform channel estimation.

```
cec.FreqWindow = 9;
cec.TimeWindow = 9;
cec.InterpType = 'Cubic';
cec.PilotAverage = 'UserDefined';
cec.InterpWinSize = 3;
cec.InterpWindow = 'Causal';
[hest,noiseEst] = lteDLChannelEstimate(enb, cec, rxGrid);
```

Equalize and plot received and equalized grids.

```
eqGrid = lteEqualizeMMSE(rxGrid, hest, noiseEst);
subplot(2,1,1)
surf(abs(rxGrid))
title('Received grid')
xlabel('OFDM symbol')
ylabel('Subcarrier')

subplot(2,1,2)
surf(abs(eqGrid))
title('Equalized grid')
xlabel('OFDM symbol')
ylabel('Subcarrier')
```



### Equalize MMSE for RMC R.5

This example applies MMSE equalization on the received signal for reference measurement channel (RMC) R.5, after channel estimation.

Set the DL reference measurement channel to R.5

```
enb = lteRMCDL('R.5');
```

Set channel estimator configuration `PilotAverage` field to `UserDefined`. as follows: averaging window of 9 resource elements in both frequency and time domain, cubic interpolation with a casual window.

```
cec = struct('FreqWindow',9,'TimeWindow',9,'InterpType','cubic');
cec.PilotAverage = 'UserDefined';
cec.InterpWinSize = 1;
cec.InterpWindow = 'Causal';
```

Generate the `txWaveform`.

```
txWaveform = lteRMCDLTool(enb,[1;0;0;1]);
n = length(txWaveform);
```

Apply some random noise to the transmitted signal and save as the `rxWaveform`.

```
rxWaveform = repmat(txWaveform,1,2)+complex(randn(n,2),randn(n,2))*1e-3;
```

Next, demodulate the received data.

```
rxGrid = lteOFDMDemodulate(enb,rxWaveform);
```

Then, perform channel estimation.

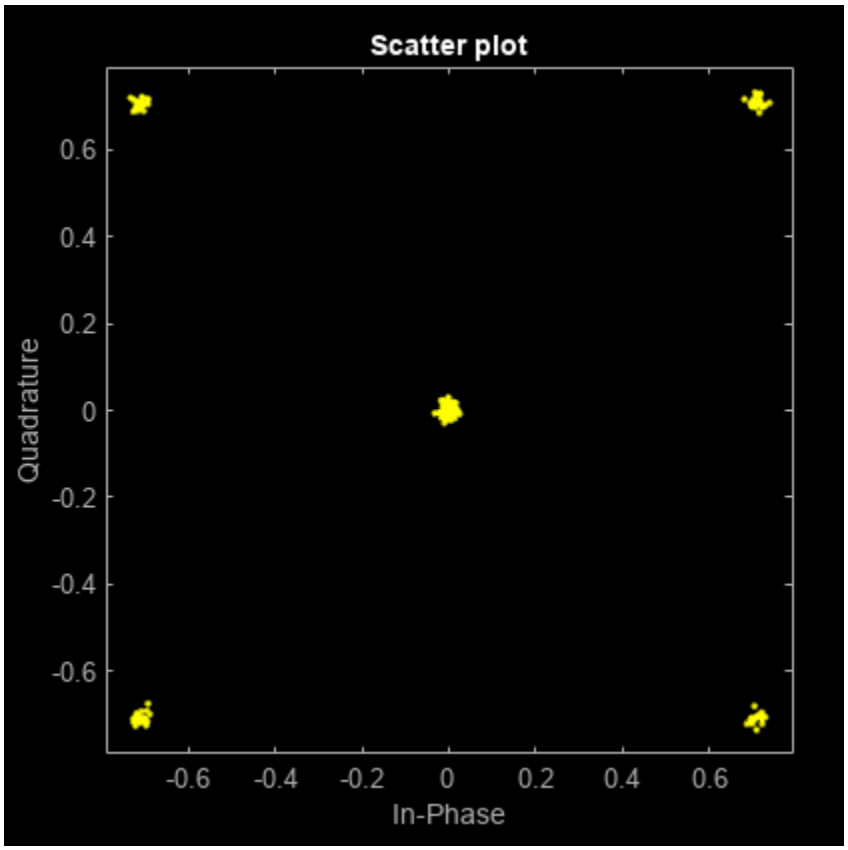
```
[hest,n0] = lteDLChannelEstimate(enb,cec,rxGrid);
```

Finally, apply the MMSE equalization.

```
out = lteEqualizeMMSE(rxGrid,hest,n0);
```

Show scatter plot of one component carrier.

```
scatterplot(out(:,1))
```




## Input Arguments

### `rxgrid` — Received data resource grid

3-D numeric array | 2-D numeric matrix

Received data resource grid, specified as a 3-D numeric array or a 2-D numeric matrix. As a 3-D numeric array, it has size *N*-by-*M*-by-`NRxAnts`, where *N* is the number of subcarriers, *M* is the number of OFDM symbols, and `NRxAnts` is the number of receive antennas.

Alternatively, as a 2-D numeric matrix, it has size *NRE*-by-`NRxAnts`. In this case, the first two dimensions have been reduced to one dimension by appropriate indexing through the frequency and time locations of the resource elements of interest, typically for a single physical channel.

Data Types: `double`
Complex Number Support: Yes

### `channelest` — Channel information

4-D numeric array | 3-D numeric array

Channel information, specified as a 4-D numeric array or a 3-D numeric array. As a 4-D numeric array, it has size *N*-by-*M*-by-`NRxAnts`-by-*P*. *N* is the number of subcarriers, *M* is the number of OFDM symbols, `NRxAnts` is the number of receive antennas, and *P* is the number of transmit antennas. Each element is a complex number representing the narrowband channel for each resource element and for each link between transmit and receive antennas. This matrix can be obtained using the channel estimation command `lteDLChannelEstimate`.

Alternatively, as a 3-D numeric array, it has size *NRE*-by-`NRxAnts`-by-*P*. In this case, the first two dimensions have been reduced to one dimension by appropriate indexing through the frequency and time locations of the resource elements of interest, typically for a single physical channel.

Data Types: `double`
Complex Number Support: Yes

**`noiseest` — Noise power estimate**
numeric scalar

Noise power estimate, specified as a numeric scalar. It is an estimate of the received noise power spectral density per RE on `rxgrid`.

Data Types: `double`

## Output Arguments

**`out` — Equalized output data**
3-D numeric array | 2-D numeric matrix

Equalized output data, returned as a 3-D numeric array or a 2-D numeric matrix. As a 3-D numeric array, it has size *N*-by-*M*-by-*P*, where *N* is the number of subcarriers, *M* is the number of OFDM symbols, and *P* is the number of transmit antennas.

Alternatively, if `channelest` is provided as a 3-D array, `out` is a 2-D numeric matrix of size (*N*×*M*)-by-*P*. In this case, the first two dimensions have been reduced to one dimension by appropriate indexing through the frequency and time locations of the resource elements of interest, typically for a single physical channel.

Data Types: `double`
Complex Number Support: Yes

**`csi` — Soft channel state information**
3-D numeric array | 2-D numeric matrix

Soft channel state information, returned as a 3-D numeric array of the same size as `out`. As a 3-D numeric array, it has size *N*-by-*M*-by-*P*, where *N* is the number of subcarriers, *M* is the number of OFDM symbols, and *P* is the number of transmit antennas. `csi` provides an estimate (via MMSE) of the received RE gain for each received RE.

Alternatively, if `channelest` is provided as a 3-D array, `csi` is a 2-D numeric matrix of size (*N*×*M*)-by-*P*. In this case, the first two dimensions have been reduced to one dimension by appropriate indexing through the frequency and time locations of the resource elements of interest, typically for a single physical channel.

Data Types: `double`

# Version History

**Introduced in R2014a**

## See Also

`lteDLChannelEstimate` | `lteEqualizeZF` | `lteEqualizeMIMO` | `lteEqualizeULMIMO` | `lteOFDMDemodulate` | `lteSCFDMADemodulate` | `lteULChannelEstimate`

# lteEqualizeULMIMO

MMSE-based joint uplink equalization and combining

## Syntax

```
[out,csi] = lteEqualizeULMIMO(ue,chs,in,hest,noiseest)
```

## Description

`[out,csi] = lteEqualizeULMIMO(ue,chs,in,hest,noiseest)` performs joint equalization and combining of the received PUSCH symbols in `in`, given UE-specific settings structure, `ue`, PUSCH configuration structure, `chs`, channel estimate, `hest` and noise power estimate, `noiseest`. MMSE equalization is performed on the product of the channel matrix and precoding matrices, thus performing MMSE equalization between transmit and receive layers and returning the result in `out`.

## Examples

### Equalize and Deprecode PUSCH Symbols

Extract, equalize, and deprecode PUSCH symbols from an RMC A3-2 grid.

Generate a resource grid using multiple antennas to transmit a single PUSCH codeword.

```
ue = lteRMCUL('A3-2');
ue.TotSubframes = 1;
ue.NTxAnts = 2;
ue.PUSCH.NLayers = 2;
[~,txGrid] = lteRMCULTool(ue,[1;0;0;1]);
```

Extract the PUSCH symbols from this transmit grid.

```
[ind,indInfo] = ltePUSCHIndices(ue,ue.PUSCH);
puschSym = txGrid(ind);
```

Create an ideal, or identity, channel estimate and an ideal, or zero, noise estimate.

```
hest = permute(repmat(eye(ue.NTxAnts),[1,1,indInfo.Gd]),[3,1,2]);
nest = 0.0;
```

Equalize and deprecode the PUSCH symbols, using the channel and noise estimates.

```
[out,csi] = lteEqualizeULMIMO(ue,ue.PUSCH,puschSym,hest,nest);
NPRB = size(ue.PUSCH.PRBSet,1);
deprecoded = lteULDeprecode(out,NPRB);
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific settings, specified as a structure that can contain the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| `NTxAnts` | Optional | 1 (default), 2, 4 | Number of transmission antennas. |

Data Types: `struct`

### `chs` — PUSCH configuration structure

structure

PUSCH configuration structure, specified as a structure that can contain the following fields. The `PMI` parameter field is only required if `ue.NTxAnts` is set to 2 or 4.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| `NLayers` | Optional | 1 (default), 2, 3, 4 | Number of transmission layers. |
| The following parameter is required only when `ue.NTxAnts` is set to 2 or 4. | | | |
| `PMI` | Required | Nonnegative scalar integer from 0 (default) to 23 | Precoder matrix indication. This PMI is to be used during precoding of the DRS reference symbols. For more information, see `lteULPMIInfo`. |

Data Types: `struct`

### `in` — Received PUSCH input symbols

numeric matrix

Received PUSCH input symbols, specified as a numeric matrix of size *M*-by-`NRxAnts`, where *M* is the number of received symbols for each of the `NRxAnts` receive antennas.

Data Types: `double`
Complex Number Support: Yes

### `hest` — Channel estimate

3-D numeric array

Channel estimate, specified as a 3-D numeric array of size *M*-by-`NRxAnts`-by-`NTxAnts`, where *M* is the number of received symbols in `in`, `NRxAnts` is the number of receive antennas, and `NTxAnts` is the number of transmit antenna ports, given by `ue.NTxAnts`.

Data Types: `double`

### `noiseest` — Noise power estimate

numeric scalar

Noise power estimate as power spectral density per RE on `rxgrid`, specified as a numeric scalar. Such an estimate is provided by the `lteULChannelEstimate` function.

Data Types: `double`

## Output Arguments

### `out` — Equalized output symbols

complex-valued numeric matrix

Equalized output symbols, returned as a complex-valued numeric matrix of size *M*-by-*NU*, where *M* is the number of received symbols for each receive antenna and *NU* is the number of transmit layers.

Data Types: `double`
Complex Number Support: Yes

### `csi` — Soft channel state information

numeric matrix

Soft channel state information, returned as a numeric matrix of the same size as `out`, *M*-by-*NU*. This output provides an estimate, via MMSE, of the received gain for each received layer.

Data Types: `double`

# Version History

**Introduced in R2013b**

## See Also

`lteEqualizeZF` | `lteEqualizeMMSE` | `lteEqualizeMIMO` | `ltePUSCHDecode` | `ltePUSCHPrecode` | `lteULChannelEstimate`

# lteEqualizeZF

Zero-forcing equalization

## Syntax

```
[out,csi] = lteEqualizeZF(rxgrid,channelest)
```

## Description

`[out,csi] = lteEqualizeZF(rxgrid,channelest)` returns equalized data in multidimensional array, `out`, by applying MIMO zero-forcing equalization to the received data resource grid in matrix `rxgrid`, using the channel information in the `channelest` input matrix.

For each resource element, the function calculates the pseudoinverse of the channel and equalizes the corresponding received signal.

Alternatively, the `channelest` input can be provided as a 3-D array of size *NRE*-by-*NRxAnts*-by-*P* and the `rxgrid` input can be provided as a matrix of size *NRE*-by-*NRxAnts*. In this case, the first two dimensions have been reduced to one dimension by appropriate indexing through the frequency and time locations of the resource elements of interest, typically for a single physical channel. The outputs, `out` and `csi`, are of size ($N \times M$)-by-*P*.

## Examples

### Perform Zero-Forcing Equalization for RMC R.4

Equalize the received signal for RMC R.4 after channel estimation. Use the zero forcing equalizer.

Create cell-wide configuration structure and generate transmit signal. Configure propagation channel.

```
enb = lteRMCDL('R.4');
[txSignal,~,info] = lteRMCDLTool(enb,[1;0;0;1]);

chcfg.DelayProfile = 'EPA';
chcfg.NRxAnts = 1;
chcfg.DopplerFreq = 70;
chcfg.MIMOCorrelation = 'Low';
chcfg.SamplingRate = info.SamplingRate;
chcfg.Seed = 1;
chcfg.InitPhase = 'Random';
chcfg.InitTime = 0;

txSignal = [txSignal; zeros(15,1)];
N = length(txSignal);
noise = 1e-3*complex(randn(N,chcfg.NRxAnts),randn(N,chcfg.NRxAnts));
rxSignal = lteFadingChannel(chcfg,txSignal)+noise;
```

Perform synchronization and OFDM demodulation.

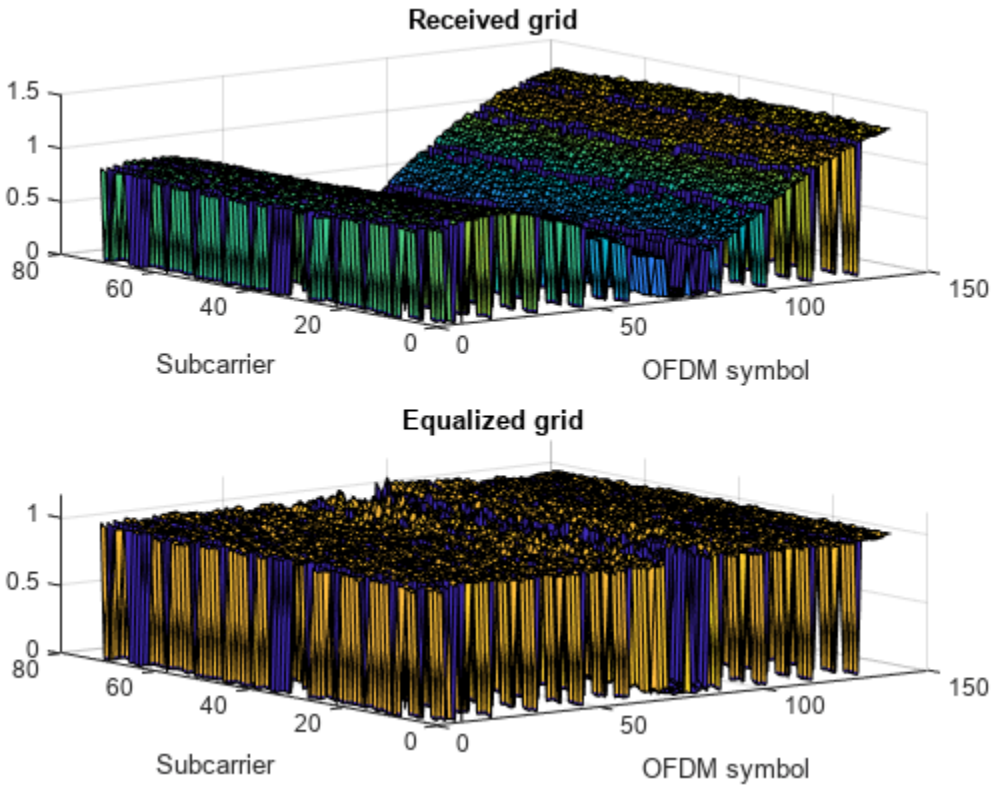
```
offset = lteDLFrameOffset(enb,rxSignal);
rxGrid = lteOFDMDemodulate(enb,rxSignal(1+offset:end,:));
```

Create channel estimation configuration structure and perform channel estimation.

```
cec.FreqWindow = 9;
cec.TimeWindow = 9;
cec.InterpType = 'Cubic';
cec.PilotAverage = 'UserDefined';
cec.InterpWinSize = 3;
cec.InterpWindow = 'Causal';
hest = lteDLChannelEstimate(enb,cec,rxGrid);
```

Equalize and plot received and equalized grids.

```
eqGrid = lteEqualizeZF(rxGrid,hest);

subplot(2,1,1);
surf(abs(rxGrid));
title('Received grid');
xlabel('OFDM symbol');
ylabel('Subcarrier');

subplot(2,1,2);
surf(abs(eqGrid));
title('Equalized grid');
xlabel('OFDM symbol');
ylabel('Subcarrier');
```



## Input Arguments

### `rxgrid` — Received data resource grid

3-D numeric array | 2-D numeric matrix

Received data resource grid, specified as a 3-D numeric array or a 2-D numeric matrix. As a 3-D numeric array, it has size *N*-by-*M*-by-`NRxAnts`, where *N* is the number of subcarriers, *M* is the number of OFDM symbols, and `NRxAnts` is the number of receive antennas.

Alternatively, as a 2-D numeric matrix, it has size *NRE*-by-`NRxAnts`. In this case, the first two dimensions have been reduced to one dimension by appropriate indexing through the frequency and time locations of the resource elements of interest, typically for a single physical channel.

Data Types: `double`
Complex Number Support: Yes

### `channelest` — Channel information

4-D numeric array | 3-D numeric array

Channel information, specified as a 4-D numeric array or a 3-D numeric array. As a 4-D numeric array, it has size *N*-by-*M*-by-`NRxAnts`-by-*P*. *N* is the number of subcarriers, *M* is the number of OFDM symbols, `NRxAnts` is the number of receive antennas, and *P* is the number of transmit antennas. Each element is a complex number representing the narrowband channel for each resource element and for each link between transmit and receive antennas. This matrix can be obtained using a channel estimation function, such as `lteDLChannelEstimate`.

Alternatively, as a 3-D numeric array, it has size *NRE*-by-`NRxAnts`-by-*P*. In this case, the first two dimensions have been reduced to one dimension by appropriate indexing through the frequency and time locations of the resource elements of interest, typically for a single physical channel.

Data Types: `double`
Complex Number Support: Yes

## Output Arguments

### `out` — Equalized output data

3-D numeric array | 2-D numeric matrix

Equalized output data, returned as a 3-D numeric array or a 2-D numeric matrix. As a 3-D numeric array, it has size *N*-by-*M*-by-*P*. *N* is the number of subcarriers, *M* is the number of OFDM symbols, and *P* is the number of transmit antennas.

Alternatively, if `channelest` is provided as a 3-D array, `out` is a 2-D numeric matrix of size ($N \times M$)-by-*P*. In this case, the first two dimensions have been reduced to one dimension by appropriate indexing through the frequency and time locations of the resource elements of interest, typically for a single physical channel.

Data Types: `double`
Complex Number Support: Yes

### `csi` — Soft channel state information

3-D numeric array | 2-D numeric matrix

Soft channel state information, returned as a 3-D numeric array or a 2-D numeric matrix of the same size as `out`. As a 3-D numeric array, it has size *N*-by-*M*-by-*P*. *N* is the number of subcarriers, *M* is the number of OFDM symbols, and *P* is the number of transmit antennas. `csi` provides an estimate of the received RE gain for each received RE.

Alternatively, if `channelest` is provided as a 3-D array, `csi` is a 2-D numeric matrix of size (*N*×*M*)-by-*P*. In this case, the first two dimensions have been reduced to one dimension by appropriate indexing through the frequency and time locations of the resource elements of interest, typically for a single physical channel.

Data Types: `double`

# Version History

**Introduced in R2014a**

## See Also

`lteEqualizeMMSE` | `lteEqualizeMIMO` | `lteEqualizeULMIMO` | `lteOFDMDemodulate` | `lteSCFDMADemodulate` | `lteDLChannelEstimate` | `lteULChannelEstimate`

# lteExtractResources

Extract resource elements

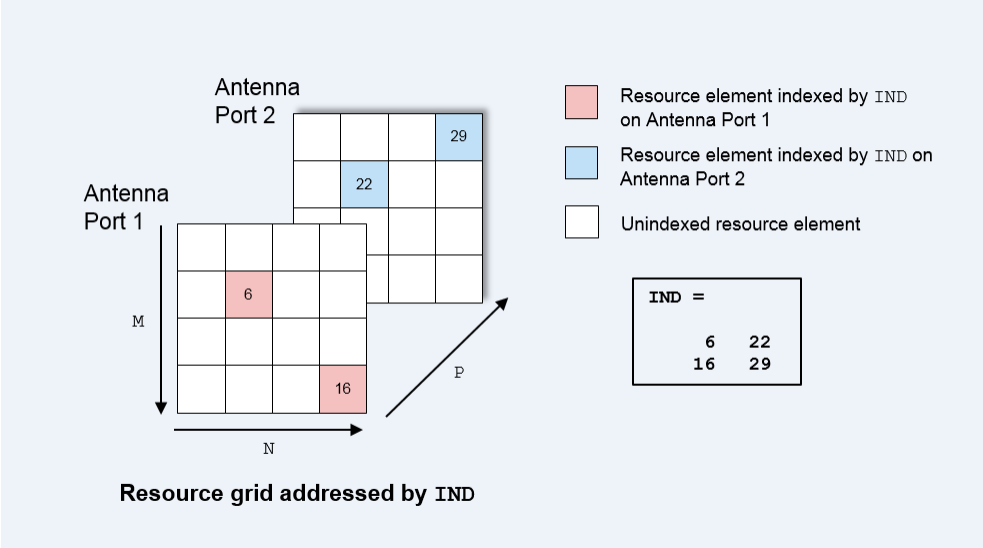
## Syntax

```
[re,reind] = lteExtractResources(ind,grid)
[re1,...,reK,reind1,...,reindK] = lteExtractResources(ind,grid1,...,gridK)
re = lteExtractResources( ___ ,opts)
```

## Description

`[re,reind] = lteExtractResources(ind,grid)` extracts resource elements `re` their indices `reind` from resource array `grid` using resource elements indices `ind`. You can extract resource elements from a resource grid with different dimensionality than the resource grid addressed by the indices. The indices specified and returned are in 1–based linear indexing form. Other indexing options are available. The resource extraction process is further explained in “Algorithms” on page 2-315.

In LTE Toolbox, indices are generated for mapping sequences of physical channel and signal symbols to a resource grid. These indices are generated using channel-or signal-specific functions and address resource elements in an array sized, *M*-by-*N*-by-*P*. *M* is the number of subcarriers, *N* is the number of OFDM or SC-FDMA symbols and *P* is the number of planes. The diagram highlights the resource elements of a resource grid addressed by indices, `ind`. The indices are in a 1–based linear indexing form. *P* = *2* is the number of antenna ports.




Typically the resource array extracts resource elements from one of the following:

- A 3-D received grid, sized *M*-by-*N*-by-*NRxAnts*. *NRxAnts* is the number of receive antennas. This grid is created after OFDM or SC-FDMA demodulation.
- A 4-D channel estimation grid, sized *M*-by-*N*-by-*NRxAnts*-by-*P*. This grid is created by channel estimation functions (refer “Channel Estimation”).

You can describe the size of the 3- D received grid as a 4-D grid that has a trailing singleton dimension.

`[re1,...,reK,reind1,...,reindK] = lteExtractResources(ind,grid1,...,gridK)` extracts resource elements from *K* resource arrays by using the specified resource element indices.

`re = lteExtractResources( ___ ,opts)` specifies the format of the indices and the extraction method used with a cell array of options, `opts`.

## Examples

### Extract PDCCH Symbols and Channel Estimates for Decoding

Extract PDCCH symbols from a received grid and associated channel estimates in preparation for decoding.

Create a transmit waveform for one subframe.

```
enb = lteRMCDL('R.12');
enb.TotSubframes = 1;
txWaveform = lteRMCDLTool(enb,[1;0;0;1]);
```

Receive sum of transmit antenna waveforms on three receive antennas.

```
NRxAnts = 3;
rxWaveform = repmat(sum(txWaveform,2),1,NRxAnts);
rxGrid = lteOFDMDemodulate(enb,rxWaveform);
```

Compute the channel estimation.

```
cec.FreqWindow = 1;
cec.TimeWindow = 1;
cec.InterpType = 'cubic';
cec.PilotAverage = 'UserDefined';
cec.InterpWinSize = 3;
cec.InterpWindow = 'Causal';
[hEstGrid,nEst] = lteDLChannelEstimate(enb,cec,rxGrid);
```

Generate PDCCH indices and extract symbols from received and channel estimate grids in preparation for PDCCH decoding.

```
ind = ltePDCCHIndices(enb);
[pdcchRxSym,pdcchHestSym] = lteExtractResources(ind,rxGrid,hEstGrid);
```

pdcchRxSym is sized `NRE-by-NRxAnts` and `pdcchHestSym` is sized `NRE-by-NRxAnts-by-CellRefP`.

```
rxSymSize = size(pdcchRxSym)
```

```
rxSymSize = 1×2

   212     3
```

```
hestSymSize = size(pdcchHestSym)
```

```
hestSymSize = 1×3

   212     3     4
```

Decode PDCCH with extracted resource elements.

```
pdcchBits = ltePDCCHDecode(enb,pdcchRxSym,pdcchHestSym,nEst);
```

### Extract Resources From 3D Receive Grid and 4D Channel Estimate Grid

Extract resources from a 3D receive grid and 4D channel estimate grid. Show the location of the indices within the grid.

Setup sizes of the grids: [M N P] and [M N NRxAnts P], where M is the number of subcarriers, N is the number of OFDM symbols, NRxAnts is the number of rx antennas, and P is the number of tx antennas.

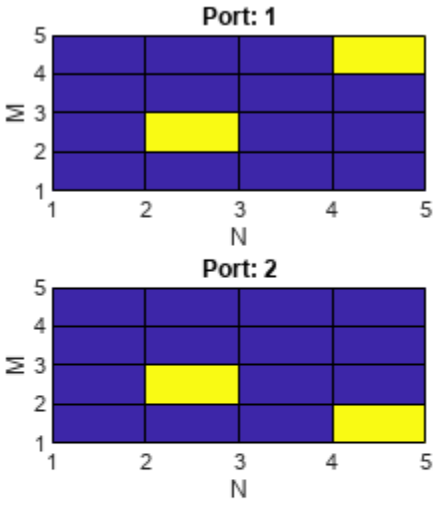
```
M = 4;
N = 4;
P = 2;
NRxAnts = 3;
```

Create indices and show the locations within the transmit grid addressed by these indices. As you will notice, different resource elements are addressed on each antenna port. Addressed resource element locations contain 1.

```
ind = [6 22; 16 29];
txGrid = zeros(M,N,P);
txGrid(ind) = 1;
```

Visualize locations of indexed resource elements in the transmit grid.

```
visualizeGrid = zeros(M+1,N+1,P);
visualizeGrid(1:M,1:N,:) = txGrid;

figure

subplot(321)
pcolor(visualizeGrid(:,:,1))
title('Port: 1')
xlabel('N')
ylabel('M')

subplot(323)
pcolor(visualizeGrid(:,:,2))
title('Port: 2')
xlabel('N')
ylabel('M')
```



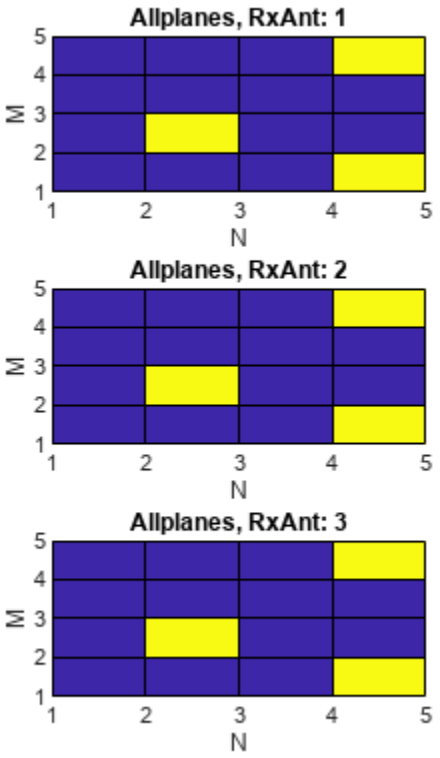

Create a 3D received grid to extract resource elements. Extract resource elements from the received grid. Show the locations of these extracted resource elements. Addressed resource element locations contain 1.

```
rxGrid = zeros(M,N,NRxAnts);

[re, indOut] = lteExtractResources(ind,rxGrid);
rxGrid(indOut) = 1;
```

Visualize locations of indexed resource elements in the receive grid.

```
figure
visualizeGrid = zeros(M+1,N+1,NRxAnts);
visualizeGrid(1:M,1:N,:) = rxGrid;

subplot(321)
pcolor(visualizeGrid(:,:,1))
title('Allplanes, RxAnt: 1');
xlabel('N')
ylabel('M')

subplot(323)
pcolor(visualizeGrid(:,:,2))
title('Allplanes, RxAnt: 2')
xlabel('N')
ylabel('M')
```

```
subplot(325)
pcolor(visualizeGrid(:,:,3))
title('Allplanes, RxAnt: 3')
xlabel('N')
ylabel('M')
```



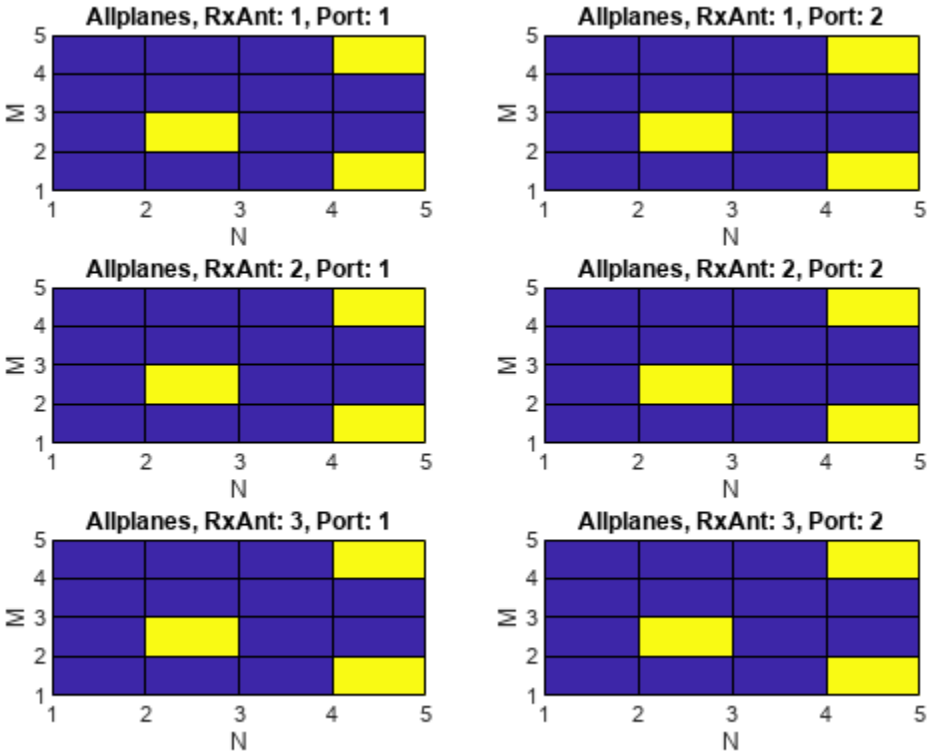

Create a 4D channel estimate grid to extract resource elements. Extract resource elements from the channel estimate grid. Show the locations of these extracted resource elements. Addressed resource element locations contain 1.

```
hEstGrid = zeros(M,N,NRxAnts,P);

[re, indOut] = lteExtractResources(ind,hEstGrid);
hEstGrid(indOut) = 1;
```

Visualize locations of the resource elements extracted using `'allplanes'` mode from 3D receive grid.

```
figure;
visualizeGrid = zeros(M+1,N+1,NRxAnts,P);
visualizeGrid(1:M,1:N,:,:) = hEstGrid;

subplot(321)
pcolor(visualizeGrid(:,:,1,1))
title('Allplanes, RxAnt: 1, Port: 1')
xlabel('N')
ylabel('M')
```

```
subplot(323)
pcolor(visualizeGrid(:,:,2,1))
title('Allplanes, RxAnt: 2, Port: 1')
xlabel('N')
ylabel('M')

subplot(325)
pcolor(visualizeGrid(:,:,3,1))
title('Allplanes, RxAnt: 3, Port: 1')
xlabel('N')
ylabel('M')

subplot(322)
pcolor(visualizeGrid(:,:,1,2))
title('Allplanes, RxAnt: 1, Port: 2')
xlabel('N')
ylabel('M')

subplot(324)
pcolor(visualizeGrid(:,:,2,2))
title('Allplanes, RxAnt: 2, Port: 2')
xlabel('N')
ylabel('M')

subplot(326)
pcolor(visualizeGrid(:,:,3,2))
title('Allplanes, RxAnt: 3, Port: 2')
xlabel('N')
ylabel('M')
```



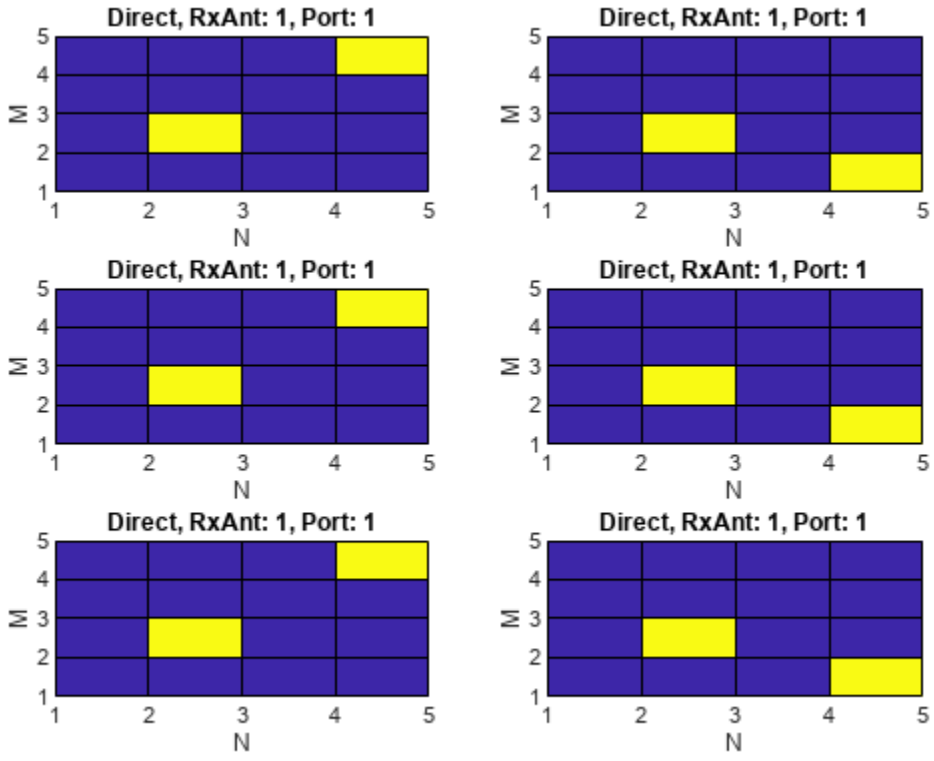

Create a 4D channel estimate grid to extract resource elements. Extract resource elements from the channel estimate grid using `'direct'` extraction mode. Show the locations of these extracted resource elements. Addressed resource element locations contain 1.

```
hEstGridDirect = zeros(M,N,NRxAnts,P);

[re, indOut] = lteExtractResources(ind,hEstGridDirect,'direct');
hEstGridDirect(indOut) = 1;
```

Visualize locations of the resource elements extracted using `'direct'` mode from 4D channel estimate grid.

```
figure
visualizeGrid = zeros(M+1,N+1,NRxAnts,P);
visualizeGrid(1:M,1:N,:,:) = hEstGridDirect;

subplot(321)
pcolor(visualizeGrid(:,:,1,1))
title('Direct, RxAnt: 1, Port: 1')
xlabel('N')
ylabel('M')

subplot(323)
pcolor(visualizeGrid(:,:,2,1))
title('Direct, RxAnt: 1, Port: 1')
xlabel('N')
ylabel('M')
```

```
subplot(325)
pcolor(visualizeGrid(:,:,3,1))
title('Direct, RxAnt: 1, Port: 1')
xlabel('N')
ylabel('M')

subplot(322)
pcolor(visualizeGrid(:,:,1,2))
title('Direct, RxAnt: 1, Port: 1')
xlabel('N')
ylabel('M')

subplot(324)
pcolor(visualizeGrid(:,:,2,2))
title('Direct, RxAnt: 1, Port: 1')
xlabel('N')
ylabel('M')

subplot(326)
pcolor(visualizeGrid(:,:,3,2))
title('Direct, RxAnt: 1, Port: 1')
xlabel('N')
ylabel('M')
```



### Extract Cell-Specific Reference Signal (CRS) Symbols

Use `'direct'` and `'allplanes'` extraction methods and subscript indices to extract cell-specific reference signal (CRS) symbols in subcarrier 7 from `grid`.

Generate a resource grid and CRS indices in the subscript form: [subcarrier, OFDM symbol, CRS port].

```
enb = lteRMCDL('R.12');
enb.TotSubframes = 1;
enb.CellRefP = 2;
enb.PDSCH.NLayers = 2;
[waveform,grid] = lteRMCDLTool(enb,[1;0;0;1]);
crsInd = lteCellRSIndices(enb,'sub');
```

There are 2 resource elements used on CRS ports 1 & 2; all are on different OFDM symbols (1, 5, 8, 12).

```
crsIndSC7 = crsInd(crsInd(:,1)==7,:)
```

```
crsIndSC7 = 4x3 uint32 matrix

    7    1    1
    7    8    1
    7    5    2
    7   12    2
```

Use `'direct'` method to extract resource elements. The extracted resource element indices are same as the generated CRS indices as the resource array indexed by `crsInd` in `grid`.

```
[dirREs,dirInd] = lteExtractResources(crsInd,grid,{'direct','sub'});
directIndSC7 = dirInd(dirInd(:,1)==7,:)
```

```
directIndSC7 = 4x3 uint32 matrix

    7    1    1
    7    8    1
    7    5    2
    7   12    2
```

Use `'allplanes'` method to extract resource elements. There are 4 extracted CRS indices as per the CRS port on subcarrier 7. Indices addressing unique OFDM symbols in the indexed resource grid are used to extract resource elements from all the CRS ports in `'grid`. Therefore indices are extracted at OFDM symbols (1, 5, 8,12) on both CRS ports.

```
[apREs,apInd] = lteExtractResources(crsInd,grid,{'allplanes','sub'});
allPlanesIndSC7 = apInd(apInd(:,1)==7,:)
```

```
allPlanesIndSC7 = 8x3 uint32 matrix

    7    1    1
    7    8    1
    7    5    1
    7   12    1
    7    1    2
    7    8    2
```

```
     7     5     2
     7    12     2
```

## Input Arguments

### `ind` — Resource elements indices

numeric array

Resource elements indices, specified as a numeric array. The indices address elements of a *N*-by-*M*-by-*P* resource array. *M* is the number of subcarriers, *N* is the number of OFDM or SC-FDMA symbols, and *P* is the number of planes.

If you specify an element of this array as a value greater than the number of elements in the `grid` input, the function uses the value of `mod(ind,numel(grid))`.

### `grid` — Resource array

3–D numeric array (default) | 4–D numeric array

Resource array, specified as a 3-D or 4-D numeric array. Typically the resource array to extract resource elements from in one of the following:

- A 3–D received grid, sized *M*-by-*N*-by-*NRxAnts*. *NRxAnts* is the number of receive antennas. This grid is created after OFDM or SC-FDMA demodulation.
- A 4–D channel estimation grid, sized *M*-by-*N*-by-*NRxAnts*-by-*P*. This grid is created by channel estimation functions (refer “Channel Estimation”).

You can describe the size of the 3D received grid as a 4D grid that has a trailing singleton dimension.

Data Types: `double`

### `opts` — Resource elements extraction options

character vector | cell array of character vectors | string array

Resource elements extraction options, specified as a character vector, cell array of character vectors, or string array. Values for `opts` when specified as a character vector include (use double quotes for string):

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`Indexing Style`** | Required | `'ind'` (default) or `'sub'` | Indexing style of the specified or returned indices, `ind` and `reind`, specified as one of the following options:<br>• `'ind'` — linear index form<br>• `'sub'` — subscript form |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`Index Base`** | Required | `'1based'` (default) or `'0based'` | Base value of the specified or returned indices, `ind` and `reind`, specified as one of the following options:<br>• `'1based'` — the first value of index sequence is one<br>• `'0based'` — the first value of the index sequence is zero |
| **`Extraction Method`** | Required | `'allplanes'` (default) or `'direct'` | Resource element extraction methods. The methods are described in “Algorithms” on page 2-315.<br>• `'allplanes'` — uses indices addressing unique subcarrier and symbol location over all planes of the indexed resource array for extraction.<br>• `'direct'` — only resource elements relevant to each plane of the indexed resource grid are extracted. |

## Output Arguments

### `re` — Extracted resource elements

column vector | numeric array

Extracted resource elements, returned as a column vector or numeric array.

When `'allplanes'` extraction method is used, the extracted resource elements array is of size $N_{RE}$-by-*NRxAnts*-by-*P* where:

- $N_{RE}$ is the number of resource elements per *M*-by-*N* plane of `grid`.
- *M* is the number of subcarriers.
- *N* is the number of OFDM or SC-FDMA symbols.
- *P* is the number of planes.

When using `'direct'` extraction method, the size of the extracted resource elements array, `re`, depends on the number of indices addressing each plane of the indexed source grid:

- If the same number of indices address each plane then `re` is of size $N_{RE}$-by-*NRxAnts*-by-*P*.
- If a different number of indices address each plane then `re` is a column vector containing all extracted resource elements.

### `reind` — Indices of extracted resource elements

numeric array

Indices of extracted resource elements within `grid`, returned as numeric array. `reind` is the same size as extracted resource elements array `re`.

## Algorithms

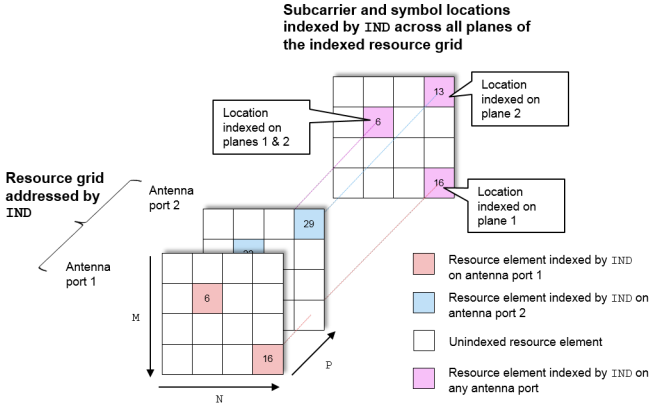
`lteExtractResources` can extract resource elements using one of two methods. The `'allplanes'` method is used by default. You can optionally specify `'direct'` extraction method.

### All Planes Extraction Method

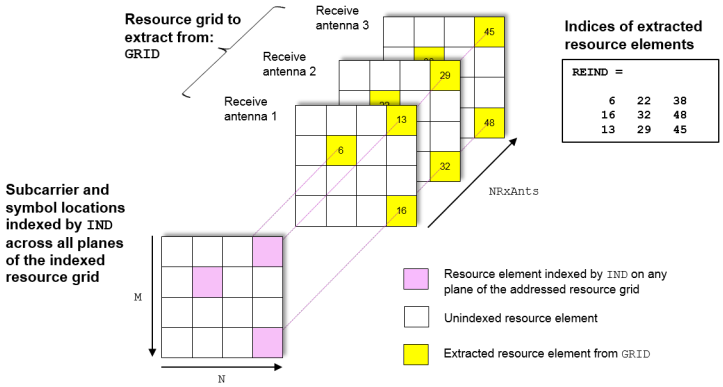
The `'allplanes'` method extracts resource elements from each *M*-by-*N* plane within `grid` using indices that address unique subcarrier and symbol locations over all the planes of the indexed resource array.

The following diagrams illustrate the resource extraction process for a 3D received grid and a 4D channel estimation grid. The example, “Extract Resources From 3D Receive Grid and 4D Channel Estimate Grid” on page 2-306 recreates these diagrams.

Indices addressed by unique subcarrier and symbol locations across all planes of the indexed resource gird are used for the extraction. The diagram highlights the indices used to extract resource elements address the resource grid with $P = 2$. In this case, $P$ is the number of antenna ports.

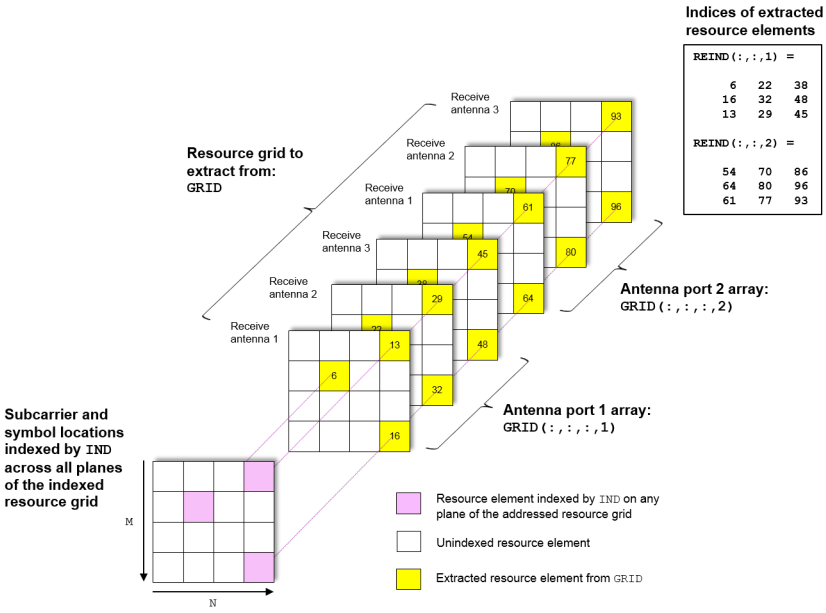



Resource elements are extracted from `grid` at the symbol and subcarrier locations. The following diagrams illustrate the resource element extraction from a 3D received grid, `grid`, with *NRxAnts* = *3*.




The following diagram shows the extraction process for a 4D channel estimate grid, `grid`, with *NRxAnts* = *3* and *P*= *2*. In this case, *P* is the number for antenna ports. The 4D resource grid consists

of *P* *M*-by-*N*-by-*NRxAnts* arrays, each associated with an antenna port. Resource elements are extracted from all planes within these arrays.

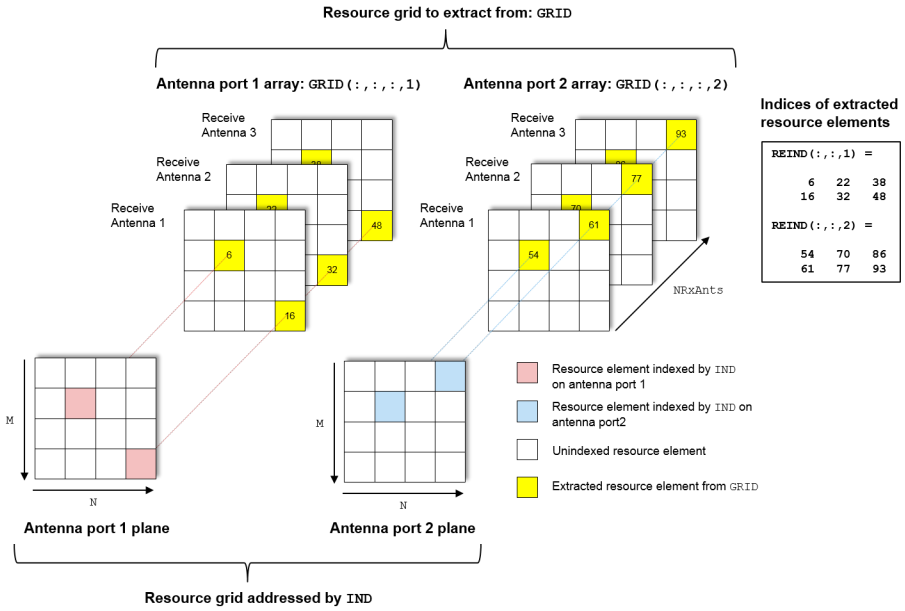



### Direct Extraction Method

The `'direct'` method extracts resource elements from `grid` with the assumption that third and fourth dimension of the `grid` represents the same property as the planes of the indexed resource array such as antenna ports, layers, transmit antennas. Therefore the only resource elements relevant to each plane of the indexed resource grid are extracted:

- For a 3D `grid`, the `'direct'` method extracts elements from each *M*-by-*N* plane of `grid` using indices addressing the same plane of the indexed resource array. This is the same as the standard MATLAB operation `re = grid(ind)`. Therefore `reind = ind`.
- For a 4D `grid`, the `'direct'` method extracts elements from each *M*-by-*N*-by-*NRxAnts* array of `grid` using indices addressing the same plane of the indexed resource array. Therefore it is assumed the property represented by the planes of the indexed resource array is the same as the fourth dimension of `grid`.

The extraction of a 4D estimation grid, `grid`, using the `'direct'` method is illustrated in the following diagram with *NRxAnts* = *3* and *P*= *2*, which is the number of antenna ports. The 4D resource grid consists of *P* *M*-by-*N*-by-*NRxAnts* arrays, each associated with an antenna port. Therefore the indices corresponding to each individual antenna port in the indexed resource array are used to extract resource elements from each of these arrays. The example, “Extract Resources From 3D Receive Grid and 4D Channel Estimate Grid” on page 2-306 creates a version of this diagram.




# Version History

**Introduced in R2014b**

## See Also

`ltePUSCHDecode` | `ltePUSCHIndices` | `ltePDCCHDecode` | `ltePDCCHIndices` | `ltePBCHDecode` | `ltePBCHIndices` | `ltePDSCHDecode` | `ltePDSCHIndices` | `ltePDCCHDecode` | `ltePDCCHIndices` | `ltePCFICHDecode` | `ltePCFICHIndices` | `ltePHICHDecode` | `ltePHICHIndices` | `lteCellRSIndices` | `lteSCFDMADemodulate` | `lteOFDMDemodulate` | `lteDLChannelEstimate` | `lteULChannelEstimate` | `ltePUCCH1Decode` | `ltePUCCH1Indices` | `lteULChannelEstimatePUCCH1` | `ltePUCCH2Decode` | `ltePUCCH2Indices` | `lteULChannelEstimatePUCCH2` | `ltePUCCH3Decode` | `ltePUCCH3Indices` | `lteULChannelEstimatePUCCH3` | `lteDLResourceGrid` | `lteULResourceGrid`

# lteFadingChannel

Multipath fading MIMO channel propagation conditions

## Syntax

```
[rx,info] = lteFadingChannel(model,tx)
```

## Description

`[rx,info] = lteFadingChannel(model,tx)` filters waveform `tx` through the Rayleigh fading channel parameterized by `model`. The function returns channel output waveform `rx` and channel model information `info`. For information about the multiple-input multiple-output (MIMO) multipath fading channel that this function implements, see “Fading Channel Model Delay” on page 2-326.

## Examples

### Transmit Multiple Subframes over Fading Channel

Define the channel configuration structure.

```
model = struct(DelayProfile="EPA",NRxAnts=1, ...
    DopplerFreq = 5,MIMOCorrelation="Low", ...
    Seed=1,InitPhase="Random",ModelType="GMEDS", ...
    NTerms=16,NormalizeTxAnts="On", ...
    NormalizePathGains="On");
```

Define the transmission waveform configuration structure, initialized to reference measurement channel (RMC) R.10 and one subframe.

```
rmc = lteRMCDL("R.10");
rmc.TotSubframes = 1;
```

Generate ten subframes, one subframe at a time, by following these steps.

1. Define `delay`, which accounts for a combination of implementation delay and channel delay spread.
2. Set the subframe number and initialize the subframe start time, allocating 1 ms per subframe.
3. Generate a transmit waveform.
4. Initialize the number of transmit antennas and the waveform sampling rate.
5. Send the waveform through the channel. Append `delay` zeros to the generated waveform prior to channel filtering.

```
delay = 25;
for subframeNumber = 0:9

    rmc.NSubframe = mod(subframeNumber,10);
    model.InitTime = subframeNumber/1000;

    [waveform,txGrid,info] = lteRMCDLTool(rmc,[1; 0; 1; 1]);
```

```
    numTxAnt = size(waveform,2);
    model.SamplingRate = info.SamplingRate;
    tx = [waveform; zeros(delay,numTxAnt)];

    [rx,info] = lteFadingChannel(model,tx);
end
```

### Transmit Two Consecutive Frames over Fading Channel

Transmit two consecutive frames over a fading channel while maintaining continuity in the fading process between the end of the first frame and the beginning of the second frame.

Initialize a resource grid to RMC R.10 and generate a transmit waveform for the first frame.

```
rmc = lteRMCDL("R.10");
[waveform,~,info] = lteRMCDLTool(rmc,[1; 0; 1]);
```

Initialize a propagation channel configuration structure and set the start time for the first frame.

```
model = struct(DelayProfile="EPA",NRxAnts=1, ...
    DopplerFreq=5,MIMOCorrelation="Low", ...
    SamplingRate=info.SamplingRate,Seed=1, ...
    InitPhase="Random",ModelType="GMEDS", ...
    NTerms=16,NormalizeTxAnts="On", ...
    NormalizePathGains="On",InitTime=0);
nTxAnts = size(waveform,2);
```

Define `delay` and append zeros to the generated waveform prior to channel filtering.

```
delay = 25;
tx = [waveform; zeros(delay,nTxAnts)];
```

Filter the first frame through the channel.

```
[rx1,info1] = lteFadingChannel(model,tx);
```

Update the frame number, and then generate a transmit waveform for the second frame with the start time set to 10 ms.

```
model.NFrame = 1;
[waveform,txGrid] = lteRMCDLTool(rmc,[1; 0; 1]);
tx = [waveform; zeros(delay,nTxAnts)];
model.InitTime = 10e-3;
```

Pass the second frame through the channel.

```
[rx2,info2] = lteFadingChannel(model,tx);
```

## Input Arguments

### `model` — Multipath fading channel model

structure

Multipath fading channel model, specified as a structure containing these fields.

| Field | Required or Optional | Values | Description | Dependencies |
|---|---|---|---|---|
| `NRxAnts` | Required | Positive integer | Number of receive antennas | Not applicable |
| `MIMOCorrelation` | Required | `"Low"`, `"Medium"`, `"UplinkMedium"`, `"High"`, `"Custom"` | Correlation between UE and eNodeB antennas.<br>• To specify no correlation between antennas, set this field to `"Low"`<br>• To specify the correlation level defined in Annex B.2.3.2 of TS 36.101 [1], which applies to tests defined in TS 36.101, set this field to `"Medium"`<br>• To specify the correlation level defined in Annex B.5.2 of TS 36.104 [2], which applies to tests defined in TS 36.104, set this field to `"UplinkMedium"`<br>• To specify strong correlation between antennas, set this field to `"High"`<br>• To specify correlation between antennas in the `TxCorrelationMatrix` and `RxCorrelationMatrix` fields, set this field to `"Custom"`.<br>**Note** Because the `"Low"` and `"High"` correlation levels are the same for uplink and downlink, they apply to tests defined in TS 36.101 and TS 36.104. | |
| `NormalizeTxAnts` | Optional | `"On"` (default), `"Off"` | Transmit antenna number normalization. To normalize the output waveform by $1/\sqrt{P}$, where $P$ is the number of transmit antennas, set this field to `"On"`. Normalization by the number of transmit antennas ensures that the number of transmit antennas does not affect the output power per receive antenna. | |

| Field | Required or Optional | Values | Description | Dependencies |
|---|---|---|---|---|
| `DelayProfile` | Required | `"EPA"`, `"EVA"`, `"ETU"`, `"Custom"`, `"Off"` | Delay profile model. For more information, see “Propagation Channel Models”.<br><br>To completely switch off fading and implement a MIMO channel model that is constant in time and frequency, set this field to `"Off"`. In this case, the number of columns in the `in` input specifies the number of transmit antennas, the `NRxAnts` field specifies the number of receive antennas, and the `MIMOCorrelation` field specifies the MIMO correlation. The temporal part of the model for each link between transmit and receive antennas consists of a single path with zero delay and constant unit gain. This setting does not implement the channel model defined in Annex B.1 of [1]. | |
| `DopplerFreq` | Required | Nonnegative scalar | Maximum Doppler frequency in Hz | To enable these fields, set the `DelayProfile` field to a value other than `"Off"`. |
| `SamplingRate` | Required | Positive scalar | Input waveform sampling rate | |
| `InitTime` | Required | Nonnegative scalar | Fading process time offset in seconds | |
| `NTerms` | Optional | `16` (default)<br><br>Power of 2 | Number of oscillators used in fading path modeling | |

| Field | Required or Optional | Values | Description | Dependencies |
|---|---|---|---|---|
| `ModelType` | Optional | `"GMEDS"` (default), `"Dent"` | Rayleigh fading model type.<br>• To model Rayleigh fading by using the generalized method of exact Doppler spread (GMEDS) described in [4], set this field to `"GMEDS"`.<br>• To model Rayleigh fading by using the modified Jakes fading model described in [3], set this field to `"Dent"`.<br>**Note** Setting this field to `"Dent"` is not recommended. Use `"GMEDS"` instead. | |
| `NormalizePathGains` | Optional | `"On"` (default), `"Off"` | Model output normalization.<br>• To normalize the `rx` output such that the average power is unity, set this field to `"On"`.<br>• To normalize the `rx` output such that the average output power is the sum of the powers of the taps of the delay profile, set this field to `"Off"`. | |

| Field | Required or Optional | Values | Description | Dependencies |
|---|---|---|---|---|
| `InitPhase` | Optional | `"Random"` (default)<br><br>Scalar<br><br>4-D array | Phase initialization for the sinusoidal components of the model.<br>• To randomly initialize the phases according to the value of the `Seed` field, set this field to `"Random"`.<br>• To specify the same phase, in radians, of all components, set this field to a scalar.<br>• To specify the phase, in radians, of each component explicitly, set this field to a 4-D array of size *N*-by-*L*-by-*P*-by-`NRxAnts`.<br>  • *N* is the number of phase initialization values per path.<br>  • *L* is the number of paths.<br>  • *P* is the number of transmit antennas.<br>  • `NRxAnts` is the number of receive antennas.<br>**Note**<br>• When you set the `ModelType` field to `"GMEDS"`, *N* is equal to 2 ×`NTerms`.<br>• When you set the `ModelType` field to `"Dent"`, *N* is equal to `NTerms`. | |

| Field | Required or Optional | Values | Description | Dependencies |
|---|---|---|---|---|
| `Seed` | Required | Scalar | Random number generator seed. To use a random seed, set this field to `0`.<br><br>**Note**<br>• To produce distinct results, set this field to a value in the interval<br>$[0, 2^{31} - 1 - K(K-1)/2]$,<br>where $K = P \times$ `model.NRxAnts`, which is the product of the number of transmit and receive antennas. Avoid using values outside of this recommended range, because doing so can result in random sequences that repeat results produced using values inside the recommended range.<br>• The state of MATLAB random number generators, for example by calls to the `rng` function, does not affect fading channel random seed behavior. | To enable this field, set the `DelayProfile` field to a value other than `"Off"` and the `InitPhase` field to `"Random"`. |
| `AveragePathGaindB` | Required | Vector | Average gains of the discrete paths in dB | To enable these fields, set the `DelayProfile` field to `"Custom"`. |
| `PathDelays` | Required | Vector | Delays of the discrete paths in seconds. This vector must be the same size as `AveragePathGaindB`. If these delays are not a multiple of the sampling period, the function uses fractional delay filters to implement them. | |
| `TxCorrelationMatrix` | Required | Complex-valued matrix | Correlation between each of the transmit antennas, specified as a complex-valued matrix of size *P*-by-*P*. | To enable these fields, set the `MIMOCorrelation` field to `"Custom"`. |

| Field | Required or Optional | Values | Description | Dependencies |
|---|---|---|---|---|
| `RxCorrelationMatrix` | Required | Complex-valued matrix | Correlation between each of the receive antennas, specified as a complex-valued matrix of size `NRxAnts`-by-`NRxAnts`. | |

Data Types: `struct`

**`tx` — Input samples**
complex-valued matrix

Input samples, specified as a complex-valued matrix of size *T*-by-*P*, where *T* is the number of time-domain samples, and *P* is the number of transmit antennas. Each column of this input corresponds to the waveform at one transmit antenna.

Data Types: `double` | `single`
Complex Number Support: Yes

## Output Arguments

**`rx` — Channel output waveform**
complex-valued matrix

Channel output waveform, returned as a complex-valued matrix. Each column of `rx` corresponds to the waveform at a receive antenna. This output has the same number of rows as the `tx` input.

Data Types: `double` | `single`
Complex Number Support: Yes

**`info` — Channel modeling information**
structure

Channel modeling information, returned as a structure containing these fields.

| Parameter Field | Values | Description |
|---|---|---|
| `ChannelFilterDelay` | Scalar value | Implementation delay of the internal channel filtering, in samples |
| `PathGains` | Numeric array | Complex gain of the discrete channel paths, returned as a numeric array of size *T*-by-*L*-by-*P*-by-`NRxAnts`.<br>• *T* is the number of output samples.<br>• *L* is the number of paths.<br>• *P* is the number of transmit antennas.<br>• `NRxAnts` is the number of receive antennas. |
| `PathSampleDelays` | Row vector | Delays of the discrete channel paths, in samples, returned at the sampling rate specified in the `SamplingRate` field of the `model` input. |

| Parameter Field | Values | Description |
|---|---|---|
| `AveragePathGaindB` | Row vector | Average gains of the discrete paths in dB |

Data Types: `struct`

## Algorithms

### Fading Channel Model Delay

This function implements the MIMO multipath fading channel model specified in [1] and [2]. The transmitted waveform passes through the multipath Rayleigh fading channel model specified by the input structure `model`. The function resamples the delay profile of the `model` input to match the input waveform sampling rate. When the path delays are not a multiple of the sampling rate, the function uses fractional delay filters to implement them. These filters introduce an implementation delay of `info.ChannelFilterDelay` samples. The waveform passing through the channel passes through these filters and incurs this channel filter delay regardless of the value of the path delays.

# Version History

**Introduced in R2013b**

## References

[1] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. https://www.3gpp.org.

[2] 3GPP TS 36.104. “Evolved Universal Terrestrial Radio Access (E-UTRA); Base Station (BS) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. https://www.3gpp.org.

[3] Dent, P., G. E. Bottomley, and T. Croft. “Jakes Fading Model Revisited.” *Electronics Letters*. 29, no. 13 (1993): 1162–1163.

[4] Pätzold, Matthias, Cheng-Xiang Wang, and Bjørn Olav Hogstad. “Two New Sum-of-Sinusoids-Based Methods for the Efficient Generation of Multiple Uncorrelated Rayleigh Fading Waveforms.” *IEEE Transactions on Wireless Communications*. 8, no. 6 (2009): 3122–3131.

## See Also

`lteMovingChannel` | `lteHSTChannel` | `lteOFDMModulate` | `lteSCFDMAModulate` | `lteDLPerfectChannelEstimate` | `lte3DChannel`

# lteFrequencyCorrect

Frequency offset correction

## Syntax

```
out = lteFrequencyCorrect(cfg,in,foffset)
```

## Description

`out = lteFrequencyCorrect(cfg,in,foffset)` corrects for a specified frequency offset, `foffset`, in the time-domain waveform, `in`, by performing simple frequency modulation (FM). The parameters of the waveform, `in`, are specified in a settings structure, `cfg`, which must contain either the field `NDLRB` or `NULRB` to control whether a downlink or uplink signal is expected in `in`.

The input, `foffset` is the frequency offset, in hertz, present on the waveform, `in`. Therefore, the correction applied is FM modulation by –`foffset`.

## Examples

### Correct for Specified Frequency Offset

Perform frequency offset estimation and correction on an uplink signal, to which a frequency offset has been applied.

Generate uplink RMC A3-2.

```
[txWaveform,rgrid,cfg] = lteRMCULTool('A3-2',[1;0;0;1],'Fdd',2);
```

Apply an arbitrary frequency offset of 51.2 Hz.

```
t = (0:length(txWaveform)-1).'/cfg.SamplingRate;
txWaveform = txWaveform .* exp(1i*2*pi*51.2*t);
```

Estimate and display the frequency offset.

```
offset = lteFrequencyOffset(cfg,txWaveform);
disp(['Frequency offset: ' num2str(offset) ' Hz'])
```

```
Frequency offset: 51.2 Hz
```

Correct for the frequency offset.

```
rxWaveform = lteFrequencyCorrect(cfg,txWaveform,offset);
```

Finally, perform SC-FDMA demodulation.

```
rxGrid = lteSCFDMADemodulate(cfg,rxWaveform);
```

## Input Arguments

### `cfg` — Waveform parameter settings

structure

Waveform parameter settings, specified as a structure. `cfg` must contain either the field `NDLRB`, to specify a downlink configuration, or the field `NULRB`, to specify an uplink configuration.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NDLRB`** | Required | Positive scalar integer | Number of downlink resource blocks ($N_{RB}^{DL}$)<br><br>Set this parameter field to specify a downlink configuration. |
| **`CyclicPrefix`** | Required | `'Normal'` (default), `'Extended'` | Cyclic prefix length in the downlink<br><br>Only set this parameter field if you are specifying a downlink configuration. |
| **`NULRB`** | Required | Scalar integer from 6 to 110 | Number of uplink resource blocks. ($N_{RB}^{UL}$)<br><br>Set this parameter field to specify an uplink configuration. |
| **`CyclicPrefixUL`** | Required | `'Normal'` (default), `'Extended'` | Uplink cyclic prefix length. Only set this parameter field if you are specifying an uplink configuration. |

Data Types: `struct`

### `in` — Time-domain waveform

numeric column vector

Time-domain waveform, specified as a numeric column vector.

Data Types: `double` | `single`
Complex Number Support: Yes

### `foffset` — Waveform frequency offset

scalar value

Waveform frequency offset, specified as a scalar value expressed in Hertz. The correction applied to `in` is FM modulation by –`foffset`.

Data Types: `double`

## Output Arguments

### `out` — Offset-corrected waveform

numeric column vector

Offset-corrected waveform, returned as a numeric column vector.

Data Types: `double` | `single`
Complex Number Support: Yes

## Version History

**Introduced in R2014a**

## See Also

`lteFrequencyOffset` | `lteCellSearch` | `lteDLFrameOffset` | `lteULFrameOffset` | `lteOFDMDemodulate` | `lteSCFDMADemodulate`

# lteFrequencyOffset

Frequency offset estimation using cyclic prefix

## Syntax

```
foffset = lteFrequencyOffset(cfgdl,waveform)
foffset = lteFrequencyOffset(cfgul,waveform)
[foffset, corr] = lteFrequencyOffset( ___ )
[foffset, corr] = lteFrequencyOffset( ___ ,toffset)
```

## Description

`foffset = lteFrequencyOffset(cfgdl,waveform)` estimates the average frequency offset, `foffset`, of the time-domain waveform, `waveform`, by calculating correlation of the cyclic prefix. The parameters of `waveform` are given in the downlink settings structure, `cfgdl`. `cfgdl` must contain the field `NDLRB` to specify that a downlink signal is expected in `waveform`.

`foffset = lteFrequencyOffset(cfgul,waveform)` estimates the average frequency offset, `foffset`, of the time-domain waveform, `waveform`, by calculating correlation of the cyclic prefix. The parameters of `waveform` are given in the uplink settings structure, `cfgul`. `cfgul` must contain the field `NULRB` to specify that an uplink signal is expected in `waveform`.

`[foffset, corr] = lteFrequencyOffset( ___ )` also returns a complex matrix, `corr`, spanning one slot and containing the same number of antennas, or columns, as `waveform`. `corr` is the signal used to extract the timing of the correlation for the estimation of the frequency offset.

`[foffset, corr] = lteFrequencyOffset( ___ ,toffset)` provides control over the position in the correlator output used to estimate the frequency offset. When present `toffset` is the timing offset in samples from the start of the correlator output to the position used for the frequency offset estimation. This input allows a timing offset to be calculated externally on a signal of longer duration than the input `waveform`. Which allows a short-term frequency offset estimate to be obtained while retaining the benefit of a longer-term timing estimate.

---

**Note** If `toffset` is absent, the quality of the internal timing estimate is subject to the length and signal quality of the input `waveform` and, therefore, it may result in inaccurate frequency offset measurements.

---

## Examples

### Estimate Frequency Offset

Perform frequency offset estimation and correction on an uplink signal, to which a frequency offset has been applied.

Generate uplink RMC A3-2.

```
[txWaveform,rgrid,cfg] = lteRMCULTool('A3-2',[1;0;0;1],'Fdd',2);
```

Apply an arbitrary frequency offset of 51.2 Hz.

```
t = (0:length(txWaveform)-1).'/cfg.SamplingRate;
txWaveform = txWaveform .* exp(1i*2*pi*51.2*t);
```

Estimate and display the frequency offset.

```
offset = lteFrequencyOffset(cfg,txWaveform);
disp(['Frequency offset: ' num2str(offset) ' Hz'])
```

```
Frequency offset: 51.2 Hz
```

Correct for frequency offset.

```
rxWaveform = lteFrequencyCorrect(cfg,txWaveform,offset);
```

Perform SC-FDMA demodulation.

```
rxGrid = lteSCFDMADemodulate(cfg,rxWaveform);
```

## Input Arguments

### `cfgdl` — Downlink configuration

structure

Downlink configuration, specified as a structure having the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NDLRB`** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **`CyclicPrefix`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **`DuplexMode`** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| The following apply when `DuplexMode` is set to `'TDD'`. | | | |
| **`TDDConfig`** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **`SSC`** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |
| **`NSubframe`** | Optional | 0 (default), nonnegative scalar integer | Subframe number |

Data Types: `struct`

### `cfgul` — Uplink configuration

structure

Uplink configuration, specified as a structure having the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NULRB** | Required | Scalar integer from 6 to 110 | Number of uplink resource blocks. ($N_{RB}^{UL}$) |
| **CyclicPrefixUL** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **DuplexMode** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| The following apply when `DuplexMode` is set to `'TDD'`. | | | |
| **TDDConfig** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **SSC** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |
| **NSubframe** | Optional | 0 (default), nonnegative scalar integer | Subframe number |

Data Types: `struct`

**`waveform` — Input time-domain waveform**
numeric column vector

Input time-domain waveform, specified as a numeric column vector.

Data Types: `double` | `single`
Complex Number Support: Yes

**`toffset` — Timing offset**
scalar value

Timing offset, specified as a scalar value expressed in samples. Use `toffset` to control the position in the correlator output used to estimate the frequency offset. If `toffset` is absent, or empty, the position of the peak magnitude of the correlator output is used.

Data Types: `double` | `single`

## Output Arguments

**`foffset` — Average frequency offset estimate**
scalar value

Average frequency offset estimate, returned as a scalar value expressed in Hertz. This function can only accurately estimate frequency offsets of up to ±7.5 kHz (a range of 15 kHz, the subcarrier spacing).

Data Types: `double` | `single`

**`corr` — Correlation timing signal**
numeric matrix

Correlation timing signal, returned as a numeric matrix. `corr` is a complex matrix that spans one slot and contains the same number of antennas, or columns, as `waveform`. It is the signal used to extract the timing of the correlation for the frequency offset estimation.

Data Types: `double` | `single`
Complex Number Support: Yes

# Version History

**Introduced in R2014a**

## See Also

`lteFrequencyCorrect` | `lteCellSearch` | `lteDLFrameOffset` | `lteULFrameOffset` | `lteOFDMDemodulate` | `lteSCFDMADemodulate`

# lteHSTChannel

High-speed train MIMO channel propagation conditions

## Syntax

```
out = lteHSTChannel(model,in)
```

## Description

`out = lteHSTChannel(model,in)` implements the high-speed train (HST) MIMO channel model specified in TS 36.101 [1] and TS 36.104 [2]. The high-speed train propagation condition is composed of a non-fading single path of unit amplitude and zero phase with a changing Doppler shift. The columns of matrix `in` correspond to the channel input waveforms at each transmit antenna. The channel model filters `in` with the characteristics specified in structure `model`. The matrix `out` stores the filtered waveform. Each column of `out` corresponds to the waveform at one of the receive antennas.

## Examples

### Model High-Speed Train Propagation Channel

Generate a frame and filter it with the high-speed train channel model.

Create a reference channel configuration structure initialized to `'R.10'`. Generate a waveform.

```
rmc = lteRMCDL('R.10');
[txWaveform,txGrid,info] = lteRMCDLTool(rmc,[1;0;1]);
```

Initialize a propagation channel configuration structure for high-speed train profile. Pass the transmission waveform through the propagation channel.

```
chcfg.NRxAnts = 1;
chcfg.Ds = 100;
chcfg.Dmin = 500;
chcfg.Velocity = 350;
chcfg.DopplerFreq = 5;
chcfg.SamplingRate = info.SamplingRate;
chcfg.InitTime = 0;

rxWaveform = lteHSTChannel(chcfg,txWaveform);
```

## Input Arguments

### `model` — High-speed train propagation channel model

structure

High-speed train propagation channel model, specified as a structure containing these fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NRxAnts** | Required | Positive scalar integer | Number of receive antennas |
| **Ds** | Required | Numeric scalar | Train-to-eNodeB double initial distance, in meters.<br>`Ds`/2 is initial distance between train and eNodeB, in meters |
| **Dmin** | Required | Scalar value | eNodeB to railway track distance, in meters |
| **Velocity** | Required | Scalar value | Train velocity, in kilometers per hour |
| **DopplerFreq** | Required | Scalar value | Maximum Doppler frequency, in Hz. |
| **SamplingRate** | Required | Scalar value | Input signal sampling rate, the rate of each sample in the rows of the input matrix, `in`. |
| **InitTime** | Required | Scalar value | Doppler shift timing offset, in seconds |
| **NormalizeTxAnts** | Optional | `'On'` (default), `'Off'` | Transmit antenna number normalization, specified as:<br>• `'On'`, `lteHSTChannel` normalizes the model output by `1/sqrt(P)`, where *P* is the number of transmit antennas. Normalization by the number of transmit antennas ensures that the output power per receive antenna is unaffected by the number of transmit antennas.<br>• `'Off'`, normalization is not performed. |

Data Types: `struct`

**`in` — Channel input waveforms at transmit antennas**
numeric matrix

Channel input waveforms at transmit antennas, specified as a numeric matrix. `in` has size *T*-by-*P*, where *P* is the number of antennas and *T* is the number of time-domain samples. These waveforms are filtered with the high-speed train channel model with the Doppler shift as specified in parameter structure `model`.

Data Types: `double` | `single`
Complex Number Support: Yes

## Output Arguments

**`out` — Filtered waveform**
numeric matrix

Filtered waveform, returned as a numeric matrix. Each column of `out` corresponds to the waveform at one of the receive antennas.

Data Types: `double` | `single`
Complex Number Support: Yes

# Version History

**Introduced in R2013b**

## References

[1] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.104. “Evolved Universal Terrestrial Radio Access (E-UTRA); Base Station (BS) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteFadingChannel` | `lteMovingChannel` | `lteOFDMDemodulate` | `lteSCFDMADemodulate`

# lteLayerDemap

Layer demapping onto scrambled and modulated codewords

## Syntax

```
out = lteLayerDemap(in,ncw)
out = lteLayerDemap(in,ncw,txscheme)
out = lteLayerDemap(chs,in)
```

## Description

`out = lteLayerDemap(in,ncw)` performs the layer demapping required to undo the processing described in TS 36.211, Sections 5.3.2A and 6.3.3 [1]. The function returns `out`, a cell array containing one, or two vectors of modulation symbols, one for each codeword. The function demaps the `NU` layers specified in the input matrix, `in`, into `ncw` codewords using `'Port0'` transmission scheme if `NU = 1` and `'SpatialMux'` transmission scheme otherwise.

`out = lteLayerDemap(in,ncw,txscheme)` performs the layer demapping using the transmission scheme, `txscheme`.

`out = lteLayerDemap(chs,in)` performs layer demapping according to the parameters specified in the channel transmission configuration structure, `chs`.

## Examples

### Demap Codeword for Transmit Diversity

Map a codeword onto four symbols for 'TxDiversity' transmission scheme. Recover the codeword by demapping the four layers onto one codeword.

```
nCodewords = 1;
codeword = ones(16,1);
nLayers = 4;
txScheme = 'TxDiversity';

layerMap = lteLayerMap(codeword,nLayers,txScheme);

out = lteLayerDemap(layerMap,nCodewords,txScheme);
```

## Input Arguments

### `in` — Modulation symbols

numeric matrix

Modulation symbols, specified as an *M*-by-*NU* numeric matrix consisting of *M* modulation symbols for *NU* transmission layers. You can generate this matrix using `lteDLDeprecode` or `ltePUSCHDeprecode`.

Data Types: `double`

Complex Number Support: Yes

**`ncw` — Number of codewords**
1 | 2

Number of codewords, specified as 1 or 2.

Data Types: `double`

**`txscheme` — Transmission scheme**
`'Port0'` | `'TxDiversity'` | `'CDD'` | `'SpatialMux'` | `'MultiUser'` | `'Port5'` | `'Port7-8'` | `'Port8'` | `'Port7-14'`

PDSCH transmission scheme, specified as one of the following options.

| Transmission scheme | Description |
|---|---|
| `'Port0'` | Single antenna port, port 0 |
| `'TxDiversity'` | Transmit diversity |
| `'CDD'` | Large delay cyclic delay diversity scheme |
| `'SpatialMux'` | Closed loop spatial multiplexing |
| `'MultiUser'` | Multi-user MIMO |
| `'Port5'` | Single-antenna port, port 5 |
| `'Port7-8'` | Single-antenna port, port 7, when `NLayers` = 1. Dual layer transmission, ports 7 and 8, when `NLayers` = 2. |
| `'Port8'` | Single-antenna port, port 8 |
| `'Port7-14'` | Up to eight layer transmission, ports 7–14 |

When provided as an optional input this setting overrides any setting provided in `chs.TxScheme`.

Data Types: `char` | `string`

**`chs` — Channel-specific transmission configuration**
structure

Channel-specific transmission configuration, specified as a structure that can contain the following parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`TxScheme`** | Optional | `'Port0'`, `'TxDiversity'`, `'CDD'`, `'SpatialM` `'MultiUs` `'Port5'`, `'Port7-8` `'Port8'`, `'Port7-1` <br><br> The default `TxScheme` `'Port0'` f `NLayers` = `'SpatialM` otherwise. | PDSCH transmission scheme, specified as one of the following options. |

| Transmission scheme | Description |
|---|---|
| `'Port0'` | Single antenna port, port 0 |
| `'TxDiversity'` | Transmit diversity |
| `'CDD'` | Large delay cyclic delay diversity scheme |
| `'SpatialMux'` | Closed loop spatial multiplexing |
| `'MultiUser'` | Multi-user MIMO |
| `'Port5'` | Single-antenna port, port 5 |
| `'Port7-8'` | Single-antenna port, port 7, when `NLayers` = 1. Dual layer transmission, ports 7 and 8, when `NLayers` = 2. |
| `'Port8'` | Single-antenna port, port 8 |
| `'Port7-14'` | Up to eight layer transmission, ports 7–14 |

| | | | |
|---|---|---|---|
| Also specify one of these fields: | | | |
| **`Modulation`** | Required, if `NCodewords` is not set | `'QPSK'`, `'16QAM'`, `'64QAM'`, `'256QAM'`, `'1024QAM'` | Modulation type, specified as a character vector, cell array of character vectors, or string array. If blocks, each cell is associated with a transport block. |
| **`NCodewords`** | Required, if `Modulation` is not set | 1, 2 | Number of codewords |

**1** The number of codewords is established from the number of modulation formats in the `Modulation` field. This allows the correct number of codewords to be returned by using the channel transmission configuration structure `chs` as provided to the `ltePDSCH` or `ltePUSCH` function on the transmit side. Alternatively the number of codewords can be directly specified in the `NCodewords` field. The `NCodewords` field takes precedence if present.

## Output Arguments

### `out` — Modulation symbols

cell array of one or two vectors

Modulation symbols, specified as a cell array of one or two vectors. The cell array contains one or two vectors of symbols, one for each codeword.

Data Types: `cell`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteLayerMap` | `lteDLDeprecode` | `lteULDeprecode` | `ltePUSCHDeprecode` | `ltePHICHDeprecode` | `lteSymbolDemodulate`

# lteLayerMap

Layer mapping of modulated and scrambled codewords

## Syntax

```
out = lteLayerMap(in,nu)
out = lteLayerMap(in,nu,txscheme)
out = lteLayerMap(chs,in)
```

## Description

`out = lteLayerMap(in,nu)` performs layer mapping of the codeword or codewords, `in`, onto `nu` layers. It carries out the layer mapping according to TS 36.211 [1], Sections 5.3.2A and 6.3.3. The function returns an *M*-by-`nu` matrix consisting of the modulation symbols for transmission upon `nu` layers. These transmission layers are formed by multiplexing the modulation symbols from either one or two codewords. The overall operation of the layer mapper is the transpose of that defined in the specification. In other words, the symbols for layers lie in columns rather than rows.

`out = lteLayerMap(in,nu,txscheme)` performs layer mapping using the transmission scheme, `txscheme`.

`out = lteLayerMap(chs,in)` performs layer mapping of the codeword or codewords, `in`, according to the parameters in the channel transmission configuration structure, `chs`.

## Examples

### Map Codeword for Spatial Multiplexing

Map one codeword to four layers for the spatial multiplexing transmission scheme.

When no transmission scheme is specified, the default layer mapping is spatial multiplexing.

```
out = lteLayerMap(ones(40,1),4);
sizeOut = size(out)
```

```
sizeOut = 1×2

    10     4
```

### Map Codeword for Transmit Diversity

Map one codeword to four layers for the transmit diversity transmission scheme.

```
out = lteLayerMap(ones(40,1),4,'TxDiversity');
sizeOut = size(out)
```

```
sizeOut = 1×2
```

```
    10     4
```

## Input Arguments

### `in` — Scrambled and modulated codeword or codewords

numeric vector | cell array of numeric vectors

Scrambled and modulated codeword or codewords, specified as a numeric vector or a cell array of numeric vectors. As a cell array, `in` contains one or two vectors of modulation symbols that result from the scrambling and modulation of DL-SCH or UL-SCH codewords.

### `nu` — Number of transmission layers

integer from 1 to 8

Number of transmission layers, specified as a scalar integer from 1 to 8.

Data Types: `double`

### `txscheme` — Transmission scheme

`'Port0'` | `'TxDiversity'` | `'CDD'` | `'SpatialMux'` | `'MultiUser'` | `'Port5'` | `'Port7-8'` | `'Port8'` | `'Port7-14'` | optional

PDSCH transmission scheme, specified as one of the following options.

| Transmission scheme | Description |
|---|---|
| `'Port0'` | Single antenna port, port 0 |
| `'TxDiversity'` | Transmit diversity |
| `'CDD'` | Large delay cyclic delay diversity scheme |
| `'SpatialMux'` | Closed loop spatial multiplexing |
| `'MultiUser'` | Multi-user MIMO |
| `'Port5'` | Single-antenna port, port 5 |
| `'Port7-8'` | Single-antenna port, port 7, when `NLayers` = 1. Dual layer transmission, ports 7 and 8, when `NLayers` = 2. |
| `'Port8'` | Single-antenna port, port 8 |
| `'Port7-14'` | Up to eight layer transmission, ports 7–14 |

This optional input takes precedence over `chs.TxScheme`.

Data Types: `char` | `string`

### `chs` — Channel-specific transmission configuration

structure

Channel-specific transmission configuration, specified as a structure that can contain the following parameter fields.

<table>
<tr><th>Parameter Field</th><th>Required or Optional</th><th>Values</th><th>Description</th></tr>
<tr><td>NLayers</td><td>Required</td><td>Integer from 1 to 8</td><td>Number of transmission layers.</td></tr>
<tr><td>TxScheme</td><td>Optional</td><td>'Port0', 'TxDiversity', 'CDD', 'SpatialMux', 'MultiUser', 'P<br>'Port7-8', 'Por<br>'Port7-14'.<br><br>The default TxSc<br>'Port0' for NLay<br>1, and 'SpatialM<br>otherwise.</td><td>PDSCH transmission scheme, specified as one of the following options.
<table>
<tr><th>Transmission scheme</th><th>Description</th></tr>
<tr><td>'Port0'</td><td>Single antenna port, port 0</td></tr>
<tr><td>'TxDiversity'</td><td>Transmit diversity</td></tr>
<tr><td>'CDD'</td><td>Large delay cyclic delay diversity scheme</td></tr>
<tr><td>'SpatialMux'</td><td>Closed loop spatial multiplexing</td></tr>
<tr><td>'MultiUser'</td><td>Multi-user MIMO</td></tr>
<tr><td>'Port5'</td><td>Single-antenna port, port 5</td></tr>
<tr><td>'Port7-8'</td><td>Single-antenna port, port 7, when NLayers = 1. Dual layer transmission, ports 7 and 8, when NLayers = 2.</td></tr>
<tr><td>'Port8'</td><td>Single-antenna port, port 8</td></tr>
<tr><td>'Port7-14'</td><td>Up to eight layer transmission, ports 7–14</td></tr>
</table></td></tr>
</table>

## Output Arguments

### out — Modulation symbols

numeric matrix

Modulation symbols, returned as a numeric matrix. out is an *M*-by-nu matrix consisting of *M* modulation symbols for transmission upon nu layers.

Data Types: double

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

lteLayerDemap | lteSymbolModulate | lteDLPrecode | lteULPrecode | ltePUSCHPrecode | ltePHICHPrecode

# lteMCS

Modulation and coding scheme lookup

## Syntax

```
[itbs,mod,rv]=lteMCS()
[itbs,mod,rv]=lteMCS(imcs)

[itbs,mod,rv]= lteMCS(table)
[itbs,mod,rv]=lteMCS(imcs,table)
```

## Description

Use `lteMCS` to look up the modulation and coding scheme (MCS) information as defined by MCS index mapping to modulation and TBS index tables in TS 36.213 [1] Table 7.1.7.1-1, Table 7.1.7.1-1A, and Table 8.6.1-1.

`[itbs,mod,rv]=lteMCS()` returns the physical downlink shared channel (PDSCH) MCS information for all MCS index values specified in Table 7.1.7.1-1 of [1].

The function returns the columns of the indexed MCS table entries as separate outputs. The `itbs` output is a vector of the corresponding transport block size (TBS) indices. The `mod` output is a vector of the corresponding modulation schemes. The `rv` output is a vector of the corresponding redundancy version (RV) indices.

The function returns reserved values of `itbs` as `NaN` and reserved values of `mod` as an empty character vector. For the PDSCH, the RV is not defined, so the function returns the `rv` output as a vector of zeros when you use this syntax.

`[itbs,mod,rv]=lteMCS(imcs)` returns PDSCH MCS information for one or more MCS index values, `imcs`, specified in Table 7.1.7.1-1 of [1].

For the PDSCH, the RV is not defined, so the function returns the `rv` output as a vector of zeros when you use this syntax.

`[itbs,mod,rv]= lteMCS(table)` returns PDSCH or physical uplink shared channel (PUSCH) MCS information associated with one or more rows of `table`, the specified table of [1].

`[itbs,mod,rv]=lteMCS(imcs,table)` returns PDSCH or PUSCH MCS information associated with one or more rows of the specified table for one or more MCS indices.

## Examples

### Return TBS Index, Modulation Order, and RV Index

Return the TBS index, modulation order, and RV index for MCS index 17.

```
imcs = 17;
[itbs,mod,rv] = lteMCS(imcs)
```

```
itbs = 15

mod =
'64QAM'

rv = 0
```

### Get MCS from PDSCH Table 2

Return the PDSCH transport block size index and modulation scheme for the set of indices `imcs` = 20,...,27 used to configure a first transport block transmission with Release 12 256QAM modulation.

```
[ITBS,Modulation] = lteMCS(20:27,'PDSCHTable2')
```

```
ITBS = 1×8

    25    27    28    29    30    31    32    33


Modulation = 1x8 cell
    {'256QAM'}    {'256QAM'}    {'256QAM'}    {'256QAM'}    {'256QAM'}    {'256QAM'}    {'256QAM
```

## Input Arguments

### `imcs` — MCS indices

vector of integers in the interval [–1, 31] | integer in the interval [–1, 31]

MCS indices, specified as an integer or vector of integers in the interval [–1, 31].

If you specify any element of this input as `-1`, the function interprets the value as a discontinuous transmission. In this case, the function returns the corresponding elements of the `itbs` and `mod` outputs as `-1` and `'QPSK'`, respectively.

If you specify this input as a scalar, the function returns the `mod` as a single character vector instead of a single element cell array of character vectors.

Data Types: `double`

### `table` — MCS index mapping table

`'PDSCH'` | `'PDSCHTable2'` | `'PDSCHTable3` | `'PUSCH'`

MCS index mapping table, specified as a character vector or string scalar, identifying the desired table from [1]:

- `'PDSCH'` indicates PDSCH, Table 7.1.7.1-1
- `'PDSCHTable2'` indicates Table 2 for PDSCH, Table 7.1.7.1-1A, which was added in 3GPP Release 12
- `'PDSCHTable3'` indicates Table 2 for PDSCH, Table 7.1.7.1-1B, which was added in 3GPP Release 15
- `'PUSCH'` indicates PUSCH, Table 8.6.1-1

Data Types: `char` | `string`

## Output Arguments

### `itbs` — Transport block size indices

vector of integers in the interval [–1, 37] | integer in the interval [–1, 37]

Transport block size indices, returned as an integer or vector of integers in the interval [–1, 37].

If you specify any element of the `imcs` input as `-1`, the function interprets the value as a discontinuous transmission. In this case, the function returns the corresponding element of this output as `-1`.

If you specify the `itbs` input as a scalar, the function returns this output as a single integer instead of a vector.

### `mod` — Modulation orders

cell array of character vectors | `'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'` | `'1024QAM'`

Modulation orders, returned as a cell array of character vectors.

If you specify any element of the `imcs` input as `-1`, the function interprets the value as a discontinuous transmission. In this case, the function returns the corresponding element of this output as `'QPSK'`.

If you specify the `itbs` input as a scalar, the function returns this output as a single character vector instead of a single element cell array of character vectors.

### `rv` — Redundancy versions

column vector | `0` | `1` | `2` | `3`

Redundancy versions, returned as an integer or vector of integers in the interval [0, 3]. Each entry of this output corresponds to the values you specify in the `imcs` input in accordance with Table 8.6.1-1 of [1].

# Version History

**Introduced in R2014b**

## References

[1] 3GPP TS 36.213. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteTBS` | `lteDLSCH` | `lteDLSCHDecode` | `lteULSCH` | `lteULSCHDecode`

# lteMIB

MIB encoding and decoding

## Syntax

```
mib = lteMIB(enb)
enb = lteMIB(mib)
enb = lteMIB(mib,enb)
```

## Description

`mib = lteMIB(enb)` allows encoding and decoding of the MIB broadcast control channel (BCCH) message from cell-wide settings.

It creates the 24-bit-long MIB message, `mib`, from the fields of cell-wide settings structure, `enb`. See TS 36.331 [1], Sections 5.2.1.1 and 6.2.2 for further description of the MIB.

`enb = lteMIB(mib)` performs the inverse processing of the preceding syntax, taking as input the MIB message bits, `mib`, and creating the cell-wide settings structure, `enb`.

`enb = lteMIB(mib,enb)` includes in the `enb` output structure any fields contained in the `enb` input structure. For any of the fields already present in the input structure, the value decoded from the MIB replaces the existing value.

---

**Note** Within the MIB, the system frame number (SFN) is stored as `floor(SFN/4)`. Therefore, when `enb` is created from an MIB bit sequence, `enb.NFrame` satisfies `mod(enb.NFrame,4)==0` and the frame number modulo 4 must be established by other means. For example, this can be done by using the `nfmod4` output of `ltePBCHDecode`.

---

## Examples

### Decode MIB Message Bits

Create a column vector of MIB message bits.

```
mib = [0,1,0,0,1,0,1,1,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0].';
```

Decode MIB message bits.

```
enb = lteMIB(mib)
```

```
enb = struct with fields:
            NDLRB: 25
    PHICHDuration: 'Normal'
               Ng: 'One'
           NFrame: 828
```

## Input Arguments

### `enb` — Cell-wide settings

structure

Cell-wide settings, specified as a structure. `enb` can contain the following fields.

#### `NDLRB` — Number of downlink resource blocks

scalar value (6...110)

Number of downlink resource blocks, specified as a positive integer scalar value. `NDLRB` must be between 6 and 110.

**Note** If `NDLRB` is a nonstandard bandwidth, not one of the set {6,15,25,50,75,100}, all ones are inserted into the first 3 bits, the *dl-Bandwidth* bit field, of the MIB message, `mib`.

Data Types: `double`

#### `Ng` — HICH group multiplier

`'Sixth'` (default) | optional | `'Half'` | `'One'` | `'Two'`

HICH group multiplier, specified as `'Sixth'`, `'Half'`, `'One'`, or `'Two'`.

Data Types: `char` | `string`

#### `NFrame` — Frame number

0 (default) | optional | nonnegative scalar integer

Frame number, specified as a nonnegative scalar integer.

Data Types: `double`

#### `PHICHDuration` — PHICH duration

`'Normal'` (default) | optional | `'Extended'`

PHICH duration, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

Data Types: `struct`

### `mib` — MIB message bit sequence

24-bit column vector

MIB message bit sequence, specified as a 24-bit column vector.

**Note** If the first 3 bits, the *dl-Bandwidth* bit field, of the MIB message do not contain the equivalent of a decimal between 0 and 5 (MSB first, corresponding to the RB set {6,15,25,50,75,100}), the returned `NDLRB` is 0.

Data Types: `double` | `int8` | `logical`

## Output Arguments

### `mib` — MIB message

24-bit column vector

MIB message, returned as a 24-bit column vector.

**Note** If the `enb.NDLRB` input parameter field is a nonstandard bandwidth, not one of the set {6,15,25,50,75,100}, the first 3 bits of `mib`, the *dl-Bandwidth* bit field, are all ones.

Data Types: `int8`

### `enb` — Cell-wide settings created from MIB

structure

Cell-wide settings created from MIB, returned as a structure. `enb` contains the following fields.

### `NDLRB` — Number of downlink resource blocks

nonnegative scalar integer

Number of downlink resource blocks, returned as a nonnegative scalar integer.

**Note** If the first 3 bits, the *dl-Bandwidth* bit field, of the input MIB message, `mib`, do not contain the equivalent of a decimal between 0 and 5 (MSB first, corresponding to the RB set {6,15,25,50,75,100}), `NDLRB` is 0. The MIB message should have 24 bits. Longer messages are truncated to 24 elements, while shorter messages are zero padded.

Data Types: `int32`

### `PHICHDuration` — PHICH duration

`'Normal'` | `'Extended'`

PHICH duration, returned as `'Normal'` or `'Extended'`.

Data Types: `char`

### `Ng` — HICH group multiplier

`'Sixth'` | `'Half'` | `'One'` | `'Two'`

HICH group multiplier, specified as `'Sixth'`, `'Half'`, `'One'`, or `'Two'`.

Data Types: `char`

### `NFrame` — Frame number

scalar value

Frame number, specified as a scalar value.

Data Types: `int32`

Data Types: `struct`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.331. “Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Resource Control (RRC); Protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteBCH` | `lteBCHDecode` | `lteSLMIB`

# lteMovingChannel

Moving channel propagation conditions

## Syntax

```
out = lteMovingChannel(model,in)
```

## Description

`out = lteMovingChannel(model,in)` implements the moving propagation conditions specified in TS 36.104 [1]. The filtered waveform is stored in matrix `out`, where each column corresponds to the waveform at each of the receive antennas. The columns of matrix `in` correspond to the channel input waveforms at each transmit antenna. The input waveforms are filtered with the delay profiles as specified in the parameter structure `model`. The delay profiles are resampled to match the input signal sampling rate. The modeling process introduces delay on top of the channel group delay.

The time difference between the first multipath component and the reference time (assumed to be 0) follows a sinusoidal characteristic.

$$\Delta\tau = \frac{A}{2}(1 + \sin(\Delta\omega(t + t_0)))$$

Where the offset $t_0$ is

$$t_0 = InitTime + \frac{3\pi}{2(\Delta\omega)}$$

If `model.InitTime` is 0, the delay of the first multipath component is 0. If $t = 0$, $\Delta\tau = 0$. Relative delay between all multipath components is fixed.

Two moving propagation scenarios are specified in TS 36.104 [1], Annex B.4:

- Scenario 1 implements an extended typical urban with 200 Hz Doppler shift (ETU200) Rayleigh fading model with changing delays. The Rayleigh fading model can be modeled using two different methods as described in `model.ModelType`. For Scenario 1, `model.InitTime` also controls the fading process timing offset. Changing this value produces parts of the fading process at different points in time.
- Scenario 2 consists of a single non-fading path with unit amplitude and zero phase degrees with changing delay. No AWGN is introduced internally in this model.

## Examples

### Model Moving Propagation Channel

Generate a frame and filter it with the LTE moving propagation channel.

```
rmc = lteRMCDL('R.10');
[txWaveform,txGrid,info] = lteRMCDLTool(rmc,[1;0;1]);
chcfg.Seed = 1;
```

```
chcfg.NRxAnts = 1;
chcfg.MovingScenario = 'Scenario1';
chcfg.SamplingRate = 100000;
chcfg.InitTime = 0;
rxWaveform = lteMovingChannel(chcfg,txWaveform);
```

## Input Arguments

### `model` — Moving channel model

structure

Moving channel model, specified as a structure containing these fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **Seed** | Required | Scalar value | Random number generator seed. To use a random seed, set `Seed` to zero.<br><br>**Note**<br>• To produce distinct results, use `Seed` values in the range<br>$[0, 2^{31} - 1 - K(K-1)/2]$,<br>Where $K = P \times$ `model.NRxAnts`, the product of the number of transmit and receive antennas. `Seed` values outside of this recommended range should be avoided as they may result in random sequences that repeat results produced using Seed values inside the recommended range.<br>• The moving channel random seed behavior is not affected by the state of MATLAB random number generators, `rng`. |
| **NRxAnts** | Required | Positive scalar integer | Number of receive antennas |
| **MovingScenario** | Required | `'Scenario1'`, `'Scenario2'` | Moving channel scenario |
| **SamplingRate** | Required | Numeric scalar | Input signal sampling rate, the rate of each sample in the rows of the input matrix, `in`. |
| **InitTime** | Required | Scalar value | Fading process and timing adjustment offset, in seconds |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NormalizeTxAnts** | Optional | `'On'` (default), `'Off'` | Transmit antenna number normalization, specified as:<br>• `'On'`, `lteFadingChannel` normalizes the model output by `1/sqrt(P)`, where *P* is the number of transmit antennas. Normalization by the number of transmit antennas ensures that the output power per receive antenna is unaffected by the number of transmit antennas.<br>• `'Off'`, normalization is not performed. |
| The following fields are required or optional (as indicated) only if `MovingScenario` is set to `'Scenario1'`. | | | |
| **NTerms** | Optional | 16 (default)<br>scalar power of 2 | Number of oscillators used in fading path modeling. |
| **ModelType** | Optional | `'GMEDS'` (default), `'Dent'` | Rayleigh fading model type.<br>• `'GMEDS'`, the Rayleigh fading is modeled using the Generalized Method of Exact Doppler Spread (GMEDS), as described in [3].<br>• `'Dent'`, the Rayleigh fading is modeled using the modified Jakes fading model described in [2]<br>**Note** `ModelType = 'Dent'` is not recommended. Use `ModelType = 'GMEDS'` instead. |
| **NormalizePathGains** | Optional | `'On'` (default), `'Off'` | Model output normalization.<br>• `'On'`, the model output is normalized such that the average power is unity.<br>• `'Off'`, the average output power is the sum of the powers of the taps of the delay profile. |

Data Types: `struct`

### `in` — Input samples

numeric matrix

Input samples, specified as a numeric matrix. `in` has size *T*-by-*P*, where *P* is the number of transmit antennas and *T* is the number of time-domain samples. These waveforms are filtered with the delay profiles as specified in the parameter structure `model`. These delay profiles are resampled to match

the input signal sampling rate. Each column of `in` corresponds to the waveform at each of the transmit antennas.

Data Types: `double` | `single`
Complex Number Support: Yes

## Output Arguments

### `out` — Filtered waveform

numeric matrix

Filtered waveform, returned as a numeric matrix. Each column of `out` corresponds to the waveform at each of the receive antennas.

Data Types: `double` | `single`
Complex Number Support: Yes

# Version History

**Introduced in R2013b**

## References

[1] 3GPP TS 36.104. “Evolved Universal Terrestrial Radio Access (E-UTRA); Base Station (BS) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] Dent, P., G. E. Bottomley, and T. Croft. “Jakes Fading Model Revisited.” *Electronics Letters*. Vol. 29, 1993, Number 13, pp. 1162–1163.

[3] Pätzold, Matthias, Cheng-Xiang Wang, and Bjørn Olav Hogstad. “Two New Sum-of-Sinusoids-Based Methods for the Efficient Generation of Multiple Uncorrelated Rayleigh Fading Waveforms.” *IEEE Transactions on Wireless Communications*. Vol. 8, 2009, Number 6, pp. 3122–3131.

## See Also

`lteFadingChannel` | `lteHSTChannel` | `lteOFDMModulate` | `lteSCFDMAModulate`

# lteNBDLFrameOffset

Estimate timing offset of first downlink frame

## Syntax

```
offset = lteNBDLFrameOffset(enb,waveform)
[offset,corr] = lteNBDLFrameOffset(enb,waveform)
[ ___ ] = lteNBDLFrameOffset(enb,waveform,cfgCorr)
```

## Description

`offset = lteNBDLFrameOffset(enb,waveform)` returns `offset`, the timing offset between the start of `waveform`, the input time-domain waveform, and the start of the first downlink frame. To measure `offset`, the function performs synchronization using the specified synchronization signals of `waveform` for specified cell-wide settings `enb`.

This function estimates the timing offset by performing these steps.

1. Extract the timing of the peak correlation between `waveform` and internally generated reference waveforms containing the synchronization signal symbols.
2. Calculate the correlation for each antenna.
3. Compute the offset for the correlation that displays the earliest peak with a magnitude of at least 50% of the maximum correlation across all antennas

`[offset,corr] = lteNBDLFrameOffset(enb,waveform)` also returns `corr`, the correlation matrix that the function uses to estimate the timing offset.

`[ ___ ] = lteNBDLFrameOffset(enb,waveform,cfgCorr)` specifies `cfgCorr`, reference signal configuration options, in addition to arguments from any of the previous syntaxes. This input sets the reference signals that the function uses to estimate the timing offset.

## Examples

### Estimate Offset of Narrowband Waveform

Create a subframe resource array for the cell-wide settings structure, `enb`. Map the subframe array into a frame resource array.

```
enb.OperationMode = 'Standalone';
enb.NSubframe = 5;
enb.NFrame = 2;
enb.NNCellID = 1;
ue = struct('NBULSubcarrierSpacing','15kHz');
subframeGrid = repmat(lteNBResourceGrid(ue),1,2);
frameGrid = repmat(subframeGrid,1,10);
subframeGrid(lteNPSSIndices(enb)) = lteNPSS(enb);
frameGrid(:,14*enb.NSubframe + (1:14)) = subframeGrid;
```

Generate an OFDM-modulated waveform for the resource array and user-equipment-specific settings ue, specifying a timing offset of 25 samples.

```
waveform = [zeros(25,1); lteSCFDMAModulate(ue,frameGrid)];
```

Estimate the timing offset and display the result.

```
offset = lteNBDLFrameOffset(enb,waveform)
```

```
offset = 25
```

### Estimate Offset of Narrowband Waveform with NPSS

Create a subframe resource grid for the cell-wide settings structure, enb. Map the subframe grid into a frame resource grid.

```
subframeGrid = zeros(12,14);
frameGrid = zeros(12,14*10);
enb.OperationMode = 'Standalone';
enb.NSubframe = 5;
subframeGrid(lteNPSSIndices(enb)) = lteNPSS(enb);
frameGrid(:,14*enb.NSubframe + (1:14)) = subframeGrid;
```

Generate an SC-FDMA-modulated waveform for the resource grid and user-equipment-specific settings ue, specifying a timing offset of five samples.

```
ue.NBULSubcarrierSpacing = '15kHz'; % NB DL OFDM is the same as NB UL SC-FDMA
                                    % with 15-kHz subcarrier spacing
modulatedWaveform = lteSCFDMAModulate(ue,frameGrid);
waveform = [zeros(5,1); modulatedWaveform];
```

Use the NPSS to estimate the timing offset of the waveform, and return the signal used to perform the estimate. Display the timing offset estimate.

```
cfgCorr.NPSS = 'On';
cfgCorr.NSSS = 'Off';
cfgCorr.NRS = 'Off';
[offset,corr] = lteNBDLFrameOffset(enb,waveform,cfgCorr);
disp(offset)
```

```
     5
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure containing these fields.

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **OperationMode** | Optional | `'Standalone'` (default), `'Inband-SamePCI'`, `'Inband-DifferentPCI'`, `'Guardband'` | NB-IoT operation mode, specified as one of these values:<br>• `'Standalone'` – NB-IoT standalone operation within any 180-kHz band outside any LTE carrier bandwidth<br>• `'Inband-SamePCI'` – NB-IoT in-band operation with the same physical layer cell identity (PCI) as an LTE carrier<br>• `'Inband-DifferentPCI'` – NB-IoT in-band operation with a different PCI to an LTE carrier<br>• `'Guardband'` – NB-IoT guard-band operation utilizing unused resource blocks within the guard-band | Not applicable | `char`, `string` |

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| | | | of an LTE carrier | | |
| **NCellID** | Required when you set the `OperationMode` field to `'Inband-SamePCI'` or `'Inband-DifferentPCI'` | Integer in the interval [0, 503] | Physical layer cell identity (PCI). | To enable this field, set the `OperationMode` field to `'Inband-SamePCI'` or `'Inband-DifferentPCI'` | `double` |
| **CellRefP** | Required when you set the `OperationMode` field to `'Inband-SamePCI'` or `'Inband-DifferentPCI'` | 1, 2, 4 | Number of cell-specific antenna ports | To enable this field, set the `OperationMode` field to `'Inband-SamePCI'` or `'Inband-DifferentPCI'` | `double` |
| **NNCellID** | Required when you set the `NSSS` or `NRS` field of the `cfgCorr` input to `'On'` | Integer in the interval [0, 503] | Narrowband PCI | To enable this field, set the `NSSS` or `NRS` field of the `cfgCorr` input to `'On'` | `double` |
| **NBRefP** | Required when you set the `NSSS` or `NRS` field of the `cfgCorr` input to `'On'` | 1, 2 | Number of narrowband reference signal (NRS) antenna ports | To enable this field, set the `NRS` field of the `cfgCorr` input to `'On'` | `double` |

**Note** To exclude cell reference signal (RS) locations, specify the `NCellID` and `CellRefP` fields. If you do not specify the `NCellID` and `CellRefP` fields, the function assumes that the cell RS is absent.

Data Types: `struct`

**`waveform` — Time-domain waveform**
complex-valued matrix

Time-domain waveform, specified as a *T*-by-*P* complex-valued matrix, where:

- *T* is the number of time-domain samples.
- *P* is the number of receive antennas.

You can generate a time-domain waveform by performing OFDM modulation on a signal or by using one of these channel model functions: `lteFadingChannel`, `lteHSTChannel`, or `lteMovingChannel`.

Data Types: `double`
Complex Number Support: Yes

#### `cfgCorr` — Reference signal configuration options
structure

Reference signal configuration options, specified as a structure containing these fields.

#### `NPSS` — NPSS correlation mode indicator
`'On'` (default) | `'Off'`

Narrowband primary synchronization signal (NPSS) correlation mode indicator, specified as `'On'` or `'Off'`. To use the NPSSs for estimating the timing offset, specify `'On'`. To disable the use of the NPSSs for estimating the timing offset, specify `'Off'`.

Data Types: `char` | `string`

#### `NSSS` — NSSS correlation mode indicator
`'On'` (default) | `'Off'`

Narrowband secondary synchronization signal (NSSS) correlation mode indicator, specified as `'On'` or `'Off'`. To use the NSSSs for estimating the timing offset, specify `'On'`. To disable the use of the NSSSs for estimating the timing offset, specify `'Off'`.

Data Types: `char` | `string`

#### `NRS` — NRS correlation mode indicator
`'Off'` (default) | `'On'`

Narrowband reference signal (NRS) correlation mode indicator, specified as `'On'` or `'Off'`. To use the NRSs for estimating the timing offset, specify `'On'`. To disable the use of the NRSs for estimating the timing offset, specify `'Off'`.

Data Types: `char` | `string`

Data Types: `struct`

## Output Arguments

#### `offset` — Timing offset
integer

Timing offset, in samples, between the start of the `waveform` input and the start of the first downlink frame within `waveform`, returned as an integer.

Data Types: `double`

#### `corr` — Correlation matrix used to estimate timing offset
complex-valued matrix

Correlation matrix that the function uses to estimate timing offset, returned as a complex-valued matrix of the same dimensions as the `waveform` input.

Data Types: `double`
Complex Number Support: Yes

## Version History

**Introduced in R2019b**

## References

[1] 3GPP TS 36.104. “Evolved Universal Terrestrial Radio Access (E-UTRA); Base Station (BS) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

**Functions**

`lteDLFrameOffset` | `lteFadingChannel` | `lteHSTChannel` | `lteMovingChannel` | `lteNPSS` | `lteNRS` | `lteNSSS` | `lteSCFDMAModulate`

# lteNBResourceGrid

Narrowband resource array

## Syntax

```
grid = lteNBResourceGrid(cfg)
```

## Description

`grid = lteNBResourceGrid(cfg)` generates an empty resource array for the specified configuration settings. If you specify an uplink transmission, the elements of the `grid` output represent elements for one slot of a narrowband internet of things (NB-IoT) resource array, as described in section 10.1.2.1 of [1]. If you specify a downlink transmission, the elements of the `grid` output represent elements for one subframe of an NB-IoT resource array, as described in section 10.2.2.1 of [1].

For more information about resource grids and LTE Toolbox resource element representation by using multidimensional arrays, see “Represent Resource Grids”.

## Examples

### Create Empty Narrowband Slot Resource Array for Uplink Transmission

Create a resource array for an uplink NB-IoT transmission with a subcarrier spacing of 3.75 kHz and a single antenna port.

Configure a transmission with a subcarrier spacing of 3.75 kHz.

```
cfg = struct('NBULSubcarrierSpacing','3.75kHz');
```

Create the resource array and display its size.

```
grid = lteNBResourceGrid(cfg);
disp(size(grid))
```

```
    48     7
```

### Create Empty Narrowband Resource Array for Downlink Transmission

Specify a downlink NB-IoT configuration for one subframe and two antenna ports.

```
cfg = struct('NBRefP',2);
```

Create the resource array and display its size.

```
grid = lteNBResourceGrid(cfg);
disp(size(grid))
```

```
    12    14     2
```

## Input Arguments

### `cfg` — Configuration settings

structure

Configuration settings, specified as a structure whose fields depend on the desired configuration.

To generate a resource array for an uplink configuration, the input structure must contain this field.

| Field | Values | Description | Data Type |
|---|---|---|---|
| **`NBULSubcarrierSpacing`** | `'3.75kHz'`, `'15kHz'` | NB-IoT uplink subcarrier spacing. To set a subcarrier spacing of 3.75 kHz, specify this field as `'3.75kHz'`. To set a subcarrier spacing of 15 kHz, specify this field as `'15kHz'`. | `char`, `string` |

To generate a resource array for a downlink configuration, the input structure must contain this field.

| Field | Values | Description | Data Type |
|---|---|---|---|
| **`NBRefP`** | 1, 2 | Number of narrowband reference signal (NRS) antenna ports. | `double` |

If you specify both fields, the function ignores the value of the `NBRefP` field and generates a resource array for the uplink configuration corresponding to the value of the `NBULSubcarrierSpacing` field.

Data Types: `struct`

## Output Arguments

### `grid` — Empty resource array

*N*-by-*M*-by-*P* array of zeros

Empty resource array, returned as an *N*-by-*M*-by-*P* array of zeros, where:

- *N* is the number of subcarriers
- For an uplink transmission, *M* is the number of single-carrier frequency-division multiple access (SC-FDMA) symbols in a slot. For a downlink transmission, *M* is the number of orthogonal frequency-division multiplexing (OFDM) or SC-FDMA symbols in a subframe.
- *P* is the number of transmit antenna ports. For an uplink transmission, *P* is 1. For a downlink transmission, *P* is the value of `cfg.NBRefP`.

If you specify an uplink configuration, the elements of this output represent elements for the slot resource array described in section 10.1.2.1 of [1]. If you specify a downlink configuration, the elements of this output represent elements of the subframe resource array described in section 10.2.2.1 of [1].

# Version History

**Introduced in R2021a**

## References

[1] 3GPP TS 36.211. “Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. https://www.3gpp.org.

## See Also

**Functions**

`lteResourceGrid` | `lteSCFDMAModulate`

**Topics**

“Represent Resource Grids”

# lteNDLSCH

Generate NB-IoT DL-SCH codeword

## Syntax

```
cwout = lteNDLSCH(outlen,trblkin)
```

## Description

`cwout = lteNDLSCH(outlen,trblkin)` applies the complete NB-IoT downlink shared channel (DL-SCH) transport channel coding chain to the input data, `trblkin`, and returns the codeword in `cwout`. The encoding process includes type-24A CRC calculation, convolutional encoding, and rate matching. This function applies to a single transport block.

## Examples

### Generate NB-IoT DL-SCH Codeword Bits

Generate 960 NB-IoT DL-SCH codeword bits with a transport block.

Set the transport block length to 208 and the output codeword length to 960. Generate the transport block information bits as a random binary sequence.

```
trblklen = 208;
outlen = 960;
trblkin = randi([0 1],trblklen,1);
```

Generate the 960 NB-IoT DL-SCH codeword bits given the transport block information bits and the output codeword length.

```
cw = lteNDLSCH(outlen,trblkin);
```

## Input Arguments

### `outlen` — Codeword length

nonnegative integer

Codeword length, specified as a nonnegative integer. This input represents the NPDSCH capacity for the associated codeword and the lengths of the vector in the `cwout` output. The input transport blocks are rate-matched to the codeword length.

Data Types: `int8` | `double`

### `trblkin` — Transport block information bits to be encoded

numeric vector

Transport block information bits to be encoded, specified as a numeric vector.

Data Types: `int8` | `double`

## Output Arguments

**`cwout` — DL-SCH encoded codeword**
numeric column vector

DL-SCH encoded codewords, returned as a numeric column vector of size `outlen`.

Data Types: `int8`

# Version History

**Introduced in R2018a**

## See Also

`lteNDLSCHDecode` | `lteDLSCH`

# lteNDLSCHDecode

Decode NB-IoT DL-SCH codeword

## Syntax

```
[trblkout,blkcrc,stateout] = lteNDLSCHDecode(trblklen,cwin)
[trblkout,blkcrc,stateout] = lteNDLSCHDecode(trblklen,cwin,statein)
```

## Description

`[trblkout,blkcrc,stateout] = lteNDLSCHDecode(trblklen,cwin)` returns the information bits, `trblkout`, decoded from the input soft LLR codeword data, `cwin`. The NB-IoT downlink shared channel (DL-SCH) decoder includes rate recovery, Viterbi decoding, and CRC calculations. The function also returns the type-24A transport block CRC decoding result in `blkcrc` and the HARQ process decoding state in `stateout`.

`[trblkout,blkcrc,stateout] = lteNDLSCHDecode(trblklen,cwin,statein)` specifies the initial HARQ process state in the `statein` structure. The initial transmission and the re-transmission are both bundles containing multiple subframes, defined in Section 5.3.2.1 of [1].

## Examples

### Generate and Decode NB-IoT DL-SCH Transmissions

This example shows how to transmit a bundle carrying the same transport block twice. The LLR soft bits from repeated subframes in a bundle are combined in structure `dstate`, the LLR soft bits from two bundles are combined in structure `state`. Note that `dstate` is reset before the re-transmission of the bundle.

Specify the cell-wide settings and channel transmission configuration in parameter structures `enb` and `chs`.

```
enb.NNCellID = 0;
enb.NBRefP = 1;
enb.NFrame = 1;
chs.NSF = 3;
chs.NRep = 4;
chs.RNTI = 0;
chs.NPDSCHDataType = 'NotBCCH';
```

Set the transport block length to 208 and the output codeword length to 960. Initialize the decoder states for the first HARQ transmission. Generate transport block data.

```
trblklen = 208;
outlen = 960;
estate = [];
dstate = [];
state = [];
trblkin = randi([0 1],trblklen,1);
```

Generate the 960 NB-Iot DL-SCH codeword bits.

```
cw = lteNDLSCH(outlen,trblkin);
```

Perform the initial transmission of a bundle containing the transport block. Verify if there are errors in the transmission.

```
for subframeIdx = 0:(chs.NSF*chs.NRep-1)
    enb.NSubframe = subframeIdx;
    [sym,estate] = lteNPDSCH(enb,chs,cw,estate);
    [rxcw, dstate] = lteNPDSCHDecode(enb,chs,sym,dstate);
end
[trblkout1,blkerr1,state] = lteNDLSCHDecode(trblklen,rxcw,state);
blkerr1
```

```
blkerr1 = logical
   0
```

Perform the re-transmission of the bundle containing the same transport block. The information obtained from the initial bundle transmission is saved in state and used as an input to lteNDLSCHDecode. Verify if there are errors in the re-transmission.

```
for subframeIdx = chs.NSF*chs.NRep:(2*chs.NSF*chs.NRep-1)
    enb.NSubframe = subframeIdx;
    [sym,estate] = lteNPDSCH(enb,chs,cw,estate);
    [rxcw,dstate] = lteNPDSCHDecode(enb,chs,sym,dstate);
end
[trblkout2,blkerr2,state] = lteNDLSCHDecode(trblklen,rxcw,state);
blkerr2
```

```
blkerr2 = logical
   0
```

## Input Arguments

### `trblklen` — Transport block length

nonnegative integer

Transport block length, specified as a nonnegative integer. `trblklen` defines the transport block lengths to which the input code blocks should be rate-recovered and decoded.

Data Types: `double`

### `cwin` — Soft LLR codeword data

numeric vector

Soft LLR codeword data, specified as a numeric vector. `cwin` contains the floating-point soft LLR data of the codeword to be decoded.

Data Types: `double`

### `statein` — Initial HARQ process state

structure

Initial HARQ process state, specified as a structure. The structure can be empty or contain this field:

| Name | Values | Description | Data Types |
|---|---|---|---|
| **CBSBuffers** | Cell array of vectors | LLR soft buffer state associated with a single transport block. The buffer is positioned at the input to the Viterbi decoder after explicit rate recovery. | `cell` |

The updated buffer states after decoding are returned in the `CBSBuffers` field of the `stateout` output. The `statein` array is normally generated and recycled from the `stateout` of previous calls to `lteNDLSCHDecode` as part of a sequence of HARQ transmissions.

Data Types: `struct`

## Output Arguments

### `trblkout` — Decoded information bits

numeric vector

Decoded information bits, returned as a numeric vector.

Data Types: `int8`

### `blkcrc` — Type-24A transport block CRC decoding result

`true` or `1` | `false` or `0`

Type-24A transport block CRC decoding result, returned as a numeric or logical `1` (`true`) or `0` (`false`).

Data Types: `logical`

### `stateout` — HARQ process decoding state

structure

HARQ process decoding state, returned as a structure containing the internal state of a transport block in these fields:

| Parameter Field | Values | Description | Data types |
|---|---|---|---|
| **CBSBuffers** | Cell array of vectors | LLR soft buffer states for the set of code blocks associated with a single transport block. The buffers are positioned at the input to the Viterbi decoder after explicit rate recovery. | `cell` |
| **BLKCRC** | `true` or `1`, `false` or `0` | Type-24A transport block CRC decoding error | `logical` |

Data Types: `struct`

# Version History

**Introduced in R2018a**

## References

[1] 3GPP TS 36.321. “Evolved Universal Terrestrial Radio Access (E-UTRA); Medium Access Control (MAC) protocol Specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteNDLSCH` | `lteDLSCHDecode`

# lteNPBCH

Generate NPBCH symbols

## Syntax

```
sym = lteNPBCH(enb,cw)
```

## Description

`sym = lteNPBCH(enb,cw)` generates `sym`, a matrix containing the narrowband physical broadcast channel (NPBCH) symbols for cell-wide settings `enb`. The function generates the symbols by applying NPBCH encoding to and codeword `cw`, which comprises scrambling, QPSK modulation, layer mapping, and precoding in accordance with section 10.2.4 of [1].

## Examples

### Generate NPBCH Symbols

Generate NPBCH symbols using the MIB.

Generate the MIB and pass it through broadcast channel (BCH) encoding to obtain the 1600-bit codeword.

```
mib = randi([0 1],34,1);
cw = lteBCH(mib,1600,1);
```

Specify the cell-wide settings in the structure `enb`.

```
enb.NNCellID = 0;
enb.NBRefP = 1;
enb.NSubframe = 10;
```

Generate the NPBCH symbols for each of the subframes.

```
sym = lteNPBCH(enb,cw);
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure containing these fields.

| Parameter Field | Required or Optional | Values | Description | Data Types |
|---|---|---|---|---|
| **`NNCellID`** | Required | Nonnegative integer | NB-IoT physical layer cell identity | `double` |
| **`NBRefP`** | Required | `1`, `2` | Number of narrowband reference signal antenna ports | `double` |
| **`NSubframe`** | Required | Nonnegative integer | Subframe number | `double` |
| **`NFrame`** | Optional | `0` (default),<br><br>nonnegative integer | Frame number | `double` |

Data Types: `struct`

**`cw` — Codeword to be modulated**
binary-valued column vector

Codeword to be modulated, specified as a binary-valued column vector of length 1600.

Data Types: `double` | `logical`

## Output Arguments

**`sym` — NPBCH symbols**
100-by-*P* complex matrix

NPBCH symbols, returned as a 100-by-*P* complex valued matrix, where 100 is the number of modulation symbols for one antenna port for one subframe, and *P* is the number of transmission antenna ports.

Data Types: `double`
Complex Number Support: Yes

# Version History

**Introduced in R2019b**

## References

[1] 3GPP TS 36.211. “Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. https://www.3gpp.org.

## See Also

`lteNPBCHIndices` | `lteNPBCHDecode` | `lteBCH`

# lteNPBCHDecode

Decode NPBCH symbols

## Syntax

```
[bits,stateout,symbols,nfmod64,trblk,NBRefP] = lteNPBCHDecode(enb,sym)
[ ___ ] = lteNPBCHDecode(enb,sym,statein)
[ ___ ] = lteNPBCHDecode(enb,sym,hest,noiseest)
[ ___ ] = lteNPBCHDecode(enb,sym,hest,noiseest,statein)
```

## Description

`[bits,stateout,symbols,nfmod64,trblk,NBRefP] = lteNPBCHDecode(enb,sym)` decodes `sym`, the NB-IoT physical broadcast channel (NPBCH) symbols, for cell-wide settings `enb`. The NPBCH decoding inverts the NPBCH encoding process described in section 10.2.4 of [1] and the broadcast channel (BCH) encoding process described in section 5.3.1 of [2]. The function returns `bits`, a codeword of soft bits, the decoder state for reception of a bundle (a full set of repeated transmissions of a single transport block) `stateout`, and the received constellation symbols, `symbols`, by performing the inverse of NPBCH encoding. Frame number modulo 64 `nfmod64`, decoded BCH information bits `trblk`, and the number of narrowband reference signal (NRS) antenna ports `NBRefP`, are returned by performing the inverse of BCH encoding.

`[ ___ ] = lteNPBCHDecode(enb,sym,statein)` decodes the NPBCH symbols for the initial decode state `statein`.

`[ ___ ] = lteNPBCHDecode(enb,sym,hest,noiseest)` decodes the NPBCH symbols for the channel estimate `hest` and noise estimate `noiseest`.

`[ ___ ] = lteNPBCHDecode(enb,sym,hest,noiseest,statein)` decodes the NPBCH symbols for the channel estimate, noise estimate, and the initial decoder state.

## Examples

### Generate and Decode NPBCH Symbols

Generate and decode the NPBCH symbols subframe-by-subframe for a bundle of 64 NPBCH subframes.

Specify the cell-wide settings.

```
enb = struct('NNCellID',0,'NBRefP',1);
```

To obtain the codeword, generate the MIB and pass it through broadcast channel (BCH) encoding.

```
mib = randi([0 1],34,1);
cw = lteBCH(mib,1600,enb.NBRefP);
```

Specify the encoder state as empty.

```
statein = [];
```

Generate the NPBCH symbols for each of the 64 NPBCH subframes and then decode them.

```
for subframeIdx = 0:63
    enb.NSubframe = subframeIdx*10; % As NPBCH is mapped only on 0th subframe of each frame
    sym = lteNPBCH(enb,cw);
    [decoderOut,stateout,symbols,NfMod64,trblk,NBRefP] = ...
        lteNPBCHDecode(enb,sym,statein);
    statein = stateout;
end
```

To check whether the decoding is successful, display the value of `NBRefP`.

```
NBRefP
```

```
NBRefP = uint32
    1
```

### Decode NPBCH Symbols from Generated Waveform

To obtain NPBCH symbols, decode the waveform that is generated using NB-IoT Downlink Waveform Generator.

Specify the NB-IoT eNodeB configuration with 65 frames.

```
ngen = NBIoTDownlinkWaveformGenerator;
ngen.Config.NNCellID = 120;
ngen.Config.NBRefP = 2;
ngen.Config.TotSubframes = 650;
```

Generate the waveform, `eNodeBOutput`.

```
[eNodeBOutput,~,ofdmInfo] = ngen.generateWaveform;
```

Start the decoding process by first initializing the fields of the structure `enb`.

```
enb.NNCellID = 120;
enb.NBRefP = 2;
```

To obtain the resource grid `rxgrid`, perform OFDM demodulation and generate the NPBCH resource element (RE) indices.

The NB-IoT Downlink OFDM is the same as UL-SC-FDMA. Use `lteSCFDMADemodulate` to perform OFDM demodulation.

```
enb.NBULSubcarrierSpacing = '15kHz';
rxgrid = lteSCFDMADemodulate(enb,eNodeBOutput); % NB-IoT Downlink OFDM Demodulation
npbchIndices = lteNPBCHIndices(enb);
```

Extract the REs from the resource grid using the RE indices.

```
npbchRx = lteExtractResources( ...
    npbchIndices, rxgrid(1:12,1:14,:));
```

Specify the encoder state as empty.

```
statein = [];
```

Decode the NPBCH symbols.

```
[decoderOut,stateout,symbols,nfmod64,trblk,NBRefP] = lteNPBCHDecode(enb,npbchRx,statein);
```

To check whether the decoding is successful, display the value of `NBRefP`.

```
NBRefP
```

```
NBRefP = uint32
    2
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure containing these fields.

| Parameter Field | Required or Optional | Values | Description | Data Types |
|---|---|---|---|---|
| **`NNCellID`** | Required | Nonnegative integer | NB-IoT physical layer cell identity | `double` |
| **`NBRefP`** | Optional | `1`, `2` | Number of narrowband reference signal (NRS) antenna ports. The default is to establish `NBRefP` by decoding the input symbols, `sym`. | `double` |
| **`NSubframe`** | Optional | Nonnegative integer | Subframe number | `double` |
| **`NFrame`** | Optional | `0` (default),<br><br>nonnegative integer | Initial frame number | `double` |

Data Types: `struct`

### sym — Modulated NPBCH symbols

complex-valued matrix

Modulated NPBCH symbols, specified as an $N_{RE}$-by-$N_{RxAnts}$ complex-valued matrix, where:

- $N_{RE}$ is a multiple of the number of quadrature phase-shift keying (QPSK) symbols per antenna and per subframe assigned to the NPBCH.
- $N_{RxAnts}$ is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

### statein — Initial encoder state

structure

Initial encoder state for transmission of a bundle, specified as a structure containing the fields listed in the `stateout` output. At the start of the bundle transmission, set `statein` to empty. The `lteNPBCHDecode` function manages the state during subsequent calls for the transmissions of the bundle and resets it automatically at the end of the bundle.

Data Types: `struct`

**`hest` — Channel estimate**
complex-valued 3-D array

Channel estimate, specified as an $N_{RE}$-by-$N_{RxAnts}$-by-$N_{NBRefP}$ complex-valued array, where:

- $N_{RE}$ is a multiple of the number of QPSK symbols per antenna and per subframe.
- $N_{RxAnts}$ is the number of receive antennas.
- $N_{NBRefP}$ is the number of NRS antenna ports you specify in the `NBRefP` field of the `enb` input.

The `lteNPBCHDecode` function assumes that this estimate uses the NRSs.

Data Types: `double`
Complex Number Support: Yes

**`noiseest` — Noise estimate**
numeric scalar

Noise estimate, specified as a numeric scalar. It is an estimate of the noise power spectral density per resource element on the received subframe. This estimate is provided by the `lteDLChannelEstimate` function.

Data Types: `double`

## Output Arguments

**`bits` — Codeword of soft bits**
binary vector

Codeword of soft bits, returned as an *N*-by-1 binary vector, where *N* can be a part of the 1600-bit codeword length in multiples of 200 or the entire codeword, depending upon the input NPBCH symbols, `sym`.

Data Types: `double`

**`stateout` — Output decoder state**
structure

Output decoder state for the next subframe, returned as a structure. This output contains the internal state of each transport block in these fields.

| Name | Values | Description | Data Types |
|---|---|---|---|
| **SubframeIdx** | Integer in the interval [0, 63] | Index of a subframe within a bundle, in zero-based form, returned as an integer in the interval [0,63]. The `lteNPBCHDecode` function returns this field as the `SubframeIdx` field of the `statein` input increased by one. When the input value of `SubframeIdx` in the `statein` input reaches its maximum value, the function returns this field as `0`. If no input exists in `statein`, the default input is `0`. A value of `0` indicates that the transmission has reached the end of a bundle, which the function also indicates by setting the `EndOfTx` field to `true`. | `double` |
| **CWBuffer** | 1600-by-1 numeric vector | Buffer to store the soft-combined log-likelihood ratio (LLR) bits after codeword descrambling, returned as a 1600-by-1 numeric vector. The length of this field is the same as the length of the codeword. At the beginning of the bundle, the `lteNPBCHDecode` function resets this field. | `double` |

| Name | Values | Description | Data Types |
|---|---|---|---|
| **EndOfTx** | Logical `1` (`true`) or `0` (`false`) | End of bundle indicator. The `lteNPBCHDecode` function returns this field as `1` (`true`) when the transmission reaches the end of the bundle. Otherwise, the `lteNPBCHDecode` function returns this field as `0` (`false`). At the beginning of the bundle, the `lteNPBCHDecode` function resets this field. | `logical` |

Data Types: `struct`

**`symbols` — Received constellation symbols**
complex-valued vector

Received constellation symbols, returned as a complex-valued vector.

Data Types: `double`

**`nfmod64` — System frame number modulo 64**
integer in the interval [0, 63]

System frame number modulo 64, `mod(NFrame,64)`, returned as an integer in the interval [0, 63]. `nfmod64` is obtained when determining the scrambling phase of the input NPBCH symbols, `sym`.

Data Types: `double`

**`trblk` — Decoded BCH information bits**
34-by-1 binary column vector

Decoded BCH information bits, returned as a 34-by-1 binary column vector.

Data Types: `int8`

**`NBRefP` — Number of NRS ports**
0 | 1 | 2

Number of NRS ports, returned as 0, 1, or 2 as determined during the BCH decoding. If the value is 0, a CRC error has been detected, and decoding is unsuccessful.

Data Types: `uint32`

## Tips

To use this function for a bundle transmission, follow these steps:

**1** Call the `lteNPBCHDecode` function and specify the initial encoder state using the `statein` input. The `stateout` output represents the decoder state of the first transmission of the bundle.

2. Call the `lteNPBCHDecode` function again and specify the `statein` input as the `stateout` output returned by the previous call to the function.
3. Repeat step 2 until the `lteNPBCHDecode` function returns the `EndOfTx` field of the `stateout` output as `1` (`true`), indicating the end of the bundle. The `lteNPBCHDecode` automatically resets the state at the end of the bundle transmission.

# Version History

**Introduced in R2019b**

## References

[1] 3GPP TS 36.211. “Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. https://www.3gpp.org.

[2] 3GPP TS 36.212. “Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. https://www.3gpp.org.

## See Also

`lteNPBCH` | `lteNPBCHIndices` | `lteBCH` | `ltePBCHDecode`

# lteNPDCCH

Generate NPDCCH symbols

## Syntax

```
[sym,stateout] = lteNPDCCH (enb,chs,cw)
[sym,stateout] = lteNPDCCH (enb,chs,cw,statein)
```

## Description

`[sym,stateout] = lteNPDCCH (enb,chs,cw)` generates `sym`, a matrix containing the NB-IoT physical downlink control channel (NPDCCH) complex symbols in a subframe for cell-wide settings structure, `enb`, channel transmission configuration, `chs`, and codeword, `cw`. The channel encoding process comprises stages of scrambling, QPSK modulation, layer mapping, and precoding in accordance with 3GPP TS 36.211 Section 10.2.5 of [1]. The function also returns `stateout`, a structure containing the encoder state of a bundle (a full set of repeated transmissions of a single downlink control information).

`[sym,stateout] = lteNPDCCH (enb,chs,cw,statein)` returns the NPDCCH symbols and the initial encoder state specified by `statein`.

## Examples

### Generate NPDCCH Symbols

Generate the NPDCCH symbols subframe-by-subframe for a bundle of 10 subframes.

Specify the cell-wide settings and channel transmission configuration in parameter structures `enb` and `chs`.

```
enb.NNCellID = 0;
enb.NBRefP = 1;
chs.NRep = 10;
```

Set the output code length to 320 and generate the codeword bits. Specify the encoder state as empty at the start of the bundle.

```
cwLen = 320;
cw = ones(cwLen,1); % Codeword bits
estate = [];
```

Generate the NPDCCH symbols for each of the 10 subframes.

```
for nsf=0:chs.NRep-1
   enb.NSubframe=nsf;
    [sym,estate]=lteNPDCCH(enb,chs,cw,estate);
end
estate.EndOfTx
```

```
ans = logical
   1
```

Display the first seven NPDCCH encoded symbols.

```
sym(1:7)
```

```
ans = 7×1 complex

   0.7071 - 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 - 0.7071i
   0.7071 - 0.7071i
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure containing these fields.

| Name | Required or Optional | Values | Description | Data Types |
|---|---|---|---|---|
| **NNCellID** | Required | Nonnegative integer | Narrowband physical layer cell identity | `double` |
| **NBRefP** | Required | 1, 2 | Number of narrowband reference signal (NRS) antenna ports. | `double` |
| **NSubframe** | Required | Nonnegative integer | Subframe number | `double` |

Data Types: `struct`

### chs — Channel transmission configuration

structure

Channel transmission configuration, specified as a structure containing this field.

| Name | Required or Optional | Values | Description | Data Types |
|---|---|---|---|---|
| **NRep** | Required | Nonnegative integer | Number of repetitions | `double` |

Data Types: `struct`

### `cw` — Codeword to be modulated

binary column vector

Codeword to be modulated, specified as a binary column vector.

### `statein` — Input encoder state

structure

Input encoder state for transmission of a bundle, specified as a structure containing the fields listed in the `stateout` output. At the start of the bundle transmission, set `statein` to empty. The `lteNPDCCH` function manages the state during subsequent calls for the transmissions of the bundle and resets it automatically at the end of the bundle.

Data Types: `struct`

## Output Arguments

### `sym` — NPDCCH symbols

complex-valued matrix

NPDCCH symbols, returned as an *N*-by-*P* complex-valued matrix, where *N* is the number of modulation symbols for one antenna port and *P* is the number of transmission antennas.

Data Types: `double`
Complex Number Support: Yes

### `stateout` — Output encoder state

structure

Output encoder state, returned as a structure. This output contains the internal state of each transport block in these fields.

| Name | Values | Description | Data Types |
|---|---|---|---|
| **SubframeIdx** | 0 (default),<br><br>integer in the interval [0, chs. NRep – 1] | Index of a subframe within a bundle, in zero-based form. The lteNPDCCH function returns this field as the SubframeIdx field of the statein input increased by one. When the input value of SubframeIdx in the statein input reaches its maximum value, the function returns this field as 0. If no input exists in statein, the default input is 0. A value of 0 indicates that the transmission has reached the end of a bundle, which the function also indicates by setting the EndOfTx field to 1 (true). | double |
| **InitNSubframe** | Nonnegative integer | Subframe number, either at the initialization point of the scrambling sequence or reinitialization point of the sequence done at every 4th NPDCCH subframe. When the subframe being processed is at the initialization or reinitialization point, this field is equal to the mod(enb.Nsubframe, 10). Otherwise, it is equal to the input in statein. If no input is provided to statein, the InitNSubframe is equal to mod(enb.NSubframe, 10). | double |

| Name | Values | Description | Data Types |
|---|---|---|---|
| **EndOfTx** | Logical `1` (`true`) or `0` (`false`) | End of bundle indicator. The `lteNPDCCH` function returns this field as `1` (`true`) when the transmission reaches the end of a bundle. Otherwise, the `lteNPDCCH` function returns this field as `0` (`false`). At the beginning of a bundle, the `lteNPDCCH` function resets this field. | `logical` |

Data Types: `struct`

## Tips

To use this function for a bundle transmission, follow these steps:

1. Call the `lteNPDCCH` function and specify the initial encoder state using the `statein` input. The `stateout` output represents the output encoder state of the first transmission of the bundle.
2. Call the `lteNPDCCH` function again and specify the `statein` input as the `stateout` output returned by the previous call to the function.
3. Repeat step 2 until the `lteNPDCCH` function returns the `EndOfTx` field of the `stateout` output as `1` (`true`), indicating the end of the bundle. The `lteNPDCCH` automatically resets the state at the end of the bundle transmission.

# Version History

**Introduced in R2019b**

## References

[1] 3GPP TS 36.211. “Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. URL: https://www.3gpp.org.

## See Also

`lteNPDCCHIndices` | `lteNPDCCHDecode` | `lteNPDSCH` | `ltePDCCH`

# lteNPDCCHDecode

Decode NPDCCH symbols

## Syntax

```
[cw,stateout,symbols] = lteNPDCCHDecode(enb,chs,sym)
[___] = lteNPDCCHDecode(enb,chs,sym,statein)
[___] = lteNPDCCHDecode(enb,chs,sym,hest,noiseest)
[___] = lteNPDCCHDecode(enb,chs,sym,hest,noiseest,statein)
```

## Description

`[cw,stateout,symbols] = lteNPDCCHDecode(enb,chs,sym)` decodes `sym`, the NB-IoT physical downlink control channel (NPDCCH) symbols, for cell-wide settings `enb` and channel-specific configuration structure `chs`. The channel decoding comprises deprecoding, layer demapping, soft demodulation, descrambling, and codeword recovery. The decoding inverts the NPDCCH channel encoding process described in 3GPP TS 36.211 Section 10.2.5 of [1]. The function returns a codeword `cw` of soft bits, the decoder state `stateout` for reception of a bundle (full set of repeated transmissions of a single downlink control information), and the received constellation symbols `symbols`.

`[___] = lteNPDCCHDecode(enb,chs,sym,statein)` decodes the NPDCCH symbols for the initial decoder state `statein`.

`[___] = lteNPDCCHDecode(enb,chs,sym,hest,noiseest)` decodes the NPDCCH symbols for the channel estimate `hest` and noise estimate `noiseest`.

`[___] = lteNPDCCHDecode(enb,chs,sym,hest,noiseest,statein)` decodes the NPDCCH symbols for the channel estimate, noise estimate, and initial decoder state.

## Examples

### Generate and Receive NPDCCH Symbols

Generate and receive the NPDCCH symbols subframe by subframe for a bundle of 10 subframes.

Specify the cell-wide settings and channel transmission configuration parameter structures `enb` and `chs`.

```
enb.NNCellID = 0;
enb.NBRefP = 1;
chs.NRep = 10;
```

Set the output codeword length to 320 and generate the codeword bits. Specify the encoder and decoder state as empty at the start of the bundle.

```
cwLen = 320;
cw = ones(cwLen,1); % Codeword bits
estate = [];
dstate = [];
```

Generate the NPDCCH symbols for each of the 10 subframes and then decode them.

```
for nsf = 0:chs.NRep-1
       enb.NSubframe = nsf;
       [sym,estate] = lteNPDCCH(enb,chs,cw,estate);
       [rxcw,dstate] = lteNPDCCHDecode(enb,chs,sym,dstate);
end
```

The value of the field `CWSFCount` in structure `dstate` indicates that NPDCCH subframe has been received ten times.

```
dstate.EndOfTx
```

```
ans = logical
   1
```

```
dstate.CWSFCount
```

```
ans = 10
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure containing these fields.

| Name | Required or Optional | Values | Description | Data Types |
|---|---|---|---|---|
| **NNCellID** | Required | Nonnegative integer | Narrowband physical layer cell identity | `double` |
| **NBRefP** | Required | 1, 2 | Number of narrowband reference signal (NRS) antenna ports. | `double` |
| **NSubframe** | Required | Nonnegative integer | Subframe number | `double` |

Data Types: `struct`

### chs — Channel transmission configuration

structure

Channel transmission configuration, specified as a structure that containing these fields.

| Name | Required or Optional | Values | Description | Data Types |
|---|---|---|---|---|
| **NRep** | Required | Nonnegative integer | Number of repetitions | `double` |

| Name | Required or Optional | Values | Description | Data Types |
|---|---|---|---|---|
| **CSI** | Optional | `'On'` (default), `'Off'` | Channel state information (CSI). To scale the soft bits by CSI during the equalization process, specify this field as `'On'`. Otherwise, specify this field as `'Off'`. | `char`, `string` |

Data Types: `struct`

**`sym` — Modulated NPDCCH symbols**
complex-valued matrix

Modulated NPDCCH symbols, specified as an $N_{RE}$-by-$N_{RxAnts}$ complex-valued matrix, where:

- $N_{RE}$ is the number of quadrature phase-shift keying (QPSK) symbols per antenna and per subframe assigned to the NPDCCH.
- $N_{RxAnts}$ is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

**`statein` — Initial encoder state**
structure

Initial encoder state for transmission of a bundle, specified as a structure containing the fields listed in the `stateout` output. At the start of the bundle transmission, set `statein` to empty. The `lteNPDCCHDecode` function manages the state during subsequent calls for the transmissions of the bundle and resets it automatically at the end of the bundle.

Data Types: `struct`

**`hest` — Channel estimate**
complex-valued 3-D array

Channel estimate, specified as an $N_{RE}$-by-$N_{RxAnts}$-by-$N_{NBRefP}$ complex-valued array, where:

- $N_{RE}$ is the number of QPSK symbols per antenna and per subframe.
- $N_{RxAnts}$ is the number of receive antennas.
- $N_{NBRefP}$ is the number NRS antenna ports you specify in the `NBRefP` field of the `enb` input.

The `lteNPDCCHDecode` function assumes that this estimate uses the NRSs.

Data Types: `double`
Complex Number Support: Yes

**`noiseest` — Noise estimate**
numeric scalar

Noise estimate of the noise power spectral density per resource element on the received subframe, specified as a numeric scalar. This estimate is provided by the `lteDLChannelEstimate` function.

Data Types: `double`

## Output Arguments

### `cw` — Codeword of soft bits

numeric vector

Codeword of soft bits, returned as a numeric column vector.

Data Types: `double`

### `stateout` — Output decoder state

structure

Output decoder state for the next subframe, returned as a structure. This output contains the internal state of each transport block in these fields.

| Name | Values | Description | Data Types |
|---|---|---|---|
| **`SubframeIdx`** | 0 (default),<br><br>integer in the interval [0, `chs.NRep` – 1] | Index of a subframe within a bundle, in zero-based form. The `lteNPDCCHDecode` function returns this field as the `SubframeIdx` field of the `statein` input increased by one. When the input value of `SubframeIdx` in the `statein` input reaches its maximum value, the function returns this field as `0`. If no input exists in `statein`, the default input is `0`. A value of `0` indicates that the transmission has reached the end of a bundle, which the function also indicates by setting the `EndOfTx` field to `1` (`true`). | `double` |

| Name | Values | Description | Data Types |
|---|---|---|---|
| **`InitNSubframe`** | Nonnegative integer | Subframe number, either at the initialization point of the scrambling sequence or reinitialization point of the sequence done at every 4th NPDCCH subframe. When the subframe being processed is at the initialization or reinitialization point, this field is equal to the `mod(enb.Nsubframe, 10)`. Otherwise, it is equal to the input in `statein`. If input to `statein` is not provided, the `InitNSubframe` is equal to `mod(enb.NSubframe, 10)`. | `double` |
| **`CWBuffer`** | Numeric column vector | Buffer to store the soft-combined log-likelihood ratio (LLR) bits after codeword descrambling. The length of this field is the same as the length of the codeword, `cw`. At the beginning of a bundle, the `lteNPDCCHDecode` function resets this field. | `double` |
| **`CWSFCount`** | 0 (default),<br><br>nonnegative integer | Repetition counter which indicates how many repetitions of `cw` the `CWBuffer` field has recovered. At the beginning of a bundle, the `lteNPDCCHDecode` function resets this field. | `double` |

| Name | Values | Description | Data Types |
|---|---|---|---|
| **EndOfCW** | Logical 1 (true) or 0 (false) | Codeword receipt indicator. The lteNPDCCHDecode function returns this field as 1 (true) when the entire codeword has been received and the CWSFCount field is as least 1. At the beginning of a bundle, the lteNPDCCHDecode function resets this field. | logical |
| **EndOfTx** | Logical 1 (true) or 0 (false) | End of bundle indicator. The lteNPDCCHDecode function returns this field as 1 (true) when the transmission reaches the end of a bundle. Otherwise, the lteNPDCCHDecode function returns this field as 0 (false). At the beginning of a bundle, the lteNPDCCHDecode function resets this field. | logical |

Data Types: struct

**symbols — Received constellation symbols**
complex-valued vector

Received constellation symbols, returned as a complex-valued vector.

Data Types: double
Complex Number Support: Yes

## Tips

To use this function for transmission of a bundle, follow these steps:

1. Call the lteNPDCCHDecode function and specify the initial encoder state using the statein input. The stateout output represents the output decoder state of the first transmission of the bundle.
2. Call the lteNPDCCHDecode function again and specify the statein input as the stateout output returned by the previous call to the function.
3. Repeat step 2 until the lteNPDCCHDecode function returns the EndOfTx field of the stateout output as 1 (true), indicating the end of the bundle. The lteNPDCCHDecode automatically resets the state at the end of the bundle transmission.

## Version History

**Introduced in R2019b**

## References

[1] 3GPP TS 36.211. “Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. URL: https://www.3gpp.org.

## See Also

`lteNPDCCH` | `lteNPDCCHIndices` | `lteNPDSCHDecode` | `ltePDCCHDecode`

# lteNPDCCHIndices

Generate NPDCCH RE indices

## Syntax

```
[ind,info] = lteNPDCCHIndices(enb,chs)
[ind,info] = lteNPDCCHIndices(enb,chs,opts)
```

## Description

`[ind,info] = lteNPDCCHIndices(enb,chs)` returns `ind`, a matrix containing narrowband physical downlink control channel (NPDCCH) resource element (RE) indices, and `info`, a structure containing information related to the indices. You can use `ind` to index elements of the subframe resource grid directly for all antenna ports in accordance with 3GPP TS 36.211 Section 10.2.5.5 of [1]. Initialize this function with cell-wide settings `enb` and channel transmission configuration `chs`.

`[ind,info] = lteNPDCCHIndices(enb,chs,opts)` formats the returned indices using options specified by `opts`.

## Examples

### Generate NPDCCH RE Indices and Info Structure

Generate the NPDCCH RE indices mapping and display related information.

Create the eNodeB structure cell-wide settings for one antenna.

```
enb.NNCellID = 10;
enb.NBRefP = 1;
```

Create the channel transmission configuration. Specify the value of narrowband control channel element (NCCE).

```
chs.NCCE = 0; % NPDCCH Format 0
```

Generate the NPDCCH RE indices column vector. Display the first seven indices.

```
[ind,info] = lteNPDCCHIndices(enb,chs);
ind(1:7)
```

```
ans = 7×1

     1
     2
     3
     4
     5
     6
    13
```

Display the fields contained in the `info` structure.

```
info.G
```

```
ans = 160
```

```
info.Gd
```

```
ans = 80
```

### Generate NPDCCH RE Indices

Generate the NPDCCH RE 0-based indices mapping in linear index form for two antennas.

Create the eNodeB structure cell-wide settings for two antennas.

```
enb.NNCellID = 10;
enb.NBRefP = 2;
```

Create the channel transmission configuration.

```
chs.NCCE = [0 1]; % NPDCCH Format 1
```

Generate the NPDCCH RE indices matrix. Display the first seven indices.

```
ind = lteNPDCCHIndices(enb,chs,{'0based','ind'});
ind(1:7,:)
```

```
ans = 7×2

     0   168
     1   169
     2   170
     3   171
     4   172
     5   173
     6   174
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure containing these fields.

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **`NNCellID`** | Required | Nonnegative integer | Narrowband physical layer cell identity | — | `double` |

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **`NBRefP`** | Required | 1, 2 | Number of narrowband reference signal (NRS) antenna ports | — | `double` |

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **OperationMode** | Optional | `'Standalone'` (default), `'Inband-SamePCI'`, `'Inband-DifferentPCI'`, `'Guardband'` | NB-IoT operation mode, specified as one of these values:<br>• `'Standalone'` – NB-IoT standalone operation within any 180-kHz band outside any LTE carrier bandwidth<br>• `'Inband-SamePCI'` – NB-IoT in-band operation with the same PCI as an LTE carrier<br>• `'Inband-DifferentPCI'` – NB-IoT in-band operation with a different PCI to an LTE carrier<br>• `'Guardband'` – NB-IoT guard-band operation utilizing unused resource blocks within the guard-band of an LTE carrier | — | `char`, `string` |

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **`CellRefP`** | Optional | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports. The value of this field must be either the value to which you set the `NBRefP` field (default) or 4. | This field applies only when you specify the `OperationMode` field as `'Inband-SamePCI'` or `'Inband-DifferentPCI'`. When you specify the `OperationMode` field as `'Inband-SamePCI'`, the `lteNPDCCHIndices` function sets this field to the value of the `NBRefP` field. | `double` |
| **`ControlRegionSize`** | See Dependencies column | 3 (default), scalar in the interval [0, 13] | LTE control region size. This field sets the starting OFDM symbol index (zero-based) in a subframe. | • Required when you specify the `OperationMode` field as `'Inband-SamePCI'` or `'Inband-DifferentPCI'`.<br>• The `lteNPDCCHIndices` function sets this field to `0` when you specify the `OperationMode` field as `'Standalone'` or `'Guardband'`. | `double` |

Data Types: `struct`

### `chs` — Channel transmission configuration

structure

Channel transmission configuration, specified as a structure containing this field.

| Name | Required or Optional | Values | Description | Data Types |
|---|---|---|---|---|
| **`NCCE`** | Required | `0`, `1` | Narrow band control channel element (NCCE) corresponds to six consecutive subcarriers in a subframe. NCCE 0 occupies subcarriers from `0` to `5` and NCCE 1 occupies subcarriers from `6` to `11`. Aggregation of `1` or `2` NCCE is used to transmit NPDCCH in two formats:<br>• Format `0` - has one NCCE and the value is either `0` or `1`.<br>• Format `1` - has two NCCE and the value is [`0` `1`]. | `double` |

Data Types: `struct`

### `opts` — Output format and index base of generated indices

character vector | cell array of character vectors | string array

Output format and index base of generated indices, specified as a character vector, a cell array of character vectors, or a string array. You can specify these options as a single character vector or string scalar by a space-separated list of values placed inside quotation marks. This field can contain any of these values.

| Option | Values | Description |
|---|---|---|

| | | |
|---|---|---|
| Output format | `'ind'` (default), `'sub'` | Output format of generated indices, specified as `'ind'` or `'sub'`. The function returns the indices as an $N_{RE}$-by-`NBRefP` matrix when you specify `'ind'`. $N_{RE}$ is the number of resource elements. The function returns the indices as an $N_{RE}$-by-3 matrix when you specify `'sub'`, where each row of the matrix contains the subcarrier, symbol, and antenna port as its first, second, and third entries, respectively. |
| Index base | `'1based'` (default), `'0based'` | Index base, specified as `'1based'` or `'0based'`. To generate indices whose first value is 1, specify `'1based'`. To generate indices whose first value is 0, specify `'0based'`. |

Example: `'ind 0based'`, `"ind 0based"`, `{'ind','0based'}`, and `["ind","0based"]` specify the same output options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `ind` — NPDCCH RE indices

real-valued matrix

NPDCCH RE indices, returned as an $N_{RE}$-by-$P$ real-valued matrix, where $N_{RE}$ is the number of resource elements and $P$ is the number of resource array planes. Each column of `ind` contains the per-antenna indices for the $N_{RE}$ resource elements in each of the $P$ resource array planes.

Data Types: `double`

### `info` — Information related to NPDCCH indices

structure

Information related to NPDCCH indices, returned as a structure containing these fields.

| Name | Values | Description | Data Types |
|---|---|---|---|
| **`G`** | scalar | Number of coded and rate-matched downlink control information (DCI) data bits for a codeword. | `double` |
| **`Gd`** | scalar | Number of coded and rate-matched DCI data symbols per layer. | `double` |

Data Types: `struct`

## Version History

**Introduced in R2019b**

## References

[1] 3GPP TS 36.211. “Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. URL: https://www.3gpp.org.

## See Also

`lteNPDCCH` | `lteNPDCCHDecode` | `lteNPDSCHIndices` | `ltePDCCHIndices`

# lteNPDSCH

Generate NPDSCH symbols

## Syntax

```
[sym,stateout] = lteNPDSCH(enb,chs,cw)
[sym,stateout] = lteNPDSCH( ___ ,statein)
```

## Description

`[sym,stateout] = lteNPDSCH(enb,chs,cw)` returns `sym`, a matrix containing the encoded narrowband physical downlink shared channel (NPDSCH) symbols for cell-wide settings `enb`, channel transmission configuration `chs`, and codeword `cw`. The channel encoding process comprises subframe selection, scrambling, symbol modulation, layer mapping, and precoding in accordance with Section 10.2.3 of [1]. The function also returns `stateout`, a structure containing the encoder state for bundle transmission.

`[sym,stateout] = lteNPDSCH( ___ ,statein)` returns the NPDSCH symbols and encoder state for the initial encoder state specified by `statein`.

## Examples

### Generate NPDSCH Symbols

Generate the NPDSCH symbols subframe by subframe for a bundle of 12 subframes.

Specify the cell-wide settings and channel transmission configuration in parameter structures `enb` and `chs`.

```
enb.NNCellID = 0;
enb.NBRefP = 1;
enb.NFrame = 1;
chs.NSF = 3;
chs.NRep = 4;
chs.RNTI = 0;
chs.NPDSCHDataType = 'NotBCCH';
```

Set the output codeword length to 960 and generate the codeword bits. Do not provide the encoder state at the start of the bundle.

```
cwLen = 960;
cw = ones(cwLen,1);
statein = [];
```

Generate the NPDSCH symbols for each of the 12 subframes.

```
for subframeIdx = 0:(chs.NSF*chs.NRep-1)
    enb.NSubframe = subframeIdx;
    [txsym,stateout] = lteNPDSCH(enb,chs,cw,statein);
    statein = stateout;
```

```
end
disp(stateout.EndOfTx)
```

```
   1
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure containing these fields:

| Name | Required or Optional | Values | Description | Data Types |
|---|---|---|---|---|
| **NNCellID** | Required | Nonnegative integer | Narrowband physical layer cell identity (PCI) | `double` |
| **NSubframe** | Required | Nonnegative integer | Subframe number | `double` |
| **NFrame** | Optional | `0` (default), nonnegative integer | Frame number | `double` |
| **NBRefP** | Required | 1, 2 | Number of narrowband reference signal (NRS) antenna ports. To indicate transmission on a single antenna port (port 0) and use minimum mean squared error (MMSE) equalization for reception, specify this field as 1. To indicate transmit diversity and use an orthogonal space frequency block code (OSFBC) decoder for deprecoding, specify this field as 2. | `double` |

Data Types: `struct`

### chs — Channel transmission configuration

structure

Channel transmission configuration, specified as a structure containing these fields:

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **NPDSCHDataType** | Optional | `'NotBCCH'`, `'SIB1NB'`, `'BCCHNotSIB1NB'` | Type of data carried by the NPDSCH, specified as one of these values:<br>• `'NotBCCH'` – The NPDSCH is not carrying the broadcast control channel (BCCH).<br>• `'SIB1NB'` – The NPDSCH is carrying system information block 1 narrowband (SIB1-NB).<br>• `'BCCHNotSIB1NB'` – The NPDSCH is carrying the BCCH but not SIB1-NB. | — | `char`, `string` |

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **NSF** | See Dependencies column | Nonnegative integer | Number of subframes to which a codeword is mapped, not including repetitions | • This field is required when you specify the `NPDSCHDataType` field as a value other than `'SIB1NB'` and return the `info` output.<br>• The `lteNPDSCH` function sets this field to `8` when you specify the `NPDSCHDataType` field as `'SIB1NB'` and return the `info` output.<br>• If you do not return the `info` output, the `lteNPDSCH` function ignores this field. | `double` |
| **NRep** | Required | Nonnegative integer | Number of repetitions | — | `double` |

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **RNTI** | See Dependencies column | Nonnegative integer | 16-bit radio network temporary identifier (RNTI) | • This field is required when you specify the `NPDSCHDataType` field as a value other than `'SIB1NB'`.<br>• The `lteNPDSCH` function sets this field to the system information RNTI (SI-RNTI) value of `65535` when you specify the `NPDSCHDataType` field as `'SIB1NB'`. | `double` |
| **CSI** | Optional | `'On'` (default), `'Off'` | Channel state information (CSI). To scale the soft bits by CSI during the equalization process, specify this field as `'On'`. Otherwise, specify this field as `'Off'`. | — | `char`, `string` |

Data Types: `struct`

### `cw` — Codeword to be modulated

binary column vector

Codeword to be modulated, specified as a binary column vector.

Data Types: `double`

### `statein` — Initial encoder state

structure

Initial encoder state for the transmission of a bundle, specified as a structure containing the fields listed in the `stateout` output. This argument can be empty only when no information is provided, such as at the first subframe of a bundle.

Data Types: `struct`

## Output Arguments

### `sym` — NPDSCH symbols

complex-valued matrix

NPDSCH symbols, returned as an *N*-by-*P* complex-valued matrix, where *N* is the number of modulation symbols for one antenna port and *P* is the number of transmission antennas.

Data Types: `double`
Complex Number Support: Yes

### `stateout` — Output encoder state

structure

Output encoder state, returned as a structure. This output contains the internal state of each transport block in these fields:

| Name | Values | Description | Data Types |
|---|---|---|---|
| **`SubframeIdx`** | integer in the interval [0, `NSF` x `NRep` – 1] | Index of a subframe within a bundle, in zero-based form. The `lteNPDSCH` function returns this field as the `SubframeIdx` field of the `statein` input increased by one. When the input value of `SubframeIdx` in the `statein` input reaches its maximum value, the function returns this field as `0`. If you do not specify an input value in the `statein` input, the `lteNPDSCH` function returns this field as `0`. A value of `0` indicates that the transmission has reached the end of a bundle, which the function also indicates by setting the `EndOfTx` field to `1` (`true`). | `double` |

| Name | Values | Description | Data Types |
|---|---|---|---|
| **`InitNFrame`** | Nonnegative integer | Frame number at initialization point of scrambling sequence. When the subframe being processed is at the initialization point, this field is equal to the `NFrame` field of the `enb` input. Otherwise, the `lteNPDSCH` function returns this field as one of these values:<br>• The value of the `InitNFrame` field of the `statein` argument<br>• `0` when you do not specify the `InitNFrame` field of the `statein` input | `double` |
| **`InitNSubrame`** | Nonnegative integer | Subframe number at initialization point. When the subframe being processed is at the initialization point, this field is equal to the `NSubframe` field of the `enb` input. Otherwise, the `lteNPDSCH` function returns this field as one of these values:<br>• The value of the `InitNSubframe` field of the `statein` argument<br>• The `NSubframe` field of the `enb` input when you do not specify the `InitNSubframe` field of the `statein` input | `double` |

| Name | Values | Description | Data Types |
|---|---|---|---|
| **`CWBuffer`** | $N_{SF}$-by-1 binary vector | Buffer to store the soft-combined log-likelihood ratio (LLR) bits after codeword descrambling. The length of this field is the same as the length of the codeword, `cw`. At the beginning of a bundle, the `lteNPDSCH` function resets this field. | `double` |
| **`CWSFCount`** | $N_{SF}$-by-1 integer-valued vector | Repetition counter. The length of this field is the same as the length of the codeword, `cw`. Each element of this field indicates how many repetitions of the corresponding element of `cw` the `CWBuffer` field has recovered. At the beginning of a bundle, the `lteNPDSCH` function resets this field. | `double` |
| **`EndOfCW`** | Logical `1` (`true`) or `0` (`false`) | Codeword receipt indicator. The `lteNPDSCH` function returns this field as `1` (`true`) when the entire codeword has been received, that is, when each element of the `CWSFCount` field is as least `1`. At the beginning of a bundle, the `lteNPDSCH` function resets this field. | `logical` |

| Name | Values | Description | Data Types |
|---|---|---|---|
| **EndOfTx** | Logical `1` (`true`) or `0` (`false`) | End of bundle indicator. The `lteNPDSCH` function returns this field as `1` (`true`) when the transmission reaches the end of a bundle. Otherwise, the `lteNPDSCH` function returns this field as `0` (`false`). At the beginning of a bundle, the `lteNPDSCH` function resets this field. | `logical` |

Data Types: `struct`

## More About

### Bundle

A bundle in the medium access control (MAC) layer refers to the repeated transmissions of a transport block.

For more information, see Section 5.3.2.1 of [2]

## Tips

To use this function for transmission of a bundle, follow these steps:

1. Call the `lteNPDSCH` function, optionally specifying the initial encoder state using the `statein` input; the `stateout` output represents the first transmission of the transport block.
2. Call the `lteNPDSCH` function again, specifying the `statein` input as the `stateout` output returned by the previous call to the function.
3. Repeat step 2 until the `lteNPDSCH` function returns the `EndOfTx` field of the `stateout` output as `1` (`true`), indicating the end of the bundle.

# Version History

**Introduced in R2018a**

## References

[1] 3GPP TS 36.211. “Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.321. “Medium Access Control (MAC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. URL: https://www.3gpp.org.

## See Also

`lteNPDSCHIndices` | `lteNPDSCHDecode` | `lteNDLSCH` | `ltePDSCH`

# lteNPDSCHDecode

Decode NPDSCH symbols

## Syntax

```
[cw,stateout,symbols] = lteNPDSCHDecode(enb,chs,sym)
[cw,stateout,symbols] = lteNPDSCHDecode(enb,chs,sym,statein)
[cw,stateout,symbols] = lteNPDSCHDecode(enb,chs,sym,hest,noiseest)
[cw,stateout,symbols] = lteNPDSCHDecode(enb,chs,sym,hest,noiseest,statein)
```

## Description

`[cw,stateout,symbols] = lteNPDSCHDecode(enb,chs,sym)` decodes `sym`, the NB-IoT physical downlink shared channel (NPDSCH) symbols, for cell-wide settings `enb` and channel-specific configuration structure `chs`. The channel decoding comprises deprecoding, layer demapping, soft demodulation, descrambling, and codeword recovery. The decoding inverts the NPDSCH channel encoding process described in Section 10.2.3 of [1]. The function returns a codeword `cw` of soft bits, the decoder state `stateout` for reception of a bundle transmission, and the received constellation symbols `symbols`.

`[cw,stateout,symbols] = lteNPDSCHDecode(enb,chs,sym,statein)` decodes the NPDSCH symbols for the initial decoder state `statein`.

`[cw,stateout,symbols] = lteNPDSCHDecode(enb,chs,sym,hest,noiseest)` decodes the NPDSCH symbols for the channel estimate `hest` and noise estimate `noiseest`.

`[cw,stateout,symbols] = lteNPDSCHDecode(enb,chs,sym,hest,noiseest,statein)` decodes the NPDSCH symbols for the channel estimate, noise estimate, and initial decoder state.

## Examples

### Generate and Receive NPDSCH Symbols

Generate and receive the NPDSCH symbols subframe by subframe for a bundle of 12 subframes.

Specify the cell-wide settings and channel transmission configuration in parameter structures `enb` and `chs`.

```
enb.NNCellID = 0;
enb.NBRefP = 1;
enb.NFrame = 1;
chs.NSF = 3;
chs.NRep = 4;
chs.RNTI = 0;
chs.NPDSCHDataType = 'NotBCCH';
```

Set the output codeword length to 960 and generate the codeword bits. Do not provide the encoder or decoder states at the start of the bundle.

```
cwLen = 960;
eState = [];
statein = [];
txcw = ones(cwLen,1);
```

Generate the NPDSCH symbols for each of the 12 subframes and then decode them.

```
for subframeIdx = 0:(chs.NSF*chs.NRep-1)
    enb.NSubframe = subframeIdx;
    [sym,eState] = lteNPDSCH(enb,chs,txcw,eState);
    [cw,stateout] = lteNPDSCHDecode(enb,chs,sym,statein);
    statein = stateout;
end
```

The value of the field `CWSFCount` in structure `dstate` indicates that all three subframes in the codeword have been received four times.

```
disp(stateout.EndOfTx)
```

```
   1
```

```
disp(stateout.CWSFCount)
```

```
     4
     4
     4
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure containing these fields:

| Name | Required or Optional | Values | Description | Data Types |
|---|---|---|---|---|
| **`NNCellID`** | Required | Nonnegative integer | Narrowband physical layer cell identity (PCI) | `double` |
| **`NSubframe`** | Required | Nonnegative integer | Subframe number | `double` |
| **`NFrame`** | Optional | `0` (default), nonnegative integer | Frame number | `double` |

| Name | Required or Optional | Values | Description | Data Types |
|---|---|---|---|---|
| **NBRefP** | Required | 1, 2 | Number of narrowband reference signal (NRS) antenna ports. To indicate transmission on a single antenna port (port 0) and use minimum mean squared error (MMSE) equalization for reception, specify this field as 1. To indicate transmit diversity and use an orthogonal space frequency block code (OSFBC) decoder for deprecoding, specify this field as 2. | `double` |

Data Types: `struct`

**chs — Channel transmission configuration**
structure

Channel transmission configuration, specified as a structure that containing these fields:

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **NPDSCHDataType** | Optional | `'NotBCCH'`, `'SIB1NB'`, `'BCCHNotSIB1NB'` | Type of data carried by the NPDSCH, specified as one of these values:<br>• `'NotBCCH'` – The NPDSCH is not carrying the broadcast control channel (BCCH).<br>• `'SIB1NB'` – The NPDSCH is carrying system information block 1 narrowband (SIB1-NB).<br>• `'BCCHNotSIB1NB'` – The NPDSCH is carrying the BCCH but not SIB1-NB. | — | `char`, `string` |

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **NSF** | See Dependencies column | Nonnegative integer | Number of subframes to which a codeword is mapped, not including repetitions | • This field is required when you specify the `NPDSCHDataType` field as a value other than `'SIB1NB'` and return the `info` output.<br>• The `lteNPDSCHDecode` function sets this field to `8` when you specify the `NPDSCHDataType` field as `'SIB1NB'` and return the `info` output.<br>• If you do not return the `info` output, the `lteNPDSCHDecode` function ignores this field. | `double` |
| **NRep** | Required | Nonnegative integer | Number of repetitions | — | `double` |

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **RNTI** | See Dependencies column | Nonnegative integer | 16-bit radio network temporary identifier (RNTI) | • This field is required when you specify the `NPDSCHDataType` field as a value other than `'SIB1NB'`.<br>• The `lteNPDSCHDecode` function sets this field to the system information RNTI (SI-RNTI) value of `65535` when you specify the `NPDSCHDataType` field as `'SIB1NB'`. | `double` |
| **CSI** | Optional | `'On'` (default), `'Off'` | Channel state information (CSI). To scale the soft bits by CSI during the equalization process, specify this field as `'On'`. Otherwise, specify this field as `'Off'`. | — | `char`, `string` |

Data Types: `struct`

**`sym` — Modulated NPDSCH symbols**
complex-valued matrix

Modulated NPDSCH symbols, specified as an $N_{RE}$-by-$N_{RxAnts}$ complex-valued matrix, where:

- $N_{RE}$ is the number of quadrature phase-shift keying (QPSK) symbols per antenna and per subframe assigned to the NPDSCH;
- $N_{RxAnts}$ is the number of receive antennas.

Data Types: `double`

Complex Number Support: Yes

**`statein` — Initial encoder state**
structure

Input encoder state for the transmission of a bundle, specified as a structure containing the fields listed in the `stateout` output. This argument can be empty only when no information is provided, such as at the first subframe of a bundle.

Data Types: `struct`

**`hest` — Channel estimate for a transmission layer**
complex-valued 3-D array

Channel estimate for a transmission layer, specified as an $N_{RE}$-by-$N_{RxAnts}$-by-$N_{NBRefP}$ complex-valued array, where:

- $N_{RE}$ is the number of encoded NPDSCH symbols per antenna and per subframe;
- $N_{RxAnts}$ is the number of receive antennas;
- $N_{NBRefP}$ is the number NRS antenna ports you specify in the `NBRefP` field of the `enb` input.

The `lteNPDSCHDecode` function assumes that this estimate uses the NRSs.

Data Types: `double`

**`noiseest` — Noise estimate**
nonnegative scalar

Noise estimate of the noise power spectral density per RE on the received subframe, specified as a nonnegative scalar.

Data Types: `double`

## Output Arguments

**`cw` — Codeword of soft bits**
binary vector

Codeword of soft bits, returned as an $N_{SF}$-by-1 binary vector, where $N_{SF}$ is the number of subframes.

Data Types: `double`

**`stateout` — Output decoder state**
structure

Output decoder state for the next subframe, returned as a structure. This output contains the internal state of each transport block in these fields:

| Name | Values | Description | Data Types |
|---|---|---|---|
| **SubframeIdx** | integer in the interval [0, NSF x NRep – 1] | Index of a subframe within a bundle, in zero-based form. The lteNPDSCHDecode function returns this field as the SubframeIdx field of the statein input increased by one. When the input value of SubframeIdx in the statein input reaches its maximum value, the function returns this field as 0. If you do not specify an input value in the statein input, the lteNPDSCHDecode function returns this field as 0. A value of 0 indicates that the transmission has reached the end of a bundle, which the function also indicates by setting the EndOfTx field to 1 (true). | double |
| **InitNFrame** | Nonnegative integer | Frame number at initialization point of scrambling sequence. When the subframe being processed is at the initialization point, this field is equal to the NFrame field of the enb input. Otherwise, the lteNPDSCHDecode function returns this field as one of these values:<br>• The value of the InitNFrame field of the statein argument<br>• 0 when you do not specify the InitNFrame field of the statein input | double |

| Name | Values | Description | Data Types |
|---|---|---|---|
| **`InitNSubrame`** | Nonnegative integer | Subframe number at initialization point. When the subframe being processed is at the initialization point, this field is equal to the `NSubframe` field of the `enb` input. Otherwise, the `lteNPDSCHDecode` function returns this field as one of these values:<br>• The value of the `InitNSubframe` field of the `statein` argument<br>• The `NSubframe` field of the `enb` input when you do not specify the `InitNSubframe` field of the `statein` input | `double` |
| **`CWBuffer`** | $N_{SF}$-by-1 binary vector | Buffer to store the soft-combined log-likelihood ratio (LLR) bits after codeword descrambling. The length of this field is the same as the length of the codeword, `cw`. At the beginning of a bundle, the `lteNPDSCHDecode` function resets this field. | `double` |

| Name | Values | Description | Data Types |
|---|---|---|---|
| **CWSFCount** | $N_{SF}$-by-1 integer-valued vector | Repetition counter. The length of this field is the same as the length of the codeword, `cw`. Each element of this field indicates how many repetitions of the corresponding element of `cw` the `CWBuffer` field has recovered. At the beginning of a bundle, the `lteNPDSCHDecode` function resets this field. | `double` |
| **EndOfCW** | Logical `1` (`true`) or `0` (`false`) | Codeword receipt indicator. The `lteNPDSCHDecode` function returns this field as `1` (`true`) when the entire codeword has been received, that is, when each element of the `CWSFCount` field is as least `1`. At the beginning of a bundle, the `lteNPDSCHDecode` function resets this field. | `logical` |
| **EndOfTx** | Logical `1` (`true`) or `0` (`false`) | End of bundle indicator. The `lteNPDSCHDecode` function returns this field as `1` (`true`) when the transmission reaches the end of a bundle. Otherwise, the `lteNPDSCHDecode` function returns this field as `0` (`false`). At the beginning of a bundle, the `lteNPDSCHDecode` function resets this field. | `logical` |

Data Types: `struct`

**`symbols` — Received constellation symbols**
complex-valued vector

Received constellation symbols, returned as a complex-valued vector.

Data Types: `double`

## More About

### Bundle

A bundle in the medium access control (MAC) layer refers to the repeated transmissions of a transport block.

For more information, see Section 5.3.2.1 of [2].

## Tips

To use this function for a bundle transmission, follow these steps:

1. Call the `lteNPDSCHDecode` function, optionally specifying the initial encoder state using the `statein` input; the `stateout` output represents the first transport block in the bundle.
2. Call the `lteNPDSCHDecode` function again, specifying the `statein` input as the `stateout` output returned by the previous call to the function.
3. Repeat step 2 until the `lteNPDSCHDecode` function returns the `EndOfTx` field of the `stateout` output as `1` (`true`), indicating the end of the bundle.

# Version History

**Introduced in R2018a**

## References

[1] 3GPP TS 36.211. “Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.321. “Medium Access Control (MAC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. URL: https://www.3gpp.org.

## See Also

`lteNPDSCH` | `lteNPDSCHIndices` | `lteNDLSCHDecode` | `ltePDSCHDecode`

# lteNPBCHIndices

Generate NPBCH RE indices

## Syntax

```
[ind,info] = lteNPBCHIndices(enb)
[ind,info] = lteNPBCHIndices(enb,opts)
```

## Description

`[ind,info] = lteNPBCHIndices(enb)` generates `ind`, a matrix containing narrowband physical broadcast channel (NPBCH) resource element (RE) indices, and `info`, a structure containing information related to the indices. You can use `ind` to index elements of the subframe resource grid directly for all antenna ports in accordance with section 10.2.4.4 of [1]. Initialize this function with cell-wide settings `enb`.

`[ind,info] = lteNPBCHIndices(enb,opts)` formats the returned indices using options specified by `opts`.

## Examples

### Generate NPBCH RE Indices and Info Structure

Generate the NPBCH RE Indices mapping for one antenna and display related information.

Create the eNodeB structure cell-wide settings for one antenna.

```
enb.NNCellID=10;
enb.NBRefP=1;
```

Generate the NPBCH RE Indices column vector. Display the first 10 indices.

```
[ind,info]=lteNPBCHIndices(enb);
ind(1:10)
```

```
ans = 10×1

    37
    38
    39
    40
    41
    42
    43
    44
    45
    46
```

Display the fields contained in the `info` structure.

```
info.G
```

```
ans = 1600
```

```
info.Gd
```

```
ans = 800
```

### Generate NPBCH Indices

Generate NPBCH RE 0-based Indices in linear index form for two antennas.

Create eNodeB structure cell-wide settings for two antennas.

```
enb.NNCellID = 10;
enb.NBRefP = 2;
```

Generate the 0-based NPBCH RE indices in linear index form. The indices matrix has two columns, one for each antenna port. Display the first 10 indices.

```
[ind,info] = lteNPBCHIndices(enb,{'0based', 'ind'});
ind(1:10,:)
```

```
ans = 10×2

    36   204
    37   205
    38   206
    39   207
    40   208
    41   209
    42   210
    43   211
    44   212
    45   213
```

Display the fields contained in the `info` structure.

```
info.G
```

```
ans = 1600
```

```
info.Gd
```

```
ans = 800
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure containing these fields.

| Parameter Field | Required or Optional | Values | Description | Data Types |
|---|---|---|---|---|
| **NNCellID** | Required | Nonnegative integer | NB-IoT physical layer cell identity | `double` |
| **NBRefP** | Required | `1`, `2` | Number of narrowband reference signal antenna ports | `double` |
| **NSubframe** | Optional | `0` (default),<br><br>nonnegative integer | Subframe number | `double` |
| **NFrame** | Optional | `0` (default),<br><br>nonnegative integer | Frame number | `double` |

Data Types: `struct`

**`opts` — Output format and index base of generated indices**
character vector | string scalar | cell array of character vectors | string array

Output format and index base of generated indices, specified as one of these forms.

- `'format base'`
- `"format base"`
- `{'format','base'}`
- `["format","base"]`

Where `format` and `base` are defined in this table.

| Option | Values | Description |
|---|---|---|
| **format** | `'ind'` (default), `'sub'` | Output format of generated indices<br><br>To return the indices as a column vector, specify this option as `'ind'`.<br><br>To return the indices as an $N_{RE}$-by-3 matrix, where $N_{RE}$ is the number of REs, specify this option as `'sub'`. Each row of the matrix contains the subcarrier, symbol, and antenna port as its first, second, and third element, respectively. |

| | | |
|---|---|---|
| **base** | `'1based'` (default), `'0based'` | Index base<br><br>To generate indices whose first value is `1`, specify this option as `'1based'`. To generate indices whose first value is `0`, specify this option as `'0based'`. |

Example: `'ind 0based'`, `"ind 0based"`, `{'ind','0based'}`, and `["ind","0based"]` specify the same output options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `ind` — NPBCH RE indices

real-valued matrix

NPBCH RE indices, returned as a real-valued matrix, depending on the indexing style specified in `opts`:

- If you specify linear indexing (default), then `ind` is an 100-by-`NBRefP` matrix.
- If you specify subscript row style indexing, then `ind` is an $N_{RE}$-by-3 matrix, where $N_{RE}$ is the number of resource elements per subframe.

Data Types: `double`

### `info` — Information related to NPBCH indices

structure

Information related to NPBCH indices, returned as a structure containing these fields.

| Parameter Field | Values | Description | Data Types |
|---|---|---|---|
| **G** | 1600 | Number of coded and rate-matched NPBCH data bits for a codeword. | `double` |
| **Gd** | 800 | Number of coded and rate-matched NPBCH data symbols for each of the 8 sub blocks. | `double` |

Data Types: `struct`

# Version History

**Introduced in R2019b**

## References

[1] 3GPP TS 36.211. “Physical channels and modulation (Release 14).” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. URL: https://www.3gpp.org.

## See Also

`lteNPBCH` | `lteNPBCHDecode` | `ltePBCHIndices`

# lteNPDSCHIndices

Generate NPDSCH RE indices

## Syntax

```
[ind,info] = lteNPDSCHIndices(enb,chs)
[ind,info] = lteNPDSCHIndices(enb,chs,opts)
```

## Description

`[ind,info] = lteNPDSCHIndices(enb,chs)` returns `ind`, an array containing narrowband physical downlink shared channel (NPDSCH) resource element (RE) indices, and `info`, a structure containing information related to the indices. You can use `ind` to index elements of the subframe resource grid directly for all antenna ports in accordance with Section 10.2.3.4 of [1]. Initialize this function with cell-wide settings `enb` and channel transmission configuration `chs`.

`[ind,info] = lteNPDSCHIndices(enb,chs,opts)` formats the returned indices using options specified by `opts`.

## Examples

### Generate NPDSCH RE Indices and Info Structure

Generate the NPDSCH RE indices mapping for one antenna and display related information.

Create the eNodeB structure cell-wide settings for one antenna.

```
enb.NNCellID = 10;
enb.NBRefP = 1;
enb.OperationMode = 'Inband-SamePCI';
```

Create the channel transmission configuration. Specify the number of subframes (NSF). The NSF field is required for returning information related to the NPDSCH indices.

```
chs.NPDSCHDataType = 'BCCHNotSIB1NB';
chs.NSF = 2;
```

Generate the NPDSCH RE indices column vector. Display the first seven indices.

```
[ind,info] = lteNPDSCHIndices(enb,chs);
ind(1:7)
```

```
ans = 7×1

    37
    38
    39
    40
    41
    42
```

```
    43
```

Display the fields contained in the `info` structure.

```
info.G
```

```
ans = 472
```

```
info.Gd
```

```
ans = 236
```

### Generate NPDSCH RE Indices

Generate the NPDSCH RE 0-based indices mapping in linear index form for two antennas.

Create the eNodeB structure cell-wide settings for two antennas.

```
enb.NNCellID = 10;
enb.NBRefP = 2;
enb.OperationMode = 'Standalone';
```

Create the channel transmission configuration.

```
chs.NPDSCHDataType = 'SIB1NB';
```

Generate the 0-based NPDSCH RE indices in linear index form. The indices matrix has two columns, one for each antenna ports. Display the first seven indices.

```
ind = lteNPDSCHIndices(enb,chs,{'0based','ind'});
ind(1:7,:)
```

```
ans = 7×2

     0   168
     1   169
     2   170
     3   171
     4   172
     5   173
     6   174
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure containing these fields:

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **`NNCellID`** | Required | Nonnegative integer | Narrowband physical layer cell identity (PCI) | — | `double` |
| **`NBRefP`** | Required | 1, 2 | Number of narrowband reference signal (NRS) antenna ports | — | `double` |

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **OperationMode** | Optional | `'Standalone'` (default), `'Inband-SamePCI'`, `'Inband-DifferentPCI'`, `'Guardband'` | NB-IoT operation mode, specified as one of these values:<br>• `'Standalone'` – NB-IoT standalone operation within any 180-kHz band outside any LTE carrier bandwidth<br>• `'Inband-SamePCI'` – NB-IoT in-band operation with the same PCI as an LTE carrier<br>• `'Inband-DifferentPCI'` – NB-IoT in-band operation with a different PCI to an LTE carrier<br>• `'Guardband'` – NB-IoT guard-band operation utilizing unused resource blocks within the guard-band of an LTE carrier | — | `char`, `string` |

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **`CellRefP`** | Optional | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports. The value of this field must be either the value to which you set the `NBRefP` field (default) or 4. | This field applies only when you specify the `OperationMode` field as `'Inband-SamePCI'` or `'Inband-DifferentPCI'`. When you specify the `OperationMode` field as `'Inband-SamePCI'`, the `lteNPDSCHIndices` function sets this field to the value of the `NBRefP` field. | `double` |

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **ControlRegionSize** | See Dependencies column | 3 (default), scalar in the interval [0, 13] | LTE control region size. This field sets the starting OFDM symbol index (zero-based) in a subframe. | • This field is required when you specify the OperationMode field as 'Inband-SamePCI' or 'Inband-DifferentPCI' and the NPDSCHDataType field of the chs input as a value other than 'SIB1NB'.<br>• The lteNPDSCHIndices function sets this field to 3 when you specify the OperationMode field as 'Inband-SamePCI' or 'Inband-DifferentPCI' and the NPDSCHDataType field of the chs input as 'SIB1NB'.<br>• The lteNPDSCHIndices function sets this field to 0 when you specify the Operation | double |

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| | | | | `Mode` field as `'Standalone'` or `'Guardband'`. | |

Data Types: `struct`

**`chs` — Channel transmission configuration**
structure

Channel transmission configuration, specified as a structure containing these fields:

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **`NPDSCHDataType`** | Optional | `'NotBCCH'`, `'SIB1NB'`, `'BCCHNotSIB1NB'` | Type of data carried by the NPDSCH, specified as one of these values:<br>• `'NotBCCH'` – The NPDSCH is not carrying the broadcast control channel (BCCH).<br>• `'SIB1NB'` – The NPDSCH is carrying system information block 1 narrowband (SIB1-NB).<br>• `'BCCHNotSIB1NB'` – The NPDSCH is carrying the BCCH but not SIB1-NB. | — | `char`, `string` |

| Name | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **NSF** | See Dependencies column | Nonnegative integer | Number of subframes to which a codeword is mapped, not including repetitions | • This field is required when you specify the `NPDSCHDataType` field as a value other than `'SIB1NB'` and return the `info` output.<br>• The `lteNPDSCHIndices` function sets this field to `8` when you specify the `NPDSCHDataType` field as `'SIB1NB'` and return the `info` output.<br>• If you do not return the `info` output, the `lteNPDSCHIndices` function ignores this field. | `double` |

Data Types: `struct`

**`opts` — Output format and index base of generated indices**
character vector | string scalar | cell array of character vectors | string array

Output format and index base of generated indices, specified as one of these forms.

- `'format base'`
- `"format base"`
- `{'format','base'}`
- `["format","base"]`

Where `format` and `base` are defined in this table.

| Option | Values | Description |
|---|---|---|
| **format** | `'ind'` (default), `'sub'` | Output format of generated indices<br><br>To return the indices as a column vector, specify this option as `'ind'`.<br><br>To return the indices as an $N_{RE}$-by-3 matrix, where $N_{RE}$ is the number of REs, specify this option as `'sub'`. Each row of the matrix contains the subcarrier, symbol, and antenna port as its first, second, and third element, respectively. |
| **base** | `'1based'` (default), `'0based'` | Index base<br><br>To generate indices whose first value is `1`, specify this option as `'1based'`. To generate indices whose first value is `0`, specify this option as `'0based'`. |

Example: `'ind 0based'`, `"ind 0based"`, `{'ind','0based'}`, and `["ind","0based"]` specify the same output options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `ind` — NPDSCH RE indices

real-valued matrix

NPDSCH RE indices, returned as an $N_{RE}$-by-$P$ real-valued matrix, where $N_{RE}$ is the number of resource elements and $P$ is the number of resource array planes. Each column of `ind` contains the per-antenna indices for the $N_{RE}$ resource elements in each of the $P$ resource array planes.

Data Types: `double`

### `info` — Information related to NPDSCH indices

structure

Information related to NPDSCH indices, returned as a structure containing these fields:

| Name | Values | Description | Data Types |
|---|---|---|---|
| **G** | scalar | Number of coded and rate-matched downlink shared channel (DL-SCH) data bits for a codeword | `double` |

| Name | Values | Description | Data Types |
|---|---|---|---|
| **Gd** | integer | Number of DL-SCH data symbols per layer. The `lteNPDSCHIndices` function returns `Gd` as the value of the `NSF` field of the `chs` input multiplied by the number of rows in the `ind` output. | `double` |

Data Types: `struct`

# Version History

**Introduced in R2018a**

## References

[1] 3GPP TS 36.211. “Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. URL: https://www.3gpp.org.

## See Also

`lteNPDSCH` | `lteNPDSCH` | `lteNPDSCHDecode` | `lteNRSIndices` | `ltePDSCHIndices`

# lteNPRACH

Generate NPRACH FDD waveform

## Syntax

```
[waveform,info,resourceGrid] = lteNPRACH(ue,chs)
```

## Description

`[waveform,info,resourceGrid] = lteNPRACH(ue,chs)` generates `waveform`, a time-domain narrowband physical random access channel (NPRACH) frequency-division duplexing (FDD) waveform for user equipment (UE) settings `ue` and channel transmission configuration `chs`. The function also returns `info`, a structure containing NPRACH information, and `resourceGrid`, an NPRACH resource grid.

## Examples

### Generate NPRACH FDD Waveform

Generate an NPRACH waveform for specified UE settings and channel transmission configuration.

Specify the narrowband physical layer cell identity and subcarrier spacing.

```
ue = struct('NNCellID',0,'NBULSubcarrierSpacing','15kHz');
```

Specify the NPRACH format, periodicity, subcarrier offset, number of subcarriers, and number of preamble repetitions.

```
chs = struct('NPRACHFormat','0','Periodicity',80, ...
    'SubcarrierOffset',0,'NumSubcarriers',12,'NRep',1);
```

Generate the NPRACH waveform, NPRACH information, and resource grid.

```
[waveform,info,grid] = lteNPRACH(ue,chs);
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific settings, specified as a structure containing these fields.

| Field | Required or Optional | Values | Description | Data Type |
|---|---|---|---|---|
| **`NNCellID`** | Required | Integer in the interval [0, 503] | Narrowband physical layer cell identity. | `double` |

| Field | Required or Optional | Values | Description | Data Type |
|---|---|---|---|---|
| **NBULSubcarrierSpacing** | Required | `'3.75kHz'`, `'15kHz'` | Narrowband internet of things (NB-IoT) uplink subcarrier spacing. To set a subcarrier spacing of 3.75 kHz, specify this field as `'3.75kHz'`. To set a subcarrier spacing of 15 kHz, specify this field as `'15kHz'`. | `char`, `string` |

| Field | Required or Optional | Values | Description | Data Type |
|---|---|---|---|---|
| **`Windowing`** | Optional | Nonnegative integer, default value depends on NPRACH preamble format in accordance with section F.5.F of [1] | Number of time-domain samples over which the function applies windowing and overlapping of OFDM samples.<br><br>If you do not specify this field, this function returns the `Windowing` field of the `info` output as a function of the `NBULSubcarrierSpacing` field. This behavior compromises between the effective duration of the cyclic prefix (and therefore the channel delay spread tolerance) and the spectral characteristics of the transmitted signal (not considering any additional FIR filtering). For NPRACH preamble format 2, the function sets the default value such that the ratio of cyclic prefix length to windowing length is the same as that of NPRACH preamble format 1. | `double` |

Data Types: `struct`

### `chs` — Channel transmission configuration

structure

Channel transmission configuration, specified as a structure containing these fields.

| Field | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **NPRACHFormat** | Required | '0', '1', '2' | NPRACH preamble format | Not applicable | char, string |
| **Periodicity** | Required | 40, 80, 160, 320, 640, 1280, 2560, 5120 | NPRACH resource periodicity, in milliseconds | If you set the NPRACHFormat field to '0' or '1', then you cannot set this field to 5120. | double |
| **SubcarrierOffset** | Required | 0, 2, 6, 12, 18, 24, 34, 36, 42, 48, 54, 60, 72, 78, 84, 90, 102, 108 | Frequency location of the first subcarrier allocated to NPRACH | If you set the NPRACHFormat field to '0' or '1', then you cannot set this field to 6, 42, 48, 54, 60, 72, 78, 84, 90, 102, or 108.<br><br>If you set the NPRACHFormat field to '2', then you cannot set this field to 2 or 34. | double |
| **NumSubcarriers** | Required | 12, 24, 36, 48, 72, 108, 144 | Number of subcarriers allocated to NPRACH | If you set the NPRACHFormat field to '0' or '1', then you cannot set this field to 72, 108, or 144.<br><br>If you set the NPRACHFormat field to '2', then you cannot set this field to 12, 24, or 48. | double |
| **NRep** | Required | 1, 2, 4, 8, 16, 32, 64, 128 | Number of NPRACH repetitions | Not applicable | double |
| **StartTime** | Optional | 8 (default), 16, 32, 64, 128, 256, 512, 1024 | NPRACH starting time, in milliseconds | Not applicable | double |

| Field | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **`NInit`** | Optional | `0` (default), integer in the interval [0, `NumSubcarriers` – 1] | Initial subcarrier for NPRACH | Not applicable | `double` |
| **`NPRACHPower`** | Optional | `0` (default), real-valued scalar | NPRACH power scaling, in decibels | Not applicable | `double` |

Data Types: `struct`

## Output Arguments

### `waveform` — Time-domain NPRACH FDD waveform

complex-valued column vector

Time-domain NPRACH FDD waveform, returned as a complex-valued column vector of length (`chs.Periodicity` × `info.SamplingRate` ÷ 1000). The waveform consists of:

1. A period of zeros corresponding to the time, `chs.StartTime`, between the start of the transmission and the first frame occupied by an NPRACH symbol.
2. The NPRACH transmission, defined in section 10.1.6.1 of [2] as a repetition of `chs.NRep` NPRACH preambles. An NPRACH preamble is a set of `info.P` symbol groups. A symbol group is a sequence of `info.N` identical symbols preceded by a cyclic prefix. The duration of an NPRACH preamble is a function of the preamble format as described in Table 10.1.6.1-1 of [2].
3. A period of zeros corresponding to the time between the end of the transmission and the value of the `chs.Periodicity` input.

The function samples the waveform at the same sampling rate as for a narrowband physical uplink shared channel (NPUSCH) waveform, defined by the `NBULSubcarrierSpacing` field of the `ue` input.

For preamble formats 0 and 1, the function adds an additional 40 ms gap every 64 preambles. For preamble format 2, the function adds an additional 40 ms gap every 16 preambles.

For more information about NPRACH waveform generation, see the “NB-IoT PRACH Waveform Generation” example.

Data Types: `double`
Complex Number Support: Yes

### `info` — NPRACH resource information

structure

NPRACH resource information, returned as a structure containing these fields.

| Field | Values | Description | Data Type |
|---|---|---|---|
| **`Nfft`** | Positive integer | Number of fast Fourier transform (FFT) points. | `double` |

| Field | Values | Description | Data Type |
|---|---|---|---|
| **SamplingRate** | Positive scalar | Sampling rate, in Hz, of the time-domain waveform. | double |
| **Windowing** | Nonnegative integer | Number of time-domain samples over which the function applies windowing and overlapping of OFDM symbols. | double |
| **FrequencyLocation** | Row vector of nonnegative integers | Frequency location for all symbol groups in an NPRACH transmission. For more information about NPRACH symbol groups, see the “NB-IoT PRACH Waveform Generation” example. The *k*th element of this output represents the frequency location of the *k*th symbol group. | double |
| **K** | Positive integer | Ratio of uplink data to NPRACH subcarrier spacing. | double |
| **NULSC** | Positive integer | Number of subcarriers for the specified uplink bandwidth. | double |
| **PreambleParameters** | Structure | Random access preamble parameters for the specified preamble format and frame structure type 1, as specified in Table 10.1.6.1-1 of [2]. | struct |

**PreambleParameters Field**

The `PreambleParameters` field of this output contains these fields.

| Field | Values | Description | Data Type |
|---|---|---|---|
| **G** | Positive integer | Number of time-contiguous symbol groups. | double |
| **P** | Positive integer | Total number of symbol groups. | double |
| **N** | Positive integer | Number of symbols in a symbol group. | double |

| Field | Values | Description | Data Type |
|---|---|---|---|
| **`T_CP`** | Positive integer | Cyclic prefix length, in multiples of $T_s$, where $T_s$ is the basic time unit, defined in section 4 of [2] as 1 ÷ (15000 × 2048) seconds. | `double` |
| **`T_SEQ`** | Positive integer | Length of symbols in a symbol group, in multiples of $T_s$. | `double` |

Data Types: `struct`

**`resourceGrid` — NPRACH resource grid**
real-valued matrix

NPRACH resource grid, returned as a real-valued matrix of size *K*-by-*L*, where:

- *K* is the number of subcarriers.
- *L* is the number of NPRACH symbols that fit the value of `chs.Periodicity`.

Data Types: `double`

# Version History

**Introduced in R2021a**

## References

[1] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. https://www.3gpp.org.

[2] 3GPP TS 36.211. “Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. https://www.3gpp.org.

## See Also

**Functions**
`lteNPRACHDetect` | `lteNPRACHInfo`

**Topics**
“NB-IoT PRACH Waveform Generation”

# lteNPRACHDetect

Detect NPRACH transmission

## Syntax

```
[indout,offset,detinfo] = lteNPRACHDetect(ue,chs,waveform)
[indout,offset,detinfo] = lteNPRACHDetect(___,threshold)
```

## Description

`[indout,offset,detinfo] = lteNPRACHDetect(ue,chs,waveform)` detects a narrowband physical random access channel (NPRACH) transmission in the time-domain waveform `waveform` for UE-specific settings `ue` and channel transmission configuration `chs`. The function returns `indout`, the detected value of the initial subcarrier used to generate the NPRACH transmission, `offset`, the timing offset, and `detinfo`, the detection information.

`[indout,offset,detinfo] = lteNPRACHDetect(___,threshold)` optionally specifies the detection `threshold` in addition to the previous syntax.

## Examples

### Detect NPRACH Preamble with Timing Offset

Create the UE-specific settings structure, setting the narrowband cell identity to `0` and the NB-IoT uplink subcarrier spacing to 15 kHz.

```
ue.NNCellID = 0;
ue.NBULSubcarrierSpacing = '15kHz';
```

Create the channel transmission configuration structure, setting the value of `NInit` to `32`.

```
chs.NPRACHFormat = '0';
chs.Periodicity = 80;
chs.SubcarrierOffset = 0;
chs.NumSubcarriers = 48;
chs.NRep = 1;
chs.NInit = 32;
```

Generate a time-domain NPRACH transmit waveform.

```
tx = lteNPRACH(ue,chs);
```

Create a received waveform by introducing a delay of seven samples in the transmit waveform.

```
rx = [zeros(7,1);tx];
```

Detect NPRACH instances in the received waveform. Confirm that the delay introduced in the waveform offsets the timing of the NPRACH detection.

```
[index,offset] = lteNPRACHDetect(ue,chs,rx)
```

```
index = 32

offset = 7.0196
```

Detect NPRACH instances in the waveform again, setting the detection threshold to 1. Both outputs are empty because no detection peak exceeds the threshold.

```
threshold = 1;
[index,offset] = lteNPRACHDetect(ue,chs,rx,threshold)

index =

     []


offset =

     []
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific settings, specified as a structure containing these fields.

| Field | Required or Optional | Values | Description | Data Type |
|---|---|---|---|---|
| **NNCellID** | Required | Integer in the interval [0, 503] | Narrowband physical layer cell identity. | `double` |
| **NBULSubcarrierSpacing** | Required | `'3.75kHz'`, `'15kHz'` | Narrowband internet of things (NB-IoT) uplink subcarrier spacing. To set a subcarrier spacing of 3.75 kHz, specify this field as `'3.75kHz'`. To set a subcarrier spacing of 15 kHz, specify this field as `'15kHz'`. | `char`, `string` |

Data Types: `struct`

### chs — Channel transmission configuration

structure

Channel transmission configuration, specified as a structure containing these fields.

| Field | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **NPRACHFormat** | Required | '0', '1', '2' | NPRACH preamble format | Not applicable | char, string |
| **Periodicity** | Required | 40, 80, 160, 320, 640, 1280, 2560, 5120 | NPRACH resource periodicity, in milliseconds | If you set the NPRACHFormat field to '0' or '1', then you cannot set this field to 5120. | double |
| **SubcarrierOffset** | Required | 0, 2, 6, 12, 18, 24, 34, 36, 42, 48, 54, 60, 72, 78, 84, 90, 102, 108 | Frequency location of the first subcarrier allocated to NPRACH | If you set the NPRACHFormat field to '0' or '1', then you cannot set this field to 6, 42, 48, 54, 60, 72, 78, 84, 90, 102, or 108.<br><br>If you set the NPRACH field to '2', then you cannot set this field to 2 or 34. | double |
| **NumSubcarriers** | Required | 12, 24, 36, 48, 72, 108, 144 | Number of subcarriers allocated to NPRACH | If you set the NPRACHFormat field to '0' or '1', then you cannot set this field to 72, 108, or 144.<br><br>If you set the NPRACHFormat field to '2', then you cannot set this field to 12, 24, or 48. | double |
| **NRep** | Required | 1, 2, 4, 8, 16, 32, 64, 128 | Number of NPRACH repetitions | Not applicable | double |
| **StartTime** | Optional | 8 (default), 16, 32, 64, 128, 256, 512, 1024 | NPRACH starting time, in milliseconds | Not applicable | double |

Data Types: struct

**`waveform` — Received signal potentially containing NPRACH transmission**
complex matrix

Received signal potentially containing NPRACH transmission, specified as an *N*-by-*P* complex matrix. This matrix contains the received time-domain signal in which the function searches for NPRACH transmissions. *N* is the number of time-domain samples. *P* is the number of receive antennas.

If *N* is less than the minimum number of samples that the function needs to analyze the configuration, the function appends zeros at the end of the waveform. If the waveform contains multiple NPRACH instances, the function returns the values of `indout` and `offset` that correspond to the NPRACH instance with the highest peak of the spectrum.

Data Types: `double`
Complex Number Support: Yes

**`threshold` — Detection threshold**
real number in the range [0, 1] | `[]`

Detection threshold, specified as a real number in the range [0, 1]. If you do not specify this input, the function chooses a default value based on the `NPRACHFormat` and `NRep` fields of the `chs` input.

The default value of the detection threshold has been empirically determined based on the probability of a false alarm and the probability of detection for the conformance tests discussed in Section 8.5.3 of [2]. This table provides the formulas that the function uses to calculate default detection thresholds for different NPRACH formats.

| NPRACH Format | Default Detection Threshold |
|---|---|
| **`0` or `1`** | $0.025 - 0.003\log_2(N_{\mathrm{Rep}})$ |
| **`2`** | $0.017 - 0.002\log_2(N_{\mathrm{Rep}})$ |

Data Types: `double`

## Output Arguments

**`indout` — Detected NPRACH initial subcarrier**
nonnegative integer | `[]`

Detected NPRACH initial subcarrier, returned as one of these values.

- Nonnegative integer — The index of the initial NPRACH subcarrier that corresponds to the highest detection peak across all valid values of `NInit`.
- `[]` — No detection peak exists above the detection threshold.

Data Types: `double`

**`offset` — Timing offset of NPRACH waveform in number of samples**
real number | `[]`

Timing offset of the NPRACH waveform, in number of samples, from the origin of the input waveform, returned as one of these values.

- Real number — The position, in number of samples, of the highest detection peak. The sampling rate is determined by the `ue` and `chs` arguments.

The function estimates this position from the discrete set of initial subcarrier values using a quadratic interpolation. See details in [3].

- `[]` — No detection peak exists above the detection threshold.

Data Types: `double`

**`detinfo` — Detection information**
structure

Detection information, returned as a structure containing these fields.

| Field | Description |
|---|---|
| **`DetectionPeaks`** | Highest peaks of the spectrum used for the detection, returned as a vector. The length of the vector is equal to the number of subcarriers. Each entry of the vector corresponds to a value of `NInit` in ascending order. |
| **`DetectionThreshold`** | Threshold that the function uses for the preamble detection. |

Data Types: `struct`

## Algorithms

The function implements the "Differential processing with minimum combinations" algorithm described in [3]. This algorithm matches the received waveform with a reference frequency-hopping pattern. The function generates a reference pattern for all values of `NInit` that are allowed for the current NPRACH configuration. For each of these values, the function follows these steps.

1. Generate a vector *v* that contains the magnitude of the frequency hopping of the received waveform. The position of the received frequency hopping in *v* depends on the reference frequency hopping.
2. Generate the spectrum of *v* and store the value of its highest peak.

   The output `indout` is the initial subcarrier index that corresponds to the highest peak of the spectrum among all those that exceed a defined threshold. The output `offset` is the position, in number of samples, of this peak.

# Version History

**Introduced in R2023a**

## References

[1] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. https://www.3gpp.org.

[2] 3GPP TS 36.141. “Base Station (BS) conformance testing.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. https://www.3gpp.org.

[3] Chougrani et al. “Efficient Preamble Detection and Time-of-Arrival Estimation for Single-Tone Frequency Hopping Random Access in NB-IoT.” *IEEE Internet of Things Journal* vol. 8, no. 9 (May 2021): 7437–7449.

## See Also

**Functions**
`lteNPRACH` | `lteNPRACHInfo`

**Topics**
“NB-IoT PRACH Detection and False Alarm Conformance Test”

# lteNPRACHInfo

NPRACH resource information

## Syntax

```
info = lteNPRACHInfo(ue,chs)
```

## Description

`info = lteNPRACHInfo(ue,chs)` returns narrowband physical random access channel (NPRACH) information for user equipment (UE) settings `ue` and channel transmission configuration `chs`.

## Examples

### Get NPRACH Information

Get NPRACH information for specified UE settings and channel transmission configuration.

Specify the narrowband physical layer cell identity and subcarrier spacing.

```
ue = struct('NNCellID',0,'NBULSubcarrierSpacing','15kHz');
```

Specify the NPRACH format, periodicity, subcarrier offset, number of subcarriers, and number of preamble repetitions.

```
chs = struct('NPRACHFormat','0','Periodicity',80, ...
    'SubcarrierOffset',0,'NumSubcarriers',12,'NRep',1);
```

Generate and display the NPRACH information.

```
info = lteNPRACHInfo(ue,chs)
```

```
info = struct with fields:
     SubcarrierSpacing: 3750
                  Nfft: 512
          SamplingRate: 1920000
             Windowing: 18
     FrequencyLocation: [0 1 7 6]
                     K: 4
                 NULSC: 12
    PreambleParameters: [1x1 struct]
```

Display the random access preamble parameters, as specified in Table 10.1.6.1-1 [2].

```
disp(info.PreambleParameters)
```

```
        G: 4
        P: 4
        N: 5
     T_CP: 2048
    T_SEQ: 40960
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific settings, specified as a structure containing these fields.

| Field | Required or Optional | Values | Description | Data Type |
|---|---|---|---|---|
| **`NNCellID`** | Required | Integer in the interval [0, 503] | Narrowband physical layer cell identity. | `double` |
| **`NBULSubcarrierSpacing`** | Required | `'3.75kHz'`, `'15kHz'` | Narrowband internet of things (NB-IoT) uplink subcarrier spacing. To set a subcarrier spacing of 3.75 kHz, specify this field as `'3.75kHz'`. To set a subcarrier spacing of 15 kHz, specify this field as `'15kHz'`. | `char`, `string` |

| Field | Required or Optional | Values | Description | Data Type |
|---|---|---|---|---|
| **`Windowing`** | Optional | Nonnegative integer, default value depends on NPRACH preamble format in accordance with section F.5.F of [1] | Number of time-domain samples over which the function applies windowing and overlapping of OFDM samples.<br><br>If you do not specify this field, this function returns the `Windowing` field of the `info` output as a function of the `NBULSubcarrierSpacing` field. This behavior compromises between the effective duration of the cyclic prefix (and therefore the channel delay spread tolerance) and the spectral characteristics of the transmitted signal (not considering any additional FIR filtering). For NPRACH preamble format 2, the function sets the default value such that the ratio of cyclic prefix length to windowing length is the same as that of NPRACH preamble format 1. | `double` |

Data Types: `struct`

**`chs` — Channel transmission configuration**
structure

Channel transmission configuration, specified as a structure containing these fields.

| Field | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **NPRACHFormat** | Required | '0', '1', '2' | NPRACH preamble format. | Not applicable | char, string |
| **Periodicity** | Required | 40, 80, 160, 320, 640, 1280, 2560, 5120 | NPRACH resource periodicity, in milliseconds. | If you set the NPRACHFormat field to '0' or '1', then you cannot set this field to 5120. | double |
| **SubcarrierOffset** | Required | 0, 2, 6, 12, 18, 24, 34, 36, 42, 48, 54, 60, 72, 78, 84, 90, 102, 108 | Frequency location of the first subcarrier allocated to NPRACH. | If you set the NPRACHFormat field to '0' or '1', then you cannot set this field to 6, 42, 48, 54, 60, 72, 78, 84, 90, 102, or 108.<br><br>If you set the NPRACH field to '2', then you cannot set this field to 2 or 34. | double |
| **NumSubcarriers** | Required | 12, 24, 36, 48, 72, 108, 144 | Number of subcarriers allocated to NPRACH. | If you set the NPRACHFormat field to '0' or '1', then you cannot set this field to 72, 108, or 144.<br><br>If you set the NPRACHFormat field to '2', then you cannot set this field to 12, 24, or 48. | double |
| **NRep** | Required | 1, 2, 4, 8, 16, 32, 64, 128 | Number of NPRACH repetitions. | Not applicable | double |
| **StartTime** | Optional | 8 (default), 16, 32, 64, 128, 256, 512, 1024 | NPRACH starting time, in milliseconds. | Not applicable | double |

| Field | Required or Optional | Values | Description | Dependencies | Data Types |
|---|---|---|---|---|---|
| **NInit** | Optional | 0 (default), integer in the interval [0, NumSubcarriers – 1] | Initial subcarrier for NPRACH. | Not applicable | double |

Data Types: struct

## Output Arguments

### info — NPRACH resource information

structure

NPRACH resource information, returned as a structure containing these fields.

| Field | Values | Description | Data Type |
|---|---|---|---|
| **Nfft** | Positive integer | Number of fast Fourier transform (FFT) points. | double |
| **SamplingRate** | Positive scalar | Sampling rate, in Hz, of the time-domain waveform. | double |
| **Windowing** | Nonnegative even integer | Number of time-domain samples over which the function applies windowing and overlapping of OFDM symbols. | double |
| **FrequencyLocation** | Row vector of nonnegative integers | Frequency location for all symbol groups in an NPRACH transmission. For more information about NPRACH symbol groups, see the “NB-IoT PRACH Waveform Generation” example. The *k*th element of this output represents the frequency location of the *k*th symbol group. | double |
| **K** | Positive integer | Ratio of uplink data to NPRACH subcarrier spacing. | double |
| **NULSC** | Positive integer | Number of subcarriers for the specified uplink bandwidth. | double |

| Field | Values | Description | Data Type |
|---|---|---|---|
| **PreambleParameters** | Structure | Random access preamble parameters for the specified preamble format and frame structure type 1, as specified in Table 10.1.6.1-1 of [2]. | `struct` |

**PreambleParameters Field**

The `PreambleParameters` field of this output contains these fields.

| Field | Values | Description | Data Type |
|---|---|---|---|
| **G** | Positive integer | Number of time-contiguous symbol groups. | `double` |
| **P** | Positive integer | Total number of symbol groups. | `double` |
| **N** | Positive integer | Number of symbols in a symbol group. | `double` |
| **T_CP** | Positive integer | Cyclic prefix length, in multiples of $T_s$, where $T_s$ is the basic time unit, defined in section 4 of [2] as 1 ÷ (15000 × 2048) seconds. | `double` |
| **T_SEQ** | Positive integer | Length of symbols in a symbol group, in multiples of $T_s$. | `double` |

Data Types: `struct`

# Version History

**Introduced in R2021a**

## References

[1] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. https://www.3gpp.org.

[2] 3GPP TS 36.211. “Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. https://www.3gpp.org.

## See Also

**Functions**
`lteNPRACHDetect` | `lteNPRACH`

**Topics**
“NB-IoT PRACH Waveform Generation”

# lteNPSS

Generate NPSS symbols for subframe

## Syntax

```
sym = lteNPSS(enb)
```

## Description

`sym = lteNPSS(enb)` generates `sym`, the narrowband primary synchronization signal (NPSS) symbols in a subframe for cell-wide settings `enb`.

## Examples

### Generate NPSS Symbols

Initialize cell-wide settings by specifying a narrowband operation mode and subframe number.

```
enb.OperationMode = 'Standalone';    % Narrowband operation mode
enb.NSubframe = 5;                   % Subframe number
```

Generate the NPSS symbols.

```
sym = lteNPSS(enb);
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure containing these fields.

| Name | Required or Optional | Values | Description | Data Types |
|---|---|---|---|---|
| **`OperationMode`** | Optional | `'Standalone'` (default), `'Inband-SamePCI'`, `'Inband-DifferentPCI'`, `'Guardband'` | NB-IoT operation mode, specified as one of these values:<br>• `'Standalone'` – NB-IoT standalone operation within any 180-kHz band outside any LTE carrier bandwidth<br>• `'Inband-SamePCI'` – NB-IoT in-band operation with the same physical layer cell identity (PCI) as an LTE carrier<br>• `'Inband-DifferentPCI'` – NB-IoT in-band operation with a different PCI to an LTE carrier<br>• `'Guardband'` – NB-IoT guard-band operation utilizing unused resource blocks within the guard-band of an LTE carrier | `char`, `string` |

| Name | Required or Optional | Values | Description | Data Types |
|---|---|---|---|---|
| **`NSubframe`** | Optional | `5` (default), integer | Subframe number. Because the NPSS is defined only for subframe 5, the function returns an empty array for any value of this field other than `5`. This behavior enables resource grid indexing for any subframe number. | `double` |
| **`NCellID`** | Required when you specify `OperationMode` as `'Inband-SamePCI'` or `'Inband-DifferentPCI'` | Integer in the interval [0, 503] | PCI | `double` |
| **`CellRefP`** | Required when you specify `OperationMode` as `'Inband-SamePCI'` or `'Inband-DifferentPCI'` | `1`, `2`, `4` | Number of cell-specific antenna ports | `double` |

**Note** To exclude cell reference signal (RS) locations, specify the `NCellID` and `CellRefP` fields. If you do not specify the `NCellID` and `CellRefP` fields, the function assumes that the cell RS is absent and generates NPSS values for all cell RS locations.

Data Types: `struct`

## Output Arguments

### `sym` — NPSS symbols for a subframe

complex-valued column vector | empty array

NPSS symbols for a subframe, returned as a complex-valued column vector. If you specify the `NSubframe` field of the `enb` input as any value other than `5`, then the function returns this output as an empty array.

Data Types: `double`

## Version History

**Introduced in R2019a**

## See Also

**Functions**

`lteNBDLFrameOffset` | `lteNPSSIndices` | `lteNSSS`

**Topics**

“Resource Grid Indexing”

# lteNPSSIndices

Generate NPSS RE indices for subframe

## Syntax

```
ind = lteNPSSIndices(enb)
ind = lteNPSSIndices(enb,port)
ind = lteNPSSIndices(enb,port,opts)
```

## Description

`ind = lteNPSSIndices(enb)` generates `ind`, the narrowband primary synchronization signal (NPSS) resource element (RE) indices for cell-wide settings `enb`.

`ind = lteNPSSIndices(enb,port)` generates NPSS RE indices for the antenna port corresponding to the `port` input.

`ind = lteNPSSIndices(enb,port,opts)` generates NPSS RE indices for the specified antenna port in the format specified by `opts`.

## Examples

### Generate Zero-Based NPSS RE Indices

Generate zero-based NPSS RE indices for antenna port 2000.

Initialize cell-wide settings by specifying the operation mode, number of cell-specific RS antenna ports, physical layer cell identity, and subframe number.

```
enb.OperationMode = 'Inband-SamePCI';    % Operation mode
enb.CellRefP = 1;                        % Number of cell-specific RS antenna ports
enb.NCellID = 2;                         % Physical layer cell identity
enb.NSubframe = 5;                       % Subframe number
```

Specify the antenna port and generate the zero-based NPSS RE indices.

```
port = 0;
ind = lteNPSSIndices(enb,port,'0based');
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure containing these fields.

| Name | Required or Optional | Values | Description | Data Types |
|---|---|---|---|---|
| **`OperationMode`** | Optional | `'Standalone'` (default), `'Inband-SamePCI'`, `'Inband-DifferentPCI'`, `'Guardband'` | NB-IoT operation mode, specified as one of these values:<br>• `'Standalone'` – NB-IoT standalone operation within any 180-kHz band outside any LTE carrier bandwidth<br>• `'Inband-SamePCI'` – NB-IoT in-band operation with the same physical layer cell identity (PCI) as an LTE carrier<br>• `'Inband-DifferentPCI'` – NB-IoT in-band operation with a different PCI to an LTE carrier<br>• `'Guardband'` – NB-IoT guard-band operation utilizing unused resource blocks within the guard-band of an LTE carrier | `char`, `string` |

| Name | Required or Optional | Values | Description | Data Types |
|---|---|---|---|---|
| **NSubframe** | Optional | 5 (default), integer | Subframe number. Because the NPSS is defined only for subframe 5, the function returns an empty array for any value of this field other than 5. This behavior enables resource grid indexing for any subframe number. | double |
| **NCellID** | Required when you specify OperationMode as 'Inband-SamePCI' or 'Inband-DifferentPCI' | Integer in the interval [0, 503] | PCI | double |
| **CellRefP** | Required when you specify OperationMode as 'Inband-SamePCI' or 'Inband-DifferentPCI' | 1, 2, 4 | Number of cell-specific antenna ports | double |

**Note** To exclude cell reference signal (RS) locations, specify the NCellID and CellRefP fields. If you do not specify the NCellID and CellRefP fields, the function assumes that the cell RS is absent and generates NPSS values for all cell RS locations.

Data Types: struct

### port — Antenna port

0 | 1

Antenna port, specified as 0 or 1, corresponding to antenna port 2000 or 2001, respectively.

Data Types: double

### opts — Output format and index base of generated indices

character vector | string scalar | cell array of character vectors | string array

Output format and index base of generated indices, specified as one of these forms.

- 'format base'
- "format base"

- `{'format','base'}`
- `["format","base"]`

Where `format` and `base` are defined in this table.

| Option | Values | Description |
|---|---|---|
| **`format`** | `'ind'` (default), `'sub'` | Output format of generated indices<br><br>To return the indices as a column vector, specify this option as `'ind'`.<br><br>To return the indices as an $N_{RE}$-by-3 matrix, where $N_{RE}$ is the number of REs, specify this option as `'sub'`. Each row of the matrix contains the subcarrier, symbol, and antenna port as its first, second, and third element, respectively. |
| **`base`** | `'1based'` (default), `'0based'` | Index base<br><br>To generate indices whose first value is `1`, specify this option as `'1based'`. To generate indices whose first value is `0`, specify this option as `'0based'`. |

Example: `'ind 0based'`, `"ind 0based"`, `{'ind','0based'}`, and `["ind","0based"]` specify the same output options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `ind` — NPSS RE indices for a subframe

complex-valued array | empty array

NPSS RE indices for a subframe, returned as a complex-valued array. The array dimensions depend on the format options you specify in `opts`. To return `ind` as a column vector, specify `'ind'` in the `opts` input. To return `ind` as an $N_{RE}$-by-3 matrix, specify `'sub'` in the `opts` input. If you specify the `NSubframe` field of the `enb` input as a value other than `5`, the function returns this output as an empty array.

Data Types: `uint32`

# Version History

**Introduced in R2019a**

## See Also

### Functions

`lteNPSS` | `lteNSSSIndices`

**Topics**
“Resource Grid Indexing”

# lteNPUSCH

Generate NPUSCH symbols

## Syntax

```
[sym,stateOut] = lteNPUSCH(ue,chs,cw)
[sym,stateOut] = lteNPUSCH(ue,chs,cw,stateIn)
```

## Description

`[sym,stateOut] = lteNPUSCH(ue,chs,cw)` generates `sym`, a column vector containing the narrowband physical uplink shared channel (NPUSCH) symbols for a time slot. The function generates symbols by applying NPUSCH encoding to codeword `cw` for the specified user equipment (UE) settings `ue` and channel transmission configuration `chs`. Channel encoding comprises scrambling, modulation, layer mapping onto a single layer, transform precoding, and precoding in accordance with section 10.1.3 of [1]. The function also returns `stateOut`, a structure containing the encoder state for bundle transmission.

`[sym,stateOut] = lteNPUSCH(ue,chs,cw,stateIn)` specifies `stateIn`, the initial encoder state.

## Examples

### Generate NPUSCH Symbols for Bundle

Generate NPUSCH symbols on a slot-by-slot basis for transmitting a bundle comprising one resource unit, 16 slots, and two codeword repetitions.

Configure UE-specific settings.

```
ue = struct('NNCellID',0,'NBULSubcarrierSpacing','3.75kHz');
```

Specify a channel transmission configuration.

```
chs = struct('NPUSCHFormat','Data','NRUsc',1,'NRep',2,'NBULSubcarrierSet',17, ...
    'NRU',1,'NULSlots',16,'Modulation','BPSK','RNTI',0);
```

Generate a codeword of bits.

```
[~,info] = lteNPUSCHIndices(ue,chs);
cwLen = info.G;
cw = ones(cwLen,1);
```

Specify the initial encoder state as an empty structure, indicating the start of a bundle.

```
stateIn = struct();
```

Generate NPUSCH symbols for bundle transmission.

```
for SlotIdx = 0:(chs.NRep*chs.NRU*chs.NULSlots-1)
    ue.NSlot = SlotIdx;
```

```
    [sym,stateOut] = lteNPUSCH(ue,chs,cw,stateIn);
    stateIn = stateOut;
end
```

Confirm that the transmission reaches the end of the bundle.

```
disp(stateOut.EndOfTx)
```

```
   1
```

## Input Arguments

### `ue` — UE-specific settings

structure

UE-specific settings, specified as a structure containing these fields.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`NNCellID`** | Integer in the interval [0, 503] | Narrowband physical layer cell identity (PCI) | `double` |
| **`NBULSubcarrierSpacing`** | `'3.75kHz'`, `'15kHz'` | NB-IoT uplink subcarrier spacing<br><br>To set a subcarrier spacing of 3.75 kHz, specify this field as `'3.75kHz'`. To set a subcarrier spacing of 15 kHz, specify this field as `'15kHz'`. | `char`, `string` |
| **`NSlot`** | Nonnegative integer | Slot number | `double` |
| **`NFrame`** | `0` (default), nonnegative integer | Frame number | `double` |

Data Types: `struct`

### `chs` — Channel transmission configuration

structure

Channel transmission configuration, specified as a structure containing these fields.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **NPUSCHFormat** | `'Data'`, `'Control'` | NPUSCH format<br><br>To indicate that the NPUSCH carries narrowband uplink shared channel (UL-SCH) data, specify this field as `'Data'`. To indicate that the NPUSCH carries uplink control information, specify this field as `'Control'`. | `char`, `string` |
| **NRUsc** | 1, 3, 6, 12 | Number of consecutive subcarriers in a resource unit (RU)<br><br>If you specify the `NPUSCHFormat` field as `'Control'` or the `NBULSubcarrierSpacing` field of the `ue` input as `'3.75kHz'`, then you must specify this field as `1` . | `double` |
| **NRep** | 1, 2, 4, 8, 16, 32, 64, 128 | Number of repetitions for a codeword | `double` |
| **NRU** | 1, 2, 3, 4, 5, 6, 8, 10 | Number of RUs | `double` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **NULSlots** | 2, 4, 8, 16 | Number of slots per RU<br><br>If you specify the `NPUSCHFormat` field as `'Control'`, then you must specify this field as `4`.<br><br>If you specify the `NPUSCHFormat` field as `'Data'`, then you must specify this field as:<br>• `16` when you specify the `NRUsc` field as `1`<br>• `8` when you specify the `NRUsc` field as `3`<br>• `4` when you specify the `NRUsc` field as `6`<br>• `2` when you specify the `NRUsc` field as `12` | `double` |
| **Modulation** | `'BPSK'`, `'QPSK'` | Modulation type<br><br>To enable binary phase-shift keying (BPSK), specify this field as `'BPSK'`. To enable quadrature phase-shift keying (QPSK), specify this field as `'QPSK'`.<br><br>If you specify the `NPUSCHFormat` field as `'Control'`, then you must specify this field as `'BPSK'`. | `char`, `string` |
| **RNTI** | Nonnegative integer | Radio network temporary identifier (RNTI) value | `double` |

Data Types: `struct`

### `cw` — Codeword of bit values

binary-valued column vector

Codeword of bit values, specified as a binary-valued column vector. The length of this input must be an integer multiple of:

- `chs.NRU` x `chs.NULSlots` x `chs.NRUsc` when you specify the `Modulation` field of the `chs` input as `'BPSK'`
- 2 x `chs.NRU` x `chs.NULSlots` x `chs.NRUsc` when you specify the `Modulation` field of the `chs` input as `'QPSK'`

Data Types: `double`

**`stateIn` — Initial encoder state**
`struct()` (default) | structure

Initial encoder state for bundle transmission, specified as a structure containing these fields.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`SlotIdx`** | Integer in the interval [0, (`chs.NRU` x `chs.NULSlots` x `chs.NRep`) – 1] | Index of a slot within a bundle, in zero-based form | `double` |
| **`InitNSlot`** | Nonnegative integer | Slot number for scrambling sequence initialization | `double` |
| **`InitNFrame`** | Nonnegative integer | Frame number for scrambling sequence initialization | `double` |
| **`EndOfBlk`** | `1` or `true`, `0` or `false` | End of transport block indicator | `logical` |
| **`EndOfTx`** | `1` or `true`, `0` or `false` | End of bundle indicator | `logical` |
| **`GhpNSlot`** | Nonnegative integer | Slot number for the first slot in the RU | `double` |

Data Types: `struct`

## Output Arguments

**`sym` — NPUSCH symbols**
complex-valued column vector

NPUSCH symbols for a time slot, returned as a complex-valued column vector.

Data Types: `double`

**`stateOut` — Output encoder state**
structure

Output encoder state, returned as a structure. This output contains the internal state of each transport block in these fields.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`SlotIdx`** | Integer in the interval [0, (`chs.NRU` x `chs.NULSlots` x `chs.NRep`) – 1] | Index of a slot within a bundle, in zero-based form<br><br>The function returns this field as the `SlotIdx` field of the `stateIn` input increased by one. When the `SlotIdx` field of the `stateIn` input reaches its maximum value, the function returns this field as `0`. If you do not specify the `SlotIdx` field of the `stateIn` input, the function returns this field as `0`. A value of `0` indicates that the transmission has reached the end of a bundle, which the function also indicates by setting the `EndOfTx` field to `1`. | `double` |
| **`InitNSlot`** | Nonnegative integer | Slot number for scrambling sequence initialization<br><br>When the slot being processed is at the initialization point, this field is equal to the `NSlot` field of the `ue` input. Otherwise, the function returns this field as one of these values.<br><br>• The value of the `InitNSlot` field of the `stateIn` input<br>• The value of the `NSlot` field of the `ue` input when you do not specify the `InitNSlot` field of the `stateIn` input | `double` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`InitNFrame`** | Nonnegative integer | Frame number for scrambling sequence initialization<br><br>When the frame being processed is at the initialization point, this field is equal to the `NFrame` field of the `ue` input. Otherwise, the function returns this field as one of these values.<br><br>• The value of the `InitNFrame` field of the `stateIn` input<br>• `0` when you do not specify the `InitNFrame` field of the `stateIn` input | `double` |
| **`EndOfBlk`** | `1`, `0` | End of transport block indicator<br><br>When the transmission reaches the end of a transport block, the function returns this field as `1`.<br><br>At the beginning of a bundle, the function resets this field. | `logical` |
| **`EndOfTx`** | `1`, `0` | End of bundle indicator<br><br>When the transmission reaches the end of a bundle, the function returns this field as `1`. Otherwise, the function returns this field as `0`.<br><br>At the beginning of a bundle, the function resets this field. | `logical` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`GhpNSLot`** | Nonnegative integer | Slot number for the first slot in the RU<br><br>The function uses this field only when you specify the `NPUSCHFormat` field as `'Data'` and the `NRUsc` field as `1` in the `chs` input. | `double` |

Data Types: `struct`

## More About

### Bundle

A bundle in the medium access control (MAC) layer refers to the repeated transmissions of a transport block.

For more information, see Section 5.3.2.1 of [2].

## Tips

To use this function for transmission of a bundle, follow these steps.

1. Call the function, optionally specifying the initial encoder state using the `stateIn` input. The `stateOut` output represents the first transmission of the transport block.
2. Call the function again, specifying the `stateIn` input as the `stateOut` output returned by the previous call to the function.
3. Repeat step 2 until the function returns the `EndOfTx` field of the `stateOut` output as `1` (`true`), indicating the end of the bundle.

# Version History

**Introduced in R2020a**

## References

[1] 3GPP TS 36.211. “Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. https://www.3gpp.org.

[2] 3GPP TS 36.321. “Medium Access Control (MAC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. https://www.3gpp.org.

## See Also

**Functions**
`lteNPUSCHDecode` | `lteNPUSCHDRS` | `lteNPUSCHDRSIndices` | `lteNPUSCHIndices` | `lteNULSCH` | `lteNULSCHDecode` | `ltePUSCH`

# lteNPUSCHDecode

Decode NPUSCH symbols

## Syntax

```
[cw,stateOut,symbols] = lteNPUSCHDecode(ue,chs,sym)
[cw,stateOut,symbols] = lteNPUSCHDecode(ue,chs,sym,hEst,noiseEst)
[cw,stateOut,symbols] = lteNPUSCHDecode( ___ ,stateIn)
```

## Description

`[cw,stateOut,symbols] = lteNPUSCHDecode(ue,chs,sym)` recovers `cw`, a codeword of soft bits, by decoding `sym`, the narrowband uplink shared channel (NPUSCH) symbols for the specified user equipment (UE) settings `ue` and channel transmission configuration `chs`. The decoding process comprises inverting NPUSCH channel encoding, as described in section 10.1.3 of [1]. The function also returns `stateOut`, a structure containing the decoder state for bundle reception, and `symbols`, a vector of received constellation symbols.

`[cw,stateOut,symbols] = lteNPUSCHDecode(ue,chs,sym,hEst,noiseEst)` decodes the NPUSCH symbols for the specified channel estimate `hEst` and noise power spectral density estimate `noiseEst`.

`[cw,stateOut,symbols] = lteNPUSCHDecode( ___ ,stateIn)` specifies `stateIn`, the initial decoder state, in addition to any input argument combination from previous syntaxes.

## Examples

### Decode NPUSCH Symbols from Bundle

Generate NPUSCH symbols on a slot-by-slot basis for a bundle comprising eight slots for an input codeword. Recover the corresponding codeword of soft bits by decoding the NPUSCH symbols.

Configure UE-specific settings.

```
ue = struct('NNCellID',0,'NBULSubcarrierSpacing','15kHz');
```

Specify the channel transmission configuration.

```
chs = struct('NPUSCHFormat','Data','NRUsc',12,'NRep',4,'NRU',1,...
    'NULSlots',2,'Modulation','QPSK','RNTI',0);
```

Generate a codeword of bits.

```
cwLen = 2*20*chs.NRUsc*chs.NRU*chs.NULSlots;
cwIn = ones(cwLen,1);
```

Specify the initial encoder and decoder states as empty structures, indicating the start of a bundle.

```
encoderStateIn = struct();
stateIn = struct();
```

Generate NPUSCH symbols for bundle transmission and decode the bundle on a slot-by-slot basis.

```
for SlotIdx = 0:(chs.NRep*chs.NRU*chs.NULSlots - 1)
    ue.NSlot = SlotIdx;
    [sym,encoderStateOut] = lteNPUSCH(ue,chs,cwIn,encoderStateIn);
    encoderStateIn = encoderStateOut;
    [cw,stateOut,symbols] = lteNPUSCHDecode(ue,chs,sym,stateIn);
    stateIn = stateOut;
end
```

Confirm that the transmission and reception reach the end of the bundle and that the received codeword corresponds to the input codeword.

```
disp(encoderStateOut.EndOfTx)
```

```
   1
```

```
disp(stateOut.EndOfTx)
```

```
   1
```

```
disp(isequal(cwIn,cw>0))
```

```
   1
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific settings, specified as a structure containing these fields.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **NNCellID** | Integer in the interval [0, 503] | Narrowband physical layer cell identity (PCI) | `double` |
| **NBULSubcarrierSpacing** | `'3.75kHz'`, `'15kHz'` | NB-IoT uplink subcarrier spacing<br><br>To set a subcarrier spacing of 3.75 kHz, specify this field as `'3.75kHz'`. To set a subcarrier spacing of 15 kHz, specify this field as `'15kHz'`. | `char`, `string` |
| **NSlot** | Nonnegative integer | Slot number | `double` |
| **NFrame** | `0` (default), nonnegative integer | Frame number | `double` |

Data Types: `struct`

### chs — Channel transmission configuration

structure

Channel transmission configuration, specified as a structure containing these fields.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **NPUSCHFormat** | `'Data'`, `'Control'` | NPUSCH format<br><br>To indicate that the NPUSCH carries narrowband uplink shared channel (UL-SCH) data, specify this field as `'Data'`. To indicate that the NPUSCH carries uplink control information, specify this field as `'Control'`. | `char`, `string` |
| **NRUsc** | 1, 3, 6, 12 | Number of consecutive subcarriers in a resource unit (RU)<br><br>If you specify the `NPUSCHFormat` field as `'Control'` or the `NBULSubcarrierSpacing` field of the `ue` input as `'3.75kHz'`, then you must specify this field as `1` . | `double` |
| **NRep** | 1, 2, 4, 8, 16, 32, 64, 128 | Number of repetitions for a codeword | `double` |
| **NRU** | 1, 2, 3, 4, 5, 6, 8, 10 | Number of RUs | `double` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **NULSlots** | 2, 4, 8, 16 | Number of slots per RU<br><br>If you specify the `NPUSCHFormat` field as `'Control'`, then you must specify this field as `4`.<br><br>If you specify the `NPUSCHFormat` field as `'Data'`, then you must specify this field as:<br>• `16` when you specify the `NRUsc` field as `1`<br>• `8` when you specify the `NRUsc` field as `3`<br>• `4` when you specify the `NRUsc` field as `6`<br>• `2` when you specify the `NRUsc` field as `12` | `double` |
| **Modulation** | `'BPSK'`, `'QPSK'` | Modulation type<br><br>To enable binary phase-shift keying (BPSK), specify this field as `'BPSK'`. To enable quadrature phase-shift keying (QPSK), specify this field as `'QPSK'`.<br><br>If you specify the `NPUSCHFormat` field as `'Control'`, then you must specify this field as `'BPSK'`. | `char`, `string` |
| **RNTI** | Nonnegative integer | Radio network temporary identifier (RNTI) value | `double` |
| **CSI** | `'On'` (default), `'Off'` | Weight soft bits by channel state information (CSI)<br><br>To scale the `cw` output with CSI, specify this field as `'On'`. Otherwise, specify this field as `'Off'`. | `char`, `string` |

Data Types: `struct`

### `sym` — NPUSCH symbols

complex-valued matrix

NPUSCH symbols, specified as a complex-valued matrix of size $N_{Sym}$-by-$N_{RxAnts}$.

- $N_{Sym}$ is the number of symbols per receive antenna assigned to the NPUSCH for a slot.
- $N_{RxAnts}$ is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

### `hEst` — Channel estimate

complex-valued matrix

Channel estimate, specified as a complex-valued matrix of size $N_{Sym}$-by-$N_{RxAnts}$.

- $N_{Sym}$ is the number of symbols per receive antenna assigned to the NPUSCH for a slot.
- $N_{RxAnts}$ is the number of receive antennas.

Data Types: `double`

### `noiseEst` — Noise power spectral density estimate

real-valued scalar

Noise power spectral density estimate per resource element, specified as a real-valued scalar.

Data Types: `double`

### `stateIn` — Initial decoder state

`struct()` (default) | structure

Initial decoder state for bundle reception, specified as a structure containing these fields.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`SlotIdx`** | Integer in the interval [0, (`chs.NRU` x `chs.NULSlots` x `chs.NRep`) – 1] | Index of a slot within a bundle, in zero-based form | `double` |
| **`BlkFlushEnabled`** | `1` or `true`, `0` or `false` | To flush the buffers at every block, specify this field as `1` (`true`). Otherwise, specify this field as `0` (`false`). | `logical` |
| **`InitNSlot`** | Nonnegative integer | Slot number for scrambling sequence initialization | `double` |
| **`InitNFrame`** | Nonnegative integer | Frame number for scrambling sequence initialization | `double` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **EndOfBlk** | `1` or `true`, `0` or `false` | Transport block receipt indicator | `logical` |
| **EndOfCW** | `1` or `true`, `0` or `false` | Codeword receipt indicator | `logical` |
| **EndOfTx** | `1` or `true`, `0` or `false` | End of bundle indicator | `logical` |
| **CWBuffer** | Real-valued column vector of the same length as the `cw` output. | Buffer to store the recovered codeword | `double` |
| **CWSLCount** | (`chs.NRU` x `chs.NULSlots`)-by-1 vector of integers | Repetition counter | `double` |

Data Types: `struct`

## Output Arguments

### `cw` — Codeword of soft bits

real-valued column vector

Codeword of soft bits, returned as a real-valued column vector.

Data Types: `double`

### `stateOut` — Output decoder state

structure

Output decoder state, returned as a structure. This output contains the internal state of each transport block in these fields.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`SlotIdx`** | Integer in the interval [0, (`chs.NRU` x `chs.NULSlots` x `chs.NRep`) – 1] | Index of a slot within a bundle, in zero-based form<br><br>The function returns this field as the `SlotIdx` field of the `stateIn` input increased by one. When the `SlotIdx` field of the `stateIn` input reaches its maximum value, the function returns this field as `0`. If you do not specify the `SlotIdx` field of the `stateIn` input, the function returns this field as `0`. A value of `0` indicates that the transmission has reached the end of a bundle, which the function also indicates by setting the `EndOfTx` field to `1`. | `double` |
| **`BlkFlushEnabled`** | `1`, `0` | Indicator that the function flushes the buffers at every block | `logical` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`InitNSlot`** | Nonnegative integer | Slot number for scrambling sequence initialization<br><br>When the slot being processed is at the initialization point, this field is equal to the `NSlot` field of the `ue` input. Otherwise, the function returns this field as one of these values.<br><br>• The value of the `InitNSlot` field of the `stateIn` input<br>• The value of the `NSlot` field of the `ue` input when you do not specify the `InitNSlot` field of the `stateIn` input | `double` |
| **`InitNFrame`** | Nonnegative integer | Frame number for scrambling sequence initialization<br><br>When the frame being processed is at the initialization point, this field is equal to the `NFrame` field of the `ue` input. Otherwise, the function returns this field as one of these values.<br><br>• The value of the `InitNFrame` field of the `stateIn` input<br>• `0` when you do not specify the `InitNFrame` field of the `stateIn` input | `double` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`EndOfBlk`** | `1`, `0` | Transport block receipt indicator<br><br>When the function has received all slots of a transport block at least once, it returns this field as `1`.<br><br>At the beginning of a bundle, the function resets this field. | `logical` |
| **`EndOfCW`** | `1`, `0` | Codeword receipt indicator<br><br>When the function has received the entire codeword, that is, when each element of the `CWSLCount` field is as least `1`, the function returns this field as `1`.<br><br>At the beginning of a bundle, the function resets this field. | `logical` |
| **`EndOfTx`** | `1`, `0` | End of bundle indicator<br><br>When the transmission reaches the end of a bundle, that is, when each element of the `CWSLCount` field is as least `1`, the function returns this field as `1`. Otherwise, the function returns this field as `0`.<br><br>At the beginning of a bundle, the function resets this field. | `logical` |
| **`CWBuffer`** | Real-valued column vector of the same length as the `cw` output. | Buffer to store the recovered codeword<br><br>At the beginning of a bundle, the function resets this field. | `double` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| `CWSLCount` | (`chs.NRU` x `chs.NULSlots`)-by-1 vector of integers | Repetition counter<br><br>Each element indicates how many repetitions of the corresponding codeword portion the `CWBuffer` field has recovered.<br><br>At the beginning of a bundle, the function resets this field. | `double` |

Data Types: `struct`

**`symbols` — Received constellation symbols**
complex-valued column vector

Received constellation symbols, returned as a complex-valued column vector.

Data Types: `double`
Complex Number Support: Yes

## More About

### Bundle

A bundle in the medium access control (MAC) layer refers to the repeated transmissions of a transport block.

For more information, see Section 5.3.2.1 of [2].

## Tips

To use this function for reception of a bundle, follow these steps.

1. Call the function, optionally specifying the initial encoder state using the `stateIn` input. The `stateOut` output represents the first transmission of the transport block.
2. Call the function again, specifying the `stateIn` input as the `stateOut` output returned by the previous call to the function.
3. Repeat step 2 until the function returns the `EndOfTx` field of the `stateOut` output as `1` (`true`), indicating the end of the bundle.

# Version History

**Introduced in R2020a**

## References

[1] 3GPP TS 36.211. “Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. https://www.3gpp.org.

[2] 3GPP TS 36.321. “Medium Access Control (MAC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. https://www.3gpp.org.

## See Also

**Functions**

`lteNPUSCH` | `lteNPUSCHDRS` | `lteNPUSCHDRSIndices` | `lteNPUSCHIndices` | `lteNULSCH` | `lteNULSCHDecode` | `ltePUSCHDecode`

# lteNPUSCHDRS

Generate NPUSCH DRS symbols

## Syntax

```
[sym,stateOut] = lteNPUSCHDRS(ue,chs)
[sym,stateOut] = lteNPUSCHDRS(ue,chs,stateIn)
```

## Description

`[sym,stateOut] = lteNPUSCHDRS(ue,chs)` generates `sym`, a column vector containing the narrowband physical uplink shared channel (NPUSCH) demodulation reference signal (DRS) symbols. The function generates `sym` in accordance with section 10.1.4 of [1] for user equipment (UE) settings `ue` and channel transmission configuration `chs`. The function also returns `stateOut`, a structure containing the encoder state for bundle transmission.

`[sym,stateOut] = lteNPUSCHDRS(ue,chs,stateIn)` specifies `stateIn`, the initial encoder state.

## Examples

### Generate NPUSCH DRS Symbols

Generate NPUSCH DRS symbols for a single-tone data channel.

Configure UE-specific settings.

```
ue = struct('NNCellID',0,'NBULSubcarrierSpacing','15kHz');
```

Specify a channel transmission configuration.

```
chs = struct('NPUSCHFormat','Data','NRep',2,'NRU',1,'NRUsc',1, ...
    'NULSlots',16,'SeqGroupHopping','On','SeqGroup',0);
```

Specify the initial encoder state as an empty structure, indicating the start of a bundle.

```
stateIn  = struct();
```

Generate NPUSCH DRS symbols.

```
for SlotIdx = 0:(chs.NRep*chs.NRU*chs.NULSlots-1)
    ue.NSlot = SlotIdx;
    [sym,stateOut] = lteNPUSCHDRS(ue,chs,stateIn);
    stateIn = stateOut;
end
```

Confirming that the transmission reaches the end of the bundle.

```
disp(stateOut.EndOfTx)
```

```
   1
```

## Input Arguments

### `ue` — UE-specific settings

structure

UE-specific settings, specified as a structure containing these fields.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`NNCellID`** | Integer in the interval [0, 503] | Narrowband physical layer cell identity (PCI) | `double` |
| **`NFrame`** | `0` (default), nonnegative integer | Frame number | `double` |
| **`NBULSubcarrierSpacing`** | `'3.75kHz'`, `'15kHz'` | NB-IoT uplink subcarrier spacing<br><br>To set a subcarrier spacing of 3.75 kHz, specify this field as `'3.75kHz'`. To set a subcarrier spacing of 15 kHz, specify this field as `'15kHz'`. | `char`, `string` |
| **`NSlot`** | Nonnegative integer | Slot number | `double` |

Data Types: `struct`

### `chs` — Channel transmission configuration

structure

Channel transmission configuration, specified as a structure containing these fields.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`NPUSCHFormat`** | `'Data'`, `'Control'` | NPUSCH format<br><br>To indicate that the NPUSCH carries narrowband uplink shared channel (UL-SCH) data, specify this field as `'Data'`. To indicate that the NPUSCH carries uplink control information, specify this field as `'Control'`. | `char`, `string` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **NRUsc** | 1, 3, 6, 12 | Number of consecutive subcarriers in a resource unit (RU)<br><br>If you specify the `NPUSCHFormat` field as `'Control'` or the `NBULSubcarrierSpacing` field of the `ue` input as `'3.75kHz'`, then you must specify this field as `1` . | `double` |
| **NRep** | 1, 2, 4, 8, 16, 32, 64, 128 | Number of repetitions for a codeword | `double` |
| **NRU** | 1, 2, 3, 4, 5, 6, 8, 10 | Number of RUs | `double` |
| **NULSlots** | 2, 4, 8, 16 | Number of slots per RU<br><br>If you specify the `NPUSCHFormat` field as `'Control'`, then you must specify this field as `4`.<br><br>If you specify the `NPUSCHFormat` field as `'Data'`, then you must specify this field as:<br>• `16` when you specify the `NRUsc` field as `1`<br>• `8` when you specify the `NRUsc` field as `3`<br>• `4` when you specify the `NRUsc` field as `6`<br>• `2` when you specify the `NRUsc` field as `12` | `double` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`BaseSeqIdx`** | Integer in the interval [0, 29]<br><br>Default depends on the value of the `NRUsc` field | Multitone NPUSCH DRS base sequence index<br><br>• When you specify the `NRUsc` field as `3`, specify this field as an integer in the interval [0, 11]. If you do not specify this field, the function sets it to the value of `mod(ue.NNCellID, 12)`.<br>• When you specify the `NRUsc` field as `6`, specify this field as an integer in the interval [0, 13]. If you do not specify this field, the function sets it to the value of `mod(ue.NNCellID, 14)`.<br>• When you specify the `NRUsc` field as `12`, specify this field as an integer in the interval [0, 29]. If you do not specify this field, the function sets it to the value of `mod(ue.NNCellID, 30)`.<br>• When you specify the `NRUsc` field as any other value, the function does not use this field.<br><br>**Dependencies.** To enable this field, specify the `NRUsc` field as `3`, `6`, or `12`. | `double` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **SeqGroupHopping** | `'On'` (default), `'Off'` | To enable sequence-group hopping, specify this field as `'On'`. To disable sequence-group hopping, specify this field as `'Off'`. For more information, see section 5.5.1.3 of [1]. | `char`, `string` |
| **SeqGroup** | `0` (default), integer in the interval [0, 29] | Sequence-group assignment for sequence shift pattern calculation<br><br>For more information, see section 10.1.4.1.3 of [1].<br><br>**Dependencies.** To enable this field, specify the `SeqGroupHopping` field as `'On'`. | `double` |
| **CyclicShift** | `0` (default), integer in the interval [0, 3] | Cyclic shift<br>• When you specify the `NRUsc` field as `3`, specify this field as an integer in the interval [0, 2].<br>• When you specify the `NRUsc` field as `6`, specify this field as an integer in the interval [0, 3].<br><br>**Dependencies.** To enable this field, specify the `NRUsc` field as `3` or `6`. | `double` |

Data Types: `struct`

### `stateIn` — Initial encoder state

`struct()` (default) | structure

Initial encoder state for bundle transmission, specified as a structure containing these fields.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`SlotIdx`** | Integer in the interval [0, (`chs.NRU` x `chs.NULSlots` x `chs.NRep`) – 1] | Index of a slot within a bundle, in zero-based form | `double` |
| **`InitNSlot`** | Nonnegative integer | Slot number for scrambling sequence initialization | `double` |
| **`InitNFrame`** | Nonnegative integer | Frame number for scrambling sequence initialization | `double` |
| **`EndOfBlk`** | `1` or `true`, `0` or `false` | End of transport block indicator | `logical` |
| **`EndOfTx`** | `1` or `true`, `0` or `false` | End of bundle indicator | `logical` |
| **`GhpNSlot`** | Nonnegative integer | Slot number for the first slot in the RU | `double` |

Data Types: `struct`

## Output Arguments

### `sym` — NPUSCH DRS symbols

complex-valued column vector

NPUSCH DRS symbols, returned as a complex-valued column vector.

Data Types: `double`

### `stateOut` — Output encoder state

structure

Output encoder state, returned as a structure. This output contains the internal state of each transport block in these fields.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **SlotIdx** | Integer in the interval [0, (chs.NRU x chs.NULSlots x chs.NRep) – 1] | Index of a slot within a bundle, in zero-based form<br><br>The function returns this field as the SlotIdx field of the stateIn input increased by one. When the SlotIdx field of the stateIn input reaches its maximum value, the function returns this field as 0. If you do not specify the SlotIdx field of the stateIn input, the function returns this field as 0. A value of 0 indicates that the transmission has reached the end of a bundle, which the function also indicates by setting the EndOfTx field to 1. | double |
| **InitNSlot** | Nonnegative integer | Slot number for scrambling sequence initialization<br><br>When the slot being processed is at the initialization point, this field is equal to the NSlot field of the ue input. Otherwise, the function returns this field as one of these values.<br><br>• The value of the InitNSlot field of the stateIn input<br>• The value of the NSlot field of the ue input when you do not specify the InitNSlot field of the stateIn input | double |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`InitNFrame`** | Nonnegative integer | Frame number for scrambling sequence initialization<br><br>When the frame being processed is at the initialization point, this field is equal to the `NFrame` field of the `ue` input. Otherwise, the function returns this field as one of these values.<br><br>• The value of the `InitNFrame` field of the `stateIn` input<br>• `0` when you do not specify the `InitNFrame` field of the `stateIn` input | `double` |
| **`EndOfBlk`** | `1`, `0` | End of transport block indicator<br><br>When the transmission reaches the end of a transport block, the function returns this field as `1`.<br><br>At the beginning of a bundle, the function resets this field. | `logical` |
| **`EndOfTx`** | `1`, `0` | End of bundle indicator<br><br>When the transmission reaches the end of a bundle, the function returns this field as `1`. Otherwise, the function returns this field as `0`.<br><br>At the beginning of a bundle, the function resets this field. | `logical` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **GhpNSLot** | Nonnegative integer | Slot number for the first slot in the RU<br><br>The function uses this field only when you specify the `NPUSCHFormat` field as `'Data'` and the `NRUsc` field as `1` in the `chs` input. | `double` |

Data Types: `struct`

## More About

### Bundle

A bundle in the medium access control (MAC) layer refers to the repeated transmissions of a transport block.

For more information, see Section 5.3.2.1 of [2].

## Tips

To use this function for transmission of a bundle, follow these steps.

1. Call the function, optionally specifying the initial encoder state using the `stateIn` input. The `stateOut` output represents the first transmission of the transport block.
2. Call the function again, specifying the `stateIn` input as the `stateOut` output returned by the previous call to the function.
3. Repeat step 2 until the function returns the `EndOfTx` field of the `stateOut` output as `1` (`true`), indicating the end of the bundle.

# Version History

**Introduced in R2020a**

## References

[1] 3GPP TS 36.211. “Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. https://www.3gpp.org.

[2] 3GPP TS 36.321. “Medium Access Control (MAC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. https://www.3gpp.org.

## See Also

**Functions**

`lteNPUSCH` | `lteNPUSCHDecode` | `lteNPUSCHDRSIndices` | `lteNPUSCHIndices` | `ltePUSCHDRS`

# lteNPUSCHDRSIndices

Generate NPUSCH DRS RE indices

## Syntax

```
ind = lteNPUSCHDRSIndices(ue,chs)
ind = lteNPUSCHDRSIndices(ue,chs,opts)
```

## Description

`ind = lteNPUSCHDRSIndices(ue,chs)` generates `ind`, a column vector of narrowband physical uplink shared channel (NPUSCH) demodulation reference signal (DRS) resource element (RE) indices. The function generates indices for mapping NPUSCH symbols to physical resources, as specified in section 10.1.4.2 of [1], for user equipment (UE) settings `ue` and channel transmission configuration `chs`.

`ind = lteNPUSCHDRSIndices(ue,chs,opts)` specifies `opts`, the format in which the function returns the NPUSCH DRS RE indices.

## Examples

### Generate NPUSCH DRS RE Indices

Configure UE-specific settings.

```
ue = struct('NBULSubcarrierSpacing','15kHz');
```

Specify a channel transmission configuration.

```
chs =  struct('NPUSCHFormat','Data','NBULSubcarrierSet',0:11);
```

Generate and display the NPUSCH DRS RE indices for the specified settings.

```
ind = lteNPUSCHDRSIndices(ue,chs);
disp(ind')
```

```
    37    38    39    40    41    42    43    44    45    46    47    48
```

### Generate Zero-Based NPUSCH DRS RE Indices

Configure UE-specific settings.

```
ue = struct('NBULSubcarrierSpacing','15kHz');
```

Specify a channel transmission configuration.

```
chs =  struct('NPUSCHFormat','Data','NBULSubcarrierSet',0:11);
```

Generate the NPUSCH DRS RE indices, specifying zero-based formatting.

```
ind = lteNPUSCHDRSIndices(ue,chs,'0based');
```

## Input Arguments

### `ue` — UE-specific settings

structure

UE-specific settings, specified as a structure containing this field.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`NBULSubcarrierSpacing`** | `'3.75kHz'`, `'15kHz'` | NB-IoT uplink subcarrier spacing<br><br>To set a subcarrier spacing of 3.75 kHz, specify this field as `'3.75kHz'`. To set a subcarrier spacing of 15 kHz, specify this field as `'15kHz'`. | `char`, `string` |

Data Types: `struct`

### `chs` — Channel transmission configuration

structure

Channel transmission configuration, specified as a structure containing these fields.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`NPUSCHFormat`** | `'Data'`, `'Control'` | NPUSCH format<br><br>To indicate that the NPUSCH carries narrowband uplink shared channel (UL-SCH) data, specify this field as `'Data'`. To indicate that the NPUSCH carries uplink control information, specify this field as `'Control'`. | `char`, `string` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`NBULSubcarrierSet`** | Integer in the interval [0, 47], vector of integers in the interval [0, 11] | NB-IoT uplink subcarrier indices, in zero-based form<br><br>If you specify the `NPUSCHFormat` field as `'Control'`, specify this field as an integer in the interval [0, 11].<br><br>If you specify the `NPUSCHFormat` field as `'Data'` and the `NBULSubcarrierSpacing` field of the `ue` input as `'3.75kHz'`, specify this field as an integer in the interval [0, 47].<br><br>If you specify the `NPUSCHFormat` field as `'Data'` and the `NBULSubcarrierSpacing` field of the `ue` input as `'15kHz'`, specify this field as a vector of integers in the interval [0, 11]. | `double` |

Data Types: `struct`

**`opts` — Output format and index base of generated indices**
character vector | string scalar | cell array of character vectors | string array

Output format and index base of generated indices, specified as one of these forms.

- `'format base'`
- `"format base"`
- `{'format','base'}`
- `["format","base"]`

Where `format` and `base` are defined in this table.

| Option | Values | Description |
|---|---|---|

| | | |
|---|---|---|
| **format** | `'ind'` (default), `'sub'` | Output format of generated indices<br><br>To return the indices as a column vector, specify this option as `'ind'`.<br><br>To return the indices as an $N_{RE}$-by-3 matrix, where $N_{RE}$ is the number of REs, specify this option as `'sub'`. Each row of the matrix contains the subcarrier, symbol, and antenna port as its first, second, and third element, respectively. |
| **base** | `'1based'` (default), `'0based'` | Index base<br><br>To generate indices whose first value is `1`, specify this option as `'1based'`. To generate indices whose first value is `0`, specify this option as `'0based'`. |

Example: `'ind 0based'`, `"ind 0based"`, `{'ind','0based'}`, and `["ind","0based"]` specify the same output options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `ind` — NPUSCH DRS RE indices

integer-valued column vector

NPUSCH DRS RE indices, returned as an integer-valued column vector of length $N_{RE}$ equal to the number of REs.

Data Types: `double`

# Version History

**Introduced in R2020a**

## References

[1] 3GPP TS 36.211. “Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. https://www.3gpp.org.

## See Also

**Functions**

`lteNPUSCH` | `lteNPUSCHDecode` | `lteNPUSCHDRS` | `lteNPUSCHIndices` | `ltePUSCHDRSIndices`

# lteNPUSCHIndices

Generate NPUSCH RE indices

## Syntax

```
[ind,info] = lteNPUSCHIndices(ue,chs)
[ind,info] = lteNPUSCHIndices(ue,chs,opts)
```

## Description

`[ind,info] = lteNPUSCHIndices(ue,chs)` generates `ind`, a column vector of narrowband physical uplink shared channel (NPUSCH) resource element (RE) indices, and `info`, information related to the indices. The function generates indices for mapping NPUSCH symbols to physical resources, as specified in section 10.1.3.6 of [1], for user equipment (UE) settings `ue` and channel transmission configuration `chs`.

`[ind,info] = lteNPUSCHIndices(ue,chs,opts)` specifies `opts`, the format in which the function returns the NPUSCH RE indices.

## Examples

### Generate NPUSCH RE Indices

Configure UE-specific settings.

```
ue = struct('NBULSubcarrierSpacing','15kHz');
```

Specify a channel transmission configuration.

```
chs =  struct('NPUSCHFormat','Data','NBULSubcarrierSet',[0:11],'NRU',1, ...
    'NULSlots',2,'Modulation','BPSK');
```

Generate the NPUSCH RE indices for the specified settings. Display the corresponding information.

```
[ind,info] = lteNPUSCHIndices(ue,chs);
disp(info)
```

```
    Gd: 144
     G: 144
```

### Generate Zero-Based NPUSCH RE Indices

Configure UE-specific settings.

```
ue = struct('NBULSubcarrierSpacing','15kHz');
```

Specify a channel transmission configuration.

```
chs =  struct('NPUSCHFormat','Data','NBULSubcarrierSet',0:11, ...
    'NRU',1,'NULSlots',2,'Modulation','BPSK');
```

Generate the NPUSCH RE indices, specifying zero-based formatting.

```
[ind,info] = lteNPUSCHIndices(ue,chs,'0based');
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific settings, specified as a structure containing this field.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **NBULSubcarrierSpacing** | `'3.75kHz'`, `'15kHz'` | NB-IoT uplink subcarrier spacing<br><br>To set a subcarrier spacing of 3.75 kHz, specify this field as `'3.75kHz'`. To set a subcarrier spacing of 15 kHz, specify this field as `'15kHz'`. | `char`, `string` |

Data Types: `struct`

### chs — Channel transmission configuration

structure

Channel transmission configuration, specified as a structure containing these fields.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **NPUSCHFormat** | `'Data'`, `'Control'` | NPUSCH format<br><br>To indicate that the NPUSCH carries narrowband uplink shared channel (UL-SCH) data, specify this field as `'Data'`. To indicate that the NPUSCH carries uplink control information, specify this field as `'Control'`. | `char`, `string` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **NBULSubcarrierSet** | Integer in the interval [0, 47], vector of integers in the interval [0, 11] | NB-IoT uplink subcarrier indices, in zero-based form<br><br>If you specify the `NBULSubcarrierSpacing` field of the `ue` input as `'3.75kHz'`, specify this field as an integer in the interval [0, 47].<br><br>If you specify the `NBULSubcarrierSpacing` field of the `ue` input as `'15kHz'`, specify this field as a vector of integers in the interval [0, 11]. | `double` |
| **NRU** | 1, 2, 3, 4, 5, 6, 8, 10 | Number of RUs | `double` |
| **NULSlots** | 2, 4, 8, 16 | Number of slots per RU<br><br>If you specify the `NPUSCHFormat` field as `'Control'`, then you must specify this field as 4.<br><br>If you specify the `NPUSCHFormat` field as `'Data'`, then you must specify this field as<br>• 16 when you specify the `NRUsc` field as 1<br>• 8 when you specify the `NRUsc` field as 3<br>• 4 when you specify the `NRUsc` field as 6<br>• 2 when you specify the `NRUsc` field as 12 | `double` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **Modulation** | 'BPSK', 'QPSK' | Modulation type<br><br>To enable binary phase-shift keying (BPSK), specify this field as 'BPSK'. To enable quadrature phase-shift keying (QPSK), specify this field as 'QPSK'.<br><br>If you specify the NPUSCHFormat field as 'Control', then you must specify this field as 'BPSK'. | char, string |

Data Types: struct

**opts — Output format and index base of generated indices**
character vector | string scalar | cell array of character vectors | string array

Output format and index base of generated indices, specified as one of these forms.

- 'format base'
- "format base"
- {'format','base'}
- ["format","base"]

Where format and base are defined in this table.

| Option | Values | Description |
|---|---|---|
| **format** | 'ind' (default), 'sub' | Output format of generated indices<br><br>To return the indices as a column vector, specify this option as 'ind'.<br><br>To return the indices as an $N_{RE}$-by-3 matrix, where $N_{RE}$ is the number of REs, specify this option as 'sub'. Each row of the matrix contains the subcarrier, symbol, and antenna port as its first, second, and third element, respectively. |
| **base** | '1based' (default), '0based' | Index base<br><br>To generate indices whose first value is 1, specify this option as '1based'. To generate indices whose first value is 0, specify this option as '0based'. |

Example: `'ind 0based'`, `"ind 0based"`, `{'ind','0based'}`, and `["ind","0based"]` specify the same output options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `ind` — NPUSCH RE indices

integer-valued column vector

NPUSCH RE indices, returned as an integer-valued column vector of length $N_{RE}$ equal to the number of REs.

Data Types: `double`

### `info` — Information related to NPUSCH RE indices

structure

Information related to NPUSCH RE indices, returned as a structure containing these fields.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`G`** | Nonnegative integer | Number of coded and rate-matched uplink shared channel (UL-SCH) data bits for a codeword | `double` |
| **`Gd`** | Nonnegative integer | Number of UL-SCH data symbols<br><br>The function returns this field as the value of `chs.NRU` x `chs.NULSlots` x $N_{RE}$. | `double` |

Data Types: `struct`

# Version History

**Introduced in R2020a**

## References

[1] 3GPP TS 36.211. “Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. https://www.3gpp.org.

## See Also

**Functions**

`lteNPUSCH` | `lteNPUSCHDecode` | `lteNPUSCHDRS` | `lteNPUSCHDRSIndices` | `lteNULSCH` | `lteNULSCHDecode` | `ltePUSCHIndices`

# lteNULSCH

Generate NB-IoT UL-SCH codeword

## Syntax

```
cw = lteNULSCH(chs,outLen,trBlkIn)
cw = lteNULSCH(trBlkIn)
```

## Description

`cw = lteNULSCH(chs,outLen,trBlkIn)` generates `cw`, an NB-IoT uplink shared channel (UL-SCH) codeword of length `outLen`, by processing `trBlkIn`, the input transport block (data) or uplink control information (UCI), for channel transmission configuration settings `chs`. Use this syntax for NB-IoT UL-SCH data or UCI processing.

NB-IoT UL-SCH data processing comprises type-24A cyclic redundancy check (CRC) attachment, turbo encoding, rate matching to `outLen`, and interleaving, in accordance with section 6.3.2 of [1].

NB-IoT UL-SCH UCI processing comprises mapping control information bit `trBlkIn` to a 16-bit codeword `cw` in accordance with section 6.3.3 of [1].

`cw = lteNULSCH(trBlkIn)` generates `cw` by mapping control information bit `trBlkIn` in accordance with section 6.3.3 of [1]. Use this syntax for NB-IoT UL-SCH UCI processing.

## Examples

### Generate NB-IoT UL-SCH Data Codeword

Generate a codeword by applying NB-IoT UL-SCH data processing to a transport block.

Configure UE-specific settings.

```
ue = struct('NBULSubcarrierSpacing','15kHz');
```

Specify a channel transmission configuration.

```
chs =  struct('NPUSCHFormat','Data','NULSlots',16,'NBULSubcarrierSet',6, ...
    'Modulation','BPSK','NRU',2,'RV',0);
```

Specify the codeword length and create a transport block for encoding.

```
[~,info] = lteNPUSCHIndices(ue,chs);
outLen = info.G;
trBlkIn = randi([0,1],144,1);
```

Generate the UL-SCH codeword.

```
cw = lteNULSCH(chs,outLen,trBlkIn);
```

### Generate NB-IoT UL-SCH UCI Codeword

Generate a codeword by applying UCI processing to a control information bit.

Create a control information bit for encoding.

```
trBlkIn = 1;
```

Generate and display the NB-IoT UL-SCH codeword.

```
cw = lteNULSCH(trBlkIn);
disp(cw')
```

```
   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
```

## Input Arguments

### `chs` — Channel transmission configuration

structure

Channel transmission configuration, specified as a structure containing these fields.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`NPUSCHFormat`** | `'Data'` (default), `'Control'` | Narrowband physical uplink shared channel (NPUSCH) format<br><br>To indicate that the NPUSCH carries narrowband uplink shared channel (UL-SCH) data, specify this field as `'Data'`. To indicate that the NPUSCH carries uplink control information, specify this field as `'Control'`. | `char`, `string` |
| **`NRU`** | 1, 2, 3, 4, 5, 6, 8, 10 | Number of resource units (RUs) | `double` |
| **`NULSlots`** | 2, 4, 8, 16 | Number of slots per RU | `double` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **Modulation** | `'BPSK'`, `'QPSK'` | Modulation type, specified as one of these values:<br><br>To enable binary phase-shift keying (BPSK), specify this field as `'BPSK'`. To enable quadrature phase-shift keying (QPSK), specify this field as `'QPSK'`. | `char`, `string` |
| **RV** | `0`, `2` | Redundancy version indicator | `double` |

Data Types: `struct`

**`outLen` — Codeword length**
positive integer

Codeword length, specified as a positive integer.

When you specify the `NPUSCHFormat` field of the `chs` input as `'Data'`, specify this input as the NPUSCH capacity for the associated codeword. The `lteNULSCH` function generates the `cw` output as a vector of this length by rate matching the encoded transport block to the specified value.

When you specify the `NPUSCHFormat` field of the `chs` input as `'Control'`, the `lteNULSCH` function sets this input to `16`.

Data Types: `double`

**`trBlkIn` — Transport block (data) or UCI**
`0` | `1` | binary-valued column vector

Transport block (data) or UCI, specified as one of these values.

| NPUSCH Format | `trBlkIn` | Value |
|---|---|---|
| Data | Transport block | Binary-valued column vector |
| Control | UCI | `0`, `1` |

Data Types: `double`

## Output Arguments

**`cw` — NB-IoT UL-SCH codeword**
binary-valued column vector

NB-IoT UL-SCH codeword, returned as a binary-valued column vector of length `outLen`.

Data Types: `int8`

# Version History

**Introduced in R2020a**

## References

[1] 3GPP TS 36.212. “Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. https://www.3gpp.org.

## See Also

**Functions**

`lteNPUSCH` | `lteNPUSCHDecode` | `lteNULSCHDecode` | `lteULSCH`

# lteNULSCHDecode

Decode NB-IoT UL-SCH codeword

## Syntax

```
[trBlkOut,blkCRC,stateOut] = lteNULSCHDecode(chs,trBlkLen,cw)
[trBlkOut,blkCRC,stateOut] = lteNULSCHDecode(cw)
[trBlkOut,blkCRC,stateOut] = lteNULSCHDecode( ___ ,stateIn)
```

## Description

`[trBlkOut,blkCRC,stateOut] = lteNULSCHDecode(chs,trBlkLen,cw)` returns `trBlkOut`, a vector of length `trBlkLen` containing NB-IoT UL-SCH data or uplink control information (UCI) decoded from `cw`, an NB-IoT uplink shared channel (UL-SCH) codeword of log-likelihood ratios (LLRs). The function also returns `blkCRC`, the result of type-24A transport block cyclic redundancy check (CRC) decoding, and `stateOut`, the hybrid automatic repeat request (HARQ) process decoding status. Use this syntax for NB-IoT UL-SCH data or UCI decoding. If you use this syntax for UCI decoding, the function decodes `cw` without soft combining.

For NB-IoT UL-SCH data decoding, the function inverts the UL-SCH processing described in section 6.3.2 of [1] by deinterleaving, rate recovering, turbo decoding, and type-24A transport block CRC decoding the input codeword.

For UCI decoding, the function inverts the UL-SCH processing described in section 6.3.3 of [1] by slicing the codeword data.

`[trBlkOut,blkCRC,stateOut] = lteNULSCHDecode(cw)` decodes NB-IoT UL-SCH codeword `cw` by inverting the UL-SCH processing described in section 6.3.3 of [1]. Use this syntax for UCI decoding without soft combining.

`[trBlkOut,blkCRC,stateOut] = lteNULSCHDecode( ___ ,stateIn)` specifies `stateIn`, the initial decoder state for each transport block in an active HARQ process, in addition to any input combination from previous syntaxes.. If you use this syntax for UCI decoding, the function decodes `cw` with soft combining.

## Examples

### Decode NB-IoT UL-SCH UCI Codeword

Generate a codeword by applying UCI encoding to a control information bit. Recover the control information by decoding.

Create a control information bit for encoding.

```
trBlkIn = 1;
```

Generate the UL-SCH codeword.

```
cw = lteNULSCH(trBlkIn);
```

Recover the transport block by decoding the NB-IoT UL-SCH codeword.

```
[trBlkOut,blkCRC,stateOut] = lteNULSCHDecode(cw);
```

Confirm that the recovered bit matches the input control information bit.

```
disp(isequal(trBlkIn,trBlkOut))
```

```
   1
```

### Decode NB-IoT UL-SCH Data Codeword

Generate an NB-IoT UL-SCH data codeword by encoding a 136-bit transport block. Split the codeword into two NPUSCH transmissions in two consecutive slots. Decode the NPUSCH transmission, and then decode the received NB-IoT UL-SCH codeword.

Configure UE-specific settings.

```
ue = struct('NBULSubcarrierSpacing','15kHz','NNCellID',10,'NSlot',0);
```

Specify a channel transmission configuration.

```
chs = struct('NPUSCHFormat','Data','NBULSubcarrierSet',0:11,'NRUsc',length(0:11),'NRep',1, ...
    'NRU',1,'NULSlots',2,'Modulation','QPSK','RV',0,'NTurboDecIts',5,'RNTI',20);
```

Generate a transport block for encoding.

```
trBlkLen = 136;
trBlkIn = randi([0 1],trBlkLen,1);
```

Get the codeword length by generating the NPUSCH RE indices and associated information.

```
[~,info] = lteNPUSCHIndices(ue,chs);
outLen = info.G;
```

Generate the UL-SCH codeword.

```
cwIn = lteNULSCH(chs,outLen,trBlkIn);
```

Initialize the encoder state for the NPUSCH and the decoder states for the NPUSCH and UL-SCH.

```
npuschStateIn = struct();
npuschDecodeStateIn = struct();
stateIn = struct();
```

Generate NPUSCH symbols for slot 0, and then return the decoder state by decoding the NPUSCH symbols.

```
[symSlot0,npuschStateOut] = lteNPUSCH(ue,chs,cwIn,npuschStateIn);
[~,npuschDecodeStateOut,~] = lteNPUSCHDecode(ue,chs,symSlot0,npuschDecodeStateIn);
```

Generate NPUSCH symbols for slot 1, and then return the decoder state by decoding the NPUSCH symbols.

```
ue.NSlot = 1;
[symSlot1,~] = lteNPUSCH(ue,chs,cwIn,npuschStateOut);
[cw,~,~] = lteNPUSCHDecode(ue,chs,symSlot1,npuschDecodeStateOut);
```

Decode received NB-IoT UL-SCH data codeword.

```
[trBlkOut,blkCRC,stateOut] = lteNULSCHDecode(chs,trBlkLen,cw,stateIn);
```

Confirm that the recovered NB-IoT UL-SCH matches the original transport block.

```
disp(isequal(trBlkIn,trBlkOut))
```

```
   1
```

## Input Arguments

### `chs` — Channel transmission configuration

structure

Channel transmission configuration, specified as a structure containing these fields.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`NPUSCHFormat`** | `'Data'` (default), `'Control'` | Narrowband physical uplink shared channel (NPUSCH) format<br><br>To indicate that the NPUSCH carries narrowband uplink shared channel (UL-SCH) data, specify this field as `'Data'`. To indicate that the NPUSCH carries uplink control information, specify this field as `'Control'`. | `char`, `string` |
| **`NRU`** | `1`, `2`, `3`, `4`, `5`, `6`, `8`, `10` | Number of resource units (RUs) | `double` |
| **`NULSlots`** | `2`, `4`, `8`, `16` | Number of slots per RU | `double` |
| **`Modulation`** | `'BPSK'`, `'QPSK'` | Modulation type, specified as one of these values:<br><br>To enable binary phase-shift keying (BPSK), specify this field as `'BPSK'`. To enable quadrature phase-shift keying (QPSK), specify this field as `'QPSK'`. | `char`, `string` |
| **`RV`** | `0`, `2` | Redundancy version indicator | `double` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **NTurboDecIts** | 5 (default), integer in the interval [1, 30] | Number of turbo decoder iteration cycles | double |

Data Types: struct

**trBlkLen — Transport block length**
positive integer

Transport block length, specified as a positive integer. The function rate recovers and turbo decodes the cw input to the value of this input.

Data Types: double

**cw — NB-IoT UL-SCH codeword of LLRs**
binary-valued column vector

NB-IoT UL-SCH codeword of LLRs, specified as a binary-valued column vector.

Data Types: double

**stateIn — Initial decoder buffer state**
struct() | structure

Initial decoder buffer state for each transport block in an active HARQ process, specified as a structure containing this field.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **CBSBuffers** | Cell array | LLR soft buffer state for the transport block input to the turbo decoder after rate recovery | cell |

Data Types: struct

## Output Arguments

**trBlkOut — Decoded NB-IoT UL-SCH data or UCI information bits**
binary-valued column vector

Decoded NB-IoT UL-SCH data or UCI information bits, returned as a binary-valued column vector.

Data Types: int8

**blkCRC — Type-24A transport block CRC decoding error indicator**
0 | 1

Type-24A transport block CRC decoding error indicator, returned as 0 or 1. When the lteNULSCHDecode function decodes cw with zero errors, it returns this output as 0. Otherwise, the function returns this output as 1.

Data Types: logical

**`stateOut` — Output decoder buffer state**
structure

Output decoder buffer state for each transport block in an active HARQ process, returned as a structure containing these fields.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`CBSBuffers`** | Cell array | LLR soft buffer state for the transport block input to the turbo decoder after rate recovery | `cell` |
| **`BLKCRC`** | `1`, `0` | Type-24A transport block CRC decoding error indicator<br><br>When the function decodes `cw` with zero errors, it returns this output as `0` (`false`). Otherwise, the function returns this output as `1` (`true`). | `logical` |

Data Types: `struct`

# Version History

**Introduced in R2020a**

## References

[1] 3GPP TS 36.212. “Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. https://www.3gpp.org.

## See Also

**Functions**
`lteNPUSCH` | `lteNPUSCHDecode` | `lteNULSCH`

# lteNRS

Generate cell-specific NRS symbols

## Syntax

```
sym = lteNRS(enb)
sym = lteNRS(enb,ports)
```

## Description

`sym = lteNRS(enb)` generates `sym`, a column vector of cell-specific narrowband reference signal (NRS) symbols, for cell-wide settings `enb`. Unlike other physical channels and signals, the symbols for all antennas are not returned as a matrix with one column for each antenna, since the number of symbols varies across antenna ports.

`sym = lteNRS(enb,ports)` generates NRS symbols for input cell-wide settings for the input NRS antenna ports specified by `ports`.

## Examples

### Generate NRS Symbols

Initialize cell-wide settings by specifying a narrowband physical layer cell identity, number of NRS antenna ports, and subframe number.

```
enb.NNCellID = 42;    % Physical layer cell identity
enb.NBRefP = 1;       % Number of NRS antenna ports
enb.NSubframe = 4;    % Subframe number
```

Generate and display the NRS symbols for the specified cell-wide settings.

```
sym = lteNRS(enb);
disp(sym);
```

```
   0.7071 - 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
   0.7071 - 0.7071i
   0.7071 + 0.7071i
  -0.7071 - 0.7071i
   0.7071 + 0.7071i
   0.7071 - 0.7071i
```

### Find Number of Transmitted NRS Symbols

Find the number of NRS symbols transmitted at NRS antenna ports.

Specify the cell-wide settings as fields in the structure `enb`. Generate the NRS symbols transmitted at antenna port 2000.

```
enb.NNCellID = 10;      % Physical layer cell identity
enb.NBRefP = 2;         % Number of NRS antenna ports
enb.NSubframe = 2;      % Subframe number
ports = 0;              % Antenna port 0
sym = lteNRS(enb,ports);    % Return NRS symbols
```

Compute and display the number of NRS symbols transmitted at the antenna port.

```
nrsPort2000 = length(sym);
disp(nrsPort2000);
```

```
     8
```

Generate the NRS symbols transmitted at antenna ports 2000 and 2001. Compute and display the number of NRS symbols transmitted at both ports.

```
symAll = lteNRS(enb);
nrsPortAll = length(symAll);
disp(nrsPortAll);
```

```
    16
```

## Input Arguments

### `enb` — Cell-wide settings

structure

Cell-wide settings, specified as a structure containing these fields:

| Name | Values | Description | Data Types |
|---|---|---|---|
| **`NNCellID`** | integer in the interval [0, 503] | Narrowband physical layer cell identity | `double` |
| **`NBRefP`** | 1, 2 | Number of NRS antenna ports | `double` |
| **`NSubframe`** | integer | Subframe number | `double` |

Data Types: `struct`

### `ports` — NRS antenna ports

`0` | `1` | `[0 1]`

NRS antenna ports, specified as `0`, `1`, or `[0 1]`.

Data Types: `double`

## Output Arguments

### `sym` — NRS symbols for a subframe

complex-valued column vector

NRS symbols for a subframe, returned as a complex-valued column vector.

Data Types: `double`

## Version History

**Introduced in R2019a**

## See Also

`lteDLChannelEstimate` | `lteNRSIndices`

# lteNRSIndices

NRS resource element indices

## Syntax

```
ind = lteNRSIndices(enb)
ind = lteNRSIndices(enb,ports)
ind = lteNRSIndices(enb,opts)
ind = lteNRSIndices(enb,ports,opts)
```

## Description

`ind = lteNRSIndices(enb)` returns a column vector of resource element (RE) indices for the narrowband reference signal (NRS), given the cell-wide settings in the `enb` structure. By default, the indices are returned in 1-based linear indexing form. Using this form, you can directly index elements of a 3-D array representing the subframe resource grid for all antenna ports. The indices are ordered as the reference signal modulation symbols are mapped. Alternative indexing formats can also be generated. The indices for multiple antennas are concatenated into a single column rather than returned in a matrix with a column for each antenna. The indices for each antenna are concatenated because the number of indices varies across the antenna ports.

`ind = lteNRSIndices(enb,ports)` returns a column vector of RE indices for antenna ports specified in the `ports` vector and `enb` cell-wide settings structure. In this case, `lteNRSIndices` ignores the `NBRefP` field of `enb` and uses `ports` instead.

`ind = lteNRSIndices(enb,opts)` formats the returned indices using options specified by `opts`.

`ind = lteNRSIndices(enb,ports,opts)` returns the RE indices for antenna ports specified in the `ports` vector and formats them using options defined in `opts`.

## Examples

### Generate Narrowband Reference Signal RE Indices

Generate 0-based narrowband reference signal RE indices in subscript form for antenna port 1. Each row of the generated matrix has three columns representing the subcarrier, symbol, and antenna port, respectively.

Create cell-wide settings of the eNodeB structure.

```
enb.NNCellID = 10;
enb.NBRefP = 2;
```

Generate the 0-based narrowband reference signal RE indices.

```
ind = lteNRSIndices(enb,1,{'0based','sub'})
```

```
ind = 8×3

     1     5     1
```

```
     7     5     1
     4     6     1
    10     6     1
     1    12     1
     7    12     1
     4    13     1
    10    13     1
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure containing these fields:

| Name | Values | Description | Data Types |
|---|---|---|---|
| **NNCellID** | Integer in the interval [0, 503] | Narrowband physical layer cell identity | double |
| **NBRefP** | 1, 2 | Number of NRS antenna ports | double |

Data Types: struct

### ports — NRS antenna ports

0 | 1 | vector

NRS antenna ports, specified as 0, 1, or a vector whose elements must be 0 or 1. Ports 0 and 1 stand for NRS antenna ports 2000 and 2001, respectively.

Data Types: double

### opts — Output format and index base of generated indices

character vector | string scalar | cell array of character vectors | string array

Output format and index base of generated indices, specified as one of these forms.

- 'format base'
- "format base"
- {'format','base'}
- ["format","base"]

Where format and base are defined in this table.

| Option | Values | Description |
|---|---|---|

| | | |
|---|---|---|
| **format** | `'ind'` (default), `'sub'` | Output format of generated indices<br><br>To return the indices as a column vector, specify this option as `'ind'`.<br><br>To return the indices as an $N_{RE}$-by-3 matrix, where $N_{RE}$ is the number of REs, specify this option as `'sub'`. Each row of the matrix contains the subcarrier, symbol, and antenna port as its first, second, and third element, respectively. |
| **base** | `'1based'` (default), `'0based'` | Index base<br><br>To generate indices whose first value is `1`, specify this option as `'1based'`. To generate indices whose first value is `0`, specify this option as `'0based'`. |

Example: `'ind 0based'`, `"ind 0based"`, `{'ind','0based'}`, and `["ind","0based"]` specify the same output options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `ind` — Narrowband reference signal RE indices

numeric column vector | matrix

NRS RE indices, returned as a numeric column vector or matrix, depending on the indexing style specified in `opts`:

- If you specify linear indexing (default), then `ind` is a column vector.
- If you specify subscript row style indexing, then `ind` is an $N_{RE}$-by-3 matrix, where $N_{RE}$ is the number of resource elements.

Data Types: `uint32`

# Version History

**Introduced in R2018a**

## See Also

`lteNRS` | `lteCellRSIndices` | `lteCellRS`

# lteNSSS

Generate NSSS symbols for subframe

## Syntax

```
sym = lteNSSS(enb)
```

## Description

`sym = lteNSSS(enb)` generates narrowband secondary synchronization (NSSS) symbols `sym` for cell-wide settings specified `enb`.

## Examples

### Generate NSSS Symbols

Initialize cell-wide settings by specifying a narrowband operation mode, subframe number, frame number, and physical layer cell identity.

```
enb.OperationMode = 'Standalone';    % Narrowband operation mode
enb.NSubframe = 9;                   % Subframe number
enb.NFrame = 0;                      % Frame number
enb.NNCellID = 1;                    % Physical layer cell identity
```

Generate the NSSS symbols.

```
sym = lteNSSS(enb);
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure containing these fields.

| Name | Required or Optional | Values | Description | Data Types |
|---|---|---|---|---|
| **`OperationMode`** | Optional | `'Standalone'` (default), `'Inband-SamePCI'`, `'Inband-DifferentPCI'`, `'Guardband'` | NB-IoT operation mode, specified as one of these values:<br>• `'Standalone'` – NB-IoT standalone operation within any 180-kHz band outside any LTE carrier bandwidth<br>• `'Inband-SamePCI'` – NB-IoT in-band operation with the same physical layer cell identity (PCI) as an LTE carrier<br>• `'Inband-DifferentPCI'` – NB-IoT in-band operation with a different PCI to an LTE carrier<br>• `'Guardband'` – NB-IoT guard-band operation utilizing unused resource blocks within the guard-band of an LTE carrier | `char`, `string` |
| **`NNCellID`** | Required | Integer in the interval [0, 503] | Narrowband PCI | `double` |

| Name | Required or Optional | Values | Description | Data Types |
|---|---|---|---|---|
| **NSubframe** | Optional | 9 (default), integer | Subframe number. Because the NSSS is defined only for subframe 9 in alternate frames, the function returns an empty array for any value of this field other than 9. This behavior enables resource grid indexing for any subframe number. | double |
| **NFrame** | Optional | 0 (default), integer | Frame number. Because the NSSS is defined only for subframe 9 in alternate frames, the function returns an empty vector for odd values of this field. This behavior enables resource grid indexing for any subframe number and any frame number. | double |
| **NCellID** | Required when you specify OperationMode as 'Inband-SamePCI' or 'Inband-DifferentPCI' | Integer in the interval [0, 503] | PCI | double |
| **CellRefP** | Required when you specify OperationMode as 'Inband-SamePCI' or 'Inband-DifferentPCI' | 1, 2, 4 | Number of cell-specific antenna ports | double |

**Note** To exclude cell reference signal (RS) locations, specify the NCellID and CellRefP fields. If you do not specify the NCellID and CellRefP fields, the function assumes that the cell RS is absent and generates NSSS values for all cell RS locations.

Data Types: `struct`

## Output Arguments

### `sym` — NSSS symbols for subframe

complex-valued column vector | empty array

NSSS symbols for a subframe, returned as a complex-valued column vector. If you specify the `NSubframe` field of the `enb` input as any value other than `9` or the `NFrame` field as an odd value, then the function returns this output as an empty array.

# Version History

**Introduced in R2019a**

## See Also

### Functions

`lteNBDLFrameOffset` | `lteNSSSIndices` | `lteNPSS`

### Topics

“Resource Grid Indexing”

# lteNSSSIndices

Generate NSSS RE indices for subframe

## Syntax

```
ind = lteNSSSIndices(enb)
ind = lteNSSSIndices(enb,port)
ind = lteNSSSIndices(enb,port,opts)
```

## Description

`ind = lteNSSSIndices(enb)` generates `ind`, the narrowband secondary synchronization signal (NSSS) resource element (RE) indices for cell-wide settings `enb`.

`ind = lteNSSSIndices(enb,port)` generates the NSSS RE indices for the antenna port corresponding to the `port` input.

`ind = lteNSSSIndices(enb,port,opts)` generates the NSSS RE indices for the specified antenna port in the format specified by `opts`.

## Examples

### Generate Zero-Based NSSS RE Indices

Generate zero-based NSSS RE indices for antenna port 2001.

Initialize cell-wide settings by specifying the operation mode, number of cell-specific RS antenna ports, physical layer cell identity, frame number, and subframe number.

```
enb.OperationMode = 'Inband-SamePCI';    % Operation mode
enb.CellRefP = 1;                        % Number of cell-specific RS antenna ports
enb.NCellID = 2;                         % Physical layer cell identity
enb.NSubframe = 9;                       % Subframe number
enb.NFrame = 4;                          % Frame number
```

Specify the antenna port and generate the NSSS RE indices, specifying zero-based indexing. To return a matrix whose rows each contain the subcarrier, index, and antenna port of the corresponding RE, specify the option `'sub'`.

```
port = 1;
ind = lteNSSSIndices(enb,port,{'0based','sub'});
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure containing these fields.

| Name | Required or Optional | Values | Description | Data Types |
|---|---|---|---|---|
| **`OperationMode`** | Optional | `'Standalone'` (default), `'Inband-SamePCI'`, `'Inband-DifferentPCI'`, `'Guardband'` | NB-IoT operation mode, specified as one of these values:<br>• `'Standalone'` – NB-IoT standalone operation within any 180-kHz band outside any LTE carrier bandwidth<br>• `'Inband-SamePCI'` – NB-IoT in-band operation with the same physical layer cell identity (PCI) as an LTE carrier<br>• `'Inband-DifferentPCI'` – NB-IoT in-band operation with a different PCI to an LTE carrier<br>• `'Guardband'` – NB-IoT guard-band operation utilizing unused resource blocks within the guard-band of an LTE carrier | `char`, `string` |

| Name | Required or Optional | Values | Description | Data Types |
|---|---|---|---|---|
| **NSubframe** | Optional | 9 (default), integer | Subframe number. Because the NSSS is defined only for subframe 9 in alternate frames, the function returns an empty array for any value of this field other than 9. This behavior enables resource grid indexing for any subframe number. | double |
| **NFrame** | Optional | 0 (default), integer | Frame number. Because the NSSS is defined only for subframe 9 in alternate frames, the function returns an empty vector for odd values of this field. This behavior enables resource grid indexing for any subframe number and any frame number. | double |
| **NCellID** | Required when you specify OperationMode as 'Inband-SamePCI' or 'Inband-DifferentPCI' | Integer in the interval [0, 503] | PCI | double |
| **CellRefP** | Required when you specify OperationMode as 'Inband-SamePCI' or 'Inband-DifferentPCI' | 1, 2, 4 | Number of cell-specific antenna ports | double |

**Note** To exclude cell reference signal (RS) locations, specify the NCellID and CellRefP fields. If you do not specify the NCellID and CellRefP fields, the function assumes that the cell RS is absent and generates NSSS values for all cell RS locations.

Data Types: `struct`

**`port` — Antenna port**
`0` | `1`

Antenna port, specified as `0` or `1`, corresponding to antenna port 2000 or 2001, respectively.

Data Types: `double`

**`opts` — Output format and index base of generated indices**
character vector | string scalar | cell array of character vectors | string array

Output format and index base of generated indices, specified as one of these forms.

- `'format base'`
- `"format base"`
- `{'format','base'}`
- `["format","base"]`

Where `format` and `base` are defined in this table.

| Option | Values | Description |
|---|---|---|
| **`format`** | `'ind'` (default), `'sub'` | Output format of generated indices<br><br>To return the indices as a column vector, specify this option as `'ind'`.<br><br>To return the indices as an $N_{RE}$-by-3 matrix, where $N_{RE}$ is the number of REs, specify this option as `'sub'`. Each row of the matrix contains the subcarrier, symbol, and antenna port as its first, second, and third element, respectively. |
| **`base`** | `'1based'` (default), `'0based'` | Index base<br><br>To generate indices whose first value is `1`, specify this option as `'1based'`. To generate indices whose first value is `0`, specify this option as `'0based'`. |

Example: `'ind 0based'`, `"ind 0based"`, `{'ind','0based'}`, and `["ind","0based"]` specify the same output options.

Data Types: `char` | `string` | `cell`

## Output Arguments

**`ind` — NSSS RE indices for a subframe**
complex-valued array | empty array

NSSS RE indices for a subframe, returned as a complex-valued array. The array dimensions depend on the format options you specify in `opts`. To return `ind` as a column vector, specify `'ind'` in the

`opts` input. To return `ind` as an $N_{RE}$-by-3 matrix, specify `'sub'` in the `opts` input. If you specify the `NSubframe` field of the `enb` input as a value other than `9` or the `NFrame` field as an odd value, the function returns this output as an empty array.

Data Types: `uint32`

# Version History

**Introduced in R2019a**

## See Also

**Functions**

`lteNBDLFrameOffset` | `lteNSSS` | `lteNPSSIndices`

**Topics**

“Resource Grid Indexing”

# lteOFDMDemodulate

OFDM demodulation

## Syntax

```
grid = lteOFDMDemodulate(enb,waveform)
grid = lteOFDMDemodulate(enb,waveform,cpfraction)
grid = lteOFDMDemodulate(enb,waveform,cpfraction,Nfft)
```

## Description

`grid = lteOFDMDemodulate(enb,waveform)` performs OFDM demodulation of `waveform`, the time-domain waveform, for cell-wide settings `enb`.

The demodulation performs one FFT operation per received OFDM symbol to recover the received subcarrier values. These values are then used to construct each column of the output resource array, `grid`. The FFT is positioned partway through the cyclic prefix to allow for a certain degree of channel delay spread while avoiding the overlap between adjacent OFDM symbols. The particular position of the FFT chosen here avoids the OFDM symbol overlapping used in `lteOFDMModulate`. Since the FFT is performed away from the original zero-phase point on the transmitted subcarriers, a phase correction is applied to each subcarrier after the FFT. Then, the received subcarriers are extracted from the FFT bins, skipping unused frequency bins at either end of the spectrum and the central DC frequency bin. These extracted subcarriers form the columns of the output `grid`.

The sampling rate of the time-domain waveform, `waveform`, must be the same as used in `lteOFDMModulate` for the specified number of resource blocks, `NDLRB`. `waveform` must also be time-aligned such that the first sample is the first sample of the cyclic prefix of the first OFDM symbol in a subframe. This alignment can be achieved by using `lteDLFrameOffset`.

`grid = lteOFDMDemodulate(enb,waveform,cpfraction)` specifies the position of the demodulation through the cyclic prefix.

`grid = lteOFDMDemodulate(enb,waveform,cpfraction,Nfft)` specifies the number of FFT points to use in the demodulation.

## Examples

### Perform OFDM Demodulation

Perform modulation and demodulation of Test Model 1.1 5MHz.

```
cfg = lteTestModel('1.1','5MHz');
txWaveform = lteTestModelTool(cfg);
rxGrid = lteOFDMDemodulate(cfg,txWaveform);
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure containing these fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NDLRB** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **CyclicPrefix** | Optional | 'Normal' (default), 'Extended' | Cyclic prefix length |

Data Types: struct

### waveform — Time-domain waveform

numeric matrix

Time-domain waveform, specified as a numeric matrix of size *T*-by-*P*, where *P* is the number of antennas and *T* is the number of time-domain samples. $T = K \times 30720 / 2048 \times N_{fft}$, where $N_{fft}$ is the IFFT size and *K* is the number of subframes in the input, grid. waveform must be time-aligned such that the first sample is the first sample of the cyclic prefix of the first OFDM symbol in a subframe.

Data Types: double
Complex Number Support: Yes

### cpfraction — Demodulation position

0.55 (default) | scalar in the interval [0, 1]

Demodulation position, specified as a scalar in the interval [0, 1], with 0 representing the start of the cyclic prefix and 1 representing the end of the cyclic prefix. The default value allows for the default level of windowing in lteOFDMModulate.

Data Types: double

### Nfft — Number of FFT points

positive integer

The number of FFT points to use in the OFDM demodulation, specified as a positive integer.

Data Types: double

## Output Arguments

### grid — Resource elements

3-D numeric array

Resource elements, returned as a 3-D numeric array. grid stores the resource elements for a number of subframes across all configured antenna ports. It is an *M*-by-*N*-by-*P* array, where *M* is the number of subcarriers, *N* is the number of OFDM symbols, and *P* is the number of antennas.

Data Types: double

# Version History

**Introduced in R2014a**

### R2022b: Specify FFT size

You can specify the FFT size by using the `Nfft` input.

## See Also

`lteOFDMModulate` | `lteOFDMInfo` | `lteDLFrameOffset` | `lteDLChannelEstimate` | `lteDLPerfectChannelEstimate`

# lteOFDMModulate

OFDM modulation

## Syntax

```
[waveform,info] = lteOFDMModulate(enb,grid)
[waveform,info] = lteOFDMModulate(enb,grid,windowing)
[waveform,info] = lteOFDMModulate(enb,grid,windowing,Nfft)
```

## Description

`[waveform,info] = lteOFDMModulate(enb,grid)` performs DC subcarrier insertion, inverse fast Fourier transform (IFFT) calculation, cyclic prefix insertion, and optional raised cosine windowing and overlapping of adjacent OFDM symbols of the complex symbols in the resource array, `grid`. `grid` is a 3-D array containing the resource elements (REs) for a number of subframes across all configured antenna ports, as described in “Represent Resource Grids”. It could also be multiple concatenated matrices to give multiple subframes, using concatenation across the columns or second dimension. The antenna planes in `grid` are each OFDM modulated to yield the columns of the output `waveform`.

`grid` can span multiple subframes. Windowing and overlapping are applied between all adjacent OFDM symbols, including the last of one subframe and the first of the next. Therefore, a different result is obtained than if `lteOFDMModulate` is called on individual subframes and then those time-domain waveforms are concatenated. In that case, the resulting waveform has discontinuities at the start or end of each subframe. It is recommended that all subframes for OFDM modulation first be concatenated before calling `lteOFDMModulate` on the resulting multi-subframe array. However, individual subframes can be OFDM modulated and the resulting multi-subframe time-domain waveform created by manual overlapping.

`[waveform,info] = lteOFDMModulate(enb,grid,windowing)` allows control of the number of windowed and overlapped samples used in the time-domain windowing, specified by the `windowing` parameter. The value of `enb.Windowing`, if present, is ignored, and the output, `info.Windowing` is set to `windowing`.

`[waveform,info] = lteOFDMModulate(enb,grid,windowing,Nfft)` specifies the number of IFFT points to use in the modulation.

## Examples

### Perform OFDM Modulation

Perform OFDM modulation of one subframe of random uniformly distributed noise using a 10 MHz two-antenna configuration.

```
enb = struct('NDLRB',50,'CyclicPrefix','Normal','CellRefP',2);
dims = lteDLResourceGridSize(enb);
regrid = reshape(lteSymbolModulate(randi([0,1],prod(dims)*2,1), ...
    'QPSK'),dims);
waveform = lteOFDMModulate(enb,regrid);
```

## Input Arguments

### `enb` — Cell-wide settings

structure

Cell-wide settings, specified as a structure. `enb` can contain the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`CyclicPrefix`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **`Windowing`** | Optional | Nonnegative integer | Number of time-domain samples over which the function applies windowing and overlapping of OFDM symbols<br><br>See note |

**Note** If `enb.Windowing` is absent, a default value for the number of windowed and overlapped samples is used. The default value is chosen as a function of *NRB* to compromise between the effective duration of cyclic prefix, and thus the channel delay spread tolerance, and the spectral characteristics of the transmitted signal, not considering any additional FIR filtering. The value used is returned in `info.Windowing`. The issues concerning concatenation of subframes before OFDM modulation do not apply when `enb.Windowing` is zero.

Data Types: `struct`

### `grid` — Resource elements

3-D numeric array

Resource elements, specified as a 3-D numeric array. `grid` stores the resource elements for a number of subframes across all configured antenna ports. `grid` is an *M*-by-*N*-by-*P* array, where *M* is the number of subcarriers, *N* is the number of OFDM symbols, and *P* is the number of antennas.

*M* must be a multiple of 12 REs per Resource Block, since number of resource blocks is *NRB* =*M* / 12. *N* must be a multiple of the number of symbols in a subframe, *L*, where *L* is 14 for normal cyclic prefix and 12 for extended cyclic prefix.

Data Types: `double`
Complex Number Support: Yes

### `windowing` — OFDM sample span

nonnegative integer

Number of time-domain samples over which the function applies windowing and overlapping of OFDM symbols, specified as a nonnegative integer. This value overwrites the value of the parameter field `enb.Windowing`, if present.

Data Types: `double`

### `Nfft` — Number of IFFT points

positive integer

The number of IFFT points to use in the OFDM modulation, specified as a positive integer.

Data Types: `double`

## Output Arguments

### `waveform` — OFDM modulated waveform

numeric matrix

OFDM modulated waveform, returned as a numeric matrix of size *T*-by-*P*, where *P* is the number of antennas and *T* is the number of time-domain samples. $T = K \times 30720 / 2048 \times N_{\text{fft}}$ where $N_{\text{fft}}$ is the IFFT size and *K* is the number of subframes in the input `grid`. $N_{\text{fft}}$ is a function of the number of resource blocks (*NRB*), as shown in the following table.

| NRB | $N_{\text{fft}}$ |
|---|---|
| 6 | 128 |
| 15 | 256 |
| 25 | 512 |
| 50 | 1024 |
| 75 | 2048 |
| 100 | 2048 |

In general, $N_{\text{fft}}$ is the smallest power of 2 greater than or equal to 12×NRB/0.85. It is the smallest FFT that spans all subcarriers and results in a bandwidth occupancy, 12×NRB/0.85, of no more than 85%.

Data Types: `double`
Complex Number Support: Yes

### `info` — OFDM modulated waveform information

structure

OFDM modulated waveform information, returned as a structure. `info` contains the following fields.

#### `SamplingRate` — Time-domain waveform sampling rate

scalar

Time-domain waveform sampling rate, returned as a scalar.
`SamplingRate` = 30.72 MHz / 2048 × $N_{\text{fft}}$.

Data Types: `double`

#### `Nfft` — Number of FFT points

scalar power of 2

Number of FFT points, returned as a scalar power of 2. `Nfft` is the smallest power of 2 greater than or equal to 12 × `NRB` / 0.85. It is the smallest FFT that spans all subcarriers and results in a bandwidth occupancy (12 × `NRB` / $N_{\text{fft}}$) of no more than 85%.

Data Types: `uint32`

#### `Windowing` — OFDM sample span

nonnegative integer

Number of time-domain samples over which the function applies windowing and overlapping of OFDM symbols, returned as a nonnegative integer.

Data Types: `int32`

**`CyclicPrefixLengths` — Cyclic prefix length**
even integer

Cyclic prefix length (in samples) of each OFDM symbol in a subframe.

| `info.Nfft` | `CyclicPrefixLengths` | |
|---|---|---|
| | **for `CyclicPrefix = 'Normal'`** | **for `CyclicPrefix = 'Extended'`** |
| 2048 | [160 144 144 144 144 144 144 160 144 144 144 144 144 144] | [512 512 512 512 512 512 512 512 512 512 512 512] |
| 1024 | [80 72 72 72 72 72 72 80 72 72 72 72 72 72] | [256 256 256 256 256 256 256 256 256 256 256 256] |
| 512 | [40 36 36 36 36 36 36 40 36 36 36 36 36 36] | [128 128 128 128 128 128 128 128 128 128 128 128] |
| 256 | [20 18 18 18 18 18 18 20 18 18 18 18 18 18] | [64 64 64 64 64 64 64 64 64 64 64 64] |
| 128 | [10 9 9 9 9 9 9 10 9 9 9 9 9 9] | [32 32 32 32 32 32 32 32 32 32 32 32] |

**Note** For `info.Nfft` < 2048, `info.CyclicPrefixLengths` are the `CyclicPrefixLengths` for `info.Nfft` = 2048 scaled by `info.Nfft` / 2048.

Data Types: `uint32`

Data Types: `struct`

## Algorithms

### Windowing

The use of the IFFT within the OFDM modulator constitutes the use of a rectangular pulse shape. This use of the IFFT means that discontinuities occur from one OFDM symbol to the next, resulting in out of band emissions. (Alternatively, considering the frequency domain, the frequency response of this rectangular pulse shape is a sinc pulse.) The discontinuities between OFDM symbols can be reduced by using windowing, which smooths the transitions between OFDM symbols. LTE Toolbox performs windowing by following this procedure.

For *Windowing* = $N$ samples, the cyclic prefix added to the nominal OFDM symbol extends by $N$ additional samples.

This extended waveform is windowed by pointwise multiplication in the time domain with a raised cosine window, which applies a taper to the first $N$ and last $N$ samples, with all other values being 1. The $y$ values in the first $N$ samples are:

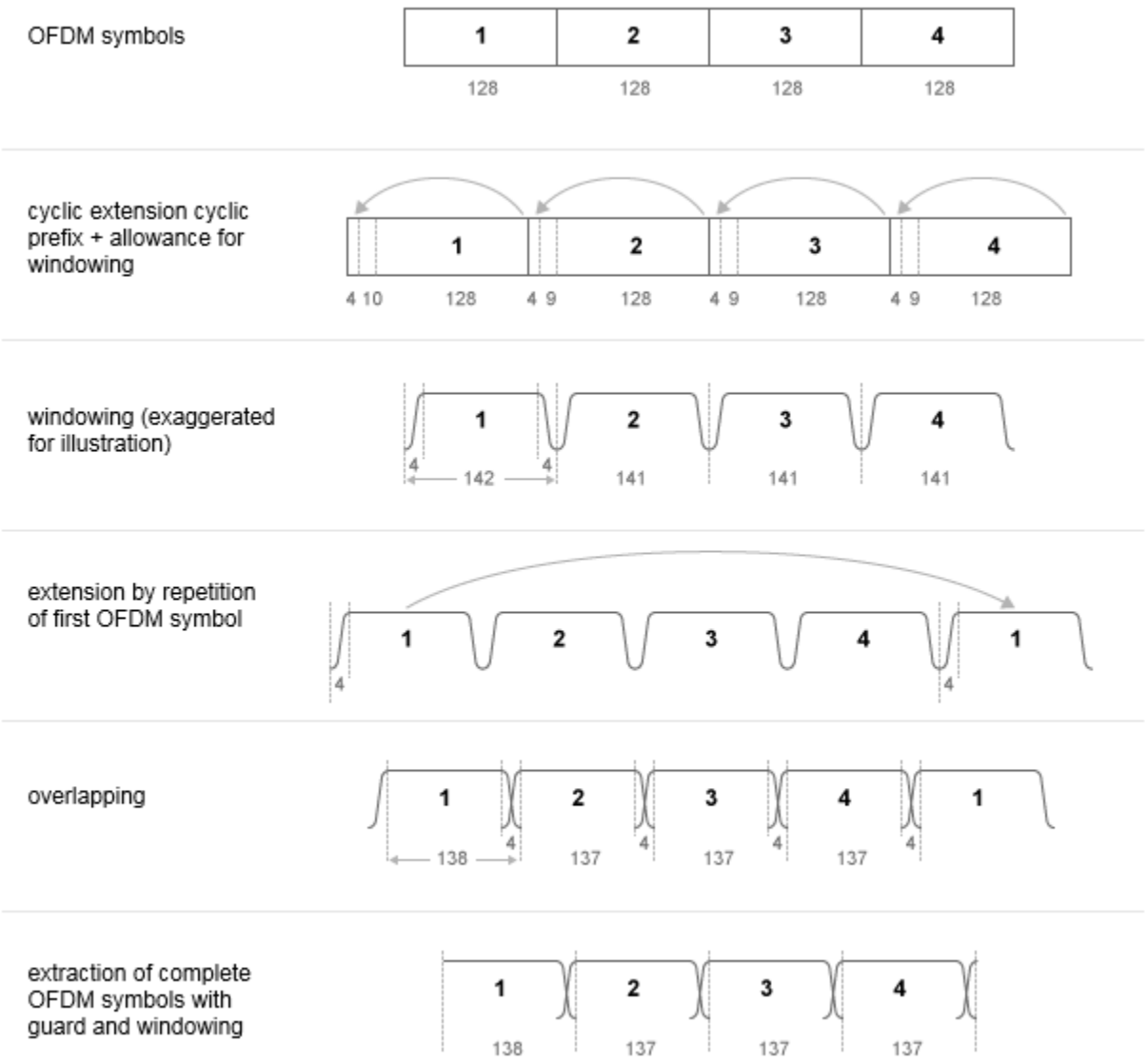
$$y = \frac{1}{2}\left(1 - \sin\left(\pi\frac{N + 1 - 2i}{2N}\right)\right), \text{ where } i = 1 \ldots N$$

The values in the last *N* samples are the same values in reverse order.

The windowed OFDM symbols are then overlapped by commencing transmission of each windowed OFDM symbol *N* samples before the end of the previous OFDM symbol. This overlapping ensures that the time between OFDM symbols is maintained as required by the standard. The taper at the start of the first OFDM symbol for transmission is removed and is overlapped with the taper at the end of the last OFDM symbol.

### Processing

The processing performed by this function is illustrated in this diagram.



The number of samples used for windowing depends on the number of resource blocks (NRB) and whether the cyclic prefix length is normal or extended. The number of samples is chosen in accordance with the *maximum* values implied by TS 36.101 [1], Tables F.5.3-1 and F.5.4-1.

| NRB | Windowing Samples for Normal Cyclic Prefix | Windowing Samples for Extended Cyclic Prefix |
|---|---|---|
| 6 | 4 | 4 |
| 15 | 6 | 6 |
| 25 | 4 | 4 |
| 50 | 6 | 6 |
| 75 | 8 | 8 |
| 100 | 8 | 8 |

The number of windowing samples is a compromise between the effective duration of cyclic prefix, and therefore the channel delay spread tolerance, and the spectral characteristics of the transmitted signal, not considering any additional FIR filtering. For a larger amount of windowing, the effective duration of the cyclic prefix is reduced but the transmitted signal spectrum has smaller out-of-band emissions.

# Version History

**Introduced in R2014a**

**R2022b: Specify IFFT size**

You can specify the IFFT size by using the `Nfft` input.

## References

[1] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteOFDMDemodulate` | `lteOFDMInfo` | `lteDLResourceGrid` | `lteFadingChannel` | `lteHSTChannel` | `lteMovingChannel`

# lteOFDMInfo

OFDM modulation related information

## Syntax

```
info = lteOFDMInfo(enb)
info = lteOFDMInfo(enb,Nfft)
```

## Description

`info = lteOFDMInfo(enb)` provides information related to the OFDM modulation performed by `lteOFDMModulate`, given the cell-wide settings structure, `enb`.

`info = lteOFDMInfo(enb,Nfft)` specifies the number of IFFT points to use in the modulation.

## Examples

### Get Information Related to OFDM Modulation

Find the sampling rate of a 50 resource block configuration, corresponding to a 10 MHz waveform after OFDM modulation.

```
enb = struct('NDLRB',50,'CyclicPrefix','Normal');
lteOFDMInfo(enb)
```

```
ans = struct with fields:
           SamplingRate: 15360000
                   Nfft: 1024
              Windowing: 6
    CyclicPrefixLengths: [80 72 72 72 72 72 72 80 72 72 72 72 72 72]
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure. `enb` contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NDLRB`** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **`CyclicPrefix`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`Windowing`** | Optional | Nonnegative integer | Number of time-domain samples over which the function applies windowing and overlapping of OFDM symbols |

Data Types: `struct`

**`Nfft` — Number of IFFT points**
positive integer

The number of IFFT points to use in the OFDM modulation, specified as a positive integer.

## Output Arguments

**`info` — OFDM information**
structure

OFDM information, returned as a structure. `info` contains the following fields.

**`SamplingRate` — Sampling rate of the OFDM modulator**
integer

Sampling rate of the OFDM modulator, returned as an integer.

Data Types: `double`

**`Nfft` — Number of FFT points**
scalar power of 2

Number of FFT points used in the OFDM modulator, returned as a scalar power of 2.

Data Types: `uint32`

**`Windowing` — OFDM sample span**
nonnegative integer

Number of time-domain samples over which the function applies windowing and overlapping of OFDM symbols, returned as a nonnegative integer.

If `enb.Windowing` is absent, `info.Windowing` returns a default value chosen as a function of `enb.NDLRB` to compromise between the effective duration of cyclic prefix (and therefore the channel delay spread tolerance) and the spectral characteristics of the transmitted signal (not considering any additional FIR filtering). See `lteOFDMModulate` for details.

Data Types: `int32`

**`CyclicPrefixLengths` — Cyclic prefix length**
even integer

Cyclic prefix length (in samples) of each OFDM symbol in a subframe.

| info.Nfft | CyclicPrefixLengths | |
|---|---|---|
| | for CyclicPrefix = 'Normal' | for CyclicPrefix = 'Extended' |
| 2048 | [160 144 144 144 144 144 144 160 144 144 144 144 144 144] | [512 512 512 512 512 512 512 512 512 512 512 512] |
| 1024 | [80 72 72 72 72 72 72 80 72 72 72 72 72 72] | [256 256 256 256 256 256 256 256 256 256 256 256] |
| 512 | [40 36 36 36 36 36 36 40 36 36 36 36 36 36] | [128 128 128 128 128 128 128 128 128 128 128 128] |
| 256 | [20 18 18 18 18 18 18 20 18 18 18 18 18 18] | [64 64 64 64 64 64 64 64 64 64 64 64] |
| 128 | [10 9 9 9 9 9 9 10 9 9 9 9 9 9] | [32 32 32 32 32 32 32 32 32 32 32 32] |

**Note** For `info.Nfft` < 2048, `info.CyclicPrefixLengths` are the `CyclicPrefixLengths` for `info.Nfft` = 2048 scaled by `info.Nfft` / 2048.

Data Types: `uint32`

Data Types: `struct`

# Version History

**Introduced in R2014a**

### R2022b: Specify IFFT size

You can specify the IFFT size by using the `Nfft` input.

## See Also

`lteOFDMModulate` | `lteDLResourceGridSize`

# ltePBCH

Physical broadcast channel

## Syntax

```
sym = ltePBCH(enb,cw)
```

## Description

`sym = ltePBCH(enb,cw)` returns a matrix containing the complex symbols of the Physical Broadcast Channel (PBCH) for cell-wide settings structure, `enb`, and codeword, `cw`. The function performs all physical channel processing steps, including the stages of scrambling, QPSK modulation, layer mapping, and precoding as defined in TS 36.211 [1], Section 6.6.

The BCH transport channel consumes information bits every 40 ms. The coded transport block is then passed to PBCH for physical channel processing. The PBCH is transmitted in the first subframe of every frame, so four successive frames are required to transmit one transport block. As the scrambling sequence is initialized at the boundary of every 40 ms, this function expects 40 ms worth of data. For example, it expects 1920 bits for normal cyclic prefix, or 1728 bits for extended cyclic prefix. Demultiplex the output of this function into quarter length blocks for transmission on the first subframe in each 10 ms frame.

## Examples

### Generate PBCH Symbols

Generate physical broadcast channel (PBCH) symbols using the MIB.

Create cell-wide configuration structure initialized to RMC R.0. Generate the MIB. Pass the MIB through broadcast channel (BCH) transport channel coding.

```
enb = lteRMCDL('R.0');
mib = lteMIB(enb);
bchCoded = lteBCH(enb,mib);
```

Generate and display the PBCH symbols.

```
pbchSymbols = ltePBCH(enb,bchCoded);
pbchSymbols(1:10)
```

```
ans = 10×1 complex

   0.7071 + 0.7071i
   0.7071 - 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
```

```
-0.7071 + 0.7071i
-0.7071 - 0.7071i
```

## Input Arguments

### `enb` — Cell-wide settings

structure

Cell-wide settings, specified as a structure. `enb` must contain the following fields.

#### `NCellID` — Physical layer cell identity

scalar integer

Physical layer cell identity, specified as a scalar integer.

Data Types: `double`

#### `CellRefP` — Number of cell-specific reference signal antenna ports

1 | 2 | 4

Number of cell-specific reference signal antenna ports, specified as 1, 2, or 4.

Data Types: `double`

Data Types: `struct`

### `cw` — PBCH codeword

vector

PBCH codeword, specified as a vector. `cw` contains the bit values of the PBCH codeword for modulation.

Data Types: `double` | `single` | `uint8` | `uint16` | `uint32` | `uint64` | `int8` | `int16` | `int32` | `int64`

## Output Arguments

### `sym` — PBCH symbols

numeric matrix

PBCH symbols, returned as a numeric matrix. `sym` contains the complex symbols of the Physical Broadcast Channel (PBCH) for cell-wide settings, `enb`, and codeword, `cw`. Its size is *N*-by-`CellRefP`, where *N* is the number of modulation symbols for one antenna port and `CellRefP` is the number of antenna ports.

Data Types: `double`
Complex Number Support: Yes

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePBCHDecode` | `ltePBCHIndices` | `ltePBCHPRBS` | `lteBCH`

# ltePBCHDecode

Physical broadcast channel decoding

## Syntax

```
[bits,symbols,nfmod4,trblk,cellrefp] = ltePBCHDecode(enb,sym)
[bits,symbols,nfmod4,trblk,cellrefp] = ltePBCHDecode(enb,sym,hest,noiseest)
[bits,symbols,nfmod4,trblk,cellrefp] = ltePBCHDecode(enb,sym,hest,noiseest,
alg)
```

## Description

`[bits,symbols,nfmod4,trblk,cellrefp] = ltePBCHDecode(enb,sym)` returns `bits`, a vector of soft bits, `symbols`, a vector of received constellation complex symbols, `nfmod4`, frame number (modulo 4), `trblk`, decoded BCH information bits, and `cellrefp`, the number of cell-specific reference signal antenna ports. For more information, see “PBCH Decoding” on page 2-545.

`[bits,symbols,nfmod4,trblk,cellrefp] = ltePBCHDecode(enb,sym,hest,noiseest)` decodes `sym`, the complex PBCH symbols, using cell-wide settings `enb`, channel estimate `hest`, and noise estimate `noiseest`. For more information, see “PBCH Decoding” on page 2-545.

`[bits,symbols,nfmod4,trblk,cellrefp] = ltePBCHDecode(enb,sym,hest,noiseest, alg)` provides control over weighting `bits`, with channel state information (CSI) calculated during the equalization stage using the algorithmic configuration structure `alg`. For more information, see “PBCH Decoding” on page 2-545.

## Examples

### Decode CellRefP from MIB

This example shows how to decode the number of cell-specific reference ports from the MIB.

Initialize a cell-wide configuration structure with RMC R.14. Generate the MIB and the broadcast channel bits.

```
enb = lteRMCDL('R.14');
mib = lteMIB(enb);
bchBits = lteBCH(enb,mib);
```

The `lteBCH` function generates bits for a 40 ms period, intended for 4 frames. Since the PBCH is transmitted every frame, encode and transmit only one quarter of these bits each frame.

```
quarterLen = length(bchBits)/4;
```

Encode the PBCH for a single frame by mapping and encoding of one quarter of the BCH to PBCH.

```
pbchSymbols = ltePBCH(enb,bchBits(1:quarterLen));
```

Decode the PBCH symbols.

```
[bits,symbols,nfmod4,trblk,cellrefp] = ltePBCHDecode(enb,pbchSymbols);
```

Check that the number of cell-specific reference ports matches number of antenna ports specified in TS 36.101 Annex 3.3.2 for RMC R.14.

```
cellrefp
```

```
cellrefp = uint32
    4
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure. `enb` contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NCellID`** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **`CellRefP`** | Optional | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports<br><br>The default is to establish `cellrefp` by decoding the input symbols, `sym`. |
| **`CyclicPrefix`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |

Data Types: `struct`

### sym — Complex modulated PBCH symbols

numeric matrix

Complex modulated PBCH symbols, specified as an `NRE`-by-`NRxAnts` numeric matrix. `NRE` is the number of QPSK symbols per antenna assigned to the PBCH and `NRxAnts` is the number of receive antennas. This input can contain 1–4 subframes of PBCH data. When you provide multiple subframes, they must be consecutive subframes within the same coded BCH block.

Data Types: `double`
Complex Number Support: Yes

### hest — Channel estimate

3-D array

Channel estimate is a 3-D array of size `NRE`-by-`NRxAnts`-by-*P*, where

- `NRE` is the number of PBCH resource elements (frequency and time locations).
- `NRxAnts` is the number of receive antennas.
- *P* is the number of cell-specific reference signal antennas.

Data Types: `double`

Complex Number Support: Yes

**`noiseest` — Noise estimate**
numeric scalar

Noise estimate, specified as a numeric scalar. It is an estimate of the noise power spectral density per resource element on the received subframe. This estimate is provided by the `lteDLChannelEstimate` function.

Data Types: `double`

**`alg` — Algorithmic configuration**
structure

Algorithmic configuration, specified as a structure. The structure must have the following field.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`CSI`** | Optional | `'On'` (default), `'Off'` | Flag provides control over weighting the soft values that are used to determine the output values with the channel state information (CSI) calculated during the equalization process. If `'On'`, soft values are weighted by CSI. |

Data Types: `struct`

## Output Arguments

**`bits` — Decoded PBCH soft bits**
real-valued column vector

Decoded PBCH soft bits, returned as a real-valued column vector. If `alg.CSI` is `'On'`, `bits` gets scaled by channel state information (CSI) calculated during the equalization process.

Data Types: `double`

**`symbols` — Received constellation of complex symbols**
complex-valued column vector

Received constellation of complex symbols, returned as a complex-valued column vector.

Data Types: `double`

**`nfmod4` — System frame number modulo 4**
nonnegative integer

System frame number modulo 4, `mod(NFrame,4)`, returned as a nonnegative integer. `nfmod4` is obtained when determining the scrambling phase of the input PBCH symbols, `sym`.

Data Types: `double`

**`trblk` — Decoded BCH information bits**
24-by-1 real-valued column vector

Decoded BCH information bits, returned as a 24-by-1 real-valued column vector.

Data Types: `int8`

**`cellrefp` — Number of CRS antenna ports**
`0` | `1` | `2` | `4`

Number of cell-specific signal (CRS) antenna ports, returned as `0`, `1`, `2`, or `4`. A value of `0` indicates that the function detects a cyclic redundancy check (CRC) error during the decoding process.

Data Types: `uint32`

## More About

### PBCH Decoding

TS 36.211 [1], Section 6.6 defines the inverse of Physical Broadcast Channel (PBCH) processing of `bits` and `symbols`. TS 36.212 [2], Section 5.3.1 defines the inverse Broadcast Channel (BCH) processing used to decode `nfmod4`, `trblk`, and `cellrefp`.

PBCH Decoding performs the inverse of PBCH processing (deprecoding, symbol demodulation, and descrambling) on the matrix of complex modulated PBCH symbols, `sym`, given a cell-wide settings structure, `enb`. It decodes PBCH data scrambled with any scrambling sequence phase. So although the scrambling sequence gets initialized every 40 ms, there is no restriction on the input `sym` to be aligned at the 40 ms boundary.

After successful synchronization with the scrambling sequence, `nfmod4`, `trblk`, and `cellrefp` are determined. The true number of transmitted cell-specific reference signals is returned in `cellrefp`, and is searched for by attempting decoding with `cellrefp` equal to 1, 2, or 4. If provided,`enb.CellRefP` is attempted first to ensure that `symbols` contains the expected constellation and `bits` contains the expected soft bit estimates for the specified value. Under good conditions, successful decoding is possible with a different value of `cellrefp`, but results in unexpected `bits` and `symbols`. If `enb.CellRefP` is not provided, the search establishes the true number of transmitted cell-specific reference signals and returns it in `cellrefp`.

For the `TxDiversity` transmission scheme (`cellrefp` = 2 or `cellrefp` = 4), the reception is performed using an OSFBC (Orthogonal Space Frequency Block Code) decoder. For the `Port0` transmission scheme (`cellrefp` = 1), the reception is performed using MMSE equalization.

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[3] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePBCH` | `ltePBCHIndices` | `ltePBCHPRBS` | `lteBCHDecode`

# ltePBCHIndices

PBCH resource element indices

## Syntax

```
ind = ltePBCHIndices(enb)
ind = ltePBCHIndices(enb,opts)
```

## Description

`ind = ltePBCHIndices(enb)` returns an *N*-by-`CellRefP` matrix of resource element (RE) indices for the Physical Broadcast Channel (PBCH) given the parameter fields of structure `enb`. By default, the indices are returned in 1-based linear indexing form that can directly index elements of a 3-D array representing the subframe resource grid for `CellRefP` antennas. These indices are ordered as the PBCH modulation symbols should be mapped. Alternative indexing formats can also be generated. The PBCH is only transmitted in first subframe of each frame.

`ind = ltePBCHIndices(enb,opts)` formats the returned indices using options defined by `opts`.

## Examples

### Generate PBCH RE Indices

Generate zero-based PBCH resource element indices in linear form for RMC R.14.

```
enb = lteRMCDL('R.14');
ind = ltePBCHIndices(enb,{'0based'});
ind(1:4,:)
```

```
ans = 4x4 uint32 matrix

    4465   12865   21265   29665
    4466   12866   21266   29666
    4468   12868   21268   29668
    4469   12869   21269   29669
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure. `enb` contains the following fields.

### NDLRB — Number of downlink resource blocks

scalar value

Number of downlink resource blocks, specified as a scalar value.

**`NCellID` — Physical layer cell identity**
scalar integer

Physical layer cell identity, specified as a scalar integer.

**`CyclicPrefix` — Cyclic prefix length**
`'Normal'` (default) | optional | `'Extended'`

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

**`CellRefP` — Number of cell-specific reference signal antenna ports**
1 (default) | optional | 2 | 4

Number of cell-specific reference signal antenna ports, specified as 1, 2, or 4.

**`NSubframe` — Subframe number**
0 (default) | optional | scalar integer

Subframe number, specified as a scalar integer.

Data Types: `struct`

**`opts` — Output format options for resource element indices**
character vector | cell array of character vectors | string array

Output format options for resource element indices, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string) :

| Category | Options | Description |
|---|---|---|
| Indexing style | `'ind'` (default) | The returned indices are in linear index style. |
| | `'sub'` | The returned indices are in `[subcarrier,symbol,port]` subscript row style. |
| Index base | `'1based'` (default) | The returned indices are one-based. |
| | `'0based'` | The returned indices are zero-based. |

Example: `'ind 1based'`, `"ind 1based"`, `{'ind','1based'}`, or `["ind","1based"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

**`ind` — PBCH resource element indices**
numeric matrix

PBCH resource element indices, returned as a numeric matrix of size *N*-by-`CellRefP`.

Data Types: `double`

## Version History

**Introduced in R2014a**

## See Also

`ltePBCH` | `ltePBCHDecode` | `ltePBCHPRBS`

# ltePBCHPRBS

PBCH pseudorandom scrambling sequence

## Syntax

```
[seq,cinit] = ltePBCHPRBS(enb,n)
[seq,cinit] = ltePBCHPRBS(enb,n,mapping)

[subseq,cinit] = ltePBCHPRBS(enb,pn)
[subseq,cinit] = ltePBCHPRBS(enb,pn,mapping)
```

## Description

`[seq,cinit] = ltePBCHPRBS(enb,n)` returns a vector with the first `n` outputs of the Physical Broadcast Channel (PBCH) scrambling sequence when initialized with the structure `enb`. It also returns an initialization value `cinit` for the pseudorandom binary sequence (PRBS) generator.

`[seq,cinit] = ltePBCHPRBS(enb,n,mapping)` allows control over the format of the returned sequence, `seq`, with the input `mapping`.

`[subseq,cinit] = ltePBCHPRBS(enb,pn)` returns a subsequence of a full PRBS sequence, specified by `pn`.

`[subseq,cinit] = ltePBCHPRBS(enb,pn,mapping)` allows control over the format of the returned subsequence, `subseq`, with the input `mapping`.

## Examples

### Scramble Broadcast Channel MIB Message

Scramble the MasterInformationBlock broadcast channel (BCCH) message.

Create a cell-wide configuration structure initialized to RMC R.0. Generate the MIB and coded BCH.

```
enb = lteRMCDL('R.0');
mib = lteMIB(enb);
bchCoded = lteBCH(enb,mib);
```

Generate the required length of the PBCH scrambling sequence. Scramble the coded BCH.

```
pbchPrbsSeq = ltePBCHPRBS(enb,length(bchCoded));
scrambled = xor(pbchPrbsSeq, bchCoded);
```

### Compare Pseudorandom Scrambling Sequences

Compare the PBCH scrambling sequence generated using both generic and PBCH-specific pseudorandom binary sequence generators.

Create a cell-wide configuration structure initialized to RMC R.0. Generate the first 25 bits of the pseudorandom binary sequence for physical layer cell identity, `NCellID` using `ltePRBS` and `ltePBCHPRBS`.

```
enb = lteRMCDL('R.0');
prbsSeq = ltePRBS(enb.NCellID, 25);
pbchPrbsSeq = ltePBCHPRBS(enb,25);
isequal(prbsSeq,pbchPrbsSeq)
```

```
ans = logical
   1
```

The generic pseudorandom binary scrambling sequence equals the PBCH-specific pseudorandom binary scrambling sequence.

## Input Arguments

### `enb` — Cell-wide settings

structure

Cell-wide settings, specified as a structure. `enb` must contain the following field.

#### `NCellID` — Physical layer cell identity

scalar integer

Physical layer cell identity, specified as a scalar integer.

Data Types: `double`

Data Types: `struct`

### `n` — Number of elements in returned sequence

numeric scalar

Number of outputs, specified as a numeric scalar.

Data Types: `double`

### `pn` — Range of elements in returned subsequence

row vector

Range of elements in returned subsequence, `subseq`, specified as a row vector of `[p n]`. The subsequence returns `n` values of the PRBS generator, starting at position `p` (0-based).

Data Types: `double`

### `mapping` — Output sequence formatting

`'binary'` (default) | `'signed'`

Output sequence formatting, specified as `'binary'` or `'signed'`. `mapping` controls the format of the returned sequence.

- `'binary'` maps `true` to 1 and `false` to 0.
- `'signed'` maps `true` to –1 and `false` to 1.

Data Types: `char` | `string`

## Output Arguments

### `seq` — PBCH pseudorandom scrambling sequence

logical column vector | numeric column vector

PBCH pseudorandom scrambling sequence, specified as a logical column vector or a numeric column vector. `seq` contains the first `n` outputs of the physical broadcast channel (PBCH) scrambling sequence. If you set `mapping` to `'signed'`, the output data type is `double`. Otherwise, the output data type is `logical`.

Data Types: `logical` | `double`

### `subseq` — PBCH pseudorandom scrambling subsequence

logical column vector | numeric column vector

PBCH pseudorandom scrambling subsequence, specified as a logical column vector or a numeric column vector. `subseq` contains the values of the PRBS generator specified by `pn`. If you set `mapping` to `'signed'`, the output data type is `double`. Otherwise, the output data type is `logical`.

Data Types: `logical` | `double`

### `cinit` — Initialization value for PRBS generator

numeric scalar

Initialization value for PRBS generator, returned as a numeric scalar.

Data Types: `uint32`

# Version History

**Introduced in R2014a**

## See Also

`ltePBCH` | `ltePBCHIndices` | `ltePBCHDecode`

# ltePCFICH

Physical control format indicator channel

## Syntax

```
sym = ltePCFICH(enb,cw)
```

## Description

`sym = ltePCFICH(enb,cw)` returns the matrix of complex modulation symbols generated by the Physical Control Format Indicator Channel (PCFICH). The channel processing includes the stages of scrambling, QPSK modulation, layer mapping, and precoding. Given input bit vector `cw`, each column of the 16-by-`CellRefP` matrix `sym` contains the 16 QPSK symbols carried by the PCFICH on each of `CellRefP` transmit antenna ports. The channel is parameterized by structure `enb`.

The PCFICH is intended to carry the 32-bit block encoding of the CFI. For more information, see `lteCFI`. The channel expects the input bit vector, `cw`, to be 32 elements in length. If `length(cw) < 32`, `cw` is padded with zeros before channel processing. If `length(cw) > 32`, only the first 32 elements are used.

## Examples

### Generate PCFICH Symbols

Modulate CFI=1 onto two antenna ports (transmit diversity). The generated PCFICH symbols are stored in a matrix.

Generate PCFICH symbols, using a control format indicator (CFI) value of one and using two antenna ports for transmit diversity.

```
cfiCodeword = lteCFI(struct('CFI',1));
enb = struct('CellRefP',2,'NCellID',0,'NSubframe',0);

pcfichSymbols = ltePCFICH(enb,cfiCodeword);
sizePCFICHSymbols = size(pcfichSymbols)
```

```
sizePCFICHSymbols = 1×2

    16     2
```

Since two antenna ports were configured, there are two columns in the output matrix.

## Input Arguments

### enb — Cell-wide settings structure

scalar structure

`enb` is a structure having the following fields.

**`NCellID` — Physical layer cell identity**
0...503

Physical layer cell identity, specified as an integer from 0 through 503.

**`CellRefP` — Number of cell-specific reference signal (CRS) antenna ports**
1 (default) | 2 | 4

Number of cell-specific reference signal (CRS) antenna ports, specified as one of the set (1, 2, 4).

**`NSubframe` — Subframe number**
scalar

Subframe number, specified as an integer.

Data Types: `struct`

**`cw` — Input bit vector**
vector

Input bit vector that is 32 elements in length, specified as a vector. If `length(cw) < 32`, `cw` is padded with zeros before channel processing. If `length(cw) > 32`, only the first 32 elements are used.

Example: `cw = lteCFI(struct('CFI',1));`

Data Types: `int8`

## Output Arguments

**`sym` — Complex modulation symbols generated by the PCFICH**
numeric matrix

Complex modulation symbols generated by the PCFICH, returned as a numeric matrix of size 16-by-`CellRefP`. The channel processing includes the stages of scrambling, QPSK modulation, layer mapping, and precoding. Given input bit vector `cw`, each column of the 16-by-`CellRefP` matrix `sym` contains the 16 QPSK symbols carried by the PCFICH on each of the `CellRefP` transmit antenna ports. The channel is parameterized by structure `enb`.

Data Types: `double`
Complex Number Support: Yes

# Version History

**Introduced in R2014a**

## See Also

`ltePCFICHDecode` | `ltePCFICHInfo` | `ltePCFICHIndices` | `ltePCFICHPRBS` | `lteCFI`

# ltePCFICHDecode

Physical control format indicator channel decoding

## Syntax

```
[bits,symbols] = ltePCFICHDecode(enb,sym)
[bits,symbols] = ltePCFICHDecode(enb,sym,hest,noiseest)
[bits,symbols] = ltePCFICHDecode(enb,sym,hest,noiseest,alg)
```

## Description

`[bits,symbols] = ltePCFICHDecode(enb,sym)` performs the inverse of Physical Control Format Indicator Channel (PCFICH) processing on the matrix of complex modulated PCFICH symbols, `sym`, using cell-wide settings structure, `enb`. It returns a column vector of soft bits, `bits`, and received constellation of complex symbol vector, `symbols`. The channel inverse processing includes deprecoding, symbol demodulation, and descrambling. See TS 36.211, Section 6.7 [1] or `ltePCFICH` for details.

The input argument, `sym`, must be a matrix of `NRE`-by-`NRxAnts` complex modulated PCFICH symbols. `NRE` is the number of QPSK symbols per antenna assigned to the PCFICH (16) and `NRxAnts` is the number of receive antennas.

`[bits,symbols] = ltePCFICHDecode(enb,sym,hest,noiseest)` decodes the complex PCFICH symbols, `sym`, using cell-wide settings, `enb`, the channel estimate, `hest`, and the noise estimate, `noiseest`. For the `'TxDiversity'` transmission scheme, when `CellRefP` is 2 or 4, the reception is performed using an orthogonal space frequency block code (OSFBC) decoder. For the `'Port0'` transmission scheme, when `CellRefP` is 1, the reception is performed using MMSE equalization.

`hest` is a 3-D `NRE`-by-`NRxAnts`-by-`enb.CellRefP` array. `NRE` contains the frequency and time locations corresponding to the PCFICH RE positions for a total of `NRE` positions. `NRxAnts` is the number of receive antennas, and `enb.CellRefP` is the number of cell-specific reference signal antennas.

`noiseest` is an estimate of the noise power spectral density per RE in the received subframe. The `lteDLChannelEstimate` function produces this estimate.

`[bits,symbols] = ltePCFICHDecode(enb,sym,hest,noiseest,alg)` same as prior except this syntax provides control over weighting the output soft bits, `bits`. If `alg.CSI` is `'On'`, `bits` get scaled by the channel state information (CSI) calculated during the equalization stage.

## Examples

### Decode PCFICH Symbols

This example shows decoding of symbols to recover CFI value.

Initialize a cell wide configuration structure, `enb`. Encode a CFI value and perform physical channel coding to create a vector of symbols, `pcfichSym`.

```
enb.NCellID = 0;
enb.NSubframe = 0;
enb.CellRefP = 1;
enb.CFI = 3;
cw = lteCFI(enb);
pcfichSym = ltePCFICH(enb,cw);
```

Demodulate and decode the symbols to recover the CFI value

```
cfiSoftBits = ltePCFICHDecode(enb,pcfichSym);
rxCFI = lteCFIDecode(cfiSoftBits)
```

```
rxCFI = int32
    3
```

Confirm recovered CFI value matches the setting in `enb`

```
enb.CFI
```

```
ans = 3
```

## Input Arguments

### `enb` — Cell-wide settings

scalar structure

Cell-wide settings, specified as a scalar structure. `enb` contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NCellID`** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **`CellRefP`** | Required | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **`NSubframe`** | Required | 0 (default), nonnegative scalar integer | Subframe number |

Data Types: `struct`

### `sym` — Complex modulated PCFICH symbols

numeric matrix

Complex modulated PCFICH symbols, specified as a numeric matrix of size `NRE`-by-`NRxAnts`. `NRE` is the number of QPSK symbols per antenna assigned to the PCFICH (16). `NRxAnts` is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

### `hest` — Channel estimate

3-D numeric array

Channel estimate, specified as a 3-D numeric array of size `NRE`-by-`NRxAnts`-by-`enb.CellRefP`, where:

- `NRE` contains the frequency and time locations corresponding to the PCFICH RE positions (a total of `NRE` positions).
- `NRxAnts` is the number of receive antennas.
- `enb.CellRefP` is the number of cell-specific reference signal antennas.

Data Types: `double`
Complex Number Support: Yes

**`noiseest` — Noise estimate**
scalar

Estimate of the noise power spectral density per RE on received subframe. Such an estimate is provided by the `lteDLChannelEstimate` function.

Data Types: `double`

**`alg` — Algorithmic configuration**
structure

Algorithmic configuration, specified as a structure. It contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`CSI`** | Optional | `'On'` (default), `'Off'` | Flag provides control over weighting the soft values that are used to determine the output values with the channel state information (CSI) calculated during the equalization process. If `'On'`, soft values are weighted by CSI. |

Data Types: `struct`

## Output Arguments

**`bits` — Soft bits**
numeric column vector

Soft bits, returned as a numeric column vector. If the input `alg.CSI` field is `'On'`, `bits` gets scaled by channel state information (CSI) calculated during the equalization process.

Data Types: `double`

**`symbols` — Received constellation symbols**
complex numeric column vector

Received constellation symbols, returned as a complex numeric column vector.

Data Types: `double`
Complex Number Support: Yes

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePCFICH` | `ltePCFICHIndices` | `ltePCFICHInfo` | `ltePCFICHPRBS` | `lteCFIDecode`

# ltePCFICHIndices

PCFICH resource element indices

## Syntax

```
ind = ltePCFICHIndices(enb)
ind = ltePCFICHIndices(enb,opts)
```

## Description

`ind = ltePCFICHIndices(enb)` returns the 16-by-`CellRefP` matrix of subframe resource element (RE) indices for the physical control format indicator channel (PCFICH), given the `enb` input structure. By default, the indices are returned in 1-based linear indexing form that directly indexes elements of a 3-D array representing the subframe resource grid for `CellRefP` antenna ports. Each column of `ind` contains per-antenna indices for 16 resource elements in one of the `CellRefP` array planes. The rows are ordered as the PCFICH modulation symbols should be mapped. The indices can also be returned in a number of alternative indexing formats.

The PCFICH is always transmitted on 16 resource elements, or 4 resource element groups (REG), in the first OFDM symbol of a subframe however their locations depend on the `NCellID` and `NDLRB` parameters.

`ind = ltePCFICHIndices(enb,opts)` formats the returned indices using options specified by `opts`.

## Examples

### Generate PCFICH RE Indices

This example generates physical CFI channel (PCFICH) resource element (RE) indices for two different physical layer cell identity values.

To show the effects of the physical layer cell identity, `NCellID`, on the indices, first set it to `0`. Generate and display the PCFICH indices.

```
enb.NDLRB = 50;
enb.NCellID = 0;
enb.CyclicPrefix = 'Normal';
enb.CellRefP = 1;
ind = ltePCFICHIndices(enb,{'0based','reg'});
disp(ind)
```

```
     0
   150
   300
   450
```

Next, set the physical layer cell identity, `NCellID`, to `1`. Regenerate and display the PCFICH indices.

```
enb.NCellID = 1;
ind = ltePCFICHIndices(enb,{'0based','reg'});
disp(ind)
```

```
     6
   156
   306
   456
```

## Input Arguments

### enb — eNodeB cell-wide settings

scalar structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NDLRB`** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **`NCellID`** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **`CyclicPrefix`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **`CellRefP`** | Optional | 1 (default), 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |

### opts — Output format, base, and unit of generated indices

character vector | string scalar | cell array of character vectors | string array

Output format, base, and unit of generated indices, specified as one of these forms.

- `'format base unit'`
- `"format base unit"`
- `{'format','base','unit'}`
- `["format","base","unit"]`

Where `format`, `base`, and `unit` are defined in this table.

| Option | Values | Description |
|---|---|---|

| | | |
|---|---|---|
| **format** | `'ind'` (default), `'sub'` | Output format of generated indices<br><br>To return the indices as a column vector, specify this option as `'ind'`.<br><br>To return the indices as an $N_{RE}$-by-3 matrix, where $N_{RE}$ is the number of REs, specify this option as `'sub'`. Each row of the matrix contains the subcarrier, symbol, and antenna port as its first, second, and third element, respectively. |
| **base** | `'1based'` (default), `'0based'` | Index base<br><br>To generate indices whose first value is `1`, specify this option as `'1based'`. To generate indices whose first value is `0`, specify this option as `'0based'`. |
| **unit** | `'re'` (default), `'reg'` | Unit of returned indices<br><br>To indicate that the returned values correspond to individual resource elements (REs), specify this option as `'re'`. To indicate that the returned values correspond to resource element groups (REGs), specify this option as `'reg'`. |

Example: `'ind 0based reg'`, `"ind 0based reg"`, `{'ind','0based','reg'}`, and `["ind","0based","reg"]` specify the same output options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `ind` — Subframe PCFICH RE indices

numeric matrix

Subframe PCFICH RE indices, returned as a numeric matrix of size 16-by-`CellRefP`. Each column of `ind` contains per-antenna indices for 16 resource elements in one of the `CellRefP` array planes. The rows are ordered as the PCFICH modulation symbols should be mapped.

# Version History

**Introduced in R2014a**

## See Also

`ltePCFICH` | `ltePCFICHInfo` | `ltePHICHIndices` | `ltePDCCHIndices` | `lteDLResourceGrid`

# ltePCFICHInfo

PCFICH resource information

## Syntax

```
info = ltePCFICHInfo
```

## Description

`info = ltePCFICHInfo` returns a structure `info` containing the Physical Control Format Indicator Channel (PCFICH) subframe resources.

For the PCFICH, `NREG` = 4, and `NRE` = 16 = 4 × `NREG`. These values are fixed for the system.

## Examples

### Get PCFICH Resource Information

Display information about the PCFICH subframe resources.

```
info = ltePCFICHInfo
```

```
info = struct with fields:
    NREG: 4
     NRE: 16
```

## Output Arguments

### `info` — PCFICH resource information

scalar structure

PCFICH resource information, returned as a scalar structure. It can contain the following fields.

| Parameter Field | Description | Values | Data Type |
|---|---|---|---|
| **`NRE`** | Number of resource elements (REs) assigned to PCFICH (4×`NREG`) | Nonnegative scalar integer | `uint64` |
| **`NREG`** | Number of resource element groups (REGs) assigned to PCFICH | Nonnegative scalar integer | `uint64` |

# Version History

**Introduced in R2014a**

## See Also

`ltePCFICH` | `ltePCFICHIndices` | `ltePCFICHPRBS` | `ltePCFICHDecode`

# ltePCFICHPRBS

PCFICH pseudorandom scrambling sequence

## Syntax

```
[seq,cinit] = ltePCFICHPRBS(enb,n)
[seq,cinit] = ltePCFICHPRBS(enb,n,mapping)

[subseq,cinit] = ltePCFICHPRBS(enb,pn)
[subseq,cinit] = ltePCFICHPRBS(enb,pn,mapping)
```

## Description

`[seq,cinit] = ltePCFICHPRBS(enb,n)` returns a vector containing the first `n` outputs of the physical control format indicator channel (PCFICH) scrambling sequence when initialized according to cell-wide settings structure, `enb`. It also returns an initialization value `cinit` for the pseudorandom binary sequence (PRBS) generator.

`[seq,cinit] = ltePCFICHPRBS(enb,n,mapping)` allows control over the format of the returned sequence, `seq`, with the input `mapping`.

`[subseq,cinit] = ltePCFICHPRBS(enb,pn)` returns a subsequence of a full PRBS sequence, specified by `pn`.

`[subseq,cinit] = ltePCFICHPRBS(enb,pn,mapping)` allows additional control over the format of the returned subsequence, `subseq`, with the input `mapping`.

## Examples

### Scramble CFI Codeword

Scramble the CFI codeword.

Create cell-wide configuration structure, initialized to RMC R.14. Generate a CFI codeword and a pseudorandom scrambling sequence for the PCFICH the same length as the codeword.

```
enb = lteRMCDL('R.14');
cw = lteCFI(struct('CFI',2));
pcfichPrbsSeq = ltePCFICHPRBS(enb,length(cw));
size(pcfichPrbsSeq)
```

```
ans = 1×2

    32     1
```

Scramble the CFI codeword using the generated scrambling sequence.

```
scrambled = xor(pcfichPrbsSeq,cw);
```

### Generate Signed PCFICH Pseudorandom Scrambling Sequence

Generate a signed pseudorandom scrambling sequence for the PCFICH. Each resource element (RE) in the PCFICH is QPSK-modulated, resulting in two bits-per-symbol mapping on each resource element.

Create cell-wide configuration structure, initialized to RMC R.14.

```
enb = lteRMCDL('R.14');
info = ltePCFICHInfo
```

```
info = struct with fields:
    NREG: 4
     NRE: 16
```

```
pcfichPrbsSeq = ltePCFICHPRBS(enb,info.NRE*2,'signed');
size(pcfichPrbsSeq)
```

```
ans = 1×2

    32     1
```

```
pcfichPrbsSeq(1:10)
```

```
ans = 10×1

     1
    -1
     1
     1
     1
     1
     1
    -1
    -1
     1
```

The scrambling sequence contains a vector of 32 signed ones.

## Input Arguments

### enb — Cell-wide settings

scalar structure

Cell-wide settings, specified as a scalar structure. `enb` contains the following fields.

### `NCellID` — Physical layer cell identity

0...503

Physical layer cell identity, specified as a nonnegative scalar integer in the range of 0 to 503.

Data Types: `double`

**NSubframe — Subframe number**
positive scalar integer

Subframe number, specified as a positive scalar integer greater than 0.

Data Types: `double`

Data Types: `struct`

**n — Length of scrambling sequence**
positive scalar integer

Length of scrambling sequence, specified as a positive scalar integer. This argument determines the number of elements in the output vector, `seq`.

Data Types: `double`

**pn — Range of scrambling subsequence**
row vector

Range of scrambling subsequence, `subseq`, specified as a row vector of `[p n]`. The subsequence returns `n` values of the PRBS generator, starting at position `p` (0-based).

Data Types: `double`

**mapping — Output sequence formatting**
`'binary'` (default) | `'signed'`

Output sequence formatting, specified as `'binary'` or `'signed'`. `mapping` controls the format of the returned sequence.

- `'binary'` maps `true` to 1 and `false` to 0.
- `'signed'` maps `true` to –1 and `false` to 1.

Data Types: `char` | `string`

## Output Arguments

**seq — PCFICH pseudorandom scrambling sequence**
logical column vector | numeric column vector

PCFICH pseudorandom scrambling sequence, returned as a logical column vector or a numeric column vector. `seq` contains the first `n` outputs of the PCFICH scrambling sequence when initialized according to cell-wide settings structure, `enb`. If you set `mapping` to `'signed'`, the output data type is `double`. Otherwise, the output data type is `logical`.

Data Types: `logical` | `double`

**subseq — PCFICH pseudorandom scrambling subsequence**
logical column vector | numeric column vector

PCFICH pseudorandom scrambling subsequence, returned as a logical column vector or a numeric column vector. `subseq` contains the values of the PRBS generator specified by `pn`. If you set `mapping` to `'signed'`, the output data type is `double`. Otherwise, the output data type is `logical`.

Data Types: `logical` | `double`

**`cinit` — Initialization value for PRBS generator**
numeric scalar

Initialization value for PRBS generator, returned as a numeric scalar.

Data Types: `uint32`

# Version History

**Introduced in R2014a**

## See Also

`ltePCFICH` | `ltePCFICHIndices` | `ltePCFICHInfo` | `ltePCFICHDecode` | `ltePRBS`

# ltePDCCH

Physical downlink control channel

## Syntax

```
[sym,info] = ltePDCCH(enb,cw)
[sym,info] = ltePDCCH(enb,cw,NREG)
[sym,info] = ltePDCCH(enb,cw,NREG,CCEGAINS)
```

## Description

`[sym,info] = ltePDCCH(enb,cw)` returns an `NRE`-by-`CellRefP` complex matrix, `sym`, of modulation symbols given the input bit vector `cw`.

The function returns a matrix (`sym`) of complex modulation symbols generated by the set of Physical Downlink Control Channels (PDCCH) in a subframe. The channel processing includes the stages of scrambling, QPSK modulation, layer mapping and precoding, followed by REG interleaving and cyclic shifting. For a given input bit vector (typically the PDCCH multiplex), the output matrix `sym` contains the QPSK symbols in column-wise antenna form. Any input bits with value < 0 are turned into `<NIL>` (`'0'`) symbols. The optional structure `info` returns control resourcing information about the output symbols (see `ltePDCCHInfo` for details).

`[sym,info] = ltePDCCH(enb,cw,NREG)` returns matrix `sym`. sets the number of output QPSK symbols, `NRE`, based on the `NREG` input value (`NRE` = 4 × `NREG`) instead of calculating it from the parameters of the `enb` structure.

`[sym,info] = ltePDCCH(enb,cw,NREG,CCEGAINS)` returns matrix `sym`. `CCEGAINS` allows control of the QPSK symbol gains on a per control channel element (CCE) basis.

## Examples

### Generate PDCCH Symbols

Generate complex modulated symbols for the PDCCH. The PDCCH symbols are QPSK modulated. Each QPSK symbol represents two bits.

Create a cell-wide configuration structure, initialized for RMC R.0. Retrieve the PDCCH information.

```
enb = lteRMCDL('R.0');
pdcchInfo = ltePDCCHInfo(enb)
```

```
pdcchInfo = struct with fields:
        NREG: 113
         NRE: 452
        NCCE: 12
    NREGUsed: 108
     NREUsed: 432
        MTot: 904
    NSymbols: 3
```

The field `pdcch.MTot` indicates the maximum number of input bits that can be transmitted on the PDCCH.

Generate a codeword that is `MTot` bits long. Using the codeword, generate PDCCH symbols.

```
cw = randi([0,1],pdcchInfo.MTot,1);
[pdcchSym,info] = ltePDCCH(enb,cw);
numCodewordBits = length(cw)
```

```
numCodewordBits = 904
```

```
numPDCCHSymbols = length(pdcchSym)
```

```
numPDCCHSymbols = 452
```

Since there are two bits per symbol, the number of output PDCCH symbols is half length of the codeword bit stream.

## Input Arguments

### enb — Cell-wide settings

scalar structure

Cell-wide settings, specified as a scalar structure. `enb` contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`CellRefP`** | Required | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **`NCellID`** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **`NSubframe`** | Required | Integer greater than 0 | Subframe number |
| **`NDLRB`** | Required | Nonnegative scalar integer (6,...,110) | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **`CyclicPrefix`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **`CFI`** | Required | 1, 2, or 3 | Control format indicator value |
| **`Ng`** | Required | `'Sixth'`, `'Half'`, `'One'`, `'Two'` | HICH group multiplier |
| **`DuplexMode`** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| The following field is required when `DuplexMode` is set to `'TDD'`. | | | |
| **`TDDConfig`** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |

Data Types: `struct`

### cw — Input bit vector

vector

Input bit vector that is 32 elements in length, specified as a vector. If `length(cw) < 32`, `cw` is padded with zeros before channel processing. If `length(cw) > 32`, only the first 32 elements are used.

Example: `cw = lteCFI(struct('CFI',1));`

Data Types: `int8`

**`NREG` — Resource element groups (REGs) assigned to PDCCH**
scalar

Resource element groups (REGs) assigned to PDCCH.

**`CCEGAINS` — Vector that controls the QPSK symbol gains on a per CCE basis**
vector

Vector that controls the QPSK symbol gains on a per CCE basis. Each CCE (Control Channel Element) is a group of 36 QPSK symbols (72 bits) and is the minimum unit that a single coded DCI can be mapped to. The number of complete CCE, NCCE = floor(NREG/9), is available via the `NCCE` field in `info`. Each element of `CCEGAINS` acts as a linear multiplier to all 36 symbols generated from the associated block of 72 input bits. If `CCEGAINS` does not cover all the `NREG` symbols, specifically length(`CCEGAINS`) < `NCCE`, then the uncovered CCE receives zero power. All symbols are interleaved before they are output.

Data Types: `double`
Complex Number Support: Yes

## Output Arguments

**`sym` — PDCCH modulation symbols**
complex matrix

PDCCH modulation symbols, given the input bit vector `cw`, returned as a `NRE`-by-`CellRefP` complex matrix. `NRE` is the number of QPSK symbols per antenna and `CellRefP` is the number of TX antenna ports. `NRE` corresponds to the number of control region resource elements assigned to the PDCCH given the structure `enb`.

Data Types: `double`
Complex Number Support: Yes

**`info` — Information for various PDCCH resourcing quantities**
structure

Information for various PDCCH resourcing quantities, returned as a structure. It contains fields including `NRE`, `NREG`, and `MTot`.

`MTot` is the maximum number of input bits that can be transmitted on the `NRE` symbols (`MTot` = 2×`NRE` = 8×`NREG`). If `length(cw)`<`MTot`, the input is padded with (`MTot`-length(`cw`)) <`NIL`> elements which translate to zero valued symbols. Any elements of input vector `cw` valued < 0 are also treated as <`NIL`> elements. If length(`cw`) > `MTot` then only the first `MTot` bits are used.

Data Types: `struct`

# Version History

**Introduced in R2014a**

## See Also

`ltePDCCHDecode` | `ltePDCCHInfo` | `ltePDCCHIndices` | `ltePDCCHPRBS` | `ltePDCCHSpace` | `ltePDCCHSearch` | `ltePDCCHInterleave` | `lteDCIEncode`

# ltePDCCHDecode

Physical downlink control channel decoding

## Syntax

```
[bits,symbols] = ltePDCCHDecode(enb,sym)
[bits,symbols] = ltePDCCHDecode(enb,sym,hest,noiseest)
[bits,symbols] = ltePDCCHDecode(enb,sym,hest,noiseest,alg)
```

## Description

`[bits,symbols] = ltePDCCHDecode(enb,sym)` performs the inverse of Physical Downlink Control Channel (PDCCH) processing on the matrix of complex modulated PDCCH symbols, `sym`, and cell-wide settings structure, `enb`. The channel inverse processing includes resource element group deinterleaving and cyclic shifting, deprecoding, symbol demodulation, and descrambling.

The function returns a column vector of soft bits, `bits`, and received constellation of complex symbol vector, `symbols`, resulting from performing the inverse of PDCCH processing. See TS 36.211 [1], Section 6.8 and `ltePDCCH` for details.

`[bits,symbols] = ltePDCCHDecode(enb,sym,hest,noiseest)` decodes the complex PDCCH symbols, `sym`, using cell-wide settings, `enb`, the channel estimate, `hest`, and the noise estimate, `noiseest`. For the `TxDiversity` transmission scheme, when `CellRefP` is 2 or 4, the reception is performed using an orthogonal space frequency block code (OSFBC) decoder. For the `Port0` transmission scheme, when `CellRefP` is 1, the reception is performed using minimum mean square error (MMSE) equalization.

`[bits,symbols] = ltePDCCHDecode(enb,sym,hest,noiseest,alg)` provides control over weighting the output soft bits, `bits`, with channel state information (CSI) calculated during the equalization stage using algorithmic configuration structure, `alg`. When `alg.CSI` is `'On'`, `bits` is scaled by channel state information calculated during the equalization process.

## Examples

### Decode PDCCH Symbols

Generate and decode the complex PDCCH modulated symbols for RMC R.0 from cell-wide settings structure, `enb`.

```
enb = lteRMCDL('R.0');
pdcchInfo = ltePDCCHInfo(enb);
codewordBits = randi([0,1],pdcchInfo.MTot,1);
pdcchSym = ltePDCCH(enb,codewordBits);
[softBits,symbols] = ltePDCCHDecode(enb,pdcchSym);
```

## Input Arguments

### `enb` — Cell-wide settings

scalar structure

Cell-wide settings, specified as a scalar structure. `enb` contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`CellRefP`** | Required | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **`NCellID`** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **`NSubframe`** | Required | 0 (default), nonnegative scalar integer | Subframe number |

Data Types: `struct`

### `sym` — PDCCH modulation symbols

complex numeric matrix

PDCCH modulation symbols, specified as a complex numeric matrix of size `NRE`-by-`NRxAnts`. `NRE` is the number of QPSK symbols per antenna assigned to the PDCCH (that is, the number of control region resource elements) and `NRxAnts` is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

### `hest` — Channel estimate

3-D numeric array

Channel estimate, specified as a 3-D numeric array of size `NRE`-by-`NRxAnts`-by-`CellRefP`. `NRE` are the frequency and time locations corresponding to the PDCCH RE positions (a total of `NRE` positions). `NRxAnts` is the number of receive antennas, and `CellRefP` is the number of cell-specific reference signal antennas, given by `enb.CellRefP`.

Data Types: `double`
Complex Number Support: Yes

### `noiseest` — Noise estimate

numeric scalar

Noise estimate, specified as a numeric scalar. This input argument is an estimate of the noise power spectral density per RE on received subframe. Produce this estimate using the `lteDLChannelEstimate` function.

Data Types: `double`

### `alg` — Algorithmic configuration to calculate CSI for weighting soft bits

structure

Algorithmic configuration to calculate CSI for weighting soft bits, specified as a structure having the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`CSI`** | Optional | `'On'` (default), `'Off'` | Flag provides control over weighting the soft values that are used to determine the output values with the channel state information (CSI) calculated during the equalization process. If `'On'`, soft values are weighted by CSI. |

Data Types: `struct`

## Output Arguments

### `bits` — Soft bits

numeric column vector

Soft bits, returned as a numeric column vector. `bits` is the received PDCCH payload containing coded downlink control information (DCI) messages. It is optionally scaled by channel state information (CSI) calculated during the equalization process.

Data Types: `double`

### `symbols` — Received constellation symbols

complex numeric column vector

Received constellation symbols, returned as a complex numeric column vector.

Data Types: `double`
Complex Number Support: Yes

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePDCCH` | `ltePDCCHInfo` | `ltePDCCHIndices` | `ltePDCCHPRBS` | `ltePDCCHSpace` | `ltePDCCHSearch` | `ltePDCCHDeinterleave` | `lteDCIDecode`

# ltePDCCHDeinterleave

PDCCH deinterleaving and cyclic shifting

## Syntax

```
out = ltePDCCHDeinterleave(enb,in)
```

## Description

`out = ltePDCCHDeinterleave(enb,in)` performs the PDCCH Resource Element Groups (REGs) deinterleaving and cyclic shifting on PDCCH complex modulated symbols, `in` given cell-wide configuration structure, `enb` This function performs the inverse of the processing described in TS 36.211 [1], Section 6.8.5.

The cyclic shifting process is the reverse of the `NCellID` dependent cyclic shift carried out during PDCCH coding to avoid intercell interference. The de-interleaving is performed to reverse the permutation operation described in TS 36.212 [2], Section 5.1.4.2.1 with the exception that “symbol quadruplets” replace “bits”.

## Examples

### Deinterleave PDCCH Symbols

Perform PDCCH resource element group (REG) deinterleaving. A vector of PDCCH symbols is first interleaved. The output is then deinterleaved and compared with the input vector. Note that instead of actual PDCCH symbols, a range of values from 1 to NRE are used to highlight the interleaved order.

Create a cell-wide configuration structure initialized for RMC R.0. Instead of actual PDCCH symbols, a range of values from 1 to NRE are used to highlight the interleaved order. Interleave the PDCCH symbols, `pdcchSym`.

```
enb = lteRMCDL('R.0');
pdcchInfo = ltePDCCHInfo(enb);
pdcchSym = (1:pdcchInfo.NRE).';
startingSymbolOrder = pdcchSym(1:4)
```

```
startingSymbolOrder = 4x1 uint64 column vector

   1
   2
   3
   4
```

```
interleavedSym = ltePDCCHInterleave(enb,pdcchSym);
interleavedSymbolOrder = interleavedSym(1:4)
```

```
interleavedSymbolOrder = 4x1 uint64 column vector
```

```
   73
   74
   75
   76
```

Deinterleave symbols and view the first four.

```
deinterleavedSym = ltePDCCHDeinterleave(enb,interleavedSym);
deinterleavedSymbolOrder = deinterleavedSym(1:4)
```

```
deinterleavedSymbolOrder = 4x1 uint64 column vector

   1
   2
   3
   4
```

Confirm deinterleaved symbol vector matches the input symbol vector.

```
isequal(pdcchSym,deinterleavedSym)
```

```
ans = logical
   1
```

## Input Arguments

### `enb` — Cell-wide settings

scalar structure

Cell-wide settings, specified as a scalar structure. `enb` can contain the following fields.

#### `NCellID` — Physical layer cell identity

integer from 0 to 503

Physical layer cell identity, specified as an integer from 0 to 503.

Data Types: `struct`

### `in` — PDCCH complex modulated input symbols

numeric matrix

PDCCH complex modulated input symbols, specified as an $N_S$-by-$N_{TX}$ numeric matrix. $N_S$ is the number of modulated symbols, and $N_{TX}$ is the number of transmit antennas. The $N_S$ modulated symbols specified in input matrix `in` must be a concatenation of symbol quadruplets. If the input `in` is a vector, it deinterleaves the elements of the vector. If `in` is a matrix, it deinterleaves the rows.

Data Types: `double`
Complex Number Support: Yes

## Output Arguments

### `out` — Deinterleaved output

numeric column vector

Deinterleaved output, returned as a numeric column vector.

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePDCCHInterleave` | `ltePDCCH` | `ltePDCCHDecode` | `ltePDCCHInfo` | `ltePDCCHIndices` | `ltePDCCHPRBS` | `ltePDCCHSpace` | `ltePDCCHSearch`

# ltePDCCHIndices

PDCCH resource element indices

## Syntax

```
ind = ltePDCCHIndices(enb)
ind = ltePDCCHIndices(enb,opts)
ind = ltePDCCHIndices(enb,exreg,opts)
```

## Description

`ind = ltePDCCHIndices(enb)` returns a `NRE`-by-`CellRefP` matrix of one-based linear indexing RE indices given the structure `enb`. It returns the subframe resource element (RE) indices for the physical downlink control channels (PDCCH).

The `NRE` indices returned cover all PDCCH resources in the control region not already assigned to PCFICH or PHICH (see `ltePDCCHInfo`). They are ordered as the complete block of padded, interleaved, and shifted PDCCH modulation symbols that ready to be mapped, as described in TS 36.211 [1], Section 6.8.5.

`ind = ltePDCCHIndices(enb,opts)` formats the returned indices using options specified by `opts`.

`ind = ltePDCCHIndices(enb,exreg,opts)` returns a matrix of indices, where the vector `exreg` explicitly defines resources not to be assigned to PDCCH. The `exreg` must contain valid resource element group (REG) indices but can be either zero-based or one-based throughout, and indices which do not fall within the control region are ignored.

## Examples

### Get PDCCH Resource Element Indices

Retrieve PDCCH resource element (RE) indices.

Create an RMC R.0 configuration structure and find its PDCCH RE indices. Display the size of the indices.

```
enb = lteRMCDL('R.0');
ind = ltePDCCHIndices(enb);
size(ind)
```

```
ans = 1×2

   452     1
```

### Get PDCCH Indices and Exclude Resources

Explicitly exclude resources when retrieving PDCCH indices.

Create a cell-wide configuration structure initialized for RMC R.0. Generate RE indices for the PDCCH providing an empty matrix for the argument `exreg` so that no resources are excluded.

```
enb = lteRMCDL('R.0');
ind = ltePDCCHIndices(enb,[],'re');
numPDCCHwithNoExclusion = size(ind)
```

```
numPDCCHwithNoExclusion = 1×2

   480     1
```

All RE indices are returned in the required mapping order.

Explicitly exclude the PCFICH and PHICH indices.

```
enb = lteRMCDL('R.0');
exreg = [ltePCFICHIndices(enb,'reg'); ltePHICHIndices(enb,'reg')];
ind = ltePDCCHIndices(enb,exreg,'re');
numPDCCHwithExclusion = size(ind)
```

```
numPDCCHwithExclusion = 1×2

   452     1
```

This call returns the same result as the default syntax call, `ltePDCCHIndices(enb)`.

## Input Arguments

### enb — Cell-wide settings

structure

`enb` is a structure having the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NDLRB`** | Required | Integer within the range (6,...,110) | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **`NCellID`** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **`CyclicPrefix`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **`CellRefP`** | Required | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **`CFI`** | Required | 1, 2, or 3 | Control format indicator value |
| **`Ng`** | Required | `'Sixth'`, `'Half'`, `'One'`, `'Two'` | HICH group multiplier |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **PHICHDuration** | Optional | `'Normal'` (default), `'Extended'` | PHICH duration |
| **DuplexMode** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| The following field is required when `DuplexMode` is set to `'TDD'`. | | | |
| **TDDConfig** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **NSubframe** | Required | Integer greater than 0 | Subframe number |

Data Types: `struct`

### `opts` — Output format, base, and unit of generated indices

character vector | string scalar | cell array of character vectors | string array

Output format, base, and unit of generated indices, specified as one of these forms.

- `'format base unit'`
- `"format base unit"`
- `{'format','base','unit'}`
- `["format","base","unit"]`

Where `format`, `base`, and `unit` are defined in this table.

| Option | Values | Description |
|---|---|---|
| **format** | `'ind'` (default), `'sub'` | Output format of generated indices<br><br>To return the indices as a column vector, specify this option as `'ind'`.<br><br>To return the indices as an $N_{RE}$-by-3 matrix, where $N_{RE}$ is the number of REs, specify this option as `'sub'`. Each row of the matrix contains the subcarrier, symbol, and antenna port as its first, second, and third element, respectively. |
| **base** | `'1based'` (default), `'0based'` | Index base<br><br>To generate indices whose first value is `1`, specify this option as `'1based'`. To generate indices whose first value is `0`, specify this option as `'0based'`. |

| | | |
|---|---|---|
| **unit** | `'re'` (default), `'reg'` | Unit of returned indices<br><br>To indicate that the returned values correspond to individual resource elements (REs), specify this option as `'re'`. To indicate that the returned values correspond to resource element groups (REGs), specify this option as `'reg'`. |

Example: `'ind 0based reg'`, `"ind 0based reg"`, `{'ind','0based','reg'}`, and `["ind","0based","reg"]` specify the same output options.

Data Types: `char` | `string` | `cell`

**`exreg` — Resources excluded from PDCCH**
vector

Resources excluded from PDCCH, specified as a vector. This vector explicitly defines those resources not to be assigned to PDCCH. `exreg` must contain valid resource element group (REG) indices but can be either zero-based or one-based throughout. Indices which do not fall within the control region are ignored.

Data Types: `double`

## Output Arguments

**`ind` — PDCCH RE indices**
numeric matrix

PDCCH RE indices, returned as an `NRE`-by-`CellRefP` numeric matrix by default. The matrix contains one-based linear indexing RE indices. Each column of `ind` identifies the same set of `NRE` subframe resource elements but with indices offset to select them in a different antenna “page” of the 3-D resource array.

The default matrix of indices in a one-based linear indexing style which can directly index elements of an *M*-by-*N*-by-`CellRefP` array, where *M* is the number of symbols, and *N* is the number of subcarriers representing the subframe grid across `CellRefP` antenna ports.

Data Types: `double`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. URL: https://www.3gpp.org.

## See Also

`ltePDCCH` | `ltePDCCHDecode` | `ltePDCCHInfo` | `ltePDCCHInterleave` | `ltePDCCHDeinterleave` | `ltePDCCHSpace` | `ltePDCCHSearch`

# ltePDCCHInfo

PDCCH resource information

## Syntax

```
info = ltePDCCHInfo(enb)
```

## Description

`info = ltePDCCHInfo(enb)` returns a structure `info` containing information about the Physical Downlink Control Channel (PDCCH) subframe resources.

Within a non-MBMS downlink subframe, the first `info.NSymbols` OFDM symbols represent its control region and carry the PCFICH, PHICH and PDCCH. `info.NRE` indicates the number of non-reference resource elements (RE), not assigned to the PCFICH or PHICH, that are associated with PDCCH transmission. These resources carry the set of PDCCH where each PDCCH carries a single encoded DCI message. Each PDCCH can be transmitted on 1,2,4, or 8 control channel elements (CCE), where 1 CCE = 9 REG = 36 RE = 72 bits. As such, not all the `NRE` elements can carry actual PDCCH instances, with (`info.NRE` – `info.NREUsed`) being unavailable for PDCCH transmission.

## Examples

### Get PDCCH Information

Get information about the PDCCH subframe resources for RMC R.0.

```
enb = lteRMCDL('R.0');
info = ltePDCCHInfo(enb)
```

```
info = struct with fields:
        NREG: 113
         NRE: 452
        NCCE: 12
    NREGUsed: 108
     NREUsed: 432
        MTot: 904
    NSymbols: 3
```

## Input Arguments

### enb — Cell-wide settings structure

scalar structure

Cell-wide settings, specified as a scalar structure. `enb` is a structure having the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NDLRB** | Required | Integer within the range (6,...,110) | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **CyclicPrefix** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **CellRefP** | Required | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **CFI** | Required | 1, 2, 3 | Control format indicator value |
| **Ng** | Required | `'Sixth'`, `'Half'`, `'One'`, `'Two'` | HICH group multiplier |
| **DuplexMode** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| **NSubframe** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| The following field is required when `DuplexMode` is set to `'TDD'`. | | | |
| **TDDConfig** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |

Data Types: `struct`

## Output Arguments

### `info` — PDCCH subframe resource information

structure

PDCCH subframe resource information, returned as a structure. `info` contains the following fields.

| Parameter Field | Values | Description |
|---|---|---|
| **NRE** | Numeric scalar | Total number of resource elements (REs) associated with PDCCHs (4×`NREG`) |
| **NREG** | Numeric scalar | Total number of resource element groups (REGs) associated with PDCCHs (4×`NRE`) |
| **MTot** | Numeric scalar | Total number of bits associated with PDCCHs, returned as a numeric scalar (8×`NREG`). `MTot` is the maximum number of input bits that can be transmitted on the `NRE` symbols (`MTot` = 2 × `NRE` = 8 × `NREG`). |
| **NCCE** | Numeric scalar | Number of control channel elements available for actual PDCCH usage |
| **NREGUsed** | Numeric scalar | Number of resource element groups (REGs) available for actual PDCCH usage |

| Parameter Field | Values | Description |
|---|---|---|
| **NREUsed** | Numeric scalar | Number of resource elements (REs) available for actual PDCCH usage |
| **NSymbols** | Numeric scalar | Total number of OFDM symbols spanned by the PDCCH |

Data Types: `struct`

## Version History

**Introduced in R2014a**

## See Also

`ltePDCCH` | `ltePDCCHDecode` | `ltePDCCHIndices` | `ltePDCCHInterleave` | `ltePDCCHDeinterleave` | `ltePDCCHPRBS` | `ltePDCCHSpace` | `ltePDCCHSearch`

# ltePDCCHInterleave

PDCCH interleaving and cyclic shift

## Syntax

```
out = ltePDCCHInterleave(enb,in)
```

## Description

`out = ltePDCCHInterleave(enb,in)` performs the interleaving and cyclic shifting on PDCCH resource element groups (REGs) as described in TS 36.211 [1], Section 6.8.5.

The permutation, or interleaving, operation is performed as described in TS 36.212 [2], Section 5.1.4.2.1, with the exception that “symbol quadruplets” replace “bits”. Then, the block of PDCCH-modulated symbol quadruplets is cyclically shifted with `NCellID` to avoid intercell interference.

## Examples

### Perform PDCCH Interleaving

Interleave a sequential input sized to the number of resource elements.

```
enb = lteRMCDL('R.0');
pdcchInfo = ltePDCCHInfo(enb);
interleavedSym = ltePDCCHInterleave(enb,(1:pdcchInfo.NRE).');
size(interleavedSym)
```

```
ans = 1×2

   452     1
```

```
interleavedSym(1:12)
```

```
ans = 12x1 uint64 column vector

    73
    74
    75
    76
   201
   202
   203
   204
   329
   330
     ⋮
```

The sequential input is interleaved, resulting in the concatenation of input quadruplets.

## Input Arguments

### `enb` — Cell-wide settings

scalar structure

Cell-wide settings, specified as a scalar structure. `enb` contains the following fields.

### `NCellID` — Physical layer cell identity

integer from 0 to 503

Physical layer cell identity, specified as an integer from 0 to 503.

Data Types: `struct`

### `in` — PDCCH complex modulated input symbols

complex-valued numeric matrix | numeric vector

PDCCH complex modulated input symbols, specified in a complex-valued numeric $N_S$-by-$N_{TX}$ matrix, or a numeric vector. $N_S$ is the number of modulated symbols, and $N_{TX}$ is the number of transmit antennas. The $N_S$ modulated symbols specified in input matrix, `in`, must be a concatenation of symbol quadruplets. If the input, `in`, is a vector, it interleaves the elements of the vector. If `in` is a matrix, it interleaves the rows.

Data Types: `double`
Complex Number Support: Yes

## Output Arguments

### `out` — Interleaved output

numeric vector

Interleaved output, returned as a numeric vector.

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePDCCHDeinterleave` | `ltePDCCH` | `ltePDCCHDecode` | `ltePDCCHInfo` | `ltePDCCHIndices` | `ltePDCCHPRBS` | `ltePDCCHSpace` | `ltePDCCHSearch`

# ltePDCCHPRBS

PDCCH pseudorandom scrambling sequence

## Syntax

```
[seq,cinit] = ltePDCCHPRBS(enb,n)
[seq,cinit] = ltePDCCHPRBS(enb,n,mapping)

[subseq,cinit] = ltePDCCHPRBS(enb,pn)
[subseq,cinit] = ltePDCCHPRBS(enb,pn,mapping)
```

## Description

`[seq,cinit] = ltePDCCHPRBS(enb,n)` returns a column vector containing the first `n` outputs of the Physical Downlink Control Channel (PDCCH) scrambling sequence when initialized according to cell-wide settings structure, `enb`. It also returns an initialization value `cinit` for the pseudorandom binary sequence (PRBS) generator.

`[seq,cinit] = ltePDCCHPRBS(enb,n,mapping)` allows control over the format of the returned sequence, `seq`, with the input `mapping`.

`[subseq,cinit] = ltePDCCHPRBS(enb,pn)` returns a subsequence of a full PRBS sequence, specified by `pn`.

`[subseq,cinit] = ltePDCCHPRBS(enb,pn,mapping)` allows additional control over the format of the returned subsequence, `subseq`, with the input `mapping`.

## Examples

### Return Binary PDCCH Scrambling Sequence

Generate the first 7 outputs of the pseudorandom scrambling sequence for PDCCH in binary (default) format.

```
enb = lteRMCDL('R.0');
pdcchSeqBinary = ltePDCCHPRBS(enb,7)
```

```
pdcchSeqBinary = 7x1 logical array

   0
   0
   0
   0
   0
   0
   1
```

### Scramble Random PDCCH Codeword

Scramble a random PDCCH codeword.

Create a cell-wide configuration structure, initialized to RMC R.0. Generate a codeword. Use `MTot` to determine the total number of bits associated with PDCCHs. Generate a PDCCH pseudorandom scrambling sequence the same length as the codeword.

```
enb = lteRMCDL('R.0');

pdcchInfo = ltePDCCHInfo(enb)
```

```
pdcchInfo = struct with fields:
        NREG: 113
         NRE: 452
        NCCE: 12
    NREGUsed: 108
     NREUsed: 432
        MTot: 904
    NSymbols: 3
```

```
cw = randi([0,1],pdcchInfo.MTot,1);

pdcchSeq = ltePDCCHPRBS(enb,length(cw));
```

Scramble codeword with PDCCH PRBS.

```
scrambled = xor(pdcchSeq,cw);
```

### Return Signed PDCCH Scrambling Sequence

Generate the first 7 outputs of the pseudorandom scrambling sequence for PDCCH in signed format.

```
enb = lteRMCDL('R.0');
pdcchSeqSigned = ltePDCCHPRBS(enb,7,'signed')
```

```
pdcchSeqSigned = 7×1

     1
     1
     1
     1
     1
     1
    -1
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure. This argument contains the following fields.

**`NCellID` — Physical layer cell identity**
0,...,503

Physical layer cell identity, specified as a nonnegative scalar integer from 0 through 503.

Data Types: `double`

**`NSubframe` — Subframe number**
positive scalar integer

Subframe number, specified as a positive scalar integer greater than 0.

Data Types: `double`

Data Types: `struct`

**`n` — Number of elements in returned sequence**
numeric scalar

Number of elements in returned sequence, `seq`, specified as a numeric scalar.

Data Types: `double`

**`pn` — Range of elements in returned subsequence**
row vector

Range of elements in returned subsequence, `subseq`, specified as a row vector of `[p n]`. The subsequence returns `n` values of the PRBS generator, starting at position `p` (0-based).

Data Types: `double`

**`mapping` — Output sequence formatting**
`'binary'` (default) | `'signed'`

Output sequence formatting, specified as `'binary'` or `'signed'`. `mapping` controls the format of the returned sequence.

- `'binary'` maps `true` to 1 and `false` to 0.
- `'signed'` maps `true` to –1 and `false` to 1.

Data Types: `char` | `string`

## Output Arguments

**`seq` — PDCCH pseudorandom scrambling sequence**
logical column vector | numeric column vector

PDCCH pseudorandom scrambling sequence, returned as a logical column vector or a numeric column vector. `seq` contains the first `n` outputs of the PDCCH scrambling sequence when initialized according to cell-wide settings structure, `enb`. If you set `mapping` to `'signed'`, the output data type is double. Otherwise, the output data type is logical.

Data Types: `logical` | `double`

**`subseq` — PDCCH pseudorandom scrambling subsequence**
logical column vector | numeric column vector

PDCCH pseudorandom scrambling subsequence, returned as a logical column vector or a numeric column vector. `subseq` contains the values of the PRBS generator specified by `pn`. If you set `mapping` to `'signed'`, the output data type is `double`. Otherwise, the output data type is `logical`.

Data Types: `logical` | `double`

**`cinit` — Initialization value for PRBS generator**
numeric scalar

Initialization value for PRBS generator, returned as a numeric scalar.

Data Types: `uint32`

# Version History

**Introduced in R2014a**

## See Also

`ltePDCCH` | `ltePDCCHDecode` | `ltePDCCHIndices` | `ltePDCCHInterleave` | `ltePDCCHDeinterleave` | `ltePDCCHInfo` | `ltePDCCHSpace` | `ltePDCCHSearch`

# ltePDCCHSearch

PDCCH downlink control information search

## Syntax

```
[dcistr,dcibits] = ltePDCCHSearch(enb,chs,softbits)
```

## Description

`[dcistr,dcibits] = ltePDCCHSearch(enb,chs,softbits)` recovers downlink control information (DCI) message structures, `dcistr`, and corresponding vectors of DCI message bits, `dcibits`, after blind decoding the multiplexed physical downlink control channels (PDCCHs) within the control region given by the `softbits` input vector, cell-wide configuration, `enb`, and UE-specific channel configuration, `chs`. For more information, see “PDCCH Search Processing” on page 2-606.

## Examples

### Get DCI Message Structure and Bits

Extract and decode the PDCCH symbols from the control region of a subframe grid created by the DL waveform generator, `lteRMCDLTool`. Use the blind search function, `ltePDCCHSearch`, to search the common and UE-specific spaces by demasking all PDCCH candidates with the configured RNTI.

Use a waveform generator to create a full subframe grid containing a reference PDSCH and associated DCI in UE-specific search space. Extract and decode all the PDCCH multiplex (control region) bits.

```
rmc = lteRMCDL('R.0');
[~,txGrid] = lteRMCDLTool(rmc,[1;0;0;1]);

pdcchSymbols = txGrid(ltePDCCHIndices(rmc));
rxPdcchBits = ltePDCCHDecode(rmc,pdcchSymbols);
```

Configure the UE-specific parameters that affect the DCI message lengths to match those of the reference UE.

```
ueConfig.RNTI = rmc.PDSCH.RNTI;
ueConfig.EnableCarrierIndication = 'Off';
ueConfig.EnableSRSRequest = 'Off';
ueConfig.EnableMultipleCSIRequest = 'Off';
ueConfig.NTxAnts = 1;
```

Use PDCCH blind search to find the DCI that schedules the PDSCH. Extract and display first DCI message structure from the search list. Compare the format of the DCI message returned in the previous step with the format used by the waveform generator.

```
[rxDCI,rxDCIBits] = ltePDCCHSearch(rmc,ueConfig,rxPdcchBits);

decDCI = rxDCI{1}
```

```
decDCI = struct with fields:
           DCIFormat: 'Format1'
                 CIF: 0
      AllocationType: 1
          Allocation: [1x1 struct]
           ModCoding: 14
              HARQNo: 0
             NewData: 1
                  RV: 0
            TPCPUCCH: 0
            TDDIndex: 0
    HARQACKResOffset: 0
```

```
decDCIFormat = decDCI.DCIFormat
```

```
decDCIFormat =
'Format1'
```

```
txDCIFormat = rmc.PDSCH.DCIFormat
```

```
txDCIFormat =
'Format1'
```

### Get Two DCI Messages

Map a pair of format 0 uplink and format 1A downlink grants into a UE-specific search space in the PDCCH multiplex. Use the blind search function to recover them. Because the search space is UE-specific, you can extend the messages to include the Release 10 SRS request field and a 2-bit CSI request field for the format 0 DCI. For simplicity, the example does not include any PDCCH channel processing steps.

Create a vector containing the control region PDCCH multiplex bits.

```
enb = lteRMCDL('R.0');
pdcchinfo = ltePDCCHInfo(enb);
pdcchmux = zeros(1,pdcchinfo.MTot);
```

Configure the UE-specific parameters to control the DCI and encoding.

```
chs = struct('RNTI',1,'PDCCHFormat',2);
chs.ControlChannelType = 'PDCCH';
chs.SearchSpace = 'UESpecific';
chs.EnableCarrierIndication = 'Off';
chs.EnableSRSRequest = 'On';
chs.EnableMultipleCSIRequest = 'On';
chs.NTxAnts = 1;
```

List the formats to create and get the UE-specific search space candidate locations in the PDCCH multiplex.

```
formats = {'Format0','Format1A'};
candidates = ltePDCCHSpace(enb,chs);
```

For each DCI format, create the DCI info bits and encode them for PDCCH mapping. Demonstrate setting of the `ModCoding` field to a nondefault value. Select a candidate to carry the target PDCCH.

```
for f = 1:length(formats)

    dciin = struct('DCIFormat',formats{f},'ModCoding',f);
    [dci,dcibits] = lteDCI(enb,chs,dciin);
    pdcch = lteDCIEncode(chs,dcibits);

    pdcchmux(candidates(f,1):candidates(f,2)) = pdcch;
end
```

Search PDCCH multiplex bits for any DCI messages directed at UE RNTI.

```
rxDCI = ltePDCCHSearch(enb,chs,pdcchmux)
```

```
rxDCI=1×2 cell array
    {1x1 struct}    {1x1 struct}
```

```
rxDCI{:}
```

```
ans = struct with fields:
           DCIFormat: 'Format1A'
                 CIF: 0
      AllocationType: 0
          Allocation: [1x1 struct]
           ModCoding: 2
              HARQNo: 0
             NewData: 0
                  RV: 0
            TPCPUCCH: 0
            TDDIndex: 0
          SRSRequest: 0
    HARQACKResOffset: 0


ans = struct with fields:
         DCIFormat: 'Format0'
               CIF: 0
       FreqHopping: 0
        Allocation: [1x1 struct]
         ModCoding: 1
           NewData: 0
               TPC: 0
        CShiftDMRS: 0
          TDDIndex: 0
        CSIRequest: 0
        SRSRequest: 0
    AllocationType: 0
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure with these fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NDLRB** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$)<br><br>For information on link bandwidth assignment, see “Specifying Number of Resource Blocks” on page 2-606. |
| **NULRB** | Required | Scalar integer from 6 to 110 | Number of uplink resource blocks. ($N_{RB}^{UL}$)<br><br>For information on link bandwidth assignment, see “Specifying Number of Resource Blocks” on page 2-606. |
| **NSubframe** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **CellRefP** | Required | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **DuplexMode** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as one of the following:<br>• `'FDD'` — Frequency division duplex (default)<br>• `'TDD'` — Time division duplex |

Data Types: `struct`

**`chs` — User-equipment-related channel configuration**
structure

User-equipment-related (UE-related) channel configuration, specified as a structure containing the following UE-specific fields.

**`RNTI` — Radio network temporary identifier**
1 (default) | numeric scalar

Radio network temporary identifier value, specified as a numeric scalar.

Data Types: `double`

**`EnableCarrierIndication` — Option to enable carrier indication**
`'Off'` (default) | `'On'` | optional

Option to enable carrier indication UE configuration, specified as `'Off'` or `'On'`. Default configuration is disabled. When enabled, `'On'`, the carrier indication field is present.

Data Types: `char` | `string`

**`EnableSRSRequest` — Option to enable SRS request**
`'Off'` (default) | `'On'` | optional

Option to enable SRS request in the UE configuration, specified as `'Off'` or `'On'`. By default, `EnableSRSRequest` is disabled. When `EnableSRSRequest` is enabled (`'On'`), the SRS request field is present in UE-specific formats 0/1A for FDD or TDD and formats 2B/2C/2D for TDD.

Data Types: `char` | `string`

**`EnableMultipleCSIRequest` — Option to enable multiple CSI requests**
`'Off'` (default) | `'On'` | optional

Option to enable multiple CSI requests in the UE configuration, specified as `'Off'` or `'On'`. By default, `EnableMultipleCSIRequest` is disabled. When `EnableMultipleCSIRequest` is enabled (`'On'`), the UE is configured to process multiple channel state information (CSI) requests from cells. Enabling multiple CSI requests affects the length of the CSI request field in UE-specific formats 0 and 4.

Data Types: `char` | `string`

**`NTxAnts` — Number of UE transmission antennas**
1 (default) | 2 | 4 | optional

Number of UE transmission antennas, specified as 1, 2, or 4. The number of UE transmission antennas affects the length of the precoding information field in DCI format 4.

Data Types: `double`

Data Types: `struct`

**`softbits` — Input vector of soft bits**
numeric column vector

Input vector of soft bits, specified as a column vector.

Data Types: `double`

## Output Arguments

**`dcistr` — Downlink control information (DCI) message structures**
cell array of structures

Downlink control information (DCI) message structures, returned as a cell array of structures. Each structure represents a successfully decoded DCI whose fields match fields of the associated DCI format. Each structure contains the fields associated with one or more decoded DCI messages. Because multiple PDCCHs can be transmitted in a subframe, the UE must monitor all possible PDCCHs directed at it. If more than one PDCCH is directed to the UE or is successfully decoded, `dcistr` contains that number of decoded DCI messages.

Each cell contains a structure with the fields associated with the DCI format of the received PDCCHs.

**`DCIFormat` — DCI format type**
`'Format0'` | `'Format1'` | `'Format1A'` | `'Format1B'` | `'Format1C'` | `'Format1D'` | `'Format2'` | `'Format2A'` | `'Format2B'` | `'Format2C'` | `'Format3'` | `'Format3A'` | `'Format4'` | `'Format5'`

DCI format type, specified as a character vector. This table presents the fields associated with each DCI format as defined in TS 36.212 [1], Section 5.3.3.

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| 'Format0' | DCIFormat | - | 'Format0' |
| | CIF | 0 or 3 bits | Carrier indicator field |
| | FreqHopping | 1 bit | PUSCH frequency hopping flag |
| | Allocation | Varies | Resource block assignment/allocation |
| | ModCoding | 5 bits | Modulation, coding scheme, and redundancy version |
| | NewData | 1 bit | New data indicator |
| | TPC | 2 bits | PUSCH TPC command |
| | CShiftDMRS | 3 bits | Cyclic shift for DM RS |
| | TDDIndex | 2 bits | For TDD config 0, this field is the Uplink Index.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | CSIRequest | 1, 2, or 3 bits | CSI request |
| | SRSRequest | 0 or 1 bit | SRS request. This field can only be present in DCI formats scheduling PUSCH which are mapped onto the UE specific search space given by the C-RNTI |
| | AllocationType | 1 bit | Resource allocation type, only present if $N_{RB}^{UL} \leq N_{RB}^{DL}$. |
| 'Format1' | DCIFormat | - | 'Format1' |
| | CIF | 0 or 3 bits | Carrier indicator field |
| | AllocationType | 1 bit | Resource allocation header: type 0, type 1. If downlink bandwidth is ≤10 PRBs there is no resource allocation header and resource allocation type 0 is assumed. |
| | Allocation | Varies | Resource block assignment/allocation |
| | ModCoding | 5 bits | Modulation and coding scheme |
| | HARQNo | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |
| | NewData | 1 bit | New data indicator |
| | RV | 2 bits | Redundancy version |
| | TPCPUCCH | 2 bits | PUCCH TPC command |

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| | TDDIndex | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | HARQACKResOffset | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| 'Format1A' | DCIFormat | - | 'Format1A' |
| | CIF | 0 or 3 bits | Carrier indicator field |
| | AllocationType | 1 bit | VRB assignment flag: 0 (localized), 1 (distributed) |
| | Allocation | Varies | Resource block assignment/allocation |
| | ModCoding | 5 bits | Modulation and coding scheme |
| | HARQNo | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |
| | NewData | 1 bit | New data indicator |
| | RV | 2 bits | Redundancy version |
| | TPCPUCCH | 2 bits | PUCCH TPC command |
| | TDDIndex | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | SRSRequest | 0 or 1 bit | SRS request. This field can only be present in DCI formats scheduling PUSCH which are mapped onto the UE specific search space given by the C-RNTI |
| | HARQACKResOffset | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| 'Format1B' | DCIFormat | - | 'Format1B' |
| | CIF | 0 or 3 bits | Carrier indicator field |
| | AllocationType | 1 bit | VRB assignment flag: 0 (localized), 1 (distributed) |
| | Allocation | Varies | Resource block assignment/allocation |
| | ModCoding | 5 bits | Modulation and coding scheme |

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| | HARQNo | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |
| | NewData | 1 bit | New data indicator |
| | RV | 2 bits | Redundancy version |
| | TPCPUCCH | 2 bits | PUCCH TPC command |
| | TDDIndex | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | TPMI | 2 bits for two antennas<br><br>4 bits for four antennas | PMI information |
| | PMI | 1 bit | PMI confirmation |
| | HARQACKResOffset | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| 'Format1C' | DCIFormat | - | 'Format1C' |
| | Allocation | Varies | Resource block assignment/allocation |
| | ModCoding | 5 bits | Modulation and coding scheme |
| 'Format1D' | DCIFormat | - | 'Format1D' |
| | CIF | 0 or 3 bits | Carrier indicator field |
| | AllocationType | 1 bit | VRB assignment flag: 0 (localized), 1 (distributed) |
| | Allocation | Varies | Resource block assignment/allocation |
| | ModCoding | 5 bits | Modulation and coding scheme |
| | HARQNo | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |
| | NewData | 1 bit | New data indicator |
| | RV | 2 bits | Redundancy version |
| | TPCPUCCH | 2 bits | PUCCH TPC command |
| | TDDIndex | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| | `TPMI` | 2 bits for two antennas<br><br>4 bits for four antennas | Precoding TPMI information |
| | `DlPowerOffset` | 1 bit | Downlink power offset |
| | `HARQACKResOffset` | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| `'Format2'` | `DCIFormat` | `-` | `'Format2'` |
| | `CIF` | 0 or 3 bits | Carrier indicator field |
| | `AllocationType` | 1 bit | Resource allocation header: type 0, type 1. If downlink bandwidth is ≤10 PRBs there is no resource allocation header and resource allocation type 0 is assumed. |
| | `Allocation` | Varies | Resource block assignment/allocation |
| | `TPCPUCCH` | 2 bits | PUCCH TPC command |
| | `TDDIndex` | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | `HARQNo` | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |
| | `SwapFlag` | 1 bit | Transport block to codeword swap flag |
| | `ModCoding1` | 5 bits | Modulation and coding scheme for transport block 1 |
| | `NewData1` | 1 bit | New data indicator for transport block 1 |
| | `RV1` | 2 bits | Redundancy version for transport block 1 |
| | `ModCoding2` | 5 bits | Modulation and coding scheme for transport block 2 |
| | `NewData2` | 1 bit | New data indicator for transport block 2 |
| | `RV2` | 2 bits | Redundancy version for transport block 2 |
| | `PrecodingInfo` | 3 bits for two antennas<br><br>6 bits for four antennas | Precoding information |

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| | `HARQACKResOffset` | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| `'Format2A'` | `DCIFormat` | - | `'Format2A'` |
| | `CIF` | 0 or 3 bits | Carrier indicator field |
| | `AllocationType` | 1 bit | Resource allocation header: type 0, type 1. If downlink bandwidth is ≤10 PRBs there is no resource allocation header and resource allocation type 0 is assumed. |
| | `Allocation` | Varies | Resource block assignment/allocation |
| | `TPCPUCCH` | 2 bits | PUCCH TPC command |
| | `TDDIndex` | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | `HARQNo` | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |
| | `SwapFlag` | 1 bit | Transport block to codeword swap flag |
| | `ModCoding1` | 5 bits | Modulation and coding scheme for transport block 1 |
| | `NewData1` | 1 bit | New data indicator for transport block 1 |
| | `RV1` | 2 bits | Redundancy version for transport block 1 |
| | `ModCoding2` | 5 bits | Modulation and coding scheme for transport block 2 |
| | `NewData2` | 1 bit | New data indicator for transport block 2 |
| | `RV2` | 2 bits | Redundancy version for transport block 2 |
| | `PrecodingInfo` | 0 bits for two antennas<br><br>2 bits for four antennas | Precoding information |
| | `HARQACKResOffset` | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| `'Format2B'` | `DCIFormat` | - | `'Format2B'` |
| | `CIF` | 0 or 3 bits | Carrier indicator field |

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| | `AllocationType` | 1 bit | Resource allocation header: type 0, type 1. If downlink bandwidth is ≤10 PRBs there is no resource allocation header and resource allocation type 0 is assumed. |
| | `Allocation` | Varies | Resource block assignment/allocation |
| | `TPCPUCCH` | 2 bits | PUCCH TPC command |
| | `TDDIndex` | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | `HARQNo` | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |
| | `ScramblingId` | 1 bit | Scrambling identity |
| | `ModCoding1` | 5 bits | Modulation and coding scheme for transport block 1 |
| | `NewData1` | 1 bit | New data indicator for transport block 1 |
| | `RV1` | 2 bits | Redundancy version for transport block 1 |
| | `ModCoding2` | 5 bits | Modulation and coding scheme for transport block 2 |
| | `NewData2` | 1 bit | New data indicator for transport block 2 |
| | `RV2` | 2 bits | Redundancy version for transport block 2 |
| | `HARQACKResOffset` | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| `'Format2C'` | `DCIFormat` | - | `'Format2C'` |
| | `CIF` | 0 or 3 bits | Carrier indicator field |
| | `AllocationType` | 1 bit | Resource allocation header: type 0, type 1. If downlink bandwidth is ≤10 PRBs there is no resource allocation header and resource allocation type 0 is assumed. |
| | `Allocation` | Varies | Resource block assignment/allocation |
| | `TPCPUCCH` | 2 bits | PUCCH TPC command |

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| | `TDDIndex` | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | `HARQNo` | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |
| | `TxIndication` | 3 bits | Antenna ports, scrambling identity, and number of layers indicator |
| | `SRSRequest` | Varies | SRS request. Only present for TDD. |
| | `ModCoding1` | 5 bits | Modulation and coding scheme for transport block 1 |
| | `NewData1` | 1 bit | New data indicator for transport block 1 |
| | `RV1` | 2 bits | Redundancy version for transport block 1 |
| | `ModCoding2` | 5 bits | Modulation and coding scheme for transport block 2 |
| | `NewData2` | 1 bit | New data indicator for transport block 2 |
| | `RV2` | 2 bits | Redundancy version for transport block 2 |
| | `HARQACKResOffset` | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| `'Format2D'` | `DCIFormat` | - | `'Format2D'` |
| | `CIF` | 0 or 3 bits | Carrier indicator field |
| | `AllocationType` | 1 bit | Resource allocation header: type 0, type 1. If downlink bandwidth is ≤10 PRBs there is no resource allocation header and resource allocation type 0 is assumed. |
| | `Allocation` | Varies | Resource block assignment/allocation |
| | `TPCPUCCH` | 2 bits | PUCCH TPC command |
| | `TDDIndex` | 2 bits | For TDD config 0, this field is not used.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | `HARQNo` | 3 bits (FDD)<br><br>4 bits (TDD) | HARQ process number |

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| | `TxIndication` | 3 bits | Antenna ports, scrambling identity, and number of layers indicator |
| | `SRSRequest` | Varies | SRS request. Only present for TDD. |
| | `ModCoding1` | 5 bits | Modulation and coding scheme for transport block 1 |
| | `NewData1` | 1 bit | New data indicator for transport block 1 |
| | `RV1` | 2 bits | Redundancy version for transport block 1 |
| | `ModCoding2` | 5 bits | Modulation and coding scheme for transport block 2 |
| | `NewData2` | 1 bit | New data indicator for transport block 2 |
| | `RV2` | 2 bits | Redundancy version for transport block 2 |
| | `REMappingAndQCL` | 2 bits | PDSCH RE Mapping and Quasi-Co-Location Indicator |
| | `HARQACKResOffset` | 2 bits | HARQ-ACK resource offset. Present when this format is carried by EPDCCH. Not present when this format is carried by PDCCH |
| `'Format3'` | `DCIFormat` | - | `'Format3'` |
| | `TPCCommands` | Varies | TPC commands for PUCCH and PUSCH |
| `'Format3A'` | `DCIFormat` | - | `'Format3A'` |
| | `TPCCommands` | Varies | TPC commands for PUCCH and PUSCH |
| `'Format4'` | `DCIFormat` | - | `'Format4'` |
| | `CIF` | 0 or 3 bits | Carrier indicator field |
| | `Allocation` | Varies | Resource block assignment/allocation |
| | `TPC` | 2 bits | PUSCH TPC command |
| | `CShiftDMRS` | 3 bits | Cyclic shift for DM-RS |
| | `TDDIndex` | 2 bits | For TDD config 0, this field is Uplink Index.<br><br>For TDD config 1–6, this field is the Downlink Assignment Index.<br><br>Not present for FDD. |
| | `CSIReq` | Varies | CSI request |
| | `SRSRequest` | 2 bits | SRS request |
| | `AllocationType` | 1 bit | Resource allocation header type 0 or type 1. |
| | `ModCoding` | 5 bits | Modulation, coding scheme, and redundancy version |

| DCI Formats | dciout Fields | Size | Description |
|---|---|---|---|
| | `NewData` | 1 bit | New data indicator |
| | `ModCoding1` | 5 bits | Modulation and coding scheme for transport block 1 |
| | `NewData1` | 1 bit | New data indicator for transport block 1 |
| | `ModCoding2` | 5 bits | Modulation and coding scheme for transport block 2 |
| | `NewData2` | 1 bit | New data indicator for transport block 2 |
| | `PrecodingInfo` | 3 bits for two antennas<br><br>6 bits for four antennas | Precoding information |
| `'Format5'` | `DCIFormat` | - | `'Format5'` |
| | `PSCCHResource` | 6 bits | Resource for PSCCH |
| | `TPC` | 1 bit | TPC command for PSCCH and PSSCH |
| | `FreqHopping` | 1 bit | Frequency hopping flag |
| | `Allocation` | Varies | Resource block assignment and hopping resource allocation |
| | `TimeResourcePattern` | 7 bits | Time resource pattern |
| `'Format5A'` | `DCIFormat` | - | `'Format5A'` |
| | `CIF` | 3 bits | Carrier indicator |
| | `FirstSubchannelIdx` | $\left\lceil \log_2\left(N_{\text{subchannel}}^{\text{SL}}\right) \right\rceil$ | Lowest index of the subchannel allocation to the initial transmission |
| | `RIV` | from 0 to 13 bits, $\left\lceil \log_2 \left( \frac{N_{\text{subchannel}}^{\text{SL}} \times \left(N_{\text{subchannel}}^{\text{SL}} + 1\right)}{2} \right) \right\rceil$ | Resource indication value |
| | `TimeGap` | 4 bits | Time gap between initial transmission and retransmission |
| | `SLIndex` | 2 bits | SL SPS configuration index |

Data Types: `char`

Data Types: `cell`

**`dcibits` — DCI message bits**
cell array of numeric vectors

DCI message bits, returned as a cell array of one or more numeric vectors. Each vector contains the bit stream of a recovered DCI message, including any zero-padding. Each vector of bit values corresponds to successfully decoded DCI messages. For more information, see `lteDCI`.

Data Types: `cell`

## More About

### PDCCH Search Processing

PDCCH search processing blindly decodes DCI messages based on their lengths. The lengths and order in which DCI messages are searched for is provided by `lteDCIInfo`. If one or more messages have the same length, the first message format in the list is used to decode the message. The other potential message formats are ignored. The `ltePDCCHSearch` function does not consider transmission mode (TM) during blind search, and no DCI message format is filtered based on transmission mode. It also does not search for format 3 and 3A (power adjustment commands for PUSCH and PUCCH). For more information on the association between transmission mode, transmission scheme, DCI format, and search space, see TS 36.213 [2], Section 7.1 and Table 7.1-5.

The UE is required to monitor multiple PDCCHs within the control region. The UE is informed only of the width, in OFDM symbols, of the control region within a subframe, and is not aware of the exact location of PDCCHs relevant to it. The UE finds the PDCCHs relevant to it by monitoring a set of PDCCH candidates, that is, a set of consecutive control candidate elements (CCEs) on which PDCCH can be mapped, in every subframe. For details, see `ltePDCCHSpace`. This process is referred to as blind decoding.

To simplify the decoding task at the UE, the whole control region is subdivided into common and UE-specific search spaces which the UE monitors (monitor implies attempting to decode each PDCCH). Each search space comprises 2, 4, or 6 PDCCH candidates whose data length depends on its corresponding PDCCH format. Each PDCCH must be transmitted on 1, 2, 4, or 8 CCE (1 CCE = 72 bits). The common search space is limited to only two aggregation levels, 4 and 8, while the UE-specific search space can have an aggregation level of 1, 2, 4, or 8.

All UEs within a cell monitor the common search space that carries control information common to all UEs. The common control information carries initial important information including paging information, system information, and random access procedures. The UE monitors the common search space by demasking each PDCCH candidate with different RNTIs, for example, P-RNTI, SI-RNTI, RA-RNTI and so on.

In the UE-specific search space, the UE finds the PDCCH relevant to it by monitoring a set of PDCCH candidates in every subframe. If no CRC error is detected when the UE demasks a PDCCH candidate with its RNTI (16-bit C-RNTI value), the UE determines that the PDCCH candidate carries its own control information.

The number and location of candidates within a search space is different for each PDCCH format. There are four PDCCH formats (0, 1, 2, or 3). If the UE fails to decode any PDCCH candidates for a given PDCCH format, it tries to decode candidates for another PDCCH format.

### Specifying Number of Resource Blocks

The number of resource blocks specifies the uplink and downlink bandwidth. The LTE Toolbox implementation assumes symmetric link bandwidth unless you specifically assign different values to

NULRB and NDLRB. If the number of resource blocks is initialized in only one link direction, then the initialized number of resource blocks (NULRB or NDLRB) is used for both uplink and downlink. When this mapping is used, no warning is displayed. An error occurs if NULRB and NDLRB are both undefined.

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.213. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

ltePDCCH | ltePDCCHDecode | ltePDCCHIndices | ltePDCCHInterleave | ltePDCCHDeinterleave | ltePDCCHInfo | ltePDCCHSpace | ltePDCCHPRBS

# ltePDCCHSpace

PDCCH search space candidates

## Syntax

```
ind = ltePDCCHSpace(enb,ue)
ind = ltePDCCHSpace(enb,ue,opts)
```

## Description

`ind = ltePDCCHSpace(enb,ue)` returns the (0,2,4,6)-by-2 matrix `ind` of search space PDCCH candidate indices given the structures `enb` and `ue`. Depending on input parameters, each search space contains (0,2,4, or 6) PDCCH candidate locations defined by the rows of `ind`. Each two-element row contains the inclusive `[begin,end]` indices of a single PDCCH candidate location. By default, the one-based indices define the PDCCH locations in the block of all multiplexed PDCCH data bits to be transmitted in that subframe.

The control region of a downlink subframe comprises the multiplexing of all PDCCHs bits into a single block of data which is then processed and interleaved before PDCCH resource mapping. A UE has to blindly decode individual PDCCH directed at it. This task is simplified by subdividing the whole region into common and UE-specific search spaces which the UE should monitor. Each space comprises 2, 4, or 6 PDCCH candidates whose data length depends on its PDCCH format. Each PDCCH must be transmitted on 1, 2, 4, or 8 control channel elements (CCE) (1 CCE = 72 bits).

The returned search space is of the UE-specific type unless the `RNTI` field is missing from the structure `ue` when a common search space is returned. The search space always contains 2, 4, or 6 candidates; therefore, `ind` has 2, 4, or 6 rows, unless the parameter combinations are not valid, in which case the `ind` output returned is empty. For more information, see TS 36.213 [1], Section 9.1.1. The candidates in a space do not need to be unique, especially for smaller bandwidths.

`ind = ltePDCCHSpace(enb,ue,opts)` formats the returned indices using options specified by `opts`.

## Examples

### Get PDCCH Search Space Candidates

Find and use PDCCH search space candidates.

To illustrate the search space structuring of the PDCCH, set up a cell wide parameter structure, `enb`, with the following field values.

```
enb.NDLRB = 50;
enb.CFI = 2;
enb.CellRefP = 2;
enb.Ng = 'Sixth';
enb.NSubframe = 0;
```

This configuration defines a control region with the following information.

```
resInfo = ltePDCCHInfo(enb)

resInfo = struct with fields:
        NREG: 240
         NRE: 960
        NCCE: 26
    NREGUsed: 234
     NREUsed: 936
        MTot: 1920
    NSymbols: 2
```

The entire data block of padded, multiplexed PDCCHs needs to be 1920 bits, `resInfo.MTot`, in length. Using -1 to represent `NIL` padding "bits", create an "empty" multiplex.

```
pdcchs = -1*ones(1,resInfo.MTot);
```

Suppose you want to transmit all zeros in the first candidate of the UE-specific search space for PDCCH format 2 and the UE's `RNTI` = 1. For this format, a PDCCH spans 4 CCE or 288 bits, and the UE-specific search space contains two PDCCH candidates.

```
candidates = ltePDCCHSpace(enb,struct('PDCCHFormat',2,'RNTI',1))

candidates = 2x2 uint32 matrix

   1441   1728
      1    288
```

These location values arise for `enb.NSubframe = 0`. They change in a pseudorandom fashion as the subframe number increases. Since the default candidate indices define inclusive, 1-based bounds, we can use them to index the PDCCH data multiplex directly by using the MATLAB® colon operator.

```
pdcchs(candidates(1,1):candidates(1,2)) = 0;
```

This command sets the 288 bits of the first PDCCH candidate to all zeros. The second candidate actually falls within the common search space also.

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure with these fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NSubframe`** | Required | 0 (default), nonnegative scalar integer<br><br>0 is default | Subframe number |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NREG** | Optional | Nonnegative scalar integer | Total number of resource element groups (REGs) associated with PDCCHs, specified as a nonnegative scalar integer. |
| If NREG is absent, then `'enb'` must contain these fields. | | | |
| **NDLRB** | Optional | Numeric scalar value<br>6, 15, 25, 50, 75, and 100 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **CyclicPrefix** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **CFI** | Optional | 1, 2, 3 | Control format indicator value, specified as a double value. |
| **CellRefP** | Optional | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **Ng** | Optional | `'Sixth'`, `'Half'`, `'One'`, `'Two'` | HICH group multiplier |
| **DuplexMode** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| The following field is required only when `DuplexMode` is set to `'TDD'`. | | | |
| **TDDConfig** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |

Data Types: `struct`

**ue — UE-specific cell-wide settings**
structure

UE-specific cell-wide settings, specified as a structure with the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **PDCCHFormat** | Required | 0, 1, 2, 3 | PDCCH format |
| **RNTI** | Optional | Scalar integer between 0 and 65535 | Radio network temporary identifier (RNTI) value (16 bits). If RNTI field is missing, see Description section for specific behaviour. |

Data Types: `struct`

**opts — Index generation options**
character vector | cell array of character vectors | string array

Index generation options, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string):

| Option | Values | Description |
|---|---|---|
| Index base | `'1based'` (default), `'0based'` | Base value of the returned indices. Specify `'1based'` to generate indices where the first value is 1. Specify `'0based'` to generate indices where the first value is 0. |
| Indexing unit | `'bits'` (default), `'cce'` | Unit of the returned indices. Specify `'bits'` to indicate that the returned values correspond to bit indices. Specify `'cce'` to indicate that the returned values correspond to control channel elements (CCEs) indices. |

Example: `'1based bits'`, `"1based bits"`, `{'1based','bits'}`, or `["1based","bits"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `ind` — Search space PDCCH candidate indices

(0,2,4,6)-by-2 matrix

Search space PDCCH candidate indices, returned as a (0,2,4,6)-by-2 matrix given the structures `enb` and `ue`. It is a matrix of indices identifying a common or UE-specific PDCCH search space. Each two-element row contains the inclusive [*begin*,*end*] indices of a single PDCCH candidate location. By default, the one-based indices define the PDCCH locations in the block of all multiplexed PDCCH data bits to be transmitted in that subframe. `opts` defines alternative formats for returning the indices.

Data Types: `double`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.213. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePDCCH` | `ltePDCCHDecode` | `ltePDCCHIndices` | `ltePDCCHInterleave` | `ltePDCCHDeinterleave` | `ltePDCCHInfo` | `ltePDCCHSearch` | `ltePDCCHPRBS`

# ltePDSCH

Physical downlink shared channel

## Syntax

```
sym = ltePDSCH(enb,chs,cws)
```

## Description

`sym = ltePDSCH(enb,chs,cws)` returns a matrix containing the physical downlink shared channel (PDSCH) complex symbols for cell-wide settings, `enb`, channel transmission configuration, `chs`, and the codeword or codewords contained in `cws`. The channel processing includes the stages of scrambling, symbol modulation, layer mapping, and precoding.

## Examples

### Generate PDSCH symbols for Test Model E-TM1.1 10MHz

Generate the configuration structure for Test Model E-TM1.1 10 MHz, as specified in TS36.141

Initialize the test model using `lteTestModel`. Generate information related to PDSCH indices and use `info.Gd` output to determine the required transport block. Execute `lteDLSCH` to create the codeword, then generate the PDSCH symbols.

```
tm = lteTestModel('1.1','10MHz');
tm.PDSCH.RNTI = 0;
tm.PDSCH.RV = 0;

prbset = (0:tm.NDLRB-1)';
[ind,info] = ltePDSCHIndices(tm,tm.PDSCH,prbset);

trBlk  = randi([0,1],info.Gd,1);
cw = lteDLSCH(tm,tm.PDSCH,info.G,trBlk);
pdschSym = ltePDSCH(tm,tm.PDSCH,cw);
```

## Input Arguments

### enb — eNodeB cell-wide settings

structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NCellID`** | Required | Integer from 0 to 503 | Physical layer cell identity |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NSubframe** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **CellRefP** | Required | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **DuplexMode** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| The following parameters are dependent upon the condition that `DuplexMode` is set to `'TDD'`. | | | |
| **TDDConfig** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **SSC** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |
| The following parameter fields are dependent upon the condition that `chs.TxScheme` is set to `'SpatialMux'` or `'MultiUser'`. | | | |
| **CFI** | Required | 1, 2, or 3<br>Scalar or if the CFI varies per subframe, a vector of length 10 (corresponding to a frame). | Control format indicator (`CFI`) value. In TDD mode, `CFI` varies per subframe for the RMCs (`'R.0'`, `'R.5'`, `'R.6'`, `'R.6-27RB'`, `'R.12-9RB'`) |
| **NDLRB** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **CyclicPrefix** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |

**`chs` — Channel-specific transmission configuration**
structure

Channel-specific transmission configuration, specified as a structure that can contain the following parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **Modulation** | Required | `'QPSK'`, `'16QAM'`, `'64QAM'`, `'256QAM'`, `'1024QAM'` | Modulation type, specified as a character vector, cell array of character vectors, or string array. If blocks, each cell is associated with a transport block. |
| **RNTI** | Required | 0 (default), scalar integer | Radio network temporary identifier (RNTI) value (16 bits) |

<table>
<tr><th>Parameter Field</th><th>Required or Optional</th><th>Values</th><th>Description</th></tr>
<tr><td>TxScheme</td><td>Optional</td><td>'Port0' (default), 'TxDiversity', 'CDD', 'SpatialMux', 'MultiUser', 'Port5', 'Port7-8', 'Port8', 'Port7-14'.</td><td>PDSCH transmission scheme, specified as one of the following options.
<table>
<tr><th>Transmission scheme</th><th>Description</th></tr>
<tr><td>'Port0'</td><td>Single antenna port, port 0</td></tr>
<tr><td>'TxDiversity'</td><td>Transmit diversity</td></tr>
<tr><td>'CDD'</td><td>Large delay cyclic delay diversity scheme</td></tr>
<tr><td>'SpatialMux'</td><td>Closed loop spatial multiplexing</td></tr>
<tr><td>'MultiUser'</td><td>Multi-user MIMO</td></tr>
<tr><td>'Port5'</td><td>Single-antenna port, port 5</td></tr>
<tr><td>'Port7-8'</td><td>Single-antenna port, port 7, when NLayers = 1. Dual layer transmission, ports 7 and 8, when NLayers = 2.</td></tr>
<tr><td>'Port8'</td><td>Single-antenna port, port 8</td></tr>
<tr><td>'Port7-14'</td><td>Up to eight layer transmission, ports 7–14</td></tr>
</table></td></tr>
<tr><td colspan="4">The following parameters are dependent upon the condition that TxScheme is set to 'CDD', 'SpatialMux', 'MultiUser', 'Port7-8' or 'Port7-14'.</td></tr>
<tr><td>NLayers</td><td>Required</td><td>Integer from 1 to 8</td><td>Number of transmission layers.<br><br>The number of layers is dependent on TxScheme.</td></tr>
<tr><td>PMISet</td><td>Required</td><td>Integer vector with element values from 0 to 15.</td><td>Precoder matrix indication (PMI) set. It can contain either a single value, corresponding to single PMI mode, or multiple values, corresponding to multiple or subband PMI mode. The number of values depends on CellRefP, transmission layers and TxScheme. For more information about setting PMI parameters, see ltePMIInfo.</td></tr>
</table>

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **PRBSet** | Required | Integer column vector or two-column matrix | Zero-based physical resource block (PRB) indices corresponding to the slot wise resource allocations for this PDSCH. `PRBSet` can be assigned as:<br>• a column vector, the resource allocation is the same in both slots of the subframe,<br>• a two-column matrix, this parameter specifies different PRBs for each slot in a subframe,<br>• a cell array of length 10 (corresponding to a frame, if the allocated physical resource blocks vary across subframes).<br>PRBSet varies per subframe for the RMCs `'R.25'`(TDD), `'R.26'`(TDD), `'R.27'`(TDD), `'R.43'`(FDD), `'R.44'`, `'R.45'`, `'R.48'`, `'R.50'`, and `'R.51'`. |
| The following parameters are dependent upon the condition that `TxScheme` is set to `'Port5'`, `'Port7-8'`, `'Port8'`, or `'Port7-14'`. | | | |
| **W** | Optional | Numeric matrix, `[]` (default) | `NLayers`-by-*P* precoding matrix for the wideband UE-specific beamforming of the PDSCH symbols. *P* is the number of transmit antennas. When `W` is not specified, no precoding is applied. |

**`cws` — Codeword or codewords**
numeric vector | cell array

Codeword or codewords, specified as a vector of bit values for one codeword to be modulated, or a cell array containing one or two vectors of bit values corresponding to the one or two codewords to be modulated.

## Output Arguments

**`sym` — PDSCH symbols**
complex numeric matrix

PDSCH symbols, returned as a complex numeric matrix. It has size *N*-by-*P*, where *N* is the number of modulation symbols for one antenna port and *P* is the number of transmission antennas. The complex symbols are generated using cell-wide settings, `enb`, channel transmission configuration, `chs`, and the codeword or codewords contained in `cws`.

Data Types: `double`
Complex Number Support: Yes

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 36.141. “Evolved Universal Terrestrial Radio Access (E-UTRA); Base Station (BS) conformance testing.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

## See Also

`ltePDSCHDecode` | `ltePDSCHIndices` | `ltePDSCHPRBS` | `lteDLSCH`

# ltePDSCHDecode

Physical downlink shared channel decoding

## Syntax

```
[cws,symbols] = ltePDSCHDecode(enb,chs,sym)
[cws,symbols] = ltePDSCHDecode(enb,chs,sym,hest,noiseest)
[cws,symbols] = ltePDSCHDecode(enb,chs,rxgrid,hest,noiseest)
```

## Description

`[cws,symbols] = ltePDSCHDecode(enb,chs,sym)` performs the inverse of physical downlink shared channel (PDSCH) processing on the matrix of complex modulated PDSCH symbols, `sym`, using cell-wide settings structure, `enb`, and channel-specific configuration structure, `chs`. The channel inverse processing includes inverting the channel precoding, layer demapping and codeword separation, soft demodulation, and descrambling. Inverting the precoding is accomplished by matrix pseudoinversion of the precoding matrices. It returns a cell array, `cws`, of soft bit vectors, and a cell array, `symbols`, of received constellation symbol vectors resulting from performing the inverse of Physical Downlink Shared Channel (PDSCH) processing. For more information, see TS 36.211 [1], Section 6.4 and `ltePDSCH`. `cws` is optionally scaled by channel state information (CSI) calculated during the equalization process.

`[cws,symbols] = ltePDSCHDecode(enb,chs,sym,hest,noiseest)` performs the decoding of the complex modulated PDSCH symbols `sym` using cell-wide settings, `enb`, channel-specific configuration, `chs`, channel estimate, `hest`, and the noise estimate, `noiseest`.

The behavior varies based on the `chs.TxScheme` setting. For the `'TxDiversity'` transmission scheme, the precoding inversion is performed using an orthogonal space frequency block code (OSFBC) decoder. For the `'SpatialMux'`, `'CDD'`, and `'MultiUser'` transmission schemes, the precoding inversion is performed using a multiple-input, multiple-output (MIMO) minimum mean square error (MMSE) equalizer, equalizing between transmitted and received layers. For the `'Port0'`, `'Port5'`, `'Port7-8'`, `'Port8'`, and `'Port7-14'` transmission schemes, the reception is performed using MMSE equalization. The input channel estimate, `hest`, is assumed to be with reference to the transmission layers, using the UE-specific reference signals, so the MMSE equalization will produce MMSE equalized layers.

`noiseest` is an estimate of the noise power spectral density per RE on the received subframe. This estimate is provided by the `lteDLChannelEstimate` function.

`[cws,symbols] = ltePDSCHDecode(enb,chs,rxgrid,hest,noiseest)` accepts the full received resource grid, `rxgrid`, for one subframe, in place of the `sym` input; the decoder will internally extract the PDSCH REs to obtain the complex modulated PDSCH symbols. `rxgrid` is a 3-D *M*-by-*N*-by-`NRxAnts` array of resource elements, where *M* and *N* are the number of subcarriers and symbols for one subframe for cell-wide settings `enb` and `NRxAnts` is the number of receive antennas. In this case, `hest` is a 4-D *M*-by-*N*-by-`NRxAnts`-by-`CellRefP` array where *M* and *N* are the number of subcarriers and symbols for one subframe for cell-wide settings `enb`, `NRxAnts` is the number of receive antennas, and `CellRefP` is the number of cell-specific reference signal antenna ports, given by `enb.CellRefP`. `hest` is processed to extract the channel estimates relevant to the PDSCH, those in the time and frequency locations corresponding to the PDSCH REs in `rxgrid`.

## Examples

### Decode PDSCH Symbols

Generate and decode PDSCH symbols.

Initialize cell parameter structure `enb` for RMC R.0.

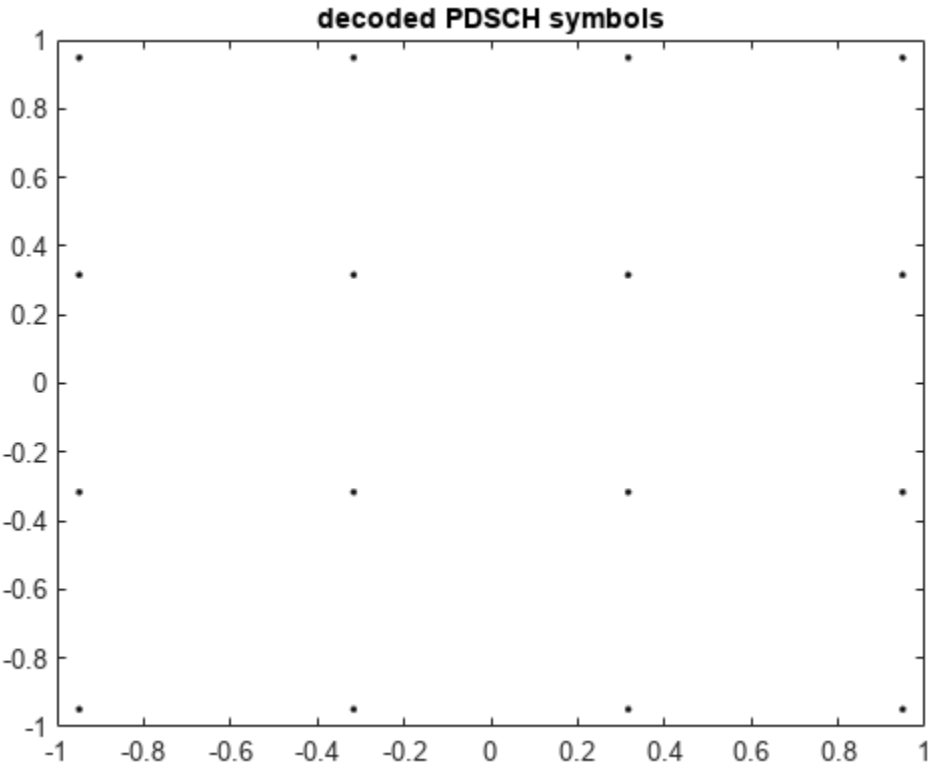
```
enb = lteRMCDL('R.0');
```

Populate a complex codeword matrix and generate modulated PDSCH symbols.

```
codewordBits = randi([0,1],enb.PDSCH.CodedTrBlkSizes(1),1);

pdschSym = ltePDSCH(enb,enb.PDSCH,codewordBits);
```

Decode and plot the PDSCH symbols.

```
[rxCodewords,rxSymbols] = ltePDSCHDecode(enb,enb.PDSCH,pdschSym);

plot (rxSymbols{:},'k.')
title('decoded PDSCH symbols')
```




Show size and first 5 elements of output codewords to be modulated, `rxCws`, and received symbols, `symbols`.

```
size_rxCodewords = size(rxCodewords{:})
```

```
size_rxCodewords = 1×2

   504     1
```

```
rxCodewords{1}(1:1:5)

ans = 5×1

    0.9487
    0.9487
   -0.3162
    0.3162
    0.3162
```

```
size_rxSymbols = size(rxSymbols{:})

size_rxSymbols = 1×2

   126     1
```

```
rxSymbols{1}(1:5)

ans = 5×1 complex

  -0.9487 - 0.9487i
  -0.3162 + 0.9487i
  -0.3162 - 0.9487i
  -0.3162 - 0.3162i
   0.9487 - 0.9487i
```

## Input Arguments

### enb — eNodeB cell-wide settings

structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NCellID`** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **`NSubframe`** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **`CellRefP`** | Required | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **`DuplexMode`** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| The following parameters are dependent upon the condition that `DuplexMode` is set to `'TDD'`. | | | |
| **`TDDConfig`** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **`SSC`** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |
| The following parameter fields are dependent upon the condition that `chs.TxScheme` is set to `'SpatialMux'` or `'MultiUser'`. | | | |
| **`NDLRB`** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **`CFI`** | Required | 1, 2, or 3<br>Scalar or if the CFI varies per subframe, a vector of length 10 (corresponding to a frame). | Control format indicator (`CFI`) value. In TDD mode, `CFI` varies per subframe for the RMCs (`'R.0'`, `'R.5'`, `'R.6'`, `'R.6-27RB'`, `'R.12-9RB'`) |
| **`CyclicPrefix`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |

**`chs` — Channel-specific transmission configuration**
structure

Channel-specific transmission configuration, specified as a structure that can contain the following parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`Modulation`** | Required | `'QPSK'`, `'16QAM'`, `'64QAM'`, `'256QAM'`, `'1024QAM'` | Modulation type, specified as a character vector, cell array of character vectors, or string array. If blocks, each cell is associated with a transport block. |
| **`RNTI`** | Required | 0 (default), scalar integer | Radio network temporary identifier (RNTI) value (16 bits) |

<table>
<tr><th>Parameter Field</th><th>Required or Optional</th><th>Values</th><th>Description</th></tr>
<tr><td>TxScheme</td><td>Required</td><td>'Port0', 'TxDiversity', 'CDD', 'SpatialMux', 'MultiUser', 'Port5', 'Port7-8', 'Port8', 'Port7-14'.</td><td>PDSCH transmission scheme, specified as one of the following options.
<table>
<tr><th>Transmission scheme</th><th>Description</th></tr>
<tr><td>'Port0'</td><td>Single antenna port, port 0</td></tr>
<tr><td>'TxDiversity'</td><td>Transmit diversity</td></tr>
<tr><td>'CDD'</td><td>Large delay cyclic delay diversity scheme</td></tr>
<tr><td>'SpatialMux'</td><td>Closed loop spatial multiplexing</td></tr>
<tr><td>'MultiUser'</td><td>Multi-user MIMO</td></tr>
<tr><td>'Port5'</td><td>Single-antenna port, port 5</td></tr>
<tr><td>'Port7-8'</td><td>Single-antenna port, port 7, when NLayers = 1. Dual layer transmission, ports 7 and 8, when NLayers = 2.</td></tr>
<tr><td>'Port8'</td><td>Single-antenna port, port 8</td></tr>
<tr><td>'Port7-14'</td><td>Up to eight layer transmission, ports 7–14</td></tr>
</table></td></tr>
<tr><td>NLayers</td><td>Required</td><td>Integer from 1 to 8</td><td>Number of transmission layers.</td></tr>
<tr><td>CSI</td><td>Optional</td><td>'Off' (default), 'On'</td><td>Flag provides control over weighting the soft values that are used to determine the output values with the channel state information (CSI) calculated during the equalization process. If 'On', soft values are weighted by CSI.</td></tr>
<tr><td colspan="4">The following parameters are dependent upon the condition that TxScheme is set to 'SpatialMux' or 'MultiUser'.</td></tr>
<tr><td>    PMISet</td><td>Required</td><td>Integer vector with element values from 0 to 15.</td><td>Precoder matrix indication (PMI) set. It can contain either a single value, corresponding to single PMI mode, or multiple values, corresponding to multiple or subband PMI mode. The number of values depends on CellRefP, transmission layers and TxScheme. For more information about setting PMI parameters, see ltePMIInfo.</td></tr>
</table>

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **PRBSet** | Required | Integer column vector or two-column matrix | Zero-based physical resource block (PRB) indices corresponding to the slot wise resource allocations for this PDSCH. `PRBSet` can be assigned as:<br>• a column vector, the resource allocation is the same in both slots of the subframe,<br>• a two-column matrix, this parameter specifies different PRBs for each slot in a subframe,<br>• a cell array of length 10 (corresponding to a frame, if the allocated physical resource blocks vary across subframes).<br>PRBSet varies per subframe for the RMCs `'R.25'`(TDD), `'R.26'`(TDD), `'R.27'`(TDD), `'R.43'`(FDD), `'R.44'`, `'R.45'`, `'R.48'`, `'R.50'`, and `'R.51'`. |
| The following parameters are dependent upon the condition that `TxScheme` is set to `'Port5'`, `'Port7-8'`, `'Port8'`, or `'Port7-14'`. | | | |
| **W** | Optional | Numeric matrix, `[]` (default) | `NLayers`-by-*P* precoding matrix for the wideband UE-specific beamforming of the PDSCH symbols. *P* is the number of transmit antennas. When `W` is not specified, no precoding is applied. |

**`sym` — Complex modulated PDSCH symbols**
numeric matrix

Complex modulated PDSCH symbols, specified as a numeric matrix of size `NRE`-by-`NRxAnts`. `NRE` is the number of QAM symbols per antenna assigned to the PDSCH and `NRxAnts` is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

**`hest` — Channel estimate**
3-D numeric array | 4-D numeric array

Channel estimate, specified as a 3-D or 4-D numeric array. For the `'Port0'`, `'TxDiversity'`, `'SpatialMux'`, `'CDD'`, and `'MultiUser'` transmission schemes, the array size is `NRE`-by-`NRxAnts`-by-`CellRefP`, where `NRE` is the number of QAM symbols per antenna assigned to the PDSCH, `NRxAnts` is the number of receive antennas, and `CellRefP` is the number of cell-specific reference signal antennas, given by `enb.CellRefP`. For the `'Port5'`, `'Port7-8'`, `'Port8'`, and `'Port7-14'` transmission schemes, the array size is `NRE`-by-`NRxAnts`-by-`NLayers`, where `NLayers` is the number of transmission layers given by `chs.NLayers`.

When `rxgrid` is supplied, `hest` is a 4-D numeric array of size *M*-by-*N*-by-`NRxAnts`-by-`CellRefP`, where *M* and *N* are the number of subcarriers and symbols for one subframe for cell-wide settings,

`enb`, `NRxAnts` is the number of receive antennas, and `CellRefP` is the number of cell-specific reference signal antenna ports, given by `enb.CellRefP`.

Data Types: `double`

**`noiseest` — Noise estimate**
numeric array

Noise estimate of the noise power spectral density per RE on the received subframe, specified as a numeric array.

Data Types: `double`

**`rxgrid` — Full received resource grid**
numeric array

Full received resource grid, specified as a 3-D *M*-by-*N*-by-`NRxAnts` array of resource elements, where *M* and *N* are the number of subcarriers and symbols for one subframe for cell-wide settings `enb` and `NRxAnts` is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

## Output Arguments

**`cws` — Codeword or codewords**
cell array

Codeword or codewords, returned as a cell array containing one or two vectors of bit values corresponding to the one or two codewords to be modulated.

Data Types: `double`

**`symbols` — Received constellation symbols**
cell array of column vectors

Received constellation symbols, returned as a cell array of complex double column vectors, resulting from performing the inverse of PDSCH processing.

Data Types: `cell`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePDSCH` | `ltePDSCHIndices` | `ltePDSCHPRBS` | `lteDLSCHDecode`

# ltePDSCHIndices

Physical downlink shared channel (PDSCH) resource element indices

## Syntax

```
[ind,info] = ltePDSCHIndices(enb,chs,prbset)
[ind,info] = ltePDSCHIndices(enb,chs,prbset,opts)
```

## Description

`[ind,info] = ltePDSCHIndices(enb,chs,prbset)` returns a matrix, `ind`, containing physical downlink shared channel (PDSCH) resource element (RE) indices and a structure, `info`, containing information related to the PDSCH indices. By default, the output indices are a one-based linear indexed 3D array representing the subframe resource element grid for all antenna ports. You can use `ind` to index elements of the subframe resource grid directly for all antenna ports. This function is initialized with cell-wide settings, `enb`, channel transmission configuration, `chs`, and physical resource block indices, `prbset`.

`prbset` contains the physical resource block (PRB) indices corresponding to the resource allocation for this PDSCH transmission. You can specify `prbset` as either a column vector or a two-column matrix. If you specify a column vector, the resource allocation is the same in both slots of the subframe. If the PRBs in the first and second slots of the subframe differ, you can use the two-column matrix to specify PRBs. The PRB indices are zero-based.

Each column of the returned *N*-by-*P* matrix, `ind`, contains the per-antenna indices for the *N* resource elements in each of the *P* resource array planes. For the `'Port0'`, `'TxDiversity'`, `'CDD'`, `'SpatialMux'`, and `'MultiUser'` transmission schemes, *P* = `enb.CellRefP`. For the other transmission schemes, *P* = `chs.NTxAnts`. If `chs.NTxAnts = 0` or is absent, `ind` is an *N*-by-*NU* matrix containing the per-layer indices for the *N* resource elements in each of *NU* resource array planes. The planes are associated with the layers, where *NU* = `chs.NLayers`.

The `info` structure contains parameter fields `G` and `Gd`. `info.G` provides the appropriate size of the DL-SCH coder output, which is required as the parameter `outlen` provided to the `lteDLSCH` function. `info.Gd` is the number of coded and rate-matched DL-SCH data symbols per layer, equal to the number of rows in the PDSCH indices. To provide accurate information in `info`, the `Modulation`, `TxScheme`, and `Nlayers` fields are required in `chs`.

**Note** The `Modulation` and `Nlayers` fields are required only if the `info` output is assigned when you call the function.

`[ind,info] = ltePDSCHIndices(enb,chs,prbset,opts)` formats the returned indices using options specified by `opts`.

## Examples

### Generate PDSCH RE Indices

This example generates the 0-based PDSCH resource element (RE) indices mapping in linear index form for the 4-antenna case.

Create the cell-wide settings structure, `enb`.

```
enb = lteRMCDL('R.14');
enb.NDLRB = 6;
enb.CFI = 1;
enb.PDSCH.PRBSet = (1:enb.NDLRB-1).';
```

Generate PDSCH RE indices, specifying the 0-based and linear options.

```
ind = ltePDSCHIndices(enb,enb.PDSCH, ...
    enb.PDSCH.PRBSet,{'0based','ind'});
ind(1:10,:)
```

```
ans = 10x4 uint32 matrix

    156   1164   2172   3180
    157   1165   2173   3181
    158   1166   2174   3182
    159   1167   2175   3183
    160   1168   2176   3184
    161   1169   2177   3185
    162   1170   2178   3186
    163   1171   2179   3187
    164   1172   2180   3188
    165   1173   2181   3189
```

The result, `ind`, is a matrix of 0-based mapping indices in linear index form. Since this is example is for the 4-antenna case, `ind`, has 4 columns.

## Input Arguments

### enb — eNodeB cell-wide settings

structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NDLRB`** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **`NCellID`** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **`CyclicPrefix`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **`CellRefP`** | Required | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **`NSubframe`** | Required | 0 (default), nonnegative scalar integer | Subframe number |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **CFI** | Required | 1, 2, or 3<br>Scalar or if the CFI varies per subframe, a vector of length 10 (corresponding to a frame). | Control format indicator (`CFI`) value. In TDD mode, `CFI` varies per subframe for the RMCs (`'R.0'`, `'R.5'`, `'R.6'`, `'R.6-27RB'`, `'R.12-9RB'`) |
| **DuplexMode** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as one of the following:<br>• `'FDD'` — Frequency division duplex (default)<br>• `'TDD'` — Time division duplex |
| The following apply when `DuplexMode` is set to `'TDD'`. | | | |
| **TDDConfig** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **SSC** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |

The following table shows `enb` structure fields only required when the transmission scheme `chs.TxScheme` is set to `'Port7-14'`.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NFrame** | Optional | 0 (default), nonnegative scalar integer | Frame number |
| **CSIRSPeriod** | Optional | `'Off'` (default), `'On'`, `Icsi-rs` (0,...,154), `[Tcsi-rs Dcsi-rs]`. You can also specify values in a cell array of configurations for each resource. | CSI-RS subframe configurations for one or more CSI-RS resources. Multiple CSI-RS resources can be configured from a single common subframe configuration or from a cell array of configurations for each resource. |
| The following CSI-RS resource parameters apply only when `CSIRSPeriod` sets one or more CSI-RS subframe configurations to any value other than `'Off'`. Each parameter length must be equal to the number of CSI-RS resources required. | | | |
| **CSIRSConfig** | Required | Nonnegative scalar integer | Array CSI-RS configuration indices. See TS 36.211, Table 6.10.5.2-1. |
| **CSIRefP** | Required | 1 (default), 2, 4, 8 | Array of number of CSI-RS antenna ports |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **ZeroPowerCSIRSPeriod** | Optional | `'Off'` (default), `'On'`, `Icsi-rs` (0,...,154), `[Tcsi-rs Dcsi-rs]`. You can also specify values in a cell array of configurations for each resource. | Zero power CSI-RS subframe configurations for one or more zero power CSI-RS resource configuration index lists. Multiple zero power CSI-RS resource lists can be configured from a single common subframe configuration or from a cell array of configurations for each resource list. |
| The following zero power CSI-RS resource parameter is only required if one or more of the above zero power subframe configurations is set to any value other than `'Off'`. | | | |
| **ZeroPowerCSIRSConfig** | Required | 16-bit bitmap character vector or string scalar (truncated if not 16 bits or `'0'` MSB extended), or a numeric list of CSI-RS configuration indices. You can also specify values in a cell array of configurations for each resource. | Zero power CSI-RS resource configuration index lists (TS 36.211 Section 6.10.5.2). Specify each list as a 16-bit bitmap character vector or string scalar (if less than 16 bits, then `'0'` MSB extended), or as a numeric list of CSI-RS configuration indices from TS 36.211 Table 6.10.5.2-1 in the `'4'` CSI reference signal column. Multiple lists can be defined using a cell array of individual lists.<br><br>See [1]. |

Data Types: `struct`

**`chs` — PDSCH-specific channel transmission configuration**
structure

PDSCH-specific channel transmission configuration, specified as a structure that can contain the following parameter fields.

<table>
<tr><th>Parameter Field</th><th>Required or Optional</th><th>Values</th><th>Description</th></tr>
<tr><td>TxScheme</td><td>Optional</td><td>'Port0' (default), 'TxDiversity', 'CDD', 'SpatialMux', 'MultiUser', 'Port5', 'Port7-8', 'Port8', 'Port7-14'.</td><td>PDSCH transmission scheme, specified as one of the following options.
<table>
<tr><th>Transmission scheme</th><th>Description</th></tr>
<tr><td>'Port0'</td><td>Single antenna port, port 0</td></tr>
<tr><td>'TxDiversity'</td><td>Transmit diversity</td></tr>
<tr><td>'CDD'</td><td>Large delay cyclic delay diversity scheme</td></tr>
<tr><td>'SpatialMux'</td><td>Closed loop spatial multiplexing</td></tr>
<tr><td>'MultiUser'</td><td>Multi-user MIMO</td></tr>
<tr><td>'Port5'</td><td>Single-antenna port, port 5</td></tr>
<tr><td>'Port7-8'</td><td>Single-antenna port, port 7, when NLayers = 1. Dual layer transmission, ports 7 and 8, when NLayers = 2.</td></tr>
<tr><td>'Port8'</td><td>Single-antenna port, port 8</td></tr>
<tr><td>'Port7-14'</td><td>Up to eight layer transmission, ports 7–14</td></tr>
</table></td></tr>
<tr><td colspan="4">The following parameters apply when TxScheme is set to 'Port5', 'Port7-8', 'Port8', or 'Port7-14'.</td></tr>
<tr><td>NTxAnts</td><td>Optional</td><td>Nonnegative integer, 0 (default)</td><td>Number of transmission antenna ports. This argument is present only for UE-specific demodulation reference symbols.</td></tr>
<tr><td colspan="4">To provide accurate information in info, you are required to define TxScheme and the following additional parameters. These fields are only required when info is output.</td></tr>
<tr><td>Modulation</td><td>Optional</td><td>'QPSK' (default), '16QAM', '64QAM', '256QAM', '1024QAM'</td><td>Codeword modulation format, specified as a character vector or string scalar for one codeword, or as cell array or string array for two codewords.</td></tr>
<tr><td>NLayers</td><td>Optional</td><td>1 (default), 2, 3, 4, 5, 6, 7, 8</td><td>Number of transmission layers.<br><br>The number of layers is dependent on TxScheme.</td></tr>
</table>

Data Types: `struct`

**`prbset` — Physical resource block indices**
column vector | 2-column numeric matrix

Physical resource block indices, specified as a column vector or a two-column numeric matrix. This argument contains the Physical Resource Block (PRB) indices corresponding to the resource allocation for this PDSCH transmission. If you specify a column vector, the resource allocation is the same in both slots of the subframe. If the PRBs in the first and second slots of the subframe differ, you can use the two-column matrix to specify PRBs. The PRB indices are zero-based.

Data Types: `double`

**`opts` — Output format options for resource element indices**
character vector | cell array of character vectors | string array

Output format options for resource element indices, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string) :

| Category | Options | Description |
|---|---|---|
| Indexing style | `'ind'` (default) | The returned indices are in linear index style. |
| | `'sub'` | The returned indices are in `[subcarrier,symbol,port]` subscript row style. |
| Index base | `'1based'` (default) | The returned indices are one-based. |
| | `'0based'` | The returned indices are zero-based. |

Example: `'ind 1based'`, `"ind 1based"`, `{'ind','1based'}`, or `["ind","1based"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

**`ind` — Physical downlink shared channel (PDSCH) resource element (RE) indices**
matrix

Physical downlink shared channel (PDSCH) resource element (RE) indices, specified as a matrix. Each column of the *N*-by-*P* matrix, `ind`, contains the per-antenna indices for the *N* resource elements in each of the *P* resource array planes. For the `'Port0'`, `'TxDiversity'`, `'CDD'`, `'SpatialMux'`, and `'MultiUser'` transmission schemes, *P* = `enb.CellRefP`. For the other transmissions schemes, *P* = `chs.NTxAnts`. If `chs.NTxAnts = 0` or is absent, the `ind` matrix is of size *N*-by-*NU*. In this case, `ind`contains the per-layer indices for the *N* resource elements in each of *NU* resource array planes associated with the layers, where *NU* = `chs.NLayers`. You can return the indices in alternative indexing formats using the argument `opts`.

**Note** The active or zero-power CSI-RS resource elements are excluded from the output indices only for the Release 10/11, `'Port7-14'` transmission scheme. For all other schemes, the CSI-RS resource

element indices are not avoided, which results in a Release 8/9 compatible PDSCH. Any active or zero-power CSI-RS would overwrite the associated PDSCH REs later in the subframe construction.

### `info` — Information related to PDSCH indices

structure

Information related to PDSCH indices, returned as a structure. To provide accurate information in `info`, the channel transmission configuration structure, `chs`, must contain the fields `TxScheme`, `Modulation`, and `NLayers`. The structure `info` has the following fields.

| Parameter Field | Description | Values | Data Type |
|---|---|---|---|
| **`G`** | Number of coded and rate-matched DL-SCH data bits for each codeword. | one or two element vector | `uint32` |
| **`Gd`** | Number of coded and rate-matched DL-SCH data symbols per layer. | Integer equal to the number of rows in the PDSCH indices | `uint32` |

## Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePDSCH` | `ltePDSCHDecode` | `ltePDSCHPRBS`

# ltePDSCHPRBS

PDSCH pseudorandom scrambling sequence

## Syntax

```
[seq,cinit] = ltePDSCHPRBS(enb,rnti,cwIndex,n)
[seq,cinit] = ltePDSCHPRBS(enb,rnti,cwIndex,n,mapping)

[subseq,cinit] = ltePDSCHPRBS(enb,rnti,cwIndex,pn)
[subseq,cinit] = ltePDSCHPRBS(enb,rnti,cwIndex,pn,mapping)
```

## Description

`[seq,cinit] = ltePDSCHPRBS(enb,rnti,cwIndex,n)` returns a column vector containing the first `n` outputs of the Physical Downlink Shared Channel (PDSCH) scrambling sequence. It also returns an initialization value `cinit` for the pseudorandom binary sequence (PRBS) generator. The function is initialized according to cell-wide settings, `enb`, 16-bit `rnti`, and `cwIndex`, indicating which codeword this sequence scrambles.

`[seq,cinit] = ltePDSCHPRBS(enb,rnti,cwIndex,n,mapping)` allows control over the format of the returned sequence `seq` with the input `mapping`.

`[subseq,cinit] = ltePDSCHPRBS(enb,rnti,cwIndex,pn)` returns a subsequence of a full PRBS sequence, specified by `pn`.

`[subseq,cinit] = ltePDSCHPRBS(enb,rnti,cwIndex,pn,mapping)` allows additional control over the format of the returned subsequence, `subseq`, with the input `mapping`

## Examples

### Scramble Codeword Using PDSCH PRBS

Scramble the contents of a codeword using the PDSCH scrambling sequence.

Create cell-wide configuration structure for reference channel R.0, get PDSCH indices, and a create codeword.

```
enb = lteRMCDL('R.0');
pdsch = enb.PDSCH;
[~,pdschInfo] = ltePDSCHIndices(enb,pdsch,pdsch.PRBSet);
codedTrBlkSize = pdschInfo.G;
cw = randi([0 1],codedTrBlkSize,1);
```

Generate PDSCH scrambling sequence, and scramble the codeword using the PDSCH scrambling sequence.

```
RNTI = 11;
ncw = 0;
pdschPrbsSeq = ltePDSCHPRBS(enb,RNTI,ncw,length(cw));
scrambled = xor(pdschPrbsSeq, cw);
```

### Scramble Two Codewords Using PDSCH PRBS

Scramble the contents of two codewords using the PDSCH scrambling sequence. When transmitting multiple codewords via spatial multiplexing, each codeword uses a different scrambling sequence.

Create cell-wide configuration structure for reference channel R.14, get PDSCH indices.

```
rmc.RC = 'R.14';
rmc = lteRMCDL(rmc,2);
pdsch = rmc.PDSCH;
[~,pdschInfo] = ltePDSCHIndices(rmc,pdsch,pdsch.PRBSet);
codedTrBlkSize1 = pdschInfo.G(1);
codedTrBlkSize2 = pdschInfo.G(2);
cws{1} = randi([0 1],codedTrBlkSize1,1);
cws{2} = randi([0 1],codedTrBlkSize2,1);
```

Generate PDSCH scrambling sequences for two codewords, and scramble the codewords using the PDSCH scrambling sequences.

```
RNTI = 11;
ncw = 0;
pdschPrbsSeq1 = ltePDSCHPRBS(rmc,RNTI,ncw,length(cws{1}));
ncw = 1;
pdschPrbsSeq2 = ltePDSCHPRBS(rmc,RNTI,ncw,length(cws{2}));
scrambled1 = xor(pdschPrbsSeq1, cws{1});
scrambled2 = xor(pdschPrbsSeq2, cws{2});
```

## Input Arguments

**`enb` — Cell-wide settings**
structure

Cell-wide settings, specified as a structure. `enb` contains the following fields.

**`NCellID` — Physical layer cell identity**
nonnegative integer

Physical layer cell identity, specified as a nonnegative integer.

Data Types: `double`

**`NSubframe` — Subframe number**
nonnegative integer

Subframe number, specified as a nonnegative integer.

Data Types: `double`

Data Types: `struct`

**`rnti` — Radio network temporary identifier**
nonnegative integer

Radio network temporary identifier, specified as nonnegative integer.

Data Types: `double`

**`cwIndex` — Codeword index**
0 | 1

Codeword index, specified as a 0 or 1. This input indicates which codeword this sequence scrambles.

Data Types: `double`

**`n` — Length of scrambling sequence**
positive integer

Length of scrambling sequence, specified as a positive integer.

Data Types: `double`

**`pn` — Range of scrambling subsequence**
row vector

Range of scrambling subsequence, `subseq`, specified as a row vector of `[p n]`. The subsequence returns `n` values of the PRBS generator, starting at position `p` (0-based).

Data Types: `double`

**`mapping` — Output sequence formatting**
`'binary'` (default) | `'signed'`

Output sequence formatting, specified as `'binary'` or `'signed'`. `mapping` controls the format of the returned sequence.

- `'binary'` maps `true` to 1 and `false` to 0.
- `'signed'` maps `true` to –1 and `false` to 1.

Data Types: `char` | `string`

## Output Arguments

**`seq` — PDSCH scrambling sequence**
logical column vector | numeric column vector

PDSCH scrambling sequence, returned as a logical column vector or a numeric column vector. `seq` argument contains the first `n` outputs of the PDSCH pseudorandom scrambling sequence. If `mapping` is set to `'signed'`, `seq` is a vector of data type double. Otherwise, it is a vector of data type logical.

Data Types: `logical` | `double`

**`subseq` — PDSCH scrambling subsequence**
logical column vector | numeric column vector

PDSCH scrambling subsequence, returned as a logical column vector or a numeric column vector. `subseq` contains the values of the PRBS generator specified by `pn`. If you set `mapping` to `'signed'`, the output data type is `double`. Otherwise, the output data type is `logical`

Data Types: `logical` | `double`

**`cinit` — Initialization value for PRBS generator**
numeric scalar

Initialization value for PRBS generator, returned as a numeric scalar.

Data Types: `uint32`

## Version History

**Introduced in R2014a**

## See Also

`ltePDSCH` | `ltePDSCHDecode` | `ltePDSCHIndices`

# ltePHICH

Physical hybrid ARQ indicator channel

## Syntax

```
[sym] = ltePHICH(enb,hiset)
[sym,info] = ltePHICH(enb,hiset)
```

## Description

`[sym] = ltePHICH(enb,hiset)` returns a matrix, `sym`, of symbols generated by the set of physical hybrid ARQ indicator channels (PHICH) in a subframe, given the cell-wide settings configuration structure, `enb`, and HARQ indicator set, `hiset`. For more information, see “PHICH Generation” on page 2-640.

`[sym,info] = ltePHICH(enb,hiset)` also returns an `info` structure, containing control resourcing information about the output symbols.

## Examples

### Generate PHICH Symbols

Generate physical HARQ indicator channel (PHICH) symbols for three different HARQ indicator (HI) sets. An HI set is comprised of the PHICH group index number, the sequence number within the group, and an ACK/NACK.

Create a cell-wide settings configuration structure with a single antenna (`enb.CellRefP=1`), normal CP, fifty downlink resource blocks (`enb.NDLRB=50`), and a one sixth HICH group multiplier (`enb.Ng='Sixth'`). For this system configuration, 16 PHICH are available. The available PHICH are split between two groups of eight sequences. The sequences are mapped to `info.NRE=24` resource elements. Calling `lteRMCDL` with RMC R.7 provides the desired configuration structure.

Generate PHICH symbols with various HI set configurations.

#### PHICH Symbols for Empty HI Set

Generate `ltePHICH` output for an empty HI set.

```
enb = lteRMCDL('R.7');
[sym,info] = ltePHICH(enb,[])
```

```
sym = 24×1 complex

   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
```

```
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
      ⋮


info = struct with fields:
             NREG: 6
              NRE: 24
           NPHICH: 16
          NGroups: 2
    NMappingUnits: 2
       NSequences: 8
    PHICHDuration: 1
      ActiveHISet: []


sizeSym = size(sym)

sizeSym = 1×2

    24     1
```

`ltePHICH` returns an NRE-by-CellRefP matrix of zeros. The transmission configuration is one antenna, as shown by the second dimension of the `sym` matrix. The `info` output structure is populated based on inputs to `ltePHICH`.

### PHICH Symbols for Single HI Set

Modulate a NACK (`hi=0`) onto the third orthogonal sequence (`nSeq=2`) of the second group (`nGroup=1`). Generate PHICH symbols specifying the HI set as [nGroup=1 nSeq=2 hi=0].

```
enb = lteRMCDL('R.7');
sym = ltePHICH(enb,[1 2 0])

sym = 24×1 complex

   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
      ⋮


sizeSym = size(sym)

sizeSym = 1×2

    24     1
```

The result remains an NRE-by-CellRefP matrix.

### PHICH Symbols for Multiple HI Sets

For the second PHICH, add an ACK (`hi=1`) on the last sequence (`nSeq=7`) of the first group (`nGroup=0`). Generate PHICH symbols specifying the HI sets as [[nGroup=1 nSeq=2 hi=0]; [nGroup=0 nSeq=7 hi=1]].

```
enb = lteRMCDL('R.7');
[sym,info] = ltePHICH(enb,[[1 2 0];[0 7 1]])
```

```
sym = 24×1 complex

   0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
      ⋮


info = struct with fields:
             NREG: 6
              NRE: 24
           NPHICH: 16
          NGroups: 2
    NMappingUnits: 2
       NSequences: 8
    PHICHDuration: 1
      ActiveHISet: [2x3 double]
```

```
sizeSym = size(sym)
```

```
sizeSym = 1×2

    24     1
```

The result remains an NRE-by-CellRefP matrix.

## Input Arguments

### enb — Cell-wide settings

scalar structure

Cell-wide settings, specified as a scalar structure. `enb` contains the following fields.

### NDLRB — Number of downlink resource blocks

positive integer

Number of downlink resource blocks, specified as a positive integer.

Data Types: `double`

**`NCellID` — Physical layer cell identity**
nonnegative integer

Physical layer cell identity, specified as a nonnegative integer.

Data Types: `double`

**`CellRefP` — Number of cell-specific reference signal antenna ports**
1 | 2 | 4

Number of cell-specific reference signal antenna ports, specified as 1, 2, or 4.

Data Types: `double`

**`CyclicPrefix` — Cyclic prefix length**
`'Normal'` (default) | optional | `'Extended'`

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

**`NSubframe` — Subframe number**
nonnegative integer

Subframe number, specified as a nonnegative integer.

Data Types: `double`

**`Ng` — HICH group multiplier**
`'Sixth'` | `'Half'` | `'One'` | `'Two'`

HICH group multiplier, specified as `'Sixth'`, `'Half'`, `'One'`, or `'Two'`.

Data Types: `char` | `string`

**`DuplexMode` — Duplex mode**
`'FDD'` (default) | optional | `'TDD'`

Duplex mode, specified as `'FDD'` or `'TDD'`.

Data Types: `char` | `string`

**`TDDConfig` — Uplink or downlink configuration**
0 (default) | optional | 1 | 2 | 3 | 4 | 5 | 6

Uplink or downlink configuration, specified as a nonnegative scalar integer from 0 through 6. This argument is only required if `DuplexMode` is set to `'TDD'`.

Data Types: `double`

Data Types: `struct`

**`hiset` — HARQ indicator set**
numeric matrix | [ ]

HARQ indicator set, specified as an *R*-by-3 numeric matrix. *R* is the number of rows. Each row of `hiset` defines a single PHICH in terms of `[nGroup,nSeq,hi]`.

- `nGroup` is the PHICH group index number.
- `nSeq` is the sequence index number within the group.
- `hi` is 1 or 0 representing hybrid ARQ indicator ACK or NACK, respectively.

The `nGroup` and `nSeq` indices are zero-based. For more information, see “PHICH Generation” on page 2-640.

Data Types: `double`
Complex Number Support: Yes

## Output Arguments

### `sym` — PHICH symbols

numeric matrix

PHICH symbols, returned as an `NRE`-by-`CellRefP` numeric matrix. `NRE` is the number of resource elements, and `CellRefP` is the number of cell-specific reference signal antenna ports. Each column of `sym` contains the per-antenna symbols for the combined set of PHICHs. The output matrix, `sym`, always contains `info.NRE` symbols to map into the total PHICH resource allocation, even if less than the full set of `NPHICH` channels are configured. For more information, see “PHICH Generation” on page 2-640.

Data Types: `double`
Complex Number Support: Yes

### `info` — PHICH subframe resourcing information about output symbols

scalar structure

PHICH subframe resourcing information about the output symbols, returned as a scalar structure. `info` contains the following fields. For background on the `info` structure, see `ltePHICHInfo`.

#### `NREG` — Number of resource element groups assigned to all PHICH

positive integer

Number of resource element groups assigned to all PHICH, returned as a positive integer.

Data Types: `uint64`

#### `NRE` — Number of resource elements assigned to all PHICH

positive integer

Number of resource elements assigned to all PHICH, returned as a positive integer.

Data Types: `uint64`

#### `NPHICH` — Number of individual PHICH available

positive integer

Number of individual PHICH available, returned as a positive integer.

Data Types: `uint64`

#### `NGroups` — Number of PHICH groups

positive integer

Number of PHICH groups, returned as a positive integer.

Data Types: `int8`

**`NMappingUnits` — Number of PHICH mapping units**
positive integer

Number of PHICH mapping units, returned as a positive integer.

Data Types: `int8`

**`NSequences` — Number of orthogonal sequences in each PHICH group**
positive integer

Number of orthogonal sequences in each PHICH group, returned as a positive integer.

Data Types: `int8`

**`PHICHDuration` — PHICH duration**
integer

PHICH duration, returned as an integer.

Data Types: `int8`

**`ActiveHISet` — Active HARQ indicator set**
matrix | [ ]

Active HARQ indicator set, returned as an *N*-by-3 matrix. *N* is the number of rows with a valid `hiset` defined. `ActiveHISet` contains only rows that define a valid HARQ indicator set, even if the input `hiset` has rows in which `NGroup` ≥ `info.NGroups`. If `hiset` is empty, `ActiveHISet` is returned as an empty matrix, [ ].

Data Types: `int8`

Data Types: `struct`

## More About

### PHICH Generation

The physical hybrid ARQ indicator channel (PHICH) processing includes the stages of BPSK modulation, orthogonal sequence spreading, scrambling, resource element group (REG) alignment, layer mapping, precoding, PHICH group summation, and mapping unit creation. For details, see TS 36.211 [1], Section 6.9.

The control region of a subframe can contain up to `info.NPHICH` separate PHICH, each carrying a single hybrid ARQ (HARQ) acknowledgment (ACK), or negative acknowledgment (NACK). Settings in the `enb` structure define *NPHICH*.

The input `hiset` specifies an *R*-by-3 matrix, configuring individual PHICH that are combined in the output. *R* is the number of rows. Each row of `hiset` defines a single PHICH in terms of `[nGroup,nSeq,hi]`. In the `info` structure fields, the following conditions apply for the sets of hybrid indicators defined:

- `nGroup` < `info.NGroups`
- `nSeq` < `info.NSequences`

To generate PHICH symbol, `ltePHICH` uses the rows of `hiset` in which `nGroup < info.NGroups`. These rows are present in `info.ActiveHISet`. Any other rows present in `hiset` are ignored.

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePHICHDecode` | `ltePHICHPrecode` | `ltePHICHIndices` | `ltePHICHInfo` | `ltePHICHPRBS`

# ltePHICHDecode

Physical hybrid ARQ indicator channel decoding

## Syntax

```
[hi,symbols] = ltePHICHDecode(enb,hires,sym)
[hi,symbols] = ltePHICHDecode(enb,hires,sym,hest,noiseest)
[hi,symbols] = ltePHICHDecode(enb,hires,sym,hest,noiseest,alg)
```

## Description

`[hi,symbols] = ltePHICHDecode(enb,hires,sym)` performs the inverse of Physical Hybrid ARQ Indicator Channel (PHICH) processing given the cell-wide settings structure, `enb`, PHICH resources, `hires`, and the matrix of PHICH symbols, `sym`. It returns a column vector of hybrid ARQ indicator values, `hi`, and the received symbol constellation matrix, `symbols`. The channel inverse processing includes deprecoding, descrambling, despreading and symbol demodulation. For details, see TS 36.211 [1], Section 6.9 and `ltePHICH`.

`[hi,symbols] = ltePHICHDecode(enb,hires,sym,hest,noiseest)`, where `hest` contains the channel estimate, and `noiseest` contains the noise estimate.

- For the `TxDiversity` transmission scheme (`CellRefP`=2 or 4), the reception is performed using an Orthogonal Space Frequency Block Code (OSFBC) decoder.
- For the `'Port0'` transmission scheme (`CellRefP`=1), the reception is performed using MMSE equalization.

`[hi,symbols] = ltePHICHDecode(enb,hires,sym,hest,noiseest,alg)`, where specifying `alg` provides control over weighting the soft values that are used to determine `hi` with the channel state information (CSI) calculated during the equalization process.

## Examples

### Decode HARQ

Decode the HARQ bits given PHICH symbols. Two HARQ bits are encoded into PHICH symbols using an `hiset` matrix, where each row is used to define an individual PHICH. The generated PHICH symbols are then decoded with the same parameters used to define the PHICHs in the encoder.

Using `lteRMCDL` to create a cell-wide settings configuration structure with RMC R.1, initializes relevant parameters for this example.

```
rc = 'R.1';
enb = lteRMCDL(rc);
```

Define the `hiset` input and generate PHICH symbols. First PHICH: [nGroup=1 nSeq=1 hi=1], second PHICH: [nGroup=1 nSeq=2 hi=0].

```
hiset = [ 1 1 1; 1 2 0 ];
phichSym = ltePHICH(enb,hiset);
```

Define the `hires` input. PHICH resources: same as the encoder, first PHICH: [nGroup=1 nSeq=1], second PHICH: [nGroup=1 nSeq=2].

```
hires = [ 1 1; 1 2];
```

Decode the PHICH.

```
hi = ltePHICHDecode(enb,hires,phichSym);
```

Check decoded hi bits are the same as encoded bits.

```
isequal(hi,hiset(:,3))
```

```
ans = logical
   1
```

## Input Arguments

### enb — Cell-wide settings

scalar structure

Cell-wide settings, specified as a scalar structure. `enb` is a structure having the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NCellID`** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **`CellRefP`** | Required | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **`NSubframe`** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **`CyclicPrefix`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |

Data Types: `struct`

### hires — PHICH resources

numeric matrix

PHICH resources, specified as an *R*-by-2 numeric matrix. This matrix configures up to *NPHICH* individual PHICH for decoding by `ltePHICHDecode`. Each row of `hires` defines a single PHICH in terms of `[nGroup,nSeq]`, where `nGroup` is the PHICH group index number and `nSeq` is the sequence index number. These indices are zero-based.

Each row of output, `hi`, represents the received Hybrid ARQ indicator values for the PHICH defined in the corresponding row of `hires`. Similarly, each row of the output, `symbols`, contains the three received symbols (after despreading) for the corresponding row of `hires`.

In terms of `ltePHICHInfo` `info` structure fields, the following conditions apply:
`nGroup` < `info.NGroups`
`nSeq` < `info.NSequences`

Rows of `hires` with `nGroup` ≥ `info.NGroups` are ignored and output for these rows in `hi` and `symbols` is set to zero.

Data Types: `double`
Complex Number Support: Yes

### `sym` — Complex modulated PHICH symbols

numeric matrix

Complex modulated PHICH symbols, specified as an `NRE`-by-`NRxAnts` numeric matrix of PHICH symbols. `NRE` is the number of BPSK symbols per antenna assigned to the PHICH, and `NRxAnts` is the number of receive antennas. Even if less than the full set of *NPHICH* channels were configured, ensure the `sym` input matrix contains `info.NRE` symbols corresponding to the total PHICH resource allocation. Use `ltePHICHInfo` to view the `info` structure fields.

Data Types: `double`
Complex Number Support: Yes

### `hest` — Channel estimate

3-D numeric array

Channel estimate, specified as an `NRE`-by-`NRxAnts`-by-`enb.CellRefP` numeric array, where:

- `NRE` are the frequency and time locations corresponding to the PCFICH RE positions (a total of `NRE` positions).
- `NRxAnts` is the number of receive antennas.
- `enb.CellRefP` is the number of cell-specific reference signal antennas.

Data Types: `double`
Complex Number Support: Yes

### `noiseest` — Noise estimate

numeric scalar

Noise estimate, specified as a numeric scalar. This input argument is an estimate of the noise power spectral density per RE on received subframe. Such an estimate is provided by the `lteDLChannelEstimate` function.

Data Types: `double`

### `alg` — Weighting algorithm

structure

Weighting algorithm, specified as a structure. This input argument controls weighting output soft bits, `bits`, with CSI. `alg` contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`CSI`** | Optional | `'On'` (default), `'Off'` | Flag provides control over weighting the soft values that are used to determine the output values with the channel state information (CSI) calculated during the equalization process. If `'On'`, soft values are weighted by CSI. |

Data Types: `struct`

## Output Arguments

### `hi` — Hybrid ARQ indicator values

numeric column vector

Hybrid ARQ indicator values, returned as a numeric column vector. Each row represents the received Hybrid ARQ indicator values for the PHICH defined in the corresponding row of `hires`.

Data Types: `double`

### `symbols` — Received symbols

numeric matrix

Received symbols, returned as a numeric matrix. Each row contains the received constellation symbols (after despreading) for the corresponding row of `hires`.

Data Types: `double`
Complex Number Support: Yes

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePHICH` | `ltePHICHDeprecode` | `ltePHICHIndices` | `ltePHICHInfo` | `ltePHICHPRBS` | `ltePHICHTransmitDiversityDecode`

# ltePHICHDeprecode

PHICH deprecoding

## Syntax

```
out = ltePHICHDeprecode(in,cp,ngroup)
out = ltePHICHDeprecode(enb,ngroup,in)
```

## Description

`out = ltePHICHDeprecode(in,cp,ngroup)` performs deprecoding of the *N*-by-*P* matrix of antennas, `in`, onto *NU*=*P* layers, given cyclic prefix length, `cp`, and PHICH group, `ngroup`. *N* is the number of symbols per antenna. It performs PHICH deprecoding using matrix pseudoinversion to undo the processing described in TS 36.211, Section 6.9.2 [1]. This function returns `out`, an *M*-by-*NU* matrix, where *NU* is the number of transmission layers and *M* is the number of symbols per layer.

`out = ltePHICHDeprecode(enb,ngroup,in)` performs deprecoding of the *N*-by-*P* matrix of antennas, `in`, onto *NU*=*P* layers, for PHICH group, `ngroup`, using the cell-wide settings structure, `enb`.

## Examples

### Deprecode PHICH symbols2

This example shows the deprecoding of a set of physical HARQ indicator channel (PHICH) symbols for reference measurement channel (RMC) R.11.

Initialize a cell-wide parameter configuration structure, `enb`, for RMC R.11.

```
rc = 'R.11';
enb = lteRMCDL(rc);
nLayers = enb.PDSCH.NLayers;
```

Generate an arbitrary set of input symbols as the PHICH symbols.

```
phichSym = reshape(lteSymbolModulate(randi([0,1],40*nLayers*2,1), ...
    'QPSK'),40,nLayers);
```

Precode the PHICH symbols using normal cyclic prefix (setting for `enb.CyclicPrefix` as per R.11) and PHICH group 1.

```
nGroup = 1;
precodedSym = ltePHICHPrecode(phichSym,enb.CyclicPrefix,nGroup);
```

Deprecode the precoded PHICH symbols using normal cyclic prefix and PHICH group 1.

```
out = ltePHICHDeprecode(precodedSym,enb.CyclicPrefix,nGroup);
```

Check PHICH symbols vs. deprecoded PHICH symbols.

```
isequal(phichSym,out)
```

```
ans = logical
   1
```

## Input Arguments

### `in` — Precoded input symbols

complex-valued numeric matrix

Precoded input symbols, specified as a complex-valued numeric matrix of antennas. It has size *N*-by-*P*, where *N* is the number of symbols per antenna and *P* is the number of antennas. The number of input symbols, *N*, must be a multiple of the number of antennas, *P*.

Data Types: `double`
Complex Number Support: Yes

### `cp` — Cyclic prefix length

`'Normal'` | `'Extended'`

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

### `ngroup` — PHICH group number

positive scalar integer (≥1)

PHICH group number, specified as a positive scalar integer of 1 or more.

Data Types: `double`

### `enb` — Cell-wide settings

scalar structure

Cell-wide settings, specified as a scalar structure. `enb` contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`CyclicPrefix`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |

Data Types: `struct`

## Output Arguments

### `out` — Deprecoded output symbols

numeric matrix

Deprecoded output symbols, returned as a numeric matrix. It has size *M*-by-*NU*, where *M* is the number of symbols per layer and *NU* is the number of transmission layers.

Data Types: `double`
Complex Number Support: Yes

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePHICHPrecode` | `ltePHICHIndices` | `ltePHICHInfo` | `ltePHICHPRBS` | `ltePHICHTransmitDiversityDecode` | `lteLayerDemap` | `ltePHICHDecode`

# ltePHICHIndices

PHICH resource element indices

## Syntax

```
ind = ltePHICHIndices(enb)
ind = ltePHICHIndices(enb,opts)
```

## Description

`ind = ltePHICHIndices(enb)` returns the subframe resource element (RE) indices, `ind`, for the Physical Hybrid-ARQ Indicator Channels (PHICH), given the parameter fields of cell-wide settings structure, `enb`. By default, the number of rows of `ind` is the number of resource elements ($N_{RE}$), and `ind` is an $N_{RE}$-by-`CellRefP` matrix of indices in a one-based linear indexing style. These indices can directly index elements of an $N$-by-$M$-by-`CellRefP` array that represents the subframe resource grid across `CellRefP` antenna ports. Each column of `ind` identifies the same set of $N_{RE}$ resource elements, but with indices offset to select them in a different antenna “page” of the 3-D resource array.

The indices returned are for all PHICH groups in a subframe, where the number of groups depends on the bandwidth and PHICH `Ng` parameter. See `ltePHICHInfo` for details. The indices are ordered as the modulation symbols should be mapped for the set of consecutive PHICH groups. The PHICH resources are normally all assigned in the first OFDM symbol of a subframe, unless the PHICH duration is of the extended type.

`ind = ltePHICHIndices(enb,opts)` formats the returned indices using options specified by `opts`.

## Examples

### Generate PHICH Indices

Generate PHICH resource element (RE) indices in linear form and resource element group (REG) indices in subscript form.

Get one-based PHICH resource element (RE) indices in linear form.

```
enb = lteRMCDL('R.14');
enb.NDLRB = 6;
indOneBased = ltePHICHIndices(enb,{'ind','re'})
```

```
indOneBased = 12x4 uint32 matrix

      8   1016   2024   3032
      9   1017   2025   3033
     11   1019   2027   3035
     12   1020   2028   3036
     26   1034   2042   3050
     27   1035   2043   3051
     29   1037   2045   3053
```

```
    30   1038   2046   3054
    50   1058   2066   3074
    51   1059   2067   3075
     ⋮
```

Get zero-based PHICH resource element group (REG) indices in subscript form, where each column of `ind` corresponds to a dimension of the 3-D resource grid array: subcarrier, OFDM symbol, and antenna port.

```
indZeroBased = ltePHICHIndices(enb,{'0based','sub','reg'})
```

```
indZeroBased = 12x3 uint32 matrix

    6    0    0
   24    0    0
   48    0    0
    6    0    1
   24    0    1
   48    0    1
    6    0    2
   24    0    2
   48    0    2
    6    0    3
      ⋮
```

## Input Arguments

### `enb` — Cell-wide settings

scalar structure

Cell-wide settings, specified as a scalar structure. `enb` can contain the following fields. The `TDDConfig` and `NSubframe` parameter fields are only required if `DuplexMode` is set to `'TDD'`.

### `NDLRB` — Number of downlink resource blocks

positive integer

Number of downlink resource blocks, specified as a positive integer.

Data Types: `double`

### `NCellID` — Physical layer cell identity

nonnegative integer

Physical layer cell identity, specified as a nonnegative integer.

Data Types: `double`

### `CyclicPrefix` — Cyclic prefix length

`'Normal'` (default) | optional | `'Extended'`

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

**`CellRefP` — Number of cell-specific reference signal antenna ports**
1 | 2 | 4

Number of cell-specific reference signal antenna ports, specified as 1, 2, or 4.

Data Types: `double`

**`Ng` — HICH group multiplier**
`'Sixth'` | `'Half'` | `'One'` | `'Two'`

HICH group multiplier, specified as `'Sixth'`, `'Half'`, `'One'`, or `'Two'`.

Data Types: `char` | `string`

**`PHICHDuration` — PHICH duration**
`'Normal'` (default) | optional | `'Extended'`

PHICH duration, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

**`DuplexMode` — Duplex mode**
`'FDD'` (default) | optional | `'TDD'`

Duplex mode, specified as `'FDD'` or `'TDD'`.

Data Types: `char` | `string`

**`TDDConfig` — Uplink or downlink configuration**
0 (default) | optional | 1 | 2 | 3 | 4 | 5 | 6

Uplink or downlink configuration, specified as a nonnegative scalar integer from 0 through 6. Only required if `DuplexMode` is set to `'TDD'`.

Data Types: `double`

**`NSubframe` — Subframe number**
nonnegative integer

Subframe number, specified as a nonnegative integer. Only required if `DuplexMode` is set to `'TDD'`.

Data Types: `double`

Data Types: `struct`

**`opts` — Output format, base, and unit of generated indices**
character vector | string scalar | cell array of character vectors | string array

Output format, base, and unit of generated indices, specified as one of these forms.

- `'format base unit'`
- `"format base unit"`
- `{'format','base','unit'}`
- `["format","base","unit"]`

Where `format`, `base`, and `unit` are defined in this table.

| Option | Values | Description |
|---|---|---|
| **format** | `'ind'` (default), `'sub'` | Output format of generated indices<br><br>To return the indices as a column vector, specify this option as `'ind'`.<br><br>To return the indices as an $N_{RE}$-by-3 matrix, where $N_{RE}$ is the number of REs, specify this option as `'sub'`. Each row of the matrix contains the subcarrier, symbol, and antenna port as its first, second, and third element, respectively. |
| **base** | `'1based'` (default), `'0based'` | Index base<br><br>To generate indices whose first value is `1`, specify this option as `'1based'`. To generate indices whose first value is `0`, specify this option as `'0based'`. |
| **unit** | `'re'` (default), `'reg'` | Unit of returned indices<br><br>To indicate that the returned values correspond to individual resource elements (REs), specify this option as `'re'`. To indicate that the returned values correspond to resource element groups (REGs), specify this option as `'reg'`. |

Example: `'ind 0based reg'`, `"ind 0based reg"`, `{'ind','0based','reg'}`, and `["ind","0based","reg"]` specify the same output options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `ind` — PHICH resource element indices

numeric matrix

PHICH resource element indices, returned as a numeric matrix. The size of the matrix is $N_{RE}$-by-`CellRefP`. By default, it contains one-based linear indexing RE indices.

Data Types: `uint32`

# Version History

**Introduced in R2014a**

## See Also

`ltePHICH` | `ltePHICHDecode` | `ltePHICHPrecode` | `ltePHICHDeprecode` | `ltePHICHInfo` | `ltePHICHPRBS`

# ltePHICHInfo

PHICH resource information

## Syntax

```
info = ltePHICHInfo(enb)
```

## Description

`info = ltePHICHInfo(enb)` returns a structure, `info`, containing information about the physical hybrid ARQ indicator channel (PHICH) subframe resources.

## Examples

### Get PHICH Resource Information

Get PHIH resource information for normal cyclic prefix system subframe with 50 DL resource blocks and HICH group multiplier set to 'Sixth'. See that there are 16 PHICH available, split between two PHICH groups of 8 sequences.

Initialize the cell-wide configuration structure, `enb`

```
enb.NDLRB = 50;
enb.Ng = 'Sixth';
```

Display the PHICH information

```
info = ltePHICHInfo(enb)
```

```
info = struct with fields:
             NREG: 6
              NRE: 24
           NPHICH: 16
          NGroups: 2
    NMappingUnits: 2
       NSequences: 8
    PHICHDuration: 1
```

### Get PHICH Resource Information from RMC

This example shows that for RMC R.14, there are 16 PHICH available, split between two PHICH groups of 8 sequences.

Initialize the cell-wide configuration structure, `enb`, using RMC R.14

```
rc = 'R.14';
enb = lteRMCDL(rc);
```

Display the PHICH information

```
info = ltePHICHInfo(enb)
```

```
info = struct with fields:
             NREG: 6
              NRE: 24
            NPHICH: 16
           NGroups: 2
     NMappingUnits: 2
        NSequences: 8
     PHICHDuration: 1
```

## Input Arguments

### enb — eNodeB cell-wide settings

scalar structure

eNodeB cell-wide settings, specified as a structure containing these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NDLRB** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **CyclicPrefix** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **Ng** | Required | `'Sixth'`, `'Half'`, `'One'`, `'Two'` | HICH group multiplier |
| **PHICHDuration** | Optional | Nonnegative scalar integer | PHICH duration |
| **DuplexMode** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| The following parameters are dependent upon the condition that `DuplexMode` is set to `'TDD'`. | | | |
| **TDDConfig** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **NSubframe** | Required | 0 (default), nonnegative scalar integer | Subframe number |

## Output Arguments

### info — PHICH subframe resource information

scalar structure

PHICH subframe resource information, returned as a scalar structure. `info` contains the following fields.

| Parameter Field | Description | Values | Data Type |
|---|---|---|---|
| **NRE** | Number of resource elements (REs) assigned to all PHICH | Nonnegative scalar integer | `uint64` |
| **NREG** | Number of resource element groups assigned to all PHICH | Nonnegative scalar integer | `uint64` |
| **NPHICH** | Number of individual PHICH available | Nonnegative scalar integer | `uint64` |
| **NGroups** | Number of PHICH groups | Nonnegative scalar integer | `int8` |
| **NMappingUnits** | Number of PHICH mapping units | Nonnegative scalar integer | `int8` |
| **NSequences** | Number of orthogonal sequences in each PHICH group | Nonnegative scalar integer | `int8` |
| **PHICHDuration** | PHICH duration | Nonnegative scalar integer | `int8` |

The control region of a subframe can contain up to `NPHICH` separate PHICHs with each carrying a single hybrid ARQ ACK or NACK. Multiple PHICHs can be mapped to the same set of resource elements through PHICH groups. Each PHICH in a group is carried on one of `NSequences` orthogonal sequences. For mapping to resources, the groups are combined into mapping units where each unit spans three resource element groups. Thus, `NREG` is `3×NMappingUnits` and `NRE` is `4×3×NMappingUnits`. The `Ng` parameter controls the number of groups available for a given bandwidth.

## Version History

**Introduced in R2014a**

## See Also

`ltePHICH` | `ltePHICHDecode` | `ltePHICHPrecode` | `ltePHICHDeprecode` | `ltePHICHIndices` | `ltePHICHPRBS` | `ltePHICHTransmitDiversityDecode`

# ltePHICHPRBS

PHICH pseudorandom scrambling sequence

## Syntax

```
[seq,cinit] = ltePHICHPRBS(enb,n)
[seq,cinit] = ltePHICHPRBS(enb,n,mapping)

[subseq,cinit] = ltePHICHPRBS(enb,pn)
[subseq,cinit] = ltePHICHPRBS(enb,pn,mapping)
```

## Description

`[seq,cinit] = ltePHICHPRBS(enb,n)` returns the first `n` outputs of the Physical Hybrid ARQ Indicator Channel (PHICH) scrambling sequence when initialized according to cell-wide settings structure, `enb`. It also returns an initialization value `cinit` for the pseudorandom binary sequence (PRBS) generator.

`[seq,cinit] = ltePHICHPRBS(enb,n,mapping)` allows control over the format of the returned sequence, `seq`, with the input `mapping`.

`[subseq,cinit] = ltePHICHPRBS(enb,pn)` returns a subsequence of a full PRBS sequence, specified by `pn`.

`[subseq,cinit] = ltePHICHPRBS(enb,pn,mapping)` allows additional control over the format of the returned subsequence, `subseq`, with the input `mapping`.

## Examples

### Generate PHICH Pseudorandom Scrambling Sequence

Create a cell-wide configuration structure initialing for RMC R.0. Generate the pseudorandom scrambling sequence for the PHICH.

```
enb = lteRMCDL('R.0');
phichInfo = ltePHICHInfo(enb);
phichPrbsSeq = ltePHICHPRBS(enb,phichInfo.NRE);
numRE = phichInfo.NRE
```

```
numRE = uint64
    12
```

```
size(phichPrbsSeq)
```

```
ans = 1×2

    12     1
```

Using RMC R.0 results in 12 BPSK modulated symbols, where one bit per symbol is mapped onto a single resource element (RE).

## Input Arguments

### `enb` — Cell-wide settings

scalar structure

Cell-wide settings, specified as a scalar structure. `enb` contains the following fields.

### `NCellID` — Physical layer cell identity

nonnegative integer

Physical layer cell identity, specified as a nonnegative integer.

Data Types: `double`

### `NSubframe` — Subframe number

nonnegative integer

Subframe number, specified as a nonnegative integer.

Data Types: `double`

Data Types: `struct`

### `n` — Length of PHICH scrambling sequence

positive integer

Length of PHICH scrambling sequence, specified as a positive integer.

Data Types: `double`

### `pn` — Range of PHICH scrambling subsequence

row vector

Range of PHICH scrambling subsequence, `subseq`, specified as a row vector of `[p n]`. The subsequence returns `n` values of the PRBS generator, starting at position `p` (0-based).

Data Types: `double`

### `mapping` — Output sequence formatting

`'binary'` (default) | `'signed'`

Output sequence formatting, specified as `'binary'` or `'signed'`. `mapping` controls the format of the returned sequence.

- `'binary'` maps `true` to 1 and `false` to 0.
- `'signed'` maps `true` to –1 and `false` to 1.

Data Types: `char` | `string`

## Output Arguments

### `seq` — PHICH pseudorandom scrambling sequence

logical column vector | numeric column vector

PHICH pseudorandom scrambling sequence, returned as a logical column vector or a numeric column vector. This argument contains the first `n` outputs of the PHICH scrambling sequence. If

`mapping` is set to `'signed'`, `seq` is a vector of data type double. Otherwise, it is a vector of data type logical.

Data Types: `logical` | `double`

**`subseq` — PHICH pseudorandom scrambling subsequence**
logical column vector | numeric column vector

PHICH pseudorandom scrambling subsequence, returned as a logical column vector or a numeric column vector. `subseq` contains the values of the PRBS generator specified by `pn`. If you set `mapping` to `'signed'`, the output data type is `double`. Otherwise, the output data type is `logical`.

Data Types: `logical` | `double`

**`cinit` — Initialization value for PRBS generator**
numeric scalar

Initialization value for PRBS generator, returned as a numeric scalar.

Data Types: `uint32`

# Version History

**Introduced in R2014a**

## See Also

`ltePHICH` | `ltePHICHDecode` | `ltePHICHPrecode` | `ltePHICHDeprecode` | `ltePHICHIndices` | `ltePHICHInfo` | `ltePHICHTransmitDiversityDecode`

# ltePHICHPrecode

PHICH precoding

## Syntax

```
out = ltePHICHPrecode(in,cp,ngroup)
out = ltePHICHPrecode(enb,ngroup,in)
```

## Description

`out = ltePHICHPrecode(in,cp,ngroup)` precodes the *N*-by-*NU* matrix of layers, `in`, onto *P*=*NU* antennas, given cyclic prefix length, `cp`, and PHICH group, `ngroup`. It performs PHICH precoding according to TS 36.211, Section 6.9.2 [1]. This function returns an *M*-by-*P* matrix, where *P* is the number of transmission antennas and *M* is the number of symbols per antenna.

`out = ltePHICHPrecode(enb,ngroup,in)` precodes the *N*-by-*NU* matrix of layers, `in`, onto *P*=*NU* antennas for PHICH group, `ngroup`, using the cell-wide settings structure, `enb`.

## Examples

### Precode PHICH symbols

This example shows precoding of an arbitrary set of PHICH symbols for reference measurement channel (RMC) R.11, PHICH group 1.

Initialize a cell-wide parameter configuration structure, `enb`, for RMC R.11.

```
rc = 'R.11';
enb = lteRMCDL(rc);
nLayers = enb.PDSCH.NLayers;
```

Generate an arbitrary set of input symbols as the PHICH symbols.

```
phichSym = reshape(lteSymbolModulate(randi([0,1],40*nLayers*2,1), ...
    'QPSK'),40,nLayers);
```

Precode the PHICH symbols using normal cyclic prefix (set in `enb.CyclicPrefix` as per R.11), and PHICH group 1.

```
nGroup = 1;
precodedSym = ltePHICHPrecode(phichSym ,enb.CyclicPrefix, nGroup);
```

Have a peek at the first 5 precoded symbols of the output, columns represent the number of transmit antennas, for this example there are two transmit antennas.

```
precodedSym(1:5, :)
```

```
ans = 5×2 complex

  -0.5000 - 0.5000i  -0.5000 - 0.5000i
   0.5000 - 0.5000i  -0.5000 + 0.5000i
```

```
   0.5000 - 0.5000i   0.5000 - 0.5000i
  -0.5000 - 0.5000i   0.5000 + 0.5000i
  -0.5000 + 0.5000i   0.5000 + 0.5000i
```

## Input Arguments

### `in` — PHICH input symbols

complex-valued numeric matrix

PHICH input symbols, specified as a complex-valued numeric matrix. `in` is a matrix of *N*-by-*NU* layers.

Data Types: `double`
Complex Number Support: Yes

### `cp` — Cyclic prefix length

`'Normal'` | `'Extended'`

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

### `ngroup` — PHICH group

positive scalar integer

PHICH group number, specified as a positive scalar integer of 1 or more.

Data Types: `double`

### `enb` — Cell-wide settings

scalar structure

Cell-wide settings, specified as a scalar structure. `enb` can contain the following field.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`CyclicPrefix`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |

Data Types: `struct`

## Output Arguments

### `out` — Precoded output

numeric matrix

Precoded output, returned as a numeric matrix of size *M*-by-*P*. *M* is the number of symbols per antenna and *P* is the number of transmission antennas.

Data Types: `double`
Complex Number Support: Yes

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePHICHDeprecode` | `ltePHICHIndices` | `ltePHICHInfo` | `ltePHICHPRBS` | `lteLayerMap` | `ltePHICH`

# ltePHICHTransmitDiversityDecode

PHICH OSFBC decoding

## Syntax

```
[out,CSI]=ltePHICHTransmitDiversityDecode(in,cp,ngroup,hest)
```

## Description

`[out,CSI]=ltePHICHTransmitDiversityDecode(in,cp,ngroup,hest)` returns Orthogonal Space Frequency Block Code (OSFBC) decoded symbols, `out`, and channel state information, `CSI`, given received PHICH symbols, `in`, along with cyclic prefix length, `cp`, PHICH resource group number, `ngroup`, and, channel estimate, `hest`.

## Examples

### Deprecode PHICH Symbols

Generate the PHICH symbols for multiple antennas using RMC R.11.

Initialize cell-wide settings for RMC R.11.

```
enb = lteRMCDL('R.11');
phichInfo = ltePHICHInfo(enb);
hiset = [1,1,1;1,2,0];
phichSym = ltePHICH(enb,hiset);
```

Create an ideal, or identity, channel estimate.

```
hest = permute(repmat(eye(enb.CellRefP),[1,1,phichInfo.NRE]),[3,1,2]);
```

Deprecode the received symbols, using the channel estimates.

```
ng = phichInfo.NGroups;
out = ltePHICHTransmitDiversityDecode(phichSym,enb.CyclicPrefix,ng,hest)
```

```
out = 24×1 complex

   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
      ⋮
```

## Input Arguments

### `in` — Received PHICH symbols

numeric matrix

Received PHICH symbols, specified as a numeric matrix of size *M*-by-`NRxAnts`, where *M* is the number of received symbols for each of `NRxAnts` receive antennas.

Data Types: `double`
Complex Number Support: Yes

### `cp` — Cyclic prefix length

`'Normal'` | `'Extended'`

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

### `ngroup` — PHICH group number

positive scalar integer (≥1)

PHICH group number, specified as a positive scalar integer of 1 or more.

Data Types: `double`
Complex Number Support: Yes

### `hest` — Channel estimate

3-D numeric array

Channel estimate, specified as an *M*-by-`NRxAnts`-by-`NTxAnts` numeric array. *M* is the number of received symbols in `in`, `NRxAnts` is the number of receive antennas, and `NTxAnts` is the number of transmit antennas.

Data Types: `double`
Complex Number Support: Yes

## Output Arguments

### `out` — OSFBC decoded symbols

numeric matrix

OSFBC decoded symbols, returned as a numeric matrix of size *M*-by-1, where *M* is the number of received symbols for each receive antenna.

Data Types: `double`
Complex Number Support: Yes

### `CSI` — Soft channel state information

numeric matrix

Soft channel state information, returned as a numeric matrix of size *M*-by-1. It provides an estimate of the received RE gain for each received RE.

Data Types: `double`
Complex Number Support: Yes

## Version History

**Introduced in R2014a**

## See Also

`ltePHICH` | `ltePHICHIndices` | `ltePHICHInfo` | `ltePHICHPRBS` | `ltePHICHDecode` | `ltePHICHDeprecode`

# ltePMIInfo

Precoder matrix indication reporting information

## Syntax

```
info = ltePMIInfo(enb,chs)
```

## Description

`info = ltePMIInfo(enb,chs)` returns a precoder matrix indication (PMI) reporting information structure, given structures containing cell-wide settings, and the channel transmission configuration settings. For more information, see TS 35.213 [1].

## Examples

### PMI Reporting Information

Get the PMI reporting information for RMC R.13.

```
enb = lteRMCDL('R.13');
pmiInfo = ltePMIInfo(enb, enb.PDSCH)
```

```
pmiInfo = struct with fields:
                      k: 50
              NSubbands: 1
                 MaxPMI: 15
    CodebookSubsetSize: 16
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a scalar structure. The structure contains the following fields:

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NDLRB`** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **`CellRefP`** | Optional | 1 (default), 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| The following parameter applies when `chs.TxScheme` is set to `Port7-14`. | | | |
| **`CSIRefP`** | Optional | 1 (default), 2, 4, 8 | Array of number of CSI-RS antenna ports |

Data Types: `struct`

### `chs` — Channel transmission configuration

structure

Channel transmission configuration, specified as a structure containing the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`PMIMode`** | Optional | `'Wideband'` (default), `'Subband'` | PMI reporting mode. `PMIMode='Wideband'` corresponds to PUSCH reporting Mode 1-2 or PUCCH reporting Mode 1-1 (PUCCH Report Type 2) and `PMIMode='Subband'` corresponds to PUSCH reporting Mode 3-1. |
| **`NLayers`** | Optional | Integer from 1 to 8<br><br>Default number of layers is 1. | Number of transmission layers. |
| **`TxScheme`** | Optional | `'Port0'`, `'TxDiversity'`, `'CDD'`, `'SpatialMux'` (default), `'MultiUser'`, `'Port7-8'`, `'P`<br>`'Port7-14'`. | PDSCH transmission scheme, specified as one of the following options.<br><br>**Transmission scheme** / **Description**<br>`'Port0'` — Single antenna port, port 0<br>`'TxDiversity'` — Transmit diversity<br>`'CDD'` — Large delay cyclic delay diversity scheme<br>`'SpatialMux'` — Closed loop spatial multiplexing<br>`'MultiUser'` — Multi-user MIMO<br>`'Port5'` — Single-antenna port, port 5<br>`'Port7-8'` — Single-antenna port, port 7, when `NLayers` = 1. Dual layer transmission, ports 7 and 8, when `NLayers` = 2.<br>`'Port8'` — Single-antenna port, port 8<br>`'Port7-14'` — Up to eight layer transmission, ports 7–14 |
| The following parameter applies when `'Port7-14'` transmission scheme with `CSIRefP` equal to 4, or for `'Port7-8'` or `'Port8'` transmission scheme with `CellRefP` equal to 4. | | | |
| **`AltCodebook4Tx`** | Optional | `'Off'` (default), `'On'` | If set to `'On'`, enables the alternative codebook for CSI reporting with four antennas defined in TS 36.213, Tables 7.2.4-0A to 7.2.4-0D. The default is `'Off'`. (*alternativeCodeBookEnabledFor4TX-r12*) |

Data Types: `struct`

## Output Arguments

### `info` — Information related to PMI reporting

structure

Information related to PMI reporting, returned as a structure containing these fields:

| Parameter Field | Description | Values |
|---|---|---|
| **k** | Subband size, in resource blocks (equal to `NRB` for wideband PMI reporting or transmission schemes without PMI reporting). | numeric scalar |
| **NSubbands** | Number of subbands for PMI reporting (equal to 1 for wideband PMI reporting) or transmission schemes without PMI reporting. | numeric scalar |
| **MaxPMI** | Maximum permitted PMI value for the given configuration. Valid PMI values range from 0 to `MaxPMI`. For CSI reporting, when `CSIRefP = 8`, or for CSI reporting with the alternative codebook for four antennas,`MaxPMI` is a 2–element vector, indicating the maximum permissible values of *i1* and *i2*, the first and second codebook indices. For transmission schemes without PMI reporting, `MaxPMI = 0`. | nonnegative numeric scalar |
| **CodeBookSubsetSize** | Size of the codebook subset restriction bitmap. For transmission schemes without PMI reporting, `CodebookSubsetSize=0`. | scalar |

`info.NSubbands` can be used to determine the correct size of the vector `PMISet` required for closed-loop spatial multiplexing operation; `PMISet` should be a column vector with `info.NSubbands` rows. For CSI reporting with `CSIRefP = 8`, or for CSI reporting with the alternative codebook for four antennas, `info.NSubbands` indicates the number of second codebook indices, *i2*, in the report. The first codebook index, *i1*, is always chosen in a wideband fashion, and is therefore a scalar. The PMI reporting mode is set with `chs.PMIMode`.

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.213. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePMISelect` | `ltePDSCH` | `ltePDSCHDecode` | `lteDLPrecode` | `lteCSICodebook`

# ltePMISelect

PDSCH precoder matrix indicator calculation

## Syntax

```
[pmiset,info,sinrs,subbandsinrs] = ltePMISelect(enb,chs,hest,noiseest)
```

## Description

`[pmiset,info,sinrs,subbandsinrs] = ltePMISelect(enb,chs,hest,noiseest)` performs PDSCH precoder matrix indication (PMI) set calculation for the given cell-wide settings, `enb`, channel configuration structure, `chs`, channel estimate resource array, `hest`, and receiver noise variance, `noiseest`. For more information, see “PMI Selection” on page 2-675.

## Examples

### PMI Selection

This example shows PMI selection and configuration of a downlink transmission with the selected PMI set.

Populate an empty resource grid for RMC R.13 with cell specific reference signal symbols. OFDM modulate the grid to create `txWaveform`. Initialize channel configuration structure. Pass `txWaveform` through channel and demodulate `rxWaveform` to recover `rxSubframe`

```
enb = lteRMCDL('R.13');
enb.PDSCH.PMIMode = 'Subband';
reGrid = lteResourceGrid(enb);
reGrid(lteCellRSIndices(enb)) = lteCellRS(enb);

[txWaveform,info] = lteOFDMModulate(enb,reGrid);

chcfg.SamplingRate = info.SamplingRate;
chcfg.DelayProfile = 'EPA';
chcfg.NRxAnts = 4;
chcfg.DopplerFreq = 5;
chcfg.MIMOCorrelation = 'Low';
chcfg.InitTime = 0;
chcfg.Seed = 1;

rxWaveform = lteFadingChannel(chcfg,txWaveform);
rxSubframe = lteOFDMDemodulate(enb,rxWaveform);
```

Initialize channel estimation structure. Perform channel estimation. Use estimates of the channel and noise power spectral density for PMI selection. This PMI set is then used to configure a downlink transmission.

```
cec.FreqWindow = 1;
cec.TimeWindow = 31;
cec.InterpType = 'cubic';
```

```
cec.PilotAverage = 'UserDefined';
cec.InterpWinSize = 1;
cec.InterpWindow = 'Centered';

[hest, noiseEst] = lteDLChannelEstimate(enb,cec,rxSubframe);

pmi = ltePMISelect(enb,enb.PDSCH,hest,noiseEst)

pmi = 9×1

     1
     1
     6
     2
    12
    12
    12
    12
    12


enb.PDSCH.PMISet = pmi;
txWaveform = lteRMCDLTool(enb,[1;0;0;1]);
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure containing the following fields:

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NDLRB** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **NCellID** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **CellRefP** | Optional | 1 (default), 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **CyclicPrefix** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **DuplexMode** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| The following parameters apply when `DuplexMode` is set to `'TDD'`. | | | |
| **TDDConfig** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **SSC** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |
| The following parameters apply when `DuplexMode` is set to `'TDD'` or `chs.TxScheme` is set to `'Port7-14'` | | | |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NSubframe** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| The following parameters apply when `chs.TxScheme` is set to `'Port7-14'` transmission scheme. | | | |
| **CSIRefP** | Required | 1, 2, 4 | Array of number of CSI-RS antenna ports |
| **CSIRSConfig** | Required | Scalar integer | Array CSI-RS configuration indices. See TS 36.211, Table 6.10.5.2-1. |
| **CSIRSPeriod** | Optional | `'On'` (default), `'Off'`, `Icsi-rs` (0,...,154), `[Tcsi-rs Dcsi-rs]`. You can also specify values in a cell array of configurations for each resource. | CSI-RS subframe configurations for one or more CSI-RS resources. Multiple CSI-RS resources can be configured from a single common subframe configuration or from a cell array of configurations for each resource. |
| **Nframe** | Optional | 0 (default), nonnegative scalar integer | Frame number |

Data Types: `struct`

**`chs` — Channel transmission configuration**
structure

Channel transmission configuration, specified as a structure containing the following fields:

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NLayers** | Required | Integer from 1 to 8 | Number of transmission layers. |
| **PMIMode** | Optional | `'Wideband'` (default), `'Subband'` | PMI reporting mode. `PMIMode='Wideband'` corresponds to PUSCH reporting Mode 1-2 or PUCCH reporting Mode 1-1 (PUCCH Report Type 2) and `PMIMode='Subband'` corresponds to PUSCH reporting Mode 3-1. |

<table>
<tr><th>Parameter Field</th><th>Required or Optional</th><th>Values</th><th>Description</th></tr>
<tr><td>TxScheme</td><td>Optional</td><td>'Port0', 'TxDiversity', 'CDD', 'SpatialMux' (default), 'MultiUser', 'Port5', 'Port7-8', 'Port8', 'Port7-14'</td><td>PDSCH transmission scheme, specified as one of the following options.
<table>
<tr><th>Transmission scheme</th><th>Description</th></tr>
<tr><td>'Port0'</td><td>Single antenna port, port 0</td></tr>
<tr><td>'TxDiversity'</td><td>Transmit diversity</td></tr>
<tr><td>'CDD'</td><td>Large delay cyclic delay diversity scheme</td></tr>
<tr><td>'SpatialMux'</td><td>Closed loop spatial multiplexing</td></tr>
<tr><td>'MultiUser'</td><td>Multi-user MIMO</td></tr>
<tr><td>'Port5'</td><td>Single-antenna port, port 5</td></tr>
<tr><td>'Port7-8'</td><td>Single-antenna port, port 7, when NLayers = 1. Dual layer transmission, ports 7 and 8, when NLayers = 2.</td></tr>
<tr><td>'Port8'</td><td>Single-antenna port, port 8</td></tr>
<tr><td>'Port7-14'</td><td>Up to eight layer transmission, ports 7–14</td></tr>
</table></td></tr>
<tr><td>CodebookSubset</td><td>Optional</td><td>Character vector, string scalar, or integer vector, all ones (default)</td><td>Codebook subset restriction, specified as a character vector or string scalar bitmap. The default values are all ones, permitting all PMI values. This parameter is configured by higher layers and indicates the values of PMI that can be reported. The bitmap, defined in TS 36.213, Section 7.2, is arranged a_A-1,a_A-2,...a_0. For example, the element CodebookSubset(1) corresponds to a_A-1 and the element CodebookSubset(end) corresponds to a_0. The length of the bitmap is given by the info.CodebookSubsetSize field returned by ltePMIInfo. You can also specify the bitmap in a hexadecimal form by adding the prefix '0x'. Alternatively, you can specify a numeric array identical to the pmiset output, indicating to restrict the selection to only those pmiset values. Specifying the parameter in this way enables you to obtain SINR estimates against an existing reported PMI for RI and CQI selection. If this parameter field is defined but is empty, no codebook subset restriction is applied. (codebookSubsetRestriction)</td></tr>
<tr><td colspan="4">The following parameter applies for 'Port7-14' transmission scheme with CSIRefP equal to 4, or for 'Port7-8' or 'Port8' transmission scheme with CellRefP equal to 4.</td></tr>
</table>

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`AltCodebook4Tx`** | Optional | `'Off'` (default), `'On'` | If set to `'On'`, enables the alternative codebook for CSI reporting with four antennas defined in TS 36.213, Tables 7.2.4-0A to 7.2.4-0D. The default is `'Off'`. (*alternativeCodeBookEnabledFor4TX-r12*) |

Data Types: `struct`

**`hest` — Channel estimate**
multidimensional array

Channel estimate, specified as a multidimensional array of size *K*-by-*L*-by-*NRxAnts*-by-*P* where:

- *K* is the number of subcarriers.
- *L* is the number of OFDM symbols.
- *NRxAnts* is the number of received antennas.
- *P* is the number of planes.

Data Types: `double`

**`noiseest` — Receiver noise variance**
numeric scalar

Receiver noise variance, specified as a numeric scalar. This input argument specifies an estimate of the received noise power spectral density.

Data Types: `double`

## Output Arguments

**`pmiset` — PMI set selected**
column vector | integer

Precoder matrix indications (PMI) set selected, returned as a column vector or an integer.

- For the `'Port7-14'` transmission scheme with eight CSI-RS ports, or for CSI reporting with the alternative codebook for four antennas, `pmiset` has `info.NSubbands + 1` rows. The first row indicates wideband codebook index *i1*. The subsequent `info.NSubbands` rows indicate the subband codebook indices *i2* or if `info.NSubbands = 1`, the wideband codebook index *i2*.
- For other numbers of CSI-RS ports in the `'Port7-14'` transmission scheme, and for other transmission schemes, `pmiset` has `info.NSubbands` rows. Each row gives the subband codebook index for that subband.
- For wideband reporting (`info.NSubbands = 1`), `pmiset` is a scalar specifying the selected wideband codebook index.

**Note** `pmiset` is empty if the noise estimate, `noiseest`, is zero or NaN, or if the channel estimate, `hest`, contains any NaNs in the locations of the reference signal REs used for PMI estimation.

### `info` — Information related to PMI reporting

structure

Information related to PMI reporting, returned as a scalar structure. `info` contains the following fields:

| Parameter Field | Description | Values |
|---|---|---|
| **`k`** | Subband size, in resource blocks (equal to `NRB` for wideband PMI reporting or transmission schemes without PMI reporting). | numeric scalar |
| **`NSubbands`** | Number of subbands for PMI reporting (equal to 1 for wideband PMI reporting) or transmission schemes without PMI reporting. | numeric scalar |
| **`MaxPMI`** | Maximum permitted PMI value for the given configuration. Valid PMI values range from 0 to `MaxPMI`. For CSI reporting, when `CSIRefP = 8`, or for CSI reporting with the alternative codebook for four antennas,`MaxPMI` is a 2–element vector, indicating the maximum permissible values of *i1* and *i2*, the first and second codebook indices. For transmission schemes without PMI reporting, `MaxPMI = 0`. | nonnegative numeric scalar |
| **`CodeBookSubsetSize`** | Size of the codebook subset restriction bitmap. For transmission schemes without PMI reporting, `CodebookSubsetSize=0`. | scalar |

### `sinrs` — Signal-to-interference plus noise ratios

multidimensional array

Signal to interference plus noise ratios, returned as a multidimensional array of size *K*-by-*L*-by-*N1*-by-*N2*, where:

- *K* is the number of subcarriers
- *L* is the number of OFDM symbols
- Definition of *N1* and *N2* depends on the CSI-RS ports:
  - For the `'Port7-14'` transmission scheme with eight CSI-RS ports, or for CSI reporting with the alternative codebook for four antennas, *N1* and *N2* are the number of possible first and second codebook indices:
    - *N1* is `info.MaxPMI(1) + 1`
    - *N2* is `info.MaxPMI(2) + 1`
  - For other numbers of CSI-RS ports in the `'Port7-14'` transmission scheme, and for other transmission schemes:
    - *N1* is 1
    - *N2* is `info.MaxPMI + 1`

The array contains non-`NaN` values in the time and frequency locations (first two dimensions) of the reference signal REs. This array is used for PMI estimation for all possible codebook indices (last two dimensions). These values are the calculated sinrs in the reference signal RE locations for each codebook index combination. You can obtain the values using a linear MMSE SINR metric. All locations not corresponding to a reference signal RE are set to `NaN`.

### `subbandsinrs` — Subband signal-to-interference plus noise ratios

multidimensional array

Subband signal-to-interference plus noise ratios (`sinrs`), returned as an `info.NSubbands`-by-*N1*-by-*N2*-by-`chs.NLayers` array. This array indicates the average linear SINR in the subband specified for each possible PMI value (*N1* and *N2* dimensions) and each layer. The `sinrs` output is formed by summing a 5–dimensional *K*-by-*L*-by-*N1*-by-*N2*-by-`chs.NLayers` estimate of the `sinrs` across all the layers. `subbandsinrs` is formed by averaging that same five-dimensional estimate across each subband that is in the appropriate region of the K dimension and across the L dimension. Dimensionality described in `sinrs` applies here.

## More About

### PMI Selection

PDSCH precoder matrix indication (PMI) selection calculates a PMI set, `pmiset`. Functions, such as `lteRMCDLTool` or `ltePDSCH`, can use the returned `pmiset` to configure the PMI for downlink transmissions they generate. PMI selection is performed using the PMI definitions specified in TS 36.213, Section 7.2.4.

- The CSI reporting codebook is used for:
  - `'Port7-14'` transmission scheme with eight CSI-RS ports
  - CSI reporting with the alternative codebook for four antennas (*alternativeCodeBookEnabledFor4TX -r12* = true).
- The codebook for closed-loop spatial multiplexing, defined in TS 36.211 Tables 6.3.4.2.3-1 and 6.3.4.2.3-2, is used for other cases.

The PMI feedback type associated with the PMI selection process can be wideband or subband:

- `PMIMode = 'Wideband'` corresponds to PUSCH reporting Mode 1-2 or PUCCH reporting Mode 1-1 (PUCCH Report Type 2).
- `PMIMode = 'Subband'` corresponds to PUSCH reporting Mode 3-1.

PMI selection is based on the rank indicated by `chs.NLayers`, except for `'TxDiversity'` transmission scheme, where the rank is 1. In PUCCH reporting Mode 1-1, you can achieve codebook subsampling for submode 2, as specified in TS 36.213, Table 7.2.2-1D, with an appropriate `chs.CodebookSubset`.

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.213. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePDSCH` | `ltePDSCHDecode` | `lteDLPrecode` | `ltePMIInfo` | `lteCSICodebook` | `lteCQISelect` | `lteRISelect`

# ltePRACH

Physical random access channel

## Syntax

```
[waveform,info]=ltePRACH(ue,chs)
```

## Description

`[waveform,info]=ltePRACH(ue,chs)` returns a column vector, `waveform`, containing complex symbols of the Physical Random Access Channel given UE-specific settings structure, `ue`, and channel transmission configuration structure, `chs`. PRACH information is returned in a structure, `info`, as described in `ltePRACHInfo`. `waveform` contains the time-domain PRACH signal spanning `info.TotSubframes`, as described in TS 211, Section 5.7 [2]. The waveform consists of a period of zeros (for the case of a time offset or Preamble Format 4), a cyclic prefix, the “useful” part of the PRACH signal, and a period of zeros to extend the waveform to span `info.TotSubframes`. The duration of the PRACH is a function of the Preamble Format as described in TS 36.211, Table 5.7.1-1 [2]. Depending on the configuration given in `ue` and `chs`, it is possible that no PRACH are generated; in this case `info.PRBSet` is empty to signal this condition, and `waveform` consists of all zeros. The conditions under which no PRACH are generated are described in the help for `ltePRACHInfo`.

`chs.PreambleIdx` can be a vector in the functions `ltePRACHInfo` and `ltePRACHDetect`. This assists with modelling of an eNodeB receiver searching for multiple preambles. However, this function, `ltePRACH` only generates a single PRACH and therefore `chs.PreambleIdx` should be a scalar. If `chs.PreambleIdx` is a vector, the first element is used.

By default, for the given `ue.NULRB`, the `waveform` output, is sampled at the same sampling rate as other uplink channels (PUCCH, PUSCH, and SRS) using the `lteSCFDMAModulate` modulator.

If the value of `chs.PreambleIdx` is such that an insufficient quantity of cyclic shifts are available at the configured logical root index, `chs.SeqIdx`, the logical root index number needs to be incremented. As such, the physical root used, `info.RootSeq`, differs from the physical root configured by `chs.SeqIdx`. The cyclic shift corresponding to `chs.PreambleIdx` can be found in `info.CyclicShift`. For High Speed mode, when `info.CyclicShift` = –1, the PRACH waveform is generated with no cyclic shift.

## Examples

### Generate PRACH Symbols

This example generates PRACH symbols of format 0 in an ue.NULRB=6 bandwidth, leaving all other parameters at their default values.

Initialize ue-specific settings and channel transmission configuration.

```
ue.DuplexMode = 'FDD';
ue.NULRB = 6;
chs.Format = 0;
chs.HighSpeed = 0;
```

```
chs.CyclicShiftIdx = 0;
chs.FreqOffset = 0;
chs.SeqIdx = 0;
chs.PreambleIdx = [ 0 ];
```

Generate PRACH symbols and PRACH info.

```
[prachSym,prachInfo] = ltePRACH(ue,chs);
prachInfo
```

```
prachInfo = struct with fields:
                  NZC: 839
    SubcarrierSpacing: 1250
                  Phi: 7
                    K: 12
         TotSubframes: 1
               Fields: [0 3168 24576 2976]
               PRBSet: [6x1 double]
                  NCS: 0
          CyclicShift: 0
              RootSeq: 129
         SamplingRate: 1920000
           BaseOffset: 0
```

### Analyze PRACH Root Sequence Indices

Analyze physical root Zadoff-Chu sequence indices by generating 64 orthogonal PRACH preambles for two different PRACH configurations.

### Root Sequence Indices with Single Value

Initialize ue-specific settings and channel transmission configuration. Use PRACH symbols of format 0 in an ue.NULRB=6 bandwidth, leaving all other parameters at their default values.

```
ue.NULRB = 6;
chs.Format = 0;
```

Set the PRACH logical root sequence index to `0`. For this value, the value of the physical root sequence index is `129`, as defined in TS 36.211 Table 5.7.2-4.

```
chs.SeqIdx = 0;
```

Set the PRACH cyclic shift configuration index to `1`. For this value, each PRACH preamble has a different cyclic shift value, based on $N_{CS}$ from TS 36.211 Table 5.7.2-2.

```
chs.CyclicShiftIdx = 1;
```

Generate 64 PRACH preambles to store the physical root sequence indices and cyclic shift values.

```
rootSequence1 = NaN(1,64);
cyclicShift1 = NaN(1,64);
for preambleIndex = 0:63
    chs.PreambleIdx = preambleIndex;
    [~,info] = ltePRACH(ue,chs);
```

```
    rootSequence1(preambleIndex+1) = info.RootSeq;
    cyclicShift1(preambleIndex+1) = info.CyclicShift;
end
```

Verify that in each preamble, the physical root sequence index is `129`, which is the expected value from configuring the logical root sequence index to `0`.

```
disp(rootSequence1)
```

```
   129   129   129   129   129   129   129   129   129   129   129   129   129   129   129   129
```

Verify that each preamble has a different cyclic shift value.

```
disp(cyclicShift1)
```

```
     0    13    26    39    52    65    78    91   104   117   130   143   156   169   182   195
```

**Root Sequence Indices with Different Values**

Initialize the same ue-specific settings and channel transmission configuration.

```
ue.NULRB = 6;
chs.Format = 0;
```

Set the PRACH logical root sequence index to `0`. For this value, the value of the physical root sequence index is `129`, as defined in TS 36.211 Table 5.7.2-4.

```
chs.SeqIdx = 0;
```

Set the PRACH cyclic shift configuration index to `0`. For this value, each PRACH premable has the same cyclic shift value, equal to `0`, based on TS 36.211 Table 5.7.2-2.

```
chs.CyclicShiftIdx = 0;
```

Generate 64 PRACH preambles to store the physical root sequence indices and cyclic shift values.

```
rootSequence2 = NaN(1,64);
cyclicShift2 = NaN(1,64);
for preambleIndex = 0:63
    chs.PreambleIdx = preambleIndex;
    [~,info] = ltePRACH(ue,chs);
    rootSequence2(preambleIndex+1) = info.RootSeq;
    cyclicShift2(preambleIndex+1) = info.CyclicShift;
end
```

Check the physical root sequence indices and cyclic shift values. Even though the logical root sequence index, `chs.SeqIdx`, is `0`, not every physical root sequence index value is the expected value of `129`. Because the cyclic shift value is zero in each preamble, the function `ltePRACH` obtains the physical root sequence indices by taking consecutive logical index values. The returned physical root sequence indices correspond to logical indices 0 to 63 from TS 36.211 Table 5.7.2-4.

```
disp(rootSequence2)
```

```
   129   710   140   699   120   719   210   629   168   671    84   755   105   734    93   746
```

```
disp(cyclicShift2)
```

```
    0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
```

## Input Arguments

### ue — UE-specific settings

scalar structure

UE-specific settings, specified as a scalar structure. `ue` can contain the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NULRB`** | Required | 6, 9, 11, 15, 25, 27, 45, 50, 64, 75, 91, 100 | Number of uplink resource blocks. ($N_{RB}^{UL}$) |
| **`DuplexMode`** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| The following parameters are applicable when `DuplexMode` is set to `'TDD'`. | | | |
| **`TDDConfig`** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **`SSC`** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |
| The following parameter fields is applicable when `DuplexMode` is set to `'TDD'` or when `chs.ConfigIdx` is present. | | | |
| **`NSubframe`** | Optional | 0 (default), Nonnegative scalar integer | Subframe number |
| **`NFrame`** | Optional | 0 (default), nonnegative scalar integer | Frame number |
| The following parameter fields are dependent upon the condition that the Preamble Format (`chs.Format`) is set to `'4'`. | | | |
| **`CyclicPrefix`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |

Data Types: `struct`

### chs — Channel transmission configuration

scalar structure

Channel transmission configuration, specified as a scalar structure. `chs` can contain the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`Format`** | Optional | 0, 1, 2, 3, 4 (default is determined by `ConfigIdx` field if present). However, the `Format` field must be specified if the `ConfigIdx` field is not specified. | Preamble format<br><br>See Note 1. |
| **`SeqIdx`** | Optional | Scalar integer from 0 to 837. The default value is 0. | Logical root sequence index (*RACH_ROOT_SEQUENCE*) |
| **`ConfigIdx`** | Optional | Scalar integer from 0 to 63. The default value is determined by `Format` field, if present. However, the `ConfigIdx` field must be specified if the `Format` field is not specified. | PRACH Configuration Index (*prach-ConfigurationIndex*)<br><br>See Note 1. |
| **`PreambleIdx`** | Optional | Scalar integer or vector of integers from 0 to 63. The default value is 0. | Preamble index within cell (*ra-PreambleIndex*) |
| **`CyclicShiftIdx`** | Optional | Scalar integer from 0 to 15. The default value is 0. | Cyclic shift configuration index (*zeroCorrelationZoneConfig*, yields $N_{CS}$) |
| **`HighSpeed`** | Optional | 0 (default) or 1 | High Speed flag (*highSpeedFlag*). A value of 1 signifies a restricted set. A value of 0 signifies an unrestricted set. |
| **`TimingOffset`** | Optional | 0.0 (default), Numeric scalar | PRACH timing offset, in microseconds<br><br>See Note 2. |
| The following parameters are applicable when `ue.DuplexMode` is set to `'TDD'`. | | | |
| **`FreqIdx`** | Optional | 0 (default), 0, 1, 2, 3, 4, 5 | Frequency resource index ($f_{RA}$). Only required for `'TDD'` duplexing mode. |
| The following parameter fields are dependent upon the condition that the Preamble Format (`chs.Format`) is set to `0, 1, 2, or 3`. | | | |
| **`FreqOffset`** | Optional | Scalar integer from 0 to 94. The default value is 0. | PRACH frequency offset ($n_{PRBoffset}$). Only required for Preamble format 0–3. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|

**Note**

1. Although the parameters `chs.Format` and `chs.ConfigIdx` are both described as 'Optional', at least one of these parameters must be specified. If both parameters are present then `chs.Format` is used and `chs.ConfigIdx` is ignored.
2. The parameter `chs.TimingOffset` is not a genuine parameter of the PRACH generation as defined in the standard. It is provided to allow easy generation of a delayed PRACH output for use in testing, to simulate the effect of the distance between UE and eNodeB. The maximum value of `chs.TimingOffset` that yields a complete PRACH transmission in the output `waveform` is a timing offset equal to the duration of the last field of `info.Fields`; this timing offset corresponds to the maximum cell size and hence maximum distance between UE and eNodeB. If this maximum timing offset is exceeded, part of the PRACH signal is lost. The end of the useful part of the PRACH signal is out with the span of `waveform`.

Data Types: `struct`

## Output Arguments

### `waveform` — PRACH waveform symbols

complex-valued numeric column vector

PRACH waveform symbols, returned as a complex-valued numeric column vector. It contains the time-domain PRACH signal spanning `info.TotSubframes`. It has size $N$-by-1, where $N$ = `info.TotSubframes` × `30720` / `2048` × $N_{\text{fft}}$, where $N_{\text{fft}}$ is a function of the Number of Resource Blocks (NRB) given by `ue.NULRB`.

| NRB | $N_{\text{fft}}$ |
|---|---|
| 6 | 128 |
| 15 | 256 |
| 25 | 512 |
| 50 | 1024 |
| 75 | 2048 |
| 100 | 2048 |

In general, $N_{\text{fft}}$ is the smallest power of 2 greater than or equal to `12×NRB/0.85`. It is the smallest FFT that spans all subcarriers and results in a bandwidth occupancy ($12\times\text{NRB}/N_{\text{fft}}$) of no more than 85%.

Data Types: `double`

### `info` — PRACH information

scalar structure

PRACH information, returned as a scalar structure. It contains the following fields.

#### `NZC` — Zadoff-Chu sequence length

positive integer

Zadoff-Chu sequence length, returned as a positive integer. ($N_{\text{ZC}}$)

Data Types: `double`

**`SubcarrierSpacing` — Subcarrier spacing of PRACH preamble**
positive integer

Subcarrier spacing of PRACH preamble, in Hz, returned as a positive integer. (*deltaf_RA*)

Data Types: `double`

**`Phi` — Frequency-domain location offset**
positive integer

Frequency-domain location offset, returned as a positive integer. (*phi*)

Data Types: `double`

**`K` — Ratio of uplink data to PRACH subcarrier spacing**
numeric scalar

Ratio of uplink data to PRACH subcarrier spacing, returned as a numeric scalar. (*K*)

Data Types: `double`

**`TotSubframes` — Number of subframes duration of PRACH**
numeric scalar

Number of subframes duration of the PRACH, returned as a numeric scalar. Each subframe lasts 30720 fundamental periods, therefore `TotSubframes` is `ceil(sum(Fields)/30720)`, the number of subframes required to hold the entire PRACH waveform. The duration of the PRACH is a function of the Preamble Format as described in TS 36.211, Table 5.7.1-1 [2].

Data Types: `double`

**`Fields` — PRACH field lengths**
1-by-4 numeric vector

PRACH field lengths, returned as a 1-by-4 numeric vector. The elements are [*OFFSET T_CP T_SEQ GUARD*]. *T_CP* and *T_SEQ* are the lengths in fundamental time periods (*T_s*), of cyclic prefix and PRACH sequence, respectively. *OFFSET* is the number of fundamental time periods from the start of configured subframe to the start of the cyclic prefix, and is nonzero only for TDD special subframes. *GUARD* is the number of fundamental time periods from the end of the PRACH sequence to the end of the number of subframes spanned by the PRACH.

Data Types: `double`

**`PRBSet` — PRBs occupied by PRACH preamble**
nonnegative integer column vector

PRBs occupied by PRACH preamble, returned as a nonnegative integer column vector. (starts at $N_{\mathrm{PRB}}$, 0-based).

- An empty `info.PRBSet` indicates that the PRACH is not present and the waveform generated by ltePRACH consists of all zeros.
- An `info.PRBSet` that contains six consecutive Physical Resource Block numbers indicates the frequency domain location of the PRACH.

**Note** The PRACH uses a different SC-FDMA symbol construction from the other channels (PUCCH, PUSCH and SRS) and therefore the PRBSet indicates the frequency range (180kHz per RB) that the PRACH occupies, it does not occupy the set of 12 subcarriers in each RB in the same fashion as other channels. The PRACH occupies a bandwidth approximately equal to 1.08MHz = 6RBs.

Data Types: `uint32`

**`NCS` — Length of zero correlation zone plus 1**
positive integer

Length of zero correlation zone plus 1, specified as a positive integer (*N_CS*). `NCS` corresponds to the complete extent of autocorrelation lags (0 and *N_CS*–1 nonzero) that exhibit perfect correlation properties (1 at 0 lag, 0 at nonzero lags). `NCS` is expressed directly, as in the standard, related to the fundamental Zadoff-Chu sequence construction. The actual sample span of the zero correlation zone in the waveform generated by `ltePRACH` is a function of the sampling rate.

Data Types: `double`

**`CyclicShift` — Cyclic shift or shifts of Zadoff-Chu sequence**
numeric row vector

Cyclic shift or shifts of Zadoff-Chu sequence, returned as a numeric row vector. (*C_v*). For High Speed mode, any element of `CyclicShift` equal to –1 indicates that there are no cyclic shifts in the restricted set for the corresponding preamble index.

Data Types: `double`

**`RootSeq` — Physical root Zadoff-Chu sequence index or indices**
numeric row vector

Physical root Zadoff-Chu sequence index or indices, returned as a numeric row vector. (*u*)

Data Types: `double`

**`CyclicOffset` — Cyclic shift or shifts corresponding to Doppler Shift**
vector

Cyclic shift or shifts corresponding to Doppler Shift of (1/*T_SEQ*), returned as a vector. This parameter is present for High Speed mode. (*d_u*)

Data Types: `double`

**`SamplingRate` — Sampling rate of PRACH modulator**
numeric scalar

Sampling rate of the PRACH modulator, returned as a numeric scalar.

Data Types: `double`

**`BaseOffset` — Base timing offset**
numeric scalar

Base timing offset, in microseconds, returned as a numeric scalar. This parameter field is used for the detection test in TS 36.104 [1]. (duration of *N_CS*/2)

Data Types: `double`

Data Types: `struct`

**Note** Logical root sequence index `chs.SeqIdx` determines the returned physical root Zadoff-Chu sequence index `RootSeq`, based on TS 36.211 Table 5.7.2-4 and Table 5.7.2-5. However, if the preamble index within the cell, specified by `chs.PreambleIdx`, results in insufficient amount of cyclic shifts available at index `chs.SeqIdx`, the function `ltePRACH` obtains the physical root sequence index by taking consecutive logical root sequence indices, following the process described in TS 36.211 Section 5.7.2. In this case, the value of `RootSeq` differs from the expected index, specified by `chs.SeqIdx`. For an example, see “Analyze PRACH Root Sequence Indices” on page 2-678.

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.104. “Evolved Universal Terrestrial Radio Access (E-UTRA); Base Station (BS) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePRACHInfo` | `ltePRACHDetect`

# ltePRACHDetect

Physical random access channel detection

## Syntax

```
[indout,offset] = ltePRACHDetect(ue,chs,waveform,indin)
```

## Description

`[indout,offset] = ltePRACHDetect(ue,chs,waveform,indin)` performs PRACH detection given UE-specific settings structure, `ue`, channel configuration structure, `chs`, received signal potentially containing a PRACH transmission, `waveform`, and range of preamble indices for which to search, specified in `indin`. The detector performs each distinct correlation required to cover all preamble indices, specified in `indin`, and searches the output of the correlations for peaks which exceed a detection threshold. The position of the peak in the correlator output is used to determine the preamble index detected and its associated timing offset. The preamble index and timing offset are returned in `indout` and `offset` respectively. For more information, see “PRACH Detector” on page 2-689.

## Examples

### Detect PRACH Preamble

Detect a PRACH preamble which has been delayed by 7 samples.

Initialize configuration structures for ue-specific (`ue`) and channel (`chs`) parameters.

```
ue.NULRB = 6;
ue.DuplexMode = 'FDD';
chs.Format = 0;
chs.CyclicShiftIdx = 1;
chs.PreambleIdx = 44;
chs.HighSpeed = 0;
chs.FreqOffset = 0;
chs.SeqIdx = 0;
```

Generate transmit waveform containing PRACH. Insert a seven sample delay. Detect the PRACH.

```
tx = ltePRACH(ue,chs);
rx = [zeros(7,1); tx];
[index,offset] = ltePRACHDetect(ue,chs,rx,(0:63).')
```

```
index = 44
```

```
offset = 7.1895
```

The timing offset fractional part is an estimate of the fractional delay present in the correlation peak. This is due to the cyclic shift present in the PRACH preamble. A cyclic shift in the frequency domain is a delay in the time domain.

## Input Arguments

### `ue` — UE-specific settings

structure array

UE-specific settings, specified as a structure array. `ue` contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NULRB`** | Required | 6, 9, 11, 15, 25, 27, 45, 50, 64, 75, 91, 100 | Number of uplink resource blocks. ($N_{RB}^{UL}$) |
| **`DuplexMode`** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| The following parameters are dependent upon the condition that `DuplexMode` is set to `'TDD'`. | | | |
| **`TDDConfig`** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **`SSC`** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |
| The following parameter fields are dependent upon the condition that `DuplexMode` is set to `'TDD'` or when `chs.ConfigIdx` is present. | | | |
| **`NSubframe`** | Optional | 0 (default), Nonnegative scalar integer | Subframe number |
| **`NFrame`** | Optional | 0 (default), nonnegative scalar integer | Frame number |
| The following parameter fields are dependent upon the condition that the Preamble Format (`chs.Format`) is set to `'4'`. | | | |
| **`CyclicPrefix`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |

Data Types: `struct`

### `chs` — Channel transmission configuration

structure array

Channel transmission configuration, specified as a structure array. `chs` contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`Format`** | Optional | 0, 1, 2, 3, 4 (default is determined by `ConfigIdx` field if present). However, the `Format` field must be specified if the `ConfigIdx` field is not specified. | Preamble format<br><br>See Note. |
| **`SeqIdx`** | Optional | Scalar integer from 0 to 837. The default value is 0. | Logical root sequence index (*RACH_ROOT_SEQUENCE*) |
| **`ConfigIdx`** | Optional | Scalar integer from 0 to 63. The default value is determined by `Format` field, if present. However, the `ConfigIdx` field must be specified if the `Format` field is not specified. | PRACH Configuration Index (*prach-ConfigurationIndex*)<br><br>See Note. |
| **`CyclicShiftIdx`** | Optional | Scalar integer from 0 to 15. The default value is 0. | Cyclic shift configuration index (*zeroCorrelationZoneConfig*, yields $N_{CS}$) |
| **`HighSpeed`** | Optional | 0 (default) or 1 | High Speed flag (*highSpeedFlag*). A value of 1 signifies a restricted set. A value of 0 signifies an unrestricted set. |
| The following parameters are dependent upon the condition that `ue.DuplexMode` is set to `'TDD'`. | | | |
| **`FreqIdx`** | Optional | 0 (default), 0, 1, 2, 3, 4, 5 | Frequency resource index ($f_{RA}$). Only required for `'TDD'` duplexing mode. |
| The following parameter fields are dependent upon the condition that the Preamble Format (`chs.Format`) is set to `0, 1, 2, or 3`. | | | |
| **`FreqOffset`** | Optional | Scalar integer from 0 to 94. The default value is 0. | PRACH frequency offset ($n_{PRBoffset}$). Only required for Preamble format 0–3. |

**Note** Although the parameters `chs.Format` and `chs.ConfigIdx` are both described as 'Optional', at least one of these parameters must be specified. If both parameters are present, then `chs.Format` is used and `chs.ConfigIdx` is ignored.

Data Types: `struct`

### `waveform` — Received signal potentially containing PRACH transmission

numeric matrix

Received signal potentially containing PRACH transmission, specified as an *N*-by-*P* numeric matrix. This matrix contains the received time-domain signal in which to search for PRACH transmissions. *N* is the number of time-domain samples. *P* is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

**`indin` — Range of preamble indices within the cell for which to search**
column vector

Range of preamble indices within the cell for which to search, specified as a column vector. It can be from 1 through 64 in length, containing values from 0 through 63.

Data Types: `double`

## Output Arguments

**`indout` — Preamble index**
scalar | [ ], empty

Preamble index, returned as:

- a scalar, if an index from `indin` results in the maximum correlation above detection threshold.
- an empty,[ ], if no index from `indin` results in the maximum correlation above the detection threshold or the maximum correlation was obtained for an index not included in `indin`.

Data Types: `double`

**`offset` — Timing offset**
scalar | [ ], empty

Timing offset expressed in samples at the input sampling rate, returned as:

- a scalar, if an index from `indin` results in the maximum correlation above detection threshold.
- an empty,[ ], if no index from `indin` results in the maximum correlation above the detection threshold or the maximum correlation was obtained for an index not included in `indin`.

The timing offset estimate has an integer part corresponding to the correlation peak sample position and a fractional part estimating the fractional delay present in the correlation peak. The cyclic shift in the frequency domain present in the PRACH preamble can contribute to this fractional delay.

Data Types: `double`

## More About

### PRACH Detector

The detector performs each distinct correlation required to cover all preamble indices, specified in `indin`, and searches the output of the correlations for peaks which exceed a detection threshold. The position of the peak in the correlator output is used to determine the preamble index detected and its associated timing offset. The preamble index and timing offset are returned in `indout` and `offset` respectively. Generate the input `waveform` for one transmit antenna with the `ltePRACH` function. Generate input `waveform` with multiple transmit antenna (for example 2 or 4) using one of the channel model functions, `lteFadingChannel`, `lteHSTChannel`, or `lteMovingChannel`. Any other waveform provided must be sampled at the same sampling rate that `ltePRACH` would produce for the same configuration, specifically the same value of `ue.NULRB` as configured for the PRACH detector (`ltePRACHDetect`). The appropriate sampling rate can be found in the `SamplingRate` field of the output of `ltePRACHInfo`. Except for the case of the appropriate delay to position the transmission of

Preamble Format 4 in the UpPTS for TDD special subframes, it is assumed that any PRACH signal in `waveform` is synchronized such that the first sample of `waveform` corresponds to the start of an uplink subframe. Therefore, the detector interprets any delay from the start of `waveform` to the first sample of the PRACH therein as a timing offset.

The detector first calls `info=ltePRACHInfo` to establish the set of root sequences `info.RootSeq` required to cover all preamble indices in `indin`. A correlation is then performed for each distinct value in `info.RootSeq`, with the inputs to the correlation being the input `waveform` and a locally generated PRACH waveform. The correlation is performed in the frequency domain. Multiplication of the FFT of the useful part of the locally generated PRACH waveform by a portion of the input `waveform` extracted with the same timing as the useful part of the locally generated PRACH waveform, followed by an IFFT to give the correlation. Further fields from `info` are then used to establish the length of the window of the correlator output that corresponds to each preamble index, the zero correlation zone. The detector establishes the preamble index by testing of the position of the peak in the correlator output to determine if it lies in the window of the correlator output given by the cyclic shift for each preamble index in turn. The offset within the current window is used to compute the timing offset.

## Version History

**Introduced in R2014a**

## See Also

`ltePRACH` | `ltePRACHInfo`

# ltePRACHInfo

PRACH resource information

## Syntax

```
info = ltePRACHInfo(ue,chs)
```

## Description

`info = ltePRACHInfo(ue,chs)` returns `info`, a structure containing PRACH resource information given UE-specific settings, `ue`, and channel transmission configuration, `chs`. For more information, see “PRACH Information” on page 2-696.

## Examples

### Find Root Zadoff-Chu Sequences from PRACH Information

Find the set of root Zadoff-Chu sequences required for all preamble indices (0,...,63) in a cell.

```
ue.NULRB = 6;
config.Format = 0;
config.CyclicShiftIdx = 8;
config.PreambleIdx = (0:63);
prachInfo = ltePRACHInfo(ue,config);
unique(prachInfo.RootSeq)
```

```
ans = 1×4

   129   140   699   710
```

## Input Arguments

### ue — UE-specific settings

structure array

UE-specific settings, specified as a structure array that can contain these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NULRB** | Required | 6, 9, 11, 15, 25, 27, 45, 50, 64, 75, 91, 100 | Number of uplink resource blocks. ($N_{RB}^{UL}$) |
| **DuplexMode** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| The following parameters are dependent upon the condition that `DuplexMode` is set to `'TDD'`. | | | |
| **`TDDConfig`** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **`SSC`** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |
| The following parameter fields are dependent upon the condition that `DuplexMode` is set to `'TDD'` or when `chs.ConfigIdx` is present. | | | |
| **`NSubframe`** | Optional | 0 (default), Nonnegative scalar integer | Subframe number |
| **`NFrame`** | Optional | 0 (default), nonnegative scalar integer | Frame number |
| The following parameter fields are dependent upon the condition that the Preamble Format, `chs.Format`, is set to `'4'`. | | | |
| **`CyclicPrefix`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |

Data Types: `struct`

**`chs` — Channel transmission configuration**
scalar structure

Channel transmission configuration, specified as a scalar structure that can contain these parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`Format`** | Optional | 0, 1, 2, 3, 4 (default is determined by `ConfigIdx` field if present). However, the `Format` field must be specified if the `ConfigIdx` field is not specified. | Preamble format<br><br>See Note. |
| **`SeqIdx`** | Optional | Scalar integer from 0 to 837. The default value is 0. | Logical root sequence index (*RACH_ROOT_SEQUENCE*) |
| **`ConfigIdx`** | Optional | Scalar integer from 0 to 63. The default value is determined by `Format` field, if present. However, the `ConfigIdx` field must be specified if the `Format` field is not specified. | PRACH Configuration Index (*prach-ConfigurationIndex*)<br><br>See Note. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`PreambleIdx`** | Optional | Scalar integer or vector of integers from 0 to 63. The default value is 0. | Preamble index within cell (*ra-PreambleIndex*) |
| **`CyclicShiftIdx`** | Optional | Scalar integer from 0 to 15. The default value is 0. | Cyclic shift configuration index (*zeroCorrelationZoneConfig*, yields $N_{CS}$) |
| **`HighSpeed`** | Optional | 0 (default) or 1 | High Speed flag (*highSpeedFlag*). A value of 1 signifies a restricted set. A value of 0 signifies an unrestricted set. |
| The following parameters are dependent upon the condition that `ue.DuplexMode` is set to `'TDD'`. | | | |
| **`FreqIdx`** | Optional | 0 (default), 0, 1, 2, 3, 4, 5 | Frequency resource index ($f_{RA}$). Only required for `'TDD'` duplexing mode. |
| The following parameter fields are dependent upon the condition that the Preamble Format, `chs.Format`, is set to `0, 1, 2, or 3`. | | | |
| **`FreqOffset`** | Optional | Scalar integer from 0 to 94. The default value is 0. | PRACH frequency offset ($n_{PRBoffset}$). Only required for Preamble format 0–3. |

**Note** Although the parameters `chs.Format` and `chs.ConfigIdx` are both described as `'Optional'`, at least one of these parameters must be specified. If both parameters are present, then `chs.Format` is used and `chs.ConfigIdx` is ignored.

Data Types: `struct`

## Output Arguments

### `info` — PRACH resource information

scalar structure

PRACH resource information, returned as a scalar structure. `info` contains the following fields.

### `NZC` — Zadoff-Chu sequence length

positive integer

Zadoff-Chu sequence length, returned as a positive integer. (*N_ZC*)

Data Types: `double`

### `SubcarrierSpacing` — Subcarrier spacing of PRACH preamble

positive integer

Subcarrier spacing of PRACH preamble, in Hz, returned as a positive integer. (*deltaf_RA*)

Data Types: `double`

### `Phi` — Frequency-domain location offset

positive integer

Frequency-domain location offset, returned as a positive integer. (*phi*)

Data Types: `double`

**K — Ratio of uplink data to PRACH subcarrier spacing**
numeric scalar

Ratio of uplink data to PRACH subcarrier spacing, returned as a numeric scalar. (*K*)

Data Types: `double`

**`TotSubframes` — Number of subframes duration of PRACH**
numeric scalar

Number of subframes duration of the PRACH, returned as a numeric scalar. Each subframe lasts 30720 fundamental periods, therefore `TotSubframes` is `ceil(sum(Fields)/30720)`, the number of subframes required to hold the entire PRACH waveform. The duration of the PRACH is a function of the Preamble Format as described in TS 36.211, Table 5.7.1-1 [2].

Data Types: `double`

**`Fields` — PRACH field lengths**
1-by-4 numeric vector

PRACH field lengths, returned as a 1-by-4 numeric vector. The elements are [*OFFSET T_CP T_SEQ GUARD*]. *T_CP* and *T_SEQ* are the lengths in fundamental time periods (*T_s*), of cyclic prefix and PRACH sequence, respectively. *OFFSET* is the number of fundamental time periods from the start of configured subframe to the start of the cyclic prefix, and is non-zero only for TDD special subframes. *GUARD* is the number of fundamental time periods from the end of the PRACH sequence to the end of the number of subframes spanned by the PRACH.

Data Types: `double`

**`PRBSet` — PRBs occupied by PRACH preamble**
nonnegative integer column vector

PRBs occupied by PRACH preamble, returned as a nonnegative integer column vector. (starts at *n_PRB*, zero-based).

- If no PRACH is present, the `info.PRBSet` field is empty.
- If PRACH is present, the `info.PRBSet` field contains six consecutive Physical Resource Block (PRB) indices, indicating the frequency-domain location of the PRACH.

**Note** The PRACH uses a different SC-FDMA symbol construction from the other channels, PUCCH, PUSCH, and SRS. Specifically, the PRACH does not occupy the set of 12 subcarriers in each RB in the same fashion as other channels. Therefore, the `PRBSet` indicates the frequency range, 180 kHz per RB, occupied by the PRACH. The PRACH occupies a bandwidth approximately equal to 1.08 MHz, or 6RBs.

Data Types: `uint32`

**`NCS` — Length of zero correlation zone plus 1**
positive integer

Length of zero correlation zone plus 1, specified as a positive integer ($N_{CS}$). `NCS` corresponds to the complete extent of autocorrelation lags (0 and $N_{CS}$–1 non-zero) that exhibit perfect correlation

properties (1 at 0 lag, 0 at non-zero lags). `NCS` is expressed directly, as in the standard, related to the fundamental Zadoff-Chu sequence construction. The actual sample span of the zero correlation zone in the waveform generated by `ltePRACH` is a function of the sampling rate.

Data Types: `double`

**`CyclicShift` — Cyclic shift or shifts of Zadoff-Chu sequence**
numeric row vector

Cyclic shift or shifts of Zadoff-Chu sequence, returned as a numeric row vector. ($C\_v$).

For High Speed mode, any element of `CyclicShift` equal to –1 indicates that there are no cyclic shifts in the restricted set for the corresponding preamble index.

Data Types: `double`

**`RootSeq` — Physical root Zadoff-Chu sequence index or indices**
numeric row vector

Physical root Zadoff-Chu sequence index or indices, required to generate the PRACH for each of the configured set of preamble indices returned as a numeric row vector. ($u$) `RootSeq` is either a vector or a scalar aligned with the configuration of `chs.PreambleIdx`

Data Types: `double`

**`CyclicOffset` — Cyclic shift or shifts corresponding to Doppler Shift**
vector

`CyclicOffset` values are cyclic shifts corresponding to a Doppler Shift of 1/$T\_SEQ$ ($d\_u$).

For High Speed mode, the field `CyclicOffset` is present. It contains cyclic offset values for each of the configured set of preamble indices. `CyclicOffset` is either a vector or a scalar aligned with the configuration of `chs.PreambleIdx`.

Data Types: `double`

**`SamplingRate` — Sampling rate of PRACH modulator**
numeric scalar

Sampling rate of the PRACH modulator, returned as a numeric scalar. The function computes the sampling rate using the following equation: `SamplingRate = 30720000 / 2048` $\times N_{\text{fft}}$ where $N_{\text{fft}}$ is a function of the Number of Resource Blocks given by `ue.NULRB`.

| **NRB** | $N_{\text{fft}}$ |
|---|---|
| 6 | 128 |
| 15 | 256 |
| 25 | 512 |
| 50 | 1024 |
| 75 | 2048 |
| 100 | 2048 |

In general, $N_{\text{fft}}$ is the smallest power of 2 greater than or equal to `12×NRB/0.85`. It is the smallest FFT that spans all subcarriers and results in a bandwidth occupancy ($12\times\text{NRB}/N_{\text{fft}}$) of no more than 85%.

Data Types: `double`

**`BaseOffset` — Base timing offset**
numeric scalar

Base timing offset, in microseconds. This field is used for the detection test in TS 36.104 [1]. (duration of $N_{CS}/2$)

Data Types: `double`

Data Types: `struct`

## More About

### PRACH Information

The parameters “PRACH Mask Index” and “PRACH Resource Index,” described in TS 36.321 [3], are not explicit in the configuration, but are implicit in the choice of `ue.NSubframe` and `ue.NFrame`.

The PRACH is always be generated provided it fits with the overall duplexing arrangement. For FDD, the PRACH is generated in any subframe. For TDD, the PRACH is generated only in special subframes for Preamble Format 4, and in uplink subframes for Preamble Format 0-3, provided there are `info.TotSubframes` consecutive uplink subframes for the chosen TDD configuration starting from the current subframe.

If `chs.ConfigIdx` is present, further validation is used to comply with TS 36.211 [2], Table 5.7.1-2 for FDD and Table 5.7.1-4 for TDD. Specifically, `chs.Format`, if present, is validated against `chs.ConfigIdx` and a preamble is only generated in appropriate frames and subframes. If `chs.Format` is absent, the format is inferred, if possible, from `chs.ConfigIdx`. If the entry in TS 36.211 [2], Table 5.7.1-2 for FDD or Table 5.7.1-4 for TDD indicates “N/A” for the preamble format, an error is issued.

For TDD, `chs.FreqIdx` corresponds to the first entry in the quadruples in TS 36.211 [2], Table 5.7.1-4. The other three entries ($t_{RA}^{(0)}$, $t_{RA}^{(1)}$, $t_{RA}^{(2)}$) in the quadruple are specified by `ue.NSubframe` and `ue.NFrame`.

The PRACH is generated if a combination of `chs.ConfigIdx`, `ue.TDDConfig`, $t_{RA}^{(0)}$, $t_{RA}^{(1)}$, and $t_{RA}^{(2)}$ given by `ue.NSubframe`, `ue.NFrame`, and `chs.FreqIdx` appears in TS 36.211 [2], Table 5.7.1-4.

---

**Note** In accordance with this logic,

- if `chs.ConfigIdx` is absent, `ue.NSubframe` and `ue.NFrame` are not required at all for FDD.
- In the case that a preamble is not generated under these rules, `info.PRBSet` is empty and the waveform generated by `ltePRACH` consists of all zeros.

---

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.104. “Evolved Universal Terrestrial Radio Access (E-UTRA); Base Station (BS) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[3] 3GPP TS 36.214. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer; Measurements.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePRACH` | `ltePRACHDetect`

# ltePRBS

Pseudorandom binary sequence

## Syntax

```
[seq,cinit] = ltePRBS(cinit,n)
[seq,cinit] = ltePRBS(cinit,n,mapping)

[subseq,cinit] = ltePRBS(cinit,pn)
[subseq,cinit] = ltePRBS(cinit,pn,mapping)
```

## Description

`[seq,cinit] = ltePRBS(cinit,n)` returns the first `n` elements of the pseudorandom binary sequence (PRBS) generator when initialized with `cinit`. For uniformity with the channel specific PRBS functions, `ltePRBS` also returns the initialization value `cinit`.

`[seq,cinit] = ltePRBS(cinit,n,mapping)` allows control over the format of the returned sequence `seq` with the input `mapping`.

`[subseq,cinit] = ltePRBS(cinit,pn)` returns a subsequence of a full PRBS sequence, specified by `pn`.

`[subseq,cinit] = ltePRBS(cinit,pn,mapping)` allows additional control over the format of the returned subsequence, `subseq`, with the input `mapping`.

## Examples

### Generate PRBS from Physical Layer Cell Identity

Generate a pseudorandom binary sequence based on physical layer cell identity for RMC R.0.

Create cell-wide configuration structure for RMC R.0. Use the physical layer cell identity, `NCellID`, as an initial value to generate the pseudorandom binary sequence.

```
enb = lteRMCDL('R.0');
prbsSeq = ltePRBS(enb.NCellID,5)
```

```
prbsSeq = 5x1 logical array

   0
   0
   0
   0
   0
```

### Generate Pseudorandom Binary Sequence

Generate an unsigned pseudorandom binary sequence.

```
seq = ltePRBS(162,4);
seq(1:4)
```

```
ans = 4x1 logical array

   1
   0
   1
   1
```

### Generate Signed Pseudorandom Binary Sequence

Generate a signed pseudorandom binary sequence.

```
seq = ltePRBS(162,4,'signed');
seq(1:4)
```

```
ans = 4×1

    -1
     1
    -1
    -1
```

## Input Arguments

### `cinit` — Initialization value

32-bit integer

Initialization value, specified as a 32-bit integer.

Data Types: `int32` | `uint32` | `double`

### `n` — Number of elements in returned sequence

numeric scalar

Number of elements in returned sequence, `seq`, specified as a numeric scalar.

Data Types: `double`

### `pn` — Range of elements in returned subsequence

row vector

Range of elements in returned subsequence, `subseq`, specified as a row vector of `[p n]`. The subsequence returns `n` values of the PRBS generator, starting at position `p` (0-based).

Data Types: `double`

**`mapping` — Output sequence formatting**
`'binary'` (default) | `'signed'`

Output sequence formatting, specified as `'binary'` or `'signed'`. `mapping` controls the format of the returned sequence.

- `'binary'` maps `true` to 1 and `false` to 0.
- `'signed'` maps `true` to –1 and `false` to 1.

Data Types: `char` | `string`

## Output Arguments

**`seq` — Pseudorandom binary sequence**
logical column vector | numeric column vector

Pseudorandom binary sequence, returned as a logical column vector, or a numeric column vector. `seq` contains the first `n` values of the PRBS generator. If `mapping` is set to `'signed'`, `seq` is a vector of data type double. Otherwise, it is a vector of data type logical.

Data Types: `logical` | `double`

**`subseq` — Pseudorandom binary subsequence**
logical column vector | numeric column vector

Pseudorandom binary subsequence, returned as a logical column vector, or a numeric column vector. `subseq` contains the values of the PRBS generator specified by `pn`. If you set `mapping` to `'signed'`, the output data type is `double`. Otherwise, the output data type is `logical`.

Data Types: `logical` | `double`

# Version History

**Introduced in R2014a**

## See Also

`ltePDSCHPRBS` | `lteEPDCCHPRBS` | `ltePDCCHPRBS` | `ltePBCHPRBS` | `ltePCFICHPRBS` | `ltePHICHPRBS` | `ltePUCCH2PRBS` | `ltePUCCH3PRBS` | `ltePSBCHPRBS` | `ltePSCCHPRBS` | `ltePSSCHPRBS`

# ltePRS

Positioning reference signal

## Syntax

```
sym = ltePRS(enb)
```

## Description

`sym = ltePRS(enb)` returns a column vector containing the positioning reference signal (PRS) symbols for transmission in a single subframe on antenna port 6. These symbols are ordered as they should be mapped into the resource elements along with `ltePRSIndices`. As determined by the PRS subframe configuration and duplex mode, the output vector is empty if no PRS is scheduled in the subframe.

The optional `PRSPeriod` parameter controls the downlink subframes in which PRS is present. See the `ltePRSIndices` function reference page for details.

## Examples

### Generate Positioning Reference Signal Symbols

Generate the PRS symbols for subframe 0 of a 10MHz downlink.

Create a cell-wide configuration structure initialized for RMC R.2. Configure for full band PRS (`NPRSRB = NDLRB`). Configure Iprs = 0, which sets [Tprs Dprs] = [160 0].

```
rmc = lteRMCDL('R.2');
rmc.NPRSRB = rmc.NDLRB;
rmc.PRSPeriod = 0;
prsSymbols = ltePRS(rmc);
```

Generate PRS symbols.

```
prsSymbols(1:4)
```

```
ans = 4×1 complex

   0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure. `enb` contains the following parameter fields.

The parameters `TDDConfig` and `SSC` are only required if `DuplexMode` is set to `'TDD'`.

**`NDLRB` — Number of downlink resource blocks**
positive scalar integer (6,...,110)

Number of downlink resource blocks, specified as a positive scalar integer from 6 through 110.

Example: 45

Data Types: `double`

**`CellRefP` — Number of cell-specific reference signal antenna ports**
1 | 2 | 4

Number of cell-specific reference signal antenna ports, specified as a 1, 2, or 4.

Example: 1

Data Types: `double`

**`NCellID` — Physical layer cell identity number**
nonnegative scalar integer

Physical layer cell identity number, specified as a nonnegative scalar integer.

Example: 4

Data Types: `double`

**`NSubframe` — Subframe number**
nonnegative scalar integer

Subframe number, specified as nonnegative scalar integer.

Example: 5

Data Types: `double`

**`NFrame` — Frame number**
0 (default) | optional | nonnegative scalar integer

Frame number, specified as nonnegative scalar integer.

Example: 6

Data Types: `double`

**`CyclicPrefix` — Cyclic prefix length**
`'Normal'` (default) | optional | `'Extended'`

Cyclic prefix length, specified as a `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

**`DuplexMode` — Duplex mode type**
`'FDD'` (default) | optional | `'TDD'`

Duplex mode type, specified as `'FDD'` or `'TDD'`. Used for separating the transmission signals.

Data Types: `char` | `string`

**`TDDConfig` — Uplink or downlink configuration for TDD**
0 (default) | optional | nonnegative scalar integer (0,...,6)

Uplink or downlink configuration for TDD, specified as a nonnegative scalar integer from 0 through 6. Required only for `'TDD'` duplex mode.

Example: 4

Data Types: `double`

**`SSC` — Special subframe configuration**
0 (default) | optional | nonnegative scalar integer (0,...,9)

Special subframe configuration, specified as nonnegative scalar integer from 0 through 9. Required only for `'TDD'` duplex mode.

Example: 6

Data Types: `double`

**`NPRSRB` — Number of PRS physical resource blocks**
0,...,`NDLRB`

Number of PRS physical resource blocks, specified as nonnegative scalar integer from 0 through `NDLRB`.

Example: 8

Data Types: `double`

**`PRSPeriod` — Positioning reference signal (PRS) period**
`'On'` (default) | optional | `'Off'` | `[Iprs]` | `[Tprs Dprs]`

Positioning reference signal (PRS) period, specified as `'On'`, `'Off'`, a numeric scalar, or a 1-by-2 vector. This parameter controls the downlink subframes in which PRS will be present. For details, see `ltePRSIndices`.

Example: 0

Example: [160 0]

Data Types: `char` | `string` | `double`

## Output Arguments

**`sym` — Positioning Reference Signal (PRS) symbols**
complex numeric column vector

Positioning Reference Signal (PRS) symbols, returned as complex numeric column vector, for transmission in a single subframe on antenna port 6.

Example: 0.7071 + 0.7071i

Data Types: `double`

## Version History

**Introduced in R2014a**

## See Also

`ltePRSIndices` | `lteCellRS` | `lteDMRS` | `lteEPDCCHDMRS` | `lteCSIRS` | `ltePRBS`

# ltePRSIndices

PRS resource element indices

## Syntax

```
ind = ltePRSIndices(enb)
ind = ltePRSIndices(enb,opts)
```

## Description

`ind = ltePRSIndices(enb)` returns a column vector of one-based linear indices for the PRS elements in the subframe, given the cell-wide settings parameter structure, `enb`. The length of `ind` is the number of resource elements (NRE). It returns the indices for the Positioning Reference Signal (PRS) resource element (RE) locations transmitted on antenna port 6. By default, these indices are in one-based linear indexing form that can directly index elements in a matrix representing a single subframe of the port 6 resource grid. Other index representations can also be created. These indices are ordered as the complex PRS symbols should be mapped and will not include any elements allocated to PBCH, PSS, and SSS. A PRS subframe configuration schedule can be defined as required. If the subframe contains no PRS, `ind` is an empty vector.

The optional `enb.PRSPeriod` parameter controls the downlink subframes in which PRS will be present, either always `'On'` or `'Off'`, or defined by the scalar subframe configuration index, *Iprs* (0,...,2399), or the explicit subframe periodicity and offset pair, `[Tprs Dprs]`, as listed in TS 36.211 [1], Section 6.10.4.3. The PRS containing subframes are located in conjunction with the parameters `enb.NSubframe` and optional `enb.NFrame`. `NSubframe` can be greater than 10; thus, setting `NSubframe` to 11 is equivalent to setting `NSubframe` to 1 and `NFrame` to 1.

`ind = ltePRSIndices(enb,opts)` formats the returned indices using options defined in `opts`.

## Examples

### Generate PRS Resource Element Indices

Generate the PRS resource element (RE) indices for subframe 0 of a 10 MHz downlink.

Create a cell-wide configuration structure initialized for RMC R.2. Configure for full band PRS (`NPRSRB = NDLRB`). Configure Iprs = 0, which sets [Tprs Dprs] = [160 0].

```
rmc = lteRMCDL('R.2');
rmc.NPRSRB = rmc.NDLRB;
rmc.PRSPeriod = 0;
```

Generate PRS indices.

```
prsIndices = ltePRSIndices(rmc,'ind');
prsIndices(1:4)
```

```
ans = 4x1 uint32 column vector

   1804
```

```
1810
1816
1822
```

## Input Arguments

### enb — Cell-wide settings

structure

Cell-wide settings, specified as a structure. `enb` contains the following fields.

The parameters `TDDConfig` and `SSC` are only required if `DuplexMode` is set to `'TDD'`.

### NDLRB — Number of downlink resource blocks

6,...,110

Number of downlink resource blocks, specified as a nonnegative scalar integer from 6 through 110.

Example: 50

Data Types: `double`

### CellRefP — Number of cell-specific reference signal antenna ports

1 (default) | 2 | 4

Number of cell-specific reference signal antenna ports, specified as 1, 2, or 4.

Example: 1

Data Types: `double`

### NCellID — Physical layer cell identity

nonnegative scalar integer

Physical layer cell identity, specified as a nonnegative scalar integer.

Example: 3

Data Types: `double`

### NSubframe — Subframe number

nonnegative scalar integer

Subframe number, specified as a nonnegative scalar integer.

Example: 3

Data Types: `double`

### NFrame — Frame number

0 (default) | optional | nonnegative scalar integer

Frame number, specified as a nonnegative scalar integer.

Example: 3

Data Types: `double`

**`CyclicPrefix` — Cyclic prefix length**
`'Normal'` (default) | optional | `'Extended'`

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

**`DuplexMode` — Duplex mode type**
`'FDD'` (default) | optional | `'TDD'`

Duplex mode type, specified as `'FDD'` or `'TDD'`.

Data Types: `char` | `string`

**`TDDConfig` — Uplink or downlink configuration for TDD**
0 (default) | optional | 0,...,6

Uplink or downlink configuration for TDD, specified as a nonnegative scalar integer from 0 through 6. Optional. Required only for `'TDD'` duplex mode.

Example: 4

Data Types: `double`

**`SSC` — Special subframe configuration for TDD**
0 (default) | optional | 0,...,9

Example: 5

Special subframe configuration for TDD, specified as a nonnegative scalar integer from 0 through 9. Required only for `'TDD'` duplex mode.

Data Types: `double`

**`NPRSRB` — Number of PRS physical resource blocks**
0,...,`NDLRB`

Number of PRS physical resource blocks, specified as a nonnegative scalar integer from 0 through `NDLRB`.

Example: 32

Data Types: `double`

**`PRSPeriod` — Positioning reference signal (PRS) period**
`'On'` (default) | optional | `'Off'` | `[Iprs]` | `[Tprs Dprs]`

Positioning reference signal (PRS) period, specified as `'On'`, `'Off'`, a numeric scalar, or a 1-by-2 vector. This parameter controls the downlink subframes in which PRS will be present. For details, see `ltePRSIndices`.

Example: 0

Example: [160 0]

Data Types: `string` | `char` | `double`

**`opts` — Output format options for resource element indices**
character vector | cell array of character vectors | string array

Output format options for resource element indices, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string) :

| Category | Options | Description |
|---|---|---|
| Indexing style | `'ind'` (default) | The returned indices are in linear index style. |
| | `'sub'` | The returned indices are in `[subcarrier,symbol,port]` subscript row style. |
| Index base | `'1based'` (default) | The returned indices are one-based. |
| | `'0based'` | The returned indices are zero-based. |

Example: `'ind 1based'`, `"ind 1based"`, `{'ind','1based'}`, or `["ind","1based"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `ind` — PRS resource element indices

integer column vector | integer matrix

PRS resource element indices, returned as an integer column vector of length *NRE* or an integer matrix of size *NRE*-by-3. These indices are for the PRS resource element (RE) locations transmitted on antenna port 6.

Example: 1804

Data Types: `uint32`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePRS` | `lteCellRSIndices` | `lteCSIRSIndices` | `lteDMRSIndices`

# ltePSBCH

Physical sidelink broadcast channel

## Syntax

```
sym = ltePSBCH(ue,cw)
```

## Description

`sym = ltePSBCH(ue,cw)` returns a column vector containing the physical sidelink broadcast channel (PSBCH) symbols for the specified UE settings structure and PSBCH codeword bits. The function performs PSBCH-specific scrambling, QPSK modulation, and SC-FDMA transform precoding, as defined in TS 36.211 [1], Section 9.6. For more information, see “Physical Sidelink Broadcast Channel Processing” on page 2-711.

## Examples

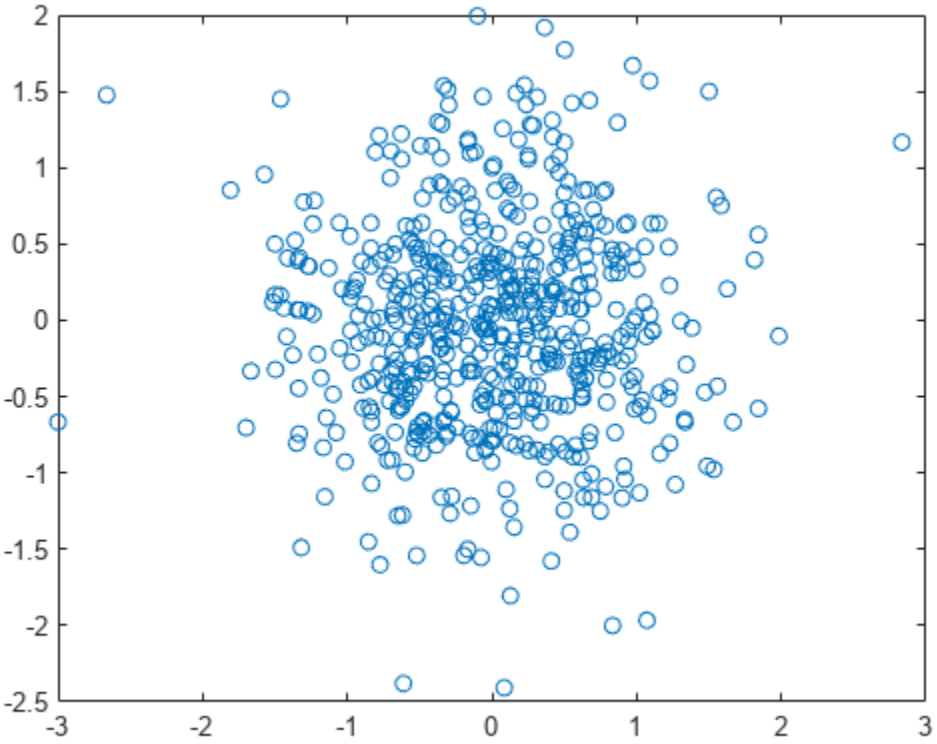
### Encode PSBCH Codeword

Create a codeword using the SL-BCH transport channel and encode the bits on the PSBCH.

```
ue.NSLID = 1;
ue.CyclicPrefixSL = 'Normal';
codeword = lteSLBCH(ue,zeros(40,1));
symbols = ltePSBCH(ue,codeword);
```

The plot shows the effects of the SC-FDMA precoding on the QPSK modulation symbols.

```
plot(symbols,'o')
```




## Input Arguments

### ue — User equipment settings

structure

User equipment settings, specified as a parameter structure containing this field:

### NSLID — Physical layer sidelink synchronization identity

integer from 0 to 355

Physical layer sidelink synchronization identity, specified as an integer from 0 to 355. ($N_{ID}^{SL}$)

Data Types: `double`

Data Types: `struct`

### cw — PSBCH codeword

vector

PSBCH codeword, specified as a vector that must be a multiple of 144 bits in length. Since the PSBCH is QPSK modulated, there are 2 bits per symbol. Nominally, the length of `cw` is 2*$N_{RE}$ bits, specifically 1152 bits for normal cyclic prefix or 864 for extended cyclic prefix. For V2X sidelink mode, the nominal length will be 1008 bits corresponding to 504 resource elements (it is defined for normal cyclic prefix only).

$N_{RE}$ is the number of resource elements in a subframe, including the SC-FDMA guard symbol, and is a multiple of 72. Nominally, $N_{RE}$ is 576 for normal cyclic prefix or 432 for extended cyclic prefix. For V2X sidelink mode, the nominal length will be 504 resource elements (it is defined for normal cyclic prefix only).

Data Types: `double`

## Output Arguments

### `sym` — Modulated PSBCH symbols

column vector

Modulated PSBCH symbols, returned as an $N_{RE}$-by-1 column vector.

$N_{RE}$ is the number of resource elements in a subframe, including the SC-FDMA guard symbol, and is a multiple of 72. Nominally, $N_{RE}$ is 576 for normal cyclic prefix or 432 for extended cyclic prefix. For V2X sidelink mode, the nominal length will be 504 resource elements (it is defined for normal cyclic prefix only).

Data Types: `double`

## More About

### Physical Sidelink Broadcast Channel Processing

The physical sidelink broadcast channel (PSBCH) is transmitted in the central 72 resource elements in the available SC-FDMA symbols of synchronization subframes. For D2D sidelink mode, the available symbols exclude the three symbols per slot assigned to the PSBCH DRS and sidelink synchronization signals. For V2X sidelink, a total of seven symbols will be excluded in a subframe (three symbols for PSBCH DRS and 4 for the PSSS/SSSS). The resource elements in the last SC-FDMA symbol within a subframe are counted in the mapping process. Before transmission, the PSBCH resource elements are removed from the last SC-FDMA symbol by `lteSCFDMAModulate` during the sidelink-specific SC-FDMA modulation and guard symbol creation.

If a terminal is transmitting a synchronization subframe, then it should be sent every 40 ms for D2D sidelink mode or every 160 ms for V2X, with the exact subframe dependent on the RRC-signaled subframe number offset (*syncOffsetIndicator-r12*). The subframe also contains values for the `ltePSBCHDRSIndices` on port 1010 and `ltePSSSIndices` and `lteSSSSIndices` on port 1020. No PSCCH or PSSCH transmission will occur in a sidelink subframe configured for synchronization purposes.

### Physical Sidelink Broadcast Channel Indexing

Use the `ltePSBCHIndices` indexing function and the corresponding `ltePSBCH` sequence function to populate the resource grid for the desired synchronization subframe number. The indices are ordered as the PSBCH QPSK modulation symbols should be mapped, applying frequency-first mapping, and include indices for the last SC-FDMA guard symbol. The PSBCH values returned by `ltePSBCH` are ordered as they should be mapped into the resource elements of the adjacent symbols using `ltePSBCHIndices`. For more information on mapping symbols to the resource element grid, see “Resource Grid Indexing”.

# Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePSBCHDecode` | `ltePSBCHIndices` | `ltePSBCHDRS`

# ltePSBCHDecode

PSBCH decoding

## Syntax

```
[softbits,symbols] = ltePSBCHDecode(ue,sym)
```

## Description

`[softbits,symbols] = ltePSBCHDecode(ue,sym)` returns a vector of log-likelihood ratio (LLR) soft bits and the intermediate QPSK modulation symbols for the specified UE settings structure (`ue`) and modulated PSBCH symbols (`sym`).

The PSBCH decoder performs the inverse of the `ltePSBCH` function processing, as defined in TS 36.211 [1], Section 9.6, which includes SC-FDMA transform deprecoding, QPSK demodulation, and PSBCH-specific descrambling.

## Examples

### Decode PSBCH

Demodulate PSBCH symbols for a SL-BCH codeword containing a modulated MIB-SL message with noise added. Plot the noisy RE symbols, the symbols prior to QPSK demodulation. and the resulting LLR soft bits.

Create a UE settings structure.

```
ue.NSLRB = 25;
ue.InCoverage = 1;
ue.DuplexMode = 'FDD';
ue.NFrame = 0;
ue.NSubframe = 0;
ue.CyclicPrefixSL = 'Normal';
ue.NSLID = 0;
```

Encode the MIB-SL message and add noise.

```
cw = lteSLBCH(ue,lteSLMIB(ue));
sym = ltePSBCH(ue,cw);
rxsym = awgn(sym,13,'measured');
```

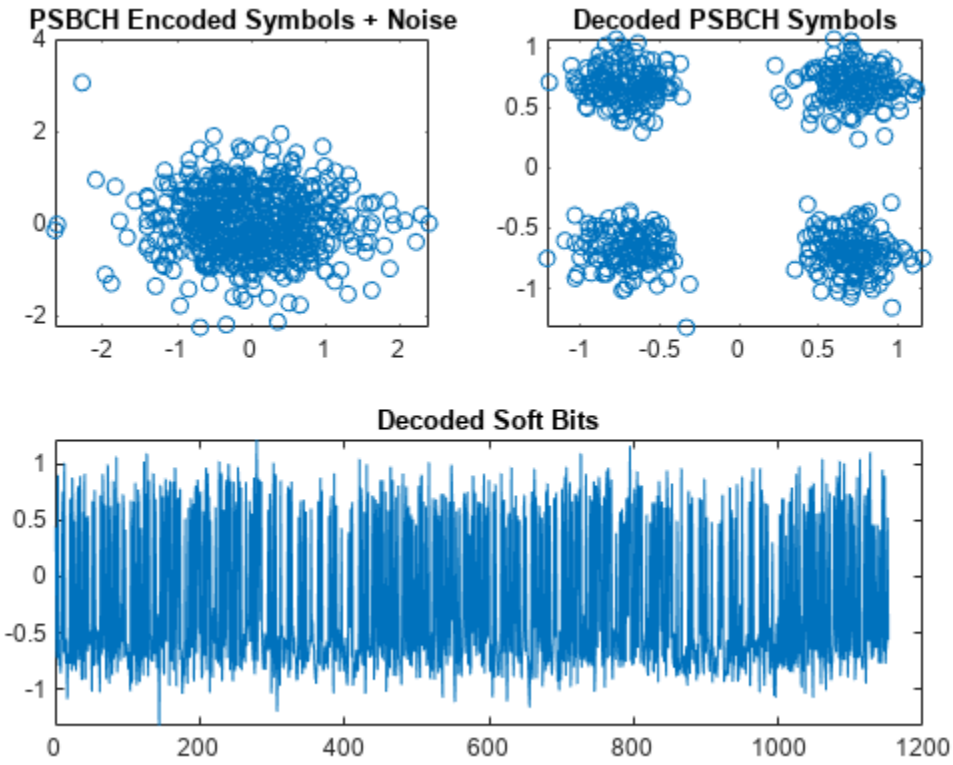
Decode the received symbols. The recovered codeword contains LLR soft bits. Hard decisions map positive soft bits to 1 and negative soft bits to 0. Compare the hard decisions on the recovered soft bits to verify that the recovered message matches the transmitted message.

```
[rxcw,rxmodsym] = ltePSBCHDecode(ue,rxsym);
isequal(cw,rxcw>0)
```

```
ans = logical
   1
```

Plot the noisy RE symbols, the symbols prior to QPSK demodulation, and the resulting LLR soft bits.

```
subplot(2,2,[1,1])
plot(rxsym,'o')
title('PSBCH Encoded Symbols + Noise')

subplot(2,2,[2,2])
plot(rxmodsym,'o')
title('Decoded PSBCH Symbols')

subplot(2,2,[3,4])
plot(rxcw)
title('Decoded Soft Bits')
```




## Input Arguments

### ue — User equipment settings

structure

User equipment settings, specified as a parameter structure containing this field:

### NSLID — Physical layer sidelink synchronization identity

integer from 0 to 355

Physical layer sidelink synchronization identity, specified as an integer from 0 to 355.

Data Types: `double`

Data Types: `struct`

**`sym` — Modulated PSBCH symbols**
column vector

Modulated PSBCH symbols, specified as a $N_{RE}$-by-1 column vector.

$N_{RE}$ is the number of resource elements in a subframe, including the SC-FDMA guard symbol, and is a multiple of 72. Nominally, $N_{RE}$ is 576 for normal cyclic prefix or 432 for extended cyclic prefix. For V2X sidelink mode, the nominal length will be 504 resource elements (it is defined for normal cyclic prefix only).

Data Types: `double`
Complex Number Support: Yes

## Output Arguments

**`softbits` — Log-likelihood ratio soft bits**
vector

Log-likelihood ratio (LLR) soft bits, returned as a vector with $2*N_{RE}$ elements.

$N_{RE}$ is the number of resource elements in a subframe, including the SC-FDMA guard symbol, and is a multiple of 72. Nominally, $N_{RE}$ is 576 for normal cyclic prefix or 432 for extended cyclic prefix. For V2X sidelink mode, the nominal length will be 504 resource elements (it is defined for normal cyclic prefix only).

Data Types: `double`

**`symbols` — Modulated PSBCH symbols**
column vector

Modulated PSBCH symbols, returned as a column vector with $N_{RE}$ elements.

$N_{RE}$ is the number of resource elements in a subframe, including the SC-FDMA guard symbol, and is a multiple of 72. Nominally, $N_{RE}$ is 576 for normal cyclic prefix or 432 for extended cyclic prefix. For V2X sidelink mode, the nominal length will be 504 resource elements (it is defined for normal cyclic prefix only).

Data Types: `double`

# Version History
**Introduced in R2016b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePSBCH` | `ltePSBCHIndices` | `ltePSBCHDRS` | `ltePSBCHDRSIndices`

# ltePSBCHDRS

PSBCH demodulation reference signal

## Syntax

```
[seq,info] = ltePSBCHDRS(ue)
```

## Description

`[seq,info] = ltePSBCHDRS(ue)` returns a 144-by-1 complex column vector sequence containing PSBCH demodulation reference signal (DM-RS) values and an associated information structure for the specified UE settings structure. For more information, see “PSBCH Demodulation Reference Signal” on page 2-720.

## Examples

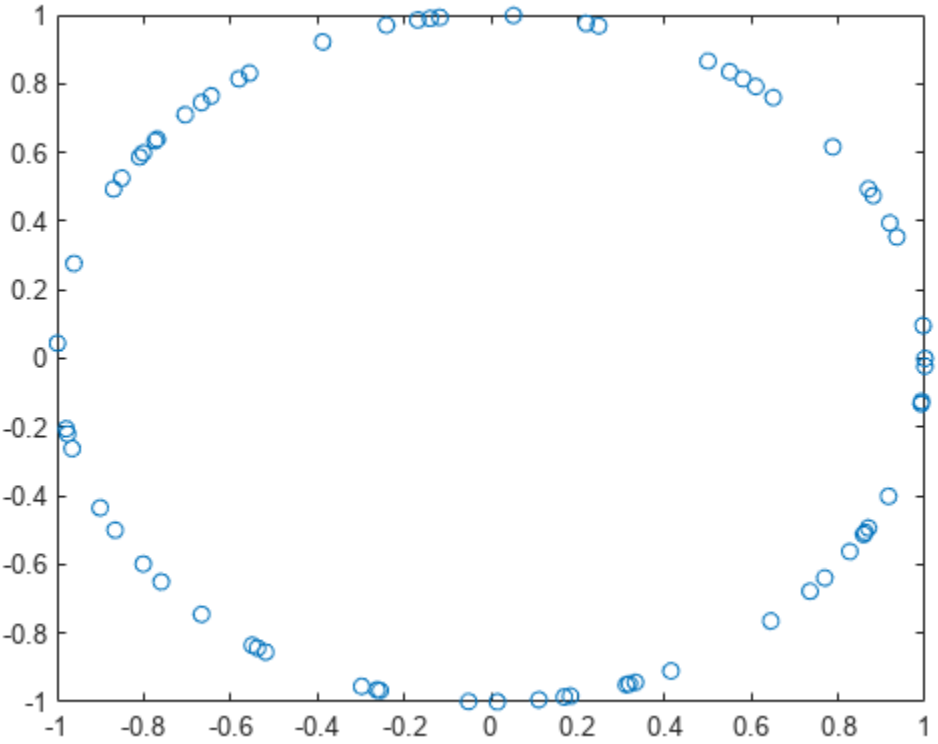
### Generate PSBCH DM-RS Sequence

Generate a PSBCH DM-RS sequence associated with both DM-RS SC-FDMA symbols in a subframe.

```
ue.NSLID = 170;
[psbchdrs_seq,info] = ltePSBCHDRS(ue);
```

Plot the DM-RS sequence (real vs. imaginary).

```
plot(psbchdrs_seq,'o')
```




## Input Arguments

### `ue` — User equipment settings

structure

User equipment settings, specified as a parameter structure containing this field:

### `SidelinkMode` — Sidelink mode

`'D2D'` (default) | `'V2X'` | optional

Sidelink mode, specified as `'D2D'` or `'V2X'`.

Data Types: `char` | `string`

### `NSLID` — Physical layer sidelink synchronization identity

integer from 0 to 355

Physical layer sidelink synchronization identity, specified as an integer from 0 to 355. ($N_{ID}^{SL}$)

Data Types: `double`

Data Types: `struct`

## Output Arguments

### `seq` — PSBCH DM-RS values

column vector

PSBCH DM-RS values, returned as a column vector. For the D2D sidelink mode, `seq` is a 144-length column containing the values for each DM-RS symbol in each slot in a subframe. For V2X, it is a 216-by-1 complex column containing the concatenated values for the three DM- RS symbols in a subframe.

Data Types: `double`

### `info` — PSBCH DM-RS information

structure

PSBCH DM-RS information about the intermediate variables used to create the DM-RS, returned as a parameter structure containing these fields:

#### `Alpha` — Reference signal cyclic shift for each slot

two-column vector

Reference signal cyclic shift for each slot, returned as a two-column vector. ($\alpha$)

`Alpha` is proportional to `NCS`, where $\alpha = \frac{2\pi n_{cs,\lambda}}{12}$.

#### `SeqGroup` — Base sequence group number for each slot

two-column vector

Base sequence group number for each slot, returned as a two-column vector. ($u$)

#### `SeqIdx` — Base sequence number for each slot

two-column vector

Base sequence number for each slot, returned as a two-column vector. ($v$)

#### `RootSeq` — Root Zadoff-Chu sequence index for each slot

two-column vector

Root Zadoff-Chu sequence index for each slot, returned as a two-column vector. ($q$)

#### `NCS` — Cyclic shift values for each slot

two-column vector

Cyclic shift values for each slot, returned as a two-column vector. ($n_{\mathrm{cs},\lambda}$)

#### `NZC` — Zadoff-Chu sequence length

integer

Zadoff-Chu sequence length, returned as an integer. ($N_{ZC}^{RS}$)

#### `OrthSeq` — Orthogonal cover value for each slot

matrix

Orthogonal cover value for each slot, returned as a matrix. ($\overline{w}$)

Data Types: `struct`

## More About

### PSBCH Demodulation Reference Signal

The PSBCH demodulation reference signal (DM-RS) is transmitted alongside the `ltePSBCH` values in the central 72 resource elements and in two SC-FDMA symbols in a synchronization subframe for D2D sidelink mode and three SC-FDMA symbols for V2X.. For zero-based indexing, the SC-FDMA symbol indices are {3,10} for normal cyclic prefix and {2,8} for extended cyclic prefix. These are the same symbols used by the PUSCH DM-RS, see `ltePUSCHDRSIndices`. For the V2X sidelink mode, the symbol indices are {4,6,9}.

**Note** The indicated symbol indices are based on TS 36.211, Section 9.8, but expanded from symbol index per slot to symbol index per subframe to align with the LTE Toolbox subframe orientation. For more information on mapping symbols to the resource element grid, see “Resource Grid Indexing”.

### PSBCH Demodulation Reference Signal Indexing

Use the indexing function, `ltePSBCHDRSIndices`, and the corresponding sequence function, `ltePSBCHDRS`, to index the resource grid for any synchronization subframe number. The indices are ordered as the PSBCH DM-RS symbols should be, applying frequency-first mapping, into the two DM-RS SC-FDMA symbols. For more information on mapping symbols to the resource element grid, see “Resource Grid Indexing”.

# Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePSBCHDRSIndices` | `ltePSBCH` | `ltePSBCHDecode` | `ltePSBCHIndices`

# ltePSBCHDRSIndices

PSBCH DM-RS resource element indices

## Syntax

```
ind = ltePSBCHDRSIndices(ue)
ind = ltePSBCHDRSIndices(ue,opts)
```

## Description

`ind = ltePSBCHDRSIndices(ue)` returns the subframe resource element (RE) indices for the demodulation reference signal (DM-RS) associated with a PSBCH transmission for the specified UE settings structure. By default, the indices are returned in one-based linear indexing form. You can use this form to directly index elements of a matrix representing the subframe resource grid for antenna port 1010. For more information, see “PSBCH Demodulation Reference Signal Indexing” on page 2-723.

`ind = ltePSBCHDRSIndices(ue,opts)` formats the returned indices using options specified by `opts`.

## Examples

### Create PSBCH DM-RS Values

Write the complex PSBCH DM-RS values into the PSBCH DM-RS resource elements in a synchronization subframe for both D2D and V2X sidelink modes and display an image of their locations.

Create a user equipment settings structure and an empty resource grid subframe for 10 MHz bandwidth and D2D sidelink mode.

```
ue = struct();
ue.NSLRB = 50;
ue.CyclicPrefixSL = 'Normal';
ue.NSLID = 1;
subframe_D2D = lteSLResourceGrid(ue);
```

Generate PSBCH DM-RS indices and load PSBCH DM-RS values into `subframe`.

```
psbchdrs_indices = ltePSBCHDRSIndices(ue);
subframe_D2D(psbchdrs_indices) = ltePSBCHDRS(ue);
```

Change user equipment settings to V2X sidelink mode.

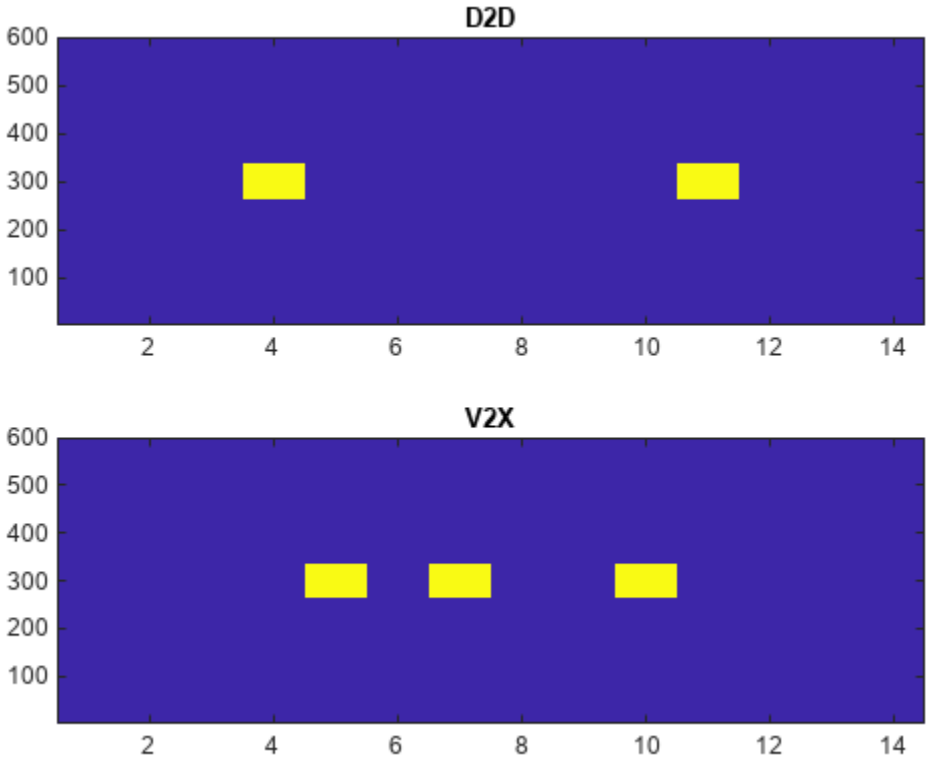
```
ue.SidelinkMode = 'V2X';
```

Generate the grid subframe, the PSBCH DM-RS indices and load PSBCH DM-RS values into `subframe_V2X`.

```
subframe_V2X = lteSLResourceGrid(ue);
psbchdrs_indices = ltePSBCHDRSIndices(ue);
subframe_V2X(psbchdrs_indices) = ltePSBCHDRS(ue);
```

Display the PSBCH DM-RS locations for both sidelink modes.

```
subplot(2,1,1);
imagesc(100*abs(subframe_D2D));
axis xy; title('D2D');
subplot(2,1,2);
imagesc(100*abs(subframe_V2X));
axis xy; title(ue.SidelinkMode);
```




## Input Arguments

### ue — User equipment settings

structure

UE equipment settings, specified as a parameter structure containing these fields:

### SidelinkMode — Sidelink mode

`'D2D'` (default) | `'V2X'` | optional

Sidelink mode, specified as `'D2D'` or `'V2X'`.

Data Types: `char` | `string`

**`NSLRB` — Number of sidelink resource blocks**
integer scalar from 6 to 110

Number of sidelink resource blocks, specified as an integer scalar from 6 to 110.

Example: `6`, which corresponds to a channel bandwidth of 1.4 MHz.

Data Types: `double`

**`CyclicPrefixSL` — Cyclic prefix length**
`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

**`opts` — Output format options for resource element indices**
character vector | cell array of character vectors | string array

Output format options for resource element indices, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string) :

| Category | Options | Description |
|---|---|---|
| Indexing style | `'ind'` (default) | The returned indices are in linear index style. |
| | `'sub'` | The returned indices are in `[subcarrier,symbol,port]` subscript row style. |
| Index base | `'1based'` (default) | The returned indices are one-based. |
| | `'0based'` | The returned indices are zero-based. |

Example: `'ind 1based'`, `"ind 1based"`, `{'ind','1based'}`, or `["ind","1based"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

**`ind` — Resource element indices**
integer column vector | three-column integer matrix

Resource element indices, returned as an integer column vector or a three-column integer matrix. By default, the indices are returned in a 144-by-1 column vector in one-based linear indexing form. You can use this form to directly access elements of a matrix representing the subframe resource grid for antenna port 1010. For V2X sidelink, the output is a 216-by-1 complex column for the three DM-RS symbols in a subframe. To specify alternative indexing formats, use the `opts` input argument.

## More About

### PSBCH Demodulation Reference Signal Indexing

Use the indexing function, `ltePSBCHDRSIndices`, and the corresponding sequence function, `ltePSBCHDRS`, to index the resource grid for any synchronization subframe number. The indices are

ordered as the PSBCH DM-RS symbols should be, applying frequency-first mapping, into the two DM-RS SC-FDMA symbols. For more information on mapping symbols to the resource element grid, see “Resource Grid Indexing”.

#### PSBCH Demodulation Reference Signal

The PSBCH demodulation reference signal (DM-RS) is transmitted alongside the `ltePSBCH` values in the central 72 resource elements and in two SC-FDMA symbols in a synchronization subframe for D2D sidelink mode and three SC-FDMA symbols for V2X.. For zero-based indexing, the SC-FDMA symbol indices are {3,10} for normal cyclic prefix and {2,8} for extended cyclic prefix. These are the same symbols used by the PUSCH DM-RS, see `ltePUSCHDRSIndices`. For the V2X sidelink mode, the symbol indices are {4,6,9}.

**Note** The indicated symbol indices are based on TS 36.211, Section 9.8, but expanded from symbol index per slot to symbol index per subframe to align with the LTE Toolbox subframe orientation. For more information on mapping symbols to the resource element grid, see “Resource Grid Indexing”.

## Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePSBCHDRS` | `ltePSBCH` | `ltePSBCHDecode` | `ltePUSCHDRSIndices`

# ltePSBCHIndices

PSBCH resource element indices

## Syntax

```
ind = ltePSBCHIndices(ue)
ind = ltePSBCHIndices(ue,opts)
```

## Description

`ind = ltePSBCHIndices(ue)` returns a column vector of physical sidelink broadcast channel (PSBCH) resource element (RE) indices for the specified UE settings structure. By default, the indices are returned in one-based linear indexing form. You can use this form to directly index elements of a matrix representing the subframe resource grid for antenna port 1010. For more information, see “Physical Sidelink Broadcast Channel Indexing” on page 2-728.

`ind = ltePSBCHIndices(ue,opts)` formats the returned indices using options specified by `opts`.

## Examples

### Generate PSBCH Indices

Generate PSBCH values and indices. Write the values into the PSBCH resource elements in a synchronization subframe for both D2D and V2X sidelink modes, and display an image of their locations. This mapping also writes the PSBCH values into the last SC-FDMA guard symbol within a subframe. The sidelink SC-FDMA modulator removes PSBCH values from the last SC-FDMA guard symbol in a separate processing step.

Create a user equipment settings structure and a resource grid that has 10 MHz bandwidth and normal cyclic prefix for D2D sidelink mode.

```
ue.NSLRB = 50;
ue.CyclicPrefixSL = 'Normal';
ue.NSLID = 1;
```

Generate an empty resource grid and PSBCH indices. Load the PSBCH indices into the resource grid.

```
grid_D2D = lteSLResourceGrid(ue);
psbch_indices = ltePSBCHIndices(ue);
grid_D2D(psbch_indices) = ltePSBCH(ue,zeros(2*576,1));
```

Change user equipment settings to V2X sidelink mode.

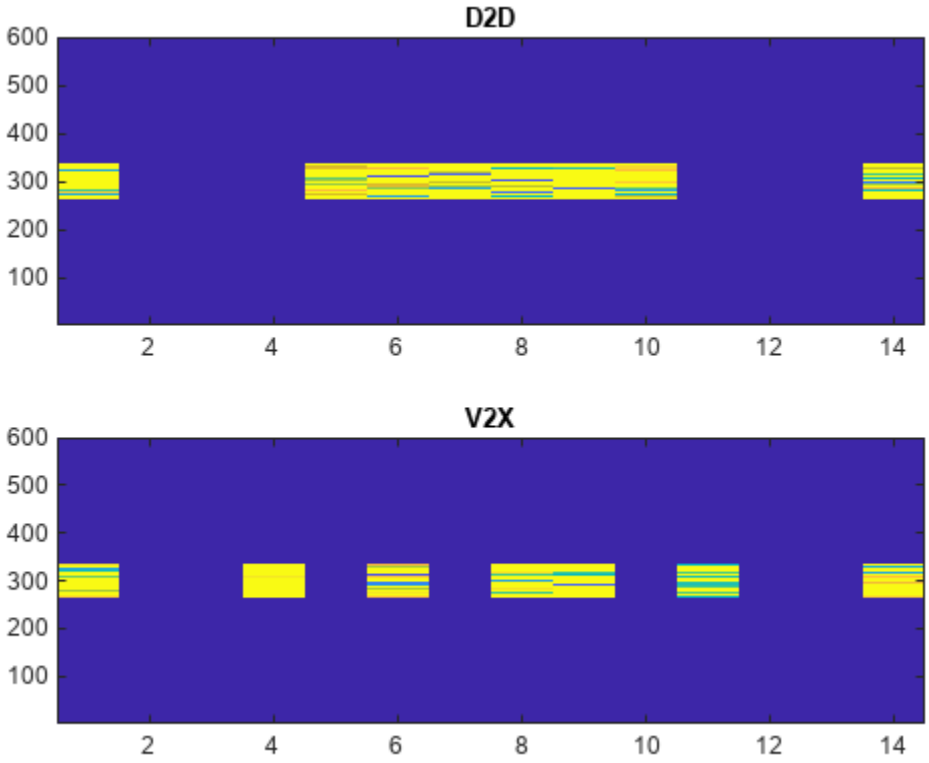
```
ue.SidelinkMode = 'V2X';
```

Generate an empty resource grid and PSBCH indices. Load the PSBCH indices into the resource grid.

```
grid_V2X = lteSLResourceGrid(ue);
psbch_indices = ltePSBCHIndices(ue);
grid_V2X(psbch_indices) = ltePSBCH(ue,zeros(2*504,1));
```

Display the locations of the PSBCH indices for both sidelink modes.

```
subplot(2,1,1);
image(400*abs(grid_D2D));
axis xy; title('D2D');
subplot(2,1,2);
image(400*abs(grid_V2X));
axis xy; title(ue.SidelinkMode);
```




### Generate Zero-Based PSBCH Indices

Generate PSBCH indices using zero-based indexing style. Compare these indices to one-based indices.

Create a user equipment settings structure with 10 MHz bandwidth and normal cyclic prefix.

```
ue.NSLRB = 50;
ue.CyclicPrefixSL = 'Normal';
ue.NSLID = 1;
```

Generate PSBCH zero-based indices. View the first five indices.

```
psbch_indices = ltePSBCHIndices(ue,'0based');
psbch_indices_size = size(psbch_indices)
```

```
psbch_indices_size = 1×2

   576     1
```

```
psbch_indices(1:5)
```

```
ans = 5x1 uint32 column vector

   264
   265
   266
   267
   268
```

Generate PSBCH one-based indices and view the first five indices.

```
psbch_indices = ltePSBCHIndices(ue,'1based');
psbch_indices_size = size(psbch_indices)
```

```
psbch_indices_size = 1×2

   576     1
```

```
psbch_indices(1:5)
```

```
ans = 5x1 uint32 column vector

   265
   266
   267
   268
   269
```

For zero-based indexing, the first assigned index is one lower than the one-based indexing style.

## Input Arguments

### ue — User equipment settings

structure

User equipment settings, specified as a parameter structure containing these fields:

#### SidelinkMode — Sidelink mode

`'D2D'` (default) | `'V2X'` | optional

Sidelink mode, specified as `'D2D'` or `'V2X'`.

Data Types: `char` | `string`

#### NSLRB — Number of sidelink resource blocks

integer scalar from 6 to 110

Number of sidelink resource blocks, specified as an integer scalar from 6 to 110.

Example: 6, which corresponds to a channel bandwidth of 1.4 MHz.

Data Types: double

**CyclicPrefixSL — Cyclic prefix length**
'Normal' (default) | 'Extended' | optional

Cyclic prefix length, specified as 'Normal' or 'Extended'.

Data Types: char | string

**opts — Output format options for resource element indices**
character vector | cell array of character vectors | string array

Output format options for resource element indices, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for opts when specified as a character vector include (use double quotes for string) :

| Category | Options | Description |
|---|---|---|
| Indexing style | 'ind' (default) | The returned indices are in linear index style. |
| | 'sub' | The returned indices are in [subcarrier,symbol,port] subscript row style. |
| Index base | '1based' (default) | The returned indices are one-based. |
| | '0based' | The returned indices are zero-based. |

Example: 'ind 1based', "ind 1based", {'ind','1based'}, or ["ind","1based"] specify the same formatting options.

Data Types: char | string | cell

## Output Arguments

**ind — PSBCH resource element indices**
integer column vector | three-column integer matrix

PSBCH resource element indices, returned as an integer column vector or a three-column integer matrix. By default, the returned vector or matrix has 576 PSBCH resource element indices for normal cyclic prefix or 432 PSBCH resource element indices for extended cyclic prefix. For the V2X sidelink mode, there are 504 PSBCH resource element indices. For more information, see “Physical Sidelink Broadcast Channel Indexing” on page 2-728.

Data Types: uint32

## More About

### Physical Sidelink Broadcast Channel Indexing

Use the ltePSBCHIndices indexing function and the corresponding ltePSBCH sequence function to populate the resource grid for the desired synchronization subframe number. The indices are ordered as the PSBCH QPSK modulation symbols should be mapped, applying frequency-first mapping, and

include indices for the last SC-FDMA guard symbol. The PSBCH values returned by `ltePSBCH` are ordered as they should be mapped into the resource elements of the adjacent symbols using `ltePSBCHIndices`. For more information on mapping symbols to the resource element grid, see “Resource Grid Indexing”.

#### Physical Sidelink Broadcast Channel

The physical sidelink broadcast channel (PSBCH) is transmitted in the central 72 resource elements in the available SC-FDMA symbols of synchronization subframes. For D2D sidelink mode, the available symbols exclude the three symbols per slot assigned to the PSBCH DRS and sidelink synchronization signals. For V2X sidelink, a total of seven symbols will be excluded in a subframe (three symbols for PSBCH DRS and 4 for the PSSS/SSSS). The resource elements in the last SC-FDMA symbol within a subframe are counted in the mapping process. Before transmission, the PSBCH resource elements are removed from the last SC-FDMA symbol by `lteSCFDMAModulate` during the sidelink-specific SC-FDMA modulation and guard symbol creation.

If a terminal is transmitting a synchronization subframe, then it should be sent every 40 ms for D2D sidelink mode or every 160 ms for V2X, with the exact subframe dependent on the RRC-signaled subframe number offset (*syncOffsetIndicator-r12*). The subframe also contains values for the `ltePSBCHDRSIndices` on port 1010 and `ltePSSSIndices` and `lteSSSSIndices` on port 1020. No PSCCH or PSSCH transmission will occur in a sidelink subframe configured for synchronization purposes.

## Version History

**Introduced in R2016b**

## See Also

`ltePSBCH` | `ltePSBCHDecode` | `ltePSBCHDRSIndices` | `lteSLBCHDecode`

# ltePSBCHPRBS

PSBCH pseudorandom binary scrambling sequence

## Syntax

```
[seq,cinit] = ltePSBCHPRBS(ue,n)
[seq,cinit] = ltePSBCHPRBS(ue,n,mapping)

[subseq,cinit] = ltePSBCHPRBS(ue,pn)
[subseq,cinit] = ltePSBCHPRBS(ue,pn,mapping)
```

## Description

`[seq,cinit] = ltePSBCHPRBS(ue,n)` returns a column vector containing the first `n` outputs of the PSBCH pseudorandom binary scrambling sequence (PRBS) for the specified UE settings structure. It also returns an initialization value `cinit` for the PRBS generator.

The scrambling sequence generated should be applied to the coded PSBCH data carried by the associated subframe. The PRBS generator is initialized with $c_{\text{init}}$ = `ue.NSLID`.

`[seq,cinit] = ltePSBCHPRBS(ue,n,mapping)` specifies the format of the returned sequence, `seq`, through the `mapping` input.

`[subseq,cinit] = ltePSBCHPRBS(ue,pn)` returns a subsequence of a full PRBS sequence, specified by `pn`.

`[subseq,cinit] = ltePSBCHPRBS(ue,pn,mapping)` specifies the format of the returned subsequence, `subseq`, through the `mapping` input.

## Examples

### Scramble PSBCH Codeword

Scramble a PSBCH codeword by generating the PSBCH pseudorandom binary sequence (PRBS) and applying an exclusive OR operation on the two sequences.

Create a UE configuration structure and SL-BCH codeword. Generate the required length of the PRBS and scramble the PSBCH codeword with the PRBS sequence using `xor`.

```
ue = struct('NSLID',2);
codeword = lteSLBCH(ue,ones(40,1));
psbchPrbs = ltePSBCHPRBS(ue,length(codeword));
scrambled = xor(psbchPrbs,codeword);
```

### Descramble PSBCH Codeword

Descramble a received PSBCH codeword.

### Scramble PSBCH Codeword

- Create a UE configuration structure and SL-BCH codeword.
- Generate the required length of the PRBS and scramble the PSBCH codeword with the PRBS sequence using `xor`.
- Modulate the logical scrambled data.

```
ue = struct('NSLID',2);
codeword = lteSLBCH(ue,ones(40,1));

psbchPrbs = ltePSBCHPRBS(ue,length(codeword));
scrambled = xor(psbchPrbs,codeword);

txsym = lteSymbolModulate(scrambled,'QPSK');
```

### Descramble Recovered Codeword

- Add noise to transmitted symbols and demodulate received soft data.
- Generate the PSBCH PRBS in signed form.
- Descramble a vector of noisy demodulated symbols representing a sequence of soft bits. To do so, perform a pointwise multiplication between the PRBS sequence and the recovered data.
- Compare the transmitted codeword to the recovered codeword.

```
rxsym = awgn(double(txsym),30,'measured');
softdata = lteSymbolDemodulate(rxsym,'QPSK');

scramblingSeq = ltePSBCHPRBS(ue,length(softdata),'signed');

descrambled = softdata.*scramblingSeq;

isequal(codeword,descrambled > 0)
```

```
ans = logical
   1
```

The transmitted codeword matches the hard decision on the descrambled data.

## Input Arguments

### ue — User equipment settings

structure

User equipment settings, specified as a parameter structure containing this field:

#### NSLID — Physical layer sidelink synchronization identity

integer from 0 to 355

Physical layer sidelink synchronization identity, specified as an integer from 0 to 355. ($N_{\mathrm{ID}}^{\mathrm{SL}}$)

Data Types: `double`

Data Types: `struct`

**n — Number of elements in returned sequence**
nonnegative integer

Number of elements in returned sequence, `seq`, specified as a nonnegative integer.

Data Types: `double`

**pn — Range of elements in returned subsequence**
row vector

Range of elements in returned subsequence, `subseq`, specified as a row vector of `[p n]`. The subsequence returns `n` values of the PRBS generator, starting at position `p` (0-based).

Data Types: `double`

**mapping — Output sequence formatting**
`'binary'` (default) | `'signed'`

Output sequence formatting, specified as `'binary'` or `'signed'`.

- `'binary'` maps `true` to 1 and `false` to 0.
- `'signed'` maps `true` to –1 and `false` to 1.

Data Types: `char` | `string`

## Output Arguments

**seq — PSBCH pseudorandom scrambling sequence**
logical column vector | numeric column vector

PSBCH pseudorandom scrambling sequence, returned as a logical column vector or a numeric column vector. `seq` contains the first `n` outputs of the physical sidelink broadcast channel (PSBCH) scrambling sequence. If you set `mapping` to `'signed'`, the output data type is `double`. Otherwise, the output data type is `logical`.

Data Types: `logical` | `double`

**subseq — PSBCH pseudorandom scrambling subsequence**
logical column vector | numeric column vector

PSBCH pseudorandom scrambling subsequence, returned as a logical column vector or a numeric column vector. `subseq` contains the values of the PRBS generator specified by `pn`. If you set `mapping` to `'signed'`, the output data type is `double`. Otherwise, the output data type is `logical`.

Data Types: `logical` | `double`

**cinit — Initialization value for PRBS generator**
numeric scalar

Initialization value for PRBS generator, returned as a numeric scalar.

Data Types: `uint32`

# Version History

**Introduced in R2016b**

## See Also

`ltePSBCH` | `ltePSBCHIndices` | `ltePSBCHDecode` | `ltePRBS` | `ltePBCHPRBS`

# ltePSCCH

Physical sidelink control channel

## Syntax

```
sym = ltePSCCH(cw)
```

## Description

`sym = ltePSCCH(cw)` returns a column vector containing the physical sidelink control channel (PSCCH) complex symbols for the input codeword bits. Channel processing performed by the function includes PSCCH-specific scrambling, QPSK modulation, and SC-FDMA transform precoding, as defined in TS 36.211 [1], Section 9.4.

For more information, see “Physical Sidelink Control Channel Processing” on page 2-736.

## Examples

### Generate SCI Message on PSCCH

Create a codeword using an encoded SCI message payload and process the bits on the PSCCH.

Create a UE settings structure and use it to generate SCI message bits. Produce an encoded SCI message codeword.

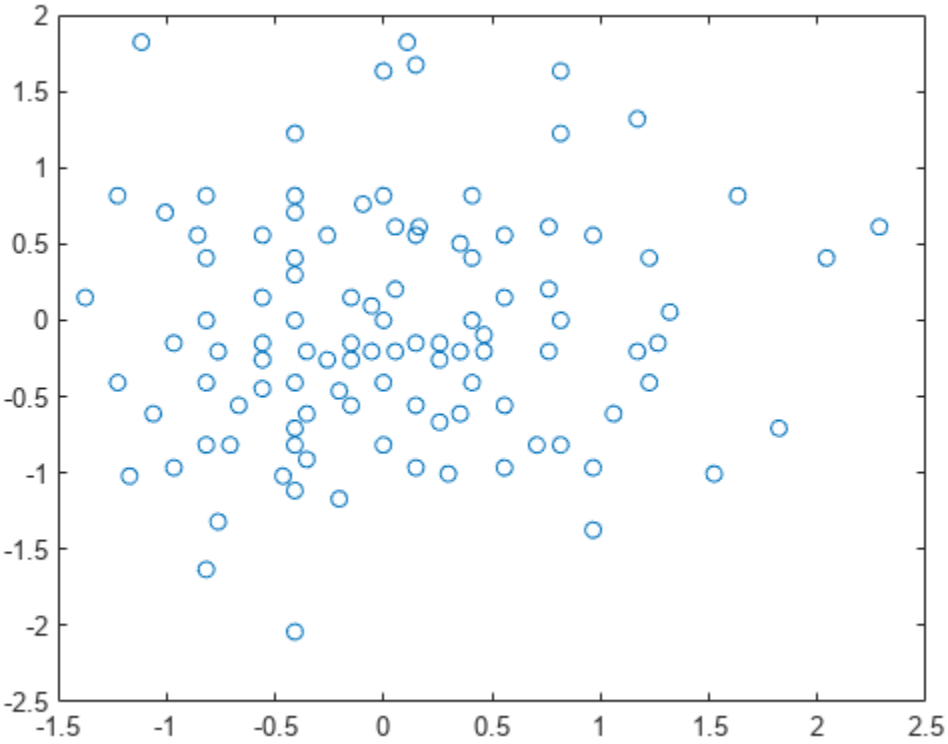
```
ue = struct('NSLRB',50,'CyclicPrefixSL','Normal');
[~,scibits] = lteSCI(ue);

cw = lteSCIEncode(ue,scibits);
```

Generate PSCCH symbols. View the length of the symbol column vector. Plot the constellation to show the effect of the SC-FDMA precoding on the modulation symbols.

```
symbols = ltePSCCH(cw);
numSymbols = size(symbols)
```

```
numSymbols = 1×2

   144     1
```

```
plot(symbols,'o')
```




## Input Arguments

### `cw` — PSCCH codeword

vector

PSCCH codeword, specified as an $M_{bit}$-by-1 integer vector. $M_{bit}$ is the number of bits transmitted on the physical sidelink control channel in one subframe and must be a multiple of 12. For more information, see “Physical Sidelink Control Channel Processing” on page 2-736.

Data Types: `double` | `int8`

## Output Arguments

### `sym` — Modulated PSCCH symbols

column vector

Modulated PSCCH symbols, returned as an $N_{RE}$-by-1 column vector. $N_{RE}$ is number of PSCCH resource elements in a subframe. For more information, see “Physical Sidelink Control Channel Processing” on page 2-736.

## More About

### Physical Sidelink Control Channel Processing

Physical sidelink control channel (PSCCH) processing includes PSCCH-specific scrambling, QPSK modulation, and SC-FDMA transform precoding. PSCCH processing follows the processing steps used for PUSCH, with variations defined in TS 36.211, Section 9.4.

For PSCCH, the input codeword length is $M_{\text{bits}} = N_{\text{RE}} \times N_{\text{bps}}$, where $N_{\text{RE}}$ is the number of PSCCH resource elements in a subframe and $N_{\text{bps}}$ is the number of bits per symbol. Because the PSCCH is QPSK modulated, there are 2 bits per symbol. Nominally, the codeword length for PSCCH is 288 bits for D2D normal cyclic prefix, 240 bits for D2D extended cyclic prefix and 480 for V2X. Nominally, $N_{\text{RE}}$ is 144 for D2D normal cyclic prefix or 120 for D2D extended cyclic prefix. For V2X, it is 240 defined for normal cyclic prefix only. Specifically, $N_{\text{RE}} = N_{\text{PRB}} \times N_{\text{REperPRB}} \times N_{\text{SYM}}$ and includes symbols associated with the sidelink SC-FDMA guard symbol.

- $N_{\text{PRB}}$ is the number of physical resource blocks (PRB) used for transmission. PSCCH is transmitted on a single PRB.
- $N_{\text{REperPRB}}$ is the number of resource elements in a PRB. Each PRB has 12 resource elements.
- $N_{\text{SYM}}$ is the number of SC-FDMA symbols in a PSCCH subframe, including symbols associated with the sidelink SC-FDMA guard symbol. The number of SC-FDMA symbols is a PSCCH subframe is 12 for D2D normal cyclic prefix or 10 for D2D extended cyclic prefix and V2X.

For D2D sidelink, when an SCI message is sent as a sidelink shared grant, it is transmitted twice on two separate PSCCH instances within the associated PSCCH resource pool. For V2X, only a single instance of PSCCH is transmitted for each scheduling grant.

### Physical Sidelink Control Channel Indexing

Use the `ltePSCCHIndices` indexing function and the corresponding `ltePSCCH` sequence function to populate the PSCCH subframe resource grid. The PSCCH is transmitted in the available SC-FDMA symbols in a PSCCH subframe, using a single layer representing antenna port 1000. It excludes each symbol per slot assigned to PSCCH DM-RS. For more information on PSCCH DM-RS, see the `ltePSCCHDRSIndices` function.

The indices are ordered as the PSCCH QPSK modulation symbols should be mapped, applying frequency-first mapping. The resource elements in the last SC-FDMA symbol within a subframe are counted in the mapping process but should not be transmitted. The sidelink-specific SC-FDMA modulation creates this guard symbol.

For more information on mapping symbols to the resource element grid, see “Resource Grid Indexing”.

# Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePSCCHDecode` | `ltePSCCHIndices` | `ltePSCCHDRS` | `ltePSCCHDRSIndices` | `ltePSCCHPRBS` | `lteULPrecode` | `lteSCI`

# ltePSCCHDecode

PSCCH decoding

## Syntax

```
softbits = ltePSCCHDecode(sym)
[softbits,symbols] = ltePSCCHDecode(sym)
```

## Description

`softbits = ltePSCCHDecode(sym)` returns a vector of log-likelihood ratio (LLR) soft bits for the input modulated PSCCH symbols.

The PSCCH decoder performs SC-FDMA transform deprecoding, QPSK demodulation, and PSCCH-specific descrambling. These operations are the inverse of the `ltePSCCH` function processing, as defined in TS 36.211 [1], Section 9.4. For more information, see “Physical Sidelink Control Channel Processing” on page 2-741.

`[softbits,symbols] = ltePSCCHDecode(sym)` also returns the intermediate QPSK modulation symbols.

## Examples

### Decode PSCCH Symbols

Decode PSCCH symbols that contain a fully encoded SCI format 0 message with noise added. After PSCCH demodulation, decode and recover the SCI message structure.

Create UE settings and SCI message configuration structures. Generate a PSCCH transmission. Add noise to the symbols.

```
ue = struct('NSLRB',50,'CyclicPrefixSL','Normal');
sci0 = struct('FreqHopping',1,'ModCoding',3);

[sci0,scibits] = lteSCI(ue,sci0);
cw = lteSCIEncode(ue,scibits);
sym = ltePSCCH(cw);

rxsym = sym + 0.1*randn(size(sym));
```

Decode the PSCCH symbols and SCI message. View the SCI message structure settings. Confirm that the transmitted and recovered SCI messages match.

```
[rxsoftbits,sym] = ltePSCCHDecode(rxsym);
[rxinfo,rxerr] = lteSCIDecode(ue,rxsoftbits);

[recsci0,recscibits] = lteSCI(ue,rxinfo);
recsci0
```

```
recsci0 = struct with fields:
              SCIFormat: 'Format0'
```

```
            FreqHopping: 1
             Allocation: [1x1 struct]
    TimeResourcePattern: 0
              ModCoding: 3
            TimeAdvance: 0
                  NSAID: 0
```

```
isequal(scibits,recscibits)
```

```
ans = logical
   1
```

### Plot Decoded PSCCH Symbols

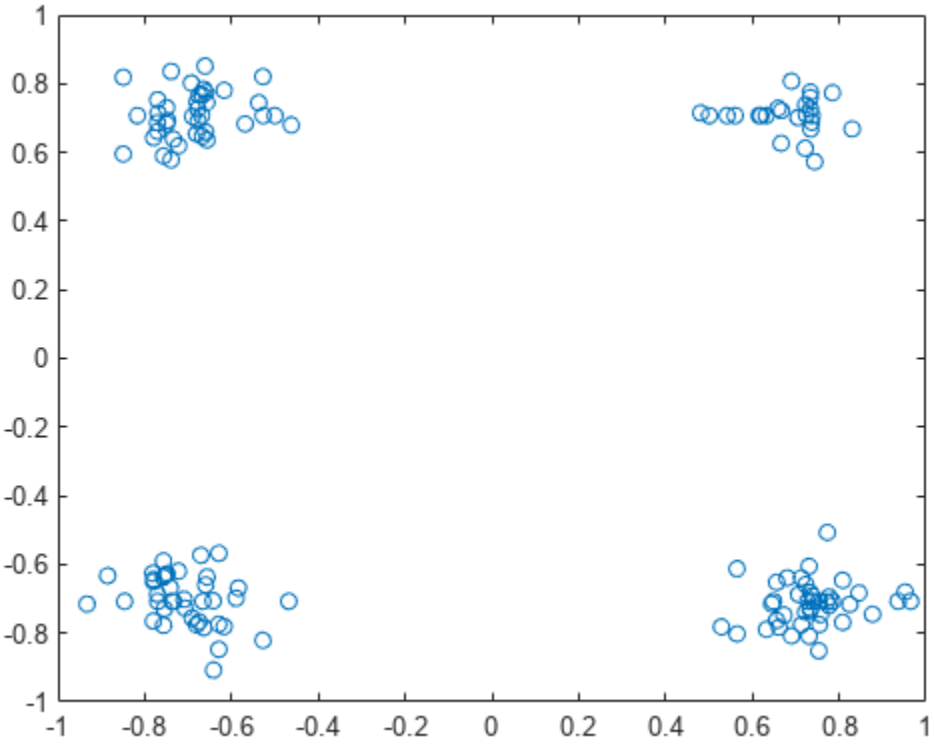
Decode PSCCH symbols that contain a fully encoded SCI format 0 message with noise added. After PSCCH demodulation, plot the intermediate QPSK modulated symbols.

Create UE settings and SCI message configuration structures. Generate a PSCCH transmission. Add noise to the symbols.

```
ue = struct('NSLRB',50,'CyclicPrefixSL','Normal');
sci0 = struct('FreqHopping',1,'ModCoding',3);

[sci0,scibits] = lteSCI(ue,sci0);
cw = lteSCIEncode(ue,scibits);
sym = ltePSCCH(cw);

rxsym = sym + 0.1*randn(size(sym));
```

Decode the PSCCH symbols and plot the output intermediate QPSK modulated symbols.

```
[rxsoftbits,symbols] = ltePSCCHDecode(rxsym);
plot(symbols,'o')
```




## Input Arguments

### `sym` — Modulated PSCCH symbols

column vector

Modulated PSCCH symbols, specified as an $N_{RE}$-by-1 column vector. $N_{RE}$ is the number of resource elements in a PSCCH subframe, including the SC-FDMA guard symbol. For D2D sidelink, nominally $N_{RE}$ is 144 or 120 for normal and extended cyclic prefix respectively. For V2X sidelink, nominally $N_{RE}$ is 240 bits, defined for normal cyclic prefix only. For more information, see “Physical Sidelink Control Channel Processing” on page 2-741.

Data Types: `double`
Complex Number Support: Yes

## Output Arguments

### `softbits` — Log-likelihood ratio soft bits

vector

Log-likelihood ratio (LLR) soft bits, returned as a $(2 \times N_{RE})$-by-1 vector. $N_{RE}$ is the number of resource elements in a PSCCH subframe, including the SC-FDMA guard symbol. The LLR of the punctured soft bits associated with the last SC-FDMA symbol in the subframe are set to 0. For more information, see “Physical Sidelink Control Channel Processing” on page 2-741.

**`symbols` — Modulated PSCCH symbols**
column vector of complex numbers

Modulated PSCCH symbols, returned as an $N_{RE}$-by-1 column vector. $N_{RE}$ is the number of resource elements in a PSCCH subframe, including the SC-FDMA guard symbol. For more information, see “Physical Sidelink Control Channel Processing” on page 2-741.

## More About

### Physical Sidelink Control Channel Processing

Physical sidelink control channel (PSCCH) processing includes PSCCH-specific scrambling, QPSK modulation, and SC-FDMA transform precoding. PSCCH processing follows the processing steps used for PUSCH, with variations defined in TS 36.211, Section 9.4.

For PSCCH, the input codeword length is $M_{bits} = N_{RE} \times N_{bps}$, where $N_{RE}$ is the number of PSCCH resource elements in a subframe and $N_{bps}$ is the number of bits per symbol. Because the PSCCH is QPSK modulated, there are 2 bits per symbol. Nominally, the codeword length for PSCCH is 288 bits for D2D normal cyclic prefix, 240 bits for D2D extended cyclic prefix and 480 for V2X. Nominally, $N_{RE}$ is 144 for D2D normal cyclic prefix or 120 for D2D extended cyclic prefix. For V2X, it is 240 defined for normal cyclic prefix only. Specifically, $N_{RE} = N_{PRB} \times N_{REperPRB} \times N_{SYM}$ and includes symbols associated with the sidelink SC-FDMA guard symbol.

- $N_{PRB}$ is the number of physical resource blocks (PRB) used for transmission. PSCCH is transmitted on a single PRB.
- $N_{REperPRB}$ is the number of resource elements in a PRB. Each PRB has 12 resource elements.
- $N_{SYM}$ is the number of SC-FDMA symbols in a PSCCH subframe, including symbols associated with the sidelink SC-FDMA guard symbol. The number of SC-FDMA symbols is a PSCCH subframe is 12 for D2D normal cyclic prefix or 10 for D2D extended cyclic prefix and V2X.

For D2D sidelink, when an SCI message is sent as a sidelink shared grant, it is transmitted twice on two separate PSCCH instances within the associated PSCCH resource pool. For V2X, only a single instance of PSCCH is transmitted for each scheduling grant.

# Version History
**Introduced in R2016b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also
`ltePSCCH` | `ltePSCCHIndices` | `ltePSCCHDRS` | `ltePSCCHDRSIndices` | `ltePSCCHPRBS`

# ltePSCCHDRS

PSCCH demodulation reference signal

## Syntax

```
[seq,info] = ltePSCCHDRS
[seq,info] = ltePSCCHDRS(ue)
```

## Description

`[seq,info] = ltePSCCHDRS` returns a 24-by-1 complex column vector sequence containing PSCCH demodulation reference signal (DM-RS) values and an associated information structure. For more information, see “PSCCH Demodulation Reference Signal Processing” on page 2-745.

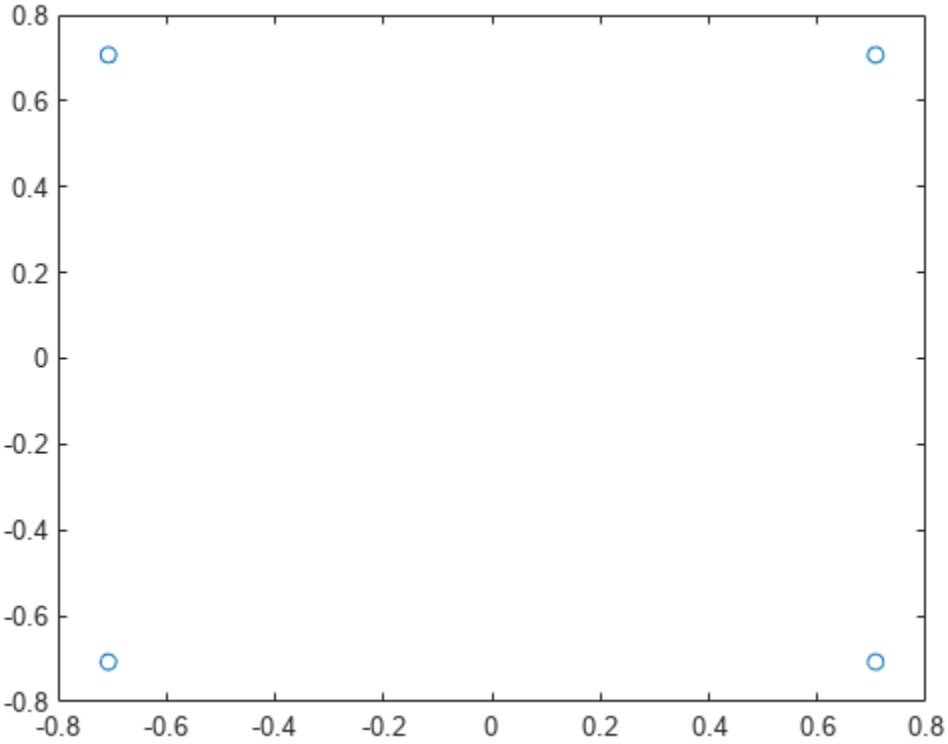
`[seq,info] = ltePSCCHDRS(ue)` returns a vector of DM-RS values for either D2D or V2X sidelink given the specified UE settings structure. For more information, see “PSCCH Demodulation Reference Signal Processing” on page 2-745.

## Examples

### Generate PSCCH DM-RS Sequence

Generate a PSCCH DM-RS sequence associated with both DM-RS SC-FDMA symbols in a subframe. Plot the constellation of the sequence, which is QPSK modulated.

```
[pscchDrsSeq,info] = ltePSCCHDRS;
plot(pscchDrsSeq,'o')
```




### Generate PSCCH DM-RS sequence for V2X sidelink

Generate the PSCCH DM-RS sequence associated with the four DM-RS SC-FDMA symbols of a PRB pair for a PSCCH transmission and V2X sidelink.

Create a user equipment settings structure for the V2X sidelink mode with PRB set indices of 0 and 1 and a zero cyclic shift.

```
ue = struct('SidelinkMode','V2X');
ue.PRBSet = [0 1]';
ue.CyclicShift = 0;
```

Generate a PSCCH DM-RS sequence associated with both DM-RS SC-FDMA symbols in a subframe. The output sequence is a 96-length vector to be mapped onto the 24 subcarriers in each of the pair of DM-RS SC-FDMA symbols per slot for two consecutive resource blocks.

```
[pscchDrsSeq,info] = ltePSCCHDRS(ue);
size(pscchDrsSeq)
```

```
ans = 1×2

    96     1
```

## Input Arguments

### `ue` — User equipment settings

structure

User equipment settings, specified as a parameter structure containing these fields:

#### `SidelinkMode` — Sidelink mode

`'D2D'` (default) | `'V2X'` | optional

Sidelink mode, specified as `'D2D'` or `'V2X'`.

Data Types: `char` | `string`

#### `PRBSet` — Zero-based physical resource block index

integer | integer column vector | two-column integer matrix

Zero-based physical resource block (PRB) index, specified as an integer, an integer column vector, or a two-column integer matrix.

For D2D sidelink, the PSCCH is intended to be transmitted in a single PRB in a subframe and therefore, specifying `PRBSet` as a scalar PRB index is recommended. For V2X sidelink, the PSCCH is intended to be transmitted in a pair of consecutive PRB in a subframe, therefore `PRBSet` must be a column vector containing two consecutive indices. However, for a more general nonstandard multi-PRB allocation, `PRBSet` can be a set of indices specified as an integer column vector or as a two-column integer matrix corresponding to slot-wise resource allocations for PSCCH.

Data Types: `double`

#### `CyclicShift` — Cyclic shift for DM-RS

0 (default) | 3 | 6 | 9

Cyclic shift for DM-RS, specified as `0`, `3`, `6` or `9`. The function uses this input only for V2X sidelink.

Data Types: `double`

Data Types: `struct`

## Output Arguments

### `seq` — PSCCH DM-RS values

column vector

PSCCH DM-RS values, returned as a complex column vector. For more information, see “PSCCH Demodulation Reference Signal Processing” on page 2-745.

Data Types: `double`

### `info` — PSCCH DM-RS information

structure

PSCCH DM-RS information about the intermediate variables used to create the DM-RS, returned as a parameter structure containing these fields:

#### `Alpha` — Reference signal cyclic shift for each slot

two-column vector

Reference signal cyclic shift for each slot, returned as a two-column vector. ($\alpha$)

`Alpha` is proportional to `NCS`, where $\alpha = \frac{2\pi n_{cs,\lambda}}{12}$.

**`SeqGroup` — Base sequence group number for each slot**
two-column vector

Base sequence group number for each slot, returned as a two-column vector. ($u$)

**`SeqIdx` — Base sequence number for each slot**
two-column vector

Base sequence number for each slot, returned as a two-column vector. ($v$)

**`RootSeq` — Root Zadoff-Chu sequence index for each slot**
two-column vector

Root Zadoff-Chu sequence index for each slot, returned as a two-column vector. ($q$)

**`NCS` — Cyclic shift values for each slot**
two-column vector

Cyclic shift values for each slot, returned as a two-column vector. ($n_{\mathrm{cs},\lambda}$)

**`NZC` — Zadoff-Chu sequence length**
integer

Zadoff-Chu sequence length, returned as an integer. ($N_{ZC}^{RS}$)

**`OrthSeq` — Orthogonal cover value for each slot**
matrix

Orthogonal cover value for each slot, returned as a matrix. ($\overline{w}$)

Data Types: `struct`

## More About

### PSCCH Demodulation Reference Signal Processing

The PSCCH demodulation reference signal (DM-RS) sequence is transmitted alongside the `ltePSCCH` values using the two SC-FDMA symbols allocated to DM-RS in a PSCCH subframe. By default, the output vector is the repetition of a 12-element sequence and specified in TS 36.211, Section 9.8. The output vector is mapped onto the 12 subcarriers of the DM-RS SC-FDMA symbol in each slot of the single PSCCH physical resource block (PRB) transmission on antenna port 1000. For a V2X configured PSCCH, the output will be a 96-by-1 vector to be mapped onto the 24 subcarriers in each of the pair of DRS SC-FDMA symbols per slot for two consecutive resource blocks.

The single-PRB PSCCH DM-RS is transmitted using a short base QPSK reference sequence instead of the Zadoff-Chu sequence that is normally used for reference signals. Because the Zadoff-Chu sequence is not used, the `RootSeq` and `NZC` fields are set to –1 in the `info` structure returned by `ltePSCCHDRS`.

#### PSCCH Demodulation Reference Signal Indexing

Use the indexing function, `ltePSCCHDRSIndices`, and the corresponding sequence function, `ltePSCCHDRS`, to populate the resource grid for any PSCCH subframe. The PSCCH DM-RS is transmitted in the available SC-FDMA symbols in a PSCCH subframe, using a single layer on antenna port 1000.

The indices are ordered as the PSCCH DM-RS QPSK modulation symbols should be, applying frequency-first mapping. One-based linear indexing is the default return format but you can also generate alternative indexing formats by using the `opts` input.

The resource elements in the last SC-FDMA symbol within a subframe are counted in the mapping process but should not be transmitted. The sidelink-specific SC-FDMA modulation creates the last symbol, which serves as a guard symbol.

For D2D sidelink, in zero-based indexing, the SC-FDMA symbol indices used are {3,10} for normal cyclic prefix and {2,8} for extended cyclic prefix. The same symbols are used by the `ltePUSCHDRSIndices` function. For V2X sidelink, there are four DM-RS SC-FDMA symbols with indices {2,5,8,11}, defined for normal cyclic prefix only.

**Note** The indicated symbol indices are based on TS 36.211, Section 9.8. However to align with the LTE Toolbox subframe orientation, these indices are expanded from symbol index per slot to symbol index per subframe.

For more information on mapping symbols to the resource element grid, see “Resource Grid Indexing”.

## Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePSCCHDRSIndices` | `ltePSCCH` | `ltePSCCHDecode` | `ltePSCCHIndices`

# ltePSCCHDRSIndices

PSCCH DM-RS resource element indices

## Syntax

```
ind = ltePSCCHDRSIndices(ue)
ind = ltePSCCHDRSIndices(ue,opts)
```

## Description

`ind = ltePSCCHDRSIndices(ue)` returns a column vector of PSCCH demodulation reference signal (DM-RS) resource element indices for the specified UE settings structure. For more information, see “PSCCH Demodulation Reference Signal Indexing” on page 2-751.

`ind = ltePSCCHDRSIndices(ue,opts)` formats the returned indices using options specified by `opts`.

## Examples

### Create PSCCH DM-RS Values

Write the complex PSCCH DM-RS values into the PSCCH DM-RS resource elements in a PSCCH subframe for both D2D normal cyclic prefix and V2X. Display an image of their locations to compare both sidelink modes.

Create a user equipment settings structure and an empty resource grid subframe for 10 MHz bandwidth and D2D normal cyclic prefix. Assign a PRB set index of 5.

```
ue.NSLRB = 50;
ue.CyclicPrefixSL = 'Normal';
ue.NSLID = 1;
subframe_D2D = lteSLResourceGrid(ue);
ue.PRBSet = 5;
```

Generate PSCCH DM-RS indices and write PSCCH DM-RS values into `subframe`.

```
pscchdrs_indices = ltePSCCHDRSIndices(ue);
subframe_D2D(pscchdrs_indices) = ltePSCCHDRS();
```

Change user equipment settings to V2X sidelink mode. Assign a PRB set indices of 5 and 6.

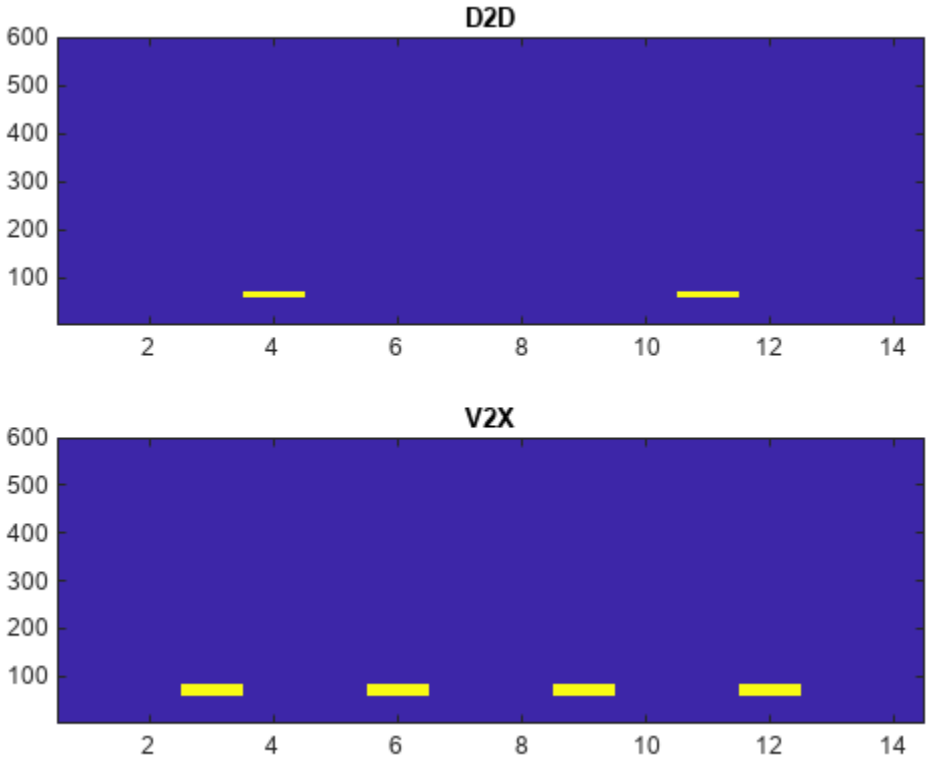
```
ue.SidelinkMode = 'V2X';
subframe_V2X = lteSLResourceGrid(ue);
ue.PRBSet = [5 6]';
```

Generate PSCCH DM-RS indices and write PSCCH DM-RS values into `subframe`.

```
pscchdrs_indices = ltePSCCHDRSIndices(ue);
subframe_V2X(pscchdrs_indices) = ltePSCCHDRS(ue);
```

Display the PSCCH DM-RS locations for both sidelink modes.

```
subplot(2,1,1);
imagesc(100*abs(subframe_D2D))
axis xy
title('D2D');
subplot(2,1,2);
imagesc(100*abs(subframe_V2X));
axis xy;
title(ue.SidelinkMode)
```




### Compare PSCCH DM-RS Resource Element Indexing

Compare PSCCH DM-RS resource element indexing formats.

Create a UE settings structure.

```
ue = struct('NSLRB',15,'CyclicPrefixSL','Normal','PRBSet',5);
```

#### One-based linear indexing, this is the default output style

Generate PSCCH DM-RS indices, using the default one-based linear indexing style.

```
ind1 = ltePSCCHDRSIndices(ue);
ind1(1)
```

```
ans = uint32
    601
```

### Zero-based linear indexing

Generate PSCCH DM-RS indices, using zero-based linear indexing style.

```
opts = '0based';
ind0 = ltePSCCHDRSIndices(ue,opts);
ind0(1)
```

```
ans = uint32
    600
```

For zero-based indexing, the first assigned index is one lower than the one-based indexing.

### One-based indexing in `[subcarrier,symbol,port]` subscript row style

Generate PSCCH DM-RS indices, one-based subscript row style.

```
opts = {'sub','1based'};
ind1sub = ltePSCCHDRSIndices(ue,opts);
size(ind1sub)
```

```
ans = 1×2

    24     3
```

```
ind1sub(1,:)
```

```
ans = 1x3 uint32 row vector

   61    4    1
```

The subscript row style outputs a 24-by-3 matrix. Viewing the first row you can see symbol number 4 is occupied.

Two PSCCH subframe symbols are reserved for transmission of the PSCCH DM-RS. Inspecting the output matrix for unique symbol values, shows that the PSCCH DM-RS occupy symbols 4 and 11 for one-based indexing.

```
unique(ind1sub(:,2,:))
```

```
ans = 2x1 uint32 column vector

    4
   11
```

## Input Arguments

### ue — User equipment settings

structure

User equipment settings, specified as a parameter structure containing these fields:

#### `SidelinkMode` — Sidelink mode

`'D2D'` (default) | `'V2X'` | optional

Sidelink mode, specified as `'D2D'` or `'V2X'`.

Data Types: `char` | `string`

**`NSLRB` — Number of sidelink resource blocks**
integer scalar from 6 to 110

Number of sidelink resource blocks, specified as an integer scalar from 6 to 110.

Example: `6`, which corresponds to a channel bandwidth of 1.4 MHz.

Data Types: `double`

**`CyclicPrefixSL` — Cyclic prefix length**
`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

**`PRBSet` — Zero-based physical resource block index**
integer | integer column vector | two-column integer matrix

Zero-based physical resource block (PRB) index, specified as an integer, an integer column vector, or a two-column integer matrix.

For D2D sidelink, the PSCCH is intended to be transmitted in a single PRB in a subframe and therefore, specifying `PRBSet` as a scalar PRB index is recommended. For V2X sidelink, the PSCCH is intended to be transmitted in a pair of consecutive PRB in a subframe, therefore `PRBSet` must be a column vector containing two consecutive indices. However, for a more general nonstandard multi-PRB allocation, `PRBSet` can be a set of indices specified as an integer column vector or as a two-column integer matrix corresponding to slot-wise resource allocations for PSCCH.

Data Types: `double`

Data Types: `struct`

**`opts` — Output format options for resource element indices**
character vector | cell array of character vectors | string array

Output format options for resource element indices, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string) :

| Category | Options | Description |
|---|---|---|
| Indexing style | `'ind'` (default) | The returned indices are in linear index style. |
| | `'sub'` | The returned indices are in `[subcarrier,symbol,port]` subscript row style. |
| Index base | `'1based'` (default) | The returned indices are one-based. |
| | `'0based'` | The returned indices are zero-based. |

Example: `'ind 1based'`, `"ind 1based"`, `{'ind','1based'}`, or `["ind","1based"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `ind` — PSCCH DM-RS resource element indices

integer column vector | three-column integer matrix

PSCCH DM-RS resource element indices, returned as an integer column vector or a three-column integer matrix. For D2D sidelink, the returned vector or matrix has 24 PSCCH DM-RS resource element indices. For a V2X configured PSCCH, `ind` is a 96-by-1 vector with indices of the resource elements in the four DRS symbols in a subframe. For more information, see “PSCCH Demodulation Reference Signal Indexing” on page 2-751.

Data Types: `uint32`

## More About

### PSCCH Demodulation Reference Signal Indexing

Use the indexing function, `ltePSCCHDRSIndices`, and the corresponding sequence function, `ltePSCCHDRS`, to populate the resource grid for any PSCCH subframe. The PSCCH DM-RS is transmitted in the available SC-FDMA symbols in a PSCCH subframe, using a single layer on antenna port 1000.

The indices are ordered as the PSCCH DM-RS QPSK modulation symbols should be, applying frequency-first mapping. One-based linear indexing is the default return format but you can also generate alternative indexing formats by using the `opts` input.

The resource elements in the last SC-FDMA symbol within a subframe are counted in the mapping process but should not be transmitted. The sidelink-specific SC-FDMA modulation creates the last symbol, which serves as a guard symbol.

For D2D sidelink, in zero-based indexing, the SC-FDMA symbol indices used are {3,10} for normal cyclic prefix and {2,8} for extended cyclic prefix. The same symbols are used by the `ltePUSCHDRSIndices` function. For V2X sidelink, there are four DM-RS SC-FDMA symbols with indices {2,5,8,11}, defined for normal cyclic prefix only.

**Note** The indicated symbol indices are based on TS 36.211, Section 9.8. However to align with the LTE Toolbox subframe orientation, these indices are expanded from symbol index per slot to symbol index per subframe.

For more information on mapping symbols to the resource element grid, see “Resource Grid Indexing”.

### PSCCH Demodulation Reference Signal

The PSCCH demodulation reference signal (DM-RS) sequence is transmitted alongside the `ltePSCCH` values using the two SC-FDMA symbols allocated to DM-RS in a PSCCH subframe. By default, the output vector is the repetition of a 12-element sequence and specified in TS 36.211, Section 9.8. The output vector is mapped onto the 12 subcarriers of the DM-RS SC-FDMA symbol in each slot of the single PSCCH physical resource block (PRB) transmission on antenna port 1000. For a V2X

configured PSCCH, the output will be a 96-by-1 vector to be mapped onto the 24 subcarriers in each of the pair of DRS SC-FDMA symbols per slot for two consecutive resource blocks.

The single-PRB PSCCH DM-RS is transmitted using a short base QPSK reference sequence instead of the Zadoff-Chu sequence that is normally used for reference signals. Because the Zadoff-Chu sequence is not used, the `RootSeq` and `NZC` fields are set to –1 in the `info` structure returned by `ltePSCCHDRS`.

# Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePSCCHDRS` | `ltePSCCH` | `ltePSCCHDecode`

# ltePSCCHIndices

PSCCH resource element indices

## Syntax

```
[ind] = ltePSCCHIndices(ue)
[ind,info] = ltePSCCHIndices(ue)
[___] = ltePSCCHIndices(ue,opts)
```

## Description

`[ind] = ltePSCCHIndices(ue)` returns a column vector of physical sidelink control channel (PSCCH) resource element (RE) indices for the specified UE settings structure for either D2D or V2X sidelink. By default, the indices are returned in one-based linear indexing form. You can use this form to directly index elements of a matrix representing the subframe resource grid for antenna port 1000. For more information, see “Physical Sidelink Control Channel Indexing” on page 2-758.

`[ind,info] = ltePSCCHIndices(ue)` also returns a structure containing PSCCH-related information for the specified UE settings structure.

`[___] = ltePSCCHIndices(ue,opts)` formats the returned indices using options specified by `opts`. This syntax supports output options from prior syntaxes.

## Examples

### Map PSCCH Resource Elements

Write the complex PSCCH values into the PSCCH resource elements in a PSCCH subframe for D2D sidelink and with normal cyclic prefix. Do the same for V2X. Display an image of their locations and compare both sidelink modes. This mapping writes PSCCH values into the last SC-FDMA guard symbol within a subframe. The sidelink SC-FDMA modulator removes these values before transmission of the waveform.

Create a UE settings structure for D2D sidelink and an empty sidelink resource grid. Assign a PRB set index of 5.

```
ue = struct('NSLRB',15,'CyclicPrefixSL','Normal');
subframe_D2D = lteSLResourceGrid(ue);
ue.PRBSet = 5;
```

Generate PSCCH indices. Populate the PSCCH resource elements in the subframe. For D2D normal cyclic prefix, a PSCCH subframe contains 144 REs.

```
[pscch_indices, pscch_info] = ltePSCCHIndices(ue);
subframe_D2D(pscch_indices) = ltePSCCH(zeros(pscch_info.G,1));
```

Change user equipment settings to V2X sidelink mode. Assign a PRB set indices of 5 and 6.

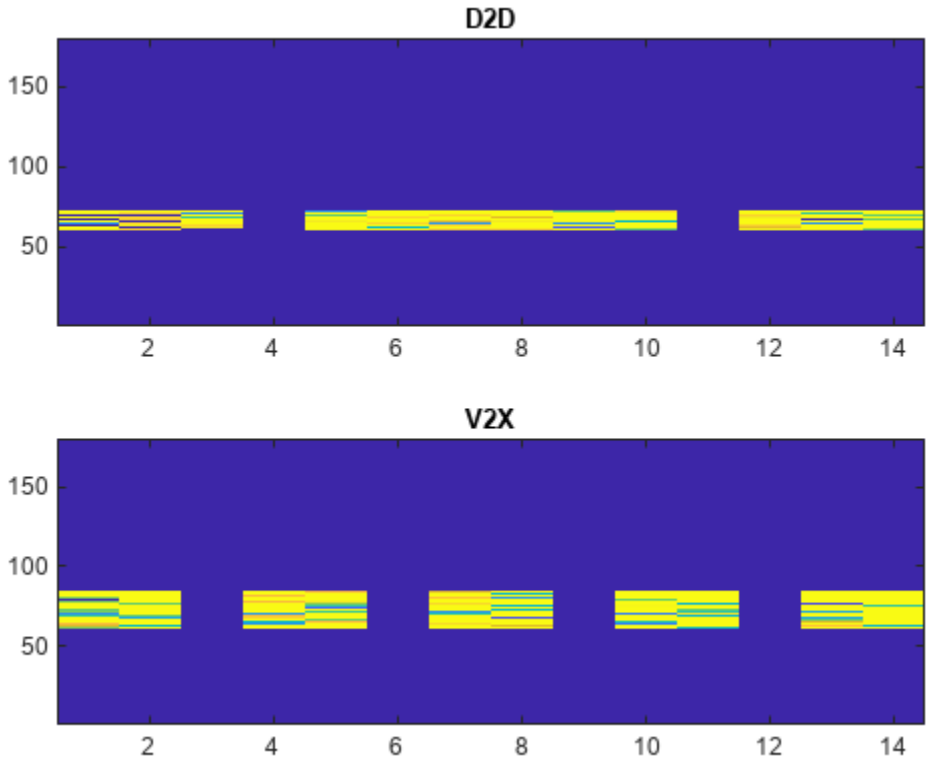
```
ue.SidelinkMode = 'V2X';
subframe_V2X = lteSLResourceGrid(ue);
ue.PRBSet = [5;6];
```

Generate PSCCH indices. Populate the PSCCH resource elements in the subframe using a codeword filled with zeros. For V2X, a PSCCH subframe contains 240 REs.

```
[pscch_indices, pscch_info] = ltePSCCHIndices(ue);
subframe_V2X(pscch_indices) = ltePSCCH(zeros(pscch_info.G,1));
```

View the resource grid and compare the indices for both sidelink modes.

```
subplot(2,1,1);
image(400*abs(subframe_D2D));
axis xy; title('D2D');
subplot(2,1,2);
image(400*abs(subframe_V2X));
axis xy; title(ue.SidelinkMode);
```




### View PSCCH Information Structure

View the information structure output by the PSCCH resource element indexing function.

Create a UE settings structure.

```
ue = struct('NSLRB',15,'CyclicPrefixSL','Normal','PRBSet',5);
```

Generate PSCCH indices and the information structure. View the information structure.

```
[pscch_indices,info] = ltePSCCHIndices(ue);
info
```

```
info = struct with fields:
     G: 288
    Gd: 144
```

### Compare PSCCH Resource Element Indexing

Compare PSCCH resource element indexing formats. Options include one-based or zero-based indices in linear or subscript row indexing style.

Create a UE settings structure.

```
ue = struct('NSLRB',15,'CyclicPrefixSL','Normal','PRBSet',5);
```

#### One-based linear indexing, this is the default output style

Generate PSCCH indices using the default one-based linear indexing.

```
pscch1ind = ltePSCCHIndices(ue);
pscch1ind(1)
```

```
ans = uint32
    61
```

#### Zero-based linear indexing

Generate PSCCH indices using zero-based linear indexing.

```
opts = '0based';
pscch0ind = ltePSCCHIndices(ue,opts);
pscch0ind(1)
```

```
ans = uint32
    60
```

For zero-based indexing, the first assigned index is one lower than the one-based indexing.

#### One-based indices in `[subcarrier,symbol,port]` subscript row style

Generate PSCCH indices using one-based subscript row style.

```
opts = {'sub','1based'};
pscch1sub = ltePSCCHIndices(ue,opts);
pscch1sub(1,:)
```

```
ans = 1x3 uint32 row vector

   61    1    1
```

The subscript row style outputs a 24-by-3 matrix. Viewing the first row you see from the second column value that symbol number 1 is occupied.

Inspecting the output matrix for unique symbol values, shows the symbols 4 and 11 are not occupied by PSCCH. Two PSCCH subframe symbols are reserved for transmission of PSCCH DM-RS. When one-based indexing is specified, symbols 4 and 11 transmit the PSCCH DM-RS.

```
unique(pscch1sub(:,2,:))
```

```
ans = 12x1 uint32 column vector

    1
    2
    3
    5
    6
    7
    8
    9
   10
   12
    ⋮
```

## Input Arguments

### ue — User equipment settings

structure

User equipment settings, specified as a parameter structure containing these fields:

### SidelinkMode — Sidelink mode

`'D2D'` (default) | `'V2X'` | optional

Sidelink mode, specified as `'D2D'` or `'V2X'`.

Data Types: `char` | `string`

### NSLRB — Number of sidelink resource blocks

integer scalar from 6 to 110

Number of sidelink resource blocks, specified as an integer scalar from 6 to 110.

Example: 6, which corresponds to a channel bandwidth of 1.4 MHz.

Data Types: `double`

### CyclicPrefixSL — Cyclic prefix length

`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

### PRBSet — Zero-based physical resource block index

integer | integer column vector | two-column integer matrix

Zero-based physical resource block (PRB) index, specified as an integer, an integer column vector, or a two-column integer matrix.

For D2D sidelink, the PSCCH is intended to be transmitted in a single PRB in a subframe and therefore, specifying `PRBSet` as a scalar PRB index is recommended. For V2X sidelink, the PSCCH is intended to be transmitted in a pair of consecutive PRB in a subframe, therefore `PRBSet` must be a column vector containing two consecutive indices. However, for a more general nonstandard multi-PRB allocation, `PRBSet` can be a set of indices specified as an integer column vector or as a two-column integer matrix corresponding to slot-wise resource allocations for PSCCH.

Data Types: `double`

Data Types: `struct`

#### `opts` — Output format options for resource element indices

character vector | cell array of character vectors | string array

Output format options for resource element indices, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string) :

| Category | Options | Description |
|---|---|---|
| Indexing style | `'ind'` (default) | The returned indices are in linear index style. |
| | `'sub'` | The returned indices are in `[subcarrier,symbol,port]` subscript row style. |
| Index base | `'1based'` (default) | The returned indices are one-based. |
| | `'0based'` | The returned indices are zero-based. |

Example: `'ind 1based'`, `"ind 1based"`, `{'ind','1based'}`, or `["ind","1based"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

#### `ind` — PSCCH resource element indices

integer column vector | three-column integer matrix

PSCCH resource element indices, returned as an integer column vector or a three-column integer matrix. For D2D sidelink, the returned vector has 144 PSCCH resource element indices for normal cyclic prefix or 120 PSCCH resource element indices for extended cyclic prefix. For V2X, the nominal output is a 240-length column vector and it is defined for normal cyclic prefix only. For more information, see “Physical Sidelink Control Channel Indexing” on page 2-758.

Data Types: `uint32`

#### `info` — PSCCH subframe resource information

structure

PSCCH subframe resource information, returned as a parameter structure containing these fields:

**G — PSCCH bit capacity**
integer

PSCCH bit capacity, returned as an integer. For D2D sidelink, this value is 288 for normal cyclic prefix or 240 for extended cyclic prefix. For V2X, it is 480.

Data Types: `double`

**Gd — PSCCH QPSK symbol capacity**
integer

PSCCH QPSK symbol capacity, returned as an integer. For D2D sidelink, this value is 144 for normal cyclic prefix or 120 for extended cyclic prefix. For V2X, it is 240.

Data Types: `double`

Data Types: `struct`

## More About

### Physical Sidelink Control Channel Indexing

Use the `ltePSCCHIndices` indexing function and the corresponding `ltePSCCH` sequence function to populate the PSCCH subframe resource grid. The PSCCH is transmitted in the available SC-FDMA symbols in a PSCCH subframe, using a single layer representing antenna port 1000. It excludes each symbol per slot assigned to PSCCH DM-RS. For more information on PSCCH DM-RS, see the `ltePSCCHDRSIndices` function.

The indices are ordered as the PSCCH QPSK modulation symbols should be mapped, applying frequency-first mapping. The resource elements in the last SC-FDMA symbol within a subframe are counted in the mapping process but should not be transmitted. The sidelink-specific SC-FDMA modulation creates this guard symbol.

For more information on mapping symbols to the resource element grid, see “Resource Grid Indexing”.

### Physical Sidelink Control Channel Processing

Physical sidelink control channel (PSCCH) processing includes PSCCH-specific scrambling, QPSK modulation, and SC-FDMA transform precoding. PSCCH processing follows the processing steps used for PUSCH, with variations defined in TS 36.211, Section 9.4.

For PSCCH, the input codeword length is $M_{\text{bits}} = N_{\text{RE}} \times N_{\text{bps}}$, where $N_{\text{RE}}$ is the number of PSCCH resource elements in a subframe and $N_{\text{bps}}$ is the number of bits per symbol. Because the PSCCH is QPSK modulated, there are 2 bits per symbol. Nominally, the codeword length for PSCCH is 288 bits for D2D normal cyclic prefix, 240 bits for D2D extended cyclic prefix and 480 for V2X. Nominally, $N_{\text{RE}}$ is 144 for D2D normal cyclic prefix or 120 for D2D extended cyclic prefix. For V2X, it is 240 defined for normal cyclic prefix only. Specifically, $N_{\text{RE}} = N_{\text{PRB}} \times N_{\text{REperPRB}} \times N_{\text{SYM}}$ and includes symbols associated with the sidelink SC-FDMA guard symbol.

- $N_{\text{PRB}}$ is the number of physical resource blocks (PRB) used for transmission. PSCCH is transmitted on a single PRB.
- $N_{\text{REperPRB}}$ is the number of resource elements in a PRB. Each PRB has 12 resource elements.

- $N_{SYM}$ is the number of SC-FDMA symbols in a PSCCH subframe, including symbols associated with the sidelink SC-FDMA guard symbol. The number of SC-FDMA symbols is a PSCCH subframe is 12 for D2D normal cyclic prefix or 10 for D2D extended cyclic prefix and V2X.

For D2D sidelink, when an SCI message is sent as a sidelink shared grant, it is transmitted twice on two separate PSCCH instances within the associated PSCCH resource pool. For V2X, only a single instance of PSCCH is transmitted for each scheduling grant.

# Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePSCCH` | `ltePSCCHDecode` | `ltePSCCHPRBS`

# ltePSCCHPRBS

PSCCH pseudorandom binary scrambling sequence

## Syntax

```
[seq,cinit] = ltePSCCHPRBS(n)
[seq,cinit] = ltePSCCHPRBS(n,mapping)

[subseq,cinit] = ltePSCCHPRBS(pn)
[subseq,cinit] = ltePSCCHPRBS(pn,mapping)
```

## Description

`[seq,cinit] = ltePSCCHPRBS(n)` returns a column vector containing the first `n` outputs of the PSCCH pseudorandom binary scrambling sequence (PRBS). It also returns an initialization value `cinit` for the PRBS generator.

The scrambling sequence generated should be applied to the coded PSCCH data carried by the associated subframe. The PRBS sequence generator used is initialized with $c_{\text{init}} = 510$.

`[seq,cinit] = ltePSCCHPRBS(n,mapping)` specifies the format of the returned sequence, `seq`, through the `mapping` input.

`[subseq,cinit] = ltePSCCHPRBS(pn)` returns a subsequence of a full PRBS sequence, specified by `pn`.

`[subseq,cinit] = ltePSCCHPRBS(pn,mapping)` specifies the format of the returned subsequence, `subseq`, through the `mapping` input.

## Examples

### Scramble PSCCH Codeword

Scramble a PSCCH codeword by generating the PSCCH pseudorandom binary sequence (PRBS) and applying an exclusive OR operation on the two sequences.

Generate the required length of the PRBS and scramble the PSCCH codeword with the PRBS sequence using `xor`.

```
codeword = ones(288,1);
pscchPrbs = ltePSCCHPRBS(length(codeword));
scrambled = xor(pscchPrbs,codeword);
```

### Descramble PSCCH Codeword

Descramble a received PSCCH codeword.

### Scramble PSCCH Codeword

- Generate the required length of the PRBS and scramble the PSCCH codeword with the PRBS sequence using `xor`.
- Modulate the logical scrambled data.

```
codeword = ones(288,1);
pscchPrbs = ltePSCCHPRBS(length(codeword));
scrambled = xor(pscchPrbs,codeword);

txsym = lteSymbolModulate(scrambled,'QPSK');
```

### Descramble Recovered Codeword

- Add noise to transmitted symbols and demodulate received soft data.
- Generate the PSCCH PRBS in signed form.
- Descramble a vector of noisy demodulated symbols representing a sequence of soft bits. To do so, perform a pointwise multiplication between the PRBS sequence and the recovered data.
- Compare the transmitted codeword to the recovered codeword.

```
sym = awgn(txsym,30,'measured');
softdata = lteSymbolDemodulate(sym,'QPSK');

scramblingSeq = ltePSCCHPRBS(length(softdata),'signed');
descrambled = softdata.*scramblingSeq;

isequal(codeword,descrambled > 0)
```

```
ans = logical
   1
```

The transmitted codeword matches the hard decision on the descrambled data.

## Input Arguments

### `n` — Number of elements in returned sequence

numeric scalar

Number of elements in returned sequence, `seq`, specified as a numeric scalar.

Data Types: `double`

### `pn` — Range of elements in returned subsequence

row vector

Range of elements in returned subsequence, `subseq`, specified as a row vector of `[p n]`. The subsequence returns `n` values of the PRBS generator, starting at position `p` (0-based).

Data Types: `double`

### `mapping` — Output sequence formatting

`'binary'` (default) | `'signed'`

Output sequence formatting, specified as `'binary'` or `'signed'`. `mapping` controls the format of the returned sequence.

- `'binary'` maps `true` to 1 and `false` to 0.
- `'signed'` maps `true` to –1 and `false` to 1.

Data Types: `char` | `string`

## Output Arguments

### `seq` — PSCCH pseudorandom scrambling sequence

logical column vector | numeric column vector

PSCCH pseudorandom scrambling sequence, returned as a logical column vector or a numeric column vector. `seq` contains the first `n` outputs of the physical control channel (PCCH) scrambling sequence. If you set `mapping` to `'signed'`, the output data type is `double`. Otherwise, the output data type is `logical`.

### `subseq` — PSCCH pseudorandom scrambling subsequence

logical column vector | numeric column vector

PSCCH pseudorandom scrambling sequence, returned as a logical column vector or a numeric column vector. `subseq` contains the values of the PRBS generator specified by `pn`. If you set `mapping` to `'signed'`, the output data type is `double`. Otherwise, the output data type is `logical`.

### `cinit` — Initialization value for PRBS generator

numeric scalar

Initialization value for PRBS generator, returned as a numeric scalar.

Data Types: `uint32`

# Version History

**Introduced in R2016b**

## See Also

`ltePSCCH` | `ltePSCCHIndices` | `ltePSCCHDecode` | `ltePRBS`

# ltePSSCH

Physical sidelink shared channel

## Syntax

```
sym = ltePSSCH(ue,cw)
```

## Description

`sym = ltePSSCH(ue,cw)` returns a complex symbol column vector containing the physical sidelink shared channel (PSSCH) for the specified UE settings structure and codeword bits. Channel processing performed by the function includes PSSCH-specific scrambling, QPSK or 16-QAM modulation, and SC-FDMA transform precoding, as defined in TS 36.211 [1], Section 9.3.

For more information, see “Physical Sidelink Shared Channel Processing” on page 2-766.

## Examples

### Create PSSCH Symbols

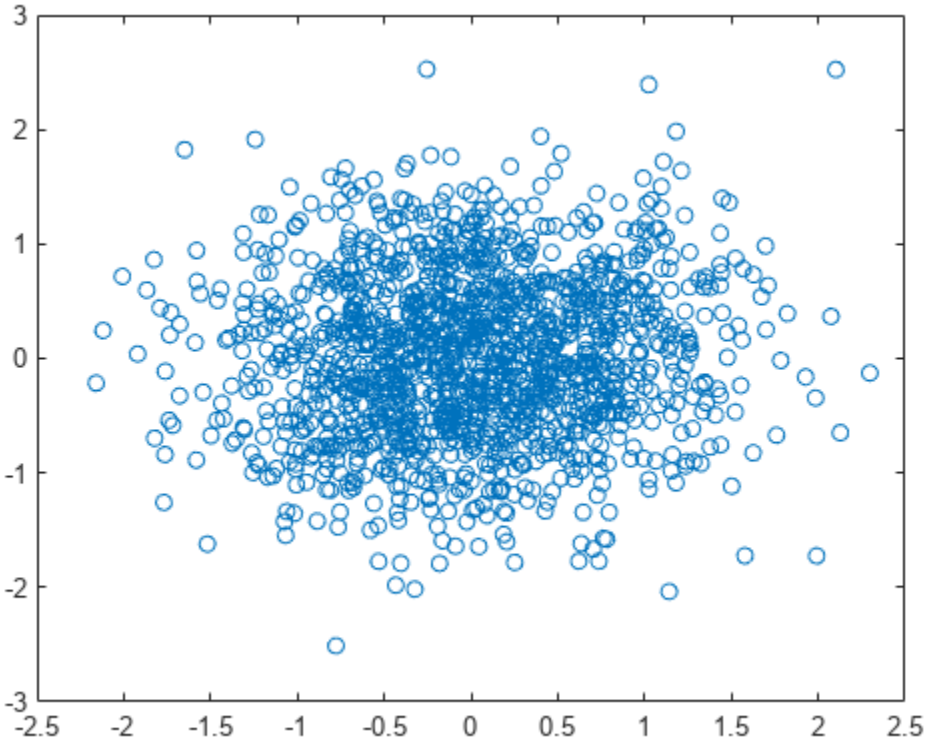
Create a codeword using the SL-SCH transport channel and encode the bits on the PSSCH.

Initialize a UE settings structure. Specify the codeword length to use for the SL-SCH. Choose a length that is a multiple of 12 symbols for normal cyclic prefix and has 4 bits per symbol for 16-QAM modulation. Pick a standard number of resource blocks, such as 10.

```
ue = struct('CyclicPrefixSL','Normal');
ue.RV = 0;
ue.Modulation = '16QAM';
ue.NSAID = 255;
ue.NSubframePSSCH = 0;
ue.SidelinkMode = 'D2D';

codewordlength = 5760; % (12 symbols)(4 bps)(12 REperRB)(10 PRB)
```

Create a codeword using the `lteSLSCH` function and encode the bits on the PSSCH. Plot the constellation to show the effects of the SC-FDMA precoding on the 16-QAM modulation symbols.

```
codeword = lteSLSCH(ue,codewordlength,zeros(100,1));
symbols = ltePSSCH(ue,codeword);

plot(symbols,'o')
```




## Input Arguments

### ue — User equipment settings

structure

User equipment settings, specified as a parameter structure containing these fields:

### SidelinkMode — Sidelink mode

`'D2D'` (default) | `'V2X'` | optional

Sidelink mode, specified as `'D2D'` or `'V2X'`.

Data Types: `char` | `string`

### CyclicPrefixSL — Cyclic prefix length

`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

### Modulation — Modulation type

`'QPSK'` | `'16QAM'`

Modulation type, specified as `'QPSK'` or `'16QAM'`.

Data Types: `char` | `string`

**`NSAID` — Sidelink group destination identity**
integer in the interval [0, 255]

Sidelink group destination identity, specified as an integer in the interval [0, 255].

This field is the lower eight bits of the full 24-bit ProSe Layer-2 group destination ID. This field and the `NSubframePSSCH` field control the value of the scrambling sequence at the start of each subframe. This field is required only for D2D sidelink.

Data Types: `double`

**`NXID` — V2X scrambling identity**
integer scalar

V2X scrambling identity, specified as an integer scalar. `NXID` is the 16 bit CRC associated with the PSCCH SCI grant. It is only required for V2X sidelink.

Data Types: `double`

**`NSubframePSSCH` — PSSCH subframe number**
integer scalar

PSSCH subframe number in the PSSCH subframe pool, specified as an integer scalar. ($n_{\mathrm{ssf}}^{\mathrm{PSSCH}}$)

`NSubframePSSCH` and `NSAID` control the values of the scrambling sequence. It is only required for D2D sidelink.

Data Types: `double`

Data Types: `struct`

**`cw` — PSSCH codeword**
integer vector

PSSCH codeword, specified as an $M_{\mathrm{bit}}$-by-1 integer vector. $M_{\mathrm{bit}}$ is the number of bits transmitted on the physical sidelink shared channel in one subframe and must be a multiple of 12. For more information, see “Physical Sidelink Shared Channel Processing” on page 2-766.

## Output Arguments

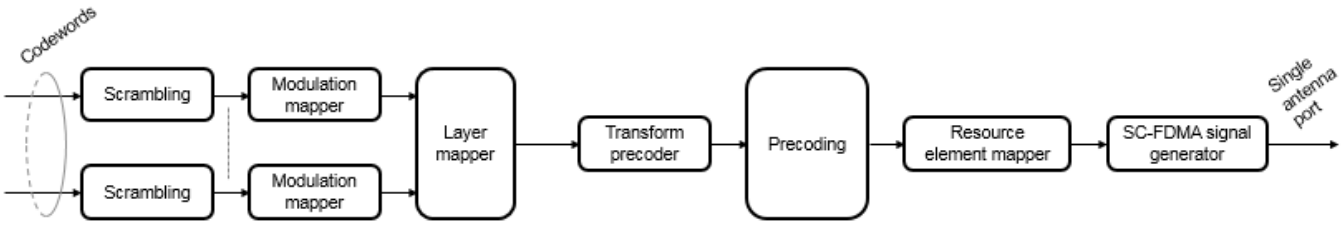
**`sym` — Modulated PSSCH symbols**
column vector

Modulated PSSCH symbols, returned as an $N_{\mathrm{RE}}$-by-1 column vector. $N_{\mathrm{RE}}$ is number of PSSCH resource elements in a subframe. For more information, see “Physical Sidelink Shared Channel Processing” on page 2-766.

## More About

### Physical Sidelink Shared Channel Processing

Physical sidelink shared channel (PSSCH) processing includes PSSCH-specific scrambling, QPSK or 16-QAM modulation, and SC-FDMA transform precoding. PSSCH processing follows the processing steps used for PUSCH, with variations defined in TS 36.211, Section 9.3.




For PSSCH, the input codeword length is $M_{\text{bits}} = N_{\text{RE}} \times N_{\text{bps}}$, where $N_{\text{bps}}$ is the number of bits per symbol. PSSCH modulation is either QPSK (2 bits per symbol) or 16 QAM (4 bits per symbol).

The number of PSSCH resource elements ($N_{\text{RE}}$) in a subframe is $N_{\text{RE}} = N_{\text{PRB}} \times N_{\text{REperPRB}} \times N_{\text{SYM}}$ and includes symbols associated with the sidelink SC-FDMA guard symbol.

- $N_{\text{PRB}}$ is the number of physical resource blocks (PRB) used for transmission.
- $N_{\text{REperPRB}}$ is the number of resource elements in a PRB. Each PRB has 12 resource elements.
- $N_{\text{SYM}}$ is the number of SC-FDMA symbols in a PSSCH subframe, including symbols associated with the sidelink SC-FDMA guard symbol. The number of SC-FDMA symbols in a PSSCH subframe is 12 for D2D normal cyclic prefix or 10 for D2D extended cyclic prefix and V2X.

The `info` structure output by `ltePSSCHIndices` provides $M_{\text{bits}}$ and $N_{\text{RE}}$ as `info.G` and `info.Gd` respectively.

The scrambling sequence generator is initialized with $c_{\text{init}} = n_{\text{ID}}^{\text{X}} \times 2^{14} + n_{\text{ssf}}^{\text{PSSCH}} \times 2^{9} + 510$ at the start of every PSSCH subframe. For D2D sidelink, $n_{\text{ID}}^{\text{SA}}$ is the destination identity (`NSAID`) obtained from the sidelink shared channel. For V2X, $n_{\text{ID}}^{\text{SA}}$ is the V2X scrambling identity (`NXID`). $n_{\text{ssf}}^{\text{PSSCH}}$ is the subframe number in the PSSCH subframe pool (`NSubframePSSCH`).

`ltePSSCH` requires `CyclicPrefixSL` to deduce the number of resource blocks allocated for SC-FDMA precoding symbols.

### Physical Sidelink Shared Channel Indexing

Use the `ltePSSCHIndices` function and the corresponding `ltePSSCH` sequence function to populate the PSSCH subframe resource grid. The PSSCH is transmitted in the available SC-FDMA symbols in a PSSCH subframe, using a single layer on antenna port 1000. It excludes each symbol per slot assigned to PSSCH DM-RS. For more information on PSSCH DM-RS, see the `ltePSSCHDRSIndices` function. The indices are ordered as the PSSCH modulation symbols should be mapped, applying frequency-first mapping. The resource elements in the last SC-FDMA symbol within a subframe are counted in the mapping process but should not be transmitted. The sidelink-specific SC-FDMA modulation creates this guard symbol. For more information on mapping symbols to the resource element grid, see “Resource Grid Indexing”.

# Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePSSCHDecode` | `ltePSSCHIndices` | `ltePSSCHDRS`

# ltePSSCHDecode

PSSCH decoding

## Syntax

```
[softbits,symbols] = ltePSSCHDecode(ue,sym)
```

## Description

`[softbits,symbols] = ltePSSCHDecode(ue,sym)` returns a vector of log-likelihood ratio (LLR) soft bits and the intermediate QPSK or 16QAM modulation symbols for the specified UE settings structure and modulated PSSCH symbols.

The PSSCH decoder performs the inverse of the `ltePSSCH` function processing, as defined in TS 36.211 [1], Section 9.3. The `ltePSSCHDecode` processing includes SC-FDMA transform deprecoding, symbol demodulation, and PSSCH-specific descrambling. For more information, see “Physical Sidelink Shared Channel Decoding” on page 2-772.

## Examples

### Demodulate PSSCH Symbols

Demodulate PSSCH symbols plus noise for an SL-SCH codeword created by encoding a vector of information bits. Plot the noisy RE symbols, the symbols prior to QPSK demodulation, and the resulting LLR soft bits.

#### Create a UE settings structure

Specify normal cyclic prefix and 16-QAM modulation.

```
ue = struct('CyclicPrefixSL','Normal');
ue.RV = 0;
ue.Modulation = '16QAM';
ue.NSAID = 255;
ue.NSubframePSSCH = 0;
```

#### Generate symbols to recover

- Specify the codeword length to use for the SL-SCH. Choose a length that is a multiple of 12 symbols for normal cyclic prefix and has 4 bits per symbol for 16-QAM modulation. Pick a standard number of resource blocks, such as 10.
- Create the SL-SCH codeword.
- Create the PSSCH symbols and add noise.

```
codewordlength = 5760; % (12 symbols)(4 bps)(12 REperRB)(10 PRB)
cw = lteSLSCH(ue,codewordlength,ones(100,1));

sym = ltePSSCH(ue,cw);
rxsym = sym + 0.1*randn(size(sym));
```

### Decode received PSSCH symbols

Recover the soft bits representing the transmitted SL-SCH codeword. Compare the soft bits to the transmitted codeword.

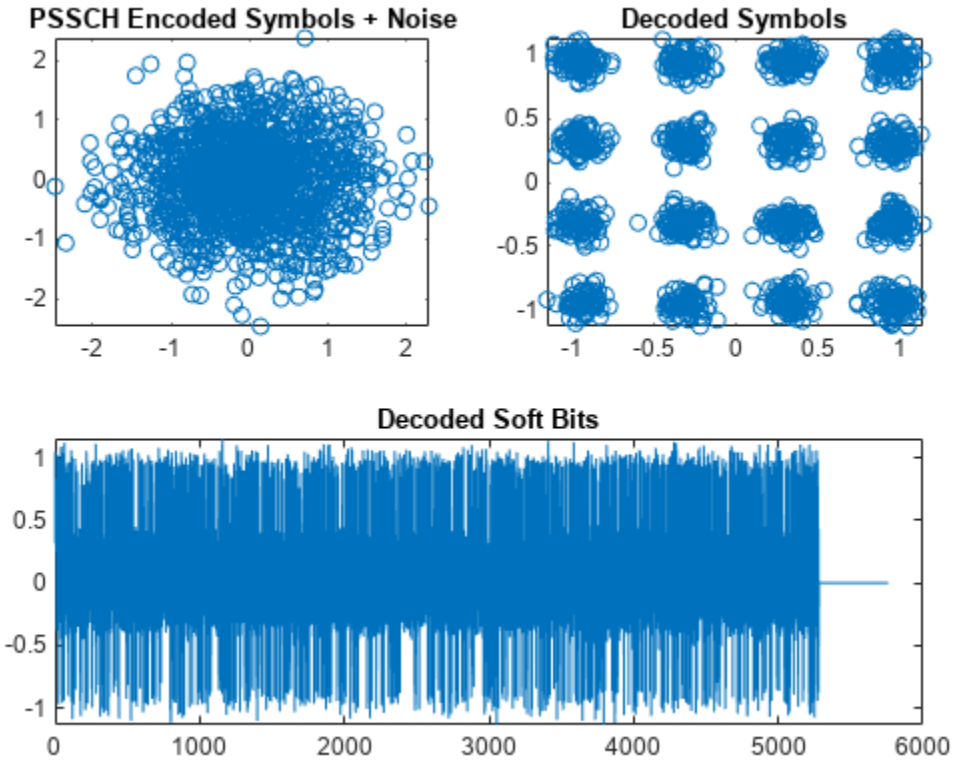
```
[rxcw,rxmodsym] = ltePSSCHDecode(ue,rxsym);
isequal(cw,rxcw>0)
```

```
ans = logical
   0
```

Using a random noise seed and the level of noise added sometimes results in decoding errors. If the comparison returns '1' there were no decoding errors. If the comparison returns '0' there were decoding errors.

Plot the received and recovered signals.

```
subplot(2,2,1)
plot(rxsym,'o')
title('PSSCH Encoded Symbols + Noise')

subplot(2,2,2)
plot(rxmodsym,'o')
title('Decoded Symbols')

subplot(2,2,[3,4])
plot(rxcw)
title('Decoded Soft Bits')
```




## Input Arguments

### ue — User equipment settings

structure

User equipment settings, specified as a parameter structure containing these fields:

### SidelinkMode — Sidelink mode

`'D2D'` (default) | `'V2X'` | optional

Sidelink mode, specified as `'D2D'` or `'V2X'`.

Data Types: `char` | `string`

### CyclicPrefixSL — Cyclic prefix length

`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

### Modulation — Modulation type

`'QPSK'` | `'16QAM'`

Modulation type, specified as `'QPSK'` or `'16QAM'`.

Data Types: `char` | `string`

**`NSAID` — Sidelink group destination identity**
integer in the interval [0, 255]

Sidelink group destination identity, specified as an integer in the interval [0, 255].

This field is the lower eight bits of the full 24-bit ProSe Layer-2 group destination ID. This field and the `NSubframePSSCH` field control the value of the scrambling sequence at the start of each subframe. This field is required only for D2D sidelink.

Data Types: `double`

**`NXID` — V2X scrambling identity**
integer scalar

V2X scrambling identity, specified as an integer scalar. `NXID` is the 16 bit CRC associated with the PSCCH SCI grant. It is only required for V2X sidelink.

Data Types: `double`

**`NSubframePSSCH` — PSSCH subframe number**
integer scalar

PSSCH subframe number in the PSSCH subframe pool, specified as an integer scalar. ($n_{\mathrm{ssf}}^{\mathrm{PSSCH}}$)

`NSubframePSSCH` and `NSAID` control the values of the scrambling sequence. It is only required for D2D sidelink.

Data Types: `double`

Data Types: `struct`

**`sym` — Encoded modulated PSSCH symbols**
column vector

Encoded modulated PSSCH symbols, specified as an $N_{\mathrm{RE}}$-by-1 column vector. $N_{\mathrm{RE}}$ is the number of RE in a subframe associated with the PSSCH allocation for normal and extended cyclic prefix (including the SC-FDMA guard symbol) and NPRB resource blocks.

$N_{\mathrm{RE}}$ is $N_{\mathrm{PRB}} \times 144$ for D2D normal cyclic prefix or $N_{\mathrm{PRB}} \times 120$ for D2D extended cyclic prefix and V2X. $N_{\mathrm{PRB}}$ is the number of physical resource blocks (PRB) used for transmission.

The function requires the contents of all PSSCH resource elements to be input, including those in the last guard symbol. For more information, see “Physical Sidelink Shared Channel Decoding” on page 2-772.

Data Types: `double`
Complex Number Support: Yes

## Output Arguments

**`softbits` — Log-likelihood ratio soft bits**
vector

Log-likelihood ratio (LLR) soft bits, returned as a vector with $N_{\mathrm{bps}} \times N_{\mathrm{RE}}$ softbits. $N_{\mathrm{bps}}$ is the number of bits per symbol. PSSCH modulation is either QPSK (2 bits per symbol) or 16 QAM (4 bits per

symbol). $N_{RE}$ is the number of PSSCH resource elements in the subframe. The LLR of the punctured soft bits associated with the last SC-FDMA symbol are set to 0.

For more information, see “Physical Sidelink Shared Channel Decoding” on page 2-772.

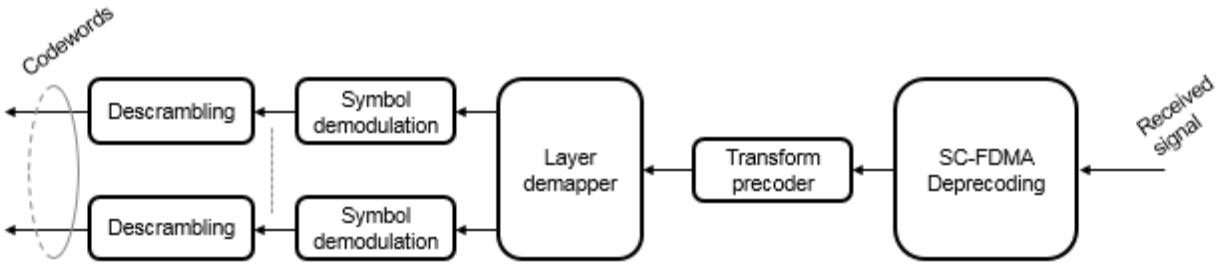
**`symbols` — Decoded modulated PSSCH symbols**
column vector

Decoded modulated PSSCH symbols, returned as a column vector with $N_{RE}$ elements. $N_{RE}$ is the number of PSSCH resource elements in the subframe. For more information, see “Physical Sidelink Shared Channel Decoding” on page 2-772.

## More About

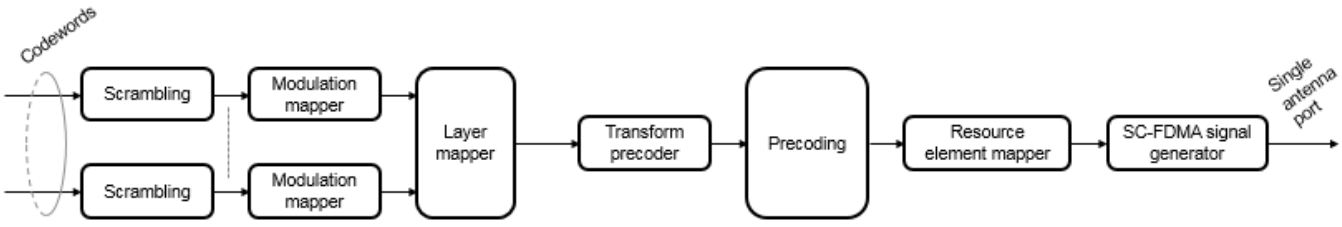
### Physical Sidelink Shared Channel Decoding

The physical sidelink shared channel (PSSCH) decoder performs the inverse of the `ltePSSCH` function processing. For more information, see “Physical Sidelink Shared Channel Processing” on page 2-772. PSSCH decoding includes SC-FDMA transform deprecoding, symbol demodulation, and PSSCH-specific descrambling.




### Physical Sidelink Shared Channel Processing

Physical sidelink shared channel (PSSCH) processing includes PSSCH-specific scrambling, QPSK or 16-QAM modulation, and SC-FDMA transform precoding. PSSCH processing follows the processing steps used for PUSCH, with variations defined in TS 36.211, Section 9.3.




For PSSCH, the input codeword length is $M_{bits} = N_{RE} \times N_{bps}$, where $N_{bps}$ is the number of bits per symbol. PSSCH modulation is either QPSK (2 bits per symbol) or 16 QAM (4 bits per symbol).

The number of PSSCH resource elements ($N_{RE}$) in a subframe is $N_{RE} = N_{PRB} \times N_{REperPRB} \times N_{SYM}$ and includes symbols associated with the sidelink SC-FDMA guard symbol.

- $N_{PRB}$ is the number of physical resource blocks (PRB) used for transmission.
- $N_{REperPRB}$ is the number of resource elements in a PRB. Each PRB has 12 resource elements.
- $N_{SYM}$ is the number of SC-FDMA symbols in a PSSCH subframe, including symbols associated with the sidelink SC-FDMA guard symbol. The number of SC-FDMA symbols in a PSSCH subframe is 12 for D2D normal cyclic prefix or 10 for D2D extended cyclic prefix and V2X.

The `info` structure output by `ltePSSCHIndices` provides $M_{bits}$ and $N_{RE}$ as `info.G` and `info.Gd` respectively.

The scrambling sequence generator is initialized with $c_{init} = n_{ID}^{X} \times 2^{14} + n_{ssf}^{PSSCH} \times 2^{9} + 510$ at the start of every PSSCH subframe. For D2D sidelink, $n_{ID}^{SA}$ is the destination identity (`NSAID`) obtained from the sidelink shared channel. For V2X, $n_{ID}^{SA}$ is the V2X scrambling identity (`NXID`). $n_{ssf}^{PSSCH}$ is the subframe number in the PSSCH subframe pool (`NSubframePSSCH`).

`ltePSSCH` requires `CyclicPrefixSL` to deduce the number of resource blocks allocated for SC-FDMA precoding symbols.

# Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePSSCH` | `ltePSSCHIndices` | `ltePSSCHDRS` | `ltePSSCHDRSIndices`

# ltePSSCHDRS

PSSCH demodulation reference signal

## Syntax

```
[seq,info] = ltePSSCHDRS(ue)
```

## Description

`[seq,info] = ltePSSCHDRS(ue)` returns a complex column vector sequence containing PSSCH demodulation reference signal (DM-RS) values and an associated information structure for the specified UE settings structure. For more information, see “PSSCH Demodulation Reference Signal Processing” on page 2-778.

## Examples

### Generate PSSCH DM-RS Sequence

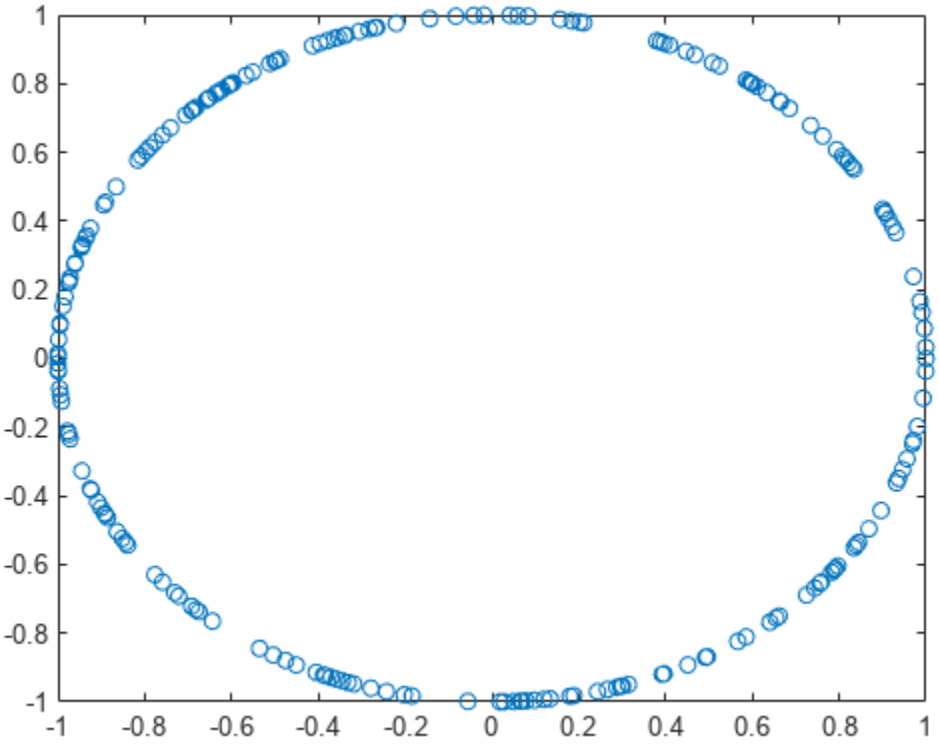
Generate a PSSCH DM-RS sequence associated with both DM-RS SC-FDMA symbols in a subframe. Plot the constellation of the sequence.

Create a user equipment settings structure.

```
ue = [];
ue.NSAID = 34;
ue.NSubframePSSCH = 5;
ue.PRBSet = (1:10)';
```

Generate a PSSCH DM-RS sequence. Plot the constellation.

```
[psschDrsSeq,info] = ltePSSCHDRS(ue);

plot(psschDrsSeq,'o')
```




### Generate a PSSCH DM-RS Sequence for V2X

Generate a PSSCH DM-RS sequence for V2X using the format 1 SCI PSCCH CRC.

Create a user equipment settings structure.

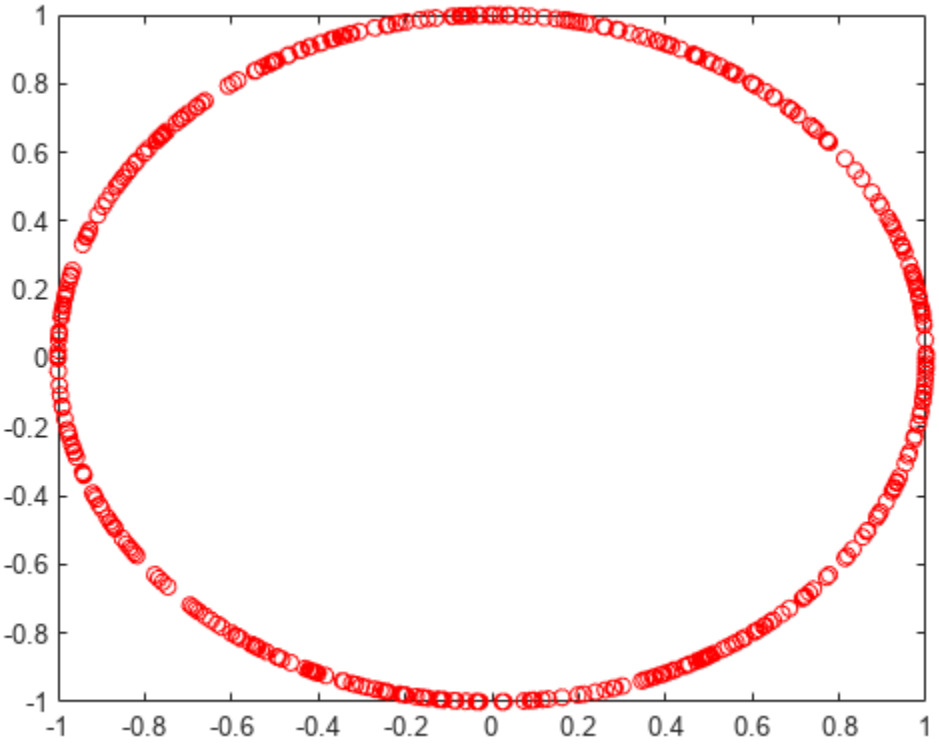
```
ue = [];
ue.SidelinkMode = 'V2X';
ue.PRBSet = (1:10)';
ue.NSLRB = 50;
```

Generate the format 1 SCI PSCCH CRC and assign it to the UE V2X scrambling identity.

```
sciinfo = lteSCIInfo(ue);
scibits = ones(1,sciinfo.Format1);
[cw,crc] = lteSCIEncode(ue,scibits);
ue.NXID = crc;
```

Generate a PSSCH DM-RS sequence. Plot the constellation.

```
[psschDrsSeq,info] = ltePSSCHDRS(ue);
plot(psschDrsSeq,'or')
```




## Input Arguments

### ue — User equipment settings

structure

User equipment settings, specified as a parameter structure containing these fields:

### SidelinkMode — Sidelink mode

`'D2D'` (default) | `'V2X'` | optional

Sidelink mode, specified as `'D2D'` or `'V2X'`.

Data Types: `char` | `string`

### NSAID — Sidelink group destination identity

integer in the interval [0, 255]

Sidelink group destination identity, specified as an integer in the interval [0, 255].

This field is the lower eight bits of the full 24-bit ProSe Layer-2 group destination ID. This field and the `NSubframePSSCH` field control the value of the scrambling sequence at the start of each subframe. This field is required only for D2D sidelink.

Data Types: `double`

**`NXID` — V2X scrambling identity**
integer scalar

V2X scrambling identity, specified as an integer scalar. `NXID` is the 16 bit CRC associated with the PSCCH SCI grant. It is only required for V2X sidelink.

Data Types: `double`

**`NSubframePSSCH` — PSSCH subframe number**
integer scalar

PSSCH subframe number in the PSSCH subframe pool, specified as an integer scalar. ($n_{ssf}^{PSSCH}$)

`NSubframePSSCH` and `NSAID` control the values of the scrambling sequence. It is only required for D2D sidelink.

Data Types: `double`

**`PRBSet` — Zero-based physical resource block indices**
integer column vector | two-column integer matrix

Zero-based physical resource block (PRB) indices, specified as an integer column vector or a two-column integer matrix.

The PSSCH is intended to be transmitted in the same PRB in each slot of a subframe. Therefore, specifying `PRBSet` as a single column of PRB indices is recommended. However, for a nonstandard slot-hopping PRB allocation, `PRBSet` can be specified as a two-column matrix of indices corresponding to slot-wise resource allocations for PSSCH.

Data Types: `double`

Data Types: `struct`

## Output Arguments

**`seq` — PSSCH DM-RS values**
column vector

PSSCH DM-RS values, returned as a $N \times 12 \times N_{PRB}$-by-1 column vector. For more information, see “PSSCH Demodulation Reference Signal Processing” on page 2-778.

**`info` — PSSCH DM-RS information**
structure

PSSCH DM-RS information about the intermediate variables used to create the DM-RS, returned as a parameter structure containing these fields:

**`Alpha` — Reference signal cyclic shift for each slot**
two-column vector

Reference signal cyclic shift for each slot, returned as a two-column vector. ($\alpha$)

`Alpha` is proportional to `NCS`, where $\alpha = \frac{2\pi n_{cs,\lambda}}{12}$.

**SeqGroup — Base sequence group number for each slot**
two-column vector

Base sequence group number for each slot, returned as a two-column vector. ($u$)

**SeqIdx — Base sequence number for each slot**
two-column vector

Base sequence number for each slot, returned as a two-column vector. ($v$)

**RootSeq — Root Zadoff-Chu sequence index for each slot**
two-column vector

Root Zadoff-Chu sequence index for each slot, returned as a two-column vector. ($q$)

**NCS — Cyclic shift values for each slot**
two-column vector

Cyclic shift values for each slot, returned as a two-column vector. ($n_{\mathrm{cs},\lambda}$)

**NZC — Zadoff-Chu sequence length**
integer

Zadoff-Chu sequence length, returned as an integer. ($N_{ZC}^{RS}$)

**OrthSeq — Orthogonal cover value for each slot**
matrix

Orthogonal cover value for each slot, returned as a matrix. ($\overline{w}$)

Data Types: `struct`

## More About

### PSSCH Demodulation Reference Signal Processing

The PSSCH demodulation reference signal (DM-RS) sequence is transmitted alongside the `ltePSSCH` values using the two SC-FDMA symbols allocated to DM-RS in a PSSCH subframe. The output vector is the repetition of a 12-element sequence and specified in TS 36.211, Section 9.8. The vector is mapped onto the 12 DM-RS SC-FDMA symbol subcarriers in each subframe slot for each PSSCH physical resource block (PRB) transmission on antenna port 1000.

The output PSSCH DM-RS sequence is the concatenation of the two sequences to be mapped onto the DM-RS SC-FDMA symbol subcarriers in each subframe slot carrying a `ltePSSCH` transmission. Its length is $N \times 12 \times N_{\mathrm{PRB}}$, where $N_{\mathrm{PRB}}$ is the number of PRBs associated with the PSSCH. For D2D sidelink, there is one DM-RS symbol per slot and therefore $N$=2, and for V2X sidelink, there are two symbols per slot and $N$=4.

### PSSCH Demodulation Reference Signal Indexing

Use the `ltePSSCHDRSIndices` indexing function and the corresponding `ltePSCCHDRS` sequence function to populate the resource grid for any PSSCH subframe. The PSSCH DM-RS is transmitted in the available SC-FDMA symbols in a PSSCH subframe, using a single layer on antenna port 1000.

The indices are ordered as the PSSCH DM-RS QPSK modulation symbols should be, applying frequency-first mapping. One-based linear indexing is the default return format, but alternative indexing formats can also be generated.

The resource elements in the last SC-FDMA symbol within a subframe are counted in the mapping process but should not be transmitted. The sidelink-specific SC-FDMA modulation creates the last symbol, which serves as a guard symbol.

For D2D sidelink, when indexing is zero-based, the SC-FDMA symbol indices used are {3,10} for normal cyclic prefix and {2,8} for extended cyclic prefix. The same symbols are used by the `ltePUSCHDRSIndices` function. For V2X sidelink, there are four DM-RS SC-FDMA symbols with indices {2,5,8,11} for normal cyclic prefix only.

**Note** The indicated symbol indices are based on TS 36.211, Section 9.8. However, to align with the LTE Toolbox subframe orientation, these indices are expanded from symbol index per slot to symbol index per subframe.

For more information on mapping symbols to the resource element grid, see “Resource Grid Indexing”.

# Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePSSCHDRSIndices` | `ltePSSCH` | `ltePSSCHDecode` | `ltePSSCHIndices`

# ltePSSCHDRSIndices

PSSCH DM-RS resource element indices

## Syntax

```
ind = ltePSSCHDRSIndices(ue)
ind = ltePSSCHDRSIndices(ue,opts)
```

## Description

`ind = ltePSSCHDRSIndices(ue)` returns a column vector of PSSCH demodulation reference signal (DM-RS) resource element indices for the specified UE settings structure. For more information, see “PSSCH Demodulation Reference Signal Indexing” on page 2-784.

`ind = ltePSSCHDRSIndices(ue,opts)` formats the returned indices using options specified by `opts`.

## Examples

### Create PSSCH DM-RS Values

Write the complex PSSCH DM-RS values into the PSSCH DM-RS resource elements in a PSSCH subframe both for D2D normal cyclic prefix and V2X. Display an image of their locations to compare both sidelink modes.

Create a user equipment settings structure and an empty resource grid subframe for 10 MHz bandwidth and normal cyclic prefix. Define a PRB allocation, `ue.PRBSet`, with RB values from 30 to 39.

```
ue = struct('NSLRB',50,'CyclicPrefixSL','Normal');
ue.NSAID = 1;
ue.NSubframePSSCH = 1;
ue.PRBSet = [30:39]';
subframe_D2D = lteSLResourceGrid(ue);
```

Generate PSSCH DM-RS indices and load PSSCH DM-RS values into the subframe.

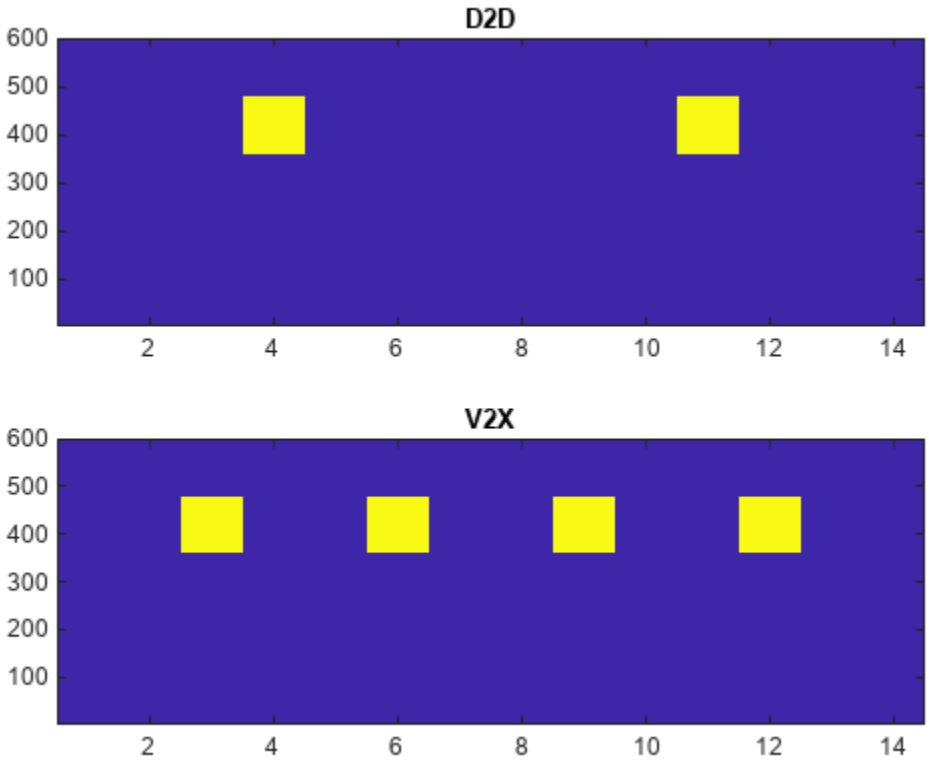
```
psschdrs_indices = ltePSSCHDRSIndices(ue);
subframe_D2D(psschdrs_indices) = ltePSSCHDRS(ue);
```

Change user equipment settings to V2X sidelink mode. Set the V2X scrambling identity to 5334.

```
ue.SidelinkMode = 'V2X';
ue.NXID = 5334;
subframe_V2X = lteSLResourceGrid(ue);
psschdrs_indices = ltePSSCHDRSIndices(ue);
subframe_V2X(psschdrs_indices) = ltePSSCHDRS(ue);
```

Display the PSCCH DM-RS locations for both sidelink modes.

```
subplot(2,1,1);
imagesc(100*abs(subframe_D2D))
axis xy; title('D2D');
subplot(2,1,2);
imagesc(100*abs(subframe_V2X));
axis xy; title(ue.SidelinkMode);
```




### Compare PSSCH DM-RS Resource Element Indexing

Compare PSSCH DM-RS resource element indexing formats.

Create a UE settings structure.

```
ue = struct('NSLRB',15,'CyclicPrefixSL','Normal','PRBSet',5);
```

Generate PSSCH DM-RS indices using one-based linear indexing (default), zero-based linear indexing, and one-based subscript row style.

### One-based linear indexing

```
psschdmrs_indices = ltePSSCHDRSIndices(ue);
psschdmrs_indices(1)
```

```
ans = uint32
    601
```

#### Zero-based linear indexing

```
opts = '0based';
psschdmrs_indices_0based = ltePSSCHDRSIndices(ue,opts);
psschdmrs_indices_0based(1)
```

```
ans = uint32
    600
```

For zero-based indexing, the first assigned index is one lower than the one-based indexing.

#### One-based indexing in `[subcarrier,symbol,port]` subscript row style

Inspect the unique symbol values to see which symbols are occupied by the PSSCH DM-RS.

```
opts = {'sub' '1based'};
psschdmrs_indices_sub = ltePSSCHDRSIndices(ue,opts);
unique(psschdmrs_indices_sub(:,2,:))
```

```
ans = 2x1 uint32 column vector

     4
    11
```

Only symbols 4 and 11 are occupied. For one-based indexing, these two PSSCH subframe symbols are always reserved for transmission of the PSSCH DM-RS.

## Input Arguments

#### `ue` — User equipment settings

structure

User equipment settings, specified as a parameter structure containing these fields:

#### `SidelinkMode` — Sidelink mode

`'D2D'` (default) | `'V2X'` | optional

Sidelink mode, specified as `'D2D'` or `'V2X'`.

Data Types: `char` | `string`

#### `NSLRB` — Number of sidelink resource blocks

integer scalar from 6 to 110

Number of sidelink resource blocks, specified as an integer scalar from 6 to 110.

Example: `6`, which corresponds to a channel bandwidth of 1.4 MHz.

Data Types: `double`

#### `CyclicPrefixSL` — Cyclic prefix length

`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

### `PRBSet` — Zero-based physical resource block indices

integer column vector | two-column integer matrix

Zero-based physical resource block (PRB) indices, specified as an integer column vector or a two-column integer matrix.

The PSSCH is intended to be transmitted in the same PRB in each slot of a subframe. Therefore, specifying `PRBSet` as a single column of PRB indices is recommended. However, for a nonstandard slot-hopping PRB allocation, `PRBSet` can be specified as a two-column matrix of indices corresponding to slot-wise resource allocations for PSSCH.

Data Types: `double`

Data Types: `struct`

### `opts` — Output format options for resource element indices

character vector | cell array of character vectors | string array

Output format options for resource element indices, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string) :

| Category | Options | Description |
|---|---|---|
| Indexing style | `'ind'` (default) | The returned indices are in linear index style. |
| | `'sub'` | The returned indices are in `[subcarrier,symbol,port]` subscript row style. |
| Index base | `'1based'` (default) | The returned indices are one-based. |
| | `'0based'` | The returned indices are zero-based. |

Example: `'ind 1based'`, `"ind 1based"`, `{'ind','1based'}`, or `["ind","1based"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `ind` — PSSCH DM-RS resource element indices

integer column vector | three-column integer matrix

PSSCH DM-RS resource element indices, returned as an integer column vector or a three-column integer matrix. The returned vector or matrix has $24 \times N_{\mathrm{PRB}}$ PSSCH DM-RS resource element indices. where $N_{\mathrm{PRB}}$ is the number of PRBs associated with the PSSCH. For more information, see “PSSCH Demodulation Reference Signal Indexing” on page 2-784.

## More About

### PSSCH Demodulation Reference Signal Indexing

Use the `ltePSSCHDRSIndices` indexing function and the corresponding `ltePSCCHDRS` sequence function to populate the resource grid for any PSSCH subframe. The PSSCH DM-RS is transmitted in the available SC-FDMA symbols in a PSSCH subframe, using a single layer on antenna port 1000.

The indices are ordered as the PSSCH DM-RS QPSK modulation symbols should be, applying frequency-first mapping. One-based linear indexing is the default return format, but alternative indexing formats can also be generated.

The resource elements in the last SC-FDMA symbol within a subframe are counted in the mapping process but should not be transmitted. The sidelink-specific SC-FDMA modulation creates the last symbol, which serves as a guard symbol.

For D2D sidelink, when indexing is zero-based, the SC-FDMA symbol indices used are {3,10} for normal cyclic prefix and {2,8} for extended cyclic prefix. The same symbols are used by the `ltePUSCHDRSIndices` function. For V2X sidelink, there are four DM-RS SC-FDMA symbols with indices {2,5,8,11} for normal cyclic prefix only.

**Note** The indicated symbol indices are based on TS 36.211, Section 9.8. However, to align with the LTE Toolbox subframe orientation, these indices are expanded from symbol index per slot to symbol index per subframe.

For more information on mapping symbols to the resource element grid, see “Resource Grid Indexing”.

### PSSCH Demodulation Reference Signal

The PSSCH demodulation reference signal (DM-RS) sequence is transmitted alongside the `ltePSSCH` values using the two SC-FDMA symbols allocated to DM-RS in a PSSCH subframe. The output vector is the repetition of a 12-element sequence and specified in TS 36.211, Section 9.8. The vector is mapped onto the 12 DM-RS SC-FDMA symbol subcarriers in each subframe slot for each PSSCH physical resource block (PRB) transmission on antenna port 1000.

The output PSSCH DM-RS sequence is the concatenation of the two sequences to be mapped onto the DM-RS SC-FDMA symbol subcarriers in each subframe slot carrying a `ltePSSCH` transmission. Its length is $N \times 12 \times N_{\mathrm{PRB}}$, where $N_{\mathrm{PRB}}$ is the number of PRBs associated with the PSSCH. For D2D sidelink, there is one DM-RS symbol per slot and therefore $N$=2, and for V2X sidelink, there are two symbols per slot and $N$=4.

# Version History

**Introduced in R2016b**

## See Also

`ltePSSCHDRS` | `ltePSSCH` | `ltePSSCHDecode`

# ltePSSCHIndices

PSSCH resource element indices

## Syntax

```
[ind] = ltePSSCHIndices(ue)
[ind,info] = ltePSSCHIndices(ue)
[___] = ltePSSCHIndices(ue,opts)
```

## Description

`[ind] = ltePSSCHIndices(ue)` returns a column vector of physical sidelink shared channel (PSSCH) resource element (RE) indices for the specified UE settings structure. By default, the indices are returned in one-based linear indexing form. You can use this form to directly index elements of a matrix representing the subframe resource grid for antenna port 1000. For more information, see “Physical Sidelink Shared Channel Indexing” on page 2-790.

`[ind,info] = ltePSSCHIndices(ue)` also returns a structure containing PSSCH-related information for the specified UE settings structure.

`[___] = ltePSSCHIndices(ue,opts)` formats the returned indices using options specified by `opts`. This syntax supports output options from prior syntaxes.

## Examples

### Map PSSCH Resource Elements

Write the complex PSSCH values into the PSSCH resource elements in a PSSCH subframe both for D2D normal cyclic prefix and V2X. Display an image of their locations to compare both sidelink modes. This mapping writes PSSCH values into the last SC-FDMA guard symbol within a subframe. The sidelink SC-FDMA modulator removes these values before transmission of the waveform.

Create a UE settings structure, an empty sidelink resource grid and D2D normal cyclic prefix. Define a PRB allocation, `ue.PRBSet`, with RB values from 30 to 39.

```
ue = struct('NSLRB',50,'CyclicPrefixSL','Normal');
ue.NSAID = 1;
ue.NSubframePSSCH = 1;
ue.PRBSet = [30:39]';
ue.Modulation = 'QPSK';
subframe_D2D = lteSLResourceGrid(ue);
```

Generate PSSCH indices. Populate the PSSCH resource elements in the subframe using a vector filled with zeros. For D2D normal cyclic prefix a PSSCH subframe contains (144 * `nprb` ) REs. The number of resource blocks is set to 10. Because the PSSCH uses QPSK modulation, there are 2 bits per symbol.

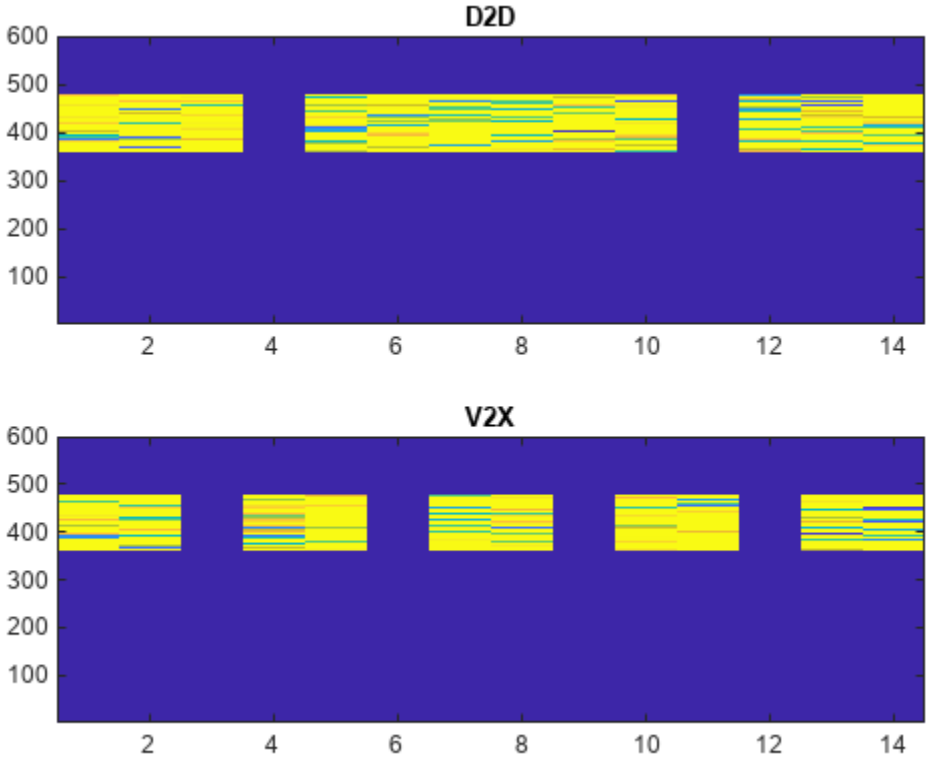
```
pssch_indices = ltePSSCHIndices(ue);
subframe_D2D(pssch_indices) = ltePSSCH(ue,zeros(2*10*144,1));
```

Change user equipment settings to V2X sidelink mode. Set the V2X scrambling identity to 4567.

```
ue.SidelinkMode = 'V2X';
ue.NXID = 4567;
subframe_V2X = lteSLResourceGrid(ue);
pssch_indices = ltePSSCHIndices(ue);
subframe_V2X(pssch_indices) = ltePSSCH(ue,zeros(2*10*120,1));
```

View the resource grid for both sidelink modes.

```
subplot(2,1,1);
image(400*abs(subframe_D2D));
axis xy; title('D2D');
subplot(2,1,2);
image(400*abs(subframe_V2X));
axis xy; title(ue.SidelinkMode);
```




### View PSSCH Information Structure

View the information structure output by the PSSCH resource element indexing function.

Create a UE settings structure.

```
ue = struct('NSLRB',25,'CyclicPrefixSL','Normal','PRBSet',[5:22]', ...
    'Modulation','16QAM');
```

Generate PSSCH indices and the information structure. View the information structure to see the bit and symbol capacity of the PSSCH for this configuration.

```
[pssch_indices,info] = ltePSSCHIndices(ue);
info
```

```
info = struct with fields:
     G: 10368
    Gd: 2592
```

### Compare PSSCH Resource Element Indexing

Compare PSSCH resource element indexing formats.

Create a UE settings structure.

```
ue = struct('NSLRB',15,'CyclicPrefixSL','Normal','PRBSet',12);
```

Generate PSSCH indices using one-based linear indexing (default), zero-based linear indexing, and one-based subscript row style.

#### One-based linear indexing

```
pssch_indices = ltePSSCHIndices(ue);
pssch_indices(1)
```

```
ans = uint32
    145
```

#### Zero-based linear indexing

```
opts = '0based';
pssch_indices_0based = ltePSSCHIndices(ue,opts);
pssch_indices_0based(1)
```

```
ans = uint32
    144
```

For zero-based indexing, the first assigned index is one lower than the one-based indexing.

#### One-based indexing in `[subcarrier,symbol,port]` subscript row style

Inspect the unique symbol values to see which symbols are occupied by the PSSCH.

```
opts = {'sub' '1based'};
pssch_indices_sub = ltePSSCHIndices(ue,opts);
unique(pssch_indices_sub(:,2,:))
```

```
ans = 12x1 uint32 column vector

    1
    2
    3
    5
    6
```

```
 7
 8
 9
10
12
   ⋮
```

Only the symbols 4 and 11 are not occupied. For one-based indexing, these two PSSCH subframe symbols are always reserved for transmission of the PSSCH DM-RS.

## Input Arguments

### `ue` — User equipment settings

structure

User equipment settings, specified as a parameter structure containing these fields:

#### `SidelinkMode` — Sidelink mode

`'D2D'` (default) | `'V2X'` | optional

Sidelink mode, specified as `'D2D'` or `'V2X'`.

Data Types: `char` | `string`

#### `NSLRB` — Number of sidelink resource blocks

integer scalar from 6 to 110

Number of sidelink resource blocks, specified as an integer scalar from 6 to 110.

Example: `6`, which corresponds to a channel bandwidth of 1.4 MHz.

Data Types: `double`

#### `CyclicPrefixSL` — Cyclic prefix length

`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

#### `PRBSet` — Zero-based physical resource block indices

integer column vector | two-column integer matrix

Zero-based physical resource block (PRB) indices, specified as an integer column vector or a two-column integer matrix.

The PSSCH is intended to be transmitted in the same PRB in each slot of a subframe. Therefore, specifying `PRBSet` as a single column of PRB indices is recommended. However, for a nonstandard slot-hopping PRB allocation, `PRBSet` can be specified as a two-column matrix of indices corresponding to slot-wise resource allocations for PSSCH.

Data Types: `double`

#### `Modulation` — Modulation type

`'QPSK'` (default) | `'16QAM'`

Modulation type, specified as `'QPSK'` or `'16QAM'`. Only required when the `info` output is assigned. `Modulation` is used to set the `info.G` output field.

Data Types: `char` | `string`

Data Types: `struct`

### `opts` — Output format options for resource element indices

character vector | cell array of character vectors | string array

Output format options for resource element indices, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string) :

| Category | Options | Description |
|---|---|---|
| Indexing style | `'ind'` (default) | The returned indices are in linear index style. |
| | `'sub'` | The returned indices are in `[subcarrier,symbol,port]` subscript row style. |
| Index base | `'1based'` (default) | The returned indices are one-based. |
| | `'0based'` | The returned indices are zero-based. |

Example: `'ind 1based'`, `"ind 1based"`, `{'ind','1based'}`, or `["ind","1based"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `ind` — PSSCH resource element indices

integer column vector | three-column integer matrix

PSSCH resource element indices, returned as an integer column vector or a three-column integer matrix. The returned vector or matrix has $N_{PRB} \times 144$ PSSCH resource element indices for D2D normal cyclic prefix or $N_{PRB} \times 120$ PSSCH resource element indices for D2D extended cyclic prefix and V2X. $N_{PRB}$ is the number of physical resource blocks (PRB) used for transmission. For more information, see “Physical Sidelink Shared Channel Indexing” on page 2-790 and “Physical Sidelink Shared Channel Processing” on page 2-790.

### `info` — PSSCH subframe resource information

structure

PSSCH subframe resource information, returned as a structure containing these fields:

### `G` — PSSCH bit capacity

integer

PSSCH bit capacity, returned as an integer. For more information, see “Physical Sidelink Shared Channel Processing” on page 2-790.

**`Gd` — PSSCH symbol capacity**
integer

PSSCH symbol capacity, returned as an integer. The number of PSSCH resource elements ($N_{RE}$) in a subframe. For more information, see “Physical Sidelink Shared Channel Processing” on page 2-790.

Data Types: `struct`

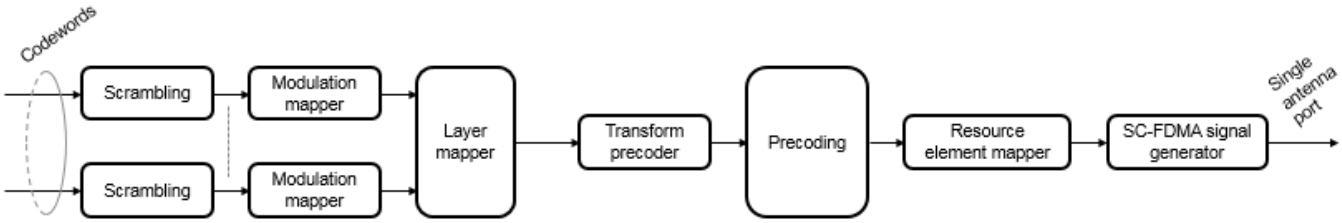
## More About

### Physical Sidelink Shared Channel Indexing

Use the `ltePSSCHIndices` function and the corresponding `ltePSSCH` sequence function to populate the PSSCH subframe resource grid. The PSSCH is transmitted in the available SC-FDMA symbols in a PSSCH subframe, using a single layer on antenna port 1000. It excludes each symbol per slot assigned to PSSCH DM-RS. For more information on PSSCH DM-RS, see the `ltePSSCHDRSIndices` function. The indices are ordered as the PSSCH modulation symbols should be mapped, applying frequency-first mapping. The resource elements in the last SC-FDMA symbol within a subframe are counted in the mapping process but should not be transmitted. The sidelink-specific SC-FDMA modulation creates this guard symbol. For more information on mapping symbols to the resource element grid, see “Resource Grid Indexing”.

### Physical Sidelink Shared Channel Processing

Physical sidelink shared channel (PSSCH) processing includes PSSCH-specific scrambling, QPSK or 16-QAM modulation, and SC-FDMA transform precoding. PSSCH processing follows the processing steps used for PUSCH, with variations defined in TS 36.211, Section 9.3.




For PSSCH, the input codeword length is $M_{bits} = N_{RE} \times N_{bps}$, where $N_{bps}$ is the number of bits per symbol. PSSCH modulation is either QPSK (2 bits per symbol) or 16 QAM (4 bits per symbol).

The number of PSSCH resource elements ($N_{RE}$) in a subframe is $N_{RE} = N_{PRB} \times N_{REperPRB} \times N_{SYM}$ and includes symbols associated with the sidelink SC-FDMA guard symbol.

- $N_{PRB}$ is the number of physical resource blocks (PRB) used for transmission.
- $N_{REperPRB}$ is the number of resource elements in a PRB. Each PRB has 12 resource elements.
- $N_{SYM}$ is the number of SC-FDMA symbols in a PSSCH subframe, including symbols associated with the sidelink SC-FDMA guard symbol. The number of SC-FDMA symbols in a PSSCH subframe is 12 for D2D normal cyclic prefix or 10 for D2D extended cyclic prefix and V2X.

The `info` structure output by `ltePSSCHIndices` provides $M_{bits}$ and $N_{RE}$ as `info.G` and `info.Gd` respectively.

The scrambling sequence generator is initialized with $c_{\text{init}} = n_{\text{ID}}^{\text{X}} \times 2^{14} + n_{\text{ssf}}^{\text{PSSCH}} \times 2^{9} + 510$ at the start of every PSSCH subframe. For D2D sidelink, $n_{\text{ID}}^{\text{SA}}$ is the destination identity (`NSAID`) obtained from the sidelink shared channel. For V2X, $n_{\text{ID}}^{\text{SA}}$ is the V2X scrambling identity (`NXID`). $n_{\text{ssf}}^{\text{PSSCH}}$ is the subframe number in the PSSCH subframe pool (`NSubframePSSCH`).

`ltePSSCH` requires `CyclicPrefixSL` to deduce the number of resource blocks allocated for SC-FDMA precoding symbols.

# Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePSSCH` | `ltePSSCHDecode` | `ltePSSCHPRBS`

# ltePSSCHPRBS

PSSCH pseudorandom binary scrambling sequence

## Syntax

```
[seq,cinit] = ltePSSCHPRBS(ue,n)
[seq,cinit] = ltePSSCHPRBS(ue,n,mapping)

[subseq,cinit] = ltePSSCHPRBS(ue,pn)
[subseq,cinit] = ltePSSCHPRBS(ue,pn,mapping)
```

## Description

`[seq,cinit] = ltePSSCHPRBS(ue,n)` returns the first `n` outputs of the PSSCH pseudorandom binary scrambling sequence (PRBS) for the specified UE settings structure. It also returns an initialization value `cinit` for the pseudorandom binary sequence (PRBS) generator.

The scrambling sequence generated should be applied to the coded PSSCH data carried by the associated subframe. The PRBS sequence generator used is initialized with $c_{\text{init}} = n_{\text{ID}}^{\text{X}} \times 2^{14} + n_{\text{ssf}}^{\text{PSSCH}} \times 2^{9} + 510$. For more information, see "Physical Sidelink Shared Channel Processing" on page 2-795, `ue.NSAID` and `ue.NXID`

`[seq,cinit] = ltePSSCHPRBS(ue,n,mapping)` specifies the format of the returned sequence, `seq`, through the `mapping` input.

`[subseq,cinit] = ltePSSCHPRBS(ue,pn)` returns a subsequence of a full PRBS sequence, specified by `pn`.

`[subseq,cinit] = ltePSSCHPRBS(ue,pn,mapping)` specifies the format of the returned subsequence, `subseq`, through the `mapping` input.

## Examples

### Scramble PSSCH Codeword

Scramble a PSSCH codeword by generating the PSSCH pseudorandom binary sequence (PRBS) and applying an exclusive OR operation on the two sequences.

Create a UE settings structure with required fields. Generate the required length of the PRBS. Scramble the PSSCH codeword with the PRBS sequence using `xor`.

```
ue = struct('NSAID',255,'NSubframePSSCH',0);

codeword = ones(1152,1);
psschPrbs = ltePSSCHPRBS(ue,length(codeword));

scrambled = xor(psschPrbs,codeword);
```

### Descramble PSSCH Codeword

Descramble a received PSSCH codeword.

### Scramble PSSCH Codeword

- Create a UE settings structure with required fields.
- Generate the required length of the PRBS.
- Scramble the PSSCH codeword with the PRBS sequence using `xor`.
- Modulate the logical scrambled data.

```
ue = struct('NSAID',255,'NSubframePSSCH',0);

codeword = ones(1152,1);
psschPrbs = ltePSSCHPRBS(ue,length(codeword));

scrambled = xor(psschPrbs,codeword);

txsym = lteSymbolModulate(scrambled,'16QAM');
```

### Descramble Recovered Codeword

- Add noise to transmitted symbols and demodulate received soft data.
- Generate the PSSCH PRBS in signed form.
- Descramble the vector representing a sequence of soft bits by generating the PSSCH PRBS in signed form and performing a pointwise multiplication between the PRBS sequence and the recovered soft data.
- Compare the transmitted codeword to the recovered codeword.

```
sym = awgn(txsym,30,'measured');
softdata = lteSymbolDemodulate(sym,'16QAM');

scramblingSeq = ltePSSCHPRBS(ue,length(softdata),'signed');
descrambled = softdata.*scramblingSeq;

isequal(codeword,descrambled > 0)
```

```
ans = logical
   1
```

The transmitted codeword matches the hard decision on the descrambled data.

## Input Arguments

### ue — User equipment settings

structure

User equipment settings, specified as a parameter structure containing these fields:

### SidelinkMode — Sidelink mode

`'D2D'` (default) | `'V2X'` | optional

Sidelink mode, specified as `'D2D'` or `'V2X'`.

Data Types: `char` | `string`

**`NSAID` — Sidelink group destination identity**
integer in the interval [0, 255]

Sidelink group destination identity, specified as an integer in the interval [0, 255].

This field is the lower eight bits of the full 24-bit ProSe Layer-2 group destination ID. This field and the `NSubframePSSCH` field control the value of the scrambling sequence at the start of each subframe. This field is required only for D2D sidelink.

Data Types: `double`

**`NXID` — V2X scrambling identity**
integer scalar

V2X scrambling identity, specified as an integer scalar. `NXID` is the 16 bit CRC associated with the PSCCH SCI grant. It is only required for V2X sidelink.

Data Types: `double`

**`NSubframePSSCH` — PSSCH subframe number**
integer scalar

PSSCH subframe number in the PSSCH subframe pool, specified as an integer scalar. ($n_{\mathrm{ssf}}^{\mathrm{PSSCH}}$)

`NSubframePSSCH` and `NSAID` control the values of the scrambling sequence. It is only required for D2D sidelink.

Data Types: `double`

Data Types: `struct`

**`n` — Number of elements in returned sequence**
numeric scalar

Number of elements in returned sequence, `seq`, specified as a numeric scalar.

Data Types: `double`

**`pn` — Range of elements in returned subsequence**
row vector

Range of elements in returned subsequence, `subseq`, specified as a row vector of `[p n]`. The subsequence returns `n` values of the PRBS generator, starting at position `p` (0-based).

Data Types: `double`

**`mapping` — Output sequence formatting**
`'binary'` (default) | `'signed'`

Output sequence formatting, specified as `'binary'` or `'signed'`.

- `'binary'` maps `true` to 1 and `false` to 0.
- `'signed'` maps `true` to –1 and `false` to 1.

Data Types: `char` | `string`

## Output Arguments

### `seq` — PSSCH pseudorandom scrambling sequence

logical column vector | numeric column vector

PSSCH pseudorandom scrambling sequence, returned as a logical column vector or a numeric column vector. `seq` contains the first `n` outputs of the physical sidelink shared channel (PSSCH) scrambling sequence. If you set `mapping` to `'signed'`, the output data type is `double`. Otherwise, the output data type is `logical`.

Data Types: `logical` | `double`

### `subseq` — PSSCH pseudorandom scrambling subsequence

logical column vector | numeric column vector

PSSCH pseudorandom scrambling subsequence, returned as a logical column vector or a numeric column vector. `subseq` contains the values of the PRBS generator specified by `pn`. If you set `mapping` to `'signed'`, the output data type is `double`. Otherwise, the output data type is `logical`.

Data Types: `logical` | `double`

### `cinit` — Initialization value for PRBS generator

numeric scalar

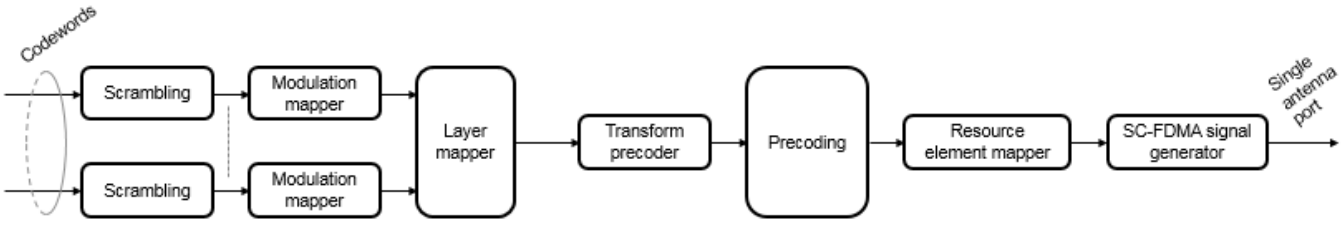
Initialization value for PRBS generator, returned as a numeric scalar.

Data Types: `uint32`

## More About

### Physical Sidelink Shared Channel Processing

Physical sidelink shared channel (PSSCH) processing includes PSSCH-specific scrambling, QPSK or 16-QAM modulation, and SC-FDMA transform precoding. PSSCH processing follows the processing steps used for PUSCH, with variations defined in TS 36.211, Section 9.3.




For PSSCH, the input codeword length is $M_{\text{bits}} = N_{\text{RE}} \times N_{\text{bps}}$, where $N_{\text{bps}}$ is the number of bits per symbol. PSSCH modulation is either QPSK (2 bits per symbol) or 16 QAM (4 bits per symbol).

The number of PSSCH resource elements ($N_{\text{RE}}$) in a subframe is $N_{\text{RE}} = N_{\text{PRB}} \times N_{\text{REperPRB}} \times N_{\text{SYM}}$ and includes symbols associated with the sidelink SC-FDMA guard symbol.

- $N_{\text{PRB}}$ is the number of physical resource blocks (PRB) used for transmission.
- $N_{\text{REperPRB}}$ is the number of resource elements in a PRB. Each PRB has 12 resource elements.

- $N_{SYM}$ is the number of SC-FDMA symbols in a PSSCH subframe, including symbols associated with the sidelink SC-FDMA guard symbol. The number of SC-FDMA symbols in a PSSCH subframe is 12 for D2D normal cyclic prefix or 10 for D2D extended cyclic prefix and V2X.

The `info` structure output by `ltePSSCHIndices` provides $M_{bits}$ and $N_{RE}$ as `info.G` and `info.Gd` respectively.

The scrambling sequence generator is initialized with $c_{init} = n_{ID}^{X} \times 2^{14} + n_{ssf}^{PSSCH} \times 2^{9} + 510$ at the start of every PSSCH subframe. For D2D sidelink, $n_{ID}^{SA}$ is the destination identity (`NSAID`) obtained from the sidelink shared channel. For V2X, $n_{ID}^{SA}$ is the V2X scrambling identity (`NXID`). $n_{ssf}^{PSSCH}$ is the subframe number in the PSSCH subframe pool (`NSubframePSSCH`).

`ltePSSCH` requires `CyclicPrefixSL` to deduce the number of resource blocks allocated for SC-FDMA precoding symbols.

# Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePSSCH` | `ltePSSCHIndices` | `ltePSSCHDecode` | `ltePRBS`

# ltePSS

Primary synchronization signal

## Syntax

```
s = ltePSS(enb)
```

## Description

`s = ltePSS(enb)` returns a complex column vector containing the primary synchronization signal (PSS) values for cell-wide settings in the `enb` structure.

This signal is only defined for subframes 0 and 5 in FDD, and subframes 1 and 6 in TDD. Therefore, an empty vector is returned for other values of `NSubframe`. This behavior allows this function and the corresponding sequence function `ltePSSIndices` to index the resource grid for any subframe number as described in “Resource Grid Indexing”. However, the resource grid is only modified in subframes 0 and 5 in FDD, or subframes 1 and 6 in TDD.

## Examples

### Generate Primary Synchronization Signal Values

Generate the primary synchronization signal (PSS) values using the cell-wide settings provided.

```
pss = ltePSS(struct('NCellID',1,'NSubframe',0,'DuplexMode','FDD'));
pss(1:4)
```

```
ans = 4×1 complex

   1.0000 + 0.0000i
  -0.9691 - 0.2468i
  -0.7331 - 0.6802i
   0.0747 + 0.9972i
```

## Input Arguments

### `enb` — Cell-wide settings

structure

Cell-wide settings, specified as a structure. `enb` contains the following fields.

### `NCellID` — Physical layer cell identity number

nonnegative scalar integer

Physical layer cell identity number, specified as a nonnegative scalar integer.

Example: 6

Data Types: `double`

**`NSubframe` — Subframe number**
0 (default) | optional | nonnegative scalar integer

Subframe number, specified as nonnegative scalar integer.

Example: 8

Data Types: `double`

**`DuplexMode` — Duplex mode type**
`'FDD'` (default) | optional | `'TDD'`

Duplex mode type, specified as `'FDD'` or `'TDD'`. Used for separating the transmission signals.

Data Types: `char` | `string`

## Output Arguments

**`s` — Primary synchronization signal (PSS) values**
complex-valued numeric column vector

Primary synchronization signal (PSS) values, returned as a complex-valued numeric column vector. These values are created for the cell-wide settings in the `enb` structure.

Example: 1.0000 + 0.0000i

Data Types: `double`

# Version History
**Introduced in R2014a**

## See Also
`ltePSSIndices` | `lteSSS` | `ltePSSS`

# ltePSSIndices

PSS resource element indices

## Syntax

```
ind = ltePSSIndices(enb)
ind = ltePSSIndices(enb,port)
ind = ltePSSIndices(enb,port,opts)
```

## Description

`ind = ltePSSIndices(enb)` returns a column vector, `ind`, of resource element (RE) indices, Port 0 oriented, for the Primary Synchronization Signal (PSS) for the given cell-wide settings structure. By default, the indices are returned in one-based linear indexing form that can directly index elements of a 3-D array representing the resource array. These indices are ordered as the PSS modulation symbols should be mapped. Alternative indexing formats can also be generated.

---

**Note** These indices are only defined for subframes 0 and 5 in FDD, and subframes 1 and 6 in TDD. Therefore, an empty vector is returned for other values of `NSubframe`. This behavior allows this function and the corresponding sequence function `ltePSS` to index the resource grid for any subframe number as described in “Resource Grid Indexing”. However, the resource grid is only modified in subframes 0 and 5 in FDD, or subframes 1 and 6 in TDD.

---

`ind = ltePSSIndices(enb,port)` returns indices appropriate for antenna port, `port`.

`ind = ltePSSIndices(enb,port,opts)` formats the returned indices using options specified by `opts`.

## Examples

### Get PSS Resource Element Indices

Get PSS resource element indices in linear form for antenna port 0.

Create a cell-wide configuration structure initialed for RMC R.4. Generate PSS indices for RMC R.4 for antenna port 0.

```
enb = lteRMCDL('R.4');
ind = ltePSSIndices(enb,0);
ind(1:4)
```

```
ans = 4x1 uint32 column vector

   438
   439
   440
   441
```

### Get Zero-based PSS Resource Element Indices

Get zero-based PSS resource element indices in linear form for antenna port 0.

```
enb = lteRMCDL('R.4');
ind = ltePSSIndices(enb,0,{'0based','ind'});
ind(1:4)
```

```
ans = 4x1 uint32 column vector

   437
   438
   439
   440
```

## Input Arguments

**enb — Cell-wide settings**
structure

Cell-wide settings, specified as a structure. `enb` contains the following fields.

**`NDLRB` — Number of downlink resource blocks**
integer from 6 to 110

Number of downlink resource blocks, specified as integer from 6 to 110.

Example: 9

Data Types: `double`

**`CyclicPrefix` — Cyclic prefix length**
`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

**`NSubframe` — Subframe number**
0 (default) | nonnegative scalar integer | optional

Subframe number, specified as nonnegative scalar integer.

Example: 9

Data Types: `double`

**`DuplexMode` — Duplex mode type**
`'FDD'` (default) | `'TDD'` | optional

Duplex mode type, specified as `'FDD'` or `'TDD'`.

Data Types: `char` | `string`

**`port` — Antenna port number**
non-negative scalar integer

Antenna port number, specified as a non-negative scalar integer.

Example: 2

Data Types: `double`

**`opts` — Output format options for resource element indices**
character vector | cell array of character vectors | string array

Output format options for resource element indices, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string) :

| Category | Options | Description |
|---|---|---|
| Indexing style | `'ind'` (default) | The returned indices are in linear index style. |
| | `'sub'` | The returned indices are in `[subcarrier,symbol,port]` subscript row style. |
| Index base | `'1based'` (default) | The returned indices are one-based. |
| | `'0based'` | The returned indices are zero-based. |

Example: `'ind 1based'`, `"ind 1based"`, `{'ind','1based'}`, or `["ind","1based"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

**`ind` — PSS resource element indices**
integer column vector | 3-column integer matrix

PSS resource element indices, returned as an integer column vector or a three-column integer matrix. This output is generated using the cell-wide settings structure, `enb`.

Data Types: `uint32`

# Version History

**Introduced in R2014a**

## See Also

`ltePSS` | `lteSSSIndices` | `ltePSSSIndices`

**Topics**
“Resource Grid Indexing”

# ltePSSS

Primary sidelink synchronization signal

## Syntax

```
s = ltePSSS(ue)
```

## Description

`s = ltePSSS(ue)` returns a 124-by-1 complex column vector containing the primary sidelink synchronization signal (PSSS) values for user equipment settings in the `ue` structure. For more information, see “Primary Sidelink Synchronization Signal” on page 2-804.

## Examples

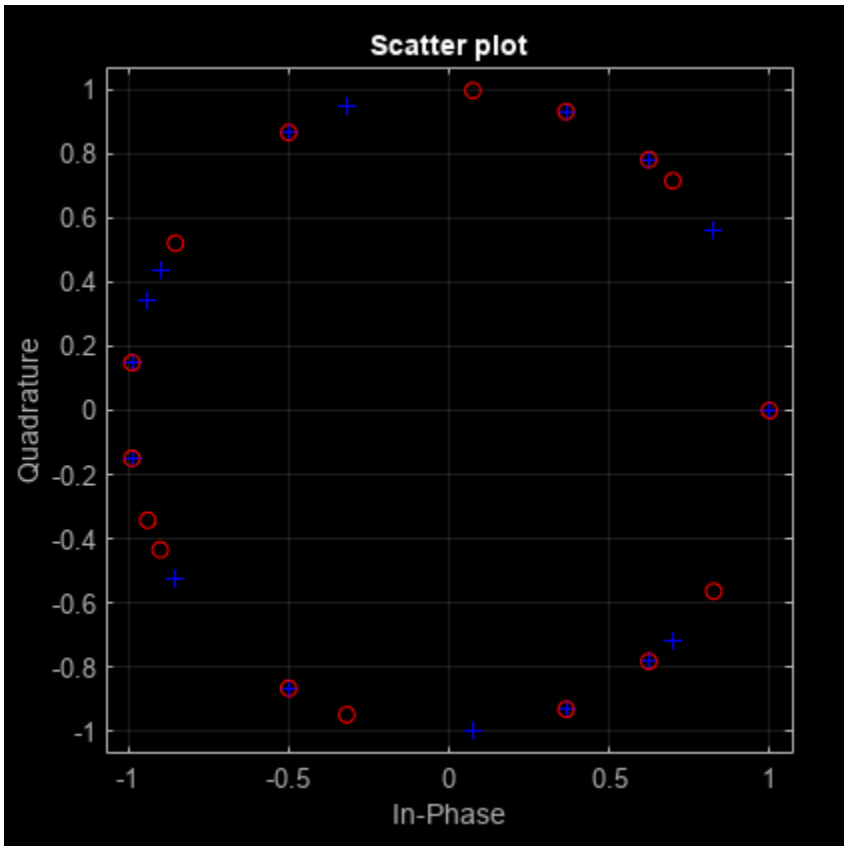
### Generate PSSS

Generate PSSS values for in-coverage and out-of-coverage identities.

```
psss_net = ltePSSS(struct('NSLID',0));
psss_oon = ltePSSS(struct('NSLID',168));
```

Plot the returned synchronization signal for the in-coverage identities (blue, +) and the out-of-coverage identities (red, o).

```
scatPlot = scatterplot(psss_net,1,0,'b+');
grid
hold on
scatterplot(psss_oon,1,0,'ro',scatPlot)
```




## Input Arguments

### ue — User equipment settings

structure

UE-specific settings, specified as a structure containing this parameter field:

### `NSLID` — Physical layer sidelink synchronization identity

integer from 0 to 335

Physical layer sidelink synchronization identity, specified as an integer from 0 to 335. ($N_{\mathrm{ID}}^{\mathrm{SL}}$)

For more information, see “Primary Sidelink Synchronization Signal” on page 2-804.

Example: 6

Data Types: `double`

## Output Arguments

### s — PSSS values

complex-valued numeric column vector

PSSS values, returned as a 124-by-1 complex-valued numeric column vector. These values are created for the user equipment settings in the `ue` structure. For more information, see “Primary Sidelink Synchronization Signal” on page 2-804.

## More About

### Primary Sidelink Synchronization Signal

The primary sidelink synchronization signal (PSSS) is transmitted in the central 62 resource elements of two adjacent SC-FDMA symbols in a synchronization subframe. The same sequence of 62 complex values is repeated in each of the symbols, resulting in a 124-by-1 element vector returned by the `ltePSSS` function. The values of this sequence are ordered as they should be mapped into the resource elements of the adjacent symbols using `ltePSSSIndices`. If a terminal is transmitting PSSS, then the PSSS should be sent every 40 ms with the exact subframe dependent on the RRC signaled subframe number offset (syncOffsetIndicator-r12).

The PSSS is sent on antenna port 1020, along with the secondary sidelink synchronization signal (SSSS). A synchronization subframe also contains the PSBCH, which is sent on antenna port 1010. The transmission power of the PSSS symbols should be the same as the PSBCH therefore they should be scaled by $\sqrt{72/62}$ in a subframe. No PSCCH or PSSCH transmission will occur in a sidelink subframe configured for synchronization purposes.

As specified in TS 36.211, Section 9.7, the PSSS identity assignment depends on the network coverage. The set of all $N_{\mathrm{ID}}^{\mathrm{SL}}$ is divided into two sets, id_net {0, ..., 167} and id_oon {168, ..., 335}, which are used by terminals that are in-network and out-of-network coverage, respectively. The sidelink physical layer cell identity number, $N_{\mathrm{ID}}^{\mathrm{SL}}$, corresponds to the `ltePSSS` input UE settings structure field `ue.NSLID`. Within each set, all identities result in the same PSSS. For an in-network terminal, the `ue.NSLID` value corresponds to the RRC sidelink synchronization signal identity (slssid-r12) associated with the cell.

### Primary Sidelink Synchronization Signal Indexing

Use the indexing function, `ltePSSSIndices`, and the corresponding sequence function, `ltePSSS`, to populate the resource grid for the desired subframe number. The PSSS values are output by `ltePSSS`, ordered as they should be mapped, applying frequency-first mapping into the resource elements of the adjacent symbols using `ltePSSSIndices`. When indexing is zero-based, the SC-FDMA symbols used are {1,2} for normal cyclic prefix and {0, 1} for extended cyclic prefix.

**Note** The indicated symbol indices are based on TS 36.211, Section 9.7. However to align with the LTE Toolbox subframe orientation, these indices are expanded from symbol index per slot to symbol index per subframe.

For more information on mapping symbols to the resource element grid, see “Resource Grid Indexing”.

# Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePSSSIndices` | `lteSSSS` | `ltePSS` | `ltePSBCH`

**Topics**

“Resource Grid Indexing”

# ltePSSSIndices

PSSS resource element indices

## Syntax

```
ind = ltePSSSIndices(ue)
ind = ltePSSSIndices(ue,opts)
```

## Description

`ind = ltePSSSIndices(ue)` returns a 124-by-1 complex column vector of resource element (RE) indices for the primary sidelink synchronization signal (PSSS) values for user equipment settings in the `ue` structure. By default, the indices are returned in one-based linear indexing form. You can use this form to directly index elements of a matrix representing the subframe resource grid for antenna port 1020. For more information, see “Primary Sidelink Synchronization Signal Indexing” on page 2-809.

`ind = ltePSSSIndices(ue,opts)` formats the returned indices using options specified by `opts`.

## Examples

### Generate PSSS Indices

Generate PSSS values and indices. Write the values into the PSSS resource elements in a synchronization subframe (normal cyclic prefix) and display an image of their locations.

Create a user equipment settings structure and a resource grid that has a 10 MHz bandwidth and normal cyclic prefix.

```
ue.NSLRB = 50;
ue.CyclicPrefixSL = 'Normal';
ue.NSLID = 1;
subframe = lteSLResourceGrid(ue);
```

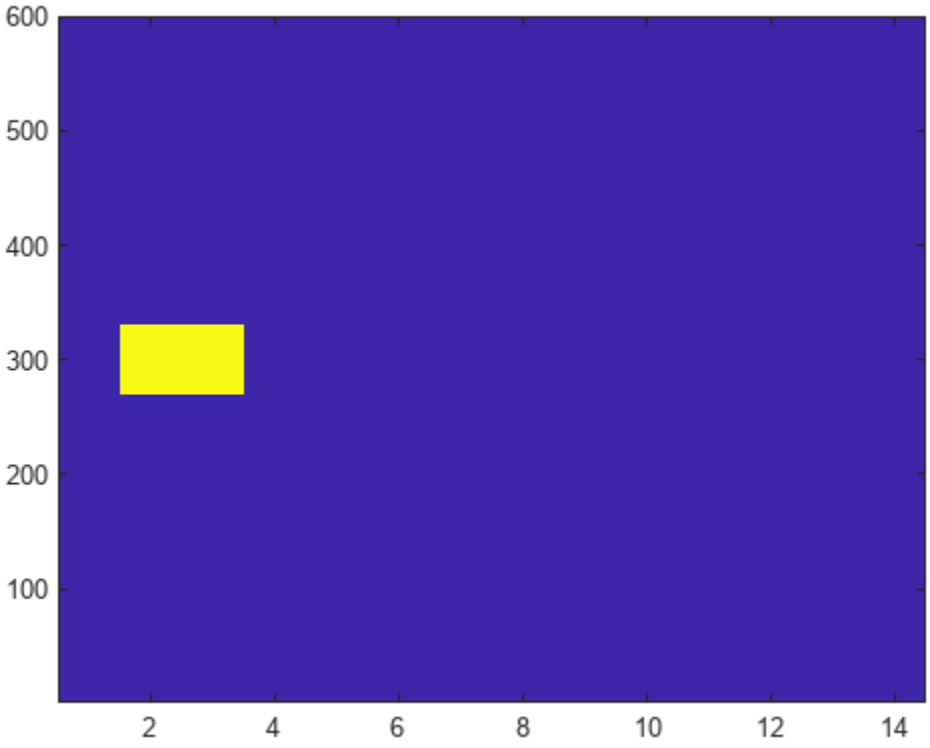
Generate PSSS indices and display the first five indices. Load the PSSS symbols into the resource grid. Display an image showing the PSSS symbol locations.

```
psss_indices = ltePSSSIndices(ue);
psss_indices(1:5)
```

```
ans = 5x1 uint32 column vector

   870
   871
   872
   873
   874
```

```
subframe(psss_indices) = ltePSSS(ue);
```

```
imagesc(100*abs(subframe));
axis xy;
```




### Generate Zero-Based PSSS Indices

Generate PSSS indices using zero-based indexing style. Compare these indices to one-based indices.

Create a user equipment settings structure and a resource grid that has a 10 MHz bandwidth and normal cyclic prefix.

```
ue.NSLRB = 50;
ue.CyclicPrefixSL = 'Normal';
ue.NSLID = 1;
subframe = lteSLResourceGrid(ue);
```

Generate PSSS zero-based indices and view the first five indices.

```
psss_indices = ltePSSSIndices(ue,'0based');
psss_indices_size = size(psss_indices)
```

```
psss_indices_size = 1×2

   124     1
```

```
psss_indices(1:5)
```

```
ans = 5x1 uint32 column vector

   869
   870
   871
   872
   873
```

Generate PSSS one-based indices and view the first five indices.

```
psss_indices = ltePSSSIndices(ue,'1based');
psss_indices_size = size(psss_indices)
```

```
psss_indices_size = 1×2

   124     1
```

```
psss_indices(1:5)
```

```
ans = 5x1 uint32 column vector

   870
   871
   872
   873
   874
```

For zero-based indexing, the first assigned index is one lower than the one-based indexing style.

## Input Arguments

### ue — User equipment settings

structure

UE-specific settings, specified as a structure containing these parameter fields:

#### NSLRB — Number of sidelink resource blocks

integer scalar from 6 to 110

Number of sidelink resource blocks, specified as an integer scalar from 6 to 110.

Example: 6, which corresponds to a channel bandwidth of 1.4 MHz.

Data Types: `double`

#### CyclicPrefixSL — Cyclic prefix length

`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

### opts — Output format options for resource element indices

character vector | cell array of character vectors | string array

Output format options for resource element indices, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string) :

| Category | Options | Description |
|---|---|---|
| Indexing style | `'ind'` (default) | The returned indices are in linear index style. |
| | `'sub'` | The returned indices are in `[subcarrier,symbol,port]` subscript row style. |
| Index base | `'1based'` (default) | The returned indices are one-based. |
| | `'0based'` | The returned indices are zero-based. |

Example: `'ind 1based'`, `"ind 1based"`, `{'ind','1based'}`, or `["ind","1based"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `ind` — PSSS resource element indices

integer column vector | three column integer matrix

PSSS resource element indices, returned as an integer column vector or a three-column integer matrix. This output is generated using the UE-settings structure, `ue`. For more information, see “Primary Sidelink Synchronization Signal Indexing” on page 2-809.

## More About

### Primary Sidelink Synchronization Signal Indexing

Use the indexing function, `ltePSSSIndices`, and the corresponding sequence function, `ltePSSS`, to populate the resource grid for the desired subframe number. The PSSS values are output by `ltePSSS`, ordered as they should be mapped, applying frequency-first mapping into the resource elements of the adjacent symbols using `ltePSSSIndices`. When indexing is zero-based, the SC-FDMA symbols used are {1,2} for normal cyclic prefix and {0, 1} for extended cyclic prefix.

**Note** The indicated symbol indices are based on TS 36.211, Section 9.7. However to align with the LTE Toolbox subframe orientation, these indices are expanded from symbol index per slot to symbol index per subframe.

For more information on mapping symbols to the resource element grid, see “Resource Grid Indexing”.

### Primary Sidelink Synchronization Signal

The primary sidelink synchronization signal (PSSS) is transmitted in the central 62 resource elements of two adjacent SC-FDMA symbols in a synchronization subframe. The same sequence of 62 complex values is repeated in each of the symbols, resulting in a 124-by-1 element vector returned by the

`ltePSSS` function. The values of this sequence are ordered as they should be mapped into the resource elements of the adjacent symbols using `ltePSSSIndices`. If a terminal is transmitting PSSS, then the PSSS should be sent every 40 ms with the exact subframe dependent on the RRC signaled subframe number offset (syncOffsetIndicator-r12).

The PSSS is sent on antenna port 1020, along with the secondary sidelink synchronization signal (SSSS). A synchronization subframe also contains the PSBCH, which is sent on antenna port 1010. The transmission power of the PSSS symbols should be the same as the PSBCH therefore they should be scaled by $\sqrt{72/62}$ in a subframe. No PSCCH or PSSCH transmission will occur in a sidelink subframe configured for synchronization purposes.

As specified in TS 36.211, Section 9.7, the PSSS identity assignment depends on the network coverage. The set of all $N_{\mathrm{ID}}^{\mathrm{SL}}$ is divided into two sets, id_net {0, ..., 167} and id_oon {168, ..., 335}, which are used by terminals that are in-network and out-of-network coverage, respectively. The sidelink physical layer cell identity number, $N_{\mathrm{ID}}^{\mathrm{SL}}$, corresponds to the `ltePSSS` input UE settings structure field `ue.NSLID`. Within each set, all identities result in the same PSSS. For an in-network terminal, the `ue.NSLID` value corresponds to the RRC sidelink synchronization signal identity (slssid-r12) associated with the cell.

# Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePSSS` | `lteSSSSIndices` | `ltePSSIndices`

**Topics**

“Resource Grid Indexing”

# ltePUCCH1

Physical uplink control channel format 1

## Syntax

```
sym = ltePUCCH1(ue,chs,ack)
[sym,info] = ltePUCCH1(ue,chs,ack)
```

## Description

`sym = ltePUCCH1(ue,chs,ack)` returns a matrix containing physical uplink control channel (PUCCH) format 1 symbols given a structure of UE-specific settings, a structure of channel transmission configuration settings, and hybrid ARQ (HARQ) indicator values.

If the configured PUCCH resource indices match indices configured for a scheduling request (SR), as specified in TS 36.213 [1], Section 10.1.5, you can also use this function to generate an SR.

`[sym,info] = ltePUCCH1(ue,chs,ack)` also returns a PUCCH information structure array, `info`.

## Examples

### Generate PUCCH Format 1 Symbols

Generate the PUCCH format 1 symbols for UE-specific settings.

```
ue.NCellID = 1;
ue.NSubframe = 0;
chs.ResourceIdx = 0;
pucch1Sym = ltePUCCH1(ue,chs,[]);
```

### Generate PUCCH Format 1 Symbols for Two Antennas

Generate the physical uplink control channel (PUCCH) format 1 symbols for two transmit antenna paths.

Initialize parameters for a UE-specific configuration structure and a channel configuration structure. Generate PUCCH1 symbols and information outputs.

```
ue.NCellID = 1;
ue.NSubframe = 0;
ue.CyclicPrefixUL = 'Normal';
ue.Hopping = 'Off';

chs.ResourceIdx = [0 3];
chs.DeltaShift = 1;
chs.CyclicShifts = 0;
chs.Shortened = 0;
```

```
[pucch1Sym,info] = ltePUCCH1(ue,chs,[]);
```

Because there are two antennas, the symbols are output as a two-column vector, and the `info` output structure contains two elements.

```
pucch1Sym(1:10,:)
```

```
ans = 10×2 complex

   0.5000 + 0.5000i  -0.5000 + 0.5000i
  -0.6830 + 0.1830i   0.6830 - 0.1830i
   0.6830 + 0.1830i   0.1830 - 0.6830i
  -0.5000 + 0.5000i  -0.5000 + 0.5000i
  -0.1830 - 0.6830i   0.6830 - 0.1830i
  -0.6830 - 0.1830i   0.6830 + 0.1830i
   0.5000 + 0.5000i   0.5000 - 0.5000i
   0.6830 - 0.1830i   0.6830 - 0.1830i
  -0.1830 + 0.6830i  -0.6830 - 0.1830i
   0.5000 + 0.5000i  -0.5000 - 0.5000i
```

```
size(info)
```

```
ans = 1×2

     1     2
```

View the contents of the `info` structure elements.

```
info(2)
```

```
ans = struct with fields:
               Alpha: [3.6652 2.0944 4.1888 1.0472 5.2360 2.0944 2.6180 4.1888]
            SeqGroup: [1 1]
              SeqIdx: [0 0]
         NResourceIdx: [3 11]
    NCellCyclicShift: [64 193 89 191 71 101 234 105]
          OrthSeqIdx: [0 0]
             Symbols: [1.0000 + 0.0000i 1.0000 + 0.0000i 1.0000 + 0.0000i 1.0000 + 0.0000i 1.000
             OrthSeq: [4x2 double]
           ScrambSeq: [0.0000 + 1.0000i 0.0000 + 1.0000i]
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific configuration settings, specified as a structure containing these fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NCellID`** | Required | Integer from 0 to 503 | Physical layer cell identity |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NSubframe`** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **`CyclicPrefixUL`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **`Hopping`** | Optional | `'Off'` (default), `'Group'` | Frequency hopping method. |
| **`Shortened`** | Optional | 0 (default), 1 | Option to shorten the subframe by omitting the last symbol, specified as 0 or 1. If `1`, the last symbol of the subframe is not used. For subframes with possible SRS transmission, set `Shortened` to 1 to maintain a standard compliant configuration. |

Data Types: `struct`

**`chs` — Channel transmission configuration**
structure

Channel transmission configuration, specified as a structure containing these fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`ResourceIdx`** | Optional | 0 (default), integer from 0 to 2047 or vector of integers. | PUCCH resource indices, specified as an integer or a vector of integers. Values range from 0 to 2047. These indices determine the physical resource blocks, cyclic shift and orthogonal cover used for transmission. ($n^{(1)}_{PUCCH}$). Define one index for each transmission antenna. |
| **`DeltaShift`** | Optional | 1 (default), 2, 3 | Delta shift, specified as 1, 2, or 3. ($\Delta_{\text{shift}}$) |
| **`DeltaOffset`** | Optional | 0 (default), 1, 2 | ($\Delta_{\text{offset}}$). Warning: The use of this parameter field is not advised. It applies only to 3GPP releases preceding v8.5.0. This parameter will be removed in a future release. |
| **`CyclicShifts`** | Optional | 0 (default), integer from 0 to 7 | Number of cyclic shifts used for format 1 in resource blocks (RBs) with a mixture of format 1 and format 2 PUCCH, specified as an integer from 0 to 7. ($N^{(1)}_{cs}$) |

**`ack` — Hybrid ARQ indicator values**
nonnegative integer vector containing 0, 1 or 2 elements

Hybrid ARQ indicator values, specified as a nonnegative integer vector. This vector is expected to be the block of bits $b(0),...,b(M_{bit}-1)$ specified in TS 36.211 [2], Section 5.4.1. An $M_{bit}$ value of 0, 1, or 2 corresponds to PUCCH format 1, 1a, or 1b, respectively, as described in TS 36.211 [2], Table 5.4-1.

Example: [ ] indicates that no HARQ are transmitted in the subframe.

## Output Arguments

### `sym` — PUCCH format 1 symbols

numeric column vector

PUCCH format 1 symbols, returned as a numeric column vector. The symbols for each antenna are in the columns of `sym`, with the number of columns determined by the number of PUCCH resource indices specified in `chs.ResourceIdx`.

Example: [0.7071 + 0.7071i,...]

### `info` — PUCCH format 1 resource information

structure array

PUCCH format 1 resource information, returned as a structure array with elements corresponding to each transmit antenna and containing these fields.

### `Alpha` — Reference signal cyclic shift for each OFDM symbol

two-column vector

Reference signal cyclic shift for each OFDM symbol, returned as a two-column vector. ($\alpha$)

### `SeqGroup` — PUCCH base sequence group number for each slot

two-column vector

PUCCH base sequence group number for each slot, returned as a two-column vector. ($u$)

### `SeqIdx` — PUCCH base sequence group number indices

two-column vector

PUCCH base sequence group number indices for each slot, returned as a two-column vector. ($v$)

### `NResourceIdx` — PUCCH resource indices for each slot

two-column vector

PUCCH resource indices for each slot, returned as a two-column vector. ($n'$)

### `NCellCyclicShift` — Cell-specific cyclic shift for each OFDM symbol

vector

Cell-specific cyclic shift for each OFDM symbol, returned as a vector. ($n_{cs}^{cell}$)

### `OrthSeqIdx` — Orthogonal sequence index for each slot

vector

Orthogonal sequence index for each slot, returned as a two-element vector. ($n_{oc}$)

### `Symbols` — Modulated data symbols for each OFDM symbol

vector

Modulated data symbols for each OFDM symbol, returned as a vector. (*d(0)*)

Example: [0.7071 + 0.7071i,...]

**`OrthSeq` — Orthogonal sequence of each slot**
numeric matrix

Orthogonal sequence of each slot, returned as a numeric matrix. Each column in the matrix contains the orthogonal sequence ($w_{n_{oc}}$) for each slot.

**Note** When `ue.Shortened` is 1, the transmission is shortened and the second column of `info.OrthSeq` has a 0 in the last row. This 0 value occurs because, in this case, the spreading factor for the second slot is 3 rather than 4.

Example: [1.000 + 1.000i,...]

**`ScrambSeq` — Scrambling value**
two-element vector

Scrambling value for each slot (*S*), returned as two-element vector.

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.213. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePUCCH1Decode` | `ltePUCCH1Indices` | `ltePUCCH1DRS` | `ltePUCCH1DRSIndices` | `ltePUCCH2` | `ltePUCCH3`

# ltePUCCH1Decode

Physical uplink control channel format 1 decoding

## Syntax

```
ack = ltePUCCH1Decode(ue,chs,oack,sym)
```

## Description

`ack = ltePUCCH1Decode(ue,chs,oack,sym)` returns a vector of hybrid-ARQ (HARQ) indicator values, `ack`, obtained by performing PUCCH Format 1 decoding of the complex matrix `sym`. The decoder uses a maximum likelihood (ML) approach, assuming that `sym` has already been equalized to best restore the original transmitted complex values. The symbols for each antenna are in the columns of `sym`. The number of columns in `sym` should match the number of PUCCH resource indices specified in the structure `chs`.

The output argument `ack` is a vector containing `oack` hybrid-ARQ indicator values.

## Examples

### Decode PUCCH Format 1B Symbols

Decoding of a PUCCH Format 1b received symbol vector `pucch1Sym`.

Initialize a UE-specific configuration structure (`ue`), channel configuration structure (`chs`) and ACK vector (`txAck`)

```
ue.NCellID = 0;
ue.NSubframe = 0;
ue.CyclicPrefixUL = 'Normal';
ue.Hopping = 'Off';
ue.Shortened = 0;

chs.DeltaShift = 1;
chs.ResourceIdx = 0;
chs.CyclicShifts = 0;

txAck = [0;1];
```

Generate PUCCH symbols. Then decode the symbols and verify that the Rx ACK vector matches the Tx ACK vector.

```
pucch1Sym = ltePUCCH1(ue,chs,txAck);

rxAck = ltePUCCH1Decode(ue,chs,length(txAck),pucch1Sym)
```

```
rxAck = 2x1 logical array

   0
   1
```

## Input Arguments

### `ue` — UE-specific settings

structure

UE-specific configuration settings, specified as a structure that can contain the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NCellID`** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **`NSubframe`** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **`CyclicPrefixUL`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **`Hopping`** | Optional | `'Off'` (default), `'Group'` | Frequency hopping method. |
| **`Shortened`** | Optional | 0 (default), 1 | Option to shorten the subframe by omitting the last symbol, specified as 0 or 1. If `1`, the last symbol of the subframe is not used. For subframes with possible SRS transmission, set `Shortened` to 1 to maintain a standard compliant configuration. |

Data Types: `struct`

### `chs` — Channel transmission configuration

structure

PUCCH channel settings, specified as a structure that can contain the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`ResourceIdx`** | Optional | 0 (default), integer from 0 to 2047 or vector of integers. | PUCCH resource indices, specified as an integer or a vector of integers. Values range from 0 to 2047. These indices determine the physical resource blocks, cyclic shift and orthogonal cover used for transmission. ($n^{(1)}_{PUCCH}$). Define one index for each transmission antenna. |
| **`DeltaShift`** | Optional | 1 (default), 2, 3 | Delta shift, specified as 1, 2, or 3. ($\Delta_{\text{shift}}$) |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`DeltaOffset`** | Optional | 0 (default), 1, 2 | ($\Delta_{\text{offset}}$). Warning: The use of this parameter field is not advised. It applies only to 3GPP releases preceding v8.5.0. This parameter will be removed in a future release. |
| **`CyclicShifts`** | Optional | 0 (default), integer from 0 to 7 | Number of cyclic shifts used for format 1 in resource blocks (RBs) with a mixture of format 1 and format 2 PUCCH, specified as an integer from 0 to 7. ($N_{cs}^{(1)}$) |

Data Types: `struct`

**`oack` — Number of uncoded HARQ-ACK bits**
0 (default) | nonnegative integer vector

Uncoded HARQ-ACK bits, specified as a nonnegative integer vector. `oack` specifies the number of Hybrid ARQ indicator values expected: 1 (PUCCH Format 1a) or 2 (PUCCH Format 1b).

Data Types: `double`

**`sym` — Symbols of each antenna**
complex numeric matrix

Symbols for each antenna, specified as complex numeric matrix. The number of columns in `sym` should match the number of PUCCH resource indices specified in the structure, `chs`.

Example: 0.25881 + 0.9659i

Data Types: `double`
Complex Number Support: Yes

## Output Arguments

**`ack` — Hybrid ARQ indicator values**
logical column vector or matrix

`oack` Hybrid ARQ indicator values, specified as a logical column vector or matrix. This vector is obtained by performing PUCCH Format 1 decoding of the complex matrix, `sym`. A Scheduling Request (SR), which is transmitted on PUCCH Format 1 (no ACK bits), can be detected by setting `oack` = 1; in this case the received Hybrid ARQ indicator value, `ack`, is expected to be zero.

If multiple decoded Hybrid ARQ indicator vectors have a likelihood equal to the maximum, `ack` is a matrix where each column represents one of the equally likely Hybrid ARQ indicator vectors. If a minimum likelihood threshold is not met, `ack` is empty.

Data Types: `logical`

# Version History

**Introduced in R2014a**

## See Also

`ltePUCCH1` | `ltePUCCH1Indices` | `ltePUCCH1DRS` | `ltePUCCH1DRSIndices` | `ltePUCCH2Decode` | `ltePUCCH3Decode`

# ltePUCCH1DRS

PUCCH format 1 demodulation reference signal

## Syntax

```
seq = ltePUCCH1DRS(ue,chs)
[seq,info] = ltePUCCH1DRS(ue,chs)
```

## Description

`seq = ltePUCCH1DRS(ue,chs)` returns a matrix containing demodulation reference signal (DRS) associated with PUCCH format 1 transmission, given structures containing the UE-specific settings, and the channel transmission configuration settings.

`[seq,info] = ltePUCCH1DRS(ue,chs)` also returns a PUCCH information structure array, `info`.

## Examples

### Generate PUCCH Format 1 DM-RS

Generate PUCCH format 1 DM-RS values for UE-specific settings.

Initialize UE-specific and channel configuration structures. Generate PUCCH format 1 DM-RS values.

```
ue.NCellID = 1;
ue.NSubframe = 0;
ue.CyclicPrefixUL = 'Normal';
ue.Hopping = 'Off';

chs.ResourceIdx = 0;
chs.DeltaShift = 1;
chs.CyclicShifts = 0;

drsSeq = ltePUCCH1DRS(ue,chs);
```

### Generate PUCCH Format 1 DM-RS Using Virtual Cell ID

Demonstrate Uplink Release 11 coordinated multipoint (CoMP) operation. Intercell interference can be avoided by using a virtual cell identity and a distinct DM-RS cyclic shift hopping identity for a potentially interfering UE in a neighboring cell.

Configuration for UE of interest, UE 1 in cell 1.

```
ue1.NCellID = 1;
ue1.NSubframe = 0;
ue1.CyclicPrefixUL = 'Normal';
ue1.Hopping = 'Off';
```

```
chs1.ResourceIdx = 0;
chs1.DeltaShift = 1;
chs1.CyclicShifts = 0;
```

Configuration for interferer, UE 2 in cell 2.

```
ue2.NCellID = 2;
ue2.NSubframe = 0;
ue2.CyclicPrefixUL = 'Normal';
ue2.Hopping = 'Off';

chs2.ResourceIdx = 1;
chs2.DeltaShift = 1;
chs2.CyclicShifts = 0;
```

Measure the interference between the DM-RS signals.

```
interferenceNoCoMP = abs(sum(ltePUCCH1DRS(ue1,chs1).*conj(ltePUCCH1DRS(ue2,chs2))))
```

```
interferenceNoCoMP = 2.0706
```

Reconfigure interferer for CoMP operation: use virtual cell identity equal to the cell identity for the UE of interest.

```
ue2.NPUCCHID = ue1.NCellID;
```

Measure the interference between the DM-RS signals when using CoMP:

```
interferenceUsingCoMP = abs(sum(ltePUCCH1DRS(ue1,chs1).*conj(ltePUCCH1DRS(ue2,chs2))))
```

```
interferenceUsingCoMP = 2.3645e-14
```

Compare the correlations between the DM-RS signals for two UEs with and without CoMP, `interferenceUsingCoMP` and `interferenceNoCoMP` respectively. Using CoMP, the interference is reduced to effectively zero.

### Generate PUCCH Format 1 DM-RS for Two Antennas

Generate the PUCCH format 1 DM-RS for two transmit antenna paths.

Initialize UE-specific and channel configuration structures. Generate PUCCH1 DM-RS and information outputs.

```
ue.NCellID = 1;
ue.NSubframe = 0;
ue.CyclicPrefixUL = 'Normal';
ue.Hopping = 'Off';

chs.ResourceIdx = [0 3];
chs.DeltaShift = 1;
chs.CyclicShifts = 0;

[drsSeq,info] = ltePUCCH1DRS(ue,chs);
```

Because there are two antennas, the DM-RS sequences are output as a two-column vector and the `info` output structure contains two elements. View `ind` and the size of `info` to confirm this.

```
drsSeq(1:10,:)

ans = 10×2 complex

   0.5000 + 0.5000i   0.5000 + 0.5000i
   0.5000 + 0.5000i  -0.5000 + 0.5000i
  -0.5000 + 0.5000i   0.5000 - 0.5000i
  -0.5000 + 0.5000i   0.5000 + 0.5000i
  -0.5000 + 0.5000i  -0.5000 + 0.5000i
   0.5000 - 0.5000i   0.5000 + 0.5000i
   0.5000 + 0.5000i  -0.5000 - 0.5000i
  -0.5000 - 0.5000i  -0.5000 + 0.5000i
  -0.5000 - 0.5000i  -0.5000 - 0.5000i
   0.5000 + 0.5000i  -0.5000 + 0.5000i


size(info)

ans = 1×2

     1     2
```

View the contents of the two `info` structure elements.

```
info(1)

ans = struct with fields:
               Alpha: [0 5.2360 4.1888 4.7124 1.0472 1.5708]
            SeqGroup: [1 1]
              SeqIdx: [0 0]
        NResourceIdx: [0 2]
    NCellCyclicShift: [192 46 212 91 84 25]
          OrthSeqIdx: [0 0]
             Symbols: [1.0000 + 0.0000i 1.0000 + 0.0000i 1.0000 + 0.0000i 1.0000 + 0.0000i 1.0000
             OrthSeq: [3x2 double]


info(2)

ans = struct with fields:
               Alpha: [1.5708 0.5236 5.7596 3.1416 5.7596 0]
            SeqGroup: [1 1]
              SeqIdx: [0 0]
        NResourceIdx: [3 11]
    NCellCyclicShift: [192 46 212 91 84 25]
          OrthSeqIdx: [0 0]
             Symbols: [1.0000 + 0.0000i 1.0000 + 0.0000i 1.0000 + 0.0000i 1.0000 + 0.0000i 1.0000
             OrthSeq: [3x2 double]
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific configuration settings, specified as a structure containing these fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NCellID`** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **`NSubframe`** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **`CyclicPrefixUL`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **`Hopping`** | Optional | `'Off'` (default), `'Group'` | Frequency hopping method. |
| **`NPUCCHID`** | Optional | `NCellID` (default)<br><br>Integer from 0 to 503 | PUCCH virtual cell identity. If this field is not present, `NCellID` is used as the identity. |

**`chs` — Channel transmission configuration**
structure

PUCCH channel settings, specified as a structure containing the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`ResourceIdx`** | Optional | 0 (default), integer from 0 to 2047 or vector of integers. | PUCCH resource indices, specified as an integer or a vector of integers. Values range from 0 to 2047. These indices determine the physical resource blocks, cyclic shift and orthogonal cover used for transmission. ($n^{(1)}_{PUCCH}$). Define one index for each transmission antenna. |
| **`DeltaShift`** | Optional | 1 (default), 2, 3 | Delta shift, specified as 1, 2, or 3. ($\Delta_{\text{shift}}$) |
| **`DeltaOffset`** | Optional | 0 (default), 1, 2 | ($\Delta_{\text{offset}}$). Warning: The use of this parameter field is not advised. It applies only to 3GPP releases preceding v8.5.0. This parameter will be removed in a future release. |
| **`CyclicShifts`** | Optional | 0 (default), integer from 0 to 7 | Number of cyclic shifts used for format 1 in resource blocks (RBs) with a mixture of format 1 and format 2 PUCCH, specified as an integer from 0 to 7. ($N^{(1)}_{cs}$) |

## Output Arguments

**`seq` — PUCCH format 1 DRS values**
numeric matrix

PUCCH format 1 DRS values, returned as a numeric matrix. The symbols for each antenna are in the columns of `seq`, with the number of columns determined by the number of PUCCH resource indices specified in `chs.ResourceIdx`.

Example: [0.707+0.707i,...]

**`info` — PUCCH format 1 DRS information**
structure array

PUCCH format 1 DRS information, returned as a structure array with elements corresponding to each transmit antenna and containing these fields.

**`Alpha` — Reference signal cyclic shift for each OFDM symbol**
two-column vector

Reference signal cyclic shift for each OFDM symbol, returned as a two-column vector. ($\alpha$)

**`SeqGroup` — PUCCH base sequence group number for each slot**
two-column vector

PUCCH base sequence group number for each slot, returned as two-column vector. ($u$)

**`SeqIdx` — PUCCH base sequence number for each slot**
two-column vector

PUCCH base sequence number for each slot, returned as two-column vector. ($v$)

**`NResourceIdx` — PUCCH resource indices for each slot**
vector

PUCCH resource indices for each slot, returned as two-column vector. ($n'$)

**`NCellCyclicShift` — Cell-specific cyclic shift for each OFDM symbol**
vector

Cell-specific cyclic shift for each OFDM symbol, returned as vector. ($n_{cs}^{cell}$)

**`OrthSeqIdx` — Orthogonal sequence index for each slot**
two-column vector

Orthogonal sequence index for each slot, returned as two-column vector. ($\bar{n}_{\mathrm{oc}}$)

**`Symbols` — Modulated data symbols**
vector

Modulated data symbols, returned as a vector. There is one element for each OFDM symbol. ($z$)

Example: [0.7071 + 0.7071i,...]

**`OrthSeq` — Orthogonal sequence for each slot**
numeric matrix

Orthogonal sequence for each slot, returned as a numeric matrix. ($\bar{w}$)

Example: [1.000 + 1.000i,...]

## Version History

**Introduced in R2014a**

## See Also

`ltePUCCH1` | `ltePUCCH1Decode` | `ltePUCCH1Indices` | `ltePUCCH1DRSIndices` | `ltePUCCH2DRS` | `ltePUCCH3DRS`

# ltePUCCH1DRSIndices

PUCCH format 1 DRS resource element indices

## Syntax

```
ind = ltePUCCH1DRSIndices(ue,chs)
[ind,info] = ltePUCCH1DRSIndices(ue,chs)
[___] = ltePUCCH1DRSIndices(ue,chs,opts)
```

## Description

`ind = ltePUCCH1DRSIndices(ue,chs)` returns a matrix of resource element indices for the demodulation reference signal (DRS) associated with PUCCH format 1 transmission given structures containing the UE-specific settings, and the channel transmission configuration settings.

`[ind,info] = ltePUCCH1DRSIndices(ue,chs)` also returns a PUCCH information structure array, `info`.

`[___] = ltePUCCH1DRSIndices(ue,chs,opts)` formats the returned indices using options specified by `opts`.

This syntax supports output options from prior syntaxes.

## Examples

### Generate PUCCH Format 1 DM-RS Indices

Generate PUCCH format 1 DM-RS RE indices for a 1.4 MHz bandwidth and PUCCH resource index 0. Use default values for all other parameters.

Initialize UE-specific and channel configuration structures. Generate PUCCH format 1 DM-RS indices.

```
ue.NULRB = 6;
ue.CyclicPrefixUL = 'Normal';

chs.ResourceIdx = 0;
chs.DeltaShift = 1;
chs.CyclicShifts = 0;
chs.ResourceSize = 0;

ind = ltePUCCH1DRSIndices(ue,chs);
ind(1:4)
```

```
ans = 4x1 uint32 column vector

   145
   146
   147
   148
```

### Generate PUCCH Format 1 DM-RS Indices for Two Antennas

Generate the PUCCH format 1 DM-RS indices for two transmit antenna paths.

Initialize UE-specific and channel configuration structures. Generate PUCCH1 DRS indices and information outputs.

```
ue.NULRB = 6;
ue.CyclicPrefixUL = 'Normal';

chs.ResourceIdx = [0 4];
chs.ResourceSize = 0;
chs.DeltaShift = 1;
chs.CyclicShifts = 0;

[ind,info] = ltePUCCH1DRSIndices(ue,chs);
```

Because there are two antennas, the DM-RS indices are output as a two-column vector, and the `info` output structure contains two elements. View `ind` and the size of `info` to confirm this.

```
ind(1:6,:)
```

```
ans = 6x2 uint32 matrix

    145   1153
    146   1154
    147   1155
    148   1156
    149   1157
    150   1158
```

```
size(info)
```

```
ans = 1×2

     1     2
```

View the contents of the two `info` structure elements.

```
info(1)
```

```
ans = struct with fields:
    PRBSet: [0 5]
     RBIdx: 0
```

```
info(2)
```

```
ans = struct with fields:
    PRBSet: [0 5]
     RBIdx: 0
```

### Generate PUCCH Format 1 DM-RS Indices for Two Antennas Varying Indexing Style

Generate the PUCCH format 1 DM-RS indices for two transmit antenna paths, and output in subscript indexing form.

Initialize UE-specific and channel configuration structures, and the indexing option parameter. Generate PUCCH1 DM-RS indices and information outputs.

```
ue.NULRB = 6;
ue.CyclicPrefixUL = 'Normal';

chs.ResourceIdx = [0 4];
chs.ResourceSize = 0;
chs.DeltaShift = 1;
chs.CyclicShifts = 0;

opts = {'sub'};

[ind,info] = ltePUCCH1DRSIndices(ue,chs,opts);
```

Using 'sub' indexing style, the indices are output in [subcarrier, symbol, antenna] subscript form. View the midpoint of ind and observe the antenna index change.

```
size(ind)
```

```
ans = 1×2

   144     3
```

```
ind(70:74,:)
```

```
ans = 5x3 uint32 matrix

   70   12    1
   71   12    1
   72   12    1
    1    3    2
    2    3    2
```

```
size(info)
```

```
ans = 1×2

     1     2
```

Because there are two antennas, the info output structure contains two elements. View one of the `info` structure elements.

```
info(1)
```

```
ans = struct with fields:
    PRBSet: [0 5]
     RBIdx: 0
```

## Input Arguments

### `ue` — UE-specific settings

structure

UE-specific settings, specified as a structure containing these fields.

### `NULRB` — Number of uplink resource blocks

integer from 6 to 110

Number of uplink resource blocks, specified as an integer from 6 to 110.

Data Types: `double`

### `CyclicPrefixUL` — Cyclic prefix length for uplink channels

`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length for uplink channels, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

### `chs` — Channel transmission configuration

structure

Channel transmission configuration, specified as a structure containing these fields.

### `ResourceIdx` — PUCCH resource indices

0 (default) | 0,...,2047 | integer | vector of integers | optional

PUCCH resource indices, specified as an integer or a vector of integers. Values range from 0 to 2047. These indices determine the physical resource blocks, cyclic shift and orthogonal cover used for transmission. ($n_{PUCCH}^{(1)}$). Define one index for each transmission antenna.

Data Types: `double`

### `ResourceSize` — Size of resources allocated to PUCCH format 2

0 (default) | 0,...,98 | integer | optional

Size of resources allocated to PUCCH format 2, specified as an integer from 0 to 98. This parameter affects the location of this transmission. ($N_{RB}^{(2)}$)

Data Types: `double`

### `DeltaShift` — Delta shift

1 (default) | 2 | 3 | optional

Delta shift, specified as 1, 2, or 3. ($\Delta_{\text{shift}}$)

Data Types: `double`

### `CyclicShifts` — Number of cyclic shifts used for format 1

0 (default) | optional | 0,...,7 | integer

Number of cyclic shifts used for format 1 in resource blocks (RBs) with a mixture of format 1 and format 2 PUCCH, specified as an integer from 0 to 7. ($N_{cs}^{(1)}$)

Data Types: `double`

Data Types: `struct`

#### `opts` — Output format options for resource element indices

character vector | cell array of character vectors | string array

Output format options for resource element indices, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string) :

| Category | Options | Description |
|---|---|---|
| Indexing style | `'ind'` (default) | The returned indices are in linear index style. |
| | `'sub'` | The returned indices are in `[subcarrier,symbol,port]` subscript row style. |
| Index base | `'1based'` (default) | The returned indices are one-based. |
| | `'0based'` | The returned indices are zero-based. |

Example: `'ind 1based'`, `"ind 1based"`, `{'ind','1based'}`, or `["ind","1based"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

#### `ind` — Resource element indices

integer column vector | three-column integer matrix

Resource element indices, returned as an integer column vector or a three-column integer matrix. By default the indices are returned in one-based linear indexing form that can directly index elements of a resource matrix. These indices are ordered according to PUCCH format 1 DRS modulation symbol mapping. The `opts` input offers alternative indexing formats. The indices for each antenna are in the columns of `ind`, with the number of columns determined by the number of PUCCH resource indices specified in `chs.ResourceIdx`.

Example: [145,146,147,...]

Data Types: `uint32`

#### `info` — PUCCH format 1 DRS information

structure array

PUCCH format 1 DRS information, returned as a structure array with elements corresponding to each transmit antenna and containing these fields.

#### `PRBSet` — Indices occupied by PRB in each slot of subframe

nonnegative integer vector

Indices occupied by PRB in each slot of the subframe, returned as a nonnegative integer vector. The indices are zero-based.

Example: [0,5]

Data Types: `double`

**`RBIdx` — PUCCH logical resource block index**
nonnegative integer

PUCCH logical resource block index, returned as a nonnegative integer. (*m*)

Data Types: `double`

Data Types: `struct`

# Version History

**Introduced in R2014a**

## See Also

`ltePUCCH1` | `ltePUCCH1Decode` | `ltePUCCH1Indices` | `ltePUCCH1DRS` | `ltePUCCH2DRSIndices` | `ltePUCCH3DRSIndices`

# ltePUCCH1Indices

PUCCH format 1 resource element indices

## Syntax

```
ind = ltePUCCH1Indices(ue,chs)
[ind,info] = ltePUCCH1Indices(ue,chs)
[___] = ltePUCCH1Indices(ue,chs,opts)
```

## Description

`ind = ltePUCCH1Indices(ue,chs)` returns a matrix of resource element (RE) indices for the physical uplink control channel (PUCCH) format 1 transmission, given structures containing the UE-specific settings, and the channel transmission configuration.

`[ind,info] = ltePUCCH1Indices(ue,chs)` also returns a PUCCH information structure array.

`[___] = ltePUCCH1Indices(ue,chs,opts)` formats the returned indices using options specified by `opts`.

This syntax supports output options from prior syntaxes.

## Examples

### Generate PUCCH Format 1 Indices

Generate PUCCH format 1 RE indices for a 1.4 MHz bandwidth, PUCCH resource index 0. Use default values for all other parameters.

Initialize UE-specific and channel configuration structures (`ue` and `chs`). Generate PUCCH format 1 indices (`ind`).

```
ue.NULRB = 6;
ue.CyclicPrefixUL = 'Normal';

chs.ResourceIdx = 0;
chs.DeltaShift = 1;
chs.CyclicShifts = 0;
chs.ResourceSize = 0;
chs.Shortened = 0;

ind = ltePUCCH1Indices(ue,chs);
ind(1:4)
```

```
ans = 4x1 uint32 column vector

   1
   2
   3
   4
```

### Generate PUCCH Format 1 Indices for Three Antennas

Generate the physical uplink control channel (PUCCH) format 1 indices for three transmit antenna paths, and display the information structure output.

Initialize UE-specific and channel configuration structures. Generate PUCCH 1 indices and information outputs.

```
ue.NULRB = 6;
ue.CyclicPrefixUL = 'Normal';
ue.Shortened = 0;

chs.ResourceIdx = [0 129 2];
chs.ResourceSize = 0;
chs.DeltaShift = 1;
chs.CyclicShifts = 0;

[ind,info] = ltePUCCH1Indices(ue,chs);
```

Because there are three antennas, the indices are output as a three-column vector, and the `info` output structure contains three elements. View `ind` and the size of `info` to confirm this.

```
ind(1:5,:)
```

```
ans = 5x3 uint32 matrix

      1   1057   2017
      2   1058   2018
      3   1059   2019
      4   1060   2020
      5   1061   2021
```

```
size(info)
```

```
ans = 1×2

     1     3
```

View the contents of one of the `info` structure elements.

```
info(3)
```

```
ans = struct with fields:
    PRBSet: [0 5]
     RBIdx: 0
```

### Generate PUCCH Format 1 Indices Varying Indexing Style

Generate the physical uplink control channel (PUCCH) format 1 indices for two transmit antenna paths and output in subscript indexing form.

Initialize UE-specific and channel configuration structures (`ue` and `chs`) and the indexing option parameter, `opt`. Generate PUCCH1 indices and information outputs (`ind` and `info`).

```
ue.NULRB = 6;
ue.CyclicPrefixUL = 'Normal';

chs.ResourceIdx = [0 4];
chs.ResourceSize = 0;
chs.DeltaShift = 1;
chs.CyclicShifts = 0;
chs.Shortened = 0;

[ind,info] = ltePUCCH1Indices(ue,chs,{'sub'});
```

Using `'sub'` indexing style, the indices are output in `[subcarrier, symbol, antenna]` subscript form. View the midpoint of `ind` and observe the antenna index change.

```
size(ind)
```

```
ans = 1×2

   192     3
```

```
ind(94:99,:)
```

```
ans = 6x3 uint32 matrix

   70   14    1
   71   14    1
   72   14    1
    1    1    2
    2    1    2
    3    1    2
```

Because there are two antennas, the `info` output structure contains two elements. View the contents of the second `info` structure element.

```
size(info)
```

```
ans = 1×2

     1     2
```

```
info(2)
```

```
ans = struct with fields:
    PRBSet: [0 5]
     RBIdx: 0
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific settings, specified as a structure containing these fields.

**`NULRB` — Number of uplink resource blocks**
nonnegative integer

Number of uplink resource blocks, specified as a nonnegative integer.

Data Types: `double`

**`CyclicPrefixUL` — Cyclic prefix length for uplink channels**
`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length for uplink channels, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

**`Shortened` — Option to shorten the subframe**
0 (default) | 1 | optional

Option to shorten the subframe by omitting the last symbol, specified as 0 or 1. If `1`, the last symbol of the subframe is not used. For subframes with possible SRS transmission, set `Shortened` to 1 to maintain a standard compliant configuration.

Data Types: `struct`

**`chs` — Channel transmission configuration**
structure

Channel transmission configuration, specified as a structure containing these fields.

**`ResourceIdx` — PUCCH resource indices**
0 (default) | 0,...,2047 | integer | vector of integers | optional

PUCCH resource indices, specified as an integer or a vector of integers. Values range from 0 to 2047. There is one index for each transmission antenna. These indices determine the cyclic shift and orthogonal cover used for transmission. ($n^{(1)}_{PUCCH}$)

**`ResourceSize` — Size of resources allocated to PUCCH format 2**
0 (default) | 0,...,98 | integer | optional

Size of resources allocated to PUCCH format 2, specified as an integer from 0 to 98. This parameter affects the location of this transmission. ($N^{(2)}_{RB}$)

**`DeltaShift` — Delta shift**
1 (default) | 2 | 3 | optional

Delta shift, specified as 1, 2, or 3. ($\Delta_{\text{shift}}$)

**`CyclicShifts` — Number of cyclic shifts used for format 1**
0 (default) | 0,...,7 | integer | optional

Number of cyclic shifts used for format 1 in RBs with a mixture of Format 1 and Format 2 PUCCH, specified as an integer from 0 to 7. ($N^{(1)}_{cs}$)

### `opts` — Output format options for resource element indices

character vector | cell array of character vectors | string array

Output format options for resource element indices, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string) :

| Category | Options | Description |
|---|---|---|
| Indexing style | `'ind'` (default) | The returned indices are in linear index style. |
| | `'sub'` | The returned indices are in `[subcarrier,symbol,port]` subscript row style. |
| Index base | `'1based'` (default) | The returned indices are one-based. |
| | `'0based'` | The returned indices are zero-based. |

Example: `'ind 1based'`, `"ind 1based"`, `{'ind','1based'}`, or `["ind","1based"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `ind` — PUCCH format 1 resource element indices

integer column vector | three-column integer matrix

PUCCH format 1 resource element indices, returned as an integer column vector or a three-column integer matrix. By default, the indices are returned in one-based linear indexing form that can directly index elements of a resource matrix. These indices are ordered according to PUCCH format 1 modulation symbol mapping as specified in TS 36.211 [1], Section 5.4. The `opts` input offers alternative indexing formats. The indices for each antenna are in the columns of `ind`, with the number of columns determined by the number of PUCCH resource indices specified in `chs.ResourceIdx`.

Example: [1,2,3,4...]

Data Types: `uint32`

### `info` — PUCCH format 1 information

structure array

PUCCH format 1 information, returned as a structure array with elements corresponding to each transmit antenna and containing these fields.

### `PRBSet` — Set of PRB indices

column vector | two-column matrix

Set of PRB indices, returned as a column vector or two-column matrix corresponding to the resource allocations.

- When returned as a column integer vector, the resource allocation is the same in both slots of the subframe.

- When returned as a two-column integer matrix, the resource allocations can vary for each slot in the subframe.

The PRB indices are zero-based.

Example: [0,5]

Data Types: `double`

**`RBIdx` — PUCCH logical resource block index**
nonnegative integer

PUCCH logical resource block index, returned as a nonnegative integer. (*m*)

Data Types: `double`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePUCCH1` | `ltePUCCH1Decode` | `ltePUCCH1DRS` | `ltePUCCH1DRSIndices` | `ltePUCCH2Indices` | `ltePUCCH3Indices`

# ltePUCCH2

Physical uplink control channel format 2

## Syntax

```
sym = ltePUCCH2(ue,chs,bits)
[sym,info] = ltePUCCH2(ue,chs,bits)
```

## Description

`sym = ltePUCCH2(ue,chs,bits)` returns a matrix containing physical uplink control channel (PUCCH) format 2 symbols given a structure of UE-specific settings, a structure with channel transmission configuration settings, and a vector of coded CQI/PMI or RI `bits`.

`[sym,info] = ltePUCCH2(ue,chs,bits)` also returns a PUCCH information structure array, `info`.

## Examples

### Generate PUCCH Format2 symbols

Generate PUCCH format 2 symbol values, using `NCellID` set to 1 and `NSubframe` set to 0.

Initialize `ue` and `chs` configuration structures. Generate symbols.

```
ue.NCellID = 1;
ue.NSubframe = 0;
ue.RNTI = 1;
ue.CyclicPrefixUL = 'Normal';
ue.Hopping = 'Off';

chs.ResourceIdx = 0;
chs.ResourceSize = 0;
chs.CyclicShifts = 0;

sym = ltePUCCH2(ue,chs,ones(20,1));
sym(1:5)
```

```
ans = 5×1 complex

   0.0000 + 1.0000i
  -0.5000 - 0.8660i
  -0.5000 + 0.8660i
  -0.0000 - 1.0000i
   0.5000 + 0.8660i
```

### Generate PUCCH Format 2 Symbols for Two Antennas

Generate the physical uplink control channel (PUCCH) format 2 symbols for two transmit antenna paths.

Initialize parameters for a UE-specific configuration structure and a channel configuration structure. Generate PUCCH 2 symbols and the information structure.

```
ue.NCellID = 1;
ue.NSubframe = 0;
ue.RNTI = 1;
ue.CyclicPrefixUL = 'Normal';
ue.Hopping = 'Off';

chs.ResourceIdx = [0 3];
chs.ResourceSize = 0;
chs.CyclicShifts = 0;

[pucch2Sym,info] = ltePUCCH2(ue,chs,[]);
```

Because there are two antennas, the symbols are output as a two-column vector, and the `info` output structure contains two elements.

```
pucch2Sym(1:10,:)
```

```
ans = 10×2 complex

   0.5000 + 0.5000i   0.5000 + 0.5000i
  -0.6830 - 0.1830i   0.1830 - 0.6830i
   0.1830 + 0.6830i  -0.1830 - 0.6830i
  -0.5000 - 0.5000i  -0.5000 + 0.5000i
   0.6830 + 0.1830i   0.6830 + 0.1830i
   0.6830 - 0.1830i   0.1830 + 0.6830i
  -0.5000 - 0.5000i   0.5000 + 0.5000i
  -0.6830 - 0.1830i  -0.1830 + 0.6830i
   0.6830 - 0.1830i   0.6830 - 0.1830i
   0.5000 - 0.5000i   0.5000 + 0.5000i
```

```
size(info)
```

```
ans = 1×2

     1     2
```

View the contents of the second `info` structure element.

```
info(2)
```

```
ans = struct with fields:
               Alpha: [4.1888 2.0944 1.0472 0 1.5708 3.1416 1.0472 3.6652 4.1888 2.0944]
            SeqGroup: [1 1]
              SeqIdx: [0 0]
        NResourceIdx: [4 7]
    NCellCyclicShift: [64 192 46 212 191 71 91 84 25 105]
             Symbols: [1.0000 + 0.0000i 1.0000 + 0.0000i 1.0000 + 0.0000i 1.0000 + 0.0000i 1.0000
```

## Input Arguments

### `ue` — UE-specific settings

structure

UE-specific configuration settings, specified as a structure containing these fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NCellID`** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **`NSubframe`** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **`RNTI`** | Required | 0 (default), scalar integer | Radio network temporary identifier (RNTI) value (16 bits) |
| **`CyclicPrefixUL`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **`Hopping`** | Optional | `'Off'` (default), `'Group'` | Frequency hopping method. |

### `chs` — Channel transmission configuration

structure

Channel transmission configuration, specified as a structure containing these fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`ResourceIdx`** | Optional | 0 (default), integer from 0 to 1185 or vector of integers. | PUCCH resource indices which determine the physical resource blocks, cyclic shift, and orthogonal cover used for transmission. ($n^{(2)}_{PUCCH}$). Define one index for each transmission antenna. |
| **`ResourceSize`** | Optional | 0 (default), integer from 0 to 98. | Size of resource allocated to PUCCH format 2 ($N^{(2)}_{RB}$) |
| **`CyclicShifts`** | Optional | 0 (default), integer from 0 to 7 | Number of cyclic shifts used for format 1 in resource blocks (RBs) with a mixture of format 1 and format 2 PUCCH, specified as an integer from 0 to 7. ($N^{(1)}_{cs}$) |

### `bits` — Coded CQI/PMI or RI bits

vector

Coded CQI/PMI or RI bits (coded UCI), specified as a vector that is formed by performing UCI encoding of a bit vector representing the CQI/PMI or RI information fields described in TS 36.212 [2], Section 5.2.3.3. This 20 bit long coded bit vector is denoted block of bits *b(0),...,b(19)* in TS 36.211 [1], Section 5.4.2. If $M_{\text{bit}}$ is 21 or 22, corresponding to PUCCH format 2a or 2b, respectively, as described in TS 36.211 [1], Table 5.4-1, the further bits, $b(20),...,b(M_{bit}-1)$, should be provided as input to the `ltePUCCH2DRS` function for transmission. An $M_{bit}$ value of 20 corresponds to PUCCH format 2, with no additional bits being transmitted on the PUCCH format 2 DRS.

Data Types: `logical` | `double`

## Output Arguments

### `sym` — PUCCH format 2 symbols

numeric column vector

PUCCH format 2 symbols, returned as numeric column vector. The symbols for each antenna are in the columns of `sym`, with the number of columns determined by the number of PUCCH resource indices specified in `chs.ResourceIdx`.

Example: 0.7071 + 0.7071i

Data Types: `double`

### `info` — PUCCH format 2 information

structure array

PUCCH format 2 information, returned as a structure array with elements corresponding to each transmit antenna and containing these fields.

### `Alpha` — Reference signal cyclic shift for each OFDM symbol

two-column vector

Reference signal cyclic shift for each OFDM symbol, returned as a two-column vector. ($\alpha$)

### `SeqGroup` — PUCCH base sequence group number for each slot

two-column vector

PUCCH base sequence group number for each slot, returned as a two-column vector. ($u$)

### `SeqIdx` — PUCCH base sequence group number indices

two-column vector

PUCCH base sequence group number indices for each slot, returned as a two-column vector. ($v$)

### `NResourceIdx` — PUCCH resource indices for each slot

two-column vector

PUCCH resource indices for each slot, returned as a two-column vector. ($n$')

### `NCellCyclicShift` — Cell-specific cyclic shift for each OFDM symbol

vector

Cell-specific cyclic shift for each OFDM symbol, returned as a vector. ($n_{cs}^{cell}$)

### `Symbols` — Modulated data symbols for each OFDM symbol

vector

Modulated data symbols for each OFDM symbol, returned as a vector. (*d(0)*)

Example: [0.7071 + 0.7071i,...]

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePUCCH2Decode` | `ltePUCCH2Indices` | `ltePUCCH2DRS` | `ltePUCCH2DRSIndices` | `lteUCIEncode` | `ltePUCCH1` | `ltePUCCH3`

# ltePUCCH2DRS

PUCCH format 2 demodulation reference signal

## Syntax

```
seq = ltePUCCH2DRS(ue,chs,ack)
[seq,info] = ltePUCCH2DRS(ue,chs,ack)
```

## Description

`seq = ltePUCCH2DRS(ue,chs,ack)` returns a matrix containing demodulation reference signal (DRS) associated with PUCCH format 2 transmission, given a structure of UE-specific settings, a structure with channel transmission configuration settings, and hybrid ARQ (HARQ) indicator values, `ack`.

`[seq,info] = ltePUCCH2DRS(ue,chs,ack)` also returns a PUCCH information structure array, `info`.

## Examples

### Generate PUCCH Format 2 DM-RS

Generate PUCCH Format 2 DM-RS symbols for UE specific settings.

Initialize input configuration structures (`ue` and `chs`). Here no HARQ bits will be sent by inputting an empty `ack` vector. Generate the PUCCH Format 2 DM-RS symbols.

```
ue.NCellID = 1;
ue.NSubframe = 0;
ue.CyclicPrefixUL = 'Normal';
ue.Hopping = 'Off';

chs.ResourceIdx = 0;
chs.ResourceSize = 0;
chs.CyclicShifts = 0;

sym = ltePUCCH2DRS(ue,chs,[]);
```

### Generate PUCCH Format 2 DM-RS Using Virtual Cell ID

Demonstrate Uplink Release 11 coordinated multipoint (CoMP) operation. Intercell interference can be avoided by using a virtual cell identity for a potentially interfering UE in a neighboring cell.

Configuration for UE of interest, UE 1 in cell 1.

```
ue1.NCellID = 1;
ue1.NSubframe = 0;
ue1.CyclicPrefixUL = 'Normal';
```

```
ue1.Hopping = 'Off';

chs1.ResourceIdx = 0;
chs1.ResourceSize = 0;
chs1.CyclicShifts = 0;

ack1 = 0;
```

Configuration for interferer, UE 2 in cell 2.

```
ue2.NCellID = 2;
ue2.NSubframe = 0;
ue2.CyclicPrefixUL = 'Normal';
ue2.Hopping = 'Off';

chs2.ResourceIdx = 1;
chs2.ResourceSize = 0;
chs2.CyclicShifts = 0;

ack2 = 0;
```

Measure the interference between the DM-RS signals.

```
interferenceNoCoMP = abs(sum(ltePUCCH2DRS(ue1,chs1,ack1).*conj(ltePUCCH2DRS(ue2,chs2,ack2))))
```

```
interferenceNoCoMP = 5.4903
```

Reconfigure interferer for CoMP operation: use virtual cell identity equal to the cell identity for the UE of interest.

```
ue2.NPUCCHID = ue1.NCellID;
```

Measure the interference between the DM-RS signals when using CoMP.

```
interferenceUsingCoMP = abs(sum(ltePUCCH2DRS(ue1,chs1,ack1).*conj(ltePUCCH2DRS(ue2,chs2,ack2))))
```

```
interferenceUsingCoMP = 4.2221e-15
```

Comparing the correlations between the DM-RS signals for two UEs with and without CoMP, `interferenceUsingCoMP` and `interferenceNoCoMP` respectively. Using CoMP, the interference is reduced to effectively zero.

### Generate PUCCH Format 2 DM-RS for Two Antennas

Generate the PUCCH format 2 DM-RS sequences for two transmit antenna paths.

Initialize UE-specific and channel configuration structures. Provide an empty vector for the `ack`, indicating there are no HARQ bits for this PUCCH transmission. Generate PUCCH 2 DM-RS and information outputs.

```
ue.NCellID = 1;
ue.NSubframe = 0;
ue.CyclicPrefixUL = 'Normal';
ue.Hopping = 'Off';

chs.ResourceIdx = [0 3];
```

```
chs.ResourceSize = 0;
chs.CyclicShifts = 0;

ack = [];

[drsSeq,info] = ltePUCCH2DRS(ue,chs,ack);
```

Because there are two antennas, the DM-RS sequences are output as a two- column vector, and the `info` output structure contains two elements.

```
drsSeq(1:10,:)
```

```
ans = 10×2 complex

   0.5000 + 0.5000i   0.5000 + 0.5000i
  -0.1830 + 0.6830i  -0.6830 - 0.1830i
  -0.1830 - 0.6830i   0.1830 + 0.6830i
   0.5000 - 0.5000i  -0.5000 - 0.5000i
   0.6830 + 0.1830i   0.6830 + 0.1830i
  -0.1830 - 0.6830i   0.6830 - 0.1830i
   0.5000 + 0.5000i  -0.5000 - 0.5000i
   0.1830 - 0.6830i  -0.6830 - 0.1830i
   0.6830 - 0.1830i   0.6830 - 0.1830i
  -0.5000 - 0.5000i   0.5000 - 0.5000i
```

```
size(info)
```

```
ans = 1×2

     1     2
```

View the contents of the two `info` structure elements.

```
info(1)
```

```
ans = struct with fields:
               Alpha: [1.0472 3.1416 1.5708 2.0944]
            SeqGroup: [1 1]
              SeqIdx: [0 0]
        NResourceIdx: [1 10]
    NCellCyclicShift: [193 89 101 234]
             Symbols: [1.0000 + 0.0000i 1.0000 + 0.0000i 1.0000 + 0.0000i 1.0000 + 0.0000i]
             OrthSeq: [2x2 double]
```

```
info(2)
```

```
ans = struct with fields:
               Alpha: [2.6180 4.7124 0 0.5236]
            SeqGroup: [1 1]
              SeqIdx: [0 0]
        NResourceIdx: [4 7]
    NCellCyclicShift: [193 89 101 234]
             Symbols: [1.0000 + 0.0000i 1.0000 + 0.0000i 1.0000 + 0.0000i 1.0000 + 0.0000i]
             OrthSeq: [2x2 double]
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific configuration settings, specified as a structure that can contain the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NCellID** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **NSubframe** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **CyclicPrefixUL** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **Hopping** | Optional | `'Off'` (default), `'Group'` | Frequency hopping method. |
| **NPUCCHID** | Optional | `NCellID` (default)<br><br>Integer from 0 to 503 | PUCCH virtual cell identity. If this field is not present, `NCellID` is used as the identity. |

Data Types: `struct`

### chs — Channel transmission configuration

structure

PUCCH channel settings, specified as a structure that can contain the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **ResourceIdx** | Optional | 0 (default), integer from 0 to 1185 or vector of integers. | PUCCH resource indices which determine the physical resource blocks, cyclic shift, and orthogonal cover used for transmission. ($n^{(2)}_{PUCCH}$). Define one index for each transmission antenna. |
| **ResourceSize** | Optional | 0 (default), integer from 0 to 98. | Size of resource allocated to PUCCH format 2 ($N^{(2)}_{RB}$) |
| **CyclicShifts** | Optional | 0 (default), integer from 0 to 7 | Number of cyclic shifts used for format 1 in resource blocks (RBs) with a mixture of format 1 and format 2 PUCCH, specified as an integer from 0 to 7. ($N^{(1)}_{cs}$) |

### ack — Hybrid ARQ indicator values

binary vector containing 0, 1 or 2 elements

Hybrid ARQ indicator values, specified as nonnegative integer vector. This vector is expected to be the block of bits $b(0),...,b(M_{\text{bit}}-1)$ specified in TS 36.211 [1], Section 5.4.2. An $M_{\text{bit}}$ value of 20, 21, or 22 corresponds to PUCCH format 2, 2a, or 2b, respectively, as described in TS 36.211 [1], Table 5.4-1.

Example: [ ] indicates that no HARQ are transmitted in the subframe.

## Output Arguments

### `seq` — PUCCH format 2 DRS values

numeric matrix

PUCCH format 2 DRS values, returned as a numeric matrix. The symbols for each antenna are in the columns of `seq`, with the number of columns determined by the number of PUCCH resource indices specified in `chs.ResourceIdx`.

**Note** The standard does not support format 2a or 2b transmission with extended cyclic prefix. If the `ack` setting corresponds to format 2a or 2b transmission and extended cyclic prefix is set for `ue.CyclicPrefixUL`, the function returns an empty matrix for `seq`.

Data Types: `double`

### `info` — PUCCH format 2 information

structure array

PUCCH format 2 information, returned as a structure array with elements corresponding to each transmit antenna and containing these fields. When configured for format 2a or 2b transmission with extended cyclic prefix, the `info` structure contains all fields, but each field is empty.

### `Alpha` — Reference signal cyclic shift for each OFDM symbol

two-column vector

Reference signal cyclic shift for each OFDM symbol, returned as a two-column vector. ($\alpha$)

### `SeqGroup` — PUCCH base sequence group number for each slot

two-column vector

PUCCH base sequence group number for each slot, returned as two-column vector. ($u$)

### `SeqIdx` — PUCCH base sequence number for each slot

two-column vector

PUCCH base sequence number for each slot, returned as two-column vector. ($v$)

### `NResourceIdx` — PUCCH resource indices for each slot

vector

PUCCH resource indices for each slot, returned as two-column vector. ($n'$)

### `NCellCyclicShift` — Cell-specific cyclic shift for each OFDM symbol

vector

Cell-specific cyclic shift for each OFDM symbol, returned as vector. ($n_{cs}^{cell}$)

### `Symbols` — Modulated data symbols

vector

Modulated data symbols, returned as a vector. There is one element for each OFDM symbol. ($z$)

Example: [0.7071 + 0.7071i,...]

**`OrthSeq` — Orthogonal sequence for each slot**
4-by-2 numeric matrix

Orthogonal sequence for each slot, returned as a 4-by-2 numeric matrix. ($\overline{w}$)

Example: [1.000 + 1.000i,...]

Data Types: `struct`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePUCCH2DRSDecode` | `ltePUCCH2DRSIndices` | `ltePUCCH2` | `ltePUCCH2Decode` | `ltePUCCH2Indices` | `ltePUCCH1DRS` | `ltePUCCH3DRS`

# ltePUCCH2DRSDecode

PUCCH format 2 DRS decoding

## Syntax

```
ack = ltePUCCH2DRSDecode(ue,chs,oack,sym)
```

## Description

`ack = ltePUCCH2DRSDecode(ue,chs,oack,sym)` returns a vector of hybrid automatic repeat request (HARQ) indicator values, `ack`, obtained by performing PUCCH format 2 DRS decoding of the complex matrix, `sym`. The decoder uses a maximum likelihood (ML) approach, assuming that `sym` has already been equalized, to best restore the originally transmitted complex values. The symbols for each antenna are in the columns of `sym`. The number of columns in `sym` should match the number of PUCCH resource indices specified in the `chs` structure.

`oack` specifies the number of HARQ indicator values expected.

`ack` is a column vector containing `oack` HARQ indicator (HI) values.

## Examples

### Decode PUCCH Format 2A DM-RS

Decode a PUCCH format 2A DM-RS from a synchronized and equalized resource array.

Initialize input configuration structures demonstrating use of `'Name',Value` pair assignment and direct field assignment.

```
ue = struct('NULRB',6,'NCellID',0,'NSubframe',0,'Hopping','Off');
pucch2 = struct('ResourceIdx',0);

ue.CyclicPrefixUL = 'Normal';
ue.NTxAnts = 1;
pucch2.ResourceSize = 0;
pucch2.CyclicShifts = 0;
```

For the transmitter, create the PUCCH format 2A DM-RS.

```
reGrid = lteULResourceGrid(ue);
drsIndices = ltePUCCH2DRSIndices(ue,pucch2);
txAck = [1;0];
reGrid(drsIndices) = ltePUCCH2DRS(ue,pucch2,txAck);
```

On the receiver side, decode the PUCCH format 2 DM-RS symbols and view `rxAck` to confirm it matches `txAck`

```
drsSymbols = reGrid(drsIndices);
rxAck = ltePUCCH2DRSDecode(ue,pucch2,length(txAck),drsSymbols)
```

```
rxAck = 2x1 logical array
```

```
1
0
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific configuration settings, specified as a structure that can contain the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NCellID** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **NSubframe** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **CyclicPrefixUL** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **Hopping** | Optional | `'Off'` (default), `'Group'` | Frequency hopping method. |
| **NPUCCHID** | Optional | Integer from 0 to 503 | PUCCH virtual cell identity. If this field is not present, `NCellID` is used as the identity. |

### chs — Channel transmission configuration

structure

PUCCH channel settings, specified as a structure that can contain the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **ResourceIdx** | Optional | 0 (default), integer from 0 to 1185 or vector of integers. | PUCCH resource indices which determine the physical resource blocks, cyclic shift, and orthogonal cover used for transmission. ($n^{(2)}_{PUCCH}$). Define one index for each transmission antenna. |
| **ResourceSize** | Optional | 0 (default), integer from 0 to 98. | Size of resource allocated to PUCCH format 2 ($N^{(2)}_{RB}$) |
| **CyclicShifts** | Optional | 0 (default), integer from 0 to 7 | Number of cyclic shifts used for format 1 in resource blocks (RBs) with a mixture of format 1 and format 2 PUCCH, specified as an integer from 0 to 7. ($N^{(1)}_{cs}$) |

### oack — Number of uncoded HARQ-ACK bits

1 | 2

`oack` specifies the number of HARQ indicator values expected, specified as nonnegative integer vector. The number of HARQ indicator values is 1 for PUCCH format 2A and 2 for PUCCH format 2B.

Data Types: `double`

**`sym` — Symbols of each antenna**
complex numeric matrix

Symbols for each antenna, specified as complex numeric matrix. The number of columns in `sym` should match the number of PUCCH resource indices specified in the `chs` structure.

Example: 0.25881 + 0.9659i

Data Types: `double`
Complex Number Support: Yes

## Output Arguments

**`ack` — Hybrid ARQ indicator values**
logical column vector

Hybrid ARQ indicator values, returned as a logical column vector. This output is obtained by performing PUCCH format 1 decoding of the complex matrix, `sym`.

Data Types: `logical`

# Version History

**Introduced in R2014a**

## See Also

`ltePUCCH2DRS` | `ltePUCCH2DRSIndices` | `ltePUCCH2` | `ltePUCCH2Decode` | `ltePUCCH2Indices` | `ltePUCCH2PRBS` | `ltePUCCH1DRS` | `ltePUCCH3DRS`

# ltePUCCH2DRSIndices

PUCCH format 2 DRS resource element indices

## Syntax

```
ind = ltePUCCH2DRSIndices(ue,chs)
[ind,info] = ltePUCCH2DRSIndices(ue,chs)
[ ___ ] = ltePUCCH2DRSIndices(ue,chs,opts)
```

## Description

`ind = ltePUCCH2DRSIndices(ue,chs)` returns a matrix of resource element indices for the demodulation reference signal (DRS) associated with the PUCCH format 2 transmission given structures containing the UE-specific settings, and the channel transmission configuration.

`[ind,info] = ltePUCCH2DRSIndices(ue,chs)` also returns a PUCCH information structure array, `info`.

`[ ___ ] = ltePUCCH2DRSIndices(ue,chs,opts)` formats the returned indices using the options specified by `opts`.

This syntax supports output options from prior syntaxes.

## Examples

### Generate PUCCH Format 2 DM-RS Indices

Generate PUCCH format 2 DM-RS RE indices for a 1.4 MHz bandwidth and PUCCH resource index 0. Use default values for all other parameters.

Initialize UE-specific and channel configuration structures. Generate PUCCH format 2 DM-RS indices.

```
ue.NULRB = 6;
ue.CyclicPrefixUL = 'Normal';

chs.ResourceIdx = 0;

ind = ltePUCCH2DRSIndices(ue,chs);
ind(1:4)
```

```
ans = 4x1 uint32 column vector

   73
   74
   75
   76
```

### Generate PUCCH Format 2 DM-RS Indices for Four Antennas

Generate the PUCCH format 2 DM-RS indices for four transmit antenna paths, and display the output information structure.

Initialize UE-specific and channel configuration structures. Generate PUCCH 2 DM-RS indices and information outputs.

```
ue.NULRB = 6;
ue.CyclicPrefixUL = 'Normal';

chs.ResourceIdx = [0 37 4 111];

[ind,info] = ltePUCCH2DRSIndices(ue,chs);
```

Because there are four antennas, the DM-RS indices are output as a four-column vector and the `info` output structure contains four elements. View `ind` and the size of `info` to confirm.

```
ind(1:6,:)
```

```
ans = 6x4 uint32 matrix

     73   1129   2089   3109
     74   1130   2090   3110
     75   1131   2091   3111
     76   1132   2092   3112
     77   1133   2093   3113
     78   1134   2094   3114
```

```
size(info)
```

```
ans = 1×2

     1     4
```

View the contents of one of the `info` structure elements.

```
info(4)
```

```
ans = struct with fields:
    PRBSet: [1 4]
     RBIdx: 9
```

### Generate PUCCH Format 2 DM-RS Indices Varying Indexing Style

Generate the PUCCH format 2 DM-RS indices for two transmit antenna paths, and output in subscript indexing form.

Initialize UE-specific and channel configuration structures and the indexing option parameter. Generate PUCCH 2 DM-RS indices and information outputs.

```
ue.NULRB = 6;
ue.CyclicPrefixUL = 'Normal';
```

```
chs.ResourceIdx = [0 4];

[ind,info] = ltePUCCH2DRSIndices(ue,chs,{'sub'});
```

Using 'sub' indexing style, the indices are output in [subcarrier, symbol, antenna] subscript form. View the midpoint of ind and observe the antenna index change.

```
size(ind)

ans = 1×2

    96     3


ind(46:51,:)

ans = 6x3 uint32 matrix

   70   13    1
   71   13    1
   72   13    1
    1    2    2
    2    2    2
    3    2    2


size(info)

ans = 1×2

     1     2
```

Because there are two antennas, the info output structure contains two elements. View one of the `info` structure elements.

```
info(2)

ans = struct with fields:
    PRBSet: [0 5]
     RBIdx: 0
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific settings, specified as a structure containing these fields.

### NULRB — Number of uplink resource blocks

integer from 6 to 110.

Number of uplink resource blocks, specified as an integer from 6 to 110.

Data Types: `double`

**`CyclicPrefixUL` — Cyclic prefix length for uplink channels**
`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length for uplink channels, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

**`chs` — Channel transmission configuration**
structure

Channel transmission configuration, specified as a structure containing the following fields.

**`ResourceIdx` — PUCCH resource indices**
0 (default) | 0,...,1185 | integer | vector of integers | optional

PUCCH resource indices, specified as an integer or a vector of integers. Values range from 0 to 1185. There is one index for each transmission antenna. These indices determine the cyclic shift and orthogonal cover used for transmission. ($n^{(2)}_{PUCCH}$)

Data Types: `struct`

**`opts` — Output format options for resource element indices**
character vector | cell array of character vectors | string array

Output format options for resource element indices, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string) :

| Category | Options | Description |
|---|---|---|
| Indexing style | `'ind'` (default) | The returned indices are in linear index style. |
| | `'sub'` | The returned indices are in `[subcarrier,symbol,port]` subscript row style. |
| Index base | `'1based'` (default) | The returned indices are one-based. |
| | `'0based'` | The returned indices are zero-based. |

Example: `'ind 1based'`, `"ind 1based"`, `{'ind','1based'}`, or `["ind","1based"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

**`ind` — Resource element indices**
integer column vector | three-column integer matrix

Resource element indices, returned as an integer column vector or a three-column integer matrix. By default the indices are returned in one-based linear indexing form that can directly index elements of a resource matrix. These indices are ordered according to PUCCH format 2 DRS modulation symbol mapping. The `opts` input offers alternative indexing formats. The indices for each antenna are in the

columns of `ind`, with the number of columns determined by the number of PUCCH resource indices specified in `chs.ResourceIdx`.

Example: [145,146,147,...]

Data Types: `uint32`

**`info` — PUCCH format 2 DRS information**
structure array

PUCCH format 2 DRS information, returned as a structure array with elements corresponding to each transmit antenna and containing these fields.

**`PRBSet` — Indices occupied by PRB in each slot of subframe**
nonnegative integer vector

Indices occupied by PRB in each slot of the subframe, returned as a nonnegative integer vector. The indices are zero-based.

Example: [0,5]

Data Types: `double`

**`RBIdx` — PUCCH logical resource block index**
nonnegative integer

PUCCH logical resource block index, returned as a nonnegative integer. ($m$)

Data Types: `double`

Data Types: `struct`

# Version History

**Introduced in R2014a**

## See Also

`ltePUCCH2` | `ltePUCCH2Decode` | `ltePUCCH2Indices` | `ltePUCCH2DRS` | `ltePUCCH2DRSDecode` | `ltePUCCH2PRBS` | `ltePUCCH1DRSIndices` | `ltePUCCH3DRSIndices`

# ltePUCCH2Decode

Physical uplink control channel format 2 decoding

## Syntax

```
out = ltePUCCH2Decode(ue,chs,sym)
```

## Description

`out = ltePUCCH2Decode(ue,chs,sym)` performs decoding of the PUCCH format 2 given UE-specific settings `ue` and channel transmission configuration `chs`. `out` is a soft bit vector consisting of 20 bits, formed by decoding complex symbol matrix `sym`, performing demodulation with the PUCCH format 2 reference sequence, QPSK demodulation, and descrambling. The symbols for each antenna are in the columns of `sym`, and the number of columns should match the number of PUCCH Resource Indices specified in the structure, `chs`.

## Examples

### Decode PUCCH Format 2 Signal

Decode a PUCCH format 2 signal from an equalized resource array, `grid`.

First, create a UE configuration structure, `ue`, and a PUCCH configuration structure, `pucch2`.

```
ue = struct('NULRB',6,'NCellID',0,'NSubframe',0,'RNTI',1);
pucch2 = struct('ResourceIdx',0);
```

For the transmitter, create a PUCCH format 2 resource grid.

```
rgrid = lteULResourceGrid(ue);
pucch2Indices = ltePUCCH2Indices(ue,pucch2);
tx = [1;0;0;0;0;1];
encoded = lteUCIEncode(tx);
rgrid(pucch2Indices) = ltePUCCH2(ue,pucch2,encoded);
```

On the receiver side, decode the PUCCH format 2 signal contained in equalized resource array, `grid`. Also decode the UCI bits.

```
rx = ltePUCCH2Decode(ue,pucch2,rgrid(pucch2Indices));
decoded = lteUCIDecode(rx,length(tx))
```

```
decoded = 6x1 logical array

   1
   0
   0
   0
   0
   1
```

## Input Arguments

**`ue` — UE-specific settings**
structure

`ue` is a structure having the following fields.

**`NCellID` — Cell identity number**
0 (default)

Physical layer cell identity number, specified as a nonnegative scalar integer.

Example: 4

Data Types: `double`

**`NSubframe` — Subframe number**
0 (default)

Position reference signal subframe number, specified as a nonnegative scalar integer.

Example: 8

Data Types: `double`

**`RNTI` — Radio network temporary identifier (16-bit)**
scalar integer

Radio network temporary identifier (16-bit), specified as a scalar integer.

Data Types: `double`

**`CyclicPrefixUL` — Cyclic prefix length for uplink channels**
`'Normal'` (default) | optional | `'Extended'`

Cyclic prefix length for uplink channels, specified as `'Normal'` or `'Extended'`. Optional.

Data Types: `char` | `string`

**`Hopping` — Uplink frequency hopping**
`'Off'` (default) | optional | `'Group'`

Uplink frequency hopping, specified as `'Off'` or `'Group'`. Optional.

Data Types: `char` | `string`

Data Types: `struct`

**`chs` — Channel transmission configuration**
structure

Channel transmission configuration, specified as a structure. `chs` contains the following fields.

**`ResourceIdx` — PUCCH resource indices**
0 (default) | optional | 0...1185

PUCCH resource indices, specified as a nonnegative vector with one element for each transmission antenna. These indices determine the cyclic shift and orthogonal cover used for transmission. (*n2_pucch*)

Example: 78

Data Types: `double`

**`ResourceSize` — Size of resources allocated to PUCCH format 2**
0 (default) | optional | 0...98

Size of resources allocated to PUCCH format 2, specified as nonnegative scalar integer. This parameter affects location of this transmission. (*N2RB*)

Data Types: `double`

**`CyclicShifts` — Number of cyclic shifts for format 1 resource blocks**
0 (default) | optional | 0...7

Number of cyclic shifts for format 1 resource blocks, in RBs, specified as a nonnegative scalar integer. This parameter can be used in a mixture of format 1 and format 2 PUCCH. (*N1cs*)

Example: 7

Data Types: `double`

**`sym` — Symbols of each antenna**
complex numeric matrix

Symbols for each antenna, specified as complex numeric matrix. The number of columns should match the number of PUCCH resource indices specified in the channel transmission configuration structure, `chs`.

Example: 0.25881 + 0.9659i

Data Types: `double`
Complex Number Support: Yes

## Output Arguments

**`out` — PUCCH format 2 decoded soft bit output**
logical column vector

PUCCH format 2 decoded soft bit output, returned as a logical column vector. This output contains the result of decoding `sym`.

Data Types: `logical`

# Version History

**Introduced in R2014a**

## See Also

`ltePUCCH2` | `ltePUCCH2Indices` | `ltePUCCH2PRBS` | `ltePUCCH2DRS` | `ltePUCCH2DRSIndices` | `ltePUCCH2DRSDecode` | `lteUCIDecode` | `ltePUCCH1Decode` | `ltePUCCH3Decode`

# ltePUCCH2Indices

PUCCH format 2 resource element indices

## Syntax

```
ind = ltePUCCH2Indices(ue,chs)
[ind,info] = ltePUCCH2Indices(ue,chs)
[___] = ltePUCCH2Indices(ue,chs,opts)
```

## Description

`ind = ltePUCCH2Indices(ue,chs)` returns a matrix of resource element indices for the Physical Uplink Control Channel (PUCCH) Format 2 transmission given structures containing the UE-specific settings, and the channel transmission configuration.

`[ind,info] = ltePUCCH2Indices(ue,chs)` also returns a PUCCH information structure array `info`.

`[___] = ltePUCCH2Indices(ue,chs,opts)` formats the returned indices using options specified by `opts`.

This syntax supports output options from prior syntaxes.

## Examples

### Generate PUCCH Format 2 Indices

Generate PUCCH format 2 RE indices for a 1.4 MHz bandwidth and PUCCH resource index 0. Use default values for all other parameters.

Initialize UE-specific and channel configuration structures. Generate PUCCH format 2 indices.

```
ue.NULRB = 6;
ue.CyclicPrefixUL = 'Normal';

chs.ResourceIdx = 0;

ind = ltePUCCH2Indices(ue,chs);
ind(1:4)
```

```
ans = 4x1 uint32 column vector

   1
   2
   3
   4
```

### Generate PUCCH Format 2 Indices for Three Antennas

Generate the physical uplink control channel (PUCCH) format 2 indices for three transmit antenna paths, and display the information structure output.

Initialize UE-specific and channel configuration structures. Generate PUCCH 2 indices and information outputs.

```
ue.NULRB = 6;
ue.CyclicPrefixUL = 'Normal';

chs.ResourceIdx = [0 129 2];

[ind,info] = ltePUCCH2Indices(ue,chs);
```

Because there are three antennas, the indices are output as a three column vector and the `info` output structure contains three elements. View `ind` and the size of `info` to confirm this.

```
ind(1:5,:)
```

```
ans = 5x3 uint32 matrix

      1   1069   2017
      2   1070   2018
      3   1071   2019
      4   1072   2020
      5   1073   2021
```

```
size(info)
```

```
ans = 1×2

     1     3
```

View the contents of one of the `info` structure elements.

```
info(1)
```

```
ans = struct with fields:
    PRBSet: [0 5]
     RBIdx: 0
```

### Generate PUCCH Format 2 Indices Varying Indexing Style

Generate the PUCCH format 2 indices for two transmit antenna paths, and output in subscript indexing form.

Initialize UE-specific and channel configuration structures, and the indexing option parameter. Generate PUCCH 2 indices and information outputs.

```
ue.NULRB = 6;
ue.CyclicPrefixUL = 'Normal';
```

```
chs.ResourceIdx = [0 4];

opts = {'sub'};

[ind,info] = ltePUCCH2Indices(ue,chs,opts);
```

Using `'sub'` indexing style, the indices are output in `[subcarrier, symbol, antenna]` subscript form. View the midpoint of `ind` and observe the antenna index change.

```
size(ind)
```

```
ans = 1×2

   240     3
```

```
ind(118:123,:)
```

```
ans = 6x3 uint32 matrix

   70   14    1
   71   14    1
   72   14    1
    1    1    2
    2    1    2
    3    1    2
```

Because there are two antennas, the `info` output structure contains two elements. View the contents of one of the `info` structure elements.

```
size(info)
```

```
ans = 1×2

     1     2
```

```
info(2)
```

```
ans = struct with fields:
    PRBSet: [0 5]
     RBIdx: 0
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific settings, specified as a structure containing these fields.

### NULRB — Number of uplink resource blocks

integer from 6 to 110

Number of uplink resource blocks, specified as an integer from 6 to 110.

Data Types: `double`

**`CyclicPrefixUL` — Cyclic prefix length for uplink channels**
`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length for uplink channels, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

**`chs` — Channel transmission configuration**
structure

Channel transmission configuration, specified as a structure containing the following fields.

**`ResourceIdx` — PUCCH resource indices**
0 (default) | 0,...,1185 | integer | vector of integers | optional

PUCCH resource indices, specified as an integer or a vector of integers. Values range from 0 to 1185. There is one index for each transmission antenna. These indices determine the cyclic shift and orthogonal cover used for transmission. ($n^{(2)}_{PUCCH}$)

Data Types: `double`

**`opts` — Output format options for resource element indices**
character vector | cell array of character vectors | string array

Output format options for resource element indices, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string) :

| Category | Options | Description |
|---|---|---|
| Indexing style | `'ind'` (default) | The returned indices are in linear index style. |
| | `'sub'` | The returned indices are in `[subcarrier,symbol,port]` subscript row style. |
| Index base | `'1based'` (default) | The returned indices are one-based. |
| | `'0based'` | The returned indices are zero-based. |

Example: `'ind 1based'`, `"ind 1based"`, `{'ind','1based'}`, or `["ind","1based"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

**`ind` — PUCCH format 2resource element indices**
integer column vector | three-column integer matrix

PUCCH format 2 resource element indices, returned as an integer column vector or a three-column integer matrix. By default, the indices are returned in one-based linear indexing form that can directly index elements of a resource matrix. These indices are ordered according to PUCCH format 2 modulation symbol mapping as specified in TS 36.211 [1], Section 5.4. The `opts` input offers alternative indexing formats. The indices for each antenna are in the columns of `ind`, with the

number of columns determined by the number of PUCCH resource indices specified in `chs.ResourceIdx`.

Example: [1,2,3,4...]

Data Types: `uint32`

**`info` — PUCCH format 2 information**
structure array

PUCCH format 2 information, returned as a structure array with elements corresponding to each transmit antenna and containing these fields.

**`PRBSet` — Set of PRB indices**
column vector | two-column matrix

Set of PRB indices, returned as an integer column vector or two-column integer matrix corresponding to the resource allocations.

- When returned as a column vector, the resource allocation is the same in both slots of the subframe.
- When returned as a two-column matrix, the resource allocations can vary for each slot in the subframe.

The PRB indices are zero-based.

Example: [0,5]

Data Types: `double`

**`RBIdx` — PUCCH logical resource block index**
nonnegative integer

PUCCH logical resource block index, returned as a nonnegative integer. (*m*)

Data Types: `uint32`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. "Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePUCCH2` | `ltePUCCH2Decode` | `ltePUCCH2DRS` | `ltePUCCH2DRSIndices` | `ltePUCCH2DRSDecode` | `ltePUCCH2PRBS` | `ltePUCCH1Indices` | `ltePUCCH3Indices`

# ltePUCCH2PRBS

PUCCH format 2 pseudorandom scrambling sequence

## Syntax

```
[seq,cinit] = ltePUCCH2PRBS(ue,n)
[seq,cinit] = ltePUCCH2PRBS(ue,n,mapping)

[subseq,cinit] = ltePUCCH2PRBS(ue,pn)
[subseq,cinit] = ltePUCCH2PRBS(ue,pn,mapping)
```

## Description

`[seq,cinit] = ltePUCCH2PRBS(ue,n)` returns the first `n` outputs of the Physical Uplink Control Channel (PUCCH) Format 2 scrambling sequence when initialized according to UE-specific settings, `ue`. It also returns an initialization value `cinit` for the pseudorandom binary sequence (PRBS) generator.

`[seq,cinit] = ltePUCCH2PRBS(ue,n,mapping)` allows control over the format of the returned sequence, `seq`, with the input `mapping`.

`[subseq,cinit] = ltePUCCH2PRBS(ue,pn)` returns a subsequence of a full PRBS sequence, specified by `pn`.

`[subseq,cinit] = ltePUCCH2PRBS(ue,pn,mapping)` allows control over the format of the returned subsequence, `subseq`, with the input `mapping`.

## Examples

### Scramble UCI bits

Scramble the encoded UCI bits representing RI=3 using 2 bits. According to Table 5.2.2.6-6 in TS 36.212 this maps to the set of input bits [1; 0].

Create user-specific configuration structure. Generate a UCI codeword and a pseudorandom scrambling sequence for the PUCCH2 the same length as the codeword.

```
ue.NCellID = 1;
ue.NSubframe = 0;
ue.RNTI = 1;
cw = lteUCIEncode([1;0]);
seq = ltePUCCH2PRBS(ue,length(cw));
size(seq)
```

```
ans = 1×2

    20     1
```

Scramble the UCI codeword using the generated scrambling sequence.

```
scrambled = xor(seq,cw);
```

### Generate PUCCH Format 2 Pseudorandom Scrambling Sequence

Generate unsigned and signed PUCCH format 2 pseudorandom scrambling sequences.

Create user-specific configuration structure.

```
ue.NCellID = 1;
ue.NSubframe = 0;
ue.RNTI = 1;
```

Generate a PUCCH format 2 pseudorandom scrambling sequence.

```
seq = ltePUCCH2PRBS(ue,5)
```

```
seq = 5x1 logical array

   1
   1
   0
   0
   1
```

Generate a signed PUCCH format 2 pseudorandom scrambling sequence.

```
seq = ltePUCCH2PRBS(ue,5,'signed')
```

```
seq = 5×1

    -1
    -1
     1
     1
    -1
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific settings, specified as a structure. `ue` contains the following fields.

### `NCellID` — Physical layer cell identity number

nonnegative scalar integer

Physical layer cell identity number, specified as a nonnegative scalar integer.

Example: 1

Data Types: `double`

**`NSubframe` — Subframe number**
nonnegative scalar integer

Subframe number, specified as a nonnegative scalar integer.

Example: 0

Data Types: `double`

**`RNTI` — Radio network temporary identifier**
scalar integer

Radio network temporary identifier (16-bit), specified as a scalar integer.

Example: 1

Data Types: `double`

**`n` — Number of elements in returned sequence**
numeric scalar

Number of elements in returned sequence, `seq`, specified as a numeric scalar.

Data Types: `double`

**`pn` — Range of elements in returned subsequence**
row vector

Range of elements in returned subsequence, `subseq`, specified as a row vector of `[p n]`. The subsequence returns `n` values of the PRBS generator, starting at position `p` (0-based).

Data Types: `double`

**`mapping` — Output sequence formatting**
`'binary'` (default) | `'signed'`

Output sequence formatting, specified as `'binary'` or `'signed'`. `mapping` controls the format of the returned sequence.

- `'binary'` maps `true` to 1 and `false` to 0.
- `'signed'` maps `true` to –1 and `false` to 1.

Data Types: `char` | `string`

## Output Arguments

**`seq` — PUCCH format 2 pseudorandom scrambling sequence**
logical column vector | numeric column vector

PUCCH format 2 pseudorandom scrambling sequence, returned as a logical column vector or a numeric column vector. `seq` contains the first `n` outputs of the scrambling sequence. If you set `mapping` to `'signed'`, the output data type is double. Otherwise, the output data type is logical.

Data Types: `logical` | `double`

**`subseq` — PUCCH format 2 pseudorandom scrambling subsequence**
logical column vector | numeric column vector

PUCCH format 2 pseudorandom scrambling subsequence, returned as a logical column vector or a numeric column vector. `subseq` contains the values of the PRBS generator specified by `pn`. If you set `mapping` to `'signed'`, the output data type is `double`. Otherwise, the output data type is `logical`.

Data Types: `logical` | `double`

**`cinit` — Initialization value for PRBS generator**
numeric scalar

Initialization value for PRBS generator, returned as a numeric scalar.

Data Types: `uint32`

## Version History

**Introduced in R2014a**

## See Also

`ltePUCCH2` | `ltePUCCH2Decode` | `ltePUCCH2Indices` | `ltePUCCH2DRS` | `ltePUCCH2DRSDecode` | `ltePUCCH2DRSIndices` | `ltePUCCH3Indices` | `ltePUCCH3PRBS`

# ltePUCCH3

Physical uplink control channel format 3

## Syntax

```
sym = ltePUCCH3(ue,chs,bits)
[sym,info] = ltePUCCH3(ue,chs,bits)
```

## Description

`sym = ltePUCCH3(ue,chs,bits)` returns a matrix containing Physical Uplink Control Channel (PUCCH) format 3 symbols given a structure of UE-specific settings, a structure with channel transmission configuration settings, and a vector of coded hybrid ARQ (HARQ) values, `bits`.

`[sym,info] = ltePUCCH3(ue,chs,bits)` also returns a PUCCH information structure array, `info`.

## Examples

### Generate PUCCH Format 3 Symbols

Generate PUCCH format 3 modulated symbols.

Initialize `ue` and `chs` configuration structures. Generate and view PUCCH Format 3 symbols.

```
ue.NCellID = 0;
ue.NSubframe = 0;
ue.RNTI = 1;
ue.CyclicPrefixUL = 'Normal';
ue.Shortened = 0;

chs.ResourceIdx = 0;

sym = ltePUCCH3(ue,chs,ones(48,1));
sym(1:5)
```

```
ans = 5×1 complex

   1.6330 - 1.2247i
  -0.7071 + 0.7071i
  -0.5577 + 0.1494i
   0.4082 - 0.0000i
  -0.5577 - 0.9659i
```

### Generate PUCCH Format 3 Symbols for Two Antennas

Generate the physical uplink control channel (PUCCH) format 3 symbols for two transmit antenna paths and display the information structure.

Initialize parameters for a UE-specific configuration structure and a channel configuration structure. Generate PUCCH1 symbols and information outputs.

```
ue.NCellID = 1;
ue.RNTI = 1;
ue.NSubframe = 0;
ue.CyclicPrefixUL = 'Normal';
ue.Shortened = 0;

chs.ResourceIdx = [0 3];

[pucch3Sym,info] = ltePUCCH3(ue,chs,[]);
```

Because there are two antennas, the symbols are output as a two-column vector, and the `info` output structure contains two elements.

```
pucch3Sym(1:6,:)
```

```
ans = 6×2 complex

   0.0000 + 2.4495i   0.0000 + 2.4495i
   0.0000 - 0.0000i   0.0000 - 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 - 0.0000i   0.0000 - 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i
```

```
size(info)
```

```
ans = 1×2

     1     2
```

View the contents of the second `info` structure element.

```
info(2)
```

```
ans = struct with fields:
    NCellCyclicShift: [64 192 46 212 191 71 91 84 25 105]
          OrthSeqIdx: [3 4]
             Symbols: [1.0000 + 0.0000i 1.0000 + 0.0000i 1.0000 + 0.0000i 1.0000 + 0.0000i 1.000
             OrthSeq: [5x2 double]
           NSymbSlot: [5 5]
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific configuration settings, specified as a structure that can contain these fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NCellID`** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **`RNTI`** | Required | 0 (default), scalar integer | Radio network temporary identifier (RNTI) value (16 bits) |
| **`NSubframe`** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **`CyclicPrefixUL`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **`Shortened`** | Optional | 0 (default), 1 | Option to shorten the subframe by omitting the last symbol, specified as 0 or 1. If `1`, the last symbol of the subframe is not used. For subframes with possible SRS transmission, set `Shortened` to 1 to maintain a standard compliant configuration. |

Data Types: `struct`

**`chs` — Channel transmission configuration**
structure

Channel transmission configuration, specified as a structure containing these fields.

**`ResourceIdx` — PUCCH Resource Indices**
0 (default) | optional | 0,...,549 | vector | vector of integers

PUCCH Resource Indices, specified as an integer or vector of integers with values from 0 to 549. There is one index for each transmission antenna. These indices determine the cyclic shift and orthogonal cover used for transmission. ($n^{(3)}_{PUCCH}$).

Data Types: `struct`

**`bits` — Coded HARQ-ACK bits**
nonnegative integer column vector of length 48

Coded HARQ-ACK bits, specified as a nonnegative integer column vector of length 48. TS 36.211 [1], Table 5.4-1 specifies the vector length for PUCCH format 3 is $M_{bit}$ = 48. `bits` is expected to be the block of bits *b(0)...b($M_{bit}$–1)*specified in TS 36.211 [1], Section 5.4.2A. `bits` is also expected to be generated by performing uplink control information (UCI) channel coding as described TS 36.212 [2], Section 5.2.3.1. For PUCCH format 3, UCI includes encoding of concatenated HARQ-ACK bits and any appended periodic CSI bits plus Scheduling Request (SR) bit when present.

Data Types: `double`

## Output Arguments

**`sym` — PUCCH format 3 symbols**
matrix

PUCCH format 3 symbols, returned as matrix. The symbols for each antenna are in the columns of `sym`, with the number of columns determined by the number of PUCCH resource indices specified in `chs.ResourceIdx`.

Example: [ 0.7071 + 0.7071i,...]

Data Types: `double`

**`info` — PUCCH format 3 information**
structure

PUCCH format 3 information, returned as a structure array with elements corresponding to each transmit antenna and containing these fields.

**`NCellCyclicShift` — Cell-specific cyclic shift for each OFDM symbol**
vector

Cell-specific cyclic shift for each OFDM symbol, returned as a vector. ($n_{cs}^{cell}$)

**`OrthSeqIdx` — Orthogonal sequence index for each slot**
vector

Orthogonal sequence index for each slot, returned as a two-element vector. ($n_{\mathrm{oc}}$)

**`Symbols` — Modulated data symbols for each OFDM symbol**
vector

Modulated data symbols for each OFDM symbol, returned as a vector. ($d$)

Example: [0.7071 + 0.7071i,...]

**`OrthSeq` — Orthogonal sequence for each slot**
numeric matrix

Orthogonal sequence of each slot, returned as a numeric matrix. Each column in the matrix contains the orthogonal sequence ($w_{\mathrm{n_{oc}}}$) for each slot.

---

**Note** When `ue.Shortened` = 1, transmissions are shortened, and the second column of `info.OrthSeq` has a zero in the last row because the spreading factor for the second slot is 4 instead of 5.

---

Example: [1.000 + 1.000i,...]

**`NSymbSlot` — Number of OFDM symbols in each slot**
vector of integers

The number of OFDM symbols in each slot, returned as a vector of integers. ($\left[N_{\mathrm{SF},0}^{\mathrm{PUCCH}}\ N_{\mathrm{SF},1}^{\mathrm{PUCCH}}\right]$)

Data Types: `struct`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePUCCH3Decode` | `ltePUCCH3Indices` | `ltePUCCH3DRS` | `ltePUCCH3DRSIndices` | `ltePUCCH3PRBS` | `lteUCI3Encode`

# ltePUCCH3Decode

Physical uplink control channel format 3 decoding

## Syntax

```
out = ltePUCCH3Decode(ue,chs,sym)
```

## Description

`out = ltePUCCH3Decode(ue,chs,sym)` decodes the PUCCH format 3 given UE-specific settings, `ue`, and channel transmission configuration, `chs`. `out` is a soft bit vector of coded UCI consisting of 48 bits, formed by decoding the complex symbol matrix, `sym`. The symbol decoding steps are SC-FDMA deprecoding, demodulation with the PUCCH Format 3 reference sequence, QPSK demodulation, and descrambling. The symbols for each antenna are in the columns of `sym`, and the number of columns should match the number of PUCCH Resource Indices specified in the structure `chs`.

## Examples

### Decode PUCCH Format 3 Signal

Decode a PUCCH format 3 signal contained in an equalized resource array for the specified UE and PUCCH configuration structures.

```
ue.NULRB = 6;
ue.NCellID = 0;
ue.RNTI = 1;
ue.CyclicPrefixUL = 'Normal';
ue.NTxAnts = 1;
ue.Shortened = 0;
ue.NSubframe = 0;

pucch3.ResourceIdx = 0;
```

To model the transmitter, create an uplink resource grid and populate it with PUCCH format 3 symbols.

```
reGrid = lteULResourceGrid(ue);
pucch3Indices = ltePUCCH3Indices(ue,pucch3);
tx = [1; 0; 0; 1; 1; 1];
encoded = lteUCI3Encode(tx);
reGrid(pucch3Indices) = ltePUCCH3(ue,pucch3,encoded);
```

To model the receiver, decode the PUCCH format 3 symbols contained in an equalized resource array. Decode and display the UCI. Verify received `decoded` bits match `tx` bits.

```
eqGrid = reGrid;
rx = ltePUCCH3Decode(ue,pucch3,eqGrid(pucch3Indices));
decoded = lteUCI3Decode(rx,length(tx))
```

```
decoded = 6x1 logical array

   1
   0
   0
   1
   1
   1
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific configuration settings, specified as a structure that can contain the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NCellID`** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **`RNTI`** | Required | 0 (default), scalar integer | Radio network temporary identifier (RNTI) value (16 bits) |
| **`NSubframe`** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **`CyclicPrefixUL`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **`Shortened`** | Optional | 0 (default), 1 | Option to shorten the subframe by omitting the last symbol, specified as 0 or 1. If `1`, the last symbol of the subframe is not used. For subframes with possible SRS transmission, set `Shortened` to 1 to maintain a standard compliant configuration. |

Data Types: `struct`

### chs — Channel transmission configuration

structure

PUCCH channel settings, specified as a structure that can contain the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`ResourceIdx`** | Optional | 0 (default), integer from 0 to 549, or vector of integers. | PUCCH resource indices which determine the physical resource blocks, cyclic shift, and orthogonal cover used for transmission ($n^{(3)}_{PUCCH}$). Define one index for each transmission antenna. |

Data Types: `struct`

**`sym` — PUCCH format 3 symbols**
complex-valued matrix

PUCCH format 3 symbols, specified as a complex-valued matrix. The symbols for each antenna are in the columns of `sym`, and the number of columns should match the number of PUCCH resource indices specified in the structure `chs`.

Data Types: `double`
Complex Number Support: Yes

## Output Arguments

**`out` — Soft bit vector**
48-by-1 real-valued column vector

Soft bit vector consisting of 48 bits, returned as a 48-by-1 real-valued column vector. The number of PUCCH Resource Indices specified in the structure `chs` determines the number of columns in `out`.

Data Types: `double`

# Version History

**Introduced in R2014a**

## See Also

`ltePUCCH3` | `ltePUCCH3Indices` | `ltePUCCH3DRS` | `ltePUCCH3DRSIndices` | `ltePUCCH3PRBS` | `lteUCI3Decode`

# ltePUCCH3DRS

PUCCH format 3 demodulation reference signal

## Syntax

```
seq = ltePUCCH3DRS(ue,chs)
[seq,info] = ltePUCCH3DRS(ue,chs)
```

## Description

`seq = ltePUCCH3DRS(ue,chs)` returns a matrix containing demodulation reference signal (DRS) associated with PUCCH format 3 transmission given structures containing the UE-specific settings, and the channel transmission configuration settings.

`[seq,info] = ltePUCCH3DRS(ue,chs)` also returns a PUCCH information structure array, `info`.

## Examples

### Generate PUCCH Format 3 DM-RS

Generate the PUCCH Format 3 Demodulation Reference Signal (DM-RS) values for UE-specific settings.

Initialize UE specific (`ue`) and channel (`chs`) configuration structures. Generate PUCCH DM-RS values.

```
ue.NCellID = 1;
ue.NSubframe = 0;
ue.CyclicPrefixUL = 'Normal';
ue.Hopping = 'Off';
ue.Shortened = 0;

chs.ResourceIdx = 0;
chs.CyclicShifts = 0;

pucch3RefSig = ltePUCCH3DRS(ue,chs);
pucch3RefSig(1:4)
```

```
ans = 4×1 complex

   0.7071 + 0.7071i
   0.2588 + 0.9659i
  -0.9659 - 0.2588i
  -0.7071 - 0.7071i
```

### Generate PUCCH Format 3 DM-RS Using Virtual Cell ID

Demonstrate Uplink Release 11 coordinated multipoint (CoMP) operation. Intercell interference can be avoided by using a virtual cell identity for a potentially interfering UE in a neighboring cell.

Configuration for UE of interest, UE 1 in cell 1.

```
ue1.NCellID = 1;
ue1.NSubframe = 0;
ue1.CyclicPrefixUL = 'Normal';
ue1.Hopping = 'Off';
ue1.Shortened = 0;

chs1.ResourceIdx = 0;
```

Configuration for interferer, UE 2 in cell 2.

```
ue2.NCellID = 2;
ue2.NSubframe = 0;
ue2.CyclicPrefixUL = 'Normal';
ue2.Hopping = 'Off';
ue2.Shortened = 0;

chs2.ResourceIdx = 1;
```

Measure the interference between the DM-RS signals.

```
interferenceNoCoMP = abs(sum(ltePUCCH3DRS(ue1,chs1).*conj(ltePUCCH3DRS(ue2,chs2))))
```

```
interferenceNoCoMP = 6.3246
```

Reconfigure interferer for CoMP operation: use virtual cell identity equal to the cell identity for the UE of interest.

```
ue2.NPUCCHID = ue1.NCellID;
```

Measure the interference between the DM-RS signals when using CoMP.

```
interferenceUsingCoMP = abs(sum(ltePUCCH3DRS(ue1,chs1).*conj(ltePUCCH3DRS(ue2,chs2))))
```

```
interferenceUsingCoMP = 8.7702e-15
```

Comparing the correlations between the DM-RS signals for two UEs with and without CoMP, `interferenceUsingCoMP` and `interferenceNoCoMP` respectively. Using CoMP, the interference is reduced to effectively zero.

### Generate PUCCH Format 3 DM-RS for Two Antennas

Generate the PUCCH format 3 DM-RS sequences for two transmit antenna paths. Display the information structure.

Initialize UE-specific and channel configuration structures. Provide an empty vector for the `ack`, indicating there are no HARQ bits for this PUCCH transmission. Generate PUCCH 3 DM-RS and information outputs.

```
ue.NCellID = 1;
ue.NSubframe = 0;
ue.CyclicPrefixUL = 'Normal';
ue.Hopping = 'Off';
ue.Shortened = 0;

chs.ResourceIdx = [0 3];

ack = [];

[drsSeq,info] = ltePUCCH3DRS(ue,chs,ack);
```

Because there are two antennas, the DM-RS sequences are output as a two-column vector and the `info` output structure contains two elements. View `ind` and the size of `info` to confirm this.

```
drsSeq(1:6,:)
```

```
ans = 6×2 complex

   0.5000 + 0.5000i   0.5000 + 0.5000i
   0.1830 + 0.6830i   0.5000 - 0.5000i
  -0.6830 - 0.1830i   0.5000 - 0.5000i
  -0.5000 - 0.5000i  -0.5000 - 0.5000i
  -0.1830 - 0.6830i  -0.5000 + 0.5000i
  -0.1830 + 0.6830i  -0.5000 - 0.5000i
```

```
size(info)
```

```
ans = 1×2

     1     2
```

View the contents of the two `info` structure elements.

```
info(1)
```

```
ans = struct with fields:
               Alpha: [0.5236 2.6180 2.6180 3.1416]
            SeqGroup: [1 1]
              SeqIdx: [0 0]
        NResourceIdx: [0 0]
    NCellCyclicShift: [193 89 101 234]
          OrthSeqIdx: [0 0]
             Symbols: [1.0000 + 0.0000i 1.0000 + 0.0000i 1.0000 + 0.0000i 1.0000 + 0.0000i]
             OrthSeq: [2x2 double]
           NSymbSlot: [5 5]
```

```
info(2)
```

```
ans = struct with fields:
               Alpha: [4.7124 0.5236 1.5708 2.0944]
            SeqGroup: [1 1]
              SeqIdx: [0 0]
        NResourceIdx: [8 10]
    NCellCyclicShift: [193 89 101 234]
          OrthSeqIdx: [3 4]
```

```
    Symbols: [1.0000 + 0.0000i 1.0000 + 0.0000i 1.0000 + 0.0000i 1.0000 + 0.0000i]
    OrthSeq: [2x2 double]
NSymbSlot: [5 5]
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific cell-wide settings, specified as a structure containing the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NCellID** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **NSubframe** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **CyclicPrefixUL** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **NTxAnts** | Optional | 1 (default), 2, 4 | Number of transmission antennas. |
| **Hopping** | Optional | `'Off'` (default), `'Group'` | Frequency hopping method. |
| **Shortened** | Optional | 0 (default), 1 | Option to shorten the subframe by omitting the last symbol, specified as 0 or 1. If `1`, the last symbol of the subframe is not used. For subframes with possible SRS transmission, set `Shortened` to 1 to maintain a standard compliant configuration. |
| **NPUCCHID** | Optional | `NCellID` (default)<br><br>Integer from 0 to 503 | PUCCH virtual cell identity. If this field is not present, `NCellID` is used as the identity. |

Data Types: `struct`

### chs — Channel transmission configuration

structure

PUCCH channel settings, specified as a structure containing the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **ResourceIdx** | Optional | 0 (default), integer from 0 to 549, or vector of integers. | PUCCH resource indices which determine the physical resource blocks, cyclic shift, and orthogonal cover used for transmission ($n^{(3)}_{PUCCH}$). Define one index for each transmission antenna. |

Data Types: `struct`

## Output Arguments

**`seq` — PUCCH format 3 DRS values**
numeric matrix

PUCCH format 3 DRS values, returned as a numeric matrix. The symbols for each antenna are in the columns of `seq`, with the number of columns determined by the number of PUCCH resource indices specified in `chs.ResourceIdx`.

**`info` — PUCCH format 3 DRS information**
structure array

PUCCH format 3 DRS information, returned as a structure array with elements corresponding to each transmit antenna and containing these fields.

**`Alpha` — Reference signal cyclic shift for each OFDM symbol**
two-column vector

Reference signal cyclic shift for each OFDM symbol, returned as a two-column vector. ($\alpha$)

**`SeqGroup` — PUCCH base sequence group number for each slot**
two-column vector

PUCCH base sequence group number for each slot, returned as two-column vector. ($u$)

**`SeqIdx` — PUCCH base sequence number for each slot**
two-column vector

PUCCH base sequence number for each slot, returned as two-column vector. ($v$)

**`NResourceIdx` — PUCCH resource indices for each slot**
vector

PUCCH resource indices for each slot, returned as two-column vector. ($n'$)

**`NCellCyclicShift` — Cell-specific cyclic shift for each OFDM symbol**
vector

Cell-specific cyclic shift for each OFDM symbol, returned as vector. ($n_{cs}^{cell}$)

**`OrthSeqIdx` — Orthogonal sequence index for each slot**
two-column vector

Orthogonal sequence index for each slot, returned as two-column vector. ($\overline{n}_{\mathrm{oc}}$)

**`Symbols` — Modulated data symbols**
vector

Modulated data symbols, returned as a vector. There is one element for each OFDM symbol. ($z$)

Example: [0.7071 + 0.7071i,...]

**`OrthSeq` — Orthogonal sequence for each slot**
numeric matrix

Orthogonal sequence for each slot, returned as a numeric matrix. ($\overline{w}$)

Example: [1.000 + 1.000i,...]

**NSymbSlot — Number of OFDM symbols in each slot**
vector of integers

The number of OFDM symbols in each slot, returned as a vector of integers. ($\left[N_{\mathrm{SF},0}^{\mathrm{PUCCH}}\ N_{\mathrm{SF},1}^{\mathrm{PUCCH}}\right]$)

Data Types: `double`

Data Types: `struct`

# Version History

**Introduced in R2014a**

## See Also

`ltePUCCH3` | `ltePUCCH3Decode` | `ltePUCCH3Indices` | `ltePUCCH3DRSIndices` | `ltePUCCH3PRBS` | `ltePUCCH1DRS` | `ltePUCCH2DRS`

# ltePUCCH3DRSIndices

PUCCH format 3 DRS resource element indices

## Syntax

```
ind = ltePUCCH3DRSIndices(ue,chs)
[ind,info] = ltePUCCH3DRSIndices(ue,chs)
[___] = ltePUCCH3DRSIndices(ue,chs,opts)
```

## Description

`ind = ltePUCCH3DRSIndices(ue,chs)` returns a matrix of resource element indices for the demodulation reference signal (DRS) associated with PUCCH format 3 transmission given structures containing the UE-specific settings, and the channel transmission configuration settings.

`[ind,info] = ltePUCCH3DRSIndices(ue,chs)` also returns a PUCCH information structure array, `info`.

`[___] = ltePUCCH3DRSIndices(ue,chs,opts)` formats the returned indices using the options specified by `opts`.

This syntax supports output options from prior syntaxes.

## Examples

### Generate PUCCH Format 3 DM-RS Indices

Generate PUCCH format 3 DM-RS RE indices for a 5 MHz bandwidth and PUCCH resource index 0.

Initialize UE-specific and channel configuration structures. Generate PUCCH format 3 DM-RS indices.

```
ue.NULRB = 25;
ue.CyclicPrefixUL = 'Normal';

chs.ResourceIdx = 0;

ind = ltePUCCH3DRSIndices(ue,chs);
ind(1:4)
```

```
ans = 4x1 uint32 column vector

   301
   302
   303
   304
```

### Generate PUCCH Format 3 DM-RS Indices for Four Antennas

Generate the PUCCH format 3 DM-RS indices for a 3 MHz bandwidth, and four transmit antenna paths. Display the output information structure.

Initialize UE-specific and channel configuration structures. Generate PUCCH 3 DM-RS indices and information outputs.

```
ue.NULRB = 15;
ue.CyclicPrefixUL = 'Normal';

chs.ResourceIdx = [0 37 4 111];

[ind,info] = ltePUCCH3DRSIndices(ue,chs);
```

Because there are four antennas, the DM-RS indices are output as a four-column vector, and the `info` output structure contains four elements. View `ind` and the size of `info` to confirm this.

```
ind(1:6,:)
```

```
ans = 6x4 uint32 matrix

    181   2833   5221   7873
    182   2834   5222   7874
    183   2835   5223   7875
    184   2836   5224   7876
    185   2837   5225   7877
    186   2838   5226   7878
```

```
size(info)
```

```
ans = 1×2

     1     4
```

View one of the `info` structure elements.

```
info(4)
```

```
ans = struct with fields:
    PRBSet: [11 3]
     RBIdx: 22
```

### Generate PUCCH Format 3 DM-RS Indices Varying Indexing Style

Generate the PUCCH format 3 DM-RS indices for two transmit antenna paths, and output in subscript indexing form.

Initialize UE-specific and channel configuration structures and the indexing option parameter. Generate PUCCH 3 DM-RS indices and information outputs.

```
ue.NULRB = 6;
ue.CyclicPrefixUL = 'Normal';
```

```
chs.ResourceIdx = [0 4];
chs.ResourceSize = 0;
chs.DeltaShift = 1;
chs.CyclicShifts = 0;

[ind,info] = ltePUCCH3DRSIndices(ue,chs,{'sub'});
```

Using 'sub' indexing style, the indices are output in [subcarrier, symbol, antenna] subscript form. View the midpoint of ind and observe the antenna index change.

```
size(ind)

ans = 1×2

    96     3


ind(46:51,:)

ans = 6x3 uint32 matrix

   70   13    1
   71   13    1
   72   13    1
    1    2    2
    2    2    2
    3    2    2


size(info)

ans = 1×2

     1     2
```

Because there are two antennas, the info output structure contains two elements. View one of the `info` structure elements.

```
info(2)

ans = struct with fields:
    PRBSet: [0 5]
     RBIdx: 0
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific settings, specified as a structure containing these fields.

### NULRB — Number of uplink resource blocks

nonnegative integer

Number of uplink resource blocks, specified as a nonnegative integer.

**`CyclicPrefixUL` — Cyclic prefix length for uplink channels**
`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length for uplink channels, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

Data Types: `struct`

**`chs` — Channel transmission configuration**
structure

Channel transmission configuration, specified as a structure containing the following fields.

**`ResourceIdx` — PUCCH resource indices**
0 (default) | 0,...,549 | integer | vector of integers | optional

PUCCH resource indices, specified as an integer or a vector of integers. Values range from 0 to 549. There is one index for each transmission antenna. These indices determine the cyclic shift and orthogonal cover used for transmission. ($n^{(3)}_{PUCCH}$)

Data Types: `struct`

**`opts` — Output format options for resource element indices**
character vector | cell array of character vectors | string array

Output format options for resource element indices, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string) :

| Category | Options | Description |
|---|---|---|
| Indexing style | `'ind'` (default) | The returned indices are in linear index style. |
| | `'sub'` | The returned indices are in `[subcarrier,symbol,port]` subscript row style. |
| Index base | `'1based'` (default) | The returned indices are one-based. |
| | `'0based'` | The returned indices are zero-based. |

Example: `'ind 1based'`, `"ind 1based"`, `{'ind','1based'}`, or `["ind","1based"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

**`ind` — Resource element indices**
integer column vector | three-column integer matrix

Resource element indices, returned as an integer column vector or a three-column integer matrix. By default the indices are returned in one-based linear indexing form that can directly index elements of

a resource matrix. These indices are ordered according to PUCCH format 3 DRS modulation symbol mapping. The `opts` input offers alternative indexing formats. The indices for each antenna are in the columns of `ind`, with the number of columns determined by the number of PUCCH resource indices specified in `chs.ResourceIdx`.

Example: 1,2,3....

Data Types: `uint32`

**`info` — PUCCH format 3 DRS information**
structure array

PUCCH format 3 DRS information, returned as a structure array with elements corresponding to each transmit antenna and containing these fields.

**`PRBSet` — Indices occupied by PRB in each slot of subframe**
nonnegative integer vector

Indices occupied by PRB in each slot of the subframe, returned as a nonnegative integer vector. The indices are zero-based.

Example: [0,5]

Data Types: `double`

**`RBIdx` — PUCCH logical resource block index**
nonnegative integer

PUCCH logical resource block index, returned as a nonnegative integer. ($m$)

Data Types: `double`

Data Types: `struct`

# Version History

**Introduced in R2014a**

## See Also

`ltePUCCH3` | `ltePUCCH3Decode` | `ltePUCCH3Indices` | `ltePUCCH3DRS` | `ltePUCCH3PRBS` | `ltePUCCH1DRSIndices` | `ltePUCCH2DRSIndices`

# ltePUCCH3Indices

PUCCH format 3 resource element indices

## Syntax

```
ind = ltePUCCH3Indices(ue,chs)
[ind,info] = ltePUCCH3Indices(ue,chs)
[___] = ltePUCCH3Indices(ue,chs,opts)
```

## Description

`ind = ltePUCCH3Indices(ue,chs)` returns a column vector of physical uplink control channel (PUCCH) format 3 resource element indices given structures containing the UE-specific settings, and the channel transmission configuration settings.

`[ind,info] = ltePUCCH3Indices(ue,chs)` also returns a PUCCH information structure, `info`.

`[___] = ltePUCCH3Indices(ue,chs,opts)` formats the returned indices using options specified by `opts`.

This syntax supports output options from prior syntaxes.

## Examples

### Generate PUCCH Format 3 Indices

Generate PUCCH format 3 RE indices for a 1.4 MHz bandwidth and PUCCH resource index 0. Use default values for all other parameters.

Initialize UE-specific and channel configuration structures. Generate PUCCH format 3 indices.

```
ue.NULRB = 6;
ue.CyclicPrefixUL = 'Normal';
ue.Shortened = 0;

chs.ResourceIdx = 0;

ind = ltePUCCH3Indices(ue,chs);
ind(1:4)
```

```
ans = 4x1 uint32 column vector

   1
   2
   3
   4
```

### Generate PUCCH Format 3 Indices for Two Antennas

Generate the PUCCH format 3 indices for three transmit antenna paths, and display the information structure output.

Initialize UE-specific and channel configuration structures. Generate PUCCH 3 indices and information outputs.

```
ue.NULRB = 6;
ue.CyclicPrefixUL = 'Normal';
ue.Shortened = 0;

chs.ResourceIdx = [0 2];

[ind,info] = ltePUCCH3Indices(ue,chs);
```

Because there are two antennas, the indices are output as a two-column vector, and the `info` output structure contains two elements.

```
ind(1:5,:)
```

```
ans = 5x2 uint32 matrix

      1   1009
      2   1010
      3   1011
      4   1012
      5   1013
```

```
size(info)
```

```
ans = 1×2

     1     2
```

View one of the `info` structure elements.

```
info(1)
```

```
ans = struct with fields:
       PRBSet: [0 5]
        RBIdx: 0
    NSymbSlot: [5 5]
```

### Generate PUCCH Format 3 Indices Varying Indexing Style

Generate the PUCCH format 3 indices for two transmit antenna paths, and output in subscript indexing form.

Initialize UE-specific and channel configuration structures, and the indexing option parameter. Generate PUCCH 3 indices and information outputs.

```
ue.NULRB = 6;
ue.CyclicPrefixUL = 'Normal';
```

```
ue.Shortened = 0;

chs.ResourceIdx = [0 9];

opts = {'sub'};

[ind,info] = ltePUCCH3Indices(ue,chs,opts);
```

Using `'sub'` indexing style, the indices are output in `[subcarrier, symbol, antenna]` subscript form. View the midpoint of `ind` and observe the antenna index change.

```
size(ind)
```

```
ans = 1×2

   240     3
```

```
ind(118:123,:)
```

```
ans = 6x3 uint32 matrix

   70   14    1
   71   14    1
   72   14    1
   61    1    2
   62    1    2
   63    1    2
```

Because there are two antennas, the `info` output structure contains two elements. View one of the `info` structure elements.

```
size(info)
```

```
ans = 1×2

     1     2
```

```
info(1)
```

```
ans = struct with fields:
       PRBSet: [0 5]
        RBIdx: 0
    NSymbSlot: [5 5]
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific settings, specified as a structure, specified as a structure containing these fields.

### NULRB — Number of uplink resource blocks

nonnegative integer

Number of uplink resource blocks, specified as a nonnegative integer.

**`CyclicPrefixUL` — Cyclic prefix length for uplink channels**
`'Normal'` (default) | optional | `'Extended'`

Cyclic prefix length for uplink channels, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

**`Shortened` — Option to shorten the subframe**
0 (default) | optional | 1

Option to shorten the subframe by omitting the last symbol, specified as 0 or 1. For subframes with possible SRS transmission, this parameter is required. If `1`, the last symbol of the subframe is not used.

Data Types: `struct`

**`chs` — Channel transmission configuration**
structure

Channel transmission configuration, specified as a structure containing the following fields.

**`ResourceIdx` — PUCCH resource indices**
0 (default) | optional | 0,...,549 | integer | vector of integers

PUCCH resource indices, specified as an integer or a vector of integers. Values range from 0 to 549. There is one index for each transmission antenna. These indices determine the cyclic shift and orthogonal cover used for transmission. ($n^{(3)}_{PUCCH}$)

Data Types: `struct`

**`opts` — Output format options for resource element indices**
character vector | cell array of character vectors | string array

Output format options for resource element indices, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string) :

| Category | Options | Description |
|---|---|---|
| Indexing style | `'ind'` (default) | The returned indices are in linear index style. |
| | `'sub'` | The returned indices are in `[subcarrier,symbol,port]` subscript row style. |
| Index base | `'1based'` (default) | The returned indices are one-based. |
| | `'0based'` | The returned indices are zero-based. |

Example: `'ind 1based'`, `"ind 1based"`, `{'ind','1based'}`, or `["ind","1based"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `ind` — PUCCH format 3 resource element indices

integer column vector | three-column integer matrix

PUCCH format 3 resource element indices, returned as an integer column vector or a three-column integer matrix. By default, the indices are returned in one-based linear indexing form that can directly index elements of a resource matrix. These indices are ordered according to PUCCH format 3 modulation symbol mapping as specified in TS 36.211 [1], Section 5.4. The `opts` input offers alternative indexing formats. The indices for each antenna are in the columns of `ind`, with the number of columns determined by the number of PUCCH resource indices specified in `chs.ResourceIdx`.

Example: [1,2,3,4...]

### `info` — PUCCH format 3 information

scalar structure | structure array

PUCCH format 3 information, returned as a structure array with elements corresponding to each transmit antenna and containing these fields.

### `PRBSet` — Set of PRB indices

column vector | two-column matrix

Set of PRB indices, returned as a column vector or two-column matrix corresponding to the resource allocations.

- When returned as a column vector, the resource allocation is the same in both slots of the subframe.
- When returned as a two-column matrix, the resource allocations can vary for each slot in the subframe.

The PRB indices are zero-based.

Example: [0,5]

### `RBIdx` — PUCCH logical resource block index

nonnegative integer

PUCCH logical resource block index, returned as a nonnegative integer. ($m$)

### `NSymbSlot` — Number of OFDM symbols in each slot

vector of integers

The number of OFDM symbols in each slot, returned as a vector of integers. ($\left[N_{\mathrm{SF},0}^{\mathrm{PUCCH}}\ N_{\mathrm{SF},1}^{\mathrm{PUCCH}}\right]$)

Data Types: `struct`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePUCCH3` | `ltePUCCH3Decode` | `ltePUCCH3DRS` | `ltePUCCH3DRSIndices` | `ltePUCCH3PRBS` | `ltePUCCH1Indices` | `ltePUCCH2Indices`

# ltePUCCH3PRBS

PUCCH format 3 pseudorandom scrambling sequence

## Syntax

```
[seq,cinit] = ltePUCCH3PRBS(ue,n)
[seq,cinit] = ltePUCCH3PRBS(ue,n,mapping)

[subseq,cinit] = ltePUCCH3PRBS(ue,pn)
[subseq,cinit] = ltePUCCH3PRBS(ue,pn,mapping)
```

## Description

`[seq,cinit] = ltePUCCH3PRBS(ue,n)` returns a column vector containing the first `n` outputs of the Physical Uplink Control Channel (PUCCH) format 3 scrambling sequence when initialized according to UE-specific settings, `ue`, which must be a structure. It also returns an initialization value `cinit` for the pseudorandom binary sequence (PRBS) generator.

`[seq,cinit] = ltePUCCH3PRBS(ue,n,mapping)` allows control over the format of the returned sequence, `seq`, with the input `mapping`.

`[subseq,cinit] = ltePUCCH3PRBS(ue,pn)` returns a subsequence of a full PRBS sequence, specified by `pn`.

`[subseq,cinit] = ltePUCCH3PRBS(ue,pn,mapping)` allows additional control over the format of the returned subsequence, `subseq`, with the input `mapping`.

## Examples

### Scramble Random PUCCH3 Codeword

Scramble a random PUCCH3 codeword.

Create a ue-specific configuration structure. Generate a codeword. Generate a PUCCH3 pseudorandom scrambling sequence the same length as the codeword.

```
ue.NCellID = 1;
ue.NSubframe = 0;
ue.RNTI = 1;
cw = randi([0 1],48,1);
seq = ltePUCCH3PRBS(ue,length(cw));
```

Scramble codeword with PDCCH PRBS.

```
scrambled = xor(seq,cw);
```

### Scramble UCI3 Codeword

Scramble a UCI3 codeword with a PUCCH3 pseudorandom scrambling sequence.

Create a ue-specific configuration structure. Generate a UCI3 codeword. The UCI3 codeword is 48 bits long.

```
ue.NCellID = 1;
ue.NSubframe = 0;
ue.RNTI = 1;

txAck = [1;0;0;1];
cw = lteUCI3Encode(txAck);
cwLength = length(cw)
```

```
cwLength = 48
```

Generate a PUCCH3 pseudorandom scrambling sequence the same length as the codeword. Scramble codeword with PDCCH PRBS.

```
seq = ltePUCCH3PRBS(ue,length(cw));
scrambled = xor(seq,cw);
```

### Generate PUCCH Format 3 Pseudorandom Scrambling Sequence

This example shows the generation of unsigned and signed PUCCH format 3 pseudorandom scrambling sequences.

Initialize ue specific parameters.

```
ue = struct('NCellID',1,'NSubframe',0,'RNTI',1);
```

Generate a PUCCH format 3 pseudorandom scrambling sequence.

```
pucch3Seq = ltePUCCH3PRBS(ue,5)
```

```
pucch3Seq = 5x1 logical array

   1
   1
   0
   0
   1
```

Generate a signed PUCCH format 3 pseudorandom scrambling sequence.

```
pucch3Seq = ltePUCCH3PRBS(ue,5,'signed')
```

```
pucch3Seq = 5×1

    -1
    -1
     1
     1
```

```
-1
```

## Input Arguments

### `ue` — UE-specific settings

structure

UE-specific settings, specified as a structure having the following fields.

### `NCellID` — Physical layer cell identity

integer (0...503)

Physical layer cell identity, specified as an integer between 0 and 503.

Data Types: `double`

### `NSubframe` — Subframe number

integer

Subframe number, specified as an integer.

Data Types: `double`

### `RNTI` — Radio Network Temporary Identifier

integer

Radio Network Temporary Identifier (16-bit), specified as an integer.

Data Types: `double`

Data Types: `struct`

### `n` — Number of elements in returned sequence

numeric scalar

Number of elements in returned sequence, `seq`, specified as a numeric scalar.

Data Types: `double`

### `pn` — Range of elements in returned subsequence

row vector

Range of elements in returned subsequence, `subseq`, specified as a row vector of `[p n]`. The subsequence returns `n` values of the PRBS generator, starting at position `p` (0-based).

Data Types: `double`

### `mapping` — Output sequence formatting

`'binary'` (default) | `'signed'`

Output sequence formatting, specified as `'binary'` or `'signed'`. `mapping` controls the format of the returned sequence.

- `'binary'` maps `true` to 1 and `false` to 0.
- `'signed'` maps `true` to –1 and `false` to 1.

Data Types: `char` | `string`

## Output Arguments

### `seq` — PUCCH format 3 scrambling sequence

logical column vector | numeric column vector

PUCCH format 3 scrambling sequence, returned as a logical column vector or a numeric column vector. `seq` contains the first `n` outputs of the scrambling sequence. If you set `mapping` to `'signed'`, the output data type is double. Otherwise, the output data type is logical.

Data Types: `logical` | `double`

### `subseq` — PUCCH format 3 scrambling subsequence

logical column vector | numeric column vector

PUCCH format 3 scrambling subsequence, returned as a logical column vector or a numeric column vector. `subseq` contains the values of the PRBS generator specified by `pn`. If you set `mapping` to `'signed'`, the output data type is `double`. Otherwise, the output data type is `logical`.

Data Types: `logical` | `double`

### `cinit` — Initialization value for PRBS generator

numeric scalar

Initialization value for PRBS generator, returned as a numeric scalar.

Data Types: `uint32`

# Version History

**Introduced in R2014a**

## See Also

`ltePUCCH3` | `ltePUCCH3Decode` | `ltePUCCH3Indices` | `ltePUCCH3DRS` | `ltePUCCH3DRSIndices` | `ltePRBS` | `ltePUCCH2PRBS`

# ltePUSCH

Physical uplink shared channel

## Syntax

```
sym=ltePUSCH(ue,chs,cws)
```

## Description

`sym=ltePUSCH(ue,chs,cws)` returns a vector containing the Physical Uplink Shared Channel (PUSCH) complex symbols for UE-specific settings, `ue`, PUSCH channel-specific configuration, `chs`, and the codeword or codewords contained in `cws`. The size of the matrix `sym` is *N*-by-*P*. Where *N* is the number of modulation symbols for one antenna port and *P* is the number of transmission antennas.

## Examples

### Create PUSCH Complex Symbols

Generate PUSCH symbols for TS36.104 Uplink FRC A3-3 with 3MHz bandwidth.

Initialize UE specific (`ue`) and channel (`pusch`) configuration structures, and fixed reference channel (`frc`).

```
ue.NCellID = 1;
ue.NSubframe = 0;
ue.RNTI = 1;

pusch.Modulation = 'QPSK';
pusch.PRBSet = [0:14].';
pusch.RV = 0;

frc = lteRMCUL('A3-3');
```

Generate transport block (`trBlk`), UL-SCH codewords (`cw`), and PUSCH symbols (`puschSym`).

```
trBlk  = randi([0,1],frc.PUSCH.TrBlkSizes(1),1);
cw = lteULSCH(ue,pusch,trBlk );
puschSym = ltePUSCH(ue,pusch,cw);
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific settings, specified as a structure having the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NCellID** | Required | Nonnegative integer | Physical layer cell identity |
| **NSubframe** | Required | Integer | Subframe number |
| **RNTI** | Required | Integer | Radio network temporary identifier (RNTI) value (16 bits) |
| **NTxAnts** | Optional | 1 (default), 2, 4 | Number of transmission antennas. |

Data Types: `struct`

### chs — PUSCH channel-specific configuration

structure

PUSCH channel-specific configuration, specified as a structure having the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **Modulation** | Required | `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'` | Modulation format, specified as a character vector or string scalar for one codeword, or a cell array of character vectors or string array for two codewords. |
| **PRBSet** | Required | Integer column vector or two-column matrix | Physical Resource Block (PRB) indices, specified as a column vector or two-column matrix, corresponding to the slot wise resource allocations for this PUSCH.<br><br>If a column vector is provided for `PRBSet`, the resource allocation is the same in both slots of the subframe. The two-column matrix can be used to specify differing PRBs for each slot in a subframe. The PRB indices are zero-based. |
| **NLayers** | Optional | 1 (default), 2, 3, 4 | Number of transmission layers. |
| The following field is required only when `ue.NTxAnts` is set to 2 or 4. Acceptable values for `PMI` depend upon `ue.NTxAnts` and `NLayers`. | | | |
| **PMI** | Optional | Numeric scalar (0...23)<br><br>0 (default) | Scalar precoder matrix indication (PMI) to be used during precoding |

Data Types: `struct`

### cws — Codeword bit values

vector | cell array

Codeword bit values to be modulated, specified as a vector of bit values for one codeword, or a cell array containing one or two vectors of bit values corresponding to the one or two codewords.

Data Types: `double`

## Output Arguments

**`sym` — PUSCH symbols**
complex-valued numeric matrix

PUSCH symbols, returned as a complex-valued numeric matrix of size *N*-by-*P*. *N* is the number of modulation symbols for one antenna port. *P* is the number of transmission antennas.

Data Types: `double`

# Version History

**Introduced in R2013b**

## References

[1] 3GPP TS 36.104. “Evolved Universal Terrestrial Radio Access (E-UTRA); Base Station (BS) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePUSCHDecode` | `ltePUSCHIndices` | `ltePUSCHDRS` | `ltePUSCHDRSIndices` | `ltePUSCHPrecode` | `lteULScramble` | `lteULPrecode` | `lteULSCH` | `lteULPMIInfo`

# ltePUSCHDecode

Physical uplink shared channel decoding

## Syntax

```
[cws,symbols] = ltePUSCHDecode(ue,chs,sym)
[cws,symbols] = ltePUSCHDecode(ue,chs,sym,hest,noiseest)
[cws,symbols] = ltePUSCHDecode(ue,chs,sym,hest,noiseest,alg)
```

## Description

`[cws,symbols] = ltePUSCHDecode(ue,chs,sym)` returns soft bit vector or cell array of soft bit vectors `cws` containing the received codeword estimates and received constellation of complex symbol vector `symbols`. The output results from decoding of physical uplink shared channel (PUSCH) complex symbols `sym` for UE-specific settings `ue` and channel transmission configuration `chs`.

The function performs the inverse of PUSCH processing. See TS 36.211, Section 5.3 [1] or `ltePUSCH` for details.

Multiple codewords can be parameterized by two different forms of the `chs` structure. Each codeword can be defined by separate elements of a 1-by-2 structure array, or the codeword parameters can be combined together in the fields of a single scalar, or 1-by-1, structure. Any scalar field values apply to both codewords and a scalar `NLayers` is the total number. For details, see “UL-SCH Parameterization” .

If UCI control information, such as RI or HARQ-ACK, is present in the received complex PUSCH symbols, then this function performs the descrambling of the placeholder bits by establishing the correct locations with the help of the UCI-related parameters present in `chs`.

`[cws,symbols] = ltePUSCHDecode(ue,chs,sym,hest,noiseest)` also specifies channel estimate, `hest` and noise estimate `noiseest`. In this case, `sym` is an *M*-by-`NRxAnts` matrix, where *M* is the number of symbols per antenna and `NRxAnts` is the number of receive antennas. When `ue.NTxAnts` is greater than 1, the reception is performed using an MMSE equalizer, equalizing between transmitted and received layers. When `ue.NTxAnts` is 1, the reception is performed using MMSE equalization on the received antennas.

`[cws,symbols] = ltePUSCHDecode(ue,chs,sym,hest,noiseest,alg)` provides control over weighting the output soft bits with Channel State Information (CSI) calculated during the equalization stage using algorithmic configuration structure, `alg`.

## Examples

### Decode PUSCH Symbols from FRC

Decode the PUSCH modulation symbols contained in the output of a Fixed Reference Channel (FRC).

```
frc = lteRMCUL('A3-2');
trData = randi([0,1],frc.PUSCH.TrBlkSizes(1),1);
[waveform,reGrid] = lteRMCULTool(frc,trData);
```

```
puschIndices = ltePUSCHIndices(frc,frc.PUSCH);
rxCw = ltePUSCHDecode(frc,frc.PUSCH,reGrid(puschIndices));
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific settings, specified as a structure having the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NCellID** | Required | Integer | Physical layer cell identity |
| **NSubframe** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **RNTI** | Required | 0 (default), scalar integer | Radio network temporary identifier (RNTI) value (16 bits) |
| **CyclicPrefixUL** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length. |
| **NTxAnts** | Optional | 1 (default), 2, 4 | Number of transmission antennas. |
| **Shortened** | Optional | 0 (default), 1 | Option to shorten the subframe by omitting the last symbol, specified as 0 or 1. If `1`, the last symbol of the subframe is not used. For subframes with possible SRS transmission, set `Shortened` to 1 to maintain a standard compliant configuration. |

Data Types: `struct`

### chs — Channel transmission configuration

scalar structure | structure array

Channel transmission configuration, specified as a scalar structure or a structure array. `chs` is the PUSCH channel-specific structure having these fields. If `UCI` is present in the transmitted PUSCH to be decoded, the optional fields, `ORI`, `OACK`, `QdRI`, and `QdACK`,must be configured in the `chs` structure.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **Modulation** | Required | `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'` | Modulation format |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **PRBSet** | Required | Integer column vector or two-column matrix | Physical Resource Block (PRB) indices, specified as a column vector or two-column matrix, corresponding to the slot wise resource allocations for this PUSCH.<br><br>If a column vector is provided for `PRBSet`, the resource allocation is the same in both slots of the subframe. The two-column matrix can be used to specify differing PRBs for each slot in a subframe. The PRB indices are zero-based. |
| **NLayers** | Optional | 1 (default), 2, 3, 4 | Number of transmission layers. |
| The following field is required only when `ue.NTxAnts` is set to 2 or 4. Acceptable values for `PMI` depend upon `ue.NTxAnts` and `NLayers`. | | | |
| **PMI** | Optional | Numeric scalar (0...23)<br><br>0 (default) | Scalar precoder matrix indication (PMI) to be used during precoding |
| **ORI** | Optional | Integer<br><br>0 (default) | Number of uncoded RI bits |
| **OACK** | Optional | nonnegative scalar integer, 0 (default) | Number of uncoded HARQ-ACK bits. |
| **QdRI** | Optional | Integer<br><br>0 (default) | Number of coded RI symbols in UL-SCH, specified as an integer. Optional. ($Q'\_RI$) |
| **QdACK** | Optional | nonnegative scalar integer<br><br>0 (default) | Number of coded HARQ-ACK symbols in UL-SCH ($Q'\_ACK$), specified as an integer. Optional. |

Data Types: `struct`

### `sym` — PUSCH symbols

complex-valued matrix

PUSCH symbols, specified as a complex-valued matrix of size $M$-by-$P$, where $M$ is the number of symbols per antenna or layer and $P$ is the number of transmission antennas.

Data Types: `double`
Complex Number Support: Yes

### `hest` — Channel estimate

3-D numeric array

Channel estimate, specified as a 3-D numeric array of size *M*-by-`NRxAnts`-by-`NTxAnts`. Where *M* is the number of symbols per antenna, `NRxAnts` is the number of receive antennas, and `NTxAnts` is the number of transmit antennas ports, given by `ue.NTxAnts`.

Data Types: `double`

**`noiseest` — Noise estimate**
numeric scalar

Noise estimate, specified as a numeric scalar. This argument is an estimate of the noise power spectral density per RE on received subframe. The `lteULChannelEstimate` function provides such an estimate.

Data Types: `double`

**`alg` — Algorithmic configuration**
structure

Algorithmic configuration, specified as a structure having the following field.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`CSI`** | Optional | `'On'` (default), `'Off'` | Flag provides control over weighting the soft values that are used to determine the output values with the channel state information (CSI) calculated during the equalization process. If `'On'`, soft values are weighted by CSI. |

Data Types: `struct`

## Output Arguments

**`cws` — Codewords**
column vector | cell array of column vectors

Codewords, returned as a column vector or a cell array of column vectors. The soft bit vectors contain the received codeword estimates.

Data Types: `double`

**`symbols` — Received constellation of symbols**
complex-valued column vector

Received constellation of symbols, received as a complex-valued column vector.

Data Types: `double`

# Version History

**Introduced in R2013b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePUSCH` | `ltePUSCHIndices` | `ltePUSCHDRS` | `ltePUSCHDRSIndices` | `ltePUSCHDeprecode` | `lteULDescramble` | `lteULDeprecode` | `lteULSCHDecode` | `lteULPMIInfo`

# ltePUSCHDeprecode

PUSCH MIMO deprecoding onto transmission layers

## Syntax

```
out = ltePUSCHDeprecode(in,nu,codebook)
out = ltePUSCHDeprecode(chs,in)
```

## Description

`out = ltePUSCHDeprecode(in,nu,codebook)` deprecodes the precoded symbol matrix, `in`, onto `nu` layers. It performs deprecoding using matrix pseudo inversion, to undo the processing described in TS 36.211, Section 5.3.3A [1]. This function returns an *M*-by-`nu` matrix, `out`, containing `nu` layers with *M* symbols in each layer. The deprecoder transposes the operation defined in TS 36.211, Section 5.3.3A, specifically the symbols for layers and antennas lie in columns rather than rows. The input argument `in` is an *N*-by-*P* matrix, where *P* is the number of transmission antennas and *N* is the number of symbols per antenna. When *P* is 2 or 4, deprecoding for spatial multiplexing is applied with the scalar codebook index, `codebook`. TS 36.211, Section 5.3.3A [1] specifies the codebook matrix corresponding to a particular index.

`out = ltePUSCHDeprecode(chs,in)` deprecodes the precoded symbol matrix, `in`, according to channel transmission configuration, `chs`.

## Examples

### Deprecode PUSCH Symbols

By precoding with an identity matrix, we can gain access to the precoding matrices themselves. The precoded matrix is first generated with codebook index 0 for 4 layers and 4 antennas. The precoded matrix is then deprecoded, resulting in an identity matrix.

```
nLayers = 4;
nAntennas = 4;
codeBookIdx = 0;
precodingMatrix = ltePUSCHPrecode(eye(nLayers),nAntennas,codeBookIdx);
out = ltePUSCHDeprecode(precodingMatrix,nLayers,codeBookIdx)
```

```
out = 4×4 complex

   1.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   1.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   1.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   1.0000 + 0.0000i
```

## Input Arguments

### `in` — Precoded symbol input

complex-valued numeric matrix

Precoded symbol input, specified as an *N*-by-*P* complex-valued numeric matrix. Where *P* is the number of transmission antennas and *N* is the number of symbols per antenna.

Example: `[0.5000 + 0.0000i 0.5000 + 0.0000i; 0.5000 + 0.0000i -0.5000 + 0.0000i]`

Data Types: `double`
Complex Number Support: Yes

**`nu` — Number of layers**
1 | 2 | 3 | 4

Number of layers, specified as 1, 2, 3, or 4.

Example: 2

Data Types: `double`

**`codebook` — Codebook index**
numeric scalar

Codebook index, specified as a numeric scalar. When the number of transmit antennas, *P*, is 1, this input argument is ignored.

Data Types: `double`

**`chs` — Channel transmission configuration**
structure

Channel transmission configuration, specified as a structure having the following fields.

**`NLayers` — Number of transmission layers**
1 (default) | optional | 2 | 3 | 4

Number of transmission layers, specified as an integer from 1 through 4. Optional.

Data Types: `double`

**`PMI` — Precoder matrix indication**
0 (default) | optional | numeric scalar (0...23)

Precoder matrix indication, specified as a numeric scalar between 0 (default) and 23. Only required if the number of transmission antennas, *P*, is 2 or 4. Acceptable values for `PMI` depend upon *P* and the number of transmission layers, `NLayers`. The scalar PMI is used during deprecoding.

Data Types: `double`

Data Types: `struct`

## Output Arguments

**`out` — Deprecoded output**
numeric matrix

Deprecoded output, returned as an *M*-by-`nu` matrix, containing `nu` layers with *M* symbols in each layer.

Data Types: `double`

# Version History

**Introduced in R2013b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePUSCHPrecode` | `ltePUSCH` | `ltePUSCHDecode` | `ltePUSCHIndices` | `ltePUSCHDRS` | `ltePUSCHDRSIndices` | `lteULDeprecode` | `lteULPMIInfo`

# ltePUSCHDRS

PUSCH demodulation reference signal

## Syntax

```
[antseq,info,layerseq] = ltePUSCHDRS(ue,chs)
```

## Description

`[antseq,info,layerseq] = ltePUSCHDRS(ue,chs)` returns the physical uplink shared channel (PUSCH) transmission demodulation reference signal (DM-RS) antenna sequence values,`antseq`, the layer sequence values, `layerseq`, and the information structure, `info`, given input structures containing UE-specific settings, and the channel transmission configuration settings.

When the number of transmission antennas is greater than one, the DM-RS is precoded using spatial multiplexing.

For short base reference sequences, such as those used with PUSCH allocations of 1 or 2 PRBs, and when `chs.PRBSet` is empty, Zadoff-Chu sequences are not used. In this case, `RootSeq` and `NZC` are set to –1. If `antseq` is empty, such as when the input `PRBSet` is empty, the `info` structure contains all fields, but each field is either empty for vector fields or –1 for scalar fields.

## Examples

### Generate PUSCH DM-RS

Generate the PUSCH Demodulation Reference Signal (DM-RS) values for UE-specific settings.

Initialize UE specific (`ue`) and channel (`chs`) configuration structures. Generate PUSCH DM-RS values.

```
ue.NCellID = 1;
ue.NSubframe = 0;
ue.CyclicPrefixUL = 'Normal';
ue.Hopping = 'Off';
ue.SeqGroup = 0;
ue.CyclicShift = 0;
ue.NTxAnts = 1;

chs.PRBSet = (0:5).';
chs.NLayers = 1;
chs.OrthCover = 'Off';
chs.DynCyclicShift = 0;

puschSeq = ltePUSCHDRS(ue,chs);
puschSeq(1:10)
```

```
ans = 10×1 complex

   1.0000 + 0.0000i
```

```
  -0.0810 + 0.9967i
  -0.9610 + 0.2766i
  -0.8839 - 0.4677i
  -0.6886 - 0.7251i
  -0.7692 - 0.6390i
  -0.9912 - 0.1324i
  -0.6447 + 0.7645i
   0.6779 + 0.7352i
   0.4872 - 0.8733i
```

### Generate PUSCH DM-RS Using Alternate IDs

Demonstrate Uplink Release 11 coordinated multipoint (CoMP) operation. To avoid intercell interference, use a virtual cell identity (NPUSCHID) and a distinct DM-RS cyclic shift hopping identity (NDMRSID) for a potentially interfering UE in a neighboring cell.

Configure the UE of interest: UE 1 in cell 1.

```
ue1.NCellID = 1;
ue1.NSubframe = 0;
ue1.CyclicPrefixUL = 'Normal';
ue1.NTxAnts = 1;
ue1.Hopping = 'Off';
ue1.SeqGroup = 0;
ue1.CyclicShift = 0;

chs1.PRBSet = (0:5).';
chs1.NLayers = 1;
chs1.DynCyclicShift = 0;
chs1.OrthCover = 'Off';
```

Configure the interferer: UE 2 in cell 2.

```
ue2.NCellID = 2;
ue2.NSubframe = 0;
ue2.CyclicPrefixUL = 'Normal';
ue2.NTxAnts = 1;
ue2.Hopping = 'Off';
ue2.SeqGroup = 0;
ue2.CyclicShift = 0;

chs2.PRBSet = (0:5).';
chs2.NLayers = 1;
chs2.DynCyclicShift = 0;
chs2.OrthCover = 'Off';
```

Measure the interference between the DM-RS signals.

```
interferenceNoCoMP = ...
    abs(sum(ltePUSCHDRS(ue1,chs1).*conj(ltePUSCHDRS(ue2,chs2))));
```

Reconfigure for CoMP operation. Use a virtual cell identity equal to the cell identity for the UE of interest. Configure the two UEs with different cyclic shift hopping patterns using the DM-RS identity parameter.

```
ue1.NDMRSID = 1;
ue2.NPUSCHID = ue1.NCellID;
ue2.NDMRSID = 2;
```

Measure the interference between the DM-RS signals when using CoMP.

```
interferenceUsingCoMP = ...
    abs(sum(ltePUSCHDRS(ue1,chs1).*conj(ltePUSCHDRS(ue2,chs2))));
```

Compare the correlations between the DM-RS signals for the two UEs with and without CoMP, `interferenceUsingCoMP` and `interferenceNoCoMP`, respectively.

```
interferenceUsingCoMP
```

```
interferenceUsingCoMP = 1.0482e-13
```

```
interferenceNoCoMP
```

```
interferenceNoCoMP = 21.3188
```

With CoMP, the interference is reduced to effectively zero.

## Input Arguments

### ue — UE-specific settings

structure

UE-specific settings, specified as a structure. `ue` can contain the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NCellID`** | Required | Nonnegative integer | Physical layer cell identity |
| **`NSubframe`** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **`CyclicPrefixUL`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length for uplink. |
| **`NTxAnts`** | Optional | 1 (default), 2, 4 | Number of transmission antennas. |
| **`Hopping`** | Optional | `'Off'` (default), `'Group'`, or `'Sequence'` | Frequency hopping method. |
| **`SeqGroup`** | Optional | 0 (default), integer from 0 to 29 | PUSCH sequence group assignment ($\Delta_{SS}$).<br><br>Used only if `NDMRSID` or `NPUSCHID` is absent. |
| **`CyclicShift`** | Optional | 0 (default), integer from 0 to 7 | Number of cyclic shifts used for PUSCH DM-RS (yields $n_{DMRS}^{(1)}$). |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NPUSCHID** | Optional | 0 (default), nonnegative scalar integer from 0 to 509 | PUSCH virtual cell identity. If this field is not present, `NCellID` is used for group hopping sequence-shift pattern initialization.<br><br>See note. |
| **NDMRSID** | Optional | 0 (default), nonnegative scalar integer from 0 to 509 | DM-RS identity for cyclic shift hopping ($n_{ID}^{csh\_DMRS}$). If this field is not present, `NCellID` is used for cyclic shift hopping initialization.<br><br>See note. |
| **Note**<br><br>**1** The pseudorandom sequence generator for cyclic shift hopping is initialized according to `NDMRSID`, if present — otherwise it is initialized according to the cell identity `NCellID` and the sequence group assignment `SeqGroup`. Similarly, the sequence-shift pattern for group hopping is initialized according to `NPUSCHID`, if present — otherwise it is initialized according to `NCellID` and `SeqGroup`. | | | |

Data Types: `struct`

### `chs` — Channel transmission configuration

structure

PUSCH channel configuration, specified as a structure that can contain the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **PRBSet** | Required | Integer column vector or two-column matrix | Physical resource block set, specified as an integer column vector or two-column matrix. This parameter field contains the zero-based physical resource block (PRB) indices corresponding to the slot-wise resource allocations for this PUSCH.<br><br>If `PRBSet` is a column vector, the resource allocation is the same in both slots of the subframe. To specify differing PRBs for each slot in a subframe, use a two-column matrix. The PRB indices are zero-based. |
| **NLayers** | Optional | 1 (default), 2, 3, 4 | Number of transmission layers. |
| **DynCyclicShift** | Optional | 0 (default), integer from 0 to 7 | Cyclic shift for DM-RS (yields $n_{DMRS}^{(2)}$). |
| **OrthoCover** | Optional | `'Off'` (default), `'On'` | Applies (`'On'`), or does not apply (`'Off'`), orthogonal cover sequence *w* (*Activate-DMRS-with OCC*). |
| The following field is applicable only when `ue.NTxAnts` is set to 2 or 4. | | | |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **PMI** | Optional | 0 (default), integer from 0 to 23 | Scalar precoder matrix indication (PMI) used during precoding of the DM-RS reference symbols. |

Data Types: `struct`

## Output Arguments

### `antseq` — PUSCH DM-RS sequence

*M*-by-*P* matrix

PUSCH DM-RS sequence values, returned as an *M*-by-*P* complex-valued matrix. *M* is the number of DM-RS symbols per antenna, and *P* is the number of transmission antennas. When *P* is greater than one, the DM-RS is precoded using spatial multiplexing.

Data Types: `double`

### `info` — Information about PUSCH DM-RS

structure array

Information about PUSCH DM-RS, returned as a structure array, with one element per transmission layer, having the following fields.

#### `Alpha` — Reference signal cyclic shift

row vector

Reference signal cyclic shift for each slot, returned as a row vector. ($\alpha$)

`Alpha` is proportional to `NCS`, $\alpha = \frac{2\pi n_{cs,\lambda}}{12}$.

Data Types: `double`

#### `SeqGroup` — Base sequence group number

row vector

Base sequence group number for each slot, returned as a row vector. ($u$)

Data Types: `double`

#### `SeqIdx` — Base sequence number

row vector

Base sequence number for each slot, returned as a row vector. ($v$)

Data Types: `double`

#### `RootSeq` — Root Zadoff-Chu sequence index

row vector

Root Zadoff-Chu sequence index for each slot, returned as a row vector. ($q$)

Data Types: `double`

**NCS — Cyclic shift values for each slot**
two-column vector

Cyclic shift values for each slot, returned as a two-column vector ($n_{\mathrm{cs},\lambda}$).

Data Types: `double`

**NZC — Zadoff-Chu sequence length**
integer

Zadoff-Chu sequence length, returned as an integer. ($N_{ZC}^{RS}$)

Data Types: `double`

**N1DMRS — Component of reference signal cyclic shift**
integer

Component of the reference signal cyclic shift signaled from higher layers, returned as an integer. ($n_{DMRS}^{(1)}$)

Data Types: `double`

**N2DMRS — Component of the reference signal cyclic shift**
integer

Component of the reference signal cyclic shift signaled from the most recent DCI format 0 message, returned as an integer. ($n_{DMRS}^{(2)}$)

Data Types: `double`

**NPRS — Cell-specific component of reference signal cyclic shift**
row vector

Cell-specific component of the reference signal cyclic shift for each slot, returned as a row vector. ($n_{\mathrm{PRS}}$ in LTE Release 8 and 9, $n_{\mathrm{PN}}$ in LTE Release 10 and beyond)

Data Types: `double`

**OrthSeq — Orthogonal cover value**
row vector

Orthogonal cover value for each slot, specified as a row vector. ($w$)

Data Types: `double`

Data Types: `struct`

**layerseq — PUSCH DM-RS sequence by layers**
*M*-by-*NU* matrix

PUSCH DM-RS sequence by layers, returned as an *M*-by-*NU* complex matrix. *M* is the number of DM-RS symbols per layer, and *NU* is the number of transmission layers. If the number of transmission antennas is greater than one, the DM-RS is precoded using spatial multiplexing.

Data Types: `double`

## Version History

**Introduced in R2013b**

## See Also

`ltePUSCH` | `ltePUSCHDecode` | `ltePUSCHPrecode` | `ltePUSCHDeprecode` | `ltePUSCHIndices` | `ltePUSCHDRSIndices` | `lteULPMIInfo`

# ltePUSCHDRSIndices

PUSCH DM-RS resource element indices

## Syntax

```
ind = ltePUSCHDRSIndices(ue,chs)
ind = ltePUSCHDRSIndices(ue,chs,opts)
```

## Description

`ind = ltePUSCHDRSIndices(ue,chs)` returns a matrix of resource element (RE) indices for the demodulation reference signal (DM-RS) associated with the physical uplink shared channel (PUSCH) transmission, given structures containing the UE-specific settings and the channel transmission configuration settings.

If indices for a number of layers, *NU*, are required, rather than indices for `NTxAnts`, the first *NU* columns of the output can be used. The first *NU* columns of `ind` are the appropriate indices for the `layerseq` output from the `ltePUSCHDRS` function.

`ind = ltePUSCHDRSIndices(ue,chs,opts)` formats the returned indices using options specified by `opts`.

## Examples

### Generate PUSCH DM-RS RE Indices

Generate PUSCH DM-RS resource element indices for the uplink reference measurement channel A3-1.

```
ue = lteRMCUL('A3-1');
puschInd = ltePUSCHDRSIndices(ue,ue.PUSCH);
puschInd(1:4)
```

```
ans = 4x1 uint32 column vector

   217
   218
   219
   220
```

### Generate PUSCH DM-RS RE Indices Varying Output Format

Generate the zero-based PUSCH DM-RS indices for the uplink reference measurement channel A3-3.

```
ue = lteRMCUL('A3-3');
puschInd = ltePUSCHDRSIndices(ue,ue.PUSCH,{'0based','ind'});
puschInd(1:4)
```

```
ans = 4x1 uint32 column vector

   540
   541
   542
   543
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific settings, specified as a structure having the following fields.

### NULRB — Number of uplink resource blocks

integer

Number of uplink resource blocks, specified as an integer.

Data Types: `double`

### CyclicPrefixUL — Cyclic prefix length for uplink

`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length for uplink (UL), specified as either `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

### NTxAnts — Number of transmission antennas

1 (default) | 2 | 4 | optional

Number of transmission antennas, specified as 1, 2, or 4.

Data Types: `double`

Data Types: `struct`

### chs — Channel transmission configuration

structure

Channel transmission configuration, specified as a structure having the following fields.

### PRBSet — Physical resource block indices

column vector | two-column matrix

Physical resource block (PRB) indices, specified as a column integer vector or two-column integer matrix. The PRB indices correspond to the slot-wise resource allocations for this PUSCH.

- When specified as a column vector, the resource allocation is the same in both slots of the subframe.
- When specified as a two-column matrix, the resource allocations can vary for each slot in the subframe.

The PRB indices are zero-based.

Data Types: `double`

Data Types: `struct`

### `opts` — Output format options for resource element indices

character vector | cell array of character vectors | string array

Output format options for resource element indices, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string) :

| Category | Options | Description |
|---|---|---|
| Indexing style | `'ind'` (default) | The returned indices are in linear index style. |
| | `'sub'` | The returned indices are in `[subcarrier,symbol,port]` subscript row style. |
| Index base | `'1based'` (default) | The returned indices are one-based. |
| | `'0based'` | The returned indices are zero-based. |

Example: `'ind 1based'`, `"ind 1based"`, `{'ind','1based'}`, or `["ind","1based"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `ind` — PUSCH DM-RS resource element indices

integer matrix

PUSCH DM-RS resource element indices, returned as an $N_{SC}$-by-$N_{TX}$ integer matrix. $N_{SC}$ is the number of DM-RS indices per antenna, and $N_{TX}$ is the number of transmission antennas. The resource element (RE) indices for the demodulation reference signal (DM-RS) are associated with PUSCH transmission. By default, the indices are returned in one-based linear indexing form that can directly index elements of a resource matrix. These indices are ordered according to the PUSCH DM-RS modulation symbol mapping specified in TS 36.211 [1], Section 5.5.2. The `opts` input offers alternative indexing formats.

Data Types: `uint32`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePUSCH` | `ltePUSCHDecode` | `ltePUSCHPrecode` | `ltePUSCHDeprecode` | `ltePUSCHIndices` | `ltePUSCHDRS`

# ltePUSCHIndices

PUSCH resource element indices

## Syntax

```
[ind,info] = ltePUSCHIndices(ue,chs)
[ind,info] = ltePUSCHIndices(ue,chs,opts)
```

## Description

`[ind,info] = ltePUSCHIndices(ue,chs)` returns a column vector of resource element indices given the UE-specific settings structure, `ue`, and channel transmission configuration, `chs`. It returns a column vector of Physical Uplink Shared Channel (PUSCH) resource element (RE) indices and a structure, `info`, containing information related to the PUSCH indices. By default, the indices are returned in 1-based linear indexing form that can directly index elements of a resource matrix. These indices are ordered as the PUSCH modulation symbols should be mapped. Alternative indexing formats can also be generated.

Support of PUSCH frequency hopping is provided by the function `lteDCIResourceAllocation`, which creates `PRBSet` from a DCI Format 0 message.

`[ind,info] = ltePUSCHIndices(ue,chs,opts)` formats the returned indices using options specified by `opts`.

## Examples

### Generate PUSCH RE Indices

Generate 0-based PUSCH resource element (RE) indices in linear form.

```
frc = lteRMCUL('A1-1');
puschIndices = ltePUSCHIndices(frc,frc.PUSCH,{'0based','ind'});
puschIndices(1:4)
```

```
ans = 4x1 uint32 column vector

   0
   1
   2
   3
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific settings, specified as a structure having the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NULRB** | Required | Scalar integer from 6 to 110 | Number of uplink resource blocks. ($N_{RB}^{UL}$) |
| **CyclicPrefixUL** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length. |
| **NTxAnts** | Optional | 1 (default), 2, 4 | Number of transmission antennas. |
| **Shortened** | Optional | 0 (default), 1 | Option to shorten the subframe by omitting the last symbol, specified as 0 or 1. If `1`, the last symbol of the subframe is not used. For subframes with possible SRS transmission, set `Shortened` to 1 to maintain a standard compliant configuration. |

Data Types: `struct`

### `chs` — Channel transmission configuration

structure

Channel transmission configuration, specified as a structure. It contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **PRBSet** | Required | Integer column vector or two-column matrix | PRB indices, specified as a column vector or a 2-column matrix, containing the Physical Resource Block indices (PRBs) corresponding to the resource allocations for this PUSCH. |
| **Modulation** | Optional | `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'` | Modulation format, specified as a character vector or string scalar for one codeword, or a cell array of character vectors or string array for two codewords. |
| **NLayers** | Optional | 1 (default), 2, 3, 4 | Number of transmission layers. |

Data Types: `struct`

### `opts` — Output format options for resource element indices

character vector | cell array of character vectors | string array

Output format options for resource element indices, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string) :

| Category | Options | Description |
|---|---|---|
| Indexing style | `'ind'` (default) | The returned indices are in linear index style. |

| Category | Options | Description |
|---|---|---|
| | `'sub'` | The returned indices are in `[subcarrier,symbol,port]` subscript row style. |
| Index base | `'1based'` (default) | The returned indices are one-based. |
| | `'0based'` | The returned indices are zero-based. |

Example: `'ind 1based'`, `"ind 1based"`, `{'ind','1based'}`, or `["ind","1based"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `ind` — PUSCH resource element indices

column vector of integers

PUSCH resource element (RE) indices, returned as column vector of integers.

Data Types: `uint32`

### `info` — Information related to PUSCH indices

structure

Information related to the PUSCH indices, returned as a structure having the following fields.

| Parameter Field | Values | Description |
|---|---|---|
| **`G`** | 1- or 2-element vector of integers | A one- or two-element vector, specifying the number of coded and rate matched UL-SCH data bits for each codeword |
| **`Gd`** | Integer | Number of coded and rate matched UL-SCH data symbols, equal to the number of rows in the PUSCH indices |

Data Types: `struct`

# Version History

**Introduced in R2014a**

## See Also

`ltePUSCH` | `ltePUSCHDecode` | `ltePUSCHPrecode` | `ltePUSCHDeprecode` | `ltePUSCHDRS` | `ltePUSCHDRSIndices` | `lteDCIResourceAllocation`

# ltePUSCHPrecode

PUSCH MIMO precoding of transmission layers

## Syntax

```
out = ltePUSCHPrecode(in,p,codebook)
out = ltePUSCHPrecode(ue,chs,in)
```

## Description

`out = ltePUSCHPrecode(in,p,codebook)` precodes the matrix of layers, `in`, onto `p` antennas. When `p` is 2 or 4, precoding for spatial multiplexing is applied with the scalar codebook index, `codebook`. It performs precoding according to TS 36.211, Section 5.3.3A [1]. This function returns an *M*-by-*P* matrix. Where *M* is the number of symbols per antenna and *P* is the number of transmission antennas. The precoder transposes the operation defined in TS 36.211, Section 5.3.3A, specifically the symbols for layers and antennas lie in columns rather than rows.

`out = ltePUSCHPrecode(ue,chs,in)` precodes the matrix of layers, `in`, according to UE-specific settings, `ue`, and channel transmission configuration, `chs`.

## Examples

### Perform PUSCH MIMO Precoding

Generate a PUSCH precoding matrix with codebook index 1 for 3 layers and 4 antennas. By precoding an identity matrix, we can gain access to the precoding matrices themselves.

```
nLayers = 3;
nAntennas = 4;
codeBookIdx = 1;
out = ltePUSCHPrecode(eye(nLayers),nAntennas,codeBookIdx)
```

```
out = 3×4 complex

   0.5000 + 0.0000i  -0.5000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.5000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.5000 + 0.0000i
```

## Input Arguments

### `in` — Transmission input layers

numeric matrix

Transmission input layers, specified as a numeric matrix of size *N*-by-*NU*. `in` consists of the *N* modulation symbols for transmission upon *NU* layers. The `lteLayerMap` function generates such a matrix.

Example: [1 0 0; 0 1 0; 0 0 1]

Data Types: `double`
Complex Number Support: Yes

**`p` — Number of transmission antennas**
1 | 2 | 4

Number of transmission antennas, specified as an integer having the values 1, 2, or 4.

Example: 1

Data Types: `double`

**`codebook` — Codebook index**
scalar integer

`codebook` is a scalar integer specifying the codebook index to be used during precoding. This input is ignored when `p` is 1. The codebook matrix corresponding to a particular index can be found in TS 36.211, Section 5.3.3A [1].

Data Types: `double`

**`ue` — UE-specific settings**
structure

UE-specific settings, specified as a structure having the following fields.

**`NTxAnts` — Number of transmission antennas**
1 (default) | optional | 2 | 4

Number of transmission antennas, specified as 1, 2, or 4. Optional.

Data Types: `double`

Data Types: `struct`

**`chs` — Channel transmission configuration**
structure

Channel transmission configuration, specified a structure. `chs` can contain the following field. The `PMI` parameter field is only required if `ue.NTxAnts` is set to 2 or 4.

**`PMI` — Precoder matrix indication**
0 (default) | optional | numeric scalar (0...23)

Precoder matrix indication, specified as a numeric scalar between 0 (default) and 23. Only required if `ue.NTxAnts` is set to 2 or 4. Acceptable values for `PMI` depend upon `ue.NTxAnts` and the number of layers, *NU*. The scalar PMI is used during precoding.

Data Types: `double`

Data Types: `struct`

## Output Arguments

**`out` — Precoded output symbols**
*M*-by-*P* numeric matrix

Precoded output symbols, returned as a numeric matrix of size *M*-by-*P*. Where *M* is the number of symbols per antenna and *P* is the number of transmission antennas.

Data Types: `double`
Complex Number Support: Yes

# Version History

**Introduced in R2013b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePUSCHDeprecode` | `ltePUSCH` | `ltePUSCHDecode` | `ltePUSCHIndices` | `ltePUSCHDRS` | `ltePUSCHDRSIndices` | `lteULPrecode` | `lteULPMIInfo`

# lteRIDecode

Rank indication channel decoding

## Syntax

```
out = lteRIDecode(chs,in)
```

## Description

`out = lteRIDecode(chs,in)` performs the block decoding on soft input data, `in`. The input is assumed to be encoded using the procedure defined for RI in TS 36.212 [1], Section 5.2.2.6 for given channel transmission configuration, `chs`. The function returns the decoded output, `out`, as a vector of length `ORI`, the number of uncoded RI bits transmitted.

The block decoding will be performed separately on each soft input data using a maximum likelihood (ML) approach, assuming that `in` has been demodulated and equalized to best restore the originally transmitted values.

The RI decoder performs different type of block decoding depending upon the number of uncoded RI bits to be recovered. For `ORI` less than 3 bits, the decoder assumed the bits are encoded using the procedure defined in TS 36.212 [1], Section 5.2.2.6. For decoding 3 to 11 RI bits, the decoder assumes the bits are block encoded using the procedure defined in TS 36.212 [1], Section 5.2.2.6.4. For decoding greater than 11 bits, the decoder performs the inverse procedure described in TS 36.212 [1], Section 5.2.2.6.5.

## Examples

### Decode RI Bits for 64QAM

Decode coded rank indication (RI) soft input bits for a 64QAM channel transmission configuration.

Generate rank indication bits and initialize the channel transmission configuration structure. Encode logical RI bits and turn logical bits into 'LLR' data. Decode the RI bits.

```
ri = [1;0;1];
chs.Modulation = '64QAM';
chs.QdRI = 1;
chs.ORI = length(ri);
chs.NLayers = 1;

codedRI = lteRIEncode(chs,ri);
codedRI(codedRI == 0) = -1
```

```
codedRI = 6x1 int8 column vector

    1
   -1
    1
   -1
   -1
```

```
    1

decRI = lteRIDecode(chs,codedRI)

decRI = 3x1 logical array

   1
   0
   1
```

## Input Arguments

### `chs` — Channel transmission configuration

structure

Channel transmission configuration, specified as a structure. Multiple codewords can be parameterized by two different forms of the `chs` structure. Each codeword can be defined by separate elements of a 1-by-2 structure array, or the codeword parameters can be combined together in the fields of a single scalar, or 1-by-1, structure. Any scalar field values apply to both codewords and a scalar `NLayers` is the total number. See “UL-SCH Parameterization” for further details.

### `Modulation` — Modulation format

`'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'` | cell array of character vectors | string array

Modulation format, specified as `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`. Use double quotes for string. It there are two blocks, use a cell array of character vectors or a string array. Each element of the arrays is associated with a transport block.

Data Types: `char` | `string` | `cell`

### `ORI` — Number of uncoded RI bits

0 (default) | optional | nonnegative integer

Number of uncoded RI bits, specified as a nonnegative integer. The RI decoder performs different type of block decoding depending upon the number of uncoded RI bits to be recovered.

For `ORI` less than 3 bits, the decoder assumed the bits are encoded using the procedure defined in TS 36.212 [1], Section 5.2.2.6.

For decoding 3 to 11 RI bits, the decoder assumes the bits are block encoded using the procedure defined in TS 36.212 [1], Section 5.2.2.6.4. For decoding greater than 11 bits, the decoder performs the inverse procedure described in TS 36.212 [1], Section 5.2.2.6.5.

Data Types: `double`

### `NLayers` — Number of transmission layers

1 (default) | optional | 2 | 3 | 4

Number of transmission layers, specified as 1, 2, 3, or 4.

Data Types: `double`

Data Types: `struct`

**`in` — RI input bits**
numeric vector | cell array of numeric vectors

RI input bits, specified as a numeric vector or a cell array of numeric vectors. The block decoding will be performed separately on each soft input data using a maximum likelihood (ML) approach assuming that `in` has been demodulated and equalized to best restore the originally transmitted values.

Data Types: `double` | `cell`

## Output Arguments

**`out` — Decoded output**
logical column vector

Decoded output, returned as a logical column vector. The vector length is determined by the value of `ORI`.

Data Types: `logical`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteRIEncode` | `lteACKDecode` | `lteCQIDecode` | `lteULSCHDeinterleave` | `lteULSCHDecode`

# lteRIEncode

Rank indication channel encoding

## Syntax

```
out = lteRIEncode(chs,in)
```

## Description

`out = lteRIEncode(chs,in)` returns the coded rank indication (RI) bits after performing block coding, as defined for RI in TS 36.212 [1], Section 5.2.2.6. `in` should be a vector or cell array containing up to 15 RI bits. `out` contains the encoded bits in the same form.

Multiple codewords can be parameterized by two different forms of the `chs` structure. Each codeword can be defined by separate elements of a 1-by-2 structure array, or the codeword parameters can be combined together in the fields of a single scalar, or 1-by-1, structure. Any scalar field values apply to both codewords and a scalar `NLayers` is the total number. See “UL-SCH Parameterization” for further details.

Since the RI bits are carried on all defined codewords, a single input will result in a cell array of encoded outputs if multiple codewords are parameterized. This allows for easy integration with the other functions.

The RI coder performs different types of block coding depending upon the number of RI bits in vector `in`. If `in` consists of one element, it uses TS 36.212 [1], Table 5.2.2.6-3. If `in` consists of two elements, it uses TS 36.212 [1], Table 5.2.2.6-4 for encoding. The placeholder bits, *x* and *y* in the tables, are represented by –1 and –2, respectively.

Similarly, for 3 to 11 bits, the RI encoding is performed as per TS 36.212 [1], Section 5.2.2.6.4. For greater than 11 bits, the encoding is performed as described in TS 36.212 [1], Section 5.2.2.6.5.

## Examples

### Encode RI Bits for One Codeword

Generate the coded rank indication (RI) bits for a single codeword.

```
riBit = 0;
chs.Modulation = '64QAM';
chs.QdRI = 1;
chs.NLayers = 1;

codedRi = lteRIEncode(chs,riBit)
```

```
codedRi = 6x1 int8 column vector

    0
   -2
   -1
   -1
```

```
   -1
   -1
```

### Encode RI Bits for Two Codewords

Generate the coded rank indication (RI) bits for a two codewords on 3 layers.

```
riBit = 0;
chs.Modulation = {'64QAM' '64QAM'};
chs.QdRI = 1;
chs.NLayers = 3;

codedRi = lteRIEncode(chs,riBit)
```

```
codedRi=1×2 cell array
    {6x1 int8}    {12x1 int8}
```

```
codedRi{:}
```

```
ans = 6x1 int8 column vector

    0
   -2
   -1
   -1
   -1
   -1


ans = 12x1 int8 column vector

    0
   -2
   -1
   -1
   -1
   -1
    0
   -2
   -1
   -1
      ⋮
```

## Input Arguments

### chs — PUSCH-specific parameter structure

scalar structure | structure array

PUSCH-specific parameter structure, specified as a scalar structure or a structure array. `chs` contains the following fields.

**`QdRI` — Number of coded RI symbols**
nonnegative numeric scalar | nonnegative numeric vector

Number of coded RI symbols, specified as a nonnegative numeric scalar or vector ($Q'_{RI}$).

Data Types: `double`

**`Modulation` — Modulation format**
`'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'` | cell array of character vectors | string array

Modulation format, specified as `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`. Use double quotes for string. It there are two blocks, use a cell array of character vectors or a string array. Each element of the arrays is associated with a transport block.

Data Types: `char` | `string` | `cell`

**`NLayers` — Number of transmission layers**
1 (default) | optional | 2 | 3 | 4

Number of transmission layers, specified as a positive numeric scalar. Optional.

Data Types: `double`

Data Types: `struct`

**`in` — RI input bits**
logical vector of length 1 to 15 | cell array of logical vectors

RI input bits, specified as a logical vector of length 1 to 15 or a cell array of logical vectors. Each vector can contain up to 15 RI bits apiece.

Data Types: `cell` | `double`

## Output Arguments

**`out` — Encoded output bits**
integer column vector | cell array of integer column vectors

Encoded output bits, returned as an integer column vector or a cell array of integer column vectors, in the same form as `in`. If the PUSCH-specific parameter structure `chs` defines multiple codewords, `out` is a cell array.

Data Types: `int8` | `cell`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteRIDecode` | `lteACKEncode` | `lteCQIEncode` | `lteULSCHInterleave` | `lteULSCH`

# lteRMCDL

Downlink reference measurement channel configuration

## Syntax

```
rmccfgout = lteRMCDL(rc)
rmccfgout = lteRMCDL(rc,duplexmode)
rmccfgout = lteRMCDL(rc,duplexmode,totsubframes)
rmccfgout = lteRMCDL(rmccfg,ncodewords)
```

## Description

`rmccfgout = lteRMCDL(rc)` returns configuration structure `rmccfgout` for reference channel `rc`. This structure uses a channel-specific default configuration. The structure contains the configuration parameters required to generate a given reference channel waveform using the reference measurement channel (RMC) generator tool, `lteRMCDLTool`. The field names and default values comply with the definition found in TS 36.101 [1], Annex A.3.

`rmccfgout = lteRMCDL(rc,duplexmode)` specifies `duplexmode`, the duplexing mode.

`rmccfgout = lteRMCDL(rc,duplexmode,totsubframes)` specifies `totsubframes`, total number of subframes to generate.

`rmccfgout = lteRMCDL(rmccfg,ncodewords)` returns a fully configured structure for the reference channel partially, or wholly, defined by input structure `rmccfg`. You can specify the number of PDSCH codewords to modulate in the `ncodewords` input.

## Examples

### Generate RMC Configuration Where Allocated Resource Blocks Vary Per SF

Create a configuration structure for reference measurement channel R.44 as specified in TS 36.101.

```
rc = 'R.44';

rmcOut = lteRMCDL(rc);
```

For this RMC, the size of the resource allocation varies per subframe. Evidence of this is seen by viewing the `PRBSet` and observing that the length of resource allocation vectors in the `PRBSet` cell array vary per subframe.

```
rmcOut.PDSCH.PRBSet
```

```
ans=1×10 cell array
    {41x1 double}    {50x1 double}    {50x1 double}    {50x1 double}    {50x1 double}    {0x0 dou
```

### Generate RMC Configuration Where CFI Varies Per Subframe

Create a configuration structure for reference measurement channel R.0 in TDD mode as specified in TS 36.101. For this RMC and duplex mode combination, the value of CFI varies per subframe.

Set input arguments.

```
rc = 'R.0';
duplexmode = 'TDD';
```

Generate the configuration structure.

```
rmcOut = lteRMCDL(rc,duplexmode)
```

```
rmcOut = struct with fields:
                 RC: 'R.0'
              NDLRB: 15
            CellRefP: 1
             NCellID: 0
        CyclicPrefix: 'Normal'
                 CFI: [3 2 3 3 3 3 2 3 3 3]
         PCFICHPower: 0
                  Ng: 'Sixth'
       PHICHDuration: 'Normal'
               HISet: [112x3 double]
          PHICHPower: 0
              NFrame: 0
           NSubframe: 0
        TotSubframes: 10
           Windowing: 0
          DuplexMode: 'TDD'
                PDSCH: [1x1 struct]
    OCNGPDCCHEnable: 'Off'
     OCNGPDCCHPower: 0
    OCNGPDSCHEnable: 'Off'
     OCNGPDSCHPower: 0
          OCNGPDSCH: [1x1 struct]
               Nfft: []
                SSC: 4
          TDDConfig: 1
```

In TDD mode, looking at the `rmcOut.CFI` vector, we see variation which corresponds to per subframe CFI value adjustment.

```
rmcOut.CFI
```

```
ans = 1×10

     3     2     3     3     3     3     2     3     3     3
```

### Generate Downlink R.11 RMC Configuration

Create a configuration structure for reference measurement channel R.11 as specified in TS 36.101. View the contents of the configuration structure.

```
rmc.RC = 'R.11';
rmc.NCellID = 100;
rmc.PDSCH.TxScheme = 'SpatialMux';
rmcOut = lteRMCDL(rmc,2)
```

```
rmcOut = struct with fields:
                RC: 'R.11'
             NDLRB: 50
           CellRefP: 2
            NCellID: 100
       CyclicPrefix: 'Normal'
               CFI: 2
        PCFICHPower: 0
                Ng: 'Sixth'
      PHICHDuration: 'Normal'
              HISet: [112x3 double]
         PHICHPower: 0
             NFrame: 0
          NSubframe: 0
       TotSubframes: 10
          Windowing: 0
         DuplexMode: 'FDD'
              PDSCH: [1x1 struct]
    OCNGPDCCHEnable: 'Off'
     OCNGPDCCHPower: 0
    OCNGPDSCHEnable: 'Off'
     OCNGPDSCHPower: 0
          OCNGPDSCH: [1x1 struct]
               Nfft: []
```

Display the contents of the PDSCH substructure.

```
rmcOut.PDSCH
```

```
ans = struct with fields:
           TxScheme: 'SpatialMux'
         Modulation: {'16QAM'  '16QAM'}
            NLayers: 2
                Rho: 0
               RNTI: 1
              RVSeq: [2x4 double]
                 RV: [0 0]
     NHARQProcesses: 8
       NTurboDecIts: 5
             PRBSet: [50x1 double]
     TargetCodeRate: 0.5000
     ActualCodeRate: [2x10 double]
         TrBlkSizes: [2x10 double]
    CodedTrBlkSizes: [2x10 double]
          DCIFormat: 'Format2'
        PDCCHFormat: 2
         PDCCHPower: 0
            CSIMode: 'PUSCH 3-1'
            PMIMode: 'Wideband'
             PMISet: 0
```

Display the contents of the OCNGPDSCH substructure.

```
rmcOut.OCNGPDSCH

ans = struct with fields:
          RNTI: 0
    Modulation: 'QPSK'
      TxScheme: 'TxDiversity'
```

### Override Default Downlink R.13 RMC Configuration

Create a new customized parameter set by overriding selected values of an existing preset RMC. To define a single codeword full-band 10MHz PDSCH using 4 CRS port spatial multiplexing and 64QAM modulation, begin by initializing an RMC configuration structure to R.13. Looking at TS 36.101, Table A.3.1.1-1, see the RMC R.13 matches desired configuration except the default QPSK modulation must be adjusted.

Create an R.13 RMC configured structure and display `rmc.PDSCH`.

```
rmcOverride.RC = 'R.13';
rmc = lteRMCDL(rmcOverride,1);
rmc.PDSCH

ans = struct with fields:
           TxScheme: 'SpatialMux'
         Modulation: {'QPSK'}
            NLayers: 1
                Rho: 0
               RNTI: 1
              RVSeq: [0 1 2 3]
                 RV: 0
     NHARQProcesses: 8
       NTurboDecIts: 5
             PRBSet: [50x1 double]
     TargetCodeRate: 0.3333
     ActualCodeRate: [0.3032 0.3450 0.3450 0.3450 0.3450 0 0.3450 0.3450 0.3450 0.3450]
         TrBlkSizes: [3624 4392 4392 4392 4392 0 4392 4392 4392 4392]
    CodedTrBlkSizes: [12032 12800 12800 12800 12800 0 12800 12800 12800 12800]
          DCIFormat: 'Format2'
        PDCCHFormat: 2
         PDCCHPower: 0
            CSIMode: 'PUSCH 1-2'
            PMIMode: 'Wideband'
             PMISet: 0
```

Override the default modulation and execute the `lteRMCDL` function. Inspect `rmc.PDSCH`, PDSCH transport block sizes and physical channel capacities are updated to maintain the R=1/3 coding rate when the modulation is overridden.

```
rmcOverride.PDSCH.Modulation = '64QAM';
rmc = lteRMCDL(rmcOverride,1);
rmc.PDSCH

ans = struct with fields:
           TxScheme: 'SpatialMux'
```

```
      Modulation: {'64QAM'}
         NLayers: 1
             Rho: 0
            RNTI: 1
           RVSeq: [0 0 1 2]
              RV: 0
  NHARQProcesses: 8
    NTurboDecIts: 5
          PRBSet: [50x1 double]
  TargetCodeRate: 0.3333
  ActualCodeRate: [0.4255 0.4000 0.4000 0.4000 0.4000 0 0.4000 0.4000 0.4000 0.4000]
      TrBlkSizes: [15264 15264 15264 15264 15264 0 15264 15264 15264 15264]
 CodedTrBlkSizes: [36096 38400 38400 38400 38400 0 38400 38400 38400 38400]
       DCIFormat: 'Format2'
     PDCCHFormat: 2
      PDCCHPower: 0
         CSIMode: 'PUSCH 1-2'
         PMIMode: 'Wideband'
          PMISet: 0
```

Note the RV sequence is also updated to reflect appropriate values for 64QAM modulation.

## Input Arguments

### `rc` — Reference channel

character vector | string scalar

Reference channel, specified as a character vector or string scalar. The function configures the RMC in accordance with the reference channels defined in Annex A.3 of TS 36.101. This table lists the supported values of this input and their associated configuration parameters.

| Reference Channel (`rc`) | Configuration | | | | |
|---|---|---|---|---|---|
| | **Transmission Scheme (PDSCH.TxScheme)** | **Number of Resource Blocks** | **Modulation** | **Number of CRS Antenna Ports** | **Coding Rate** |
| `'R.0'` | `'Port0'` | 1 | 16-QAM | 1 | 1/2 |
| `'R.1'` | `'Port0'` | 1 | 16-QAM | 1 | 1/2 |
| `'R.2'` | `'Port0'` | 50 | QPSK | 1 | 1/3 |
| `'R.3'` | `'Port0'` | 50 | 16-QAM | 1 | 1/2 |
| `'R.4'` | `'Port0'` | 6 | QPSK | 1 | 1/3 |
| `'R.5'` | `'Port0'` | 15 | 64-QAM | 1 | 3/4 |
| `'R.6'` | `'Port0'` | 25 | 64-QAM | 1 | 3/4 |
| `'R.7'` | `'Port0'` | 50 | 64-QAM | 1 | 3/4 |
| `'R.8'` | `'Port0'` | 75 | 64-QAM | 1 | 3/4 |
| `'R.9'` | `'Port0'` | 100 | 64-QAM | 1 | 3/4 |

| Reference Channel (rc) | Configuration | | | | |
|---|---|---|---|---|---|
| | Transmission Scheme (PDSCH.TxScheme) | Number of Resource Blocks | Modulation | Number of CRS Antenna Ports | Coding Rate |
| 'R.10' | 'TxDiversity', 'SpatialMux' | 50 | QPSK | 2 | 1/3 |
| 'R.11' | 'TxDiversity''SpatialMux', 'CDD' | 50 | 16-QAM | 2 | 1/2 |
| 'R.12' | 'TxDiversity' | 6 | QPSK | 4 | 1/3 |
| 'R.13' | 'SpatialMux' | 50 | QPSK | 4 | 1/3 |
| 'R.14' | 'SpatialMux', 'CDD' | 50 | 16-QAM | 4 | 1/2 |
| 'R.25' | 'Port5' | 50 | QPSK | 1 | 1/3 |
| 'R.26' | 'Port5' | 50 | 16-QAM | 1 | 1/2 |
| 'R.27' | 'Port5' | 50 | 64-QAM | 1 | 3/4 |
| 'R.28' | 'Port5' | 1 | 16-QAM | 1 | 1/2 |
| 'R.31-3A' (with FDD) | 'CDD' | 50 | 64-QAM | 2 | 0.85-0.90 |
| 'R.31-3A (with TDD) | 'CDD' | 68 | 64-QAM | 2 | 0.87-0.90 |
| 'R.31-4' | 'CDD' | 100 | 64-QAM | 2 | 0.87-0.90 |
| 'R.43' (with FDD) | 'Port7-14' | 50 | QPSK | 2 | 1/3 |
| 'R.43' (with TDD) | 'SpatialMux' | 100 | 16-QAM | 4 | 1/2 |
| 'R.44' (with FDD) | 'Port7-14' | 50 | QPSK | 2 | 1/3 |
| 'R.44' (with TDD) | 'Port7-14' | 50 | 64-QAM | 2 | 1/2 |
| 'R.45' | 'Port7-14' | 50 | 16-QAM | 2 | 1/2 |
| 'R.45-1' | 'Port7-14' | 39 | 16-QAM | 2 | 1/2 |
| 'R.48' | 'Port7-14' | 50 | QPSK | 2 | 1/2 |
| 'R.50' (with FDD) | 'Port7-14' | 50 | 64-QAM | 2 | 1/2 |
| 'R.50' (with TDD) | 'Port7-14' | 50 | QPSK | 2 | 1/3 |
| 'R.51' | 'Port7-14' | 50 | 16 -QAM | 2 | 1/2 |

| Reference Channel (rc) | Configuration | | | | |
|---|---|---|---|---|---|
| | Transmission Scheme (PDSCH.TxScheme) | Number of Resource Blocks | Modulation | Number of CRS Antenna Ports | Coding Rate |
| 'R.68-1' (with FDD) | 'CDD' | 75 | 256-QAM | 2 | 0.74-0.88 |
| 'R.68-1' (with TDD) | 'CDD' | 75 | 256-QAM | 2 | 0.76-0.88 |
| 'R.105' (with FDD) | 'CDD' | 100 | 1024-QAM | 2 | 0.76-0.79 |
| 'R.105' (with TDD) | 'CDD' | 100 | 1024-QAM | 2 | 0.76-0.78 |
| Custom RMCs configured for non-standard bandwidths but with the same code rate as the standard versions. | | | | | |
| 'R.6-27RB' | 'Port0' | 27 | 64-QAM | 1 | 3/4 |
| 'R.12-9RB' | 'TxDiversity' | 9 | QPSK | 4 | 1/3 |
| 'R.11-45RB' | 'CDD' | 45 | 16-QAM | 2 | 1/2 |

Data Types: `char` | `string`

**`duplexmode` — Duplexing mode**
`'FDD'` (default) | `'TDD'`

Duplexing mode frame structure type, specified as `'FDD'` or `'TDD'`.

When you specify the `rc` input as `'R.25'`, `'R.26'`, `'R.27'`, or `'R.28'`, the default duplexing mode is `'TDD'`.

Data Types: `char` | `string`

**`totsubframes` — Total number of subframes**
10 (default) | positive integer

Total number of subframes, specified as a positive integer. this input defines the number of subframes that form the resource grid, used by `lteRMCDLTool`, to generate the waveform.

Data Types: `double`

**`rmccfg` — Reference channel configuration**
structure

Reference channel configuration, specified as a structure. This input defines the `rmccfgout` output. If you do not specify a field, the function returns the corresponding field of the `rmccfgout` output as the default value. This input contains one field, `RC`.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **RC** | Optional | `'R.0'` (default), `'R.1'`, `'R.2'`, `'R.3'`, `'R.4'`, `'R.5'`, `'R.6'`, `'R.7'`, `'R.8'`, `'R.9'`, `'R.10'`, `'R.11'`, `'R.12'`, `'R.13'`, `'R.14'`, `'R.25'`, `'R.26'`, `'R.27'`, `'R.28'`, `'R.31-3A'`, `'R.31-4'`, `'R.43'`, `'R.44'`, `'R.45'`, `'R.45-1'`, `'R.48'`, `'R.50'`, `'R.51'`, `'R.68-1'`, `'R.105'`, `'R.6-27RB'`, `'R.12-9RB'`, `'R.11-45RB'` | Reference measurement channel (RMC) number or type, as specified in Annex A.3 of TS 36.101.<br>• To facilitate the transmission of system information blocks (SIBs), user data is usually not scheduled on subframe 5. To schedule user data in subframe 5, use one of these sustained-data-rate RMCs: `'R.31-3A'`, `'R.31-4'`, `'R.68-1'`, or `'R.105'`.<br>• `'R.6-27RB'`, `'R.12-9RB'`, and `'R.11-45RB'` are custom RMCs configured for non-standard bandwidths that maintain the same code rate as the standardized versions defined in Annes A.3 of TS 36.101. |

Data Types: `struct`

**`ncodewords` — Number of PDSCH codewords to modulate**
1 | 2

Number of PDSCH codewords to modulate, specified as 1 or 2. The default used is the value defined in TS 36.101, [1] for the RMC configuration given by `RC`.

Data Types: `double`

## Output Arguments

**`rmccfgout` — RMC configuration**
structure

RMC configuration, returned as a structure. This output contains RMC-specific configuration parameters in these fields.

| Parameter Field | Values | Description |
|---|---|---|
| **RC** | `'R.0'`, `'R.1'`, `'R.2'`, `'R.3'`, `'R.4'`, `'R.5'`, `'R.6'`, `'R.7'`, `'R.8'`, `'R.9'`, `'R.10'`, `'R.11'`, `'R.12'`, `'R.13'`, `'R.14'`, `'R.25'`, `'R.26'`, `'R.27'`, `'R.28'`, `'R.31-3A'`, `'R.31-4'`, `'R.43'`, `'R.44'`, `'R.45'`, `'R.45-1'`, `'R.48'`, `'R.50'`, `'R.51'`, `'R.68-1'`, `'R.105'`, `'R.6-27RB'`, `'R.12-9RB'`, `'R.11-45RB'` | Reference measurement channel (RMC) number or type, as specified in Annex A.3 of TS 36.101.<br>• To facilitate the transmission of system information blocks (SIBs), user data is usually not scheduled on subframe 5. To schedule user data in subframe 5, use one of these sustained-data-rate RMCs: `'R.31-3A'`, `'R.31-4'`, `'R.68-1'`, or `'R.105'`.<br>• `'R.6-27RB'`, `'R.12-9RB'`, and `'R.11-45RB'` are custom RMCs configured for non-standard bandwidths that maintain the same code rate as the standardized versions defined in Annes A.3 of TS 36.101. |
| **NDLRB** | Integer in the interval [6, 110] | Number of downlink resource blocks |
| **CellRefP** | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **NCellID** | Integer in the interval [0, 503] | Physical layer cell identity |
| **CyclicPrefix** | `'Normal'`, `'Extended'` | Cyclic prefix length |
| **CFI** | 1, 2, 3, real-valued vector of length 10 | Control format indicator (CFI) value. When the CFI value does not vary between subframes, specify this field as a scalar. Otherwise, specify this field as a vector, where the *k*th element corresponds to the CFI value of the *k*th subframe.<br>The CFI value varies between subframes for these RMCs when you specify the `duplexmode` input as `'TDD'` mode, the `CFI` varies per subframe for these RMCs: `'R.0'`, `'R.5'`, `'R.6'`, `'R.6-27RB'`, `'R.12-9RB'`. |
| **PCFICHPower** | Real-valued scalar | PCFICH symbol power adjustment, in dB |
| **Ng** | `'Sixth'`, `'Half'`, `'One'`, `'Two'` | HICH group multiplier |
| **PHICHDuration** | `'Normal'`, `'Extended'` | PHICH duration |
| **HISet** | 112-by-3 matrix | Maximum PHICH groups (112), as specified in section 6.9 of TS 36.211, with the first PHICH sequence of each group set to ACK). For more information, see `ltePHICH`. |
| **PHICHPower** | Real-valued scalar | PHICH symbol power, in dB |
| **NFrame** | Nonnegative integer | Nonnegative integer |
| **NSubframe** | Nonnegative integer | Subframe number |
| **TotSubframes** | Nonnegative integer | Total number of subframes to generate |

| Parameter Field | Values | Description |
|---|---|---|
| **Windowing** | Nonnegative integer | Number of time-domain samples over which the function applies windowing and overlapping of OFDM symbols |
| **Nfft** | Positive integer | Number of IFFT points used in the OFDM modulation. |
| **DuplexMode** | `'FDD'`, `'TDD'` | Duplexing mode, returned as one of these values<br>• `'FDD'` — Frequency division duplex<br>• `'TDD'` — Time division duplex |
| **CSIRSPeriod** | `'On'`, `'Off'`, integer in the interval [0, 154], two-element row vector of nonnegative integers, cell array | CSI-RS subframe configurations for CSI-RS resources, returned as one of these values.<br>• `'On'` or `'Off'`<br>• An integer in the interval [0, 154] corresponding to the parameter $I_{\text{CSI-RS}}$, specified in Table 6.10.5.3-1 of TS 36.211<br>• A vector of the form $[T_{\text{CSI-RS}}\ \Delta_{\text{CSI-RS}}]$, in accordance with Table 6.10.5.3-1 of TS 36.211<br>• A cell array of configurations for each resource.<br>This field applies only when the `TxScheme` field is `'Port7-14'`. |
| The following fields are present and applicable only for the `'Port7-14'` transmission scheme (`TxScheme`) and only required in `rmccfg` if `CSIRSPeriod` is not set to `'Off'`. | | |
| **CSIRSConfig** | Nonnegative integer | Array CSI-RS configuration indices. See Table 6.10.5.2-1 of TS 36.211. |
| **CSIRefP** | 1, 2, 4, 8 | Array of number of CSI-RS antenna ports |
| These fields are present and applicable only for the `'Port7-14'` transmission scheme (`TxScheme`) | | |
| **ZeroPowerCSIRSPeriod** | `'Off'` (default), `'On'`, `Icsi-rs` (0,...,154), `[Tcsi-rs Dcsi-rs]`. You can also specify values in a cell array of configurations for each resource. | Zero power CSI-RS subframe configurations for one or more zero power CSI-RS resource configuration index lists. Multiple zero power CSI-RS resource lists can be configured from a single common subframe configuration or from a cell array of configurations for each resource list. |
| The following field is applicable only for `'Port7-14'` transmission scheme (`TxScheme`) and required only in `rmccfg` if `CSIRSPeriod` is not set to `'Off'`. | | |
| **ZeroPowerCSIRSConfig** | 16-bit bitmap character vector or string scalar (truncated if not 16 bits or `'0'` MSB extended), or a numeric list of CSI-RS configuration indices. You can also specify values in a cell array of configurations for each resource. | Zero power CSI-RS resource configuration index lists (TS 36.211 Section 6.10.5.2). Specify each list as a 16-bit bitmap character vector or string scalar (if less than 16 bits, then `'0'` MSB extended), or as a numeric list of CSI-RS configuration indices from TS 36.211 Table 6.10.5.2-1 in the `'4'` CSI reference signal column. Multiple lists can be defined using a cell array of individual lists. |

| Parameter Field | Values | Description |
|---|---|---|
| **PDSCH** | Scalar structure | PDSCH transmission configuration substructure |
| **OCNGPDCCHEnable** | `'Off'`, `'On'` | Enable PDCCH OCNG<br><br>See footnote. |
| **OCNGPDCCHPower** | Scalar integer, `0` (default) | PDCCH OCNG power in dB |
| **OCNGPDSCHEnable** | `'Off'`, `'On'` | Enable PDSCH OCNG |
| **OCNGPDSCHPower** | Scalar integer, defaults to `PDSCH.Rho` (default) | PDSCH OCNG power in dB |
| **OCNGPDSCH** | Scalar structure | PDSCH OCNG configuration substructure |
| **OCNG** | `'Off'`, `'On'`. `'Disable'` and `'Enable'` are also accepted. | OFDMA channel noise generator<br><br>**Note** This parameter will be removed in a future release. Use the PDCCH and PDSCH-specific OCNG parameters instead. |
| These fields are present and applicable only for `'TDD'` duplex mode (`DuplexMode`). | | |
| **SSC** | 4 (default), integer in the interval [0, 9] | Special subframe configuration (SSC) |
| **TDDConfig** | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink-downlink configuration.<br><br>See footnote. |

| Parameter Field | Values | Description |
|---|---|---|
| **1** | CFI is equal to the number of symbols allocated to:<br>• (PDCCH - 1) for `NDLRB < 10`<br>• PDCCH for `NDLRB ≥ 10`<br>For the RMCs, the number of symbols allocated to PDCCH varies with channel bandwidth setting,<br>• Two symbols for 20 MHz, 15 MHz, and 10 MHz<br>• Three symbols for 5 MHz and 3 MHz<br>• Four symbols for 1.4 MHz<br>• In the TDD mode, only two OFDM symbols are allocated to PDCCH in special subframes irrespective of the channel bandwidth. Therefore, the CFI value varies per subframe for the 5 MHz, 3 MHz, and 1.4 MHz channel bandwidths. Specifically, for bandwidths where PDCCH symbol allocation is not two in other subframes. | |
| **2** | The PDCCH OCNG fills the unused PDCCH resource elements with QPSK symbols using either single port or transmit diversity depending on the number of cell RS ports. | |
| **3** | All supported RMCs use TDDConfig 1 by default. When you specify a value different then the default, the full parameter set is configured according to the following rules.<br>• Preserve subframe 0 (downlink) for all TDDConfig — The values of the parameters in subframe 0 of TDDConfig 1 is applied in all other TDDConfig.<br>• Preserve special subframe behaviour — The values of the parameters in special subframes of TDDConfig 1 is applied in all other TDDConfig.<br>• Preserve subframe 5 (downlink) for all TDDConfig — The values of the parameters in subframe 5 of TDDConfig 1 is applied to all other TDDConfig. For all RMCs currently supported, subframe 5 is treated separately from other subframes. According to TS 36.101 Section A.3.1, “Unless otherwise stated, no user data is scheduled on subframes 5 in order to facilitate the transmission of system information blocks (SIB).” Hence the RC value, if present, determines the behaviour of subframe 5. This means that subframe 5 is not transmitted for other RMCs, with the exception of sustained data rate RMCs R.31-3A and R.31-4.<br>• All other downlink subframes use the same settings as subframe 9. | |

**PDSCH Substructure**

The substructure PDSCH relates to the physical channel configuration and contains these fields:

<table>
<tr><th>Parameter Field</th><th>Values</th><th>Description</th></tr>
<tr><td>TxScheme</td><td>'Port0', 'TxDiversity', 'CDD', 'SpatialMux', 'MultiUser', 'Port5', 'Port7-8', 'Port8', 'Port7-14'.</td><td>PDSCH transmission scheme, specified as one of the following options.
<table>
<tr><th>Transmission scheme</th><th>Description</th></tr>
<tr><td>'Port0'</td><td>Single antenna port, port 0</td></tr>
<tr><td>'TxDiversity'</td><td>Transmit diversity</td></tr>
<tr><td>'CDD'</td><td>Large delay cyclic delay diversity scheme</td></tr>
<tr><td>'SpatialMux'</td><td>Closed loop spatial multiplexing</td></tr>
<tr><td>'MultiUser'</td><td>Multi-user MIMO</td></tr>
<tr><td>'Port5'</td><td>Single-antenna port, port 5</td></tr>
<tr><td>'Port7-8'</td><td>Single-antenna port, port 7, when NLayers = 1. Dual layer transmission, ports 7 and 8, when NLayers = 2.</td></tr>
<tr><td>'Port8'</td><td>Single-antenna port, port 8</td></tr>
<tr><td>'Port7-14'</td><td>Up to eight layer transmission, ports 7–14</td></tr>
</table></td></tr>
<tr><td>Modulation</td><td>'QPSK', '16QAM', '64QAM', '256QAM', '1024QAM'</td><td>Modulation type, specified as a character vector, cell array of character vectors, or string array. If blocks, each cell is associated with a transport block.</td></tr>
<tr><td>NLayers</td><td>Integer from 1 to 8</td><td>Number of transmission layers.</td></tr>
<tr><td>NTxAnts</td><td>Nonnegative scalar integer</td><td>Number of transmission antenna ports. This argument is present only for UE-specific demodulation reference symbols.<br><br>Note NTxAnts is provided by lteRMCDL for information only.</td></tr>
<tr><td>Rho</td><td>0 (default), numeric scalar</td><td>PDSCH resource element power allocation, in dB</td></tr>
<tr><td>RNTI</td><td>0 (default), scalar integer</td><td>Radio network temporary identifier (RNTI) value (16 bits)</td></tr>
</table>

| Parameter Field | Values | Description |
|---|---|---|
| **RVSeq** | Integer vector (0,1,2,3), specified as a one or two row matrix (for one or two codewords) | Redundancy version (RV) indicator used by all HARQ processes, returned as a numeric matrix. `RVSeq` is a one- or two-row matrix for one or two codewords, respectively. The number of columns in `RVSeq` equals the number of transmissions of the transport blocks associated with a HARQ process. The RV sequence specified in each column is applied to the transmission of the transport blocks. If `RVSeq` is a scalar (or column vector in the case of two codewords), then there is a single initial transmission of each block with no retransmissions. If `RVSeq` is a row vector in a two-codeword transmission, then the same RV sequence is applied to both codewords.<br><br>See footnote. |
| **RV** | Integer vector (0,1,2,3). A one or two column matrix (for one or two codewords). | Specifies the redundancy version for one or two codewords used in the initial subframe number, `NSubframe`. This parameter field is only for informational purposes and is read-only. |
| **NHARQProcesses** | 1, 2, 3, 4, 5, 6, 7, or 8 | Number of HARQ processes per component carrier |
| **NTurboDecits** | 5 (default), nonnegative scalar integer | Number of turbo decoder iteration cycles |
| **PRBSet** | Integer-valued column vector, two-column matrix, cell array | Zero-based physical resource block (PRB) indices corresponding to the slot-wise resource allocations for this PDSCH. The function returns this field as one of these values.<br><br>• An integer-valued column vector. The resource allocation is the same in both slots of the subframe.<br>• A two-column matrix or two-element cell array, which specifies different PRBs for each slot in a subframe.<br>• A cell array of length 10 (corresponding to a frame, if the allocated physical resource blocks vary across subframes).<br><br>This field varies per subframe for these RMCs: `'R.25'` (with TDD), `'R.26'` (with TDD), `'R.27'` (with TDD), `'R.43'` (with FDD), `'R.44'`, `'R.45'`, `'R.48'`, `'R.50'`, `'R.51'`, `'R.68-1'`, and `'R.105'`. |

| Parameter Field | Values | Description |
|---|---|---|
| **TargetCodeRate** | Scalar or one or two row numeric matrix | Target code rates for one or two codewords for each subframe in a frame. Used for calculating the transport block sizes according to TS 36.101 [1], Annex A.3.1.<br><br>If both `TargetCodeRate` and `TrBlkSizes` are not provided at the input, and the RC does not have a single ratio target code rate in TS 36.101, Table A.3.1.1-1, `TargetCodeRate == ActualCodeRate`. |
| **ActualCodeRate** | One or two row numeric matrix | Actual code rates for one or two codewords for each subframe in a frame, calculated according to TS 36.101 [1], Annex A.3.1. The maximum actual code rate is 0.93. This parameter field is only for informational purposes and is read-only. |
| **TrBlkSizes** | One or two row numeric matrix | Transport block sizes for each subframe in a frame<br><br>See footnote. |
| **CodedTrBlkSizes** | One or two row numeric matrix | Coded transport block sizes for one or two codewords. This parameter field is for informational purposes and is read-only.<br><br>See footnote. |
| **DCIFormat** | `'Format0'`, `'Format1'`, `'Format1A'`, `'Format1B'`, `'Format1C'`, `'Format1D'`, `'Format2'`, `'Format2A'`, `'Format2B'`, `'Format2C'`, `'Format2D'`, `'Format3'`, `'Format3A'`, `'Format4'`, `'Format5'`, `'Format5A'` | Downlink control information (DCI) format type of the PDCCH associated with the PDSCH. See `lteDCI`. |
| **PDCCHFormat** | 0, 1, 2, 3 | Aggregation level of PDCCH associated with PDSCH |
| **PDCCHPower** | Numeric scalar | PDCCH power in dB |
| **CSIMode** | `'PUCCH 1-0'`, `'PUCCH 1-1'`, `'PUSCH 1-2'`, `'PUSCH 3-0'`, `'PUSCH 3-1'` | CSI reporting mode |
| **PMIMode** | `'Wideband'` (default), `'Subband'` | PMI reporting mode. `PMIMode='Wideband'` corresponds to PUSCH reporting Mode 1-2 or PUCCH reporting Mode 1-1 (PUCCH Report Type 2) and `PMIMode='Subband'` corresponds to PUSCH reporting Mode 3-1. |
| The following field is present only for `TxScheme = 'SpatialMux'`. | | |

| Parameter Field | Values | Description |
|---|---|---|
| **PMISet** | Integer vector with element values from 0 to 15. | Precoder matrix indication (PMI) set. It can contain either a single value, corresponding to single PMI mode, or multiple values, corresponding to multiple or subband PMI mode. The number of values depends on CellRefP, transmission layers and TxScheme. For more information about setting PMI parameters, see `ltePMIInfo`. |
| The following field is present only for `TxScheme` = `'Port7-8'`, `'Port8'`, or `'Port7-14'`. | | |
| **NSCID** | 0 (default), 1 | Scrambling identity (*ID*) |
| The following field is present only for UE-specific beamforming (`'Port5'`, `'Port7-8'`, `'Port8'`, or `'Port7-14'`). | | |
| **W** | Numeric matrix | `NLayers`-by-*P* precoding matrix, chosen according to TS 36.101 Annex B.4. *P* is the number of transmit antennas. The resulting precoding matrix with index zero is selected from:<br>• The set defined in TS 36.211, Section 6.3.4 for `'Port5', 'Port7-8', and 'Port8'` transmission schemes<br>• or from the set associated with CSI reporting as defined in TS 36.213, Section 7.2.4 for the `'Port7-14'` transmission scheme.<br>`W` is present only for wideband UE-specific beamforming (`'Port5', 'Port7-8', 'Port8', 'Port7-14'`). |

**1** The function returns valid `TrBlkSizes` and `CodedTrBlkSizes` set to 0 when `PRBSet` is empty, indicating there is no PDSCH allocation in this frame.

**2** Any parameters missing at the input are initialized based on the `RC` field if present or `'R.0'` otherwise.

- When the `RC` field is specified, the RMC specified defines the subframe scheduling.
- If the `RC` field is absent or set to empty, all downlink subframes and special subframes (if TDD mode) are assumed to be scheduled.
- `TrBlkSizes` and `CodedTrBlkSizes` are set according to the target code rate, the modulation scheme, and the allocated resources.
- The value of `RVSeq` is set according to the modulation scheme.

#### OCNGPDSCH Substructure

The substructure, `OCNGPDSCH`, defines the OCNG patterns in associated RMCs and tests according to TS 36.101, Section A.5. `OCNGPDSCH` contains these fields, which can also be customized with the full range of PDSCH-specific values.

| Parameter Field | Values | Description |
|---|---|---|
| **`Modulation`** | OCNG `Modulation` has same setting options as `rmccfgout.PDSCH.Modulation` | See `rmccfgout.PDSCH.Modulation` |
| **`TxScheme`** | OCNG `TxScheme` has same setting options as `rmccfgout.PDSCH.TxScheme` | See `rmccfgout.PDSCH.TxScheme` |
| **`RNTI`** | 0 (default), scalar integer | OCNG radio network temporary identifier (RNTI) value (16 bits) |

# Version History

**Introduced in R2014a**

**R2022b: IFFT size output**

The output structure `rmccfgout` includes the `Nfft` field, which contains the number of IFFT points used in the OFDM modulation.

## References

[1] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[3] 3GPP TS 36.213. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[4] 3GPP TS 36.321. “Evolved Universal Terrestrial Radio Access (E-UTRA); Medium Access Control (MAC) protocol Specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteRMCDLTool` | `lteRMCUL` | `lteTestModel`

# lteRISelect

PDSCH rank indication calculation

## Syntax

```
[ri,pmiset] = lteRISelect(enb,chs,hest,noiseest)
```

## Description

`[ri,pmiset] = lteRISelect(enb,chs,hest,noiseest)` calculates PDSCH rank indication (RI), given cell-wide settings,`enb`, channel configuration settings, `chs`, channel estimate resource array `hest`, and receiver noise variance `noiseest`. For more information, see “RI Selection” on page 2-955.

## Examples

### Rank Indication example

This example shows how to populate an empty resource grid for RMC R.13 with cell-specific reference signal symbols. The signal is passed through a channel and OFDM demodulated. Estimates of the channel and noise power spectral density are used for RI and PMI calculation. A CodebookSubset bitmap of all ones means that no codebook subset restriction is applied, allowing any PMI/RI combination applicable for the configured transmission scheme to be selected during RI selection.

Create empty resource grid and populate with cell specific reference symbols. Set `enb.PDSCH.CodebookSubset` to all ones so the PMI selection is unconstrained

```
enb = lteRMCDL('R.13');
enb.PDSCH.CodebookSubset = '111111111111111';
reGrid = lteResourceGrid(enb);
reGrid(lteCellRSIndices(enb)) = lteCellRS(enb);
[txWaveform,txInfo] = lteOFDMModulate(enb,reGrid);
```

Initialize the channel configuration structure (`chcfg`), filter the signal through a channel and demodulate the signal.

```
chcfg.DelayProfile = 'EPA';
chcfg.NRxAnts = 4;
chcfg.DopplerFreq = 5;
chcfg.MIMOCorrelation = 'Low';
chcfg.SamplingRate = txInfo.SamplingRate;
chcfg.Seed = 1;
chcfg.InitPhase = 'Random';
chcfg.ModelType = 'GMEDS';
chcfg.NTerms = 16;
chcfg.NormalizeTxAnts = 'On';
chcfg.NormalizePathGains = 'On';
chcfg.InitTime = 0;
```

```
rxWaveform = lteFadingChannel(chcfg,txWaveform);
rxSubframe = lteOFDMDemodulate(enb,rxWaveform);
```

Estimate corresponding channel, including noise spectral density and reference signal subcarriers. Use `lteRISelect` to calculate RI & PMI

```
cec.FreqWindow = 1;
cec.TimeWindow = 15;
cec.InterpType = 'cubic';
cec.PilotAverage = 'UserDefined';
cec.InterpWinSize = 1;
cec.InterpWindow = 'Centered';
[hest,noiseEst] = lteDLChannelEstimate(enb,cec,rxSubframe);
[ri,pmi] = lteRISelect(enb,enb.PDSCH,hest,noiseEst)
```

```
ri = 3
```

```
pmi = 13
```

## Input Arguments

### enb — eNodeB cell-wide settings

structure

eNodeB cell-wide settings, specified as a structure containing the following parameter fields:

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NDLRB** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **NCellID** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **CellRefP** | Required | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **CyclicPrefix** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **DuplexMode** | Optional | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| The following parameters apply when `DuplexMode` is set to, `TDD`. | | | |
| **TDDConfig** | Optional | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration |
| **SSC** | Optional | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |
| The following parameters apply when `DuplexMode` is set to `'TDD'` or `chs.TxScheme` is set to `'Port7-14'` | | | |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NSubframe** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| The following parameters apply when `chs.TxScheme` is set to `'Port7-14'`. | | | |
| **CSIRefP** | Required | 1 (default), 2, 4, 8 | Array of number of CSI-RS antenna ports |
| **CSIRSConfig** | Required | Scalar integer | Array CSI-RS configuration indices. See TS 36.211, Table 6.10.5.2-1. |
| **CSIRSPeriod** | Optional | `'On'` (default), `'Off'`, `Icsi-rs` (0,...,154), `[Tcsi-rs Dcsi-rs]`. You can also specify values in a cell array of configurations for each resource. | CSI-RS subframe configurations for one or more CSI-RS resources. Multiple CSI-RS resources can be configured from a single common subframe configuration or from a cell array of configurations for each resource. |
| **NFrame** | Optional | 0 (default), nonnegative scalar integer | Frame number |

**`chs` — Channel-specific transmission configuration**
structure | structure array

Channel specific transmission configuration, specified as scalar structure, or structure array containing the following parameter fields:

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **PMIMode** | Optional | `'Wideband'` (default), `'Subband'` | PMI reporting mode. `PMIMode='Wideband'` corresponds to PUSCH reporting Mode 1-2 or PUCCH reporting Mode 1-1 (PUCCH Report Type 2) and `PMIMode='Subband'` corresponds to PUSCH reporting Mode 3-1. |

<table>
<tr><th>Parameter Field</th><th>Required or Optional</th><th>Values</th><th>Description</th></tr>
<tr><td>TxScheme</td><td>Optional</td><td>'Port0', 'TxDiversity', 'CDD', 'SpatialMux' (default), 'MultiUser', 'Port7-8', 'P… 'Port7-14'.</td><td>PDSCH transmission scheme, specified as one of the following options.
<table>
<tr><th>Transmission scheme</th><th>Description</th></tr>
<tr><td>'Port0'</td><td>Single antenna port, port 0</td></tr>
<tr><td>'TxDiversity'</td><td>Transmit diversity</td></tr>
<tr><td>'CDD'</td><td>Large delay cyclic delay diversity scheme</td></tr>
<tr><td>'SpatialMux'</td><td>Closed loop spatial multiplexing</td></tr>
<tr><td>'MultiUser'</td><td>Multi-user MIMO</td></tr>
<tr><td>'Port5'</td><td>Single-antenna port, port 5</td></tr>
<tr><td>'Port7-8'</td><td>Single-antenna port, port 7, when NLayers = 1. Dual layer transmission, ports 7 and 8, when NLayers = 2.</td></tr>
<tr><td>'Port8'</td><td>Single-antenna port, port 8</td></tr>
<tr><td>'Port7-14'</td><td>Up to eight layer transmission, ports 7–14</td></tr>
</table></td></tr>
<tr><td>CodebookSubset</td><td>Optional</td><td>Character vector, string scalar, or integer vector, all ones (default)</td><td>Codebook subset restriction, specified as a character vector or string scalar bitmap. The default values are all ones, permitting all PMI values. This parameter is configured by higher layers and indicates the values of PMI that can be reported. The bitmap, defined in TS 36.213, Section 7.2, is arranged a_A-1,a_A-2,...a_0. For example, the element CodebookSubset(1) corresponds to a_A-1 and the element CodebookSubset(end) corresponds to a_0. The length of the bitmap is given by the info.CodebookSubsetSize field returned by ltePMIInfo. You can also specify the bitmap in a hexadecimal form by adding the prefix '0x'. Alternatively, you can specify a numeric array identical to the pmiset output, indicating to restrict the selection to only those pmiset values. Specifying the parameter in this way enables you to obtain SINR estimates against an existing reported PMI for RI and CQI selection. If this parameter field is defined but is empty, no codebook subset restriction is applied. (codebookSubsetRestriction)</td></tr>
</table>

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| The following parameter applies for `'Port7-14'` transmission scheme with `CSIRefP` equal to 4, or for `'Port7-8'` or `'Port8'` transmission scheme with `CellRefP` equal to 4. | | | |
| **`AltCodebook4Tx`** | Optional | `'Off'` (default), `'On'` | If set to `'On'`, enables the alternative codebook for CSI reporting with four antennas defined in TS 36.213, Tables 7.2.4-0A to 7.2.4-0D. The default is `'Off'`. (*alternativeCodeBookEnabledFor4TX-r12*) |

**`hest` — Channel estimate**
multidimensional array

Channel estimate, specified as a *K*-by-*L*-by-*NRxAnts*-by-*P* array where:

- *K* is the number of subcarriers.
- *L* is the number of OFDM symbols.
- *NRxAnts* is the number of receive antennas.
- *P* is the number of transmit antennas.

Data Types: `double`

**`noiseest` — Receiver noise variance**
numeric scalar

Receiver noise variance, specified as numeric scalar. It is an estimate of the received noise power spectral density.

Data Types: `double`

## Output Arguments

**`ri` — Rank indication**
scalar

Rank indication, returned as a scalar, indicates the optimal number of layers to use for transmission to maximize SINR.

**`pmiset` — Precoder matrix indications**
scalar | column vector

Precoder matrix indications, returned as a scalar, or a column vector.

- For wideband reporting (`NSubbands=1`), `pmiset` is a scalar specifying the selected wideband codebook index,*i2*.
- For the `'Port7-14'` transmission scheme with eight CSI-RS ports, or for CSI reporting with the alternative codebook for four antennas, `pmiset` has `NSubbands+1` rows. The first row indicates wideband codebook index, *i1*, and the subsequent `NSubbands` rows indicate the subband codebook indices, *i2*.

- For other numbers of CSI-RS ports in the `'Port7-14'` transmission scheme, and for other transmission schemes, `pmiset` has `NSubbands` rows, each row returns the subband codebook index for that subband.

The number of subbands, `NSubbands`, is a field in the `info` structure output by `ltePMIInfo` and `ltePMISelect`.

## More About

### RI Selection

The PDSCH rank indication (RI) selection process determines the optimal number of layers (*NLayers*) to use for transmission to maximize SINR. The range of *NLayers* to consider is calculated based on the transmission scheme and the configured reference signal ports.

1. For $\nu = 1,...,NLayers$,
   a. Use `ltePMISelect`, with `chs.NLayers` = $\nu$, to perform PMI selection.
   b. Record the selected PMI and total SINR across all layers, excluding layers with SINR below the threshold of 0 dB.
2. Select the number of transmission layers, $\nu$, that maximizes the SINR of the transmission and return as the rank indication, `ri` and corresponding PMI set, `pmiset`.

RI selection corresponds to:

- Report Type 3 (for reporting Mode 1-0 or Mode 1-1) on the PUCCH.
- Reporting Mode 1-2 or Mode 3-1 on the PUSCH.

For more information on RI selection, see TS 36.213 Section 7.2.

### PMI Selection

PDSCH precoder matrix indication (PMI) selection calculates a PMI set, `pmiset`. Functions, such as `lteRMCDLTool` or `ltePDSCH`, can use the returned `pmiset` to configure the PMI for downlink transmissions they generate. PMI selection is performed using the PMI definitions specified in TS 36.213, Section 7.2.4.

- The CSI reporting codebook is used for:
  - `'Port7-14'` transmission scheme with eight CSI-RS ports
  - CSI reporting with the alternative codebook for four antennas (*alternativeCodeBookEnabledFor4TX -r12* = true).
- The codebook for closed-loop spatial multiplexing, defined in TS 36.211 Tables 6.3.4.2.3-1 and 6.3.4.2.3-2, is used for other cases.

The PMI feedback type associated with the PMI selection process can be wideband or subband:

- `PMIMode = 'Wideband'` corresponds to PUSCH reporting Mode 1-2 or PUCCH reporting Mode 1-1 (PUCCH Report Type 2).
- `PMIMode = 'Subband'` corresponds to PUSCH reporting Mode 3-1.

PMI selection is based on the rank indicated by `chs.NLayers`, except for `'TxDiversity'` transmission scheme, where the rank is 1. In PUCCH reporting Mode 1-1, you can achieve codebook

subsampling for submode 2, as specified in TS 36.213, Table 7.2.2-1D, with an appropriate `chs.CodebookSubset`.

# Version History

**Introduced in R2014b**

## References

[1] 3GPP TS 36.213. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`ltePMISelect` | `lteCQISelect` | `ltePMIInfo`

# lteRMCDLTool

Generate downlink RMC waveform

## Syntax

```
lteRMCDLTool
[waveform,grid,rmccfgout] = lteRMCDLTool(rmccfg,trdata)
[waveform,grid,rmccfgout] = lteRMCDLTool(rc,trdata,duplexmode,totsubframes)
```

## Description

`lteRMCDLTool` starts the **LTE Waveform Generator** app configured for parameterization and generation of a reference measurement channel (RMC) waveform. The `Reference Channel` menu lists the available RMCs with their default top-level settings.

`[waveform,grid,rmccfgout] = lteRMCDLTool(rmccfg,trdata)` where `rmccfg` specifies a user-defined reference channel structure. The reference configuration structure with default parameters can easily be created using `lteRMCDL` then modified if desired.

---

**Note** SIB1 messages and the associated PDSCH and PDCCH can be added to the output `waveform` by adding the substructure `rmccfg.SIB`.

---

`[waveform,grid,rmccfgout] = lteRMCDLTool(rc,trdata,duplexmode,totsubframes)` specifies the default reference measurement channel, `rc`, and information bits `trdata`. `duplexmode` and `totsubframes` are optional input arguments, that define the duplex mode of the generated waveform and total number of subframes that make up the `grid`.

## Examples

### Generate LTE DL RMC R.31-4

Generate a time domain signal and a 3-dimensional array of the resource elements for R.31-4 FDD as specified in TS 36.101 Annex A.3.9.1-1. R.31-4 FDD is 20MHz, 64QAM, variable code rate and has user data scheduled in subframe 5.

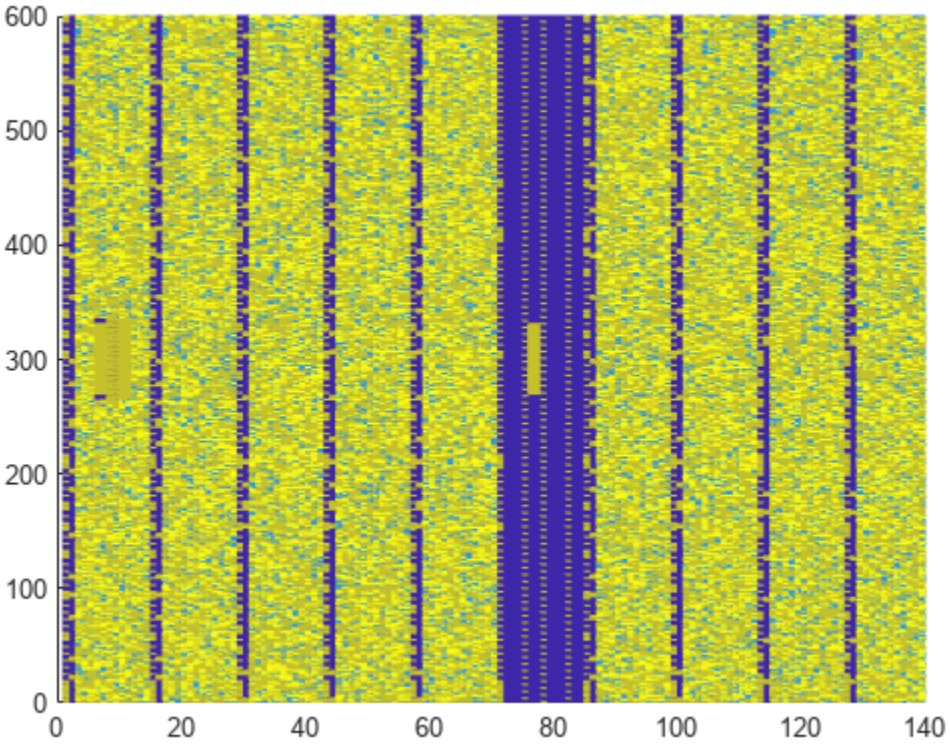
```
[txWaveform,txGrid,rmcCfgOut] = lteRMCDLTool('R.31-4',{[1;0] [1;0]});
```

### Generate RMC R.3 with SIB

This example shows use of `lteRMCDLTool` to generate a tx waveform with SIB transmission enabled using DCIFormat1A and localized allocation.

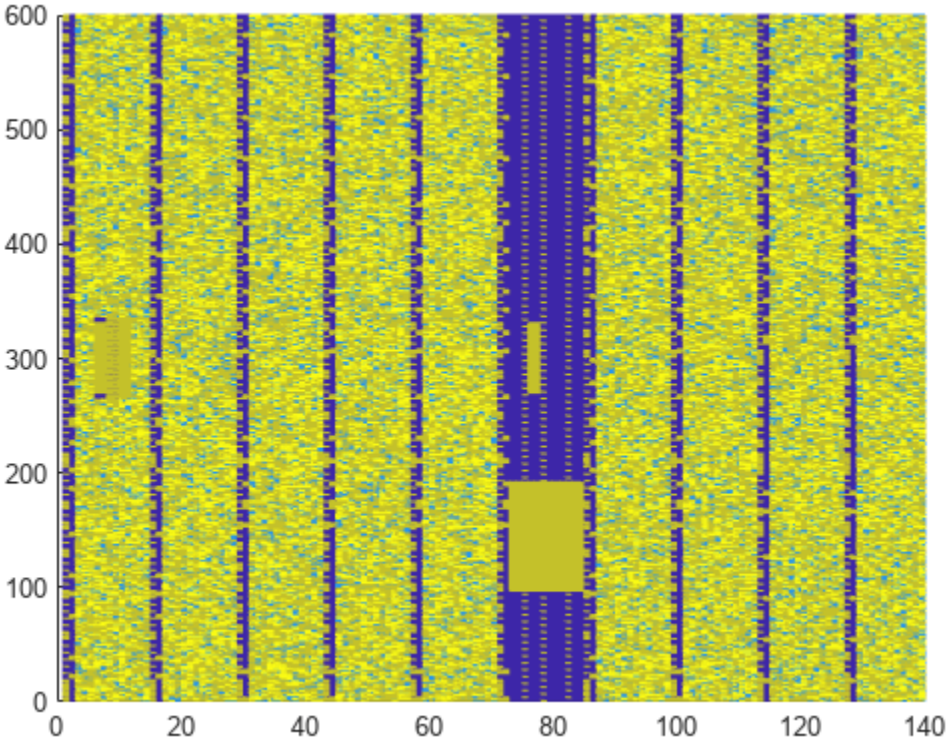
Specify desired RMC, initialize configuration structure and define `txData`. Generate `txGrid` and plot it.

```
rc = 'R.3';
rmc = lteRMCDL(rc);

txData = [1;0;0;1];
[~,txGrid,~] = lteRMCDLTool(rmc, txData);
mesh(abs(txGrid))
view(2)
```




To insert SIB1 message into the output waveform, initialize `SIB` substructure, enable SIB transmission, adjust other defaults, and regenerate txGrid. Plot `txGrid` to illustrate the presence of SIB1 message in subframe 5

```
rmc.SIB.Enable = 'On';
rmc.SIB.DCIFormat = 'Format1A';
rmc.SIB.AllocationType = 0;
rmc.SIB.VRBStart = 8;
rmc.SIB.VRBLength = 8;
rmc.SIB.Data = randi([0 1],144,1);

[txWaveform,txGrid,rmcCfgOut] = lteRMCDLTool(rmc, txData);
figure
mesh(abs(txGrid))
view(2)
```




### Generate LTE DL RMC R.12 With 16QAM Modulation

Generate a time domain waveform, and a 3D array of the resource elements for RMC R.12 as specified in TS 36.101. Modify the standard R.12 RMC to use 16QAM modulation scheme instead of the default QPSK.

Create an RMC setting structure specifying R.12 for `RC` and 16QAM for `Modulation`.

```
rmc.RC = 'R.12';
rmc.PDSCH.Modulation = '16QAM';
```

Generate the tx waveform, RE grid and also output the RMC configuration structure.

```
txData = [1;0;0;1];
[txWaveform, txGrid, rmcCfgOut] = lteRMCDLTool(rmc, txData);
```

Review the `rmcCgfOut` structure and `PDSCH` substructure.

```
rmcCfgOut
```

```
rmcCfgOut = struct with fields:
                RC: 'R.12'
             NDLRB: 6
           CellRefP: 4
            NCellID: 0
```

```
       CyclicPrefix: 'Normal'
                CFI: 3
        PCFICHPower: 0
                 Ng: 'Sixth'
      PHICHDuration: 'Normal'
              HISet: [112x3 double]
         PHICHPower: 0
             NFrame: 0
          NSubframe: 0
       TotSubframes: 10
          Windowing: 0
        DuplexMode: 'FDD'
              PDSCH: [1x1 struct]
    OCNGPDCCHEnable: 'Off'
     OCNGPDCCHPower: 0
    OCNGPDSCHEnable: 'Off'
     OCNGPDSCHPower: 0
          OCNGPDSCH: [1x1 struct]
               Nfft: 128
          SerialCat: 1
       SamplingRate: 1920000
```

```
rmcCfgOut.PDSCH
```

```
ans = struct with fields:
                TxScheme: 'TxDiversity'
              Modulation: {'16QAM'}
                 NLayers: 4
                     Rho: 0
                    RNTI: 1
                   RVSeq: [0 1 2 3]
                      RV: 0
          NHARQProcesses: 8
            NTurboDecIts: 5
                  PRBSet: [6x1 double]
          TargetCodeRate: 0.3333
          ActualCodeRate: [0 0.3846 0.3846 0.3846 0.3846 0 0.3846 0.3846 0.3846 0.3846]
              TrBlkSizes: [0 936 936 936 936 0 936 936 936 936]
         CodedTrBlkSizes: [0 2496 2496 2496 2496 0 2496 2496 2496 2496]
               DCIFormat: 'Format1'
             PDCCHFormat: 2
              PDCCHPower: 0
                 CSIMode: 'PUCCH 1-1'
                 PMIMode: 'Wideband'
    HARQProcessSequence: [0 1 2 3 4 0 5 6 7 8]
```

### Display Uplink PRB Allocation Type 1

Display the PRB allocations associated with the sequence of subframes in a frame for DCI Format 0 and uplink resource allocation type 1.

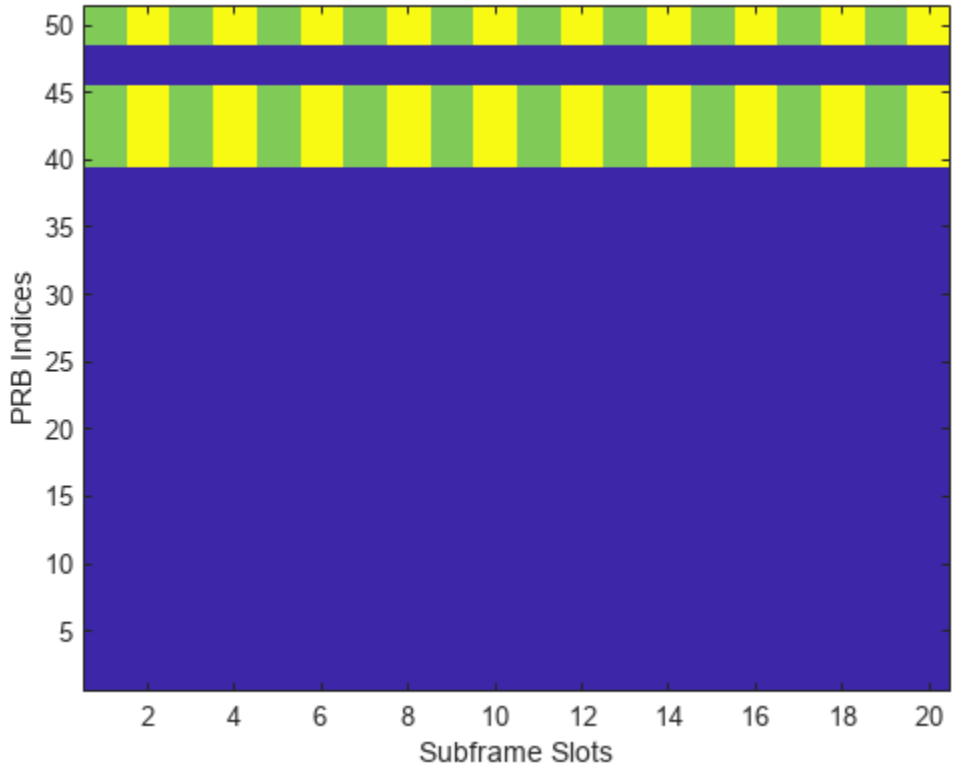
Configure a type 1 uplink resource allocation (multi-cluster). TS 36.213, Section 8.1.2 describes the resource indication value (RIV) determination.

```
enbue = struct('NDLRB',50);
dcistr = lteDCI(enbue,struct('DCIFormat','Format0','AllocationType',1));
dcistr.Allocation.RIV = 1;
```

Display an image of the PRBs used in each slot of each subframe in a frame.

- Create a `subframeslots` matrix full of zeros. There are 20 slots per frame, specifically two slots per subframe and ten subframes per frame.
- Loop through assigning a PRB set of indices for each subframe. Also assign a value in `subframeslots` for each occupied PRB index.

```
subframeslots = zeros(enbue.NDLRB,20);
for i = 0:9
    enbue.NSubframe = i;
    prbSet = lteDCIResourceAllocation(enbue,dcistr);
    prbSet = repmat(prbSet,1,2/size(prbSet,2));
    for s = 1:2
        subframeslots(prbSet(:,s)+1,2*i+s) = 20+s*20;
    end
end
imagesc(subframeslots);
axis xy;
xlabel('Subframe Slots');
ylabel('PRB Indices');
```




Observe from the image that the same set of PRB indices is used in each slot.

### Display Uplink Hopping PRB Allocation

Display the PRB allocations associated with the sequence of subframes in a frame for an uplink resource allocation with hopping.

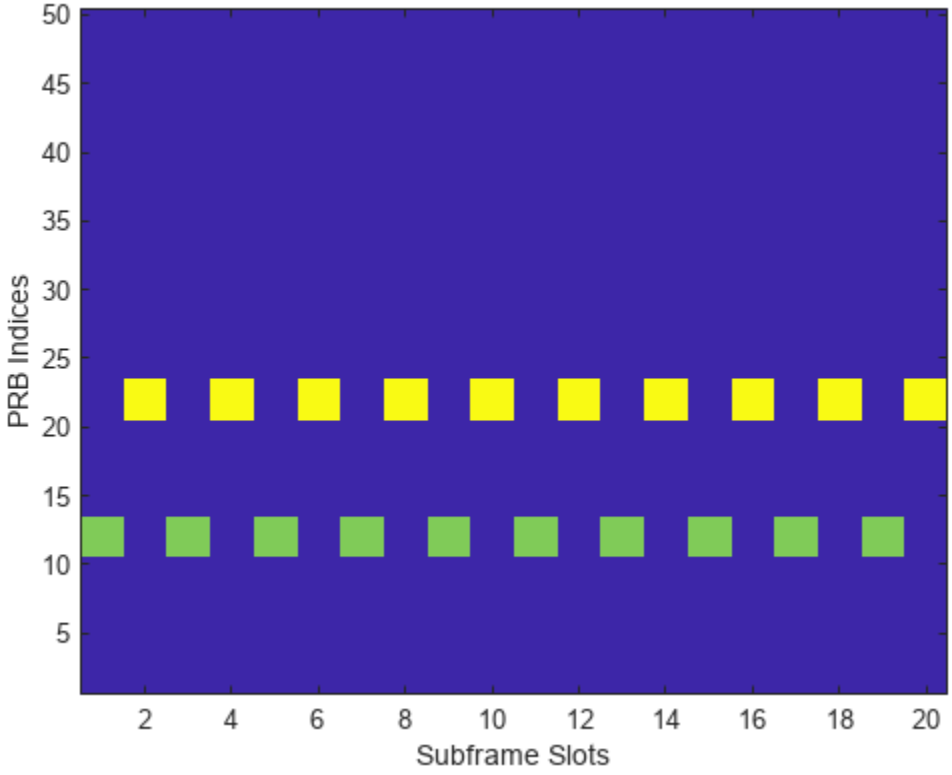
Configure a type 1 uplink resource allocation that has type 0 hopping and slot and subframe hopping.

```
enbue = struct('NDLRB',50,'NCellID',0);
dcistr = lteDCI(enbue,struct('DCIFormat','Format0','AllocationType',0,...
    'FreqHopping',1));
dcistr.Allocation.HoppingBits = 0;
dcistr.Allocation.RIV = 110;
enbue.PUSCHHopping = 'InterAndIntra';
enbue.MacTxNumber = 0;
enbue.NSubbands = 1;
enbue.PUSCHHoppingOffset = 10;
```

Display an image of the PRBs used in each slot of each subframe in a frame.

- Create a `subframeslots` matrix full of zeros. There are 20 slots per frame, specifically two slots per subframe and ten subframes per frame.
- Loop through assigning a PRB set of indices for each subframe. Also assign a value in `subframeslots` for each occupied PRB index.

```
subframeslots = zeros(enbue.NDLRB,20);
for i = 0:9
    enbue.NSubframe = i;
    prbSet = lteDCIResourceAllocation(enbue,dcistr);
    prbSet = repmat(prbSet,1,2/size(prbSet,2));
    for s = 1:2
        subframeslots(prbSet(:,s)+1,2*i+s) = 20+s*20;
    end
end
imagesc(subframeslots)
axis xy
xlabel('Subframe Slots')
ylabel('PRB Indices')
```




Observe from the image that the occupied PRB indices hops in odd and even slots.

## Input Arguments

### `rc` — Reference channel

character vector | string scalar

Reference channel, specified as a character vector or string scalar. The function configures the RMC in accordance with the reference channels defined in Annex A.3 of TS 36.101. This table lists the supported values of this input and their associated configuration parameters.

| Reference Channel (`rc`) | Configuration | | | | |
|---|---|---|---|---|---|
| | Transmission Scheme (PDSCH.TxScheme) | Number of Resource Blocks | Modulation | Number of CRS Antenna Ports | Coding Rate |
| `'R.0'` | `'Port0'` | 1 | 16-QAM | 1 | 1/2 |
| `'R.1'` | `'Port0'` | 1 | 16-QAM | 1 | 1/2 |
| `'R.2'` | `'Port0'` | 50 | QPSK | 1 | 1/3 |
| `'R.3'` | `'Port0'` | 50 | 16-QAM | 1 | 1/2 |
| `'R.4'` | `'Port0'` | 6 | QPSK | 1 | 1/3 |

| Reference Channel (rc) | Configuration | | | | |
|---|---|---|---|---|---|
| | Transmission Scheme (PDSCH.TxScheme) | Number of Resource Blocks | Modulation | Number of CRS Antenna Ports | Coding Rate |
| 'R.5' | 'Port0' | 15 | 64-QAM | 1 | 3/4 |
| 'R.6' | 'Port0' | 25 | 64-QAM | 1 | 3/4 |
| 'R.7' | 'Port0' | 50 | 64-QAM | 1 | 3/4 |
| 'R.8' | 'Port0' | 75 | 64-QAM | 1 | 3/4 |
| 'R.9' | 'Port0' | 100 | 64-QAM | 1 | 3/4 |
| 'R.10' | 'TxDiversity', 'SpatialMux' | 50 | QPSK | 2 | 1/3 |
| 'R.11' | 'TxDiversity''SpatialMux', 'CDD' | 50 | 16-QAM | 2 | 1/2 |
| 'R.12' | 'TxDiversity' | 6 | QPSK | 4 | 1/3 |
| 'R.13' | 'SpatialMux' | 50 | QPSK | 4 | 1/3 |
| 'R.14' | 'SpatialMux', 'CDD' | 50 | 16-QAM | 4 | 1/2 |
| 'R.25' | 'Port5' | 50 | QPSK | 1 | 1/3 |
| 'R.26' | 'Port5' | 50 | 16-QAM | 1 | 1/2 |
| 'R.27' | 'Port5' | 50 | 64-QAM | 1 | 3/4 |
| 'R.28' | 'Port5' | 1 | 16-QAM | 1 | 1/2 |
| 'R.31-3A' (with FDD) | 'CDD' | 50 | 64-QAM | 2 | 0.85-0.90 |
| 'R.31-3A (with TDD) | 'CDD' | 68 | 64-QAM | 2 | 0.87-0.90 |
| 'R.31-4' | 'CDD' | 100 | 64-QAM | 2 | 0.87-0.90 |
| 'R.43' (with FDD) | 'Port7-14' | 50 | QPSK | 2 | 1/3 |
| 'R.43' (with TDD) | 'SpatialMux' | 100 | 16-QAM | 4 | 1/2 |
| 'R.44' (with FDD) | 'Port7-14' | 50 | QPSK | 2 | 1/3 |
| 'R.44' (with TDD) | 'Port7-14' | 50 | 64-QAM | 2 | 1/2 |
| 'R.45' | 'Port7-14' | 50 | 16-QAM | 2 | 1/2 |
| 'R.45-1' | 'Port7-14' | 39 | 16-QAM | 2 | 1/2 |
| 'R.48' | 'Port7-14' | 50 | QPSK | 2 | 1/2 |

| Reference Channel (rc) | Configuration | | | | |
|---|---|---|---|---|---|
| | Transmission Scheme (PDSCH.TxScheme) | Number of Resource Blocks | Modulation | Number of CRS Antenna Ports | Coding Rate |
| 'R.50' (with FDD) | 'Port7-14' | 50 | 64-QAM | 2 | 1/2 |
| 'R.50' (with TDD) | 'Port7-14' | 50 | QPSK | 2 | 1/3 |
| 'R.51' | 'Port7-14' | 50 | 16 -QAM | 2 | 1/2 |
| 'R.68-1' (with FDD) | 'CDD' | 75 | 256-QAM | 2 | 0.74-0.88 |
| 'R.68-1' (with TDD) | 'CDD' | 75 | 256-QAM | 2 | 0.76-0.88 |
| 'R.105' (with FDD) | 'CDD' | 100 | 1024-QAM | 2 | 0.76-0.79 |
| 'R.105' (with TDD) | 'CDD' | 100 | 1024-QAM | 2 | 0.76-0.78 |
| Custom RMCs configured for non-standard bandwidths but with the same code rate as the standard versions. | | | | | |
| 'R.6-27RB' | 'Port0' | 27 | 64-QAM | 1 | 3/4 |
| 'R.12-9RB' | 'TxDiversity' | 9 | QPSK | 4 | 1/3 |
| 'R.11-45RB' | 'CDD' | 45 | 16-QAM | 2 | 1/2 |

Data Types: `char` | `string`

### `trdata` — Information bits

vector | cell array containing one or two vectors

Information bits, specified as a vector or cell array containing one or two vectors of bit values. Each vector contains the information bits stream to be coded across the duration of the generation, which represents multiple concatenated transport blocks. If the number of bits required across all subframes of the generation exceeds the length of the vectors provided, the `txdata` vector is looped internally. This feature allows you to enter a short pattern, such as `[1;0;0;1]`, which is repeated as the input to the transport coding. In each subframe of generation, the number of data bits taken from this stream comes from the elements of the `rmccfgout.PDSCH.TrBlkSizes` matrix.

When the `trdata` input contains empty vectors, there is no transport data. The transmission of PDSCH and its corresponding PDCCH are skipped in the `waveform` when the `trdata` contains empty vectors. The other physical channels and signals are transmitted as normal in generated `waveform`.

Example: `[1;0;0;1]`

Data Types: `double` | `cell`
Complex Number Support: Yes

### `duplexmode` — Duplexing mode

`'FDD'` (default) | optional | `'TDD'`

Duplexing mode, specified as `'FDD'` or `'TDD'` to indicate the frame structure type of the generated waveform.

Data Types: `char` | `string`

**`totsubframes` — Total number of subframes**
`10` (default) | positive integer

Total number of subframes, specified as a positive integer. This argument specifies the total number of subframes that form the resource grid.

Data Types: `double`

**`rmccfg` — Reference channel configuration**
structure

Reference channel configuration, specified as a structure. Create a reference configuration structure with default parameters by using the `lteRMCDL` function. The reference configuration structures you generate with the `lteRMCDL` function comply with those defined in Annex A.3 of [1].

To generate the `waveform` output in alignment with your simulation requirements, modify the output of the `lteRMCDL function`. To add SIB1 messages and the associated PDSCH and PDCCH to the output `waveform`, specify the `rmccfg.SIB` substructure. You can specify this input to include fields contained in the `rmccfgout` output structure.

Data Types: `struct`

## Output Arguments

**`waveform` — Generated RMC time-domain waveform**
numeric matrix

Generated RMC time-domain waveform, returned as a $N_S$-by-$N_T$ numeric matrix. $N_S$ is the number of time-domain samples and $N_T$ is the number of transmit antennas.

Data Types: `double`

**`grid` — Populated resource grid**
numeric 3-D array

Populated resource grid, returned as a numeric 3-D array of resource elements for several subframes across all configured antenna ports, as described in “Represent Resource Grids”.

`grid` represents the populated resource grid for all the physical channels specified in TS 36.101 [1], Annex A.3.

Data Types: `double`
Complex Number Support: Yes

**`rmccfgout` — RMC configuration**
structure

RMC configuration, returned as a structure. This output contains information about the OFDM-modulated waveform and RMC-specific configuration parameters. Field definitions and settings align with `rmccfg`.

For more information about the OFDM modulated waveform, see `lteOFDMInfo`. For more information about the RMC-specific configuration parameters, see `lteRMCDL`.

| Parameter Field | Values | Description |
|---|---|---|
| **`RC`** | `'R.0'`, `'R.1'`, `'R.2'`, `'R.3'`, `'R.4'`, `'R.5'`, `'R.6'`, `'R.7'`, `'R.8'`, `'R.9'`, `'R.10'`, `'R.11'`, `'R.12'`, `'R.13'`, `'R.14'`, `'R.25'`, `'R.26'`, `'R.27'`, `'R.28'`, `'R.31-3A'`, `'R.31-4'`, `'R.43'`, `'R.44'`, `'R.45'`, `'R.45-1'`, `'R.48'`, `'R.50'`, `'R.51'`, `'R.68-1'`, `'R.105'`, `'R.6-27RB'`, `'R.12-9RB'`, `'R.11-45RB'` | Reference measurement channel (RMC) number or type, as specified in Annex A.3 of TS 36.101.<br>• To facilitate the transmission of system information blocks (SIBs), user data is usually not scheduled on subframe 5. To schedule user data in subframe 5, use one of these sustained-data-rate RMCs: `'R.31-3A'`, `'R.31-4'`, `'R.68-1'`, or `'R.105'`.<br>• `'R.6-27RB'`, `'R.12-9RB'`, and `'R.11-45RB'` are custom RMCs configured for non-standard bandwidths that maintain the same code rate as the standardized versions defined in Annes A.3 of TS 36.101. |
| **`NDLRB`** | Integer in the interval [6, 110] | Number of downlink resource blocks |
| **`CellRefP`** | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **`NCellID`** | Integer in the interval [0, 503] | Physical layer cell identity |
| **`CyclicPrefix`** | `'Normal'`, `'Extended'` | Cyclic prefix length |
| **`CFI`** | 1, 2, 3, real-valued vector of length 10 | Control format indicator (CFI) value. When the CFI value does not vary between subframes, specify this field as a scalar. Otherwise, specify this field as a vector, where the *k*th element corresponds to the CFI value of the *k*th subframe.<br><br>The CFI value varies between subframes for these RMCs when you specify the `duplexmode` input as `'TDD'` mode, the `CFI` varies per subframe for these RMCs: `'R.0'`, `'R.5'`, `'R.6'`, `'R.6-27RB'`, `'R.12-9RB'`. |
| **`PCFICHPower`** | Real-valued scalar | PCFICH symbol power adjustment, in dB |
| **`Ng`** | `'Sixth'`, `'Half'`, `'One'`, `'Two'` | HICH group multiplier |
| **`PHICHDuration`** | `'Normal'`, `'Extended'` | PHICH duration |
| **`HISet`** | 112-by-3 matrix | Maximum PHICH groups (112), as specified in section 6.9 of TS 36.211, with the first PHICH sequence of each group set to ACK). For more information, see `ltePHICH`. |
| **`PHICHPower`** | Real-valued scalar | PHICH symbol power, in dB |
| **`NFrame`** | Nonnegative integer | Frame number |

| Parameter Field | Values | Description |
|---|---|---|
| **NSubFrame** | Nonnegative integer | Subframe number |
| **TotSubFrames** | Nonnegative integer | Total number of subframes to generate |
| **Windowing** | Nonnegative integer | Number of time-domain samples over which the function applies windowing and overlapping of OFDM symbols |
| **DuplexMode** | `'FDD'`, `'TDD'` | Duplexing mode, returned as one of these values<br>• `'FDD'` — Frequency division duplex<br>• `'TDD'` — Time division duplex |
| **CSIRSPeriod** | `'On'`, `'Off'`, integer in the interval [0, 154], two-element row vector of nonnegative integers, cell array | CSI-RS subframe configurations for CSI-RS resources, returned as one of these values.<br>• `'On'` or `'Off`<br>• An integer in the interval [0, 154] corresponding to the parameter $I_{\text{CSI-RS}}$, specified in Table 6.10.5.3-1 of TS 36.211<br>• A vector of the form $[T_{\text{CSI-RS}}\ \Delta_{\text{CSI-RS}}]$, in accordance with Table 6.10.5.3-1 of TS 36.211<br>• A cell array of configurations for each resource.<br>This field applies only when the `TxScheme` field is `'Port7-14'`. |
| These fields are only present and applicable for `'Port7-14'` transmission scheme (`TxScheme`) and only required in `rmccfg` if `CSIRSPeriod` is not set to `'Off'`. | | |
| **CSIRSConfig** | Nonnegative integer | Array CSI-RS configuration indices. See Table 6.10.5.2-1 of TS 36.211. |
| **CSIRefP** | 1, 2, 4, 8 | Array of number of CSI-RS antenna ports |
| These fields are only present and applicable for `'Port7-14'` transmission scheme (`TxScheme`) | | |
| **ZeroPowerCSIRSPeriod** | `'Off'` (default), `'On'`, `Icsi-rs` (0,...,154), `[Tcsi-rs Dcsi-rs]`. You can also specify values in a cell array of configurations for each resource. | Zero power CSI-RS subframe configurations for one or more zero power CSI-RS resource configuration index lists. Multiple zero power CSI-RS resource lists can be configured from a single common subframe configuration or from a cell array of configurations for each resource list. |
| The following field is only applicable for `'Port7-14'` transmission scheme (`TxScheme`) and only required in `rmccfg` if `CSIRSPeriod` is not set to `'Off'`. | | |

| Parameter Field | Values | Description |
|---|---|---|
| **ZeroPowerCSIRSConfig** | 16-bit bitmap character vector or string scalar (truncated if not 16 bits or `'0'` MSB extended), or a numeric list of CSI-RS configuration indices. You can also specify values in a cell array of configurations for each resource. | Zero power CSI-RS resource configuration index lists (TS 36.211 Section 6.10.5.2). Specify each list as a 16-bit bitmap character vector or string scalar (if less than 16 bits, then `'0'` MSB extended), or as a numeric list of CSI-RS configuration indices from TS 36.211 Table 6.10.5.2-1 in the `'4'` CSI reference signal column. Multiple lists can be defined using a cell array of individual lists. |
| **PDSCH** | Scalar structure | PDSCH transmission configuration substructure |
| **SIB** | Scalar structure | Include a SIB message by adding the `SIB` substructure to the `lteRMCDL` function configuration output structure, `rmccfgout`, after it is generated and before using the `rmccfgout` structure as input to `lteRMCDLTool`. |
| **OCNGPDCCHEnable** | `'Off'`, `'On'` | Enable PDCCH OFDMA channel noise generator (OCNG). See footnote. |
| **OCNGPDCCHPower** | Scalar integer, `0` (default) | PDCCH OCNG power in dB |
| **OCNGPDSCHEnable** | `'Off'`, `'On'` | Enable PDSCH OCNG |
| **OCNGPDSCHPower** | Scalar integer, defaults to `PDSCH.Rho` (default) | PDSCH OCNG power in dB |
| **OCNGPDSCH** | Scalar structure | PDSCH OCNG configuration substructure |
| **OCNG** | `'Off'`, `'On'`. `'Disable'` and `'Enable'` are also accepted. | OFDMA channel noise generator<br><br>**Note** This parameter will be removed in a future release. Use the PDCCH and PDSCH-specific OCNG parameters instead. |
| The following fields are only present and applicable for `'TDD'` duplex mode (`DuplexMode`). | | |
| **SSC** | 0 (default), 1, 2, 3, 4, 5, 6, 7, 8, 9 | Special subframe configuration (SSC) |
| **TDDConfig** | 0, 1 (default), 2, 3, 4, 5, 6 | Uplink–downlink configuration<br><br>See footnote. |
| **SamplingRate** | Numeric scalar | Carrier sampling rate in Hz, $(N_{SC}/N_{SYM}) \times 3.84e6$, where $N_{SC}$ is the number of subcarriers and $N_{SYM}$ is the number of OFDM symbols in a subframe. |

| Parameter Field | Values | Description |
|---|---|---|
| `Nfft` | Scalar integer, typically one of {128, 256, 512, 1024, 1536, 2048} for standard channel bandwidths {`'1.4MHz'`, `'3MHz'`, `'5MHz'`, `'10MHz'`, `'15MHz'`, `'20MHz'`}, respectively. | Number of FFT frequency bins |

**1** CFI is equal to the number of symbols allocated to:

- PDCCH - 1 for `NDLRB < 10`
- PDCCH for `NDLRB >= 10`

For the RMCs, the number of symbols allocated to PDCCH varies with channel bandwidth setting,

- 2 symbols for 20 MHz, 15 MHz, and 10 MHz
- 3 symbols for 5 MHz and 3 MHz
- 4 symbols for 1.4 MHz

In the TDD mode, only two OFDM symbols are allocated to PDCCH in subframes 1 and 6 irrespective of the channel bandwidth. Therefore, the CFI value varies per subframe for the 5 MHz and 3 MHz and 1.4 MHz channel bandwidths, that is for bandwidths where PDCCH symbol allocation is not two for other subframes.

**2** The PDCCH ONCG fills the unused PDCCH resource elements with QPSK symbols using either single port or transmit diversity depending on the number of cell RS ports.

**3** All supported RMCs use TDDConfig 1 by default. When you specify a value different then the default, the full parameter set is configured according to the following rules.

- Preserve subframe 0 (downlink) for all TDDConfig — The values of the parameters in subframe 0 of TDDConfig 1 is applied in all other TDDConfig.
- Preserve special subframe behaviour — The values of the parameters in special subframes of TDDConfig 1 is applied in all other TDDConfig.
- Preserve subframe 5 (downlink) for all TDDConfig — The values of the parameters in subframe 5 of TDDConfig 1 is applied to all other TDDConfig. For all RMCs currently supported, subframe 5 is treated separately from other subframes. According to TS 36.101 Section A.3.1, “Unless otherwise stated, no user data is scheduled on subframes 5 in order to facilitate the transmission of system information blocks (SIB).” Hence the RC value, if present, determines the behaviour of subframe 5. This means that subframe 5 is not transmitted for other RMCs, with the exception of sustained data rate RMCs R.31-3A and R.31-4.
- All other downlink subframes use the same settings as subframe 9.

#### PDSCH Substructure

The substructure PDSCH relates to the physical channel configuration and contains these fields:

<table>
<tr><th>Parameter Field</th><th>Values</th><th>Description</th></tr>
<tr><td>TxScheme</td><td>'Port0', 'TxDiversity', 'CDD', 'SpatialMux', 'MultiUser', 'Port5', 'Port7-8', 'Port8', 'Port7-14'.</td><td>PDSCH transmission scheme, specified as one of the following options.
<table>
<tr><th>Transmission scheme</th><th>Description</th></tr>
<tr><td>'Port0'</td><td>Single antenna port, port 0</td></tr>
<tr><td>'TxDiversity'</td><td>Transmit diversity</td></tr>
<tr><td>'CDD'</td><td>Large delay cyclic delay diversity scheme</td></tr>
<tr><td>'SpatialMux'</td><td>Closed loop spatial multiplexing</td></tr>
<tr><td>'MultiUser'</td><td>Multi-user MIMO</td></tr>
<tr><td>'Port5'</td><td>Single-antenna port, port 5</td></tr>
<tr><td>'Port7-8'</td><td>Single-antenna port, port 7, when NLayers = 1. Dual layer transmission, ports 7 and 8, when NLayers = 2.</td></tr>
<tr><td>'Port8'</td><td>Single-antenna port, port 8</td></tr>
<tr><td>'Port7-14'</td><td>Up to eight layer transmission, ports 7–14</td></tr>
</table></td></tr>
<tr><td>Modulation</td><td>'QPSK', '16QAM', '64QAM', or '256QAM'</td><td>Modulation type, specified as a character vector, cell array of character vectors, or string array. If blocks, each cell is associated with a transport block.</td></tr>
<tr><td>NLayers</td><td>Integer from 1 to 8</td><td>Number of transmission layers.</td></tr>
<tr><td>Rho</td><td>0 (default), numeric scalar</td><td>PDSCH resource element power allocation, in dB</td></tr>
<tr><td>RNTI</td><td>0 (default), scalar integer</td><td>Radio network temporary identifier (RNTI) value (16 bits)</td></tr>
<tr><td>RVSeq</td><td>Integer vector (0,1,2,3), specified as a one or two row matrix (for one or two codewords)</td><td>Redundancy version (RV) indicator used by all HARQ processes, returned as a numeric matrix. RVSeq is a one- or two-row matrix for one or two codewords, respectively. The number of columns in RVSeq equals the number of transmissions of the transport blocks associated with a HARQ process. The RV sequence specified in each column is applied to the transmission of the transport blocks. If RVSeq is a scalar (or column vector in the case of two codewords), then there is a single initial transmission of each block with no retransmissions. If RVSeq is a row vector in a two-codeword transmission, then the same RV sequence is applied to both codewords.</td></tr>
</table>

| Parameter Field | Values | Description |
|---|---|---|
| **RV** | Integer vector (0,1,2,3). A one or two column matrix (for one or two codewords). | Specifies the redundancy version for one or two codewords used in the initial subframe number, `NSubframe`. This parameter field is only for informational purposes and is read-only. |
| **NHARQProcesses** | 1, 2, 3, 4, 5, 6, 7, or 8 | Number of HARQ processes per component carrier |
| **NTurboDecits** | 5 (default), nonnegative scalar integer | Number of turbo decoder iteration cycles |
| **PRBSet** | Integer column vector or two-column matrix | Zero-based physical resource block (PRB) indices corresponding to the slot-wise resource allocations for this PDSCH. The function returns this field as one of these values.<br>• a column vector, the resource allocation is the same in both slots of the subframe,<br>• a two-column matrix, this parameter specifies different PRBs for each slot in a subframe,<br>• a cell array of length 10 (corresponding to a frame, if the allocated physical resource blocks vary across subframes).<br>This field varies per subframe for these RMCs: `'R.25'` (with TDD), `'R.26'` (with TDD), `'R.27'` (with TDD), `'R.43'` (with FDD), `'R.44'`, `'R.45'`, `'R.48'`, `'R.50'`, `'R.51'`, `'R.68-1'`, and `'R.105'`. |
| **TargetCodeRate** | Numeric scalar or one or two row numeric matrix | Target code rates for one or two codewords for each subframe in a frame. Used for calculating the transport block sizes according to TS 36.101 [1], Annex A.3.1.<br>If both `TargetCodeRate` and `TrBlkSizes` are not provided at the input, and the RC does not have a single ratio target code rate in TS 36.101, Table A.3.1.1-1, `TargetCodeRate == ActualCodeRate`. |
| **ActualCodeRate** | One or two row numeric matrix | Actual code rates for one or two codewords for each subframe in a frame, calculated according to TS 36.101 [1], Annex A.3.1. The maximum actual code rate is 0.93. This parameter field is only for informational purposes and is read-only. |
| **TrBlkSizes** | One or two row numeric matrix | Transport block sizes for each subframe in a frame |
| **CodedTrBlkSizes** | One or two row numeric matrix | Coded transport block sizes for one or two codewords. This parameter field is for informational purposes and is read-only. |

| Parameter Field | Values | Description |
|---|---|---|
| **DCIFormat** | 'Format0', 'Format1', 'Format1A', 'Format1B', 'Format1C', 'Format1D', 'Format2', 'Format2A', 'Format2B', 'Format2C', 'Format2D', 'Format3', 'Format3A', 'Format4', 'Format5', 'Format5A' | Downlink control information (DCI) format type of the PDCCH associated with the PDSCH. See lteDCI. |
| **PDCCHFormat** | 0, 1, 2, 3 | Aggregation level of PDCCH associated with PDSCH |
| **PDCCHPower** | Numeric scalar | PDCCH power in dB |
| **CSIMode** | 'PUCCH 1-0', 'PUCCH 1-1', 'PUSCH 1-2', 'PUSCH 3-0', 'PUSCH 3-1' | CSI reporting mode |
| **PMIMode** | 'Wideband' (default), 'Subband' | PMI reporting mode. PMIMode='Wideband' corresponds to PUSCH reporting Mode 1-2 or PUCCH reporting Mode 1-1 (PUCCH Report Type 2) and PMIMode='Subband' corresponds to PUSCH reporting Mode 3-1. |
| The following field is only present for 'SpatialMux' transmission scheme (TxScheme). | | |
| **PMISet** | Integer vector with element values from 0 to 15. | Precoder matrix indication (PMI) set. It can contain either a single value, corresponding to single PMI mode, or multiple values, corresponding to multiple or subband PMI mode. The number of values depends on CellRefP, transmission layers and TxScheme. For more information about setting PMI parameters, see ltePMIInfo. |
| The following field is only present for 'Port7-8', 'Port8', or 'Port7-14' transmission schemes (TxScheme). | | |
| **NSCID** | 0 (default), 1 | Scrambling identity (*ID*) |
| The following fields are only present for UE-specific beamforming ('Port5', 'Port7-8', 'Port8', or 'Port7-14'). | | |
| **W** | Numeric matrix | NLayers-by-*P* precoding matrix for the wideband UE-specific beamforming of the PDSCH symbols. *P* is the number of transmit antennas. When W is not specified, no precoding is applied. |
| **NTxAnts** | Nonnegative scalar integer | Number of transmission antennas. |
| **HARQProcessSequence** | 1-by-$L_{HARQ\_Seq}$ integer vector. | One-based HARQ process indices for the internal HARQ scheduling sequence. The sequence of length $L_{HARQ\_Seq}$ is optimized according to transport block sizes, number of HARQ processes, duplex mode, and when in TDD mode the UL/DL configuration.<br><br>See footnote. |

| Parameter Field | Values | Description |
|---|---|---|
| **1** | The function returns valid `TrBlkSizes` and `CodedTrBlkSizes` set to 0 when `PRBSet` is empty, indicating there is no PDSCH allocation in this frame. | |
| **2** | The HARQ process sequence table is calculated according to the procedure detailed in 3GPP Tdoc R5-095777 ("Scheduling of retransmissions and number of active HARQ processes for DL performance RMC-s")<br>• For the case when `NHARQProcesses` = 1, the `HARQProcessSequence` is `[1 0 0 0 0 0 0 0 0 0]`. Using this HARQ process sequence, only the `TrBlkSize` corresponding to subframe 0 gets transmitted. There is no transmission in other subframes, even if the transport block sizes in other subframes are nonzero. | |

#### SIB Substructure

If the substructure `SIB` has been added to `rmccfg`, SIB1 messages and the associated PDSCH and PDCCH can be generated. The `SIB` substructure includes these fields:

| Parameter Field | Values | Description |
|---|---|---|
| **`Data`** | (0,1), bit array | SIB1 transport block information bits<br>See footnote. |
| **`VRBStart`** | variable, see rules in TS 36.213 Section 7.1.6.3 | Virtual RB allocation starting resource block, $RB_{start}$. |
| **`VRBLength`** | variable, see rules in TS 36.213 Section 7.1.6.3 | Length in terms of virtual contiguously allocated resource blocks, $L_{CRBs}$. |
| **`Enable`** | `'On'` (default), `'Off'` | Enable/Disable SIB generation |
| **`DCIFormat`** | `'Format1A'` (default) or `'Format1C'` | Downlink control information (DCI) format |
| **`AllocationType`** | 0 (default) or 1, single bit flag | Localized (0) or distributed (1) allocation of virtual resource blocks for Resource allocation type 2 |
| The following parameter is only applicable when `DCIFormat = 'Format1A'`. | | |
| **`N1APRB`** | 2 or 3 | Transport block set selection parameter, $N_{PRB}^{1A}$<br>Indicates the column in TS 36.213, Table 7.1.7.2.1-1 for transport block size selection. The default is the smallest transport block size, in either column 2 or 3, that is bigger than or equal to the length of the `Data` field. Also see TS 36.212 Section 5.3.3.1.3 and TS 36.213 Section 7.1.7. |
| The following parameter is only applicable when using distributed allocation (`AllocationType` = 1). | | |
| **`Gap`** | 0 or 1 | Distributed allocation gap, '0' for $N_{\text{gap},1}$ or '1' for $N_{\text{gap},2}$ |

| Parameter Field | Values | Description |
|---|---|---|
| **1** | The set of valid transport block sizes is specified in TS 36.213 [4], Table 7.1.7.2.1-1. Only columns 2 and 3 apply to the SIB DL-SCH. The `Data` field is padded with zeros to the closest valid size from this table. | |

**Note**

- Per TS 36.321 [5], Section 6.1.1, the lowest order information bit of the `SIB.Data` field is mapped to the most significant bit of the SIB1 transport block.
- For subframe 5, per TS 36.101 [1], Annex A.3, reference PDSCH transmissions are not scheduled in subframe 5 except for the SIB1 associated PDSCH.
- Setting the `OCNG` parameter field `'On'` fills all unused, unscheduled PDSCH resource elements with QPSK modulated random data.
- The values for CFI and PRBSet can vary per subframe. If these parameters are arrays, then the function cyclically steps through the elements of the array starting with the index given by mod(*NSubframe*,length(*parameter*)). When *parameter* is `PRBSet`, the parameter must be a cell array of column vectors or slot-wise matrices.
- The PHICH symbols carry a single ACK on the first PHICH instance in each PHICH group.

#### OCNGPDSCH Substructure

The substructure, `OCNGPDSCH`, defines the OCNG patterns in associated RMCs and tests according to TS 36.101 [1], Section A.5. `OCNGPDSCH` contains these fields which can also be customized with the full range of PDSCH-specific values.

| Parameter Field | Values | Description |
|---|---|---|
| **`Modulation`** | OCNG `Modulation` has same setting options as `rmccfgout.PDSCH.Modulation` | See `rmccfgout.PDSCH.Modulation` |
| **`TxScheme`** | OCNG `TxScheme` has same setting options as `rmccfgout.PDSCH.TxScheme` | See `rmccfgout.PDSCH.TxScheme` |
| **`RNTI`** | 0 (default), scalar integer | OCNG radio network temporary identifier (RNTI) value (16 bits) |

Data Types: `struct`

# Version History

**Introduced in R2014a**

**R2019b: This function now opens the LTE Waveform Generator app**
*Behavior changed in R2019b*

In previous releases, the input-free syntaxes of this function opened the **LTE Downlink RMC Generator** app. Starting in R2019b, input-free calls to this function open the **LTE Waveform Generator** app for a downlink RMC waveform.

## References

[1] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Networ*k. URL: https://www.3gpp.org.

[2] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[3] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[4] 3GPP TS 36.213. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[5] 3GPP TS 36.321. “Evolved Universal Terrestrial Radio Access (E-UTRA); Medium Access Control (MAC) protocol Specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

**Apps**
**LTE Waveform Generator**

**Functions**
`lteRMCDL` | `lteRMCULTool` | `lteTestModelTool`

# lteRMCUL

Uplink reference measurement channel or FRC configuration

## Syntax

```
rmccfgout = lteRMCUL(rc,duplexmode,totsubframes)
rmccfgout = lteRMCUL(rmccfg)
```

## Description

`rmccfgout = lteRMCUL(rc,duplexmode,totsubframes)` returns a configuration structure for the reference channel defined by `rc` using a channel-specific default configuration. `duplexmode` and `totsubframes` are optional input parameters which define the duplexing mode and total number of subframes to be generated.

Use `rmccfgout` to generate a waveform with the fixed reference channel (FRC) generator tool, `lteRMCULTool`. The field names and default values of FRCs are in accordance with TS 36.104 [2], Annex A.

`rmccfgout = lteRMCUL(rmccfg)` returns a fully configured structure for the reference channel partially, or wholly, defined by the input structure, `rmccfg`. Any parameters missing at the input are initialized based on the `rc` field, if present in `rmccfg`, or the reference channel `'A1-1'` otherwise.

`rmccfg` can include the field `SRS` to enable SRS-related configuration parameters.

## Examples

### Create Uplink RMC

Using the reference measurement channel designation, create an uplink RMC configuration for RC 'A7-4'.

```
rmc = lteRMCUL('A7-4')
```

```
rmc = struct with fields:
                RC: 'A7-4'
             NULRB: 25
           NCellID: 0
            NFrame: 0
         NSubframe: 0
    CyclicPrefixUL: 'Normal'
       CyclicShift: 0
         Shortened: 0
           Hopping: 'Off'
          SeqGroup: 0
      TotSubframes: 10
              RNTI: 1
           NTxAnts: 1
         Windowing: 0
        DuplexMode: 'FDD'
```

```
              PUSCH: [1x1 struct]
```

### Create Uplink RMC Configuration

Create a configuration structure for RC A1-1 as specified in TS 36.104.

```
rmc.RC = 'A1-1';
rmc.NCellID = 100;
rmcOut = lteRMCUL(rmc)
```

```
rmcOut = struct with fields:
                RC: 'A1-1'
             NULRB: 6
           NCellID: 100
            NFrame: 0
         NSubframe: 0
    CyclicPrefixUL: 'Normal'
       CyclicShift: 0
         Shortened: 0
           Hopping: 'Off'
          SeqGroup: 0
      TotSubframes: 10
              RNTI: 1
           NTxAnts: 1
         Windowing: 0
        DuplexMode: 'FDD'
             PUSCH: [1x1 struct]
```

```
rmcOut.PUSCH
```

```
ans = struct with fields:
          Modulation: 'QPSK'
             NLayers: 1
     DynCyclicShift: 0
           NBundled: 0
            BetaACK: 2
            BetaCQI: 2
             BetaRI: 2
     NHARQProcesses: 8
              RVSeq: [0 2 3 1]
                 RV: 0
       NTurboDecIts: 5
          OrthCover: 'On'
                PMI: 0
             PRBSet: [6x1 double]
     TargetCodeRate: 0.3333
     ActualCodeRate: [0.3611 0.3611 0.3611 0.3611 0.3611 0.3611 0.3611 0.3611 0.3611 0.3611]
         TrBlkSizes: [600 600 600 600 600 600 600 600 600 600]
    CodedTrBlkSizes: [1728 1728 1728 1728 1728 1728 1728 1728 1728 1728]
```

### Customize Uplink RMC

Create a new customized parameter set by overriding selected values of an existing preset RMC. Define a full-band 5MHz PUSCH using 64QAM modulation and 1/3 rate.

Looking at TS 36.104 Annex A reference measurement channels, A1-3 matches this criteria but with QPSK modulation.

Create a configuration structure for RC A1-3 as specified in TS 36.104.

```
rmc.RC = 'A1-3';
rmcout = lteRMCUL(rmc,1);
rmcout.PUSCH
```

```
ans = struct with fields:
         Modulation: 'QPSK'
            NLayers: 1
     DynCyclicShift: 0
           NBundled: 0
            BetaACK: 2
            BetaCQI: 2
             BetaRI: 2
     NHARQProcesses: 8
              RVSeq: [0 2 3 1]
                 RV: 0
       NTurboDecIts: 5
          OrthCover: 'On'
                PMI: 0
             PRBSet: [25x1 double]
     TargetCodeRate: 0.3333
     ActualCodeRate: [0.3111 0.3111 0.3111 0.3111 0.3111 0.3111 0.3111 0.3111 0.3111 0.3111]
         TrBlkSizes: [2216 2216 2216 2216 2216 2216 2216 2216 2216 2216]
    CodedTrBlkSizes: [7200 7200 7200 7200 7200 7200 7200 7200 7200 7200]
```

Override the PUSCH modulation, setting it to 64QAM. Create a new configuration structure. Inspect `rmcout` to see the modulation is 64QAM and the PUSCH transport block sizes and physical channel capacities have been updated to maintain the same 1/3 code rate.

```
rmc.PUSCH.Modulation = '64QAM';
rmcOverrideOut = lteRMCUL(rmc,1);
rmcOverrideOut
```

```
rmcOverrideOut = struct with fields:
                RC: 'A1-3'
             NULRB: 25
            NCellID: 0
             NFrame: 0
         NSubframe: 0
    CyclicPrefixUL: 'Normal'
       CyclicShift: 0
         Shortened: 0
           Hopping: 'Off'
          SeqGroup: 0
      TotSubframes: 10
              RNTI: 1
            NTxAnts: 1
         Windowing: 0
```

```
        DuplexMode: 'FDD'
             PUSCH: [1x1 struct]
```

```
rmcOverrideOut.PUSCH
```

```
ans = struct with fields:
         Modulation: '64QAM'
            NLayers: 1
     DynCyclicShift: 0
           NBundled: 0
            BetaACK: 2
            BetaCQI: 2
             BetaRI: 2
     NHARQProcesses: 8
              RVSeq: [0 2 3 1]
                 RV: 0
       NTurboDecIts: 5
          OrthCover: 'On'
                PMI: 0
            PRBSet: [25x1 double]
     TargetCodeRate: 0.3333
     ActualCodeRate: [0.3378 0.3378 0.3378 0.3378 0.3378 0.3378 0.3378 0.3378 0.3378 0.3378]
         TrBlkSizes: [7224 7224 7224 7224 7224 7224 7224 7224 7224 7224]
    CodedTrBlkSizes: [21600 21600 21600 21600 21600 21600 21600 21600 21600 21600]
```

## Input Arguments

### `rc` — Reference channel number

`'A1-1'` | `'A1-2'` | `'A1-3'` | `'A1-4'` | `'A1-5'` | `'A2-1'` | `'A2-2'` | `'A2-3'` | `'A3-1'` | `'A3-2'` | `'A3-3'` | `'A3-4'` | `'A3-5'` | `'A3-6'` | `'A3-7'` | `'A4-1'` | `'A4-2'` | `'A4-3'` | `'A4-4'` | `'A4-5'` | `'A4-6'` | `'A4-7'` | `'A4-8'` | `'A5-1'` | `'A5-2'` | `'A5-3'` | `'A5-4'` | `'A5-5'` | `'A5-6'` | `'A5-7'` | `'A7-1'` | `'A7-2'` | `'A7-3'` | `'A7-4'` | `'A7-5'` | `'A7-6'` | `'A8-1'` | `'A8-2'` | `'A8-3'` | `'A8-4'` | `'A8-5'` | `'A8-6'` | `'A11-1'` | `'A3-2-9RB'` | `'A4-3-9RB'`

Reference channel number, specified as a character vector or string scalar. Use double quotes or string. This argument represents the reference measurement channel (RMC) number, or fixed reference channel (FRC), as described in TS 36.104[2]. See “UL Reference Channel Options” on page 2-985 for a list of the default top-level configuration associated with the available uplink reference channels.

Data Types: `char` | `string`

### `duplexmode` — Duplexing mode

`'FDD'` (default) | optional | `'TDD'`

Duplexing mode, specified as `'FDD'` or `'TDD'`. It represents the frame structure type.

Data Types: `char` | `string`

### `totsubframes` — Total number of subframes

10 (default) | optional | positive numeric scalar

Total number of subframes, specified as a numeric scalar. This argument specifies the total number of subframes that form the resource grid.

Data Types: `double`

### `rmccfg` — Reference channel configuration

structure

Reference channel configuration, specified as a structure. The structure defines any, or all, of the fields or subfields contained in the output structure, `rmccfgout`. Any undefined fields are given appropriate default values.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`RC`** | Optional | `'A1-1'` (default), `'A1-2'`, `'A1-3'`, `'A1-4'`, `'A1-5'`, `'A2-1'`, `'A2-2'`, `'A2-3'`, `'A3-1'`, `'A3-2'`, `'A3-3'`, `'A3-4'`, `'A3-5'`, `'A3-6'`, `'A3-7'`, `'A4-1'`, `'A4-2'`, `'A4-3'`, `'A4-4'`, `'A4-5'`, `'A4-6'`, `'A4-7'`, `'A4-8'`, `'A5-1'`, `'A5-2'`, `'A5-3'`, `'A5-4'`, `'A5-5'`, `'A5-6'`, `'A5-7'`, `'A7-1'`, `'A7-2'`, `'A7-3'`, `'A7-4'`, `'A7-5'`, `'A7-6'`, `'A8-1'`, `'A8-2'`, `'A8-3'`, `'A8-4'`, `'A8-5'`, `'A8-6'`, `'A11-1'`, `'A3-2-9RB'`, `'A4-3-9RB'` | Reference measurement channel (RMC) number or type, as specified in TS 36.104 Annex A.<br><br>[2]. |
| **`SRS`** | Optional | `'off'` (default), `'on'` | Enable SRS related configuration parameters (set SRS to `'on'`) for RMCs which optionally support SRS, or a complete or part SRS structure. If absent, no SRS configuration is created. |

Data Types: `struct`

## Output Arguments

### `rmccfgout` — Configuration parameters

structure

#### Configuration Parameters Structure

Configuration parameters, returned as a structure. `rmccfgout` contains the following fields.

| Parameter Field | Values | Description |
|---|---|---|
| **RC** | `'A1-1'` (default), `'A1-2'`, `'A1-3'`, `'A1-4'`, `'A1-5'`, `'A2-1'`, `'A2-2'`, `'A2-3'`, `'A3-1'`, `'A3-2'`, `'A3-3'`, `'A3-4'`, `'A3-5'`, `'A3-6'`, `'A3-7'`, `'A4-1'`, `'A4-2'`, `'A4-3'`, `'A4-4'`, `'A4-5'`, `'A4-6'`, `'A4-7'`, `'A4-8'`, `'A5-1'`, `'A5-2'`, `'A5-3'`, `'A5-4'`, `'A5-5'`, `'A5-6'`, `'A5-7'`, `'A7-1'`, `'A7-2'`, `'A7-3'`, `'A7-4'`, `'A7-5'`, `'A7-6'`, `'A8-1'`, `'A8-2'`, `'A8-3'`, `'A8-4'`, `'A8-5'`, `'A8-6'`, `'A11-1'`, `'A3-2-9RB'`, `'A4-3-9RB'` | Reference channel number |
| **NULRB** | Scalar integer from 6 to 110 | Number of uplink resource blocks. ($N_{\mathrm{RB}}^{\mathrm{UL}}$) |
| **NCellID** | Integer from 0 to 503 | Physical layer cell identity |
| **NFrame** | 0 (default), nonnegative scalar integer | Frame number |
| **NSubFrame** | 0 (default), nonnegative scalar integer | Initial subframe number |
| **CyclicPrefixUL** | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **CyclicShift** | 0, 1, 2, 3, 4, 5, 6, 7 | Cyclic shift. This argument yields $n_{DMRS}^{(1)}$. |
| **Shortened** | 0 (default), 1 | Subframe shortened flag. If the function sets the flag to `1`, the last symbol of the subframe is not used. Subframes with possible SRS transmission require this flag to be set. |
| **Hopping** | `'Off'` (default), `'Group'`, or `'Sequence'` | Hopping type |
| **SeqGroup** | 0 (default), integer from 0 to 29 | PUSCH sequence group assignment ($\Delta_{\mathrm{SS}}$). |
| **TotSubFrames** | `10` (default)<br><br>Positive scalar integer | Total number of subframes to generate<br><br>This argument specifies the total number of subframes that form the resource grid. |
| **RNTI** | `1` (default)<br><br>Scalar integer | Radio network temporary identifier (RNTI) value (16 bits) |
| **NTxAnts** | 1, 2, 4 | Number of transmission antennas. |
| **Windowing** | Nonnegative scalar integer | The number of time-domain samples over which windowing and overlapping of SC-FDMA symbols is applied |

| Parameter Field | Values | Description |
|---|---|---|
| **DuplexMode** | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex<br>It represents the frame structure type. |
| **PUSCH** | Structure | PUSCH transmission configuration |
| **SRS** | Structure | Sounding Reference Signal (SRS) configuration |

**PUSCH Substructure**

The substructure `PUSCH` relates to the physical channel configuration and contains these fields:

| Parameter Field | Values | Description |
|---|---|---|
| **Modulation** | `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'` | Modulation format |
| **NLayers** | 1, 2, 3, 4 | Number of transmission layers. |
| **DynCyclicShift** | 0, 1, 2, 3, 4, 5, 6, 7 | Cyclic shift for DM-RS (yields $n_{DMRS}^{(2)}$). |
| **NBundled** | 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 | HARQ-ACK bundling scrambling sequence index |
| **BetaACK** | Scalar integer | Modulation and coding scheme (MCS) offset for HARQ-ACK bits, returned as a scalar integer. |
| **BetaCQI** | Scalar integer | Modulation and coding scheme (MCS) offset for CQI and PMI bits |
| **BetaRI** | Scalar integer | Modulation and coding scheme (MCS) offset for RI bits |
| **NHARQProcesses** | 1, 2, 3, 4, 5, 6, 7, 8 | Number of HARQ processes per component carrier |
| **RVSeq** | Numeric matrix | Redundancy version (RV) indicator used by all HARQ processes, returned as a numeric matrix. `RVSeq` is a one- or two-row matrix for one or two codewords, respectively. The number of columns in `RVSeq` equals the number of transmissions of the transport blocks associated with a HARQ process. The RV sequence specified in each column is applied to the transmission of the transport blocks. If `RVSeq` is a scalar (or column vector in the case of two codewords), then there is a single initial transmission of each block with no retransmissions. If `RVSeq` is a row vector in a two-codeword transmission, then the same RV sequence is applied to both codewords.<br>See footnote. |

| Parameter Field | Values | Description |
|---|---|---|
| **RV** | Numeric matrix | Redundancy version (RV) indicator in initial subframe, returned as a numeric matrix. This argument is a one- or two-column vector that specifies the redundancy version for one or two codewords used in the initial subframe number, `NSubframe`. This parameter field is only for informational purposes and is read-only. |
| **NTurboDecIts** | Positive scalar integer | Number of turbo decoder iteration cycles |
| **OrthCover** | `'Off'` (default), `'On'` | Orthogonal cover sequence flag.<br><br>Applies (`'On'`), or does not apply (`'Off'`), orthogonal cover sequence *w* (*Activate-DMRS-with OCC*). |
| **PMI** | Integer from 0 to 23 | Scalar precoder matrix indication (PMI) to be used during precoding |
| **PRBSet** | Integer matrix | Physical resource block set of indices, returned as an integer matrix. This argument is a 1- or 2-column matrix that contains the 0-based physical resource block indices (PRBs) corresponding to the resource allocations for this PUSCH. |
| **TargetCodeRate** | Scalar or vector | Target code rates for each subframe in a frame. Used for calculating the transport block sizes according to TS 36.101[1], Annex A.2.1.2.<br><br>If `TargetCodeRate` is not provided and `TrBlkSizes` is provided at the input, `TargetCodeRate == ActualCodeRate`. |
| **ActualCodeRate** | Numeric vector | Actual code rates for each subframe in a frame. The maximum actual code rate is 0.93. This parameter field is only for informational purposes and is read-only. |
| **TrBlkSizes** | Numeric vector | Transport block sizes for each subframe in a frame<br><br>See footnote. |
| **CodedTrBlkSizes** | Numeric vector | Coded transport block sizes for each a subframe in a frame, returned as a numeric vector. This parameter field is only for informational purposes and is read-only.<br><br>See footnote. |
| **1** The values of `RVSeq`, `TrBlkSizes`, and `CodedTrBlkSizes` are set according to the modulation scheme and `TargetCodeRate`. | | |

#### SRS Substructure

The substructure SRS contains these fields:

| Parameter Field | Values | Description |
|---|---|---|
| **`NTxAnts`** | 1 (default), 2, 4 | Number of transmission antennas. |
| **`BWConfig`** | 0, 1, 2, 3, 4, 5, 6, 7 | Cell-specific SRS Bandwidth Configuration value ($C_{\mathrm{SRS}}$) |
| **`BW`** | 0, 1, 2, 3 | UE-specific SRS Bandwidth value ($B_{\mathrm{SRS}}$) |
| **`ConfigIdx`** | Integer from 0 to 644 | Configuration index ($I_{\mathrm{SRS}}$) for UE-specific periodicity ($T_{\mathrm{SRS}}$) and subframe offset ($T_{\mathrm{offset}}$). |
| **`TxComb`** | 0 or 1 | Transmission comb. Controls SRS positions; SRS is transmitted in 6 carriers per resource block on odd (1) and even (0) resource indices. |
| **`HoppingBW`** | 0, 1, 2, 3 | SRS Frequency hopping configuration index ($b_{\mathrm{hop}}$) |
| **`FreqPosition`** | Integer from 0 to 23 | Frequency domain position ($n_{\mathrm{RRC}}$) |
| **`CyclicShift`** | 0 (default), integer from 0 to 7 | UE-specific cyclic shift ($n_{SRS}^{cs}$) |
| **`SeqGroup`** | 0 (default), integer from 0 to 29 | SRS sequence group number ($u$) |
| **`SeqIdx`** | 0 or 1 | Base sequence number ($v$) |
| **`SubframeConfig`** | Integer from 0 to 15 | Sounding reference signal (SRS) subframe configuration |
| The following fields are present only when `DuplexMode` is set to `'TDD'`. | | |
| **`NF4RachPreambles`** | 0, 1, 2, 3, 4, 5, 6 | Number of RACH preamble frequency resources of Format 4 in *UpPTS* |
| **`OffsetIdx`** | 0 or 1 | Choice of SRS Subframe Offset in the case of 2 ms SRS periodicity. This parameter indexes the two SRS Subframe Offset entries in the row specified by the `ConfigIdx` parameter in table 8.2-2 of TS 36.213 for the SRS Configuration Index. |

## More About

### UL Reference Channel Options

Initialization choices available for the uplink reference channel and associated top-level configuration defaults include:

| Reference channels | Reference channels (continued) | Reference channels (continued) |
|---|---|---|
| A1-1 (6 RB, QPSK, R=1/3) | A4-3 (6 RB, 16QAM, R=3/4) | A7-5 (25 RB, 16QAM, R=3/4) |
| A1-2 (15 RB, QPSK, R=1/3) | A4-4 (15 RB, 16QAM, R=3/4) | A7-6 (25 RB, 16QAM, R=3/4) |
| A1-3 (25 RB, QPSK, R=1/3) | A4-5 (25 RB, 16QAM, R=3/4) | A8-1 (3 RB, QPSK, R=1/3) |
| A1-4 (3 RB, QPSK, R=1/3) | A4-6 (50 RB, 16QAM, R=3/4) | A8-2 (6 RB, QPSK, R=1/3) |
| A1-5 (9 RB, QPSK, R=1/3) | A4-7 (75 RB, 16QAM, R=3/4) | A8-3 (12 RB, QPSK, R=1/3) |
| A2-1 (6 RB, 16QAM, R=2/3) | A4-8 (100 RB, 16QAM, R=3/4) | A8-4 (25 RB, QPSK, R=1/3) |
| A2-2 (15 RB, 16QAM, R=2/3) | A5-1 (1 RB, 64QAM, R=5/6) | A8-5 (25 RB, QPSK, R=1/3) |
| A2-3 (25 RB, 16QAM, R=2/3) | A5-2 (6 RB, 64QAM, R=5/6) | A8-6 (25 RB, QPSK, R=1/3) |
| A3-1 (1 RB, QPSK, R=1/3) | A5-3 (15RB, 64QAM, R=5/6) | A11-1 (3 RB, QPSK, R=11/27) |
| A3-2 (6 RB, QPSK, R=1/3) | A5-4 (25 RB, 64QAM, R=5/6) | A17-1 (6 RB, 256QAM, R=5/6) |
| A3-3 (15 RB, QPSK, R=1/3) | A5-5 (50 RB, 64QAM, R=5/6) | A17-2 (15 RB, 256QAM, R=5/6) |
| A3-4 (25 RB, QPSK, R=1/3) | A5-6 (75 RB, 64QAM, R=5/6) | A17-3 (25 RB, 256QAM, R=5/6) |
| A3-5 (50 RB, QPSK, R=1/3) | A5-7 (100 RB, 64QAM, R=5/6) | A17-4 (50 RB, 256QAM, R=5/6) |
| A3-6 (75 RB, QPSK, R=1/3) | A7-1 (3 RB, 16QAM, R=3/4) | A17-5(75 RB, 256QAM, R=5/6) |
| A3-7 (100 RB, QPSK, R=1/3) | A7-2 (6 RB, 16QAM, R=3/4) | A17-6 (100 RB, 256QAM, R=5/6) |
| A4-1 (1 RB, 16QAM, R=3/4) | A7-3 (12 RB, 16QAM, R=3/4) | A3-2-9RB (9 RB, QPSK, R=1/3) |
| A4-2 (1 RB, 16QAM, R=3/4) | A7-4 (25 RB, 16QAM, R=3/4) | A4-3-9RB (9 RB, 16QAM, R=3/4) |

The fields in the output configuration structure, `rmccfgout`, are initialized in accordance with the reference channels defined in TS 36.104, Annex A.

- `'A3-2-9RB'`, and `'A4-3-9RB'` are custom RMC configured for non-standard bandwidths but with the same code rate as the standardized versions.
- `'A11-1'` enables TTI bundling and the corresponding HARQ pattern (enhanced HARQ pattern for FDD).

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.104. “Evolved Universal Terrestrial Radio Access (E-UTRA); Base Station (BS) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[3] 3GPP TS 36.213. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteRMCULTool` | `lteRMCDL` | `lteTestModel`

# lteRMCULTool

Uplink RMC or FRC waveform generation

## Syntax

```
lteRMCULTool
[waveform,grid,rmccfgout] = lteRMCULTool(rc,trdata)
[waveform,grid,rmccfgout] = lteRMCULTool(rc,trdata,duplexmode,totsubframes)
[waveform,grid,rmccfgout] = lteRMCULTool(rmccfg,trdata)
[waveform,grid,rmccfgout] = lteRMCULTool(rmccfg,trdata,cqi,ri,ack)
```

## Description

`lteRMCULTool` starts the **LTE Waveform Generator** app for the parameterization and generation of a reference measurement channel (RMC) waveform. For a list of the default top-level configuration associated with the available uplink reference channels, see “UL Reference Channel Options” on page 2-998.

`[waveform,grid,rmccfgout] = lteRMCULTool(rc,trdata)` specifies the reference channel, `rc`, and information bits, `trdata`.

`[waveform,grid,rmccfgout] = lteRMCULTool(rc,trdata,duplexmode,totsubframes)` also accepts optional input arguments to define the duplex mode of the generated waveform and total number of subframes that make up the `grid`.

`[waveform,grid,rmccfgout] = lteRMCULTool(rmccfg,trdata)` where `rmccfg` specifies a reference channel structure. The reference channel structure with default parameters can easily be created with the function `lteRMCUL` then modified as desired.

`[waveform,grid,rmccfgout] = lteRMCULTool(rmccfg,trdata,cqi,ri,ack)` where support for control information transmission on PUSCH is specified in vectors `cqi`, `ri`, and `ack`. Together, these three fields form an uplink control information (UCI) message. If these particular control information bits are not present in this transmission, `cqi`, `ri`, and `ack` can be empty vectors. The UCI is encoded for PUSCH transmission using the processing defined in TS 36.212 [3], Section 5.2.4, consisting of UCI coding and channel interleaving. The vectors `cqi`, `ri`, and `ack` are not treated as data streams. Thus, each subframe contains the same CQI, RI, and ACK information bits.

## Examples

### Generate Uplink LTE FRC A3-2

Generate a time domain signal and a 3-dimensional array of the resource elements for A3-2 as specified in TS 36.104 Annex A. The A3-2 fixed reference channel (FRC) settings include: FDD, 1.4MHz, QPSK, and 1/3 code rate.

```
rmc = lteRMCUL('A3-2');
[waveform,grid,rmccfgout] = lteRMCULTool(rmc,1);
```

Inspect the FRC configuration settings.

```
rmccfgout
```

```
rmccfgout = struct with fields:
               RC: 'A3-2'
            NULRB: 6
          NCellID: 0
           NFrame: 0
        NSubframe: 0
    CyclicPrefixUL: 'Normal'
       CyclicShift: 0
         Shortened: 0
           Hopping: 'Off'
          SeqGroup: 0
      TotSubframes: 10
              RNTI: 1
            NTxAnts: 1
         Windowing: 0
        DuplexMode: 'FDD'
             PUSCH: [1x1 struct]
      SamplingRate: 1920000
              Nfft: 128
```

```
rmccfgout.PUSCH
```

```
ans = struct with fields:
             Modulation: 'QPSK'
                NLayers: 1
         DynCyclicShift: 0
               NBundled: 0
                BetaACK: 2
                BetaCQI: 2
                 BetaRI: 2
         NHARQProcesses: 8
                  RVSeq: [0 2 3 1]
                     RV: 0
           NTurboDecIts: 5
              OrthCover: 'On'
                    PMI: 0
                 PRBSet: [6x1 double]
         TargetCodeRate: 0.3333
         ActualCodeRate: [0.3611 0.3611 0.3611 0.3611 0.3611 0.3611 0.3611 0.3611 0.3611 0.3611]
             TrBlkSizes: [600 600 600 600 600 600 600 600 600 600]
        CodedTrBlkSizes: [1728 1728 1728 1728 1728 1728 1728 1728 1728 1728]
    HARQProcessSequence: [1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8 1 2 3 4
```

```
rmccfgout.PUSCH.ActualCodeRate
```

```
ans = 1×10

    0.3611    0.3611    0.3611    0.3611    0.3611    0.3611    0.3611    0.3611    0.3611    0.3
```

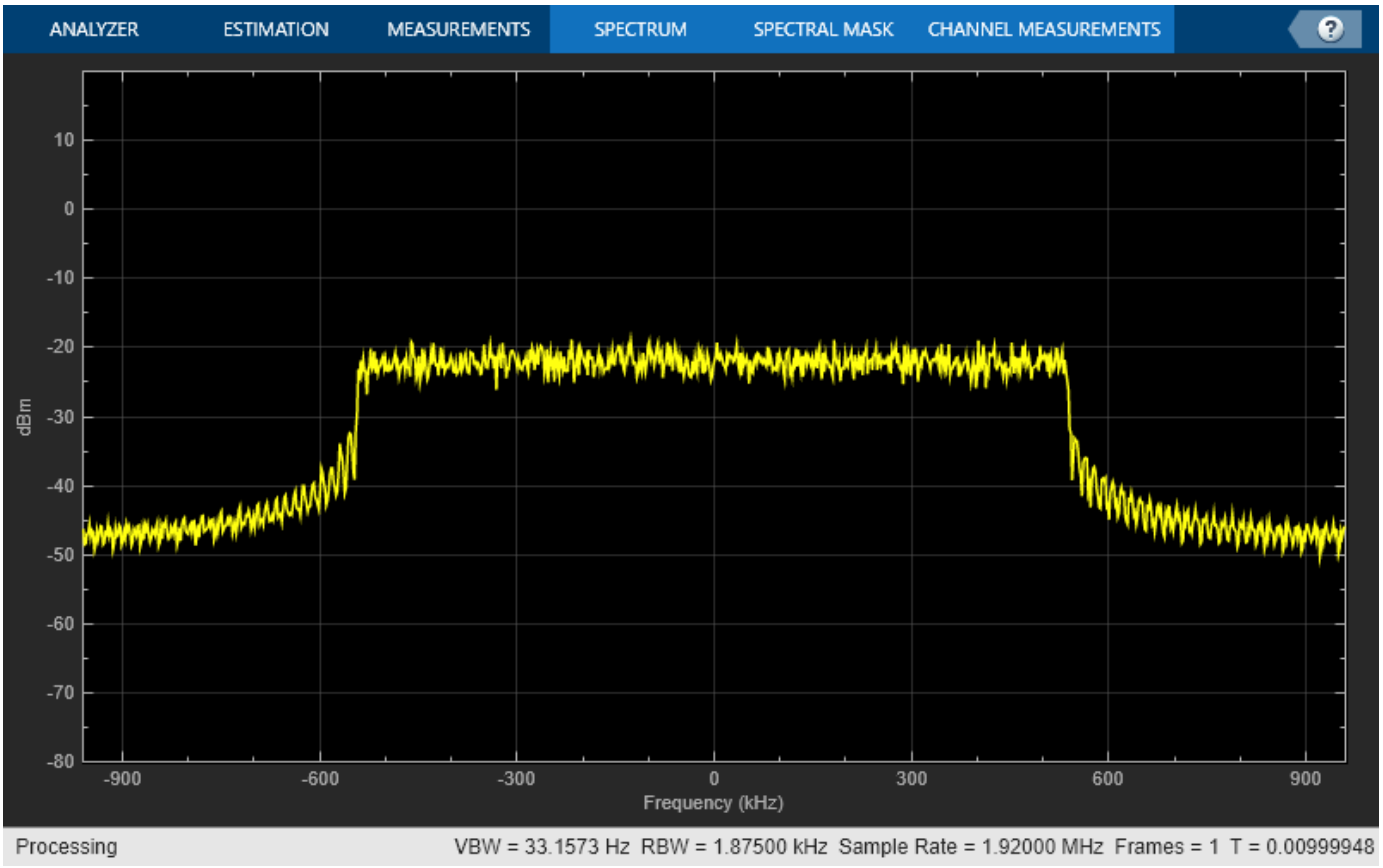
The actual code rate of 0.3611 is slightly higher then the target code rate of 1/3.

### Generate Uplink RMC Waveform

Generate a time-domain signal and a 2-D array of the resource elements for a modified A1-1 fixed reference channel.

Initialize the `frc` configuration structure and change the modulation scheme to `'16QAM'`. Generate the `txWaveform`, `txGrid`, and output the configuration structure. Create a spectrum analyzer object, setting the sampling rate. Plot the waveform.

```
frc = lteRMCUL('A1-1');
frc.PUSCH.Modulation = '16QAM';

[txWaveform,txGrid,rmcCfgOut] = lteRMCULTool(frc,[1;0;0;1]);

saScope = spectrumAnalyzer(SampleRate=rmcCfgOut.SamplingRate);
saScope(txWaveform)
```




### Customized Uplink RMC configuration

Create a new customized parameter set by overriding selected values of an existing preset RMC to define a full-band, 5MHz, PUSCH using 64QAM modulation, and 1/3 coding rate.

Begin with TS 36.104 Annex A, RMC A1-3, which matches this criteria but with QPSK modulation.

```
rmcOverride.RC = 'A1-3';
rmc = lteRMCUL(rmcOverride,1);
rmc.PUSCH
```

```
ans = struct with fields:
         Modulation: 'QPSK'
            NLayers: 1
     DynCyclicShift: 0
           NBundled: 0
            BetaACK: 2
            BetaCQI: 2
             BetaRI: 2
     NHARQProcesses: 8
              RVSeq: [0 2 3 1]
                 RV: 0
       NTurboDecIts: 5
          OrthCover: 'On'
                PMI: 0
             PRBSet: [25x1 double]
     TargetCodeRate: 0.3333
     ActualCodeRate: [0.3111 0.3111 0.3111 0.3111 0.3111 0.3111 0.3111 0.3111 0.3111 0.3111]
         TrBlkSizes: [2216 2216 2216 2216 2216 2216 2216 2216 2216 2216]
    CodedTrBlkSizes: [7200 7200 7200 7200 7200 7200 7200 7200 7200 7200]
```

Override the PUSCH modulation. `lteRMCUL` returns recomputed PUSCH transport block sizes and physical channel capacities to maintain the coding rate of R=1/3.

```
rmcOverride.PUSCH.Modulation = '64QAM';
rmc = lteRMCUL(rmcOverride,1);
rmc.PUSCH
```

```
ans = struct with fields:
         Modulation: '64QAM'
            NLayers: 1
     DynCyclicShift: 0
           NBundled: 0
            BetaACK: 2
            BetaCQI: 2
             BetaRI: 2
     NHARQProcesses: 8
              RVSeq: [0 2 3 1]
                 RV: 0
       NTurboDecIts: 5
          OrthCover: 'On'
                PMI: 0
             PRBSet: [25x1 double]
     TargetCodeRate: 0.3333
     ActualCodeRate: [0.3378 0.3378 0.3378 0.3378 0.3378 0.3378 0.3378 0.3378 0.3378 0.3378]
         TrBlkSizes: [7224 7224 7224 7224 7224 7224 7224 7224 7224 7224]
    CodedTrBlkSizes: [21600 21600 21600 21600 21600 21600 21600 21600 21600 21600]
```

## Input Arguments

### `rc` — Reference measurement channel

`'A1-1'` | `'A1-2'` | `'A1-3'` | `'A1-4'` | `'A1-5'` | `'A2-1'` | `'A2-2'` | `'A2-3'` | `'A3-1'` | `'A3-2'` | `'A3-3'` | `'A3-4'` | `'A3-5'` | `'A3-6'` | `'A3-7'` | `'A4-1'` | `'A4-2'` | `'A4-3'` | `'A4-4'` | `'A4-5'` | `'A4-6'` | `'A4-7'` | `'A4-8'` | `'A5-1'` | `'A5-2'` | `'A5-3'` | `'A5-4'` | `'A5-5'` | `'A5-6'` | `'A5-7'` | `'A7-1'` | `'A7-2'` | `'A7-3'` | `'A7-4'` | `'A7-5'` | `'A7-6'` | `'A8-1'` | `'A8-2'` | `'A8-3'` | `'A8-4'` | `'A8-5'` | `'A8-6'` | `'A11-1'` | `'A3-2-9RB'` | `'A4-3-9RB'`

Reference channel, specified as a character vector or string scalar. Use double quotes for string. This argument identifies the reference measurement channel (RMC) number, as specified in TS 36.104 [2]. See “UL Reference Channel Options” on page 2-998 for a list of the default top-level configuration associated with the available uplink reference channels.

Data Types: `char` | `string`

### `trdata` — Information bits

column vector | cell array of one or two column vectors

Information bits, specified as a column vector or a cell array containing one or two column vectors of bit values. Each vector contains the information bits stream to be coded across the duration of the generation, which represents multiple concatenated transport blocks. Internally these vectors are looped if the number of bits required across all subframes of the generation exceeds the length of the vectors provided. Looping on the information bits allows you to enter a short pattern, such as `[1;0;0;1]`, that is repeated as the input to the transport coding. The `TrBlkSizes` matrix field of `rmccfgout.PUSCH` defines the number of data bits taken from the information bit stream for each subframe of generation.

Data Types: `double` | `cell`

### `duplexmode` — Duplexing mode

`'FDD'` (default) | optional | `'TDD'`

Duplexing mode, specified as `'FDD'` or `'TDD'` to indicate the frame structure type of the generated waveform.

Data Types: `char` | `string`

### `totsubframes` — Total number of subframes

10 (default) | optional | positive numeric scalar

Total number of subframes, specified as a numeric scalar. Optional. This argument specifies the total number of subframes that form the resource grid.

Data Types: `double`

### `rmccfg` — Reference channel configuration

structure

Reference channel configuration, specified as a structure. The structure defines any (or all) of the fields or subfields. The reference configuration structure with default parameters can easily be created using the `lteRMCUL` function. `lteRMCUL` generates the various FRC configuration structures, as defined in TS 36.104 [2], Annex A.

You can specify `rmccfg` to include fields that are contained in the output structure, `rmccfgout`.

Data Types: `struct`

### `cqi` — CQI information bits

numeric vector

CQI information bits, specified as a numeric vector. CQI stands for channel quality information. `cqi` can be empty if these particular control information bits are not present in the transmission. `cqi` is not treated as a data stream, and thus each subframe contains the same CQI information bits.

Data Types: `double`

### `ri` — RI information bits

numeric vector

RI information bits, specified as a numeric vector. RI stands for rank indication. `ri` can be empty if these particular control information bits are not present in the transmission. `ri` is not treated as a data stream, and thus each subframe contains the same RI information bits.

Data Types: `double`

### `ack` — ACK information bits

numeric vector

ACK information bits, specified as a numeric vector. ACK stands for acknowledgment in automatic repeat request (ARQ) protocols. `ack` can be empty if these particular control information bits are not present in the transmission. `ack` is not treated as a data stream, and thus each subframe contains the same ACK information bits.

Data Types: `double`

## Output Arguments

### `waveform` — Generated RMC time-domain waveform

numeric matrix

Generated RMC time-domain waveform, returned as a *T*-by-*P* numeric matrix. *T* is the number of time-domain samples and *P* is the number of antennas.

`grid` is a 3-D array of resource elements for the generated subframes across all configured antenna ports, as described in “Represent Resource Grids”. `rmccfgout` is a structure containing information about the SC-FDMA modulated waveform and RMC configuration parameters.

Data Types: `double`

### `grid` — Populated resource grid

numeric 3-D array

Populated resource grid, returned as a numeric 3-D array of resource elements for several subframes across all configured antenna ports, as described in “Represent Resource Grids”.

`grid` represents the populated resource grid for all the physical channels specified in TS 36.104 [2], Annex A

Data Types: `double`
Complex Number Support: Yes

### `rmccfgout` — Configuration parameters

structure

#### Configuration Parameters Structure

Configuration parameters, returned as a structure. `rmccfgout` contains the following fields.

| Parameter Field | Values | Description |
|---|---|---|
| **`RC`** | `'A1-1'` (default), `'A1-2'`, `'A1-3'`, `'A1-4'`, `'A1-5'`, `'A2-1'`, `'A2-2'`, `'A2-3'`, `'A3-1'`, `'A3-2'`, `'A3-3'`, `'A3-4'`, `'A3-5'`, `'A3-6'`, `'A3-7'`, `'A4-1'`, `'A4-2'`, `'A4-3'`, `'A4-4'`, `'A4-5'`, `'A4-6'`, `'A4-7'`, `'A4-8'`, `'A5-1'`, `'A5-2'`, `'A5-3'`, `'A5-4'`, `'A5-5'`, `'A5-6'`, `'A5-7'`, `'A7-1'`, `'A7-2'`, `'A7-3'`, `'A7-4'`, `'A7-5'`, `'A7-6'`, `'A8-1'`, `'A8-2'`, `'A8-3'`, `'A8-4'`, `'A8-5'`, `'A8-6'`, `'A11-1'`, `'A3-2-9RB'`, `'A4-3-9RB'` | Reference channel number |
| **`NULRB`** | Scalar integer from 6 to 110 | Number of uplink resource blocks. ($N_{\mathrm{RB}}^{\mathrm{UL}}$) |
| **`NCellID`** | Integer from 0 to 503 | Physical layer cell identity |
| **`NFrame`** | 0 (default), nonnegative scalar integer | Frame number |
| **`NSubFrame`** | 0 (default), nonnegative scalar integer | Initial subframe number |
| **`CyclicPrefixUL`** | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **`CyclicShift`** | 0, 1, 2, 3, 4, 5, 6, 7 | Cyclic shift. This argument yields $n_{DMRS}^{(1)}$. |
| **`Shortened`** | 0 (default), 1 | Subframe shortened flag. If the function sets the flag to `1`, the last symbol of the subframe is not used. Subframes with possible SRS transmission require this flag to be set. |
| **`Hopping`** | `'Off'` (default), `'Group'`, or `'Sequence'` | Hopping type |
| **`SeqGroup`** | 0 (default), integer from 0 to 29 | PUSCH sequence group assignment ($\Delta_{\mathrm{SS}}$). |
| **`TotSubFrames`** | `10` (default)<br><br>Positive scalar integer | Total number of subframes to generate<br><br>This argument specifies the total number of subframes that form the resource grid. |

| Parameter Field | Values | Description |
|---|---|---|
| **RNTI** | 1 (default)<br><br>Scalar integer | Radio network temporary identifier (RNTI) value (16 bits) |
| **NTxAnts** | 1, 2, 4 | Number of transmission antennas. |
| **Windowing** | Nonnegative scalar integer | The number of time-domain samples over which windowing and overlapping of SC-FDMA symbols is applied |
| **DuplexMode** | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex<br><br>It represents the frame structure type. |
| **PUSCH** | Structure | PUSCH transmission configuration |
| **SRS** | Structure | Sounding Reference Signal (SRS) configuration |
| **SamplingRate** | Numeric scalar | Carrier sampling rate in Hz, $N_{SC}$ / $N_{SYM}$ × 3.84e6, where $N_{SC}$ is the number of subcarriers and $N_{SYM}$ is the number of SC-FDMA symbols in a subframe. |
| **Nfft** | Scalar integer, typically one of {128, 256, 512, 1024, 1536, 2048} for standard channel bandwidths {`'1.4MHz'`, `'3MHz'`, `'5MHz'`, `'10MHz'`, `'15MHz'`, `'20MHz'`}, respectively. | Number of FFT frequency bins |

#### PUSCH Substructure

The substructure `PUSCH` relates to the physical channel configuration and contains these fields:

| Parameter Field | Values | Description |
|---|---|---|
| **Modulation** | `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'` | Modulation format |
| **NLayers** | 1, 2, 3, 4 | Number of transmission layers. |
| **DynCyclicShift** | 0, 1, 2, 3, 4, 5, 6, 7 | Cyclic shift for DM-RS (yields $n_{DMRS}^{(2)}$). |
| **NBundled** | 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 | HARQ-ACK bundling scrambling sequence index |
| **BetaACK** | Scalar integer | Modulation and coding scheme (MCS) offset for HARQ-ACK bits |
| **BetaCQI** | Scalar integer | Modulation and coding scheme (MCS) offset for CQI and PMI bits |

| Parameter Field | Values | Description |
|---|---|---|
| **BetaRI** | Scalar integer | Modulation and coding scheme (MCS) offset for RI bits |
| **NHARQProcesses** | 1, 2, 3, 4, 5, 6, 7, 8 | Number of HARQ processes per component carrier |
| **RVSeq** | Numeric matrix | Redundancy version (RV) indicator used by all HARQ processes, returned as a numeric matrix. `RVSeq` is a one- or two-row matrix for one or two codewords, respectively. The number of columns in `RVSeq` equals the number of transmissions of the transport blocks associated with a HARQ process. The RV sequence specified in each column is applied to the transmission of the transport blocks. If `RVSeq` is a scalar (or column vector in the case of two codewords), then there is a single initial transmission of each block with no retransmissions. If `RVSeq` is a row vector in a two-codeword transmission, then the same RV sequence is applied to both codewords. |
| **RV** | Numeric matrix | Redundancy version (RV) indicator in initial subframe, returned as a numeric matrix. This argument is a one- or two-column vector that specifies the redundancy version for one or two codewords used in the initial subframe number, `NSubframe`. This parameter field is only for informational purposes and is read-only. |
| **NTurboDecIts** | Positive scalar integer | Number of turbo decoder iteration cycles |
| **OrthCover** | `'Off'` (default), `'On'` | Orthogonal cover sequence flag.<br><br>Applies (`'On'`), or does not apply (`'Off'`), orthogonal cover sequence *w* (*Activate-DMRS-with OCC*). |
| **PMI** | Integer from 0 to 23 | Scalar precoder matrix indication (PMI) to be used during precoding |
| **PRBSet** | Integer matrix | Physical resource block set of indices, returned as an integer matrix. This argument is a 1- or 2-column matrix that contains the 0-based physical resource block indices (PRBs) corresponding to the resource allocations for this PUSCH. |
| **TargetCodeRate** | Numeric scalar or vector | Target code rates for each subframe in a frame. Used for calculating the transport block sizes according to TS 36.101 [1], Annex A.2.1.2.<br><br>If `TargetCodeRate` is not provided and `TrBlkSizes` is provided at the input, `TargetCodeRate == ActualCodeRate`. |

| Parameter Field | Values | Description |
|---|---|---|
| **ActualCodeRate** | Numeric vector | Actual code rates for each subframe in a frame. The maximum actual code rate is 0.93. This parameter field is only for informational purposes and is read-only. |
| **TrBlkSizes** | Numeric vector | Transport block sizes for each subframe in a frame |
| **CodedTrBlkSizes** | Numeric vector | Coded transport block sizes for each a subframe in a frame, returned as a numeric vector. This parameter field is only for informational purposes and is read-only. |
| **HARQProcessSequence** | 1-by-$L_{\text{HARQ\_Seq}}$ integer vector. | One-based HARQ process indices for the internal HARQ scheduling sequence, based on same transport block size in all active subframes.<br><br>See footnote. |
| **1** When creating the HARQ process sequence, TTI bundling is considered. The length of the HARQ process sequence, $L_{\text{HARQ\_Seq}} = 10 \times \texttt{lcm}(\texttt{NHARQProcesses} \times \textit{ttiPerBundle}, \texttt{sum}(\textit{activesfs})) / \texttt{sum}(\textit{activesfs})$. The number of TTI per bundle, $\textit{ttiPerBundle} = 4$. The `sum`(*activesfs*) is the number of active subframes. For FDD, all subframes are active and for TDD, all uplink subframes are active. The uplink supports `NHARQProcesses` allowed by the standard and also the transport block sizes are the same for all the active subframes. | | |

#### SRS Substructure

The substructure `SRS` contains these fields:

| Parameter Field | Values | Description |
|---|---|---|
| **NTxAnts** | 1 (default), 2, 4 | Number of transmission antennas. |
| **BWConfig** | 0, 1, 2, 3, 4, 5, 6, 7 | Cell-specific SRS Bandwidth Configuration value ($C_{\text{SRS}}$) |
| **BW** | 0, 1, 2, 3 | UE-specific SRS Bandwidth value ($B_{\text{SRS}}$) |
| **ConfigIdx** | Integer from 0 to 644 | Configuration index ($I_{\text{SRS}}$) for UE-specific periodicity ($T_{\text{SRS}}$) and subframe offset ($T_{\text{offset}}$). |
| **TxComb** | 0 or 1 | Transmission comb. Controls SRS positions; SRS is transmitted in 6 carriers per resource block on odd (1) and even (0) resource indices. |
| **HoppingBW** | 0, 1, 2, 3 | SRS Frequency hopping configuration index ($b_{\text{hop}}$) |
| **FreqPosition** | Integer from 0 to 23 | Frequency domain position ($n_{\text{RRC}}$) |
| **CyclicShift** | 0 (default), integer from 0 to 7 | UE-specific cyclic shift ($n_{SRS}^{cs}$) |
| **SeqGroup** | 0 (default), integer from 0 to 29 | SRS sequence group number ($u$) |
| **SeqIdx** | 0 or 1 | Base sequence number ($v$) |

| Parameter Field | Values | Description |
|---|---|---|
| **`SubframeConfig`** | Integer from 0 to 15 | Sounding reference signal (SRS) subframe configuration |
| The following fields are present only when `DuplexMode` is set to `'TDD'`. | | |
| **`NF4RachPreambles`** | 0, 1, 2, 3, 4, 5, 6 | Number of RACH preamble frequency resources of Format 4 in *UpPTS* |
| **`OffsetIdx`** | 0 or 1 | Choice of SRS Subframe Offset in the case of 2 ms SRS periodicity. This parameter indexes the two SRS Subframe Offset entries in the row specified by the `ConfigIdx` parameter in table 8.2-2 of TS 36.213 for the SRS Configuration Index. |

## More About

### UL Reference Channel Options

Initialization choices available for the uplink reference channel and associated top-level configuration defaults include:

| Reference channels | Reference channels (continued) | Reference channels (continued) |
|---|---|---|
| A1-1 (6 RB, QPSK, R=1/3) | A4-3 (6 RB, 16QAM, R=3/4) | A7-5 (25 RB, 16QAM, R=3/4) |
| A1-2 (15 RB, QPSK, R=1/3) | A4-4 (15 RB, 16QAM, R=3/4) | A7-6 (25 RB, 16QAM, R=3/4) |
| A1-3 (25 RB, QPSK, R=1/3) | A4-5 (25 RB, 16QAM, R=3/4) | A8-1 (3 RB, QPSK, R=1/3) |
| A1-4 (3 RB, QPSK, R=1/3) | A4-6 (50 RB, 16QAM, R=3/4) | A8-2 (6 RB, QPSK, R=1/3) |
| A1-5 (9 RB, QPSK, R=1/3) | A4-7 (75 RB, 16QAM, R=3/4) | A8-3 (12 RB, QPSK, R=1/3) |
| A2-1 (6 RB, 16QAM, R=2/3) | A4-8 (100 RB, 16QAM, R=3/4) | A8-4 (25 RB, QPSK, R=1/3) |
| A2-2 (15 RB, 16QAM, R=2/3) | A5-1 (1 RB, 64QAM, R=5/6) | A8-5 (25 RB, QPSK, R=1/3) |
| A2-3 (25 RB, 16QAM, R=2/3) | A5-2 (6 RB, 64QAM, R=5/6) | A8-6 (25 RB, QPSK, R=1/3) |
| A3-1 (1 RB, QPSK, R=1/3) | A5-3 (15RB, 64QAM, R=5/6) | A11-1 (3 RB, QPSK, R=11/27) |
| A3-2 (6 RB, QPSK, R=1/3) | A5-4 (25 RB, 64QAM, R=5/6) | A17-1 (6 RB, 256QAM, R=5/6) |
| A3-3 (15 RB, QPSK, R=1/3) | A5-5 (50 RB, 64QAM, R=5/6) | A17-2 (15 RB, 256QAM, R=5/6) |
| A3-4 (25 RB, QPSK, R=1/3) | A5-6 (75 RB, 64QAM, R=5/6) | A17-3 (25 RB, 256QAM, R=5/6) |
| A3-5 (50 RB, QPSK, R=1/3) | A5-7 (100 RB, 64QAM, R=5/6) | A17-4 (50 RB, 256QAM, R=5/6) |
| A3-6 (75 RB, QPSK, R=1/3) | A7-1 (3 RB, 16QAM, R=3/4) | A17-5(75 RB, 256QAM, R=5/6) |
| A3-7 (100 RB, QPSK, R=1/3) | A7-2 (6 RB, 16QAM, R=3/4) | A17-6 (100 RB, 256QAM, R=5/6) |
| A4-1 (1 RB, 16QAM, R=3/4) | A7-3 (12 RB, 16QAM, R=3/4) | A3-2-9RB (9 RB, QPSK, R=1/3) |
| A4-2 (1 RB, 16QAM, R=3/4) | A7-4 (25 RB, 16QAM, R=3/4) | A4-3-9RB (9 RB, 16QAM, R=3/4) |

The fields in the output configuration structure, `rmccfgout`, are initialized in accordance with the reference channels defined in TS 36.104, Annex A.

- `'A3-2-9RB'` and `'A4-3-9RB'` are custom RMC configured for non-standard bandwidth but with the same code rate as the standardized version.
- `'A11-1'` enables TTI bundling and the corresponding HARQ pattern (enhanced HARQ pattern for FDD).

# Version History

**Introduced in R2014a**

**R2019b: This function now opens the LTE Waveform Generator app**
*Behavior changed in R2019b*

In previous releases, the input-free syntaxes of this function opened the **LTE Uplink RMC Generator** app. Starting in R2019b, input-free calls to this function open the **LTE Waveform Generator** app for an uplink RMC waveform.

## References

[1] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.104. “Evolved Universal Terrestrial Radio Access (E-UTRA); Base Station (BS) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[3] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

**Apps**
**LTE Waveform Generator**

**Functions**
`lteRMCUL` | `lteRMCDLTool` | `lteTestModelTool`

# lteRateMatchConvolutional

Convolutional rate matching

## Syntax

```
out = lteRateMatchConvolutional(in,outlen)
```

## Description

`out = lteRateMatchConvolutional(in,outlen)` rate matches the input data vector, `in`, to create an output vector, `out`, of length `outlen`. This function includes the stages of subblock interleaving, bit collection and bit selection, and pruning defined for convolutionally encoded data. For more information, see TS 36.212 [1], Section 5.1.4.2. The input data is assumed to comprise a concatenation of 3 subblocks, each of which is then interleaved prior to virtual circular buffer creation. No special processing is given to input filler bits.

## Examples

### Perform Convolutional Rate Matching

Perform convolutional rate matching of a coded block vector of length 132, with the output length set to 50.

```
rateMatched = lteRateMatchConvolutional(ones(132,1),50);
size(rateMatched)
```

```
ans = 1×2

    50     1
```

## Input Arguments

### `in` — Input data

numeric column vector

Input data, specified as a column vector. Input data is assumed to comprise a concatenation of 3 subblocks each of which is then interleaved prior to virtual circular buffer creation. No special processing is given to input filler bits.

Example: `ones(5,1)`

Data Types: `double` | `uint8` | `uint16` | `uint32` | `uint64` | `int8` | `int16` | `int32` | `int64`

### `outlen` — Output vector length

nonnegative scalar integer

Output vector length, specified as a nonnegative scalar integer.

Data Types: `double`

## Output Arguments

**`out` — Rate matched output**
numeric column vector

Rate matched output, returned as numeric column vector.

Data Types: `double` | `uint8` | `uint16` | `uint32` | `uint64` | `int8` | `int16` | `int32` | `int64`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteRateRecoverConvolutional` | `lteConvolutionalEncode` | `lteBCH` | `lteDCIEncode` | `lteRateMatchTurbo`

# lteRateMatchTurbo

Turbo rate matching

## Syntax

```
out = lteRateMatchTurbo(in,outlen,rv)
out = lteRateMatchTurbo(in,outlen,rv,chs)
```

## Description

`out = lteRateMatchTurbo(in,outlen,rv)` performs rate matching of input data `in` to create the output vector `out` with length `outlen`. The function performs subblock interleaving, bit collection and bit selection, and pruning defined for turbo encoded data. For more information, see Section 5.1.4.1 of [1].

The specified `rv` controls the redundancy version of the output. The bit selection stage assumes a quadrature phase-shift keying (QPSK) transmission mapped onto a single layer and no restriction on the number of soft bits, as in an uplink shared channel (UL-SCH).

`out = lteRateMatchTurbo(in,outlen,rv,chs)` allows additional control of the bit selection stage through selection of parameters for the soft buffer size and physical channel configuration in the `chs` input structure.

## Examples

### Perform Turbo Rate Matching

Rate match an input vector of 132 bits to a length of 100 with the RV parameter set to 0.

```
codedBlklen = 132;
invec = ones(codedBlklen,1);
outlen = 100;
rv = 0;

rmatched = lteRateMatchTurbo(invec,outlen,rv);
size(rmatched)
```

```
ans = 1×2

   100     1
```

## Input Arguments

### `in` — Input data

vector | cell array of vectors

Input data, specified as a vector or a cell array of vectors, assumed to be code blocks.

If you specify this input as a cell array, the function rate matches each vector separately and concatenates the results a single output vector `out`. The length of each nonempty input vector must be an integer multiple of 3.

The function treats negative values in this input as <NULL> filler buts inserted during code block segmentation, and does not process them during rate matching.

Example: `ones(132,1)`

Data Types: `double` | `cell`

**`outlen` — Output vector length**
nonnegative integer

Output vector length, specified as a nonnegative integer.

Data Types: `double`

**`rv` — Redundancy version control**
`0` | `1` | `2` | `3`

Redundancy version control, specified as `0`, `1`, `2`, or `3`.

Data Types: `double`

**`chs` — Channel transmission configuration**
structure

Channel transmission configuration, specified as a structure. It allows additional control of the bit selection stage through parameters for the soft buffer size and physical channel configuration.

For downlink turbo coded transport channels, you can control the soft buffer dimensions by including either `NIR` or the combined set of `NSoftbits`, `TxScheme`, and `DuplexMode`. If `DuplexMode` is `'TDD'`, also specify `TDDConfig`. If included, `NIR`, takes precedence for controlling the soft buffer dimensions. When neither of these optional `chs` fields (`NIR` or the set including `NSoftbits`) are present, the function assumes an uplink turbo coded transport channel and places no limit on the number of soft bits.

`chs` can contain the following fields.

**`Modulation` — Modulation scheme**
`'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'` | `'1024QAM'`

Modulation scheme, specified as `'QPSK'`, `'16QAM'`, `'64QAM'`, `'256QAM'`, or `'1024QAM'`.

Data Types: `char` | `string`

**`NLayers` — Number of transmission layers for transport block**
1 (default) | positive integer

Number of transmission layers for transport block, specified as a positive integer in the interval [1, 8]. Not necessary if `TxScheme` is set to `'Port0'`, `'TxDiversity'`, or `'Port5'`.

Data Types: `double`

**`TxScheme` — Transmission scheme**
`'Port0'` (default) | `'TxDiversity'` | `'CDD'` | `'SpatialMux'` | `'MultiUser'` | `'Port5'` | `'Port7-8'` | `'Port8'` | `'Port7-14'` | optional

PDSCH transmission scheme, specified as one of the following options.

| Transmission scheme | Description |
|---|---|
| `'Port0'` | Single antenna port, port 0 |
| `'TxDiversity'` | Transmit diversity |
| `'CDD'` | Large delay cyclic delay diversity scheme |
| `'SpatialMux'` | Closed loop spatial multiplexing |
| `'MultiUser'` | Multi-user MIMO |
| `'Port5'` | Single-antenna port, port 5 |
| `'Port7-8'` | Single-antenna port, port 7, when `NLayers` = 1. Dual layer transmission, ports 7 and 8, when `NLayers` = 2. |
| `'Port8'` | Single-antenna port, port 8 |
| `'Port7-14'` | Up to eight layer transmission, ports 7–14 |

Data Types: `char` | `string`

**`NIR` — Soft buffer size for entire input transport block**
nonnegative integer

Soft buffer size for entire input transport block, specified as a nonnegative integer.

Data Types: `double`

**`NSoftbits` — Total number of soft channel bits**
nonnegative integer

Total number of soft channel bits, specified as a nonnegative integer.

Data Types: `double`

**`DuplexMode` — Duplex mode**
`'FDD'` (default) | optional | `'TDD'`

Duplex mode, specified as `'FDD'` or `'TDD'`.

Data Types: `char` | `string`

**`TDDConfig` — Uplink or downlink configuration**
0 (default) | optional | nonnegative integer

Uplink or downlink configuration, specified as a nonnegative scalar integer in the interval [0, 6]. Optional. Only required if `DuplexMode` is set to `'TDD'`.

Data Types: `double`

Data Types: `struct`

## Output Arguments

**`out` — Turbo rate matched output**
numeric column vector

Turbo rate matched output, returned as a numeric column vector.

Data Types: `double` | `uint8` | `uint16` | `uint32` | `uint64` | `int8` | `int16` | `int32` | `int64`

# Version History

**Introduced in R2013b**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteRateRecoverTurbo` | `lteTurboEncode` | `lteDLSCH` | `lteULSCH` | `lteRateMatchConvolutional`

# lteRateRecoverConvolutional

Convolutional rate matching recovery

## Syntax

```
out = lteRateRecoverConvolutional(in,outlen)
```

## Description

`out = lteRateRecoverConvolutional(in,outlen)` performs rate recovery of the input data vector, `in`, to create an output vector, `out`, of length `outlen`. This function is the inverse of the rate matching operation for convolutionally encoded data. For more information, see `lteRateMatchConvolutional`. This function includes the inverses of the subblock interleaving, bit collection and bit selection, and pruning stages. This function also implements additive soft combining of the input data elements in the case where repetition occurred during the original rate matching.

## Examples

### Perform Convolutional Rate Recovery

Perform rate recovery after rate matching. The returned vector has the same length as the input to rate matching.

```
codedBlklen = 132;
rateMatched = lteRateMatchConvolutional(ones(codedBlklen ,1),50);
rateRecovered = lteRateRecoverConvolutional(rateMatched,codedBlklen);
size(rateRecovered)
```

```
ans = 1×2

   132     1
```

The output variable, `rateRecovered`, is a vector of the same length as the input to rate matching.

## Input Arguments

### `in` — Input data

numeric column vector

Input data, specified as a numeric column vector.

Data Types: `double` | `uint8` | `uint16` | `uint32` | `uint64` | `int8` | `int16` | `int32` | `int64`

### `outlen` — Output vector length

nonnegative scalar integer

Output vector length, specified as a nonnegative scalar integer.

Example: 50

Data Types: `double`

## Output Arguments

**`out` — Rate recovered output**
numeric column vector

Rate recovered output, returned as a numeric column vector.

Data Types: `double` | `uint8` | `uint16` | `uint32` | `uint64` | `int8` | `int16` | `int32` | `int64`

# Version History

**Introduced in R2014a**

## See Also

`lteRateMatchConvolutional` | `lteConvolutionalDecode` | `lteBCHDecode` | `lteDCIDecode` | `lteRateRecoverTurbo`

# lteRateRecoverTurbo

Turbo rate recovery

## Syntax

```
out = lteRateRecoverTurbo(in,trblklen,rv)
out = lteRateRecoverTurbo(in,trblklen,rv,chs,cbsbuffers)
```

## Description

`out = lteRateRecoverTurbo(in,trblklen,rv)` performs rate recovery of the input vector, `in`, creating a cell array of vectors, `out`. `out` represents the turbo encoded code blocks before concatenation. This function is the inverse of the rate matching operation for turbo encoded data. For more information, see `lteRateMatchTurbo` and TS 36.212, Section 5.1.4.1 [1]. This function includes the inverses of the subblock interleaving, bit collection, and bit selection and pruning stages. The dimensions of `out` are deduced from `trblklen`, which represents the length of the original encoded transport block. This parameterization is required to recover the original number of code blocks, their encoded lengths, and the locations of any filler bits. The parameter `rv` controls the redundancy version of the output. The bit selection recovery assumes a QPSK transmission mapped onto a single layer. It also assumes no restriction on the number of soft bits, as in an uplink UL-SCH transport channel.

`out = lteRateRecoverTurbo(in,trblklen,rv,chs,cbsbuffers)` specifies two additional inputs. The `chs` input structure allows additional control of the bit selection recovery stage through parameters for the soft buffer size and physical channel configuration. The `cbsbuffers` input allows combining with pre-existing soft information for the HARQ process.

## Examples

### Perform Turbo Rate Recovery

Create a codeword from a transport block then rate recover the codeword back into a set of coded code blocks. The transport block is originally segmented into a single code block so the `rateRecovered` output variable is a cell array containing a single turbo coded code block.

Define the transport block length prior to CRC and turbo coding, code word length, redundancy version, and CRC polynomial. Use these parameters to perform coding operations.

```
trBlkLen = 135;
codewordLen = 450;
rv = 0;
crcPoly = '24A';

trblockwithcrc = lteCRCEncode(zeros(trBlkLen,1),crcPoly);
codeblocks = lteCodeBlockSegment(trblockwithcrc);
turbocodedblocks = lteTurboEncode(codeblocks);
codeword = lteRateMatchTurbo(turbocodedblocks,codewordLen,rv);
rateRecovered = lteRateRecoverTurbo(codeword,trBlkLen,rv)
```

```
rateRecovered = 1x1 cell array
    {492x1 int8}
```

`rateRecovered` is a cell array with a single coded code block of size indicated above.

Further turbo decoding, desegmentation and CRC decoding of `rateRecovered` would result in a decoded transport block of length equal to the original transport block. Note that the `trBlkLen` parameter of the `lteRateRecoverTurbo` function is the transport block length before CRC and turbo coding, not the length after turbo coding or rate matching.

## Input Arguments

### `in` — Input data

numeric vector

Input data, specified as a numeric vector.

Data Types: `double`

### `trblklen` — Length of original encoded transport block before encoding

numeric value

Length of the original encoded transport block before encoding, specified as a numeric value.

Data Types: `double`

### `rv` — Redundancy version used to recover data

0 | 1 | 2 | 3

Redundancy version used to recover data, specified as 0, 1, 2, or 3.

Data Types: `double`

### `chs` — Channel transmission configuration

structure

Channel transmission configuration, specified as a structure. It allows additional control of the bit selection stage through parameters for the soft buffer size and physical channel configuration.

For downlink turbo coded transport channels, you can control the soft buffer dimensions by including either `NIR` or the combined set of `NSoftbits`, `TxScheme`, and `DuplexMode`. If `DuplexMode` is `'TDD'`, also specify `TDDConfig`. If included, `NIR`, takes precedence for controlling the soft buffer dimensions. When neither of these optional `chs` fields (`NIR` or the set including `NSoftbits`) are present, the function assumes an uplink turbo coded transport channel and places no limit on the number of soft bits.

`chs` can contain the following fields.

#### `Modulation` — Modulation scheme

`'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'` | `'1024QAM'`

Modulation scheme, specified as `'QPSK'`, `'16QAM'`, `'64QAM'`, `'256QAM'`, or `'1024QAM'`.

Data Types: `char` | `string`

**`NLayers` — Number of transmission layers for transport block**
1 (default) | 2 | 3 | 4

Number of transmission layers for transport block, specified as 1 (default), 2, 3, or 4. Not necessary if `TxScheme` is set to `'Port0'`, `'TxDiversity'`, or `'Port5'`.

Data Types: `double`

**`TxScheme` — Transmission scheme**
`'Port0'` (default) | optional | `'TxDiversity'` | `'CDD'` | `'SpatialMux'` | `'MultiUser'` | `'Port5'` | `'Port7-8'` | `'Port8'` | `'Port7-14'`

PDSCH transmission scheme, specified as one of the following options.

| Transmission scheme | Description |
|---|---|
| `'Port0'` | Single antenna port, port 0 |
| `'TxDiversity'` | Transmit diversity |
| `'CDD'` | Large delay cyclic delay diversity scheme |
| `'SpatialMux'` | Closed loop spatial multiplexing |
| `'MultiUser'` | Multi-user MIMO |
| `'Port5'` | Single-antenna port, port 5 |
| `'Port7-8'` | Single-antenna port, port 7, when `NLayers` = 1. Dual layer transmission, ports 7 and 8, when `NLayers` = 2. |
| `'Port8'` | Single-antenna port, port 8 |
| `'Port7-14'` | Up to eight layer transmission, ports 7–14 |

Data Types: `char` | `string`

**`NIR` — Soft buffer size for entire input transport block**
nonnegative integer

Soft buffer size for entire input transport block, specified as a nonnegative integer.

Data Types: `double`

**`NSoftbits` — Total number of soft channel bits**
nonnegative integer

Total number of soft channel bits, specified as a nonnegative integer.

Data Types: `double`

**`DuplexMode` — Duplex mode**
`'FDD'` (default) | optional | `'TDD'`

Duplex mode, specified as `'FDD'` or `'TDD'`.

Data Types: `char` | `string`

**`TDDConfig` — Uplink or downlink configuration**
0 (default) | optional | nonnegative scalar integer (0...6)

Uplink or downlink configuration, specified as a nonnegative scalar integer from 0 through 6. Optional. Only required if `DuplexMode` is set to `'TDD'`.

Data Types: `double`

Data Types: `struct`

**`cbsbuffers` — Code block soft information buffers**
cell array of numeric vectors | empty cell array | cell array of numeric scalar elements

Code block soft information buffers, specified as a cell array. This input argument represents any pre-existing code block-oriented soft information to be additively combined with the recovered turbo encoded code blocks. It allows the direct soft combining of consecutive HARQ retransmissions and is typically returned by a previous call to the function to recover an earlier transmission of the same transport block. The `cbsbuffers` cell array either:

- dimensionally matches the output code blocks, `out`
- can be empty to represent the processing of an initial HARQ transmission
- or can be scalar to add a constant offset to all the deinterleaved soft data in a code block.

Data Types: `cell`

## Output Arguments

**`out` — Turbo encoded code blocks before concatenation**
cell array of numeric column vectors

Turbo encoded code blocks before concatenation, returned as a cell array of numeric column vectors. The dimensions of `out` are deduced from `trblklen`, which represents the length of the original encoded transport block.

Data Types: `cell`

# Version History

**Introduced in R2013b**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteRateMatchTurbo` | `lteTurboDecode` | `lteDLSCHInfo` | `lteULSCHInfo` | `lteDLSCHDecode` | `lteULSCHDecode` | `lteRateRecoverConvolutional`

# lteResourceGrid

Subframe resource array

## Syntax

```
grid = lteResourceGrid(cfg)
grid = lteResourceGrid(cfg,p)
```

## Description

`grid = lteResourceGrid(cfg)` returns an empty resource array generated from the configuration settings structure, `cfg`. To create a resource array specifically for downlink or uplink, use `lteDLResourceGrid` or `lteULResourceGrid`, respectively.

For more information on the resource grid and the multidimensional array used to represent the resource elements for one subframe across all configured antenna ports, see “Represent Resource Grids”.

`grid = lteResourceGrid(cfg,p)` returns a resource array, where `p` directly specifies the number of antenna planes in the array.

## Examples

### Create Downlink Subframe Resource Array

Create an empty resource array that represents the downlink resource elements for 10MHz bandwidth, one subframe, and two antennas.

```
griddl = lteResourceGrid(struct('NDLRB',50,'CellRefP',2,'CyclicPrefix','Normal'));
size(griddl)
```

```
ans = 1×3

   600    14     2
```

### Create Uplink Subframe Resource Array

Create an empty resource array that represents the uplink resource elements for 10MHz bandwidth, one subframe, and two antennas.

```
gridul = lteResourceGrid(struct('NULRB',50,'NTxAnts',2,'CyclicPrefixUL','Normal'));
size(gridul)
```

```
ans = 1×3

   600    14     2
```

### Create Downlink Subframe Resource Array Using Optional Antenna Plane Input

Create an empty resource array that represents the downlink resource elements for 20 MHz bandwidth, one subframe, extended cyclic prefix, and four antennas planes.

```
cfg = struct('NDLRB',100,'CyclicPrefix','Extended');
p = 4;
griddl = lteResourceGrid(cfg,p);
size(griddl)
```

```
ans = 1×3

        1200          12           4
```

## Input Arguments

### `cfg` — Configuration settings

scalar structure

Configuration settings, specified as a scalar structure. To create a downlink resource array, `cfg` must contain the `NDLRB` and `CellRefP` fields. To create an uplink resource array, `cfg` must contain the `NULRB` field. The presence of field `NDLRB` takes precedence over the field `NULRB`.

For the downlink, these fields are applicable.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NDLRB`** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **`CellRefP`** | Required | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **`CyclicPrefix`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |

For the uplink, these fields are applicable.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NULRB`** | Required | scalar integer from 6 to 110 | Number of uplink resource blocks. ($N_{RB}^{UL}$) |
| **`CyclicPrefixUL`** | Optional | `'Normal'` (default), `'Extended'` | Current cyclic prefix length |
| **`NTxAnts`** | Optional | 1 (default), 2, 4 | Number of transmission antennas. |

### `p` — Number of antenna planes in the output array

nonnegative scalar integer

Number of antenna planes in the output array, specified as a nonnegative scalar integer.

Data Types: `double`

## Output Arguments

### `grid` — Empty multidimensional resource grid

3-D numeric array

Empty multidimensional resource grid, returned as an *N*-by-*M*-by-*P* numeric array. *N* is the number of subcarriers (12×`NULRB`). *M* is the number of OFDM or SC-FDMA symbols in a subframe, 14 for normal cyclic prefix and 12 for extended cyclic prefix. *P* is the number of transmit antenna ports, `cfg.CellRefP` in the downlink and `cfg.NTxAnts` in the uplink.

Data Types: `double`

# Version History

**Introduced in R2014a**

## See Also

`lteResourceGridSize` | `lteDLResourceGrid` | `lteSLResourceGrid` | `lteULResourceGrid` | `lteOFDMModulate` | `lteSCFDMAModulate`

# lteResourceGridSize

Size of subframe resource array

## Syntax

```
d = lteResourceGridSize(cfg)
d = lteResourceGridSize(cfg,p)
```

## Description

`d = lteResourceGridSize(cfg)` returns a three-element row vector of dimension lengths for the resource array generated from the settings structure, `cfg`. To get the dimension lengths specifically for a downlink or uplink resource array, use the function `lteDLResourceGridSize` or `lteULResourceGridSize` respectively. For more information on the resource grid and the multidimensional array used to represent the resource elements for one subframe across all configured antenna ports, see “Represent Resource Grids”.

`d = lteResourceGridSize(cfg,p)` returns a three-element row vector, where `p` directly specifies the number of antenna planes in the array.

## Examples

### Get Downlink Subframe Resource Array Size

Get the downlink subframe resource array size from a downlink configuration structure. Then, use the returned vector to directly create a MATLAB™ array.

```
cfgdl = struct('NDLRB',6,'CellRefP',2,'CyclicPrefix','Normal');
griddl = zeros(lteResourceGridSize(cfgdl));
size(griddl)
```

```
ans = 1×3

    72    14     2
```

The output grid, `griddl`, is a resource array. This resource array could be obtained in a similar manner using the `lteResourceGrid` function.

### Get Uplink Subframe Resource Array Size

Configure UE-specific settings.

```
cfgul = struct(NULRB=6,NTxAnts=2,CyclicPrefixUL="Normal");
```

Get the uplink subframe resource array size.

```
d = lteResourceGridSize(cfgul);
```

Generate an uplink resource array of the appropriate size.

```
gridul = zeros(d);
```

### Get Uplink Subframe Resource Array Size for Specified Antenna Planes

Configure UE-specific settings.

```
ue = struct(NULRB=25,CyclicPrefixUL="Normal");
```

Get the uplink subframe resource array size for the specified configuration and four antenna planes.

```
p = 4;
d = lteResourceGridSize(ue,p);
```

Create a resource array of the appropriate size.

```
gridul = zeros(d);
```

## Input Arguments

### `cfg` — Configuration settings

scalar structure

Configuration settings, specified as a scalar structure. To create a downlink resource array, `cfg` must contain the `NDLRB` and `CellRefP` fields. To create an uplink resource array, `cfg` must contain the `NULRB` field. If both `NDLRB` and `NULRB` fields are defined, the presence of the field `NDLRB` takes precedence over the field `NULRB`.

For the downlink, these fields are applicable.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NDLRB`** | Required | Scalar integer from 6 to 110 | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **`CellRefP`** | Required | 1, 2, 4 | Number of cell-specific reference signal (CRS) antenna ports |
| **`CyclicPrefix`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |

For the uplink, these fields are applicable.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NULRB`** | Required | scalar integer from 6 to 110 | Number of uplink resource blocks. ($N_{RB}^{UL}$) |
| **`CyclicPrefixUL`** | Optional | `'Normal'` (default), `'Extended'` | Current cyclic prefix length |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| `NTxAnts` | Optional | 1 (default), 2, 4 | Number of transmission antennas. |

**p — Number of antenna planes**
positive scalar integer

Number of antenna planes, specified as a positive scalar integer.

Data Types: `double`

## Output Arguments

**d — Dimension lengths of resource grid**
numeric vector

Dimension lengths, returned as a three-element row vector [*N M P*]. *N* is the number of subcarriers (12×`NULRB`). *M* is the number of OFDM or SC-FDMA symbols in a subframe, 14 for normal cyclic prefix and 12 for extended cyclic prefix. *P* is the number of transmit antenna ports, `cfg.CellRefP` in the downlink and `cfg.NTxAnts` in the uplink.

Data Types: `double`

# Version History

**Introduced in R2014a**

## See Also

`lteResourceGrid` | `lteDLResourceGridSize` | `lteULResourceGridSize`

# lteSCFDMADemodulate

Demodulate using SC-FDMA

## Syntax

```
grid = lteSCFDMADemodulate(ue,waveform)
grid = lteSCFDMADemodulate(ue,waveform,cpfraction)

grid = lteSCFDMADemodulate(ue,chs,waveform)
grid = lteSCFDMADemodulate(ue,chs,waveform,cpfraction)
```

## Description

`grid = lteSCFDMADemodulate(ue,waveform)` returns the resource array `grid` by performing single-carrier frequency-division multiple access (SC-FDMA) demodulation of `waveform`, the time-domain waveform, for user-equipment-specific (UE-specific) settings `ue`. You can use this syntax for LTE demodulation or NB-IoT multitone demodulation, depending on the fields you specify in `ue`.

The demodulation calculates one fast Fourier transform (FFT) operation per received SC-FDMA symbol. It recovers the received subcarrier values, which are then used to construct each column of `grid`. The FFT is positioned partway through the cyclic prefix to allow for a degree of channel delay spread while avoiding the overlap between adjacent orthogonal frequency-division multiplexing (OFDM) symbols. The input FFT is also shifted by half a subcarrier. The particular position of the FFT chosen here avoids the SC-FDMA symbol overlapping used in the `lteSCFDMAModulate` function. Since the FFT is performed away from the original zero-phase point on the transmitted subcarriers, a phase correction is applied to each subcarrier after the FFT.

`grid = lteSCFDMADemodulate(ue,waveform,cpfraction)` performs SC-FDMA demodulation of the input waveform with the specified position of demodulation with respect to cyclic prefix `cpfraction`.

`grid = lteSCFDMADemodulate(ue,chs,waveform)` performs SC-FDMA demodulation of the input waveform and the narrowband PUSCH (NPUSCH) information specified by `chs`. You can use this syntax for LTE, single-tone NB-IoT, and multitone NB-IoT configurations. When you use this syntax without configuring `ue` for NB-IoT, the function ignores `chs`.

`grid = lteSCFDMADemodulate(ue,chs,waveform,cpfraction)` performs SC-FDMA demodulation of the waveform for the specified NPUSCH information and demodulation position. You can use this syntax for LTE, single-tone NB-IoT, and multitone NB-IoT configurations. When you use this syntax without configuring `ue` for NB-IoT, the function ignores `chs`.

## Examples

### Perform SC-FDMA Demodulation

Perform SC-FDMA demodulation of uplink fixed reference channel (FRC) A3-2.

Initialize UE-specific settings as the fixed reference channel A3-2 by specifying the relevant configuration in the `lteRMCUL` function.

```
ue = lteRMCUL('A3-2');
```

Specify the waveform to be demodulated by using the `lteRMCULTool` function. Get the resource array by performing SC-FDMA demodulation.

```
waveform = lteRMCULTool(ue,randi([0,1],ue.PUSCH.TrBlkSizes(1),1));
grid = lteSCFDMADemodulate(ue,waveform);
```

### Perform NB-IoT Uplink Single-Tone SC-FDMA Demodulation

Perform NB-IoT uplink single-tone SC-FDMA demodulation with 3.75-kHz subcarrier spacing.

Specify the number of slots for waveform generation.

```
NSlots = 10;
```

Initialize the required cell-wide settings by specifying the NB-IoT subcarrier spacing as a field in the structure `ue`.

```
ue.NBULSubcarrierSpacing = '3.75kHz';
```

Specify the zero-based NB-IoT subcarrier indices field as a scalar, indicating single-tone SC-FDMA demodulation. This configuration requires specification of the modulation type, number of slots per resource unit (RU), number of RUs, and number of repetitions for a codeword.

```
chs.NBULSubcarrierSet = 41;    % Indicate single-tone demodulation
chs.Modulation = 'BPSK';       % Specify modulation type as BPSK
chs.NULSlots = 4;              % Set four slots per RU
chs.NRU = 1;                   % Specify one RU
chs.NRep = 4;                  % Repeat codeword four times
```

Set the input grid to start from the third time slot in an NPUSCH bundle.

```
chs.SlotIdx = 2;
```

Specify a random array of bits and map the values to BPSK-modulated symbols by using the `lteSymbolModulate` function. Perform uplink precoding on the modulated symbols by using the `lteULPrecode` function.

```
bits = randi([0,1],7*NSlots,1);
symbols =  lteSymbolModulate(bits,chs.Modulation);
precodedSymbols = lteULPrecode(symbols,1,'Subcarrier');
```

Initialize the resource element (RE) grid using the precoded symbols.

```
reGrid = zeros(48,7*NSlots);
NSC = length(chs.NBULSubcarrierSet);
reGrid(chs.NBULSubcarrierSet+1,:) = reshape(precodedSymbols,NSC,[]);
```

Generate the time-domain waveform for demodulation by using the `lteSCFDMAModulate` function.

```
[waveform,info] = lteSCFDMAModulate(ue,chs,reGrid);
```

Get the resource array by performing SC-FDMA demodulation on the waveform.

```
grid = lteSCFDMADemodulate(ue,chs,waveform);
```

### Perform NB-IoT Uplink Multitone SC-FDMA Demodulation

Perform NB-IoT uplink multitone SC-FDMA demodulation with 15 kHz subcarrier spacing for a chosen cyclic prefix fraction.

Specify the number of slots for waveform generation.

```
NSlots = 10;
```

Initialize the required cell-wide settings by specifying the NB-IoT subcarrier spacing as a field in the structure `ue`.

```
ue.NBULSubcarrierSpacing = '15kHz';
```

Specify the zero-based NB-IoT subcarrier indices field as a vector, indicating multitone SC-FDMA demodulation, and the modulation type.

```
chs.NBULSubcarrierSet = 0:2;    % Indicate multitone demodulation
chs.Modulation = 'QPSK';         % Specify modulation type as QPSK
```

Specify a random array of bits and map the values to QPSK-modulated symbols by using the `lteSymbolModulate` function. Perform uplink precoding on the modulated symbols by using the `lteULPrecode` function.

```
bits = randi([0,1],7*NSlots*2*length(chs.NBULSubcarrierSet),1);
symbols =  lteSymbolModulate(bits,chs.Modulation);
precodedSymbols = lteULPrecode(symbols,length(chs.NBULSubcarrierSet),'Subcarrier');
```

Initialize the RE grid using the precoded symbols.

```
grid = repmat(lteNBResourceGrid(ue),1,NSlots);
NSC = length(chs.NBULSubcarrierSet);
grid(chs.NBULSubcarrierSet+1,:) = reshape(precodedSymbols,NSC,[]);
```

Generate the time-domain waveform for demodulation by using the `lteSCFDMAModulate` function.

```
[waveform,info] = lteSCFDMAModulate(ue,chs,grid);
```

Specify the cyclic prefix fraction and get the resource array by performing SC-FDMA demodulation on the waveform.

```
cpfraction = 0.3;
grid = lteSCFDMADemodulate(ue,chs,waveform,cpfraction);
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific settings, specified as a structure. The fields you specify in `ue` and `chs` determine whether the function performs SC-FDMA demodulation for an LTE or NB-IoT configuration. To choose an LTE configuration, specify the `NULRB` field. To choose an NB-IoT configuration, specify the

NBULSubcarrierSpacing field. The CyclicPrefixUL field is optional and is applicable only for an LTE configuration.

**NULRB — Number of uplink resource blocks**
integer in the interval [6, 110]

Number of uplink resource blocks, $N_{RB}^{UL}$, specified as an integer in the interval [6, 110]. To return SC-FDMA modulation information for an LTE configuration, you must specify this field.

Data Types: double

**CyclicPrefixUL — Cyclic prefix length**
'Normal' (default) | 'Extended'

Cyclic prefix length, specified as 'Normal' or 'Extended'. This field is optional.

**Dependencies**

This field applies only when you choose an LTE configuration by specifying the NULRB field.

Data Types: char | string

**NBULSubcarrierSpacing — NB-IoT subcarrier spacing**
'3.75kHz' | '15kHz'

NB-IoT subcarrier spacing, specified as '3.75kHz' or '15kHz'. To set a subcarrier spacing of 3.75 kHz, specify NBULSubcarrierSpacing as '3.75kHz'. To set a subcarrier spacing of 15 kHz, specify NBULSubcarrierSpacing as '15kHz'.

To use lteSCFDMADemodulate for NB-IoT demodulation, you must specify this field. To indicate an LTE configuration, omit this field.

**Note** For a subcarrier spacing of 3.75 kHz, lteSCFDMADemodulate supports only single-tone NB-IoT configurations.

Data Types: char | string

Data Types: struct

**chs — NPUSCH information**
structure

NPUSCH information, specified as a structure. For an NB-IoT configuration, you can set additional uplink-specific parameters by specifying the NB-IoT-specific fields in chs. Except for the NBULSubcarrierSet field, the fields in chs are applicable either when NBULSubcarrierSpacing is '3.75kHz' or when ue.NBULSubcarrierSpacing is '15kHz' and length(chs.NBULSubcarrierSet) is 1.

**NBULSubcarrierSet — NB-IoT uplink subcarrier indices**
vector of nonnegative integers (default) | nonnegative integer

NB-IoT uplink subcarrier indices, specified as a vector of nonnegative integers in the interval [0, 11] or a nonnegative integer in the interval [0, 47]. The indices are in zero-based form. To use lteSCFDMADemodulate for single-tone NB-IoT demodulation, you must specify

`NBULSubcarrierSet` as a scalar. If you do not specify `NBULSubcarrierSet`, `lteSCFDMADemodulate` performs multi-tone NB-IoT demodulation by default. If you specify the `NBULSubcarrierSpacing` field as `'15kHz'`, this field is required.

Data Types: `double`

**`Modulation` — Modulation type**
`'BPSK'` | `'QPSK'`

Modulation type, specified as `'BPSK'` or `'QPSK'`. For binary phase shift keying (BPSK), specify `Modulation` as `'BPSK'`. For quadrature phase shift keying (QPSK), specify `Modulation` as `'QPSK'`.

Data Types: `char` | `string`

**`NULSlots` — Number of slots per resource unit**
positive integer

Number of slots per resource unit (RU), specified as a positive integer. To use `lteSCFDMADemodulate` for single-tone NB-IoT demodulation, you must specify this field.

Data Types: `double`

**`NRU` — Number of RUs**
positive integer

Number of RUs, specified as a positive integer. To use `lteSCFDMADemodulate` for single-tone NB-IoT demodulation, you must specify this field.

Data Types: `double`

**`NRep` — Number of repetitions for codeword**
nonnegative integer

Number of repetitions for a codeword, specified as a nonnegative integer. To use `lteSCFDMADemodulate` for single-tone NB-IoT demodulation, you must specify this field.

Data Types: `double`

**`SlotIdx` — Relative slot index in NPUSCH bundle**
`0` (default) | nonnegative integer

Relative slot index in an NPUSCH bundle, specified as a nonnegative integer. This field determines the zero-based relative slot index in a bundle of time slots for transmission of a transport block or control information bit.

Data Types: `double`

Data Types: `struct`

**`waveform` — Time-domain waveform**
complex-valued matrix

Time-domain waveform, specified as a complex-valued matrix. The sampling rate of `waveform` must be the same as used in the `lteSCFDMAModulate` modulator function for the number of resource blocks specified in the `NULRB` field of `ue`. The waveform must be time-aligned such that the first sample is the first sample of the cyclic prefix of the first SC-FDMA symbol in a subframe.

Data Types: `double`

Complex Number Support: Yes

**`cpfraction` — Cyclic prefix fraction**
scalar in the interval [0, 1]

Cyclic prefix fraction, specified as a scalar in the interval [0, 1]. This argument specifies the position of the demodulation with respect to the cyclic prefix. A value of `0` represents the start of the cyclic prefix. A value of `1` represents the end of the cyclic prefix.

- For LTE demodulation, the default value is `0.55`.
- For NB-IoT demodulation with 3.75-kHz subcarrier spacing, the default value is `0.18`.
- For NB-IoT demodulation with 15-kHz subcarrier spacing, the default value is `0.22`.

The default value allows for the default level of windowing in the `lteSCFDMAModulate` function.

Data Types: `double`

## Output Arguments

**`grid` — Output resource array**
complex-valued matrix

Output resource array, returned as a complex-valued matrix.

Data Types: `double`
Complex Number Support: Yes

# Version History

**Introduced in R2014a**

## See Also

`lteSCFDMAModulate` | `lteSCFDMAInfo` | `lteULFrameOffset` | `lteULFrameOffsetPUCCH1` | `lteULFrameOffsetPUCCH2` | `lteULFrameOffsetPUCCH3` | `lteULChannelEstimate` | `lteULPerfectChannelEstimate` | `lteULChannelEstimatePUCCH1` | `lteULChannelEstimatePUCCH2` | `lteULChannelEstimatePUCCH3`

# lteSCFDMAModulate

Modulate using SC-FDMA

## Syntax

```
[waveform,info] = lteSCFDMAModulate(ue,grid)
[waveform,info] = lteSCFDMAModulate(ue,grid,windowing)

[waveform,info] = lteSCFDMAModulate(ue,chs,grid)
[waveform,info] = lteSCFDMAModulate(ue,chs,grid,windowing)
```

## Description

`[waveform,info] = lteSCFDMAModulate(ue,grid)` performs single-carrier frequency-division multiple access (SC-FDMA) modulation for user-equipment-specific (UE-specific) settings `ue`. The function returns `waveform`, an SC-FDMA-modulated waveform, and its corresponding information `info`. You can use this syntax for LTE and multitone narrowband Internet of Things (NB-IoT) configurations.

The function calculates the inverse fast Fourier transform (IFFT), half-subcarrier shifts, and cyclic prefix insertions. The function also optionally performs raised-cosine windowing and overlapping of adjacent SC-FDMA symbols in resource array `grid`. For a block diagram that illustrates the steps in SC-FDMA modulation, see “Algorithms” on page 2-1033.

`[waveform,info] = lteSCFDMAModulate(ue,grid,windowing)` performs SC-FDMA modulation for the chosen number of windowed and overlapped samples, `windowing`, used in the time-domain windowing. If you specify the `Windowing` field in `ue`, the function ignores it, and the output `Windowing` field of `info` is as specified in `windowing`. You can use this syntax for LTE and multitone NB-IoT configurations.

`[waveform,info] = lteSCFDMAModulate(ue,chs,grid)` performs SC-FDMA modulation for channel transmission configuration `chs`. You can use this syntax for LTE, single-tone NB-IoT, and multitone NB-IoT configurations. When you use this syntax without configuring `ue` for NB-IoT, the function ignores `chs`.

`[waveform,info] = lteSCFDMAModulate(ue,chs,grid,windowing)` performs SC-FDMA modulation for the specified channel transmission configuration and number of windowed and overlapped samples. You can use this syntax for LTE, single-tone NB-IoT, and multitone NB-IoT configurations. When you use this syntax without configuring `ue` for NB-IoT, the function ignores `chs`.

## Examples

### Perform SC-FDMA Modulation

Perform SC-FDMA modulation of one subframe of random uniformly distributed noise.

Initialize UE-specific settings for the specified number of resource blocks.

```
ue = struct('NULRB',50);
```

Obtain the size of the resource array.

```
d = lteULResourceGridSize(ue);
```

Get the resource grid by mapping a randomly generated vector of bits to the relevant modulation symbols, specifying QPSK modulation.

```
grid = reshape(lteSymbolModulate(randi([0,1],prod(d)*2,1),'QPSK'),d);
```

Perform SC-FDMA modulation for the specified UE-specific settings and resource grid.

```
waveform = lteSCFDMAModulate(ue,grid);
```

### Perform SC-FDMA Modulation for Multitone NB-IoT with Windowing

Perform SC-FDMA modulation of ten time slots of uniformly distributed noise, specifying a multitone NB-IoT downlink configuration and a windowing value.

Initialize the UE-specific settings by specifying the NB-IoT uplink subcarrier spacing.

```
ue.NBULSubcarrierSpacing = '15kHz';
```

Get the resource grid for the specified number of time slots.

```
NSlots = 10; % Number of slots in the generated waveform
dims = [12 7*NSlots];
grid = reshape(lteSymbolModulate(randi([0,1],prod(dims)*2,1),'QPSK'),dims);
```

Specify a windowing value of 6.

```
windowing = 6;
```

Perform SC-FDMA modulation and display the first five symbols of the modulated waveform.

```
waveform = lteSCFDMAModulate(ue,grid,windowing);
disp(waveform(1:5));
```

```
   0.0152 + 0.0178i
   0.0126 + 0.0159i
   0.0092 + 0.0130i
   0.0052 + 0.0092i
   0.0006 + 0.0047i
```

### Perform SC-FDMA Modulation for Multitone NB-IoT Configuration

Perform SC-FDMA modulation of ten time slots of uniformly distributed noise, specifying a multitone NB-IoT downlink configuration.

Initialize the UE-specific settings by specifying the NB-IoT uplink subcarrier spacing.

```
ue.NBULSubcarrierSpacing = '15kHz';
```

Get the resource grid for the specified number of time slots.

```
NSlots = 10; % Number of slots in the generated waveform
dims = [12 7*NSlots];
grid = reshape(lteSymbolModulate(randi([0,1],prod(dims)*2,1),'QPSK'),dims);
```

Perform SC-FDMA modulation and display the first five symbols of the modulated waveform.

```
waveform = lteSCFDMAModulate(ue,grid);
disp(waveform(1:5));
```

```
   0.0152 + 0.0178i
   0.0126 + 0.0159i
   0.0092 + 0.0130i
   0.0052 + 0.0092i
   0.0006 + 0.0047i
```

### Perform SC-FDMA Modulation for Single-Tone NB-IoT Configuration

Perform SC-FDMA modulation for 20 time slots of uniformly distributed noise, specifying a single-tone NB-IoT configuration with 15 kHz subcarrier spacing.

Initialize UE-specific settings, specifying an NB-IoT configuration with a subcarrier spacing of 15 kHz.

```
ue.NBULSubcarrierSpacing = '15kHz';
```

Set the channel transmission configuration, specifying the fields required for the chosen NB-IoT configuration.

```
chs = struct('NULSlots',16,'NRU',2,'NRep',4,'SlotIdx',120, ...
    'NBULSubcarrierSet',0,'Modulation','QPSK');
```

Get the narrowband resource grid for the 20 time slots.

```
NSlots =  20;
grid = repmat(lteNBResourceGrid(ue),1,NSlots);
grid(chs.NBULSubcarrierSet+1,:) = lteSymbolModulate(randi([0,1],size(grid,2)*2,1),'QPSK').';
```

Perform SC-FDMA modulation and display the first five symbols in the modulated time-domain waveform.

```
waveform = lteSCFDMAModulate(ue,chs,grid);
disp(waveform(1:5));
```

```
   0.0074 + 0.0026i
   0.0078 + 0.0006i
   0.0077 - 0.0015i
   0.0070 - 0.0035i
   0.0058 - 0.0052i
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific settings, specified as a structure. The fields you specify in `ue` and `chs` determine whether the function performs SC-FDMA modulation for an LTE or NB-IoT configuration. To choose an NB-IoT configuration, specify the `NBULSubcarrierSpacing` field. To choose an LTE configuration, omit the `NBULSubcarrierSpacing` field. The `Windowing` field is optional, and you can specify it for either an LTE or NB-IoT configuration. The `CyclicPrefixUL` field is optional and is applicable only for an LTE configuration..

**`Windowing` — Number of windowing samples**
nonnegative integer

Number of time-domain samples over which the function applies windowing and overlapping of SC-FDMA symbols, specified as a nonnegative integer. This field is optional.

---

**Note** If you do not specify `Windowing`, `lteSCFDMAModulate` returns the `Windowing` field of `info` as a default value chosen as a function of `NULRB` (for LTE uplink configurations) or `NBULSubcarrierSpacing` (for NB-IoT uplink configurations). This behavior compromises between the effective duration of the cyclic prefix (and therefore the channel delay spread tolerance) and the spectral characteristics of the transmitted signal (not considering any additional FIR filtering). If `Windowing` is zero, issues identified in the description of `grid` concerning concatenation of slots before SC-FDMA modulation do not apply.

The number of samples used for windowing depends on the cyclic prefix length (normal or extended) and the number of resource blocks. The default is chosen in accordance with the maximum values implied in TS 36.104, Tables E.5.1-1 and E.5.1-2 [1]. For a larger value of `Windowing`, the effective duration of the cyclic prefix is reduced but the transmitted signal spectrum has smaller out-of-band emissions.

| Number of Resource Blocks NRB | Windowing Samples for Normal Cyclic Prefix | Windowing Samples for Extended Cyclic Prefix |
|---|---|---|
| 6 | 4 | 4 |
| 15 | 6 | 6 |
| 25 | 4 | 4 |
| 50 | 6 | 6 |
| 75 | 8 | 8 |
| 100 | 8 | 8 |

---

Data Types: `double`

**`CyclicPrefixUL` — Cyclic prefix length**
`'Normal'` (default) | `'Extended'`

Cyclic prefix length, specified as `'Normal'` or `'Extended'`. This field is optional.

**Dependencies**

This field applies only when you choose an LTE configuration by omitting the `NBULSubcarrierSpacing` field.

Data Types: `char` | `string`

**`NBULSubcarrierSpacing` — NB-IoT subcarrier spacing**
`'3.75kHz'` | `'15kHz'`

NB-IoT subcarrier spacing, specified as `'3.75kHz'` or `'15kHz'`. To set a subcarrier spacing of 3.75 kHz, specify this field as `'3.75kHz'`. To set a subcarrier spacing of 15 kHz, specify this field as `'15kHz'`.

To use `lteSCFDMAModulate` for NB-IoT modulation, you must specify this field. To indicate an LTE configuration, omit this field.

**Note** For a subcarrier spacing of 3.75 kHz, `lteSCFDMAModulate` supports only single-tone NB-IoT configurations.

Data Types: `char` | `string`

**`grid` — Resource grid**
numeric array

Resource grid, specified as a numeric array of size *M*-by-*N*-by-*P*, where:

- *M* is the number of subcarriers.
- *N* is the number of SC-FDMA symbols.
- *P* is the number of transmission antennas.

You can specify `grid` to contain REs for various time slots across all configured antenna ports, as described in “Represent Resource Grids”. Alternatively, you can specify `grid` to contain multiple such matrices concatenated across the second dimension to give multiple slots. The antenna planes in `grid` are each OFDM modulated to give the columns of the `waveform` output.

For an LTE uplink configuration, *M* must be a multiple of 12, since the number of resource blocks is $N_{RB} = M/12$, up to a maximum of 2048. For an NB-IoT downlink or uplink configuration with the `NBULSubcarrierSpacing` field of `ue` set to `'15kHz'`, $M = 12$. For an NB-IoT uplink configuration with `NBULSubcarrierSpacing` set to `'3.75kHz'`, $M = 48$. Specify *N* as a multiple of the number of symbols in a slot *L*, where $L = 14$ for normal cyclic prefix and $L = 12$ for extended cyclic prefix. You can specify *P* as 1, 2, or 4.

The grid can span multiple time slots. Windowing and overlapping are applied between all adjacent SC-FDMA symbols, including the last of one slot and the first of the next. Therefore, a different result is obtained than when `lteSCFDMAModulate` is called on individual slots and those time-domain waveforms are concatenated. The resulting waveform in the latter case has discontinuities at the start and end of each slot. It is recommended that all slots for SC-FDMA modulation first be concatenated before calling `lteSCFDMAModulate` on the resulting multislot array. However, you can perform OFDM modulation on individual slots and create the resulting multislot time-domain waveform by manually overlapping.

Data Types: `double`
Complex Number Support: Yes

**`chs` — Channel transmission configuration**
structure

Channel transmission configuration, specified as a structure. For an NB-IoT configuration, you can set additional uplink-specific parameters by specifying the NB-IoT-specific fields in `chs`. Except for the

`NBULSubcarrierSet` field, the fields in `chs` are applicable either when the `NBULSubcarrierSpacing` field of `ue` is `'3.75kHz'` or when `NBULSubcarrierSpacing` is `'15kHz'` and `length(chs.NBULSubcarrierSet)` is 1.

**`NBULSubcarrierSet` — NB-IoT uplink subcarrier indices**
vector of nonnegative integers (default) | nonnegative integer

NB-IoT uplink subcarrier indices, specified as a vector of nonnegative integers in the interval [0, 11] or a nonnegative integer in the interval [0, 47]. The indices are in zero-based form. To use `lteSCFDMAModulate` for single-tone NB-IoT modulation, you must specify `NBULSubcarrierSet` as a scalar. If you do not specify `NBULSubcarrierSet`, `lteSCFDMAModulate` performs multitone NB-IoT modulation by default. If you specify the `NBULSubcarrierSpacing` field of `ue` as `'15kHz'`, this field is required.

Data Types: `double`

**`Modulation` — Modulation type**
`'BPSK'` | `'QPSK'`

Modulation type, specified as `'BPSK'` or `'QPSK'`. For binary phase shift keying (BPSK), specify `Modulation` as `'BPSK'`. For quadrature phase shift keying (QPSK), specify `Modulation` as `'QPSK'`.

Data Types: `char` | `string`

**`NULSlots` — Number of slots per RU**
positive integer

Number of slots per resource unit (RU), specified as a positive integer. To use `lteSCFDMAModulate` for single-tone NB-IoT modulation, you must specify this field.

Data Types: `double`

**`NRU` — Number of RUs**
positive integer

Number of RUs, specified as a positive integer. To use `lteSCFDMAModulate` for single-tone NB-IoT modulation, you must specify this field.

Data Types: `double`

**`NRep` — Number of repetitions for codeword**
nonnegative integer

Number of repetitions for a codeword, specified as a nonnegative integer. To use `lteSCFDMAModulate` for single-tone NB-IoT modulation, you must specify this field.

Data Types: `double`

**`SlotIdx` — Relative slot index in NPUSCH bundle**
`0` (default) | nonnegative integer

Relative slot index in a narrowband physical uplink shared channel (NPUSCH) bundle, specified as a nonnegative integer. This field determines the zero-based relative slot index in a bundle of time slots for transmission of a transport block or control information bit.

Data Types: `double`

Data Types: `struct`

**`windowing` — Number of windowed and overlapped samples**
nonnegative integer

Number of windowed and overlapped samples, specified as a nonnegative integer. This argument controls the number of windowed and overlapped samples used in time-domain windowing. If you specify this input, the function uses the value you specify for SC-FMDA modulation (instead of the `Windowing` field of the `ue` input) and returns it as the value of the `Windowing` field in the `info` output.

Data Types: `double`

## Output Arguments

**`waveform` — SC-FDMA-modulated waveform**
complex-valued matrix

SC-FDMA-modulated waveform, returned as a complex-valued matrix. The dimensions of `waveform` are *T*-by-*P*, where *T* is the number of time-domain samples, and *P* is the number of transmission antennas. The dimension *T* is given by $T = 15K/N_{\mathrm{FFT}}$, where $N_{\mathrm{FFT}}$ is the IFFT size, and *K* is the number of time slots in the `grid` input. When $M \geq 72$, $N_{\mathrm{FFT}}$ is a function of the number of resource blocks ($N_{\mathrm{RB}}$), and $N_{\mathrm{RB}} = M/12$.

| $N_{\mathrm{RB}}$ | $N_{\mathrm{FFT}}$ |
|---|---|
| 6 | 128 |
| 15 | 256 |
| 25 | 512 |
| 50 | 1024 |
| 75 | 2048 |
| 100 | 2048 |

When $M = 12$ or the `NBULSubcarrierSpacing` field is `'15kHz'` (NB-IoT downlink or NB-IoT uplink with 15-kHz subcarrier spacing), $N_{\mathrm{FFT}} = 128$. When the `NBULSubcarrierSpacing` field is `'3.75kHz'` (NB-IoT uplink with 3.75-kHz subcarrier spacing), $N_{FFT} = 512$. When $M \geq 72$, $N_{FFT}$ is the smallest power of 2 greater than or equal to $12N_{RB}/0.85$. This value is the smallest FFT that spans all subcarriers and results in a bandwidth occupancy ($12N_{RB}/N_{FFT}$) of no more than 85%.

Data Types: `double`
Complex Number Support: Yes

**`info` — Information about SC-FDMA modulated waveform**
structure

Information about SC-FDMA modulated waveform, returned as a structure containing these fields.

**`NBULGapSamples` — Number of padded gap samples**
positive integer

Number of padded gap samples at the end of each time slot, returned as a positive integer. When the `NBULSubcarrierSpacing` field is `'3.75kHz'`, `NBULGapSamples` is `144`. Otherwise, `NBULGapSamples` is `0`.

#### Dependencies

This argument is returned only when the `NBULSubcarrierSpacing` field of `ue` is specified.

Data Types: `double`

### `CyclicPrefixLengths` — Cyclic prefix length

vector of positive integers

Cyclic prefix length, in number of time-domain samples, returned as a vector of positive integers. Each entry represents the cyclic prefix length of the corresponding orthogonal frequency-division multiplexing (OFDM) symbol in a time slot. The function returns `CyclicPrefixLengths` in accordance with the specified input fields shown in these tables.

#### LTE Configuration

| `Nfft` | `CyclicPrefixLengths` when `CyclicPrefixUL` is set to `'Normal'` | `CyclicPrefixLengths` when `CyclicPrefixUL` is set to `'Extended')` |
|---|---|---|
| 128 | [10 9 9 9 9 9 9 10 9 9 9 9 9 9] | [32 32 32 32 32 32 32 32 32 32 32 32] |
| 256 | [20 18 18 18 18 18 18 20 18 18 18 18 18 18] | [64 64 64 64 64 64 64 64 64 64 64 64] |
| 512 | [40 36 36 36 36 36 36 40 36 36 36 36 36 36] | [128 128 128 128 128 128 128 128 128 128 128 128] |
| 1024 | [80 72 72 72 72 72 72 80 72 72 72 72 72 72] | [256 256 256 256 256 256 256 256 256 256 256 256] |
| 2048 | [160 144 144 144 144 144 144 160 144 144 144 144 144 144] | [512 512 512 512 512 512 512 512 512 512 512 512] |

#### NB-IoT Configuration

| `Nfft` | `NBULSubcarrierSpacing` | `CyclicPrefixLengths` |
|---|---|---|
| 128 | `'15kHz'` | [10 9 9 9 9 9 9 10 9 9 9 9 9 9] |
| 512 | `'3.75kHz'` | [16 16 16 16 16 16 16 16 16 16 16 16 16 16] |

**Note** As shown in the tables, for values of `Nfft` less than 2048, the entries of `CyclicPrefixLengths` are given by multiplying the cyclic prefix lengths when `Nfft` is 2048 by `Nfft`/2048.

Data Types: `int32`

### `Windowing` — Number of time-domain samples

nonnegative integer

Number of time-domain samples over which the function applies windowing and overlapping of SC-FDMA symbols, returned as a nonnegative integer.

Data Types: `double`

**`Nfft` — Number of FFT points**
positive integer

Number of FFT points, $N_{FFT}$, returned as a positive integer.

Data Types: `double`

**`SamplingRate` — Sampling rate of time-domain waveform**
positive scalar

Sampling rate of time-domain waveform, returned as a positive scalar. When the `NBULSubcarrierSpacing` field is `'15kHz'` or unspecified, the sampling rate of the waveform is (30.72 MHz / 2048) × $N_{FFT}$, where $N_{FFT}$ is the number of fast Fourier transform (FFT) points. When you indicate an NB-IoT configuration by specifying `ue.NBULSubcarrierSpacing`, the sampling rate is 1.92 MHz.

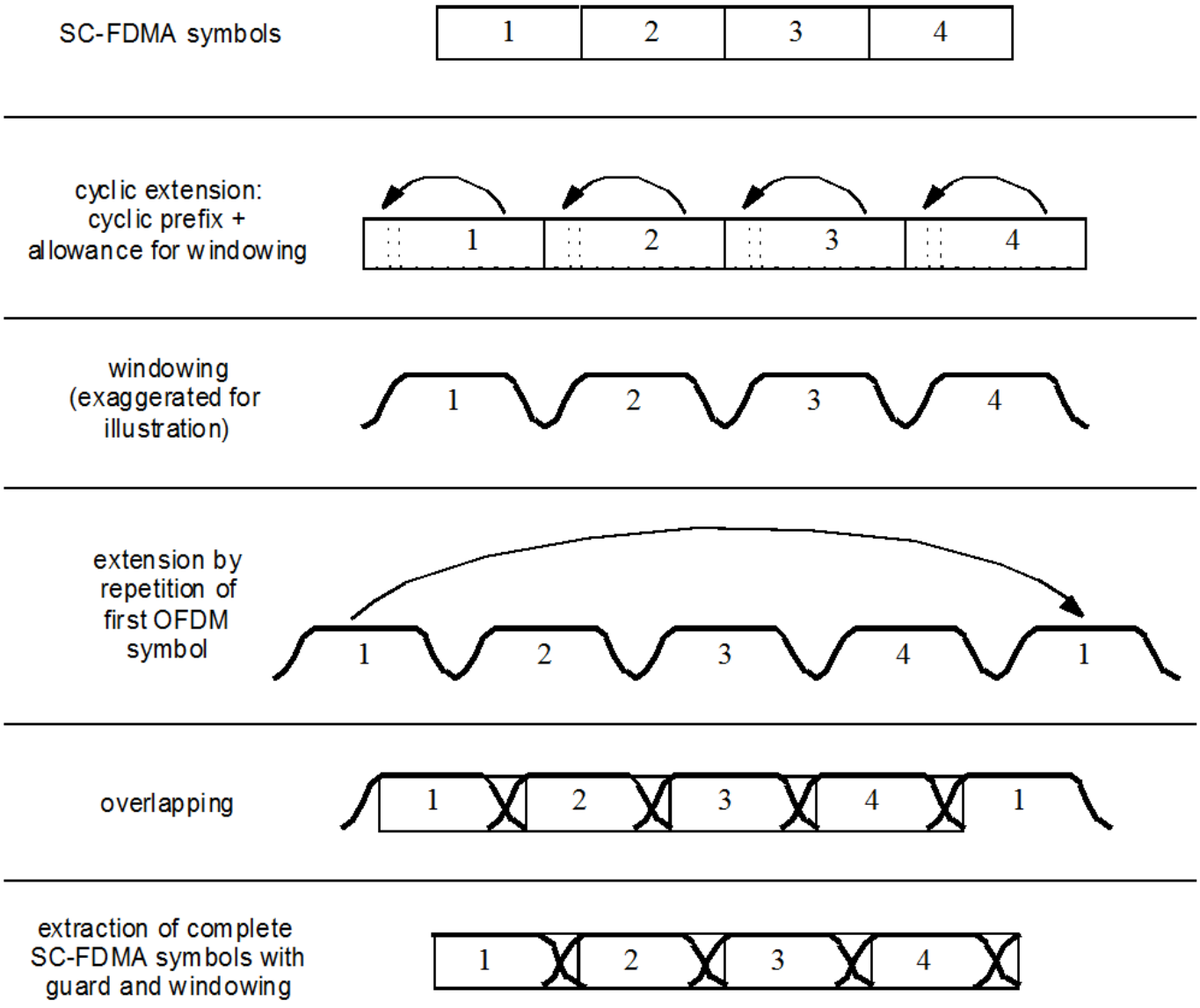
Data Types: `double`

Data Types: `struct`

## Algorithms

### SC-FDMA Modulation Processing

This diagram shows the processing performed by SC-FDMA modulation.




## Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.104. “Base Station (BS) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. URL: https://www.3gpp.org.

## See Also

`lteSCFDMADemodulate` | `lteSCFDMAInfo` | `lteULResourceGridSize` | `lteULResourceGrid` | `lteFadingChannel` | `lteHSTChannel` | `lteMovingChannel`

# lteSCFDMAInfo

Get SC-FDMA modulation information

## Syntax

```
info = lteSCFDMAInfo(ue)
```

## Description

`info = lteSCFDMAInfo(ue)` returns the structure `info`, which contains information related to the single-carrier frequency-division multiplexing (SC-FDMA) modulation performed by the `lteSCFDMAModulate` function for the user-equipment-specific (UE-specific) settings structure, `ue`.

## Examples

### Get SC-FDMA Modulation Information

Initialize UE-specific settings by setting the number of resource blocks.

```
ue = struct('NULRB',50);
```

Get SC-FDMA modulation information and display the sampling rate.

```
info = lteSCFDMAInfo(ue);
disp(info.SamplingRate);
```

```
    15360000
```

### Get Sampling Rate of NB-IoT Uplink Waveform

Get the sampling rate of an NB-IoT uplink waveform with 3.75-kHz subcarrier spacing after SC-FDMA modulation.

Specify the NB-IoT uplink subcarrier spacing.

```
ue.NBULSubcarrierSpacing = '3.75kHz';
```

Get the SC-FDMA modulation information and display the sampling rate of the time-domain waveform.

```
info = lteSCFDMAInfo(ue);
disp(info.SamplingRate);
```

```
     1920000
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific settings, specified as a structure. The fields you specify in `ue` determine whether the function returns SC-FDMA modulation information for an LTE or NB-IoT configuration. To choose an LTE configuration, specify the `NULRB` field. To choose an NB-IoT configuration, specify the `NBULSubcarrierSpacing` field. The `CyclicPrefixUL` field is optional and is applicable only for an LTE configuration. The `Windowing` field is optional, and you can specify it for either an LTE or NB-IoT configuration.

### NULRB — Number of uplink resource blocks

integer in the interval [6, 110]

Number of uplink resource blocks, $N_{RB}^{UL}$, specified as an integer in the interval [6, 110]. To return SC-FDMA modulation information for an LTE configuration, you must specify this field.

Data Types: `double`

### CyclicPrefixUL — Cyclic prefix length

`'Normal'` (default) | `'Extended'`

Cyclic prefix length, specified as `'Normal'` or `'Extended'`. This field is optional.

#### Dependencies

This field applies only when you choose an LTE configuration by specifying the `NULRB` field.

Data Types: `char` | `string`

### NBULSubcarrierSpacing — NB-IoT uplink subcarrier spacing

`'3.75kHz'` | `'15kHz'`

NB-IoT uplink subcarrier spacing, specified as `'3.75kHz'` or `'15kHz'`. To set a subcarrier spacing of 3.75 kHz, specify `NBULSubcarrierSpacing` as `'3.75kHz'`. To set a subcarrier spacing of 15 kHz, specify `NBULSubcarrierSpacing` as `'15kHz'`.

To return SC-FDMA modulation information for an NB-IoT configuration, you must specify this field. To indicate an LTE configuration, omit this field.

**Note** For a subcarrier spacing of 3.75 kHz, `lteSCFDMAInfo` supports only single-tone NB-IoT configurations.

Data Types: `char` | `string`

### Windowing — Number of windowing samples

nonnegative integer

Number of time-domain samples over which the function applies windowing and overlapping of SC-FDMA symbols, specified as a nonnegative integer. This field is optional.

**Note** If you do not specify this input, the `function` returns the `Windowing` field of the `info` output as a default value chosen as a function of `NULRB` (for LTE configurations) or

`NBULSubcarrierSpacing` (for NB-IoT configurations). This behavior compromises between the effective duration of the cyclic prefix (and therefore the channel delay spread tolerance) and the spectral characteristics of the transmitted signal (not considering any additional FIR filtering).

For more information, see the `lteSCFDMAModulate` function.

Data Types: `double`

Data Types: `struct`

## Output Arguments

### `info` — Information related to SC-FDMA modulation

structure

Information related to SC-FDMA modulation, returned as a structure containing these fields.

### `NBULGapSamples` — Number of padded gap samples

positive integer

Number of padded gap samples at the end of each time slot, returned as a positive integer. When the `NBULSubcarrierSpacing` field is `'3.75kHz'`, `NBULGapSamples` is `144`. Otherwise, `NBULGapSamples` is `0`.

#### Dependencies

This argument is returned only when the `NBULSubcarrierSpacing` field of `ue` is specified.

Data Types: `double`

### `CyclicPrefixLengths` — Cyclic prefix length

vector of positive integers

Cyclic prefix length, in number of time-domain samples, returned as a vector of positive integers. Each entry represents the cyclic prefix length of the corresponding orthogonal frequency-division multiplexing (OFDM) symbol in a time slot. The function returns `CyclicPrefixLengths` in accordance with the specified input fields shown in these tables.

**LTE Configuration**

| Nfft | CyclicPrefixLengths when CyclicPrefixUL is set to 'Normal' | CyclicPrefixLengths when CyclicPrefixUL is set to 'Extended') |
|---|---|---|
| 128 | [10 9 9 9 9 9 9 10 9 9 9 9 9 9] | [32 32 32 32 32 32 32 32 32 32 32 32] |
| 256 | [20 18 18 18 18 18 18 20 18 18 18 18 18 18] | [64 64 64 64 64 64 64 64 64 64 64 64] |
| 512 | [40 36 36 36 36 36 36 40 36 36 36 36 36 36] | [128 128 128 128 128 128 128 128 128 128 128 128] |
| 1024 | [80 72 72 72 72 72 72 80 72 72 72 72 72 72] | [256 256 256 256 256 256 256 256 256 256 256 256] |
| 2048 | [160 144 144 144 144 144 144 160 144 144 144 144 144 144] | [512 512 512 512 512 512 512 512 512 512 512 512] |

**NB-IoT Configuration**

| Nfft | NBULSubcarrierSpacing | CyclicPrefixLengths |
|---|---|---|
| 128 | '15kHz' | [10 9 9 9 9 9 9 10 9 9 9 9 9 9] |
| 512 | '3.75kHz' | [16 16 16 16 16 16 16 16 16 16 16 16 16 16] |

**Note** As shown in the tables, for values of `Nfft` less than 2048, the entries of `CyclicPrefixLengths` are given by multiplying the cyclic prefix lengths when `Nfft` is 2048 by `Nfft`/2048.

Data Types: `int32`

**`Windowing` — Number of time-domain samples**
nonnegative integer

Number of time-domain samples over which the function applies windowing and overlapping of SC-FDMA symbols, returned as a nonnegative integer.

Data Types: `double`

**`Nfft` — Number of FFT points**
positive integer

Number of FFT points, $N_{FFT}$, returned as a positive integer.

Data Types: `double`

**`SamplingRate` — Sampling rate of time-domain waveform**
positive scalar

Sampling rate of time-domain waveform, returned as a positive scalar. When the `NBULSubcarrierSpacing` field is `'15kHz'` or unspecified, the sampling rate of the waveform is (30.72 MHz / 2048) × $N_{FFT}$, where $N_{FFT}$ is the number of fast Fourier transform (FFT) points. When

you indicate an NB-IoT configuration by specifying `ue.NBULSubcarrierSpacing`, the sampling rate is 1.92 MHz.

Data Types: `double`

Data Types: `struct`

# Version History

**Introduced in R2014a**

## See Also

`lteSCFDMAModulate` | `lteSCFDMADemodulate` | `lteULResourceGridSize` | `lteOFDMInfo`

# lteSCI

Sidelink control information format structure and bit payload

## Syntax

```
[sciout,bitsout] = lteSCI(ue)
[sciout,bitsout] = lteSCI(ue,sciin)
[sciout,bitsout] = lteSCI(ue,bitsin)
[sciout,bitsout] = lteSCI( ___ ,opts)
```

## Description

`[sciout,bitsout] = lteSCI(ue)` returns a sidelink control information (SCI) message structure, `sciout`, and the SCI message bit vector, `bitsout`, for the settings specified in the user equipment structure.

This function creates and manipulates SCI format 0 messages, defined in TS 36.212 [1], Section 5.4.3. You can use `lteSCI` to create a default SCI message, to blindly decode SCI format types, and to determine the sizes of the bit fields.

By default, all returned fields are set to zero.

`[sciout,bitsout] = lteSCI(ue,sciin)` returns the SCI structure fields and bit vector using settings specified in SCI input structure `sciin`. Fields not defined in `sciin` are set to defaults specified by `ue`. You can use this syntax to initialize SCI field values, in particular the frequency hopping bit, which affects the fields that the format uses.

`[sciout,bitsout] = lteSCI(ue,bitsin)` returns the SCI structure fields and bit vector using settings specified in bit input vector `bitsin`. The input bit vector is returned as the SCI information bit payload, where `bitsout == bitsin`.

`[sciout,bitsout] = lteSCI( ___ ,opts)` formats the returned structure using options specified by `opts`.

## Examples

### Create SCI Message

Create a format 0 SCI message structure.

Create a UE settings structure.

```
ue = struct('NSLRB','15MHz');
```

Generate an SCI message and view the returned SCI message structure contents.

```
[sci0,bits] = lteSCI(ue);
sci0
```

```
sci0 = struct with fields:
              SCIFormat: 'Format0'
```

```
            FreqHopping: 0
             Allocation: [1x1 struct]
    TimeResourcePattern: 0
              ModCoding: 0
            TimeAdvance: 0
                  NSAID: 0
```

```
allocfields = sci0.Allocation
```

```
allocfields = struct with fields:
    RIV: 0
```

### Create SCI Message with Distributed VRB Allocation Type

Create a format 0 SCI message structure with the distributed VRB allocation type. The allocation message fields are contained in the `Allocation` substructure. To create the appropriate set of fields at the output, the `FreqHopping` field is initialized at the input to the function.

Create a UE settings structure and define `FreqHopping` using an input SCI message structure.

```
ue = struct('NSLRB',50);
sciin = struct('FreqHopping',1);
```

Generate an SCI message and view the returned SCI message structure contents.

```
[sci0,bits] = lteSCI(ue,sciin);
sci0
```

```
sci0 = struct with fields:
              SCIFormat: 'Format0'
            FreqHopping: 1
             Allocation: [1x1 struct]
    TimeResourcePattern: 0
              ModCoding: 0
            TimeAdvance: 0
                  NSAID: 0
```

```
allocfields = sci0.Allocation
```

```
allocfields = struct with fields:
    HoppingBits: 0
            RIV: 0
```

### Recover SCI Message from Bit Vector

Recover the contents of a format 0 SCI message bit vector.

Create a UE settings structure.

```
ue = struct('NSLRB',50);
```

Generate an SCI message structure.

```
[sci0,bits] = lteSCI(ue);
sci0
```

```
sci0 = struct with fields:
              SCIFormat: 'Format0'
            FreqHopping: 0
             Allocation: [1x1 struct]
    TimeResourcePattern: 0
              ModCoding: 0
            TimeAdvance: 0
                  NSAID: 0
```

Change the `ModCoding` setting to 22 and generate an SCI bits vector.

```
sci0.ModCoding = 22;
[~,bits_new] = lteSCI(ue,sci0);
```

Use the new bits to recover the new SCI message. View the new SCI message structure and confirm that the `ModCoding` setting is now 22.

```
[sci0_new,~] = lteSCI(ue,bits_new)
```

```
sci0_new = struct with fields:
              SCIFormat: 'Format0'
            FreqHopping: 0
             Allocation: [1x1 struct]
    TimeResourcePattern: 0
              ModCoding: 22
            TimeAdvance: 0
                  NSAID: 0
```

### View SCI Message Field Sizes

Create a format 0 SCI message structure. Use the `opts` input to view the message field sizes and to exclude fields with zero length.

Create a UE settings structure.

```
ue = struct('NSLRB','5MHz');
opts = {'fieldsizes','excludeunusedfields'}
```

```
opts = 1x2 cell
    {'fieldsizes'}    {'excludeunusedfields'}
```

Generate an SCI message and view the field sizes of the returned SCI message structure contents.

```
[sci0,bits] = lteSCI(ue,opts);
sci0
```

```
sci0 = struct with fields:
              SCIFormat: 'Format0'
```

```
            FreqHopping: 1
             Allocation: [1x1 struct]
    TimeResourcePattern: 7
              ModCoding: 5
            TimeAdvance: 11
                  NSAID: 8
```

```
allocfields = sci0.Allocation
```

```
allocfields = struct with fields:
    RIV: 9
```

Inspect the returned structure to see the bit length of each field in the SCI message.

```
fieldsLength = sci0.FreqHopping + sci0.Allocation.RIV + ...
    sci0.TimeResourcePattern + sci0.ModCoding + sci0.TimeAdvance + ...
    sci0.NSAID
```

```
fieldsLength = uint64
    41
```

```
bitsLength = size(bits,1)
```

```
bitsLength = 41
```

```
isequal(fieldsLength,bitsLength)
```

```
ans = logical
   1
```

The sum of the field sizes matches the length of the returned `bits` output.

## Input Arguments

### ue — User equipment settings

structure

User equipment settings, specified as a structure containing these parameter fields:

### PSSCHNSubchannels — Number of sub-channels in the V2X PSSCH resource pool

1 (default) | integer scalar from 1 to 110 | optional

Number of sub-channels in the V2X PSSCH resource pool, specified as an integer scalar from 1 to 110. You must specify this input when you set the `sciin` to `'Format1'`.

Data Types: `double`

### NSLRB — Number of sidelink resource blocks

integer scalar from 6 to 110

Number of sidelink resource blocks, specified as an integer scalar from 6 to 110.

Example: 6, which corresponds to a channel bandwidth of 1.4 MHz.

Data Types: `double`

### `opts` — Formatting options for output SCI structure

character vector | cell array of character vectors | string array

Formatting options for output SCI structure, specified as a character vector, cell array of character vectors, or a string array. You can specify a format for the *Field content* and *Fields to include*. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string):

| Category | Options | Description |
|---|---|---|
| *Field content* | `'fieldvalues'` (default) | Set the fields to zero or to their input values. |
| | `'fieldsizes'` | Sets the field values to their bit sizes and adds the `Padding` field to `sciout`. `Padding` indicates the number of padding bits appended. |
| *Fields to include* | `'includeallfields'` (default) | `sciout` includes all possible fields for the requested SCI format. |
| | `'excludeunusedfields'` | `sciout` excludes zero-length fields for the given parameter set. |

Example: `'fieldsizes excludeunusedfields'`, `"fieldsizes excludeunusedfields"`, `{'fieldsizes','excludeunusedfields'}`, or `["fieldsizes","excludeunusedfields"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

### `sciin` — SCI message settings

structure

SCI message settings, specified as a structure containing any fields returned in `sciout`. See `sciout` for the specific fields output for each `SCIFormat`. SCI format 0 message is defined in TS 36.212 [1], Section 5.4.3.1. It can contain the following field:

#### `SCIFormat` — SCI format type

`'Format0'` (default) | `'Format1'`

SCI format type, specified as `'Format0'` or `'Format1'`.

Data Types: `char` | `string`

Data Types: `struct`

### `bitsin` — Input bits

column vector

Input bits, specified as a column vector. `bitsin` is treated as the SCI message bit payload, that is, `bitsout == bitsin`. The length of `bitsin` must align with the number of resource blocks, `ue.NSLRB`. Use `lteSCIInfo` to determine SCI message length for the specified `ue` settings.

Data Types: `double`

## Output Arguments

### `sciout` — SCI message structure

structure

SCI message structure, returned as a structure whose fields match the associated SCI format contents.

The field names associated with `sciout` depend on the SCI format field in `sciin`. By default, all values are set to zero. However, if any of the SCI fields are already present in the input `sciin`, their values are carried forward into `sciout`. The input field values appear in the associated bit positions in `bitsout`. Carrying the values forward allows for easy initialization of SCI field values. `sciout` also carries forward the `NSLRB` field specified in `sciin`.

This table presents the fields associated with each SCI format, as defined in TS 36.212 [1], Section 5.4.3.1.

| SCI Formats | `sciout` Fields | Size | Description |
|---|---|---|---|
| `'Format0'` | `SCIFormat` | - | `'Format0'` |
| | `FreqHopping` | 1 bit | PSSCH frequency hopping flag |
| | `Allocation` | from 5 to 13 bits, $\left\lceil \log_2 \left( \frac{N_{RB}^{SL} \times \left( N_{RB}^{SL} + 1 \right)}{2} \right) \right\rceil$ | Resource block assignment and hopping resource allocation substructure, type 0 or type 1 allocation |
| | `TimeResourcePattern` | 7 bits | Time resource pattern ($I_{TRP}$) |
| | `ModCoding` | 5 bits | Modulation and coding scheme ($I_{MCS}$) |
| | `TimeAdvance` | 11 bits | Timing advance indication |
| | `NSAID` | 8 bits | Group destination ID, as defined by higher layers |
| | `Padding` | 0 bits | Always zero for SCI Format 0 |
| `'Format1'` | `SCIFormat` | - | `'Format1'` |
| | `Priority` | 3 bits | Per packet priority |
| | `ResourceReservation` | 4 bits | Resource reservation |

| SCI Formats | sciout Fields | Size | Description |
|---|---|---|---|
| | RIV | from 0 to 13 bits, $\left\lceil \log_2 \left( \frac{N_{\text{subchannel}}^{\text{SL}} \times \left( N_{\text{subchannel}}^{\text{SL}} + 1 \right)}{2} \right) \right\rceil$ | Resource indication value |
| | TimeGap | 4 bits | Time gap between initial transmission and retransmission |
| | ModCoding | 5 bits | Modulation and coding scheme |
| | RetransmissionIdx | 1 bit | Retransmission index |

**bitsout — SCI message in bit payload form**
column vector of binary values

SCI message in bit payload form, returned as a column vector. `bitsout` represents the set of message fields mapped to the information bit payload (including any zero-padding).

# Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteSCIEncode` | `lteSCIDecode` | `lteSCIInfo` | `lteDCI`

# lteSCIDecode

SCI decoding

## Syntax

```
[scibits,err] = lteSCIDecode(scilen,softbits)
[scibits,err] = lteSCIDecode(ue,softbits)
```

## Description

`[scibits,err] = lteSCIDecode(scilen,softbits)` recovers a sidelink control information (SCI) message and also returns the cyclic redundancy check indication, given the SCI vector length and input vector of soft bits. For more information, see “SCI Message Decoding” on page 2-1049.

`[scibits,err] = lteSCIDecode(ue,softbits)` uses a UE settings structure to determine the SCI message length.

## Examples

### Decode Format 0 SCI Message

Decode an SCI format 0 message given the SCI message length. Use the length of an SCI format 0 message, determined using the `lteSCIInfo` function, to create and encode an SCI message.

Create a UE settings structure with 10-MHz bandwidth and normal cyclic prefix length.

```
ue = struct('NSLRB',50,'CyclicPrefixSL','Normal');
```

Determine the SCI message length with the `lteSCIInfo` function. Encode the SCI message.

```
sciInfo = lteSCIInfo(ue);
scilen = sciInfo.Format0;
sciBits = zeros(scilen,1);
cw = lteSCIEncode(ue,sciBits);
```

Decode the SCI message payload bit vector.

```
[sciBits,crcErr] = lteSCIDecode(scilen,cw);
crcErr
```

```
crcErr = logical
   0
```

The cyclic redundancy check returns a zero, indicating that the decoded SCI message has no errors.

### Decode Format 0 SCI Message Using UE Settings

Decode an SCI format 0 message using UE settings. Encode a bit vector representing an SCI information payload, and then decode and error-check the result. Use a UE settings structure to create and encode an SCI message.

Create a UE settings structure with 5 MHz bandwidth and extended cyclic prefix length. Generate and encode an SCI format 0 message.

```
ue = struct('NSLRB','5MHz','CyclicPrefixSL','Extended');

[~,sciBits] = lteSCI(ue);
cw = lteSCIEncode(ue,sciBits);
```

Decode the SCI message payload bit vector, `cw`. Use the UE settings structure to determine the SCI message length.

```
[sciBits,crcErr] = lteSCIDecode(ue,cw);
crcErr
```

```
crcErr = logical
   0
```

The cyclic redundancy check returns a zero, indicating that the decoded SCI message has no errors.

## Input Arguments

### `scilen` — Length of recovered SCI message vector

positive integer

Length of recovered SCI message vector, specified as a positive integer. This argument is normally equal to the length of the SCI format 0 message for the sidelink bandwidth. Use `lteSCIInfo` to determine the expected SCI message length.

Data Types: `double`

### `softbits` — Floating-point soft bits

column vector

Floating-point soft bits, specified as a column vector. The length of `softbits` is nominally 288 bits for normal cyclic prefix or 240 extended cyclic prefix, matching the bit capacity of the PSCCH (ignoring the SC-FDMA guard symbol). For V2X sidelink, the nominal input length is 480 bits. Otherwise, the number of soft bits must be a multiple of 2 and should be a multiple of 12 or 10 for D2D normal and V2X normal/D2D extended cyclic prefix respectively, corresponding to the number of data SC-FDMA symbols in a PSCCH subframe.

Data Types: `double` | `int8`

### `ue` — User equipment settings

structure

User equipment settings, specified as a structure containing these parameter fields:

#### `NSLRB` — Number of sidelink resource blocks

integer scalar from 6 to 110

Number of sidelink resource blocks, specified as an integer scalar from 6 to 110.

Example: `6`, which corresponds to a channel bandwidth of 1.4 MHz.

Data Types: `double`

**`CyclicPrefixSL` — Cyclic prefix length**
`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

Data Types: `struct`

## Output Arguments

**`scibits` — Recovered SCI message bits**
column vector of binary values

Recovered SCI message bits, returned as a column vector. For more information, see “SCI Message Decoding” on page 2-1049.

**`err` — CRC error status**
0 | 1

CRC error status, returned as 0 for no errors or 1 when the CRC fails.

## More About

### SCI Message Decoding

Sidelink control information (SCI) message decoding performs the inverse SCI processing operation as specified in TS 36.212 [1], Section 5.4.3. Specifically, `lteSCIDecode` performs PUSCH deinterleaving, rate recovery, and Viterbi and CRC decoding to recover the SCI message bit vector (`scibits`) from an input vector of received soft bits previously coded by the SCI processing. `lteSCIDecode` also returns the CRC error status, signaled by 0 for no errors and 1 when CRC fails.

If `scilen` is provided as an input argument, the function uses it for the length of the SCI information payload to be recovered. Otherwise the function computes the length, using the fields in `ue` that specify the bandwidth (`NSLRB`) and cyclic prefix length (`CyclicPrefixSL`).

# Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteSCIEncode` | `lteSCI` | `lteSCIInfo` | `ltePSCCHDecode`

# lteSCIEncode

SCI encoding

## Syntax

```
cw = lteSCIEncode(ue,scibits)
cw = lteSCIEncode(ue,scibits,outlen)
```

## Description

`cw = lteSCIEncode(ue,scibits)` returns the codeword resulting from the sidelink control information (SCI) encoding of the input bit vector, `scibits`, given the field settings in the user equipment structure, `ue`. As defined in TS 36.212 [1], Section 5.4.3, the encoding process includes 16 bit CRC attachment, tail biting convolutional coding, rate matching and PUSCH interleaving. This processing takes in an SCI message generated with `lteSCI`. The codeword returned is ready for transmission on the `ltePSCCH` physical channel.

`cw = lteSCIEncode(ue,scibits,outlen)` rate matches the returned codeword to the output length provided by `outlen`.

## Examples

### Encode Format 0 SCI Message

Create an SCI format 0 message structure, modify selected information field values, and generate the new SCI message and payload bits. Encode the SCI message payload bits.

Create a UE settings structure and SCI message structure.

```
ue = struct('NSLRB',50,'CyclicPrefixSL','Normal');
sci0 = lteSCI(ue);
```

Modify the SCI message structure settings and generate an SCI message bit vector.

```
sci0.FreqHopping = 1;
sci0.ModCoding = 3;
[sci0,scibits] = lteSCI(ue,sci0);
```

Generate an encoded SCI message codeword.

```
cw = lteSCIEncode(ue,scibits);
```

### Encode Format 0 SCI Message of Specified Length

Create an SCI format 0 message structure, and generate the new SCI message and payload bits. Encode the SCI message payload bits in a codeword of length specified by `outlen`.

Create a UE settings structure and SCI message structure.

```
ue = struct('NSLRB',50,'CyclicPrefixSL','Extended');
[sci0,scibits] = lteSCI(ue);
```

Generate an encoded SCI message codeword of length specified by `outlen`.

```
outlen = 144;
cw = lteSCIEncode(ue,scibits,outlen);
size(cw)
```

```
ans = 1×2

   144     1
```

## Input Arguments

### ue — User equipment settings

structure

User equipment settings, specified as a structure containing this parameter field:

#### SidelinkMode — Sidelink mode

`'D2D'` (default) | `'V2X'` | optional

Sidelink mode, specified as `'D2D'` or `'V2X'`.

Data Types: `char` | `string`

#### CyclicPrefixSL — Cyclic prefix length

`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

Data Types: `struct`

### scibits — SCI message bit vector

column vector

SCI message bit vector, specified as a column vector. `scibits` are the SCI processing input bits to be transmitted on a single PSCCH.

Data Types: `double` | `int8`

### outlen — Codeword length

nonnegative integer | optional

Codeword length, specified as a nonnegative integer. `outlen` must be a multiple of 2. It should be a multiple of 12 for D2D normal cyclic prefix and a multiple of 10 for extended cyclic prefix and V2X. The output length is meant to match the number of data-carrying SC-FDMA symbols in a PSCCH subframe and align with the dimensions of the PUSCH interleaver stage.

## Output Arguments

### `cw` — Codeword

288 bit or 240 bit column vector | column vector with zero rows | column vector with length equal to `outlen`

Codeword resulting from SCI processing, returned as a column vector of binary values. `cw` is the result of SCI processing the input vector, `scibits`. The output codeword matches the normal or extended cyclic prefix bit capacity available in the `ltePSCCHIndices` output, not accounting for the sidelink SC-FDMA guard symbol. Depending on the function syntax used and input configuration, the length of `cw` is:

- 288 bits for nonempty data input and normal cyclic prefix
- 240 bits for nonempty data input and extended cyclic prefix and for V2X
- An empty 0-by-1 matrix for an empty data input
- Rate-matched to `outlen`

# Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteSCI` | `lteSCIDecode` | `lteSCIInfo` | `ltePSCCH` | `lteDCIEncode`

# lteSCIInfo

SCI message information

## Syntax

```
info = lteSCIInfo(ue)
```

## Description

`info = lteSCIInfo(ue)` returns an information structure indicating the payload sizes for SCI message formats. Release 12 defines a single SCI format 0, and Release 14 defines an additional format 1 for V2X sidelink. The output structure contains two fields with the message lengths for each format.

To access the individual bit field sizes for the specified format, use `lteSCI`.

## Examples

### Get SCI Message Information

Get the information payload size of SCI message format 0 and format 1 for UE settings configuration with 10 MHz channel bandwidth.

A channel bandwidth of 10 MHz requires 50 resource blocks, `NSLRB = 50`.

```
ue = struct('NSLRB',50);
sci0length = lteSCIInfo(ue)
```

```
sci0length = struct with fields:
    Format0: 43
    Format1: 32
```

### Get SCI Message Information for Standard Bandwidths

Get the information payload size of SCI message format 0 for standard bandwidths.

```
cbw = {'1.4MHz' '3MHz' '5MHz' '10MHz' '15MHz' '20MHz'};
disp('Bandwidth  SCI Message Length (bits)')
```

```
Bandwidth  SCI Message Length (bits)
```

```
for ii = 1:size(cbw,2)
    ue = struct('NSLRB',cbw(1,ii));
    sci0length = lteSCIInfo(ue);
    bw = cbw{1,ii};
    fprintf('%6s     %3d\n',bw, sci0length.Format0)
end
```

```
1.4MHz      37
  3MHz      39
  5MHz      41
 10MHz      43
 15MHz      44
 20MHz      45
```

## Input Arguments

### `ue` — User equipment settings

structure

User equipment settings, specified as a structure containing this parameter field:

### `NSLRB` — Number of sidelink resource blocks

integer scalar from 6 to 110

Number of sidelink resource blocks, specified as an integer scalar from 6 to 110.

Example: `6`, which corresponds to a channel bandwidth of 1.4 MHz.

Data Types: `double`

## Output Arguments

### `info` — Payload size for the SCI message format

structure

Payload size for the SCI message format, returned as a structure with the following parameter fields:

### `Format0` — Format 0 payload size

integer

Format 0 payload size, returned as an integer indicating the SCI message length used for the scheduling of PSSCH.

Data Types: `double`

### `Format1` — Format 1 payload size

integer

Format 1 payload size, returned as an integer indicating the SCI message length used for the scheduling of V2X PSSCH.

Data Types: `double`

# Version History

**Introduced in R2016b**

## See Also

`lteSCI` | `lteSCIEncode` | `lteSCIDecode` | `lteDCIInfo`

# lteSCIResourceAllocation

SCI message physical resource blocks allocation

## Syntax

```
prbset = lteSCIResourceAllocation(ue,scistr)
```

## Description

`prbset = lteSCIResourceAllocation(ue,scistr)` returns a column vector containing the zero-based physical resource block (PRB) indices for the specified UE settings and as defined by the resource allocation substructure of the sidelink control information (SCI) message structure. The PRB indices created are for a single PSSCH transmission in a subframe within the PSSCH subframe pool.

For more information, see “SCI Resource Allocation” on page 2-1064.

## Examples

### Allocate Nonhopping PSSCH Subframe Pool PRBs

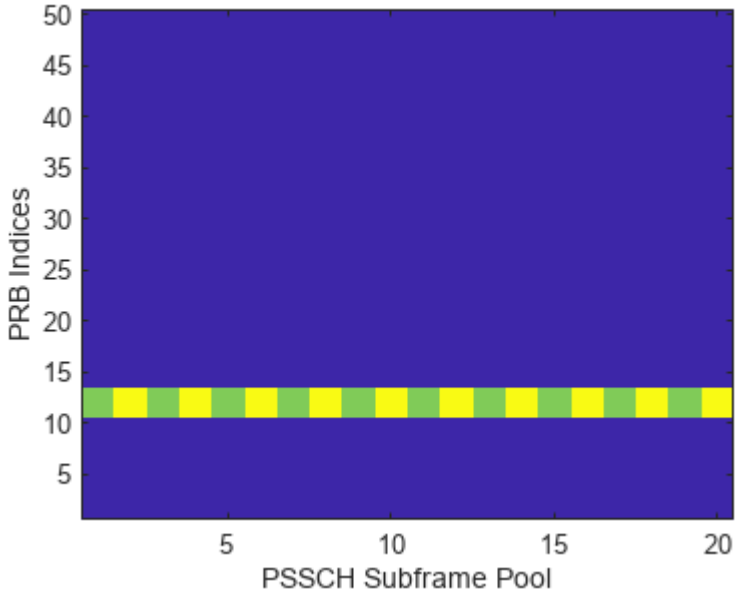
Display the PRB allocations associated with the sequence of subframes in a PSSCH subframe pool.

Configure a nonhopping allocation of 3 PRBs according to the RIV calculation specified in TS 36.213, Section 8.1.1.

```
ue = struct('NSLRB',50);
sci = struct('FreqHopping',0);
sci.Allocation.RIV = 110;
```

Display an image of the PRBs used in each slot of each subframe in a pool of 10 PSSCH subframes.

```
subframeslots = zeros(ue.NSLRB,20);
for i = 0:9
    ue.NSubframePSSCH = i;
    prbSet = lteSCIResourceAllocation(ue,sci);
    prbSet = repmat(prbSet,1,2/size(prbSet,2));
    for s = 1:2
        subframeslots(prbSet(:,s)+1,2*i+s) = 20+s*20;
    end
end
imagesc(subframeslots)
axis xy
xlabel('PSSCH Subframe Pool')
ylabel('PRB Indices')
```




### Allocate Type 2 Hopping PSSCH Subframe Pool PRBs

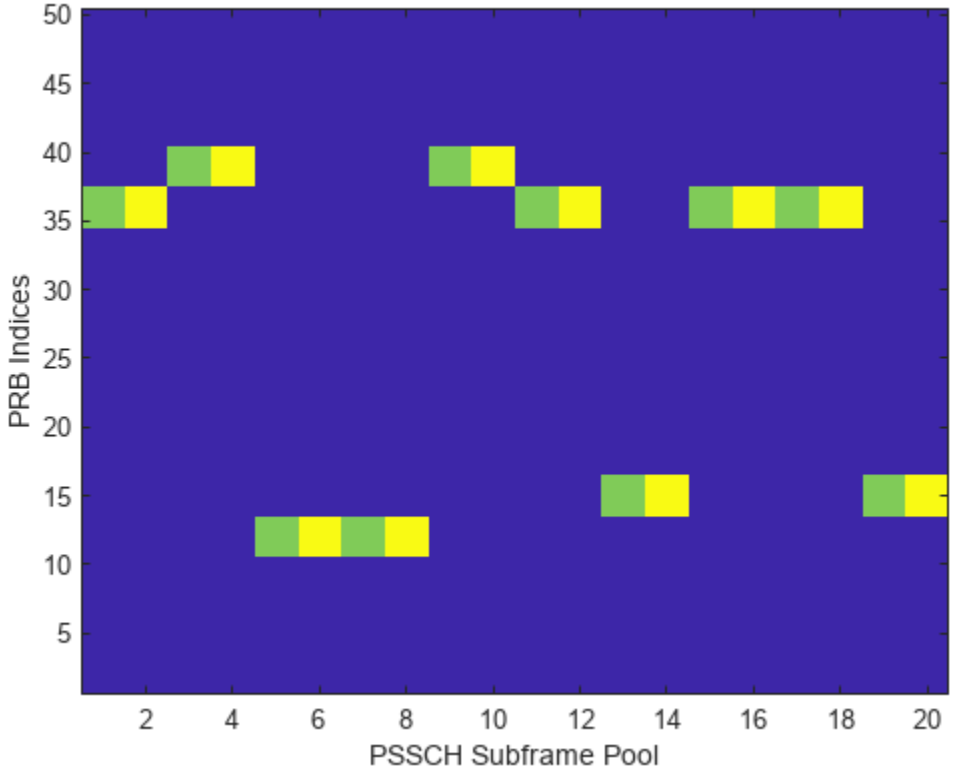
Configure a type 2 hopping allocation of 3 PRBs. Display the PRB allocations that are associated with the sequence of subframes in a PSSCH subframe pool.

Configure UE and SCI settings structures for a type 2 hopping allocation of 3 PRBs.

```
ue = struct('NSLRB',50);
ue.PSSCHHoppingParameter = 10;
ue.NSubbands = 2;
ue.PSSCHHoppingOffset = 1;
sci = struct('FreqHopping',1);
sci.Allocation.RIV = 110;
sci.Allocation.HoppingBits = 3;
```

Display an image of the PRBs used in each slot of each subframe in a pool of 10 PSSCH subframes.

```
subframeslots = zeros(ue.NSLRB,20);
for i = 0:9
    ue.NSubframePSSCH = i;
    prbSet = lteSCIResourceAllocation(ue,sci);
    prbSet = repmat(prbSet,1,2/size(prbSet,2));
    for s = 1:2
        subframeslots(prbSet(:,s)+1,2*i+s) = 20+s*20;
    end
end
imagesc(subframeslots)
axis xy
xlabel('PSSCH Subframe Pool')
ylabel('PRB Indices')
```




### Allocate Type 1 Hopping PSSCH Subframe Pool PRBs

Configure a type 1 hopping allocation of 3 PRBs. Display the PRB allocations that are associated with the sequence of subframes in a PSSCH subframe pool.

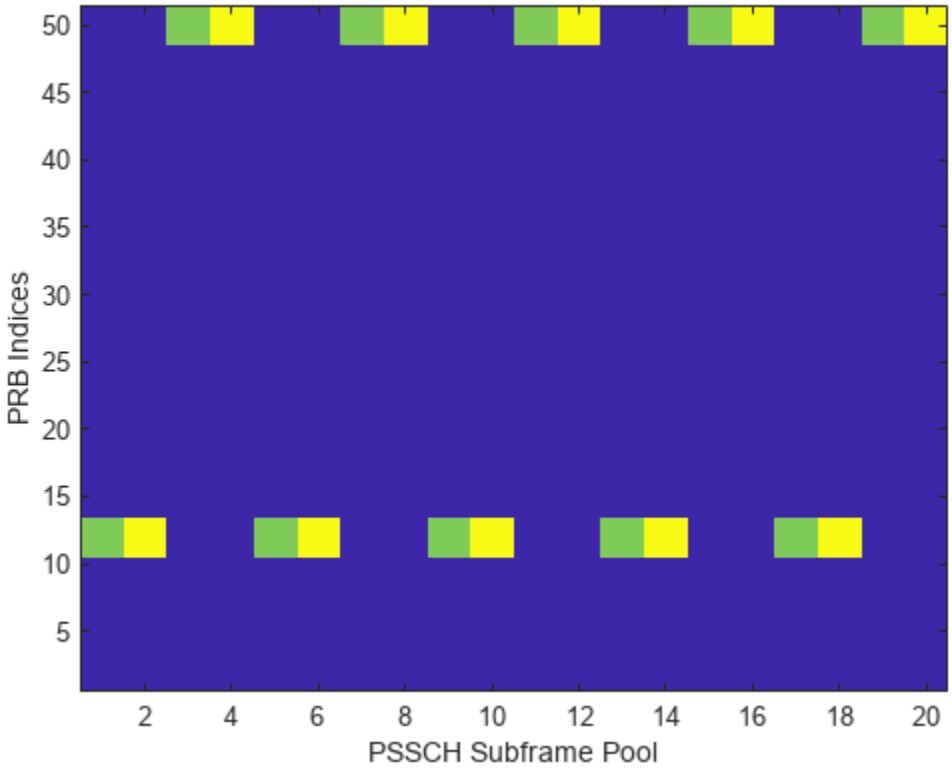
Configure UE and SCI settings structures for a type 1 hopping allocation of 3 PRBs.

```
ue = struct('NSLRB',50);
sci = struct('FreqHopping',1);
sci.Allocation.RIV = 110;
sci.Allocation.HoppingBits = 1;
```

Display an image of the PRBs used in each slot of each subframe in a pool of 10 PSSCH subframes.

```
subframeslots = zeros(ue.NSLRB,20);
for i = 0:9
    ue.NSubframePSSCH = i;
    prbSet = lteSCIResourceAllocation(ue,sci);
    prbSet = repmat(prbSet,1,2/size(prbSet,2));
    for s = 1:2
        subframeslots(prbSet(:,s)+1,2*i+s) = 20+s*20;
    end
end
imagesc(subframeslots)
axis xy
```

```
xlabel('PSSCH Subframe Pool')
ylabel('PRB Indices')
```




### Allocate Type 1 Hopping PSSCH Pool Restricting PRBs

Configure PRB pool restriction for transmission mode 2. Display the PRB allocations that are associated with the sequence of subframes in a PSSCH subframe pool.

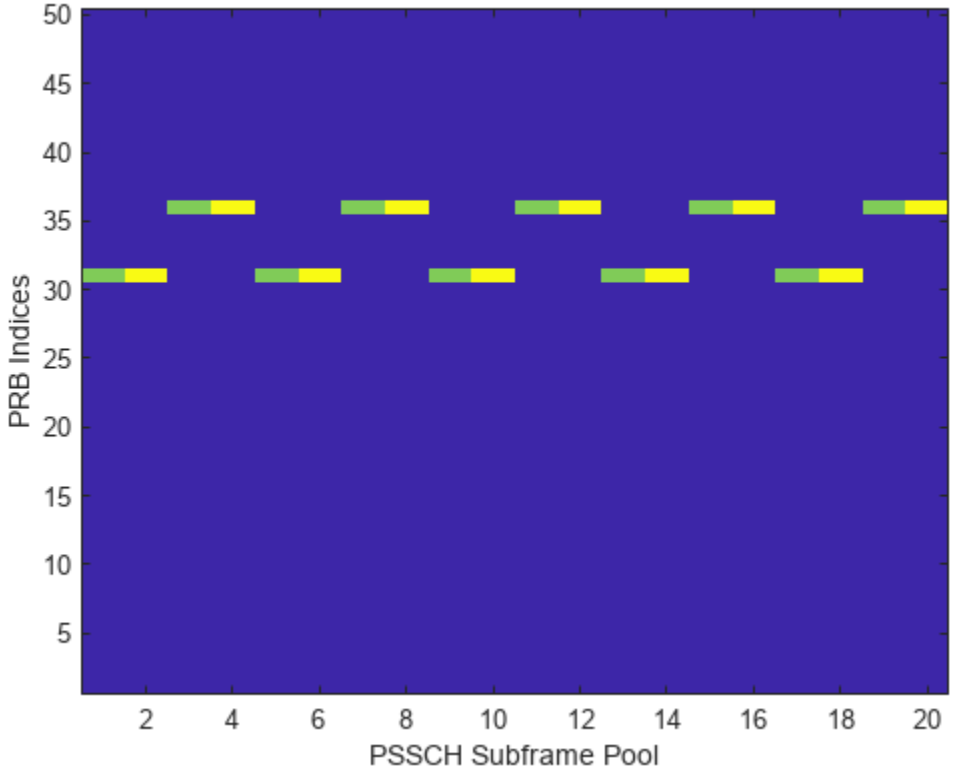
Configure a UE settings structure with specified PRB indices. Default settings are used for other UE and SCI fields.

```
ue = struct('NSLRB',50);
ue.PRBPool = (30:49);
sci = struct('FreqHopping',1);
```

Display an image of the PRBs used in each slot of each subframe in a pool of 10 PSSCH subframes.

```
subframeslots = zeros(ue.NSLRB,20);
for i = 0:9
    ue.NSubframePSSCH = i;
    prbSet = lteSCIResourceAllocation(ue,sci);
    prbSet = repmat(prbSet,1,2/size(prbSet,2));
    for s = 1:2
        subframeslots(prbSet(:,s)+1,2*i+s) = 20+s*20;
    end
```

```
end
imagesc(subframeslots)
axis xy
xlabel('PSSCH Subframe Pool')
ylabel('PRB Indices')
```




## Input Arguments

### ue — User equipment settings

structure

User equipment settings, specified as a parameter structure containing these fields:

### NSLRB — Number of sidelink resource blocks

integer scalar from 6 to 110

Number of sidelink resource blocks, specified as an integer scalar from 6 to 110.

Example: 6, which corresponds to a channel bandwidth of 1.4 MHz.

Data Types: `double`

### NSubframePSSCH — PSSCH subframe number

integer scalar | optional

PSSCH subframe number in PSSCH subframe pool, specified as an integer scalar. ($n_{ssf}^{PSSCH}$)

**Note** This parameter is required for SCI format 0 and frequency hopping. (`scistr.SCIFormat = 'Format0'` and `scistr.FreqHopping = 1`)

Data Types: `double`

**`PSSCHHoppingParameter` — PSSCH hopping parameter**
0 (default) | integer scalar from 0 to 510 | optional

PSSCH hopping parameter, specified as an integer scalar from 0 to 510. (*SL-HoppingConfigComm-r12 {hoppingParameter-r12}*)

All values ≥ 504 are treated as 510.

**Note** This parameter is required for SCI format 0 and frequency hopping. (`scistr.SCIFormat = 'Format0'` and `scistr.FreqHopping = 1`)

Data Types: `double`

**`NSubbands` — Number of subbands**
1 (default) | 2 | 4 | optional

Number of subbands, specified as 1, 2, or 4. (*SL-HoppingConfigComm-r12 {numSubbands-r12}*)

**Note** This parameter is required for SCI format 0 and frequency hopping. (`scistr.SCIFormat = 'Format0'` and `scistr.FreqHopping = 1`)

Data Types: `double`

**`PSSCHHoppingOffset` — PSSCH hopping offset**
0 (default) | integer scalar from 0 to 110 | optional

PSSCH hopping offset, specified as an integer scalar from 0 to 110. (*SL-HoppingConfigComm-r12 {rb-Offset-r12}*)

**Note** This parameter is required for SCI format 0 and frequency hopping. (`scistr.SCIFormat = 'Format0'` and `scistr.FreqHopping = 1`)

Data Types: `double`

**`PRBPool` — PSSCH resource block pool**
optional | zero-based integer vector | optional

PSSCH resource block pool (sidelink transmission mode 2), specified as a zero-based integer vector of indices giving the PRBs in the pool. If `PRBPool` is absent or empty, the pool is assumed to be the full transmission bandwidth.

**Note** This parameter is required for SCI format 0 and frequency hopping. (`scistr.SCIFormat = 'Format0'` and `scistr.FreqHopping = 1`)

Data Types: `double`

**`PSSCHNSubchannels` — Number of sub-channels in the V2X PSSCH resource pool**
`1` (default) | integer scalar from 1 to 110 | optional

Number of sub-channels in the V2X PSSCH resource pool, specified as an integer scalar from 1 to 110.

**Note** This parameter is required for SCI format 1. (`scistr.SCIFormat = 'Format1'`)

Data Types: `double`

**`PSSCHSubchannelsSize` — Number of PRB in each sub-channel**
`4` (default) | integer scalar from 1 to 110 | optional

Number of PRB in each sub-channel, specified as an integer scalar from 1 to 110.

**Note** This parameter is required for SCI format 1. (`scistr.SCIFormat = 'Format1'`)

Data Types: `double`

**`PSSCHSubchannelsPRBStart` — First PRB index associated with first sub-channel of the resource pool**
`0` (default) | integer scalar from 1 to 109 | optional

First PRB index associated with first sub-channel of the resource pool, specified as an integer scalar from 1 to 109.

**Note** This parameter is required for SCI format 1. (`scistr.SCIFormat = 'Format1'`)

Data Types: `double`

**`PSSCHAdjacency` — Whether PSCCH and PSSCH are transmitted in adjacent PRB**
`'On'` (default) | `'Off'` | optional

Whether PSCCH and PSSCH are transmitted in adjacent PRB, specified as `'On'` or `'Off'`.

**Note** This parameter is required for SCI format 1. (`scistr.SCIFormat = 'Format1'`)

Data Types: `double`

**`FirstSubchannelIdx` — First sub-channel index of PSSCH resource allocation**
`0` (default) | integer scalar from 1 to 109 | optional

First sub-channel index of PSSCH resource allocation, specified as an integer scalar from 1 to 109.

**Note** This parameter is required for SCI format 1. (`scistr.SCIFormat = 'Format1'`)

Data Types: `double`

Data Types: `struct`

**`scistr` — Sidelink control information settings**
structure

Sidelink control information settings, specified as a parameter structure containing these PRB allocation fields:

**`SCIFormat` — SCI format type**
`'Format0'` (default) | `'Format1'`

SCI format type, specified as `'Format0'` or `'Format1'`.

Data Types: `char` | `string`

**`FreqHopping` — Frequency hopping flag**
0 (default) | 1

Frequency hopping flag, specified as 0 for nonhopping allocation type or 1 for hopping allocation type. When `scistr.FreqHopping` = 1, the hopping allocation type is signalled by `scistr.Allocation.HoppingBits`.

**Note** This parameter is required for SCI format 0. (`scistr.SCIFormat = 'Format0'`)

Data Types: `double`

**`Allocation` — Resource allocation parameter substructure**
structure | optional

Resource allocation parameter substructure, specified as a structure.

**`HoppingBits` — Hopping bits**
0 (default) | bit vector with 0, 1, or 2 bits

Hopping bits, specified as a bit vector with 0, 1, or 2 bits. The `HoppingBits` parameter signals the hopping type. For more information, see “SCI Resource Allocation” on page 2-1064.

**Note** This parameter is required for SCI format 0. (`scistr.SCIFormat = 'Format0'`)

Data Types: `double`

**`RIV` — Resource indication value**
0 (default) | bit vector with 5 to 13 bits

Resource indication value, specified as a bit vector with 5 to 13 bits. The resource indication value assignment for sidelink follows the specifications for uplink, as modified in TS 36.213 [2], Sections 14.1.1.2 and 14.1.1.4. For more information, see “SCI Resource Allocation” on page 2-1064.

**Note** This parameter is required for SCI format 0. (`scistr.SCIFormat = 'Format0'`)

Data Types: `double`

Data Types: `struct`

**`RIV` — Resource indication value**
bit vector with 0 to 13 bits | optional

Resource indication value, specified as a bit vector with 0 to 13 bits. The resource indication value assignment for sidelink follows the specifications for uplink, as modified in TS 36.213 [2], Sections 14.1.1.2 and 14.1.1.4. For more information, see “SCI Resource Allocation” on page 2-1064.

**Note** This parameter is required for SCI format 1. (`scistr.SCIFormat = 'Format1'`)

Data Types: `double`

Data Types: `struct`

## Output Arguments

**`prbset` — Physical resource block indices**
nonnegative integer column vector | nonnegative integer column matrix

Physical resource block indices, returned as a nonnegative integer column vector or *N*-by-2 integer matrix of zero-based indices.

- When the allocation type defines one set of PRB indices to use in the first and second slots of the subframe, `prbset` is returned as an integer column vector.
- When the allocation type defines a different set of PRB indices in the first and second slots of the subframe, `prbset` is returned as two-column integer matrix.

The PRB indices created are for a single PSSCH transmission in a subframe within the PSSCH subframe pool.

## More About

### SCI Resource Allocation

Sidelink control information (SCI) resource allocation mapping is described in TS 36.211 [1], Section 9.3.6. The `sciout` structure returned by `lteSCI` can be directly used as the `scistr` structure input to `lteSCIResourceAllocation`. Using `lteSCI` creates a properly formatted SCI format 0 message, ensuring that the field values adhere to the underlying field bit lengths. The `scistr` field values are read modulo to the SCI message bit lengths. Any fields missing from `scistr` default to 0. PSSCH allocations are based on uplink resource allocation type 0 (see `lteDCI`, DCI format 0). In these allocations, the same single contiguous PRB allocation must be used for both slots in the subframe. As with uplink, for sidelink:

- A `FreqHopping` value of 1 signals a hopping allocation type. There are two types of hopping: type 1 PUSCH hopping and type 2 PUSCH hopping (frequency hopping with a predefined pattern). `scistr.Allocation.HoppingBits` signals the hopping type, as specified in TS 36.213 [2], Table 8.4-2.
- A `FreqHopping` value of 0 signals a nonhopping allocation type

Alternatively, you can use `lteDCIResourceAllocation` with a DCI format 5 message and the same message fields to generate the PSSCH allocations. This PSSCH allocation represents sidelink transmission mode 1, with the eNodeB using a DCI format 5 message to provide the transmitting UE with a PSSCH resource allocation.

# Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.213. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteSCI` | `ltePSCCH` | `lteSLSCH` | `lteDCIResourceAllocation`

# lteSLBCH

Sidelink broadcast channel

## Syntax

```
cw = lteSLBCH(ue,trblk)
```

## Description

`cw = lteSLBCH(ue,trblk)` returns a column vector of sidelink broadcast channel (SL-BCH) transport channel coded bits for the specified UE settings structure and transport block payload. The encoding process includes 16-bit CRC calculation and attachment, tail-biting convolutional encoding, rate matching, and PUSCH interleaving, as defined in TS 36.212 [1], Section 5.4.1. This transport channel carries the `lteSLMIB` RRC message. The sidelink BCH codeword output, `cw`, is ready for transmission on the physical sidelink broadcast channel using `ltePSBCH`.

## Examples

### Generate SL-BCH Codeword

Generate an SL-BCH coded vector of length 1152, corresponding to the SL-BCH codeword for normal cyclic prefix.

Create UE-specific configuration structure with 10 MHz bandwidth and normal cyclic prefix.

```
ue.NSLRB = 50;
ue.CyclicPrefixSL = 'Normal';
```

Generate the MIB-SL transport block and SL-BCH codeword.

```
slmib = lteSLMIB(ue);
slbchCodeword = lteSLBCH(ue,slmib);
```

## Input Arguments

### ue — User equipment settings

structure

User equipment settings, specified as a parameter structure containing the following fields:

#### SidelinkMode — Sidelink mode

`'D2D'` (default) | `'V2X'` | optional

Sidelink mode, specified as `'D2D'` or `'V2X'`.

Data Types: `char` | `string`

#### CyclicPrefixSL — Cyclic prefix length

`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

Data Types: `struct`

**`trblk` — Transport block**
40-bit vector

Transport block, specified as a 40-bit vector containing MIB-SL information bits. These bits are delivered at the input to the SL-BCH transport channel.

Data Types: `double` | `int8` | `logical`

## Output Arguments

**`cw` — Codeword representing the MIB-SL information bits**
binary-valued column vector

Codeword representing the MIB-SL information bits, returned as a binary-valued column vector. For D2D sidelink mode, this vector has length 1152 for normal cyclic prefix or 864 for extended cyclic prefix. For V2X PSBCH, this vector has length 1008, defined for normal cyclic prefix only. If the input MIB-SL message is empty, the function returns this output as an empty 0-by-1 matrix. The output codeword matches the bit capacity available in the PSBCH. The PSBCH bit capacity is based on the specified cyclic prefix setting and does not account for the sidelink SC-FDMA guard symbol. For more information, see `ltePSBCHIndices`.

Data Types: `int8`

# Version History
**Introduced in R2016b**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also
`lteSLBCHDecode` | `ltePSBCH` | `lteBCH` | `lteSLMIB` | `ltePSBCHIndices`

# lteSLBCHDecode

Sidelink broadcast channel decoding

## Syntax

```
[trblkout,crcerr] = lteSLBCHDecode(ue,softbits)
```

## Description

`[trblkout,crcerr] = lteSLBCHDecode(ue,softbits)` returns a 40-by-1 column vector of information bits and the cyclic redundancy check (CRC) result for the specified UE settings structure and recovered soft bits.

The SL-BCH decoder performs the inverse of the sidelink broadcast channel processing performed by `lteSLBCH`, and as defined in TS 36.212 [1], Section 5.4.1. The decoding operation includes PUSCH deinterleaving, rate recovery, tail-biting convolutional decoding, and CRC decoding.

## Examples

### Decode SL-BCH Codeword

Decode a sidelink broadcast channel (SL-BCH) codeword.

Create a UE-specific configuration structure with normal cyclic prefix.

```
ue.CyclicPrefixSL = 'Normal';
```

Generate an SL-BCH codeword by using an MIB-SL transport block of all ones. Display the CRC result.

```
trblk = ones(40,1);
slbchCoded = lteSLBCH(ue,trblk);
[slbchDecoded,err] = lteSLBCHDecode(ue,slbchCoded);
err
```

```
err = uint32
    0
```

The CRC result indicates no error. `isequal` reconfirms the decoded output matches the input transport block.

```
isequal(slbchDecoded,trblk)
```

```
ans = logical
   1
```

## Input Arguments

### `ue` — User equipment settings

structure

User equipment settings, specified as a parameter structure containing these fields:

#### `SidelinkMode` — Sidelink mode

`'D2D'` (default) | `'V2X'` | optional

Sidelink mode, specified as `'D2D'` or `'V2X'`.

Data Types: `char` | `string`

#### `CyclicPrefixSL` — Cyclic prefix length

`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

Data Types: `struct`

### `softbits` — Log-likelihood ratio soft bits

vector

Log-likelihood ratio (LLR) soft bits, specified as a vector. Nominally, `softbits` contains 1152 bits for normal cyclic prefix, 864 bits extended cyclic prefix, or 1008 bits for V2X. These lengths match the bit capacity of the PSBCH, ignoring the SC-FDMA guard symbol.

Because PSBCH uses a low code rate and the decoder can successfully decode much shorter blocks than the entire coded block, input `softbits` can be any length.

Data Types: `double`

## Output Arguments

### `trblkout` — Transport block

40-by-1 column bit vector

Transport block, returned as a 40-by-1 column bit vector representing the MIB-SL information bits sent by a transmitting UE on the SL-BCH transport channel. The MIB-SL information bits are decoded from the soft log-likelihood (LLR) codeword data.

Data Types: `int8`

### `crcerr` — CRC error status

0 | 1

CRC error status, returned as 0 for a pass and 1 for a block error.

Data Types: `uint32`

# Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteSLBCH` | `ltePSBCHDecode` | `lteSLMIB` | `lteBCHDecode`

# lteSLChannelEstimatePSBCH

PSBCH sidelink channel estimation

## Syntax

```
[hest] = lteSLChannelEstimatePSBCH(ue,rxgrid)
[hest] = lteSLChannelEstimatePSBCH(ue,cec,rxgrid)
[hest,noiseest] = lteSLChannelEstimatePSBCH( ___ )
```

## Description

`[hest] = lteSLChannelEstimatePSBCH(ue,rxgrid)` returns an estimate for the channel by averaging the least squares estimates of the reference symbols across time and copying these estimates across the allocated resource elements within the time frequency grid. The channel estimation configuration uses the method described in TS 36.101 [1], Annex F.

`[hest] = lteSLChannelEstimatePSBCH(ue,cec,rxgrid)` also accepts the channel estimator configuration structure, `cec`, to adjust the default method and parameters defined for estimating the channel.

`[hest,noiseest] = lteSLChannelEstimatePSBCH( ___ )` also returns an estimate of the noise power spectral density for the channel. This syntax supports input options from prior syntaxes.

## Examples

### Estimate Channel Using PSBCH DM-RS and Default CE Settings

Estimate the channel characteristics given the PSBCH-received resource grid containing PSBCH DM-RS symbols. Use the default channel estimation configuration method, as defined in TS 36.101, Annex F.

#### Create Parameter Structure

Define UE-specific settings in a parameter structure.

```
ue = struct('NSLRB',50,'CyclicPrefixSL','Normal','NSLID',1);
```

#### Populate Subframe with PSBCH Symbols

Create the subframe grid and indices for the subframe. Create broadcast channel and demodulation reference symbols and populate the subframe.

```
subframe = lteSLResourceGrid(ue);
psbchIndices = ltePSBCHIndices(ue);
psbchdmrsIndices = ltePSBCHDRSIndices(ue);
psbchSymbols = ltePSBCH(ue,lteSLBCH(ue,zeros(40,1)));
subframe(psbchIndices) = psbchSymbols;
subframe(psbchdmrsIndices) = ltePSBCHDRS(ue);
```

### Estimate Channel Characteristics

Use the received resource grid containing PSBCH DM-RS symbols to estimate the channel characteristics.

- Perform sidelink SC-FDMA modulation.
- No channel impairment is applied, so set the received waveform equal to the transmit waveform.
- Perform sidelink SC-FDMA demodulation and channel estimation.

```
txWaveform = lteSLSCFDMAModulate(ue,subframe);
rxWaveform = txWaveform;
rxGrid = lteSLSCFDMADemodulate(ue,rxWaveform);
hest = lteSLChannelEstimatePSBCH(ue,rxGrid);
```

### Estimate Channel Using PSBCH DM-RS

Estimate the channel characteristics given the PSBCH-received resource grid containing PSBCH DM-RS symbols. The default channel estimation configuration is adjusted.

#### Create parameter structures

Define UE-specific settings and channel estimation configuration settings in parameter structures.

```
ue = struct('NSLRB',50,'CyclicPrefixSL','Normal','NSLID',1);
cec = struct('FreqWindow',7,'TimeWindow',1,'InterpType','cubic','PilotAverage','UserDefined');
```

#### Populate a subframe with PSBCH symbols

Create the subframe grid and indices for the subframe. Create broadcast channel and demodulation reference (DM-RS) symbols.

```
subframe = lteSLResourceGrid(ue);
psbchIndices = ltePSBCHIndices(ue);

psbchSymbols = ltePSBCH(ue,lteSLBCH(ue,zeros(40,1)));

subframe(psbchIndices) = psbchSymbols;
subframe(ltePSBCHDRSIndices(ue)) = ltePSBCHDRS(ue);
```

#### Estimate the channel characteristics

Use the received resource grid containing PSBCH DM-RS symbols to estimate the channel characteristics.

- Perform sidelink SC-FDMA modulation.
- No channel impairment is applied, so set the received waveform equal to the transmit waveform.
- Perform sidelink SC-FDMA demodulation and channel estimation.

```
txWaveform = lteSLSCFDMAModulate(ue,subframe);

rxWaveform = txWaveform;
```

```
rxGrid = lteSLSCFDMADemodulate(ue,rxWaveform);
hest = lteSLChannelEstimatePSBCH(ue,cec,rxGrid);
```

### Estimate Channel and Noise Using PSBCH DM-RS

Estimate the channel characteristics and noise power spectral density given the PSBCH-received resource grid containing PSBCH DM-RS symbols.

### Create Parameter Structures

Define UE-specific settings and channel estimation configuration settings in parameter structures.

```
ue = struct('NSLRB',50,'CyclicPrefixSL','Normal','NSLID',1);
cec = struct('FreqWindow',7,'TimeWindow',1,'InterpType','cubic','PilotAverage','UserDefined');
```

### Populate Subframe with PSBCH Symbols

Create the subframe grid and indices for the subframe. Create broadcast channel and demodulation reference symbols.

```
subframe = lteSLResourceGrid(ue);
psbchIndices = ltePSBCHIndices(ue);

psbchSymbols = ltePSBCH(ue,lteSLBCH(ue,zeros(40,1)));
subframe(psbchIndices) = psbchSymbols;

subframe(ltePSBCHDRSIndices(ue)) = ltePSBCHDRS(ue);
```

### Estimate Channel Characteristics

Estimate the channel characteristics by using the received resource grid containing PSBCH DM-RS symbols.

- Perform sidelink SC-FDMA modulation
- Add noise to the transmitted signal
- Perform sidelink SC-FDMA demodulation and channel estimation
- View the noise estimate

```
txWaveform = lteSLSCFDMAModulate(ue,subframe);

rxWaveform = awgn(txWaveform,15,'measured');

rxGrid = lteSLSCFDMADemodulate(ue,rxWaveform);
[hest,noiseEst] = lteSLChannelEstimatePSBCH(ue,cec,rxGrid);

disp(noiseEst)
```

```
   8.7693e-04
```

## Input Arguments

### ue — UE-specific settings

structure

User equipment settings, specified as a structure containing these fields.

**`SidelinkMode` — Sidelink mode**
`'D2D'` (default) | `'V2X'` | optional

Sidelink mode, specified as `'D2D'` or `'V2X'`.

Data Types: `char` | `string`

**`NSLRB` — Number of sidelink resource blocks**
integer scalar from 6 to 110

Number of sidelink resource blocks, specified as an integer scalar from 6 to 110.

Example: `6`, which corresponds to a channel bandwidth of 1.4 MHz.

Data Types: `double`

**`CyclicPrefixSL` — Cyclic prefix length**
`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

**`NSLID` — Physical layer sidelink synchronization identity**
integer from 0 to 355

Physical layer sidelink synchronization identity, specified as an integer from 0 to 355. ($N_{ID}^{SL}$)

Data Types: `double`

Data Types: `struct`

**`rxgrid` — Received resource element grid**
3-D array

Received resource element grid, specified as an $N_{SC}$-by-$N_{Sym}$-by-$N_R$ array of complex symbols.

- $N_{SC}$ is the number of subcarriers.
- $N_{Sym} = N_{SF} \times N_{SymPerSF} = 1 \times N_{SymPerSF}$
  - $N_{SF}$ is the total number of subframes. For this function `rxgrid` must contain one subframe.
  - $N_{SymPerSF}$ is the number of SC-FDMA symbols per subframe.
    - For normal cyclic prefix, a subframe contains 14 SC-FDMA symbols.
    - For extended cyclic prefix, a subframe contains 12 SC-FDMA symbols.
  - $N_R$ is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

**`cec` — PSBCH channel estimation settings**
structure

PSBCH channel estimation settings, specified as a structure that can contain these fields.

**`FreqWindow` — Size of frequency window**
integer

Size of frequency window, specified as an integer that is odd or a multiple of 12. `FreqWindow` is the number of resource elements (REs) used to average over frequency.

Data Types: `double`

**`TimeWindow` — Size of time window**
integer

Size of time window, specified as an odd integer. `TimeWindow` is the number of resource elements (REs) used to average over time.

Data Types: `double`

**`InterpType` — Type of 2-D interpolation**
`'nearest'` | `'linear'` | `'natural'` | `'cubic'` | `'v4'` | `'none'`

Type of 2-D interpolation used during interpolation, specified as one of these supported choices.

| Value | Description |
|---|---|
| `'nearest'` | Nearest neighbor interpolation |
| `'linear'` | Linear interpolation |
| `'natural'` | Natural neighbor interpolation |
| `'cubic'` | Cubic interpolation |
| `'v4'` | MATLAB 4 `griddata` method |
| `'none'` | Disables interpolation |

For details, see `griddata`.

Data Types: `char` | `string`

**`PilotAverage` — Type of pilot averaging**
`'UserDefined'` (default) | `'TestEVM'` | optional

Type of pilot averaging, specified as `'UserDefined'` or `'TestEVM'`.

The `'UserDefined'` pilot averaging uses a rectangular kernel of size `cec.FreqWindow`-by-`cec.TimeWindow` and performs a 2-D filtering operation on the pilots. Pilots near the edge of the resource grid are averaged less because they have no neighbors outside of the grid.

For `cec.FreqWindow` = $12\times X$ (that is, any multiple of 12) and `cec.TimeWindow` = 1, the estimator enters a special case where an averaging window of $(12\times X)$-in-frequency is used to average the pilot estimates. The averaging is always applied across $(12\times X)$ subcarriers, even at the upper and lower band edges. Therefore, the first $(6\times X)$ symbols at the upper and lower band edge have the same channel estimate. This operation ensures that averaging is always done on 12 (or a multiple of 12) symbols. The `'TestEVM'` pilot averaging ignores other structure fields in `cec`, and for the transmitter EVM testing, it follows the method described in TS 36.101, Annex F.

Data Types: `char` | `string`

Data Types: `struct`

## Output Arguments

### `hest` — Channel estimate between each transmit and receive antenna

3-D array

Channel estimate between each transmit and receive antenna, returned as an $N_{SC}$-by-$N_{Sym}$-by-$N_R$ array of complex symbols. $N_{SC}$ is the total number of subcarriers, $N_{Sym}$ is the number of SC-FDMA symbols, and $N_R$ is the number of receive antennas.

For `cec.InterpType = 'none'`,

- No interpolation between the pilot symbol estimates is performed and no virtual pilots are created
- `hest` contains channel estimates in the locations of transmitted DM-RS symbols for each received antenna and all other elements of `hest` are `0`
- The averaging of pilot symbol estimates, described by `cec.TimeWindow` and `cec.FreqWindow`, is still performed

### `noiseest` — Noise estimate

numeric scalar

Noise estimate, returned as a numeric scalar. When `cec.PilotAverage` is `'UserDefined'`, this output is the power spectral density of the noise present on the estimated channel response coefficients. Otherwise, `noiseest` returns `0`.

# Version History

**Introduced in R2017a**

## References

[1] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteSLChannelEstimatePSCCH` | `lteSLChannelEstimatePSSCH`

# lteSLChannelEstimatePSCCH

PSCCH sidelink channel estimation

## Syntax

```
[hest] = lteSLChannelEstimatePSCCH(ue,rxgrid)
[hest] = lteSLChannelEstimatePSCCH(ue,cec,rxgrid)
[hest,noiseest] = lteSLChannelEstimatePSCCH( ___ )
```

## Description

`[hest] = lteSLChannelEstimatePSCCH(ue,rxgrid)` returns an estimate for the channel by averaging the least squares estimates of the reference symbols across time and copying these estimates across the allocated resource elements within the time frequency grid. The channel estimation configuration uses the method described in TS 36.101 [1], Annex F.

`[hest] = lteSLChannelEstimatePSCCH(ue,cec,rxgrid)` also accepts the channel estimator configuration structure, `cec`, to adjust the default method and parameters defined for estimating the channel.

`[hest,noiseest] = lteSLChannelEstimatePSCCH( ___ )` also returns an estimate of the noise power spectral density for the channel. This syntax supports input options from prior syntaxes.

## Examples

### Estimate Channel Using PSCCH DM-RS and Default CE Settings

Estimate the channel characteristics given the PSCCH-received resource grid containing PSCCH DM-RS symbols. Use the default channel estimation configuration method, as defined in TS 36.101, Annex F.

Create a structure defining UE-specific settings.

```
ue = struct('NSLRB',25,'CyclicPrefixSL','Normal','PRBSet',5);
```

Create the subframe grid, control channel, and indices for a subframe. Populate the subframe with PSCCH symbols.

```
subframe = lteSLResourceGrid(ue);

[pscchIndices,pscchInfo] = ltePSCCHIndices(ue);
pscchSymbols = ltePSCCH(randi([0 1],pscchInfo.G,1));

subframe(pscchIndices) = pscchSymbols;
```

Create the control DM-RS and indices. Add the PSCCH DM-RS symbols to the subframe.

```
subframe(ltePSCCHDRSIndices(ue)) = ltePSCCHDRS;
```

Perform sidelink SC-FDMA modulation.

```
txWaveform = lteSLSCFDMAModulate(ue,subframe);
```

No channel impairment is applied, so set the received waveform equal to the transmit waveform. Perform sidelink SC-FDMA demodulation and channel estimation.

```
rxWaveform = txWaveform;

rxGrid = lteSLSCFDMADemodulate(ue,rxWaveform);
hest = lteSLChannelEstimatePSCCH(ue,rxGrid);
```

### Estimate Channel Using PSCCH DM-RS

Estimate the channel characteristics given the PSCCH-received resource grid containing PSCCH DM-RS symbols. The default channel estimation configuration is adjusted.

Create structures defining UE-specific settings and channel estimation configuration settings.

```
ue = struct('NSLRB',50,'CyclicPrefixSL','Normal','PRBSet',5);

cec = struct('FreqWindow',7,'TimeWindow',1,'InterpType','cubic', ...
    'PilotAverage','UserDefined');
```

Create the subframe grid, control channel, and indices for a subframe. Populate the subframe with PSCCH symbols.

```
subframe = lteSLResourceGrid(ue);

[pscchIndices,pscchInfo] = ltePSCCHIndices(ue);
pscchSymbols = ltePSCCH(randi([0 1],pscchInfo.G,1));

subframe(pscchIndices) = pscchSymbols;
```

Create the control DM-RS and indices. Add the PSCCH DM-RS symbols to the subframe.

```
subframe(ltePSCCHDRSIndices(ue)) = ltePSCCHDRS;
```

Perform sidelink SC-FDMA modulation.

```
txWaveform = lteSLSCFDMAModulate(ue,subframe);
```

No channel impairment is applied, so set the received waveform equal to the transmit waveform. Perform sidelink SC-FDMA demodulation and channel estimation.

```
rxWaveform = txWaveform;

rxGrid = lteSLSCFDMADemodulate(ue,rxWaveform);
hest = lteSLChannelEstimatePSCCH(ue,cec,rxGrid);
```

### Estimate Channel and Noise Using PSCCH DM-RS

Estimate the channel characteristics and noise power spectral density given the PSCCH-received resource grid containing PSCCH DM-RS symbols.

Create structures defining UE-specific and channel estimation configuration settings.

```
ue = struct('NSLRB',25,'CyclicPrefixSL','Normal','PRBSet',5);
cec = struct('FreqWindow',7,'TimeWindow',1,'InterpType','cubic', ...
    'PilotAverage','UserDefined');
```

Create the subframe grid, control channel, and indices for a subframe. Populate the subframe with PSCCH symbols.

```
subframe = lteSLResourceGrid(ue);

[pscchIndices,pscchInfo] = ltePSCCHIndices(ue);
pscchSymbols = ltePSCCH(randi([0 1],pscchInfo.G,1));

subframe(pscchIndices) = pscchSymbols;
```

Create the control DM-RS and indices. Add the PSCCH DM-RS symbols to the subframe.

```
subframe(ltePSCCHDRSIndices(ue)) = ltePSCCHDRS;
```

Perform sidelink SC-FDMA modulation.

```
txWaveform = lteSLSCFDMAModulate(ue,subframe);
```

Add noise to impair the channel. Perform sidelink SC-FDMA demodulation and channel estimation. View the noise estimate.

```
rxWaveform = awgn(txWaveform,15,'measured');

rxGrid = lteSLSCFDMADemodulate(ue,rxWaveform);
[hest,noiseest] = lteSLChannelEstimatePSCCH(ue,cec,rxGrid);
noiseest
```

```
noiseest = 4.3822e-04
```

## Input Arguments

### ue — UE-specific settings

structure

User equipment settings, specified as a structure containing these fields.

### SidelinkMode — Sidelink mode

`'D2D'` (default) | `'V2X'` | optional

Sidelink mode, specified as `'D2D'` or `'V2X'`.

Data Types: `char` | `string`

### NSLRB — Number of sidelink resource blocks

integer scalar from 6 to 110

Number of sidelink resource blocks, specified as an integer scalar from 6 to 110.

Example: 6, which corresponds to a channel bandwidth of 1.4 MHz.

Data Types: `double`

**`CyclicPrefixSL` — Cyclic prefix length**
`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

**`PRBSet` — Zero-based physical resource block index**
integer | integer column vector | two-column integer matrix

Zero-based physical resource block (PRB) index, specified as an integer, an integer column vector, or a two-column integer matrix.

For D2D sidelink, the PSCCH is intended to be transmitted in a single PRB in a subframe and therefore, specifying `PRBSet` as a scalar PRB index is recommended. For V2X sidelink, the PSCCH is intended to be transmitted in a pair of consecutive PRB in a subframe, therefore `PRBSet` must be a column vector containing two consecutive indices. However, for a more general nonstandard multi-PRB allocation, `PRBSet` can be a set of indices specified as an integer column vector or as a two-column integer matrix corresponding to slot-wise resource allocations for PSCCH.

Data Types: `double`

**`CyclicShift` — Cyclic shift for DM-RS**
`0` (default) | `3` | `6` | `9`

Cyclic shift for DM-RS, specified as `0`, `3`, `6` or `9`. The function uses this input only for V2X sidelink.

Data Types: `double`

Data Types: `struct`

**`rxgrid` — Received resource element grid**
3-D array

Received resource element grid, specified as an $N_{SC}$-by-$N_{Sym}$-by-$N_R$ array of complex symbols.

- $N_{SC}$ is the number of subcarriers.
- $N_{Sym} = N_{SF} \times N_{SymPerSF} = 1 \times N_{SymPerSF}$
  - $N_{SF}$ is the total number of subframes. For this function `rxgrid` must contain one subframe.
  - $N_{SymPerSF}$ is the number of SC-FDMA symbols per subframe.
    - For normal cyclic prefix, a subframe contains 14 SC-FDMA symbols.
    - For extended cyclic prefix, a subframe contains 12 SC-FDMA symbols.
  - $N_R$ is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

**`cec` — PSCCH channel estimation settings**
structure

PSCCH channel estimation settings, specified as a structure that can contain these fields.

**`FreqWindow` — Size of frequency window**
integer

Size of frequency window, specified as an integer that is odd or a multiple of 12. `FreqWindow` is the number of resource elements (REs) used to average over frequency.

Data Types: `double`

**`TimeWindow` — Size of time window**
integer

Size of time window, specified as an odd integer. `TimeWindow` is the number of resource elements (REs) used to average over time.

Data Types: `double`

**`InterpType` — Type of 2-D interpolation**
`'nearest'` | `'linear'` | `'natural'` | `'cubic'` | `'v4'` | `'none'`

Type of 2-D interpolation used during interpolation, specified as one of these supported choices.

| Value | Description |
|---|---|
| `'nearest'` | Nearest neighbor interpolation |
| `'linear'` | Linear interpolation |
| `'natural'` | Natural neighbor interpolation |
| `'cubic'` | Cubic interpolation |
| `'v4'` | MATLAB 4 `griddata` method |
| `'none'` | Disables interpolation |

For details, see `griddata`.

Data Types: `char` | `string`

**`PilotAverage` — Type of pilot averaging**
`'UserDefined'` (default) | `'TestEVM'` | optional

Type of pilot averaging, specified as `'UserDefined'` or `'TestEVM'`.

The `'UserDefined'` pilot averaging uses a rectangular kernel of size `cec.FreqWindow`-by-`cec.TimeWindow` and performs a 2-D filtering operation on the pilots. Pilots near the edge of the resource grid are averaged less because they have no neighbors outside of the grid.

For `cec.FreqWindow` = 12×$X$ (that is, any multiple of 12) and `cec.TimeWindow` = 1, the estimator enters a special case where an averaging window of (12×$X$)-in-frequency is used to average the pilot estimates. The averaging is always applied across (12×$X$) subcarriers, even at the upper and lower band edges. Therefore, the first (6×$X$) symbols at the upper and lower band edge have the same channel estimate. This operation ensures that averaging is always done on 12 (or a multiple of 12) symbols. The `'TestEVM'` pilot averaging ignores other structure fields in `cec`, and for the transmitter EVM testing, it follows the method described in TS 36.101, Annex F.

Data Types: `char` | `string`

Data Types: `struct`

## Output Arguments

### `hest` — Channel estimate between each transmit and receive antenna

3-D array

Channel estimate between each transmit and receive antenna, returned as an $N_{SC}$-by-$N_{Sym}$-by-$N_R$ array of complex symbols. $N_{SC}$ is the total number of subcarriers, $N_{Sym}$ is the number of SC-FDMA symbols, and $N_R$ is the number of receive antennas.

For `cec.InterpType = 'none'`,

- No interpolation between the pilot symbol estimates is performed and no virtual pilots are created
- `hest` contains channel estimates in the locations of transmitted DM-RS symbols for each received antenna and all other elements of `hest` are `0`
- The averaging of pilot symbol estimates, described by `cec.TimeWindow` and `cec.FreqWindow`, is still performed

### `noiseest` — Noise estimate

numeric scalar

Noise estimate, returned as a numeric scalar. When `cec.PilotAverage` is `'UserDefined'`, this output is the power spectral density of the noise present on the estimated channel response coefficients. Otherwise, `noiseest` returns `0`.

# Version History

**Introduced in R2017a**

## References

[1] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteSLChannelEstimatePSBCH` | `lteSLChannelEstimatePSSCH`

# lteSLChannelEstimatePSSCH

PSSCH sidelink channel estimation

## Syntax

```
[hest] = lteSLChannelEstimatePSSCH(ue,rxgrid)
[hest] = lteSLChannelEstimatePSSCH(ue,cec,rxgrid)
[hest,noiseest] = lteSLChannelEstimatePSSCH( ___ )
```

## Description

`[hest] = lteSLChannelEstimatePSSCH(ue,rxgrid)` returns an estimate for the channel by averaging the least squares estimates of the reference symbols across time and copying these estimates across the allocated resource elements within the time frequency grid. The channel estimation configuration uses the method described in TS 36.101 [1], Annex F.

`[hest] = lteSLChannelEstimatePSSCH(ue,cec,rxgrid)` also accepts the channel estimator configuration structure, `cec`, to adjust the default method and parameters defined for estimating the channel.

`[hest,noiseest] = lteSLChannelEstimatePSSCH( ___ )` also returns an estimate of the noise power spectral density for the channel. This syntax supports input options from prior syntaxes.

## Examples

### Estimate Channel Using PSSCH DM-RS and Default CE Settings

Estimate the channel characteristics given the PSSCH-received resource grid containing PSSCH DM-RS symbols. Use the default channel estimation configuration method, as defined in TS 36.101, Annex F.

#### Configure UE Settings

Define UE-specific settings in a parameter structure.

```
ue = struct('NSLRB',50,'CyclicPrefixSL','Normal','NSAID',255, ...
            'Modulation','QPSK','NSubframePSSCH',0,'PRBSet',(30:39)');
```

#### Populate Subframe with PSSCH Symbols

Create the subframe grid and indices for the subframe. Create shared channel and demodulation reference signal (DM-RS) symbols. Populate the subframe with the shared channel and DM-RS symbols.

```
subframe = lteSLResourceGrid(ue);
[psschIndices,psschInfo] = ltePSSCHIndices(ue);

psschSymbols = ltePSSCH(ue,zeros(psschInfo.G,1));
subframe(psschIndices) = psschSymbols;
```

```
subframe(ltePSSCHDRSIndices(ue)) = ltePSSCHDRS(ue);
```

### Estimate Channel Characteristics

Estimate the channel characteristics by using the received resource grid containing PSSCH DM-RS symbols.

- Perform sidelink SC-FDMA modulation
- No channel impairment is applied, so set the received waveform equal to the transmit waveform
- Perform sidelink SC-FDMA demodulation and channel estimation

```
txWaveform = lteSLSCFDMAModulate(ue,subframe);

rxWaveform = txWaveform;

rxGrid = lteSLSCFDMADemodulate(ue,rxWaveform);
hest = lteSLChannelEstimatePSSCH(ue,rxGrid);
```

## Estimate Channel Using PSSCH DM-RS

Estimate the channel characteristics given the PSSCH-received resource grid containing PSSCH DM-RS symbols.

### Create Parameter Structures

Define UE-specific settings and channel estimation configuration settings in parameter structures.

```
ue = struct('NSLRB',50,'CyclicPrefixSL','Normal','NSAID',255, ...
           'Modulation','QPSK','NSubframePSSCH',0,'PRBSet',(30:39)');
cec = struct('FreqWindow',7,'TimeWindow',1,'InterpType','cubic', ...
            'PilotAverage','UserDefined');
```

### Populate Subframe with PSSCH Symbols

Create the subframe grid and indices for the subframe. Create shared channel and demodulation reference signal (DM-RS) symbols. Populate the subframe with shared channel and DM-RS symbols.

```
subframe = lteSLResourceGrid(ue);
[psschIndices,psschInfo] = ltePSSCHIndices(ue);

psschSymbols = ltePSSCH(ue,zeros(psschInfo.G,1));
subframe(psschIndices) = psschSymbols;

subframe(ltePSSCHDRSIndices(ue)) = ltePSSCHDRS(ue);
```

### Estimate Channel Characteristics

Estimate the channel characteristics by using the received resource grid containing PSSCH DM-RS symbols.

- Perform sidelink SC-FDMA modulation
- No channel impairment is applied, so set the received waveform equal to the transmit waveform
- Perform sidelink SC-FDMA demodulation and channel estimation

```
txWaveform = lteSLSCFDMAModulate(ue,subframe);

rxWaveform = txWaveform;

rxGrid = lteSLSCFDMADemodulate(ue,rxWaveform);
hest = lteSLChannelEstimatePSSCH(ue,cec,rxGrid);
```

### Estimate Channel and Noise Using PSSCH DM-RS

Estimate the channel characteristics and noise power spectral density given the PSSCH-received resource grid containing PSSCH DM-RS symbols.

#### Create Parameter Structures

Define UE-specific settings and channel estimation configuration settings in parameter structures.

```
ue = struct('NSLRB',50,'CyclicPrefixSL','Normal','NSAID',255, ...
     'Modulation','QPSK','NSubframePSSCH',0,'PRBSet',(30:39)');
cec = struct('FreqWindow',7,'TimeWindow',1,'InterpType','cubic', ...
     'PilotAverage','UserDefined');
```

#### Populate Subframe with PSSCH Symbols

Create the subframe grid and indices for the subframe. Create shared channel and demodulation reference signal (DM-RS) symbols. Populate the subframe with shared channel and DM-RS symbols.

```
subframe = lteSLResourceGrid(ue);

[psschIndices,psschInfo] = ltePSSCHIndices(ue);
psschSymbols = ltePSSCH(ue,zeros(psschInfo.G,1));

subframe(psschIndices) = psschSymbols;
```

Create the control DM-RS and indices. Add the PSSCH DM-RS symbols to the subframe.

```
subframe(ltePSSCHDRSIndices(ue)) = ltePSSCHDRS(ue);
```

#### Estimate Channel Characteristics

Estimate the channel characteristics by using the received resource grid containing PSSCH DM-RS symbols.

- Perform sidelink SC-FDMA modulation
- Add noise to the transmitted signal
- Perform sidelink SC-FDMA demodulation and channel estimation
- View the noise estimate

```
txWaveform = lteSLSCFDMAModulate(ue,subframe);

rxWaveform = awgn(txWaveform,15,'measured');

rxGrid = lteSLSCFDMADemodulate(ue,rxWaveform);
[hest,noiseest] = lteSLChannelEstimatePSSCH(ue,cec,rxGrid);

noiseest
```

```
noiseest = 0.0026
```

## Input Arguments

### ue — UE-specific settings

structure

User equipment settings, specified as a structure containing these fields.

### SidelinkMode — Sidelink mode

'D2D' (default) | 'V2X' | optional

Sidelink mode, specified as 'D2D' or 'V2X'.

Data Types: char | string

### NSLRB — Number of sidelink resource blocks

integer scalar from 6 to 110

Number of sidelink resource blocks, specified as an integer scalar from 6 to 110.

Example: 6, which corresponds to a channel bandwidth of 1.4 MHz.

Data Types: double

### NSAID — Sidelink group destination identity

integer in the interval [0, 255]

Sidelink group destination identity, specified as an integer in the interval [0, 255].

This field is the lower eight bits of the full 24-bit ProSe Layer-2 group destination ID. This field and the NSubframePSSCH field control the value of the scrambling sequence at the start of each subframe. This field is required only for D2D sidelink.

Data Types: double

### CyclicPrefixSL — Cyclic prefix length

'Normal' (default) | 'Extended' | optional

Cyclic prefix length, specified as 'Normal' or 'Extended'.

Data Types: char | string

### NXID — V2X scrambling identity

integer scalar

V2X scrambling identity, specified as an integer scalar. NXID is the 16 bit CRC associated with the PSCCH SCI grant. It is only required for V2X sidelink.

Data Types: double

### NSubframePSSCH — PSSCH subframe number

integer scalar

PSSCH subframe number in the PSSCH subframe pool, specified as an integer scalar. ($n_{\mathrm{ssf}}^{\mathrm{PSSCH}}$)

`NSubframePSSCH` and `NSAID` control the values of the scrambling sequence. It is only required for D2D sidelink.

Data Types: `double`

**`PRBSet` — Zero-based physical resource block indices**
integer column vector | two-column integer matrix

Zero-based physical resource block (PRB) indices, specified as an integer column vector or a two-column integer matrix.

The PSSCH is intended to be transmitted in the same PRB in each slot of a subframe. Therefore, specifying `PRBSet` as a single column of PRB indices is recommended. However, for a nonstandard slot-hopping PRB allocation, `PRBSet` can be specified as a two-column matrix of indices corresponding to slot-wise resource allocations for PSSCH.

Data Types: `double`

Data Types: `struct`

**`rxgrid` — Received resource element grid**
3-D array

Received resource element grid, specified as an $N_{SC}$-by-$N_{Sym}$-by-$N_R$ array of complex symbols.

- $N_{SC}$ is the number of subcarriers.
- $N_{Sym} = N_{SF} \times N_{SymPerSF} = 1 \times N_{SymPerSF}$
  - $N_{SF}$ is the total number of subframes. For this function `rxgrid` must contain one subframe.
  - $N_{SymPerSF}$ is the number of SC-FDMA symbols per subframe.
    - For normal cyclic prefix, a subframe contains 14 SC-FDMA symbols.
    - For extended cyclic prefix, a subframe contains 12 SC-FDMA symbols.
  - $N_R$ is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

**`cec` — PSSCH channel estimation settings**
structure

PSSCH channel estimation settings, specified as a structure that can contain these fields.

**`FreqWindow` — Size of frequency window**
integer

Size of frequency window, specified as an integer that is odd or a multiple of 12. `FreqWindow` is the number of resource elements (REs) used to average over frequency.

Data Types: `double`

**`TimeWindow` — Size of time window**
integer

Size of time window, specified as an odd integer. `TimeWindow` is the number of resource elements (REs) used to average over time.

Data Types: `double`

**`InterpType` — Type of 2-D interpolation**
`'nearest'` | `'linear'` | `'natural'` | `'cubic'` | `'v4'` | `'none'`

Type of 2-D interpolation used during interpolation, specified as one of these supported choices.

| Value | Description |
|---|---|
| `'nearest'` | Nearest neighbor interpolation |
| `'linear'` | Linear interpolation |
| `'natural'` | Natural neighbor interpolation |
| `'cubic'` | Cubic interpolation |
| `'v4'` | MATLAB 4 `griddata` method |
| `'none'` | Disables interpolation |

For details, see `griddata`.

Data Types: `char` | `string`

**`PilotAverage` — Type of pilot averaging**
`'UserDefined'` (default) | `'TestEVM'` | optional

Type of pilot averaging, specified as `'UserDefined'` or `'TestEVM'`.

The `'UserDefined'` pilot averaging uses a rectangular kernel of size `cec.FreqWindow`-by-`cec.TimeWindow` and performs a 2-D filtering operation on the pilots. Pilots near the edge of the resource grid are averaged less because they have no neighbors outside of the grid.

For `cec.FreqWindow` = 12×$X$ (that is, any multiple of 12) and `cec.TimeWindow` = 1, the estimator enters a special case where an averaging window of (12×$X$)-in-frequency is used to average the pilot estimates. The averaging is always applied across (12×$X$) subcarriers, even at the upper and lower band edges. Therefore, the first (6×$X$) symbols at the upper and lower band edge have the same channel estimate. This operation ensures that averaging is always done on 12 (or a multiple of 12) symbols. The `'TestEVM'` pilot averaging ignores other structure fields in `cec`, and for the transmitter EVM testing, it follows the method described in TS 36.101, Annex F.

Data Types: `char` | `string`

Data Types: `struct`

## Output Arguments

**`hest` — Channel estimate between each transmit and receive antenna**
3-D array

Channel estimate between each transmit and receive antenna, returned as an $N_{SC}$-by-$N_{Sym}$-by-$N_R$ array of complex symbols. $N_{SC}$ is the total number of subcarriers, $N_{Sym}$ is the number of SC-FDMA symbols, and $N_R$ is the number of receive antennas.

For `cec.InterpType = 'none'`,

- No interpolation between the pilot symbol estimates is performed and no virtual pilots are created

- `hest` contains channel estimates in the locations of transmitted DM-RS symbols for each received antenna and all other elements of `hest` are `0`
- The averaging of pilot symbol estimates, described by `cec.TimeWindow` and `cec.FreqWindow`, is still performed

**`noiseest` — Noise estimate**
numeric scalar

Noise estimate, returned as a numeric scalar. When `cec.PilotAverage` is `'UserDefined'`, this output is the power spectral density of the noise present on the estimated channel response coefficients. Otherwise, `noiseest` returns `0`.

# Version History

**Introduced in R2017a**

## References

[1] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteSLChannelEstimatePSBCH` | `lteSLChannelEstimatePSCCH`

# lteSLFrameOffsetPSBCH

PSBCH DM-RS sidelink subframe timing estimate

## Syntax

```
offset = lteSLFrameOffsetPSBCH(ue,waveform)
[offset,corr] = lteSLFrameOffsetPSBCH(ue,waveform)
```

## Description

`offset = lteSLFrameOffsetPSBCH(ue,waveform)` performs synchronization using PSBCH demodulation reference signal (DM-RS) symbols for the time-domain waveform, `waveform`, given UE-specific settings, `ue`.

The returned `offset` indicates the number of samples from the start of the input waveform to the position in that waveform where the first subframe containing the DM-RS begins.

`[offset,corr] = lteSLFrameOffsetPSBCH(ue,waveform)` also returns a complex matrix, `corr`, which is used to extract the timing offset.

## Examples

### Synchronize and Demodulate Transmission Containing PSBCH DM-RS

Synchronize and demodulate a transmission that has been delayed by five samples. The transmission contains PSBCH demodulation reference signal (DM-RS) symbols that are used when estimating the waveform timing offset.

Create a UE configuration specifying 15 resource blocks, a sidelink identity of 1, and a normal cyclic prefix.

```
ue = struct('NSLRB',15,'NSLID',1,'CyclicPrefixSL','Normal');
```

Create a resource grid and modulate the waveform containing PSBCH DM-RS symbols.

```
txgrid = lteSLResourceGrid(ue);
txgrid(ltePSBCHDRSIndices(ue)) = ltePSBCHDRS(ue);
txwaveform = lteSLSCFDMAModulate(ue,txgrid);
```

Add a time delay of five samples.

```
rxwaveform = [zeros(5,1); txwaveform];
```

Calculate the timing offset in samples.

```
offset = lteSLFrameOffsetPSBCH(ue,rxwaveform)
```

```
offset = 5
```

Correct the timing offset and demodulate the received waveform.

```
rxGrid = lteSLSCFDMADemodulate(ue,rxwaveform(1+offset:end));
```

### View Correlation Peak in PSBCH DM-RS Transmission

View the correlation peak for a transmission waveform that has been delayed by five samples. The transmission contains PSBCH demodulation reference signal (DM-RS) symbols available for estimating the waveform timing.

Create a UE configuration specifying 15 resource blocks, a sidelink identity of 1, and a normal cyclic prefix.

```
ue = struct('NSLRB',15,'NSLID',1,'CyclicPrefixSL','Normal');
```

Create a resource grid and modulate the waveform containing PSBCH DM-RS symbols.

```
txgrid = lteSLResourceGrid(ue);
txgrid(ltePSBCHDRSIndices(ue)) = ltePSBCHDRS(ue);
txwaveform = lteSLSCFDMAModulate(ue,txgrid);
```

Calculate the timing offset in samples.

```
[offset corr] = lteSLFrameOffsetPSBCH(ue,txwaveform);
```

Add a time delay of five samples.

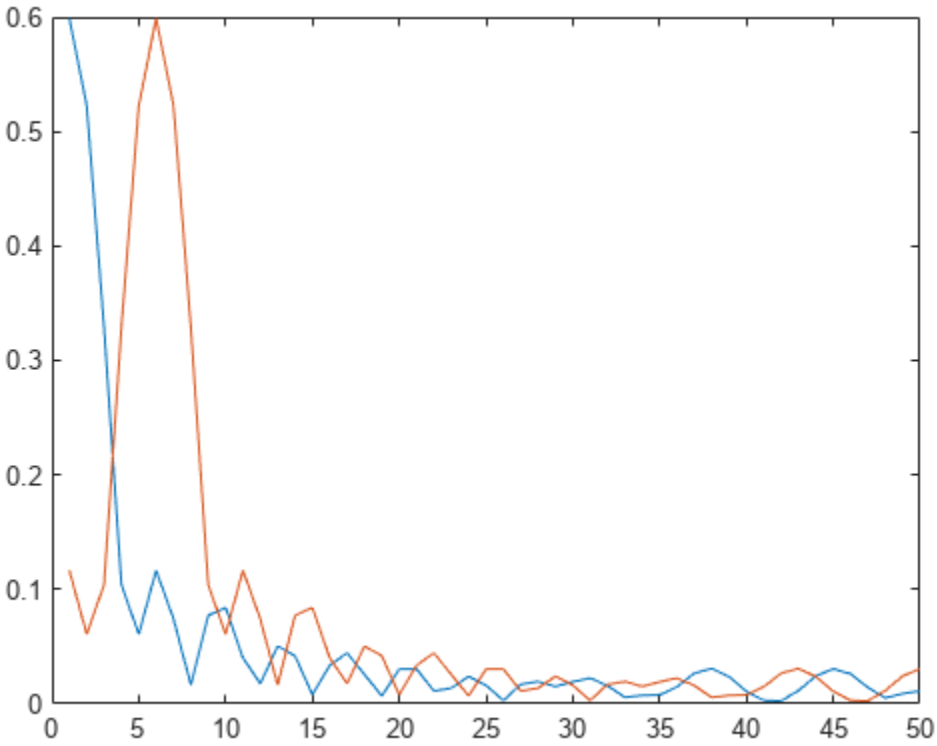
```
rxwaveform = [zeros(5,1); txwaveform];
```

Calculate the timing offset in samples.

```
[offset corrDelayed] = lteSLFrameOffsetPSBCH(ue,rxwaveform);
```

Plot the correlation data before and after delay is added. Zoom in on the *x*-axis to view correlation peaks.

```
plot(corr)
hold on
plot(corrDelayed)
hold off
xlim([0 50])
```




Correct the timing offset and demodulate the received waveform.

```
rxGrid = lteSLSCFDMADemodulate(ue,rxwaveform(1+offset:end));
```

## Input Arguments

### ue — UE-specific settings

scalar structure

User equipment settings, specified as a parameter structure containing these fields:

### SidelinkMode — Sidelink mode

`'D2D'` (default) | `'V2X'` | optional

Sidelink mode, specified as `'D2D'` or `'V2X'`.

Data Types: `char` | `string`

### NSLRB — Number of sidelink resource blocks

integer scalar from 6 to 110

Number of sidelink resource blocks, specified as an integer scalar from 6 to 110.

Example: 6, which corresponds to a channel bandwidth of 1.4 MHz.

Data Types: `double`

**`CyclicPrefixSL` — Cyclic prefix length**
`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

**`NSLID` — Physical layer sidelink synchronization identity**
integer from 0 to 355

Physical layer sidelink synchronization identity, specified as an integer from 0 to 355. ($N_{ID}^{SL}$)

Data Types: `double`

Data Types: `struct`

**`waveform` — Modulated sidelink waveform**
numeric matrix

Modulated sidelink waveform, specified as an $N_S$-by-$N_R$ numeric matrix, where $N_S$ is the number of time-domain samples and $N_R$ is the number of receive antennas. `waveform` should be at least one subframe long and contain the DM-RS signals.

You can generate this matrix by performing SC-FDMA modulation on a resource matrix. To perform this modulation, use the `lteSLSCFDMAModulate` function or one of the channel model functions, such as `lteFadingChannel` or `lteMovingChannel`.

Data Types: `double`

## Output Arguments

**`offset` — Offset number of samples**
scalar integer

Offset number of samples, returned as a scalar integer. This output is the number of samples from the start of the waveform to the position in that waveform where the first subframe containing the DM-RS begins. `offset` is computed by extracting the timing of the peak of the correlation between `waveform` and internally generated reference waveforms containing DM-RS signals. The correlation is performed separately for each antenna. The antenna with the strongest correlation is used to compute `offset`.

**Note** `offset` is the position of `mod(max(abs(corr),`$L_{SF}$`))`, where $L_{SF}$ is the subframe length.

**`corr` — Signal used to extract the timing offset**
numeric matrix

Signal used to extract the timing offset, returned as a complex numeric matrix. `corr` has the same dimensions as `waveform`.

# Version History

**Introduced in R2017a**

## See Also

`lteFadingChannel` | `lteMovingChannel` | `lteSCFDMADemodulate`

# lteSLFrameOffsetPSCCH

PSCCH DM-RS sidelink subframe timing estimate

## Syntax

```
offset = lteSLFrameOffsetPSCCH(ue,waveform)
[offset,corr] = lteSLFrameOffsetPSCCH(ue,waveform)
```

## Description

`offset = lteSLFrameOffsetPSCCH(ue,waveform)` performs synchronization using PSCCH demodulation reference signal (DM-RS) symbols for the time-domain waveform, `waveform`, given UE-specific settings, `ue`.

The returned `offset` indicates the number of samples from the start of the input waveform to the position in that waveform where the first subframe containing DM-RS begins.

`[offset,corr] = lteSLFrameOffsetPSCCH(ue,waveform)` also returns a complex matrix, `corr`, which is used to extract the timing offset.

## Examples

### Synchronize and Demodulate Transmission Containing PSCCH DM-RS

Synchronize and demodulate a transmission that has been delayed by five samples. The transmission contains PSCCH demodulation reference signal (DM-RS) symbols that are used when estimating the waveform timing offset.

Create a UE configuration specifying 15 resource blocks, a normal cyclic prefix, and a `PRBSet` of 1.

```
ue = struct('NSLRB',15,'CyclicPrefixSL','Normal','PRBSet',1);
```

Create a resource grid and modulate the waveform containing PSCCH DM-RS symbols.

```
txgrid = lteSLResourceGrid(ue);
txgrid(ltePSCCHDRSIndices(ue)) = ltePSCCHDRS;
txwaveform = lteSLSCFDMAModulate(ue,txgrid);
```

Add a time delay of five samples.

```
rxwaveform = [zeros(5,1); txwaveform];
```

Calculate the timing offset in samples.

```
offset = lteSLFrameOffsetPSCCH(ue,rxwaveform)
```

```
offset = 5
```

Correct the timing offset and demodulate the received waveform.

```
rxGrid = lteSLSCFDMADemodulate(ue,rxwaveform(1+offset:end));
```

### View Correlation Peak in PSCCH DM-RS Transmission

View the correlation peak for a transmission waveform that has been delayed by five samples. The transmission contains PSCCH demodulation reference signal (DM-RS) symbols available for estimating the waveform timing.

Create a UE configuration specifying 15 resource blocks, a normal cyclic prefix, and a `PRBSet` of 1.

```
ue = struct('NSLRB',15,'CyclicPrefixSL','Normal','PRBSet',1);
```

Create a resource grid and modulate the waveform containing PSCCH DM-RS symbols.

```
txgrid = lteSLResourceGrid(ue);
txgrid(ltePSCCHDRSIndices(ue)) = ltePSCCHDRS;
txwaveform = lteSLSCFDMAModulate(ue,txgrid);
```

Calculate the timing offset in samples.

```
[offset corr] = lteSLFrameOffsetPSCCH(ue,txwaveform);
```

Add a time delay of five samples.

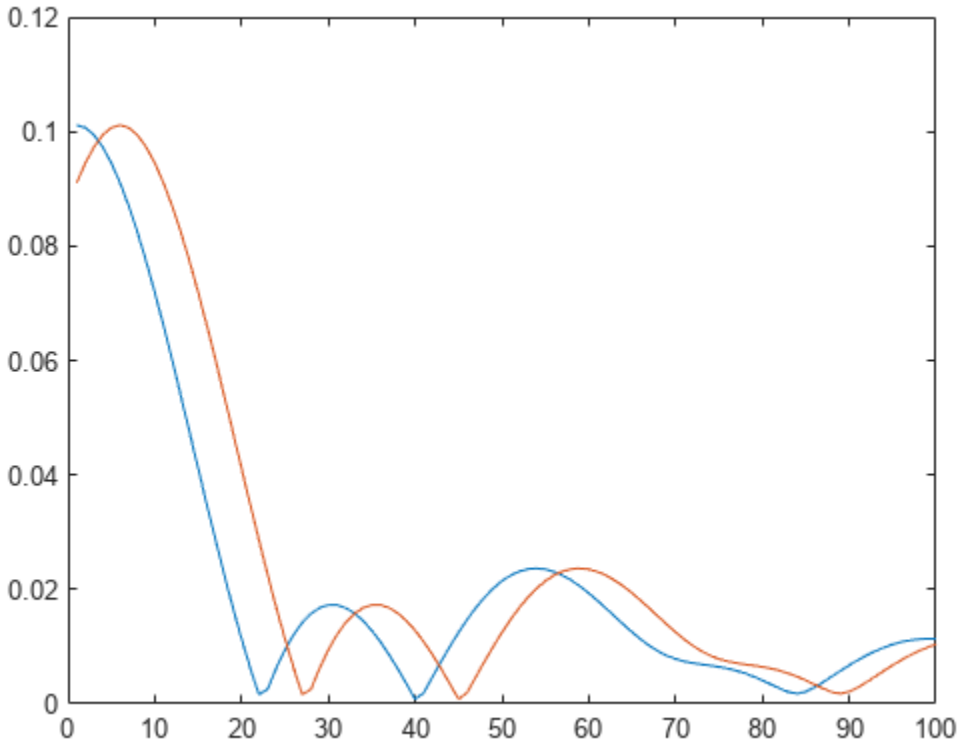
```
rxwaveform = [zeros(5,1); txwaveform];
```

Calculate the timing offset in samples.

```
[offset corrDelayed] = lteSLFrameOffsetPSCCH(ue,rxwaveform);
```

Plot the correlation data before and after delay is added. Zoom in on the *x*-axis to view correlation peaks.

```
plot(corr)
hold on
plot(corrDelayed)
hold off
xlim([0 100])
```




Correct the timing offset and demodulate the received waveform.

```
rxGrid = lteSLSCFDMADemodulate(ue,rxwaveform(1+offset:end));
```

## Input Arguments

### ue — UE-specific settings

scalar structure

User equipment settings, specified as a parameter structure containing these fields:

### SidelinkMode — Sidelink mode

`'D2D'` (default) | `'V2X'` | optional

Sidelink mode, specified as `'D2D'` or `'V2X'`.

Data Types: `char` | `string`

### NSLRB — Number of sidelink resource blocks

integer scalar from 6 to 110

Number of sidelink resource blocks, specified as an integer scalar from 6 to 110.

Example: 6, which corresponds to a channel bandwidth of 1.4 MHz.

Data Types: `double`

**`CyclicPrefixSL` — Cyclic prefix length**
`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

**`PRBSet` — Zero-based physical resource block index**
integer | integer column vector | two-column integer matrix

Zero-based physical resource block (PRB) index, specified as an integer, an integer column vector, or a two-column integer matrix.

For D2D sidelink, the PSCCH is intended to be transmitted in a single PRB in a subframe and therefore, specifying `PRBSet` as a scalar PRB index is recommended. For V2X sidelink, the PSCCH is intended to be transmitted in a pair of consecutive PRB in a subframe, therefore `PRBSet` must be a column vector containing two consecutive indices. However, for a more general nonstandard multi-PRB allocation, `PRBSet` can be a set of indices specified as an integer column vector or as a two-column integer matrix corresponding to slot-wise resource allocations for PSCCH.

Data Types: `double`

**`CyclicShift` — Cyclic shift for DM-RS**
`0` (default) | `3` | `6` | `9`

Cyclic shift for DM-RS, specified as `0`, `3`, `6` or `9`. The function uses this input only for V2X sidelink.

Data Types: `double`

Data Types: `struct`

**`waveform` — Modulated sidelink waveform**
numeric matrix

Modulated sidelink waveform, specified as an $N_S$-by-$N_R$ numeric matrix, where $N_S$ is the number of time-domain samples and $N_R$ is the number of receive antennas. `waveform` should be at least one subframe long and contain the DM-RS signals.

You can generate this matrix by performing SC-FDMA modulation on a resource matrix. To perform this modulation, use the `lteSLSCFDMAModulate` function or one of the channel model functions, such as `lteFadingChannel` or `lteMovingChannel`.

Data Types: `double`

## Output Arguments

**`offset` — Offset number of samples**
scalar integer

Offset number of samples, returned as a scalar integer. This output is the number of samples from the start of the waveform to the position in that waveform where the first subframe containing the DM-RS begins. `offset` is computed by extracting the timing of the peak of the correlation between `waveform` and internally generated reference waveforms containing DM-RS signals. The correlation is performed separately for each antenna. The antenna with the strongest correlation is used to compute `offset`.

**Note** `offset` is the position of `mod(max(abs(corr),`$L_{SF}$`))`, where $L_{SF}$ is the subframe length.

#### `corr` — Signal used to extract the timing offset

numeric matrix

Signal used to extract the timing offset, returned as a complex numeric matrix. `corr` has the same dimensions as `waveform`.

## Version History

**Introduced in R2017a**

## See Also

`lteFadingChannel` | `lteMovingChannel` | `lteSCFDMADemodulate`

# lteSLFrameOffsetPSSCH

PSSCH DM-RS sidelink subframe timing estimate

## Syntax

```
offset = lteSLFrameOffsetPSSCH(ue,waveform)
[offset,corr] = lteSLFrameOffsetPSSCH(ue,waveform)
```

## Description

`offset = lteSLFrameOffsetPSSCH(ue,waveform)` performs synchronization using PSSCH demodulation reference signal (DM-RS) symbols for the time-domain waveform, `waveform`, given UE-specific settings, `ue`.

The returned `offset` indicates the number of samples from the start of the input waveform to the position in that waveform where the first subframe containing DM-RS begins.

`[offset,corr] = lteSLFrameOffsetPSSCH(ue,waveform)` also returns a complex matrix, `corr`, which is used to extract the timing offset.

## Examples

### Synchronize and Demodulate Transmission Containing PSSCH DM-RS

Synchronize and demodulate a transmission that has been delayed by five samples. The transmission contains PSSCH demodulation reference signal (DM-RS) symbols that are used when estimating the waveform timing offset.

Create a UE configuration specifying 15 resource blocks, a sidelink identity of 1, a normal cyclic prefix, a PSSCH subframe number of 0, and a `PRBSet` of 1.

```
ue = struct('NSLRB',15,'NSAID',1,'CyclicPrefixSL','Normal', ...
    'NSubframePSSCH',0,'PRBSet',1);
```

Create a resource grid and modulate the waveform containing PSSCH DM-RS symbols.

```
txgrid = lteSLResourceGrid(ue);
txgrid(ltePSSCHDRSIndices(ue)) = ltePSSCHDRS(ue);
txwaveform = lteSLSCFDMAModulate(ue,txgrid);
```

Add a time delay of five samples.

```
rxwaveform = [zeros(5,1); txwaveform];
```

Calculate the timing offset in samples.

```
offset = lteSLFrameOffsetPSSCH(ue,rxwaveform)
```

```
offset = 5
```

Correct the timing offset and demodulate the received waveform.

```
rxGrid = lteSLSCFDMADemodulate(ue,rxwaveform(1+offset:end));
```

### View Correlation Peak in PSSCH DM-RS Transmission

View the correlation peak for a transmission waveform that has been delayed by five samples. The transmission contains PSSCH demodulation reference signal (DM-RS) symbols available for estimating the waveform timing.

Create a UE configuration specifying 15 resource blocks, a sidelink identity of 1, a normal cyclic prefix, a PSSCH subframe number of 0, and a `PRBSet` of 1.

```
ue = struct('NSLRB',15,'NSAID',1,'CyclicPrefixSL','Normal', ...
    'NSubframePSSCH',0,'PRBSet',1);
```

Create a resource grid and modulate the waveform containing PSSCH DM-RS symbols.

```
txgrid = lteSLResourceGrid(ue);
txgrid(ltePSSCHDRSIndices(ue)) = ltePSSCHDRS(ue);
txwaveform = lteSLSCFDMAModulate(ue,txgrid);
```

Calculate the timing offset in samples.

```
[offset corr] = lteSLFrameOffsetPSSCH(ue,txwaveform);
```

Add a time delay of five samples.

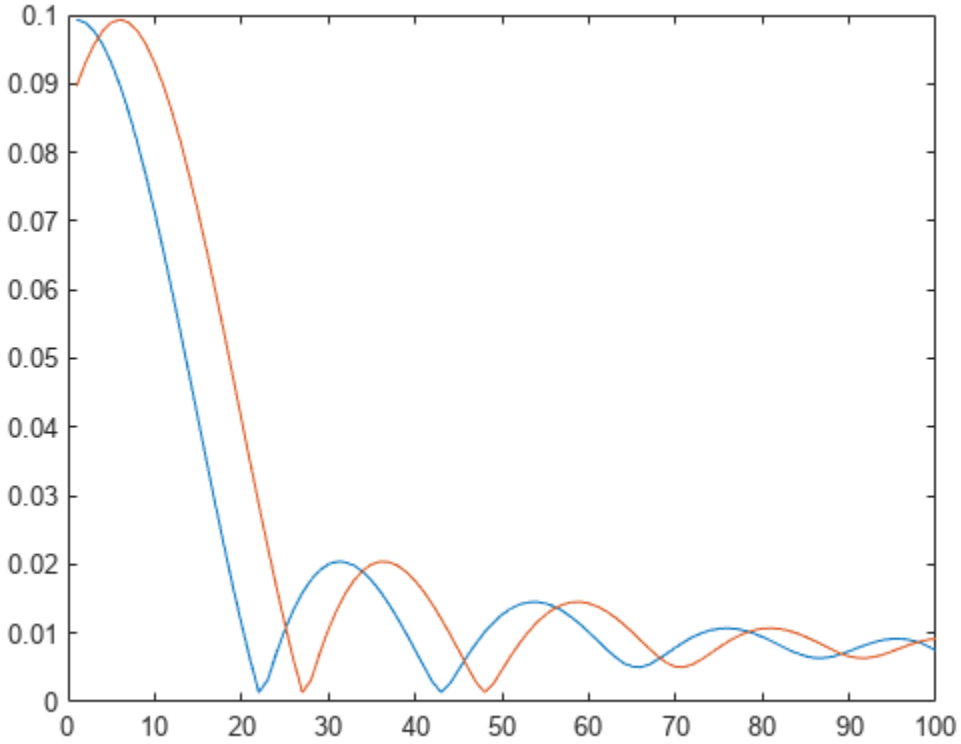
```
rxwaveform = [zeros(5,1); txwaveform];
```

Calculate the timing offset in samples.

```
[offset corrDelayed] = lteSLFrameOffsetPSSCH(ue,rxwaveform);
```

Plot the correlation data before and after delay is added. Zoom in on the *x*-axis to view correlation peaks.

```
plot(corr)
hold on
plot(corrDelayed)
hold off
xlim([0 100])
```




Correct the timing offset and demodulate the received waveform.

```
rxGrid = lteSLSCFDMADemodulate(ue,rxwaveform(1+offset:end));
```

## Input Arguments

### ue — UE-specific settings

scalar structure

User equipment settings, specified as a parameter structure containing these fields:

### SidelinkMode — Sidelink mode

`'D2D'` (default) | `'V2X'` | optional

Sidelink mode, specified as `'D2D'` or `'V2X'`.

Data Types: `char` | `string`

### NSLRB — Number of sidelink resource blocks

integer scalar from 6 to 110

Number of sidelink resource blocks, specified as an integer scalar from 6 to 110.

Example: 6, which corresponds to a channel bandwidth of 1.4 MHz.

Data Types: `double`

**`NSAID` — Sidelink group destination identity**
integer in the interval [0, 255]

Sidelink group destination identity, specified as an integer in the interval [0, 255].

This field is the lower eight bits of the full 24-bit ProSe Layer-2 group destination ID. This field and the `NSubframePSSCH` field control the value of the scrambling sequence at the start of each subframe. This field is required only for D2D sidelink.

Data Types: `double`

**`NXID` — V2X scrambling identity**
integer scalar

V2X scrambling identity, specified as an integer scalar. `NXID` is the 16 bit CRC associated with the PSCCH SCI grant. It is only required for V2X sidelink.

Data Types: `double`

**`CyclicPrefixSL` — Cyclic prefix length**
`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

**`NSubframePSSCH` — PSSCH subframe number**
integer scalar

PSSCH subframe number in the PSSCH subframe pool, specified as an integer scalar. ($n_{\mathrm{ssf}}^{\mathrm{PSSCH}}$)

`NSubframePSSCH` and `NSAID` control the values of the scrambling sequence. It is only required for D2D sidelink.

Data Types: `double`

**`PRBSet` — Zero-based physical resource block indices**
integer column vector | two-column integer matrix

Zero-based physical resource block (PRB) indices, specified as an integer column vector or a two-column integer matrix.

The PSSCH is intended to be transmitted in the same PRB in each slot of a subframe. Therefore, specifying `PRBSet` as a single column of PRB indices is recommended. However, for a nonstandard slot-hopping PRB allocation, `PRBSet` can be specified as a two-column matrix of indices corresponding to slot-wise resource allocations for PSSCH.

Data Types: `double`

Data Types: `struct`

**`waveform` — Modulated sidelink waveform**
numeric matrix

Modulated sidelink waveform, specified as an $N_S$-by-$N_R$ numeric matrix, where $N_S$ is the number of time-domain samples and $N_R$ is the number of receive antennas. `waveform` should be at least one subframe long and contain the DM-RS signals.

You can generate this matrix by performing SC-FDMA modulation on a resource matrix. To perform this modulation, use the `lteSLSCFDMAModulate` function or one of the channel model functions, such as `lteFadingChannel` or `lteMovingChannel`.

Data Types: `double`

## Output Arguments

### `offset` — Offset number of samples

scalar integer

Offset number of samples, returned as a scalar integer. This output is the number of samples from the start of the waveform to the position in that waveform where the first subframe containing the DM-RS begins. `offset` is computed by extracting the timing of the peak of the correlation between `waveform` and internally generated reference waveforms containing DM-RS signals. The correlation is performed separately for each antenna. The antenna with the strongest correlation is used to compute `offset`.

**Note** `offset` is the position of `mod(max(abs(corr),`$L_{SF}$`))`, where $L_{SF}$ is the subframe length.

### `corr` — Signal used to extract the timing offset

numeric matrix

Signal used to extract the timing offset, returned as a complex numeric matrix. `corr` has the same dimensions as `waveform`.

# Version History

**Introduced in R2017a**

## See Also

`lteFadingChannel` | `lteMovingChannel` | `lteSCFDMADemodulate`

# lteSLMIB

Sidelink MIB encoding and decoding

## Syntax

```
mibslout = lteSLMIB(ue)
ueout = lteSLMIB(mibsl)
ueout = lteSLMIB(mibsl,ue)
```

## Description

`mibslout = lteSLMIB(ue)` returns the encoded sidelink MIB (MIB-SL) RRC message bits for the specified UE settings structure.

For more information, see “MIB-SL Message Processing” on page 2-1109.

`ueout = lteSLMIB(mibsl)` performs the inverse processing of the preceding syntax, returning a UE parameter structure after decoding the input MIB-SL message bits.

`ueout = lteSLMIB(mibsl,ue)` returns the UE settings structure, updating any fields contained in the input UE parameter structure with values decoded from `mibsl`.

## Examples

### Create MIB-SL Message

Create the 40-bit MIB-SL associated with the parameter values to be carried on the message.

Initialize a UE-specific configuration structure with 10 MHz bandwidth for TDD.

```
ue.NSLRB = 50;
ue.DuplexMode = 'TDD';
ue.TDDConfig  = 6;
ue.NFrame = 5;
ue.NSubframe = 1;
ue.InCoverage = 1;
```

Generate the 40-bit MIB-SL message using the `ue` structure.

```
mibsl = lteSLMIB(ue);
```

### Decode MIB-SL Message

Decode the 40-bit MIB-SL message, creating a received parameter structure from the message.

Initialize a UE-specific configuration structure with 5 MHz bandwidth for TDD.

```
ue.NSLRB = 25;
ue.DuplexMode = 'TDD';
```

```
ue.TDDConfig = 6;
ue.NFrame = 5;
ue.NSubframe = 1;
ue.InCoverage = 1
```

```
ue = struct with fields:
         NSLRB: 25
    DuplexMode: 'TDD'
     TDDConfig: 6
        NFrame: 5
     NSubframe: 1
    InCoverage: 1
```

Generate the 40-bit MIB-SL message using the `ue` structure.

```
mibsl = lteSLMIB(ue);
```

Convert the MIB-SL bit vector back into a parameter set. Compare this parameter set with the transmission set.

```
rxparams = lteSLMIB(mibsl)
```

```
rxparams = struct with fields:
         NSLRB: 25
    DuplexMode: 'TDD'
     TDDConfig: 6
        NFrame: 5
     NSubframe: 1
    InCoverage: 1
```

```
isequal(rxparams,ue)
```

```
ans = logical
   1
```

### Update UE Structure Using MIB-SL Message

Update UE-specific parameter configuration structure settings using the 40-bit MIB-SL message. Encode an MIB-SL message based on one ue structure parameter set.

#### Encode an MIB-SL message from one UE-specific configuration

Initialize a UE-specific configuration structure with 5 MHz bandwidth for TDD. Encode a 40-bit MIB-SL message using the `ue1` structure.

```
ue1.NSLRB = 25;
ue1.DuplexMode = 'TDD';
ue1.TDDConfig = 6;
ue1.NFrame = 5;
ue1.NSubframe = 1;
ue1.InCoverage = 1;

mibsl = lteSLMIB(ue1);
```

### Create a second UE-specific configuration

Initialize a second UE-specific configuration structure with a different configuration. Compare `ue2` with `ue1`.

```
ue2.NSLRB = 75;
ue2.DuplexMode = 'TDD';
ue2.TDDConfig = 2;
ue2.NFrame = 2;
ue2.NSubframe = 2;
ue2.InCoverage = 0;

isequal(ue2,ue1)
```

```
ans = logical
   0
```

### Update the second UE-specific configuration based on the MIB-SL message

Using `mibsl`, update the settings in `ue2` to match `ue1`. Compare `ue2` with `ue1`.

```
ue2 = lteSLMIB(mibsl,ue2);
isequal(ue2,ue1)
```

```
ans = logical
   1
```

## Input Arguments

### ue — User equipment settings

structure

User equipment settings, specified as a parameter structure containing these fields:

#### NSLRB — Number of sidelink resource blocks

integer scalar from 6 to 110

Number of sidelink resource blocks, specified as an integer scalar from 6 to 110.

Example: `6`, which corresponds to a channel bandwidth of 1.4 MHz.

Data Types: `double`

#### DuplexMode — Duplexing mode

`'FDD'` (default) | `'TDD'` | optional

Duplexing mode, specified as `'FDD'` or `'TDD'`.

Data Types: `char` | `string`

#### TDDConfig — Uplink or downlink configuration

0 (default) | integer from 0 to 6 | optional

Uplink or downlink configuration, specified as an integer from 0 to 6. (*tdd-ConfigSL-r12*)

`TDDConfig` is applicable for TDD duplex mode only.

Data Types: `double`

**`NFrame` — Direct frame number**
0 (default) | nonnegative integer | optional

Direct frame number, specified as a nonnegative integer. (*directFrameNumber-r12*)

Data Types: `double`

**`NSubframe` — Direct subframe number**
0 (default) | nonnegative integer | optional

Direct subframe number, specified as a nonnegative integer. (*directSubframeNumber-r12*)

Data Types: `double`

**`InCoverage` — Indicates whether UE is in E-UTRAN coverage**
0 (default) | 1 | optional

Indicates whether the UE transmitting the MIB-SL is in E-UTRAN coverage, specified as 0 (not in coverage) or 1 (in coverage). (*inCoverage-r12*).

Data Types: `double`

Data Types: `struct`

**`mibsl` — MIB-SL message bit sequence**
40-bit column vector

MIB-SL message bit sequence, specified as a 40-bit column vector.

For more information, see “MIB-SL Message Processing” on page 2-1109.

Data Types: `double` | `int8` | `logical`

## Output Arguments

**`mibslout` — MIB-SL message bit sequence**
40-bit column vector

MIB-SL message bit sequence, returned as a 40-bit column vector.

For more information, see “MIB-SL Message Processing” on page 2-1109.

Data Types: `int8`

**`ueout` — User equipment settings**
structure

User equipment settings, returned as a parameter structure containing these fields:

**`NSLRB` — Number of sidelink resource blocks**
0, 6, 15, 25, 50, 75, or 100

Number of sidelink resource blocks, returned as an integer from the set {0, 6, 15, 25, 50, 75, 100}. ($N_{\mathrm{RB}}^{\mathrm{SL}}$)

For more information on sidelink bandwidths, see “MIB-SL Message Processing” on page 2-1109.

Data Types: `int32`

**`DuplexMode` — Duplexing mode**
`'FDD'` | `'TDD'`

Duplexing mode, returned as `'FDD'` or `'TDD'`.

Data Types: `char`

**`TDDConfig` — Uplink or downlink configuration**
integer from 0 to 6

Uplink or downlink configuration, returned as an integer from 0 to 6. (*tdd-ConfigSL-r12*)

`TDDConfig` is applicable for TDD duplex mode only.

Data Types: `int32`

**`NFrame` — Direct frame number**
nonnegative integer

Direct frame number, returned as a nonnegative integer. (*directFrameNumber-r12*)

Data Types: `int32`

**`NSubframe` — Direct subframe number**
nonnegative integer

Direct subframe number, returned as a nonnegative integer. (*directSubframeNumber-r12*)

Data Types: `int32`

**`InCoverage` — Indicates when UE is in E-UTRAN coverage**
0 | 1

Indicates when UE is in E-UTRAN coverage, returned as 0 or 1. (*inCoverage-r12*) The UE transmitting the MIB-SL is:

- Not in E-UTRAN coverage when `InCoverage` = 0.
- In E-UTRAN coverage when `InCoverage` = 1.

Data Types: `int32`

Data Types: `struct`

## More About

### MIB-SL Message Processing

The MIB-SL message is a 40 bits long and defined in TS 36.331 [1], Section 6.5.2. The message is sent from UE to UE on the PC5 interface via the SL-BCH transport channel on the SBCCH logical channel. MIB-SL contains *sl-Bandwidth-r12*, *tdd-ConfigSL-r12*, *directFrameNumber-r12*, *directSubframeNumber-r12*, *inCoverage-r12*, and 19 bits reserved for future.

- When encoding the MIB-SL message:
  - If `NSLRB` is not one of the set {6,15,25,50,75,100}, then all ones are inserted into the first three bits (*sl-Bandwidth-r12* bit field) of the MIB message.
- When decoding the MIB-SL message:
  - If the first three bits (*sl-Bandwidth-r12* bit field) of the input MIB-SL message do not contain the equivalent of a decimal from 0 to 5 (MSB first, corresponding to the PRB set {6,15,25,50,75, 100}) then NSLRB is returned as 0.
  - If the input MIB-SL messages are not 40 bits, the messages are either truncated to 40 elements or zero padded as needed.

# Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.331. “Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Resource Control (RRC); Protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteSLBCH` | `lteSLBCHDecode` | `lteMIB`

# lteSLResourceGrid

Sidelink subframe resource array

## Syntax

```
grid = lteSLResourceGrid(ue)
grid = lteSLResourceGrid(ue,ntxants)
```

## Description

`grid = lteSLResourceGrid(ue)` returns an empty resource grid matrix that represents the resource elements for one subframe, for the specified UE-specific setting structure.

For more information on the resource grid and the multidimensional array used to represent the resource elements for one subframe across all configured antenna ports, see “Represent Resource Grids”.

`grid = lteSLResourceGrid(ue,ntxants)` returns a 3-D resource grid array for the specified UE settings structure and number of antenna planes.

## Examples

### Create Empty Sidelink Resource Grid

Create an empty resource array representing the resource elements for 10 MHz bandwidth.

```
reGrid = lteSLResourceGrid(struct('NSLRB',50));
```

## Input Arguments

**ue — User equipment settings**
structure

User equipment settings, specified as a structure containing these parameter fields:

**NSLRB — Number of sidelink resource blocks**
integer scalar from 6 to 110

Number of sidelink resource blocks, specified as an integer scalar from 6 to 110.

Example: 6, which corresponds to a channel bandwidth of 1.4 MHz.

Data Types: `double`

**CyclicPrefixSL — Cyclic prefix length**
`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

**`ntxants` — Number of transmit antenna planes**
positive integer

Number of transmit antenna planes, specified as a positive integer.

Data Types: `double`

## Output Arguments

**`grid` — Resource element grid**
matrix | 3-D array

Resource element grid, returned as an $N_{SC}$-by-$N_{SYM}$-by-$N_{TX}$ array.

- $N_{SC}$ is the number of subcarriers, 12*`ue.NSLRB`.
- $N_{SYM}$ is the number of SC-FDMA symbols in a subframe—14 for normal cyclic prefix or 12 for extended cyclic prefix.
- $N_{TX}$ is the number of transmission antenna planes.

# Version History
**Introduced in R2016b**

## See Also
`lteSLResourceGridSize` | `lteULResourceGrid` | `lteSLSCFDMAModulate`

# lteSLResourceGridSize

Sidelink subframe resource array size

## Syntax

```
dim = lteSLResourceGridSize(ue)
dim = lteSLResourceGridSize(ue,ntxants)
```

## Description

`dim = lteSLResourceGridSize(ue)` returns a 3-element row vector of dimension lengths for the resource grid array that you can generate from the specified UE settings structure. By default, the number of antennas is set to 1 for single-port sidelink transmissions.

For more information on the resource grid and the multidimensional array used to represent the resource elements for one subframe across all configured antenna ports, see “Represent Resource Grids”.

`dim = lteSLResourceGridSize(ue,ntxants)` accepts the number of antenna planes as an optional input.

## Examples

### Create Empty Sidelink Resource Array Using Grid Size

Use the vector returned by `lteSLResourceGridSize` to create a MATLAB® array. Valid and equivalent sidelink subframe resource grids can be created using the `lteSLResourceGrid` function or the MATLAB `zeros` function.

Create a UE settings structure. Use the output from `lteSLResourceGridSize` as input to `zeros` to generate an empty resource grid.

```
ue = struct('NSLRB',6,'CyclicPrefixSL','Normal');

reGrid1 = zeros(lteSLResourceGridSize(ue));
```

Generate another empty resource grid, this time use `lteSLResourceGrid`.

```
reGrid2 = lteSLResourceGrid(ue);
```

Confirm the two grids are identical.

```
isequal(reGrid1,reGrid2)
```

```
ans = logical
   1
```

### Create Two Antenna Empty Sidelink Resource Array Using Grid Size

Create an empty resource grid for two antenna planes using the vector returned by `lteSLResourceGridSize` and the function `zeros`.

Create a UE settings structure and define a local variable for the number of antennas.

```
ue = struct('NSLRB',6,'CyclicPrefixSL','Normal');
ntxant = 2;
```

Generate an empty resource grid.

```
reGrid = zeros(lteSLResourceGridSize(ue,ntxant));
size(reGrid)
```

```
ans = 1×3

    72    14     2
```

The third dimension indicates that two antenna planes are defined in the output grid.

## Input Arguments

### `ue` — User equipment settings

structure

User equipment settings, specified as a structure containing these parameter fields:

#### `NSLRB` — Number of sidelink resource blocks

integer scalar from 6 to 110

Number of sidelink resource blocks, specified as an integer scalar from 6 to 110.

Example: 6, which corresponds to a channel bandwidth of 1.4 MHz.

Data Types: `double`

#### `CyclicPrefixSL` — Cyclic prefix length

`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

### `ntxants` — Number of transmit antenna planes

positive integer

Number of transmit antenna planes, specified as a positive integer.

Data Types: `double`

## Output Arguments

### `dim` — Dimension lengths of resource grid array

3-element row vector

Dimension lengths of resource grid array, returned as a 3-element row vector, [$N_{SC}$ $N_{SYM}$ $N_{TX}$].

- $N_{SC}$ is the number of subcarriers, 12*`ue.NSLRB`.
- $N_{SYM}$ is the number of SC-FDMA symbols in a subframe—14 for normal cyclic prefix or 12 for extended cyclic prefix.
- $N_{TX}$ is the number of transmission antenna planes.

# Version History

**Introduced in R2016b**

## See Also

`lteSLResourceGrid` | `lteULResourceGridSize`

# lteSLSCFDMADemodulate

Sidelink SC-FDMA demodulation

## Syntax

```
grid = lteSLSCFDMADemodulate(ue,waveform)
grid = lteSLSCFDMADemodulate(ue,waveform,cpfraction)
```

## Description

`grid = lteSLSCFDMADemodulate(ue,waveform)` performs sidelink SC-FDMA demodulation of the input time-domain waveform for the specified UE settings structure. For more information, see “Sidelink SC-FDMA Demodulation” on page 2-1119.

`grid = lteSLSCFDMADemodulate(ue,waveform,cpfraction)` allows the specification of the starting waveform sample for demodulation as a fraction of the cyclic prefix.

## Examples

### Sidelink Demodulation

Perform sidelink SC-FDMA modulation of one subframe containing the sidelink synchronization signals and add noise at an SNR of 3.0 dB. The demodulator zeros the resource elements in the last SC-FDMA symbol. This behavior is consistent with the operation of the SC-FDMA modulator which does not modulate the last SC-FDMA symbol of the subframe. Plot the received waveform and the demodulated resource grid magnitude.

Create a UE settings structure.

```
ue.NSLRB = 15;
ue.CyclicPrefixSL = 'Normal';
ue.NSLID = 17;
```

Populate the resource grid with PSSS and SSSS. Modulate the PSSS and SSSS.

```
txgrid = lteSLResourceGrid(ue);
txgrid(ltePSSSIndices(ue)) = ltePSSS(ue);
txgrid(lteSSSSIndices(ue)) = lteSSSS(ue);

[txwaveform,info] = lteSLSCFDMAModulate(ue,txgrid);
```

Add AWGN with an SNR of 3.0 dB.

```
rxwaveform = awgn(txwaveform,3.0,'measured');
```

Perform sidelink SC-FDMA demodulation.

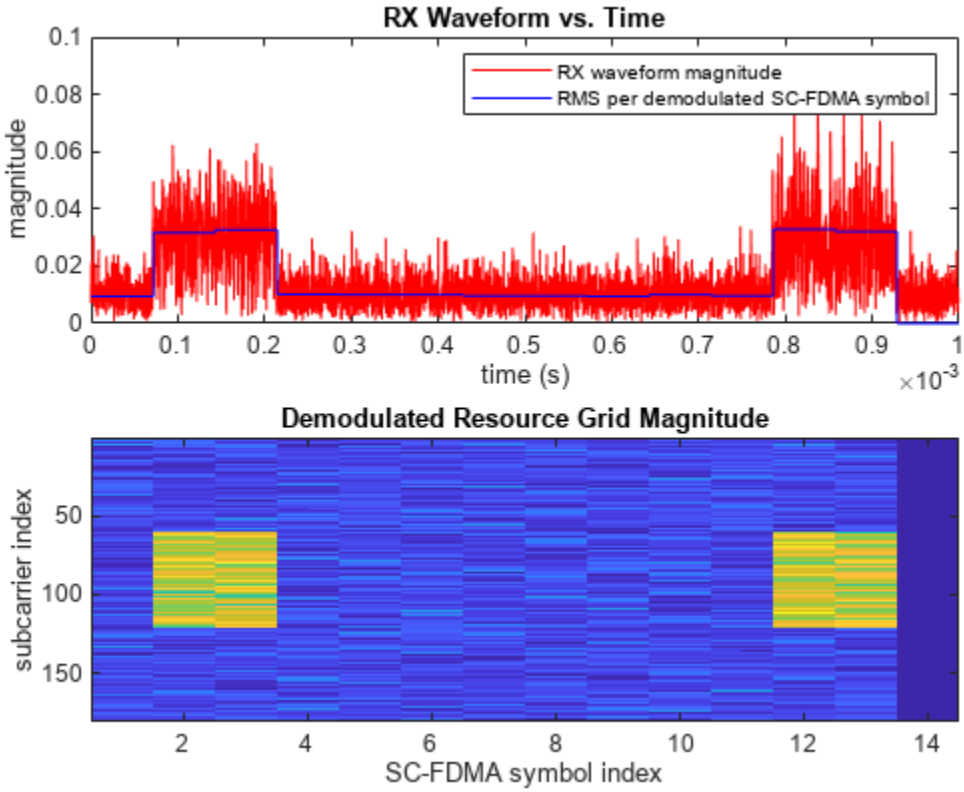
```
rxgrid = lteSLSCFDMADemodulate(ue,rxwaveform);
```

Calculate the RMS of each SC-FDMA symbol in the received resource grid.

```
rms = sqrt(sum(abs((rxgrid./double(info.Nfft)).^2)));
```

Plot the magnitude of the resulting time-domain waveform, overlaying the RMS for each SC-FDMA symbol after demodulation. Plot the demodulated resource grid magnitude.

```
t = (0:size(rxwaveform,1))/info.SamplingRate;
figure

subplot(2,1,1)
plot(t(1:end-1),abs(rxwaveform),'r')
hold on
n = cumsum([1 info.CyclicPrefixLengths + info.Nfft]);
n = [n(1:end-1); n(2:end)];
rmsplot = repmat(rms,[2 1]);
plot(t(n(:)),rmsplot(:),'b')
xlabel('time (s)')
ylabel('magnitude')
title('RX Waveform vs. Time')
legend('RX waveform magnitude','RMS per demodulated SC-FDMA symbol')

subplot(2,1,2)
imagesc(abs(rxgrid))
title('Demodulated Resource Grid Magnitude')
xlabel('SC-FDMA symbol index')
ylabel('subcarrier index')
```



## Input Arguments

### `ue` — User equipment settings

structure

User equipment settings, specified as a parameter structure containing these fields:

#### `NSLRB` — Number of sidelink resource blocks

integer scalar from 6 to 110

Number of sidelink resource blocks, specified as an integer scalar from 6 to 110.

Example: `6`, which corresponds to a channel bandwidth of 1.4 MHz.

Data Types: `double`

#### `CyclicPrefixSL` — Cyclic prefix length

`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

Data Types: `struct`

### `waveform` — Sidelink SC-FDMA modulated waveform

numeric matrix

Sidelink SC-FDMA modulated waveform, specified as an $N_S$-by-$N_T$ numeric matrix, where $N_S$ is the number of the time-domain samples and $N_T$ is the number of transmission antennas. $N_S = K \times 30720 / 2048 \times N_{fft}$, where $N_{fft}$ is the FFT size and $K$ is the number of subframes in `waveform`.

For more information about the FFT size, see `lteSLSCFDMAInfo`.

Data Types: `double`

### `cpfraction` — Fraction of cyclic prefix

`0.55` (default) | numeric scalar from 0 to 1

Fraction of cyclic prefix, specified as a numeric scalar from 0 to 1. A value of `0` represents the start of the cyclic prefix and a value of `1` represents the end of the cyclic prefix. The default value is `0.55` which assumes for the default level of windowing in the `lteSLSCFDMAModulate` function.

Data Types: `double`

## Output Arguments

### `grid` — Resource element grid

numeric 3-D array

Resource element grid, returned as an $N_{SC}$-by-$N_{SYM}$-by-$N_T$ numeric array. $N_{SC}$ is 12 × `NSLRB` subcarriers. $N_{SYM}$ is a multiple of the number of SC-FDMA symbols in a subframe (14 for normal cyclic prefix and 12 for extended cyclic prefix). $N_T$ is the number of antenna ports. `grid` defines the RE allocation across one or more subframes. Multiple subframes are defined by concatenation across the columns (second dimension).

Each antenna plane in `grid` is SC-FDMA modulated, resulting in the columns of `waveform`, as described in “Represent Resource Grids”.

Data Types: `double`
Complex Number Support: Yes

## More About

### Sidelink SC-FDMA Demodulation

Sidelink SC-FDMA demodulation recovers the received subcarrier values by performing one FFT operation per received sidelink SC-FDMA symbol. The recovered subcarrier values are used to construct each column of the output resource array grid. The FFT is positioned partway through the cyclic prefix, to account for some channel delay spread while avoiding the overlap between adjacent SC-FDMA symbols. The input FFT is also shifted by half of one subcarrier. The position of the FFT chosen in the function avoids the SC-FDMA symbol overlapping used in the `lteSLSCFDMAModulate` function. Because the FFT is performed away from the original zero-phase point on the transmitted subcarriers, `lteSLSCFDMADemodulate` applies a phase correction to each subcarrier after the FFT.

**Note**

- TS 36.211 specifies that for PSSCH (Section 9.3.6), PSCCH (9.4.6), PSDCH (9.5.6) and PSBCH (9.6.6), resource elements in the last SC-FDMA symbol within a subframe should be counted in the mapping process but not transmitted. The resource elements of the last SC-FDMA symbol in each subframe of the output resource array grid are set to zero by `lteSLSCFDMADemodulate`. This behavior is consistent with SC-FDMA modulation, performed by `lteSLSCFDMAModulate`.
- The sampling rate of the time-domain sidelink waveform must be the same as the rate used in the `lteSLSCFDMAModulate` function, for the specified number of resource blocks, *NRB*.
- The input `waveform` must be time aligned, such that the first sample is the first sample of the cyclic prefix of the first sidelink SC-FDMA symbol in a subframe.

# Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteSLSCFDMAModulate` | `lteSLSCFDMAInfo`

# lteSLSCFDMAInfo

Sidelink SC-FDMA modulation information

## Syntax

```
info = lteSLSCFDMAInfo(ue)
```

## Description

`info = lteSLSCFDMAInfo(ue)` returns a structure containing information related to the sidelink SC-FDMA modulation performed by `lteSLSCFDMAModulate`, using the specified UE settings structure.

For details, see “Sidelink SC-FDMA Modulation” on page 2-1122.

## Examples

### Sidelink Waveform Sampling Rate for 5 MHz Channel

Calculate the sampling rate of a 5 MHz sidelink waveform after sidelink SC-FDMA modulation.

Create a UE settings structure. Specify 25 resource blocks, which corresponds to 5 MHz channel bandwidth.

```
ue = struct('NSLRB',25);
```

For the specified channel bandwidth, find the sidelink SC-FDMA modulation sampling rate.

```
slscfdmaInfo = lteSLSCFDMAInfo(ue);
samplingRate = slscfdmaInfo.SamplingRate
```

```
samplingRate = 7680000
```

## Input Arguments

### ue — User equipment settings

structure

User equipment settings, specified as a parameter structure containing these fields:

#### NSLRB — Number of sidelink resource blocks

integer scalar from 6 to 110

Number of sidelink resource blocks, specified as an integer scalar from 6 to 110.

Example: 6, which corresponds to a channel bandwidth of 1.4 MHz.

Data Types: `double`

**`CyclicPrefixSL` — Cyclic prefix length**
`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

**`Windowing` — Number of time-domain samples**
positive integer scalar | optional

Number of time-domain samples over which windowing and overlapping of sidelink SC-FDMA symbols is applied, specified as a positive integer scalar.

`ue.Windowing` must be even. For the `ue.Windowing` field, the default depends on `NSLRB` and `CyclicPrefixSL`.

Data Types: `double`

Data Types: `struct`

## Output Arguments

**`info` — Sidelink SC-FDMA modulated waveform information**
structure

Sidelink SC-FDMA modulated waveform information, returned as a parameter structure containing these fields:

**`SamplingRate` — Sampling rate**
positive numeric scalar

Sampling rate of the time-domain sidelink waveform, in Hz, returned as a positive numeric scalar. `SamplingRate` = `Nfft` × (30.72e6 / 2048).

**`Nfft` — Number of FFT points**
positive integer scalar

Number of FFT points, returned as a positive integer scalar. `Nfft` is a function of the number of resource blocks, $N_{RB}^{SL}$.

| **NSLRB ($N_{RB}^{SL}$)** | **Nfft** |
|---|---|
| 6 | 128 |
| 15 | 256 |
| 25 | 512 |
| 50 | 1024 |
| 75 | 2048 |
| 100 | 2048 |

In general, `Nfft` is the smallest power of 2 greater than or equal to $(12 \times N_{RB}^{SL}) / 0.85$. Specifically, `Nfft` is the smallest FFT that spans all subcarriers and results in no more than 85% of bandwidth occupancy ($12 \times N_{RB}^{SL}$ / `Nfft`).

**`Windowing` — Number of time-domain samples**
positive integer scalar

Number of time-domain samples over which windowing and overlapping of sidelink SC-FDMA symbols is applied, returned as a positive integer scalar.

**`CyclicPrefixLengths` — Cyclic prefix length**
positive integer vector

Cyclic prefix length in symbols for each sidelink SC-FDMA symbol in a subframe, returned as an $N_{SYM}$-by-1 integer vector. $N_{SYM}$ is 14 for normal cyclic prefix and 12 for extended cyclic prefix.

The vector returned for `info.CyclicPrefixLengths` depends on the FFT size.

- When `info.Nfft` = `2048`, then `CyclicPrefixLengths` is:
  - `[160 144 144 144 144 144 144 160 144 144 144 144 144 144]` for normal cyclic prefix
  - `[512 512 512 512 512 512 512 512 512 512 512 512]` for extended cyclic prefix
- For other values of `info.Nfft`, these element values in `CyclicPrefixLengths` are scaled by `info.Nfft` / 2048.

## More About

### Sidelink SC-FDMA Modulation

The sidelink SC-FDMA modulation processing in `lteSLSCFDMAModulate` performs IFFT calculation, half-subcarrier shifting, cyclic prefix insertions, and optional raised-cosine windowing and overlapping of adjacent sidelink SC-FDMA symbols. TS 36.211 specifies that for PSSCH (Section 9.3.6), PSCCH (9.4.6), PSDCH (9.5.6) and PSBCH (9.6.6), resource elements in the last SC-FDMA symbol within a subframe should be counted in the mapping process but not transmitted. Therefore, before performing the IFFT, the last SC-FDMA symbol of each subframe in the input resource grid is set to zero.

For sidelink SC-FDMA modulation, calling `lteSLSCFDMAModulate` on a multi-subframe array of resource grids is recommended.

- When the resource element grid input to `lteSLSCFDMAModulate` spans multiple subframes, the windowing and overlapping is applied between all adjacent SC-FDMA symbols, including the last symbol of the previous subframe and the first symbol of the next subframe. Multi-subframe modulation processing results in a waveform that does not have discontinuities between subframes.
- A time-domain waveform that concatenates individually modulated subframes has discontinuities at the start and end of each subframe. To avoid these discontinuities, the resulting multi-subframe time-domain waveform must be created by manually overlapping symbols at the subframe boundaries.
- If the value for windowing is zero, issues concerning concatenation of subframes before sidelink SC-FDMA modulation do not apply.

If `ue.Windowing` is absent, `info.Windowing` returns a default value chosen as a function of *NRB*. The chosen value is a compromise between:

- The effective duration of cyclic prefix, and therefore the channel delay spread tolerance
- The spectral characteristics of the transmitted signal, not considering any additional FIR filtering

## Version History

**Introduced in R2016b**

## See Also

`lteSLSCFDMAModulate` | `lteSLSCFDMADemodulate`

# lteSLSCFDMAModulate

Sidelink SC-FDMA modulation

## Syntax

```
waveform = lteSLSCFDMAModulate(ue,grid)
[waveform,info] = lteSLSCFDMAModulate(ue,grid)
[ ___ ] = lteSLSCFDMAModulate(ue,grid,windowing)
```

## Description

`waveform = lteSLSCFDMAModulate(ue,grid)` returns a modulated sidelink SC-FDMA waveform for the specified UE settings structure and allocated resource element grid of a number of subframes across one or more antenna planes. For more information, see “Sidelink SC-FDMA Modulation” on page 2-1128.

`[waveform,info] = lteSLSCFDMAModulate(ue,grid)` also returns a SC-FDMA information structure array.

`[ ___ ] = lteSLSCFDMAModulate(ue,grid,windowing)` specifies in `windowing` the number of windowed and overlapped samples to use in the time-domain windowing. For this syntax, the value reported in `info.Windowing` equals `windowing`. Any value provided in `ue.Windowing` is ignored.

This syntax supports output options from prior syntaxes.

## Examples

### Sidelink Broadcast Channel Modulation

Perform sidelink SC-FDMA modulation of one subframe containing a sidelink broadcast transmission. Any resource elements present in the last SC-FDMA symbol of the subframe are not modulated, so the resulting waveform magnitude is zero during that SC-FDMA symbol. Plot the magnitude of the resulting time-domain waveform and the transmitted resource grid magnitude.

#### Create a UE settings structure and an empty resource grid

```
ue.NSLRB = 6;
ue.CyclicPrefixSL = 'Extended';
ue.InCoverage = 1;
ue.DuplexMode = 'FDD';
ue.NFrame = 0;
ue.NSubframe = 0;
ue.NSLID = 42;

grid = lteSLResourceGrid(ue);
```

#### Transmit the PSBCH

Populate the PSBCH resource grid with an encoded SL-MIB message, and its DM-RS. Perform sidelink SC-FDMA modulation.

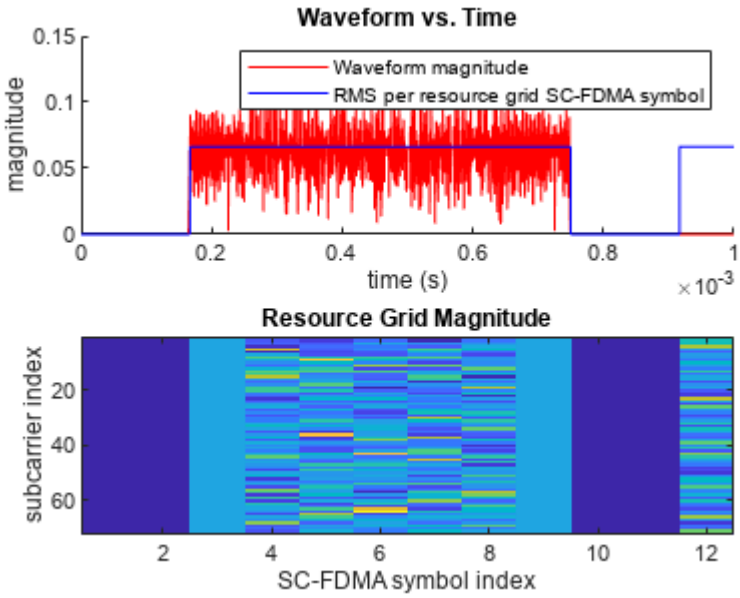
```
grid(ltePSBCHIndices(ue)) = ltePSBCH(ue,lteSLBCH(ue,lteSLMIB(ue)));
grid(ltePSBCHDRSIndices(ue)) = ltePSBCHDRS(ue);

[waveform,info] = lteSLSCFDMAModulate(ue,grid);
```

Calculate the expected RMS for each SC-FDMA symbol from the resource grid prior to modulation.

```
rms = sqrt(sum(abs((grid./double(info.Nfft)).^2)));
```

Plot the waveform magnitude overlaying the RMS for each SC-FDMA symbol. Plot the transmitted resource grid magnitude.

```
t = (0:size(waveform,1))/info.SamplingRate;
figure

subplot(2,1,1)
hold on

plot(t(1:end-1),abs(waveform),'r');
n = cumsum([1 info.CyclicPrefixLengths + info.Nfft]);
n = [n(1:end-1); n(2:end)];
rmsplot = repmat(rms,[2 1]);

plot(t(n(:)),rmsplot(:),'b')
xlabel('time (s)')
ylabel('magnitude')
title('Waveform vs. Time')
legend('Waveform magnitude','RMS per resource grid SC-FDMA symbol')

subplot(2,1,2)
imagesc(abs(grid))
title('Resource Grid Magnitude')
xlabel('SC-FDMA symbol index');
ylabel('subcarrier index');
```




## Input Arguments

### ue — User equipment settings

structure

User equipment settings, specified as a parameter structure containing these fields:

### CyclicPrefixSL — Cyclic prefix length

`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

### Windowing — Number of time-domain samples

positive integer scalar | optional

Number of time-domain samples over which the function applies windowing and overlapping of sidelink SC-FDMA symbols, specified as a positive integer scalar.

`ue.Windowing` must be even. For the `ue.Windowing` field, the default depends on *NRB* and `CyclicPrefixSL`.

Data Types: `double`

Data Types: `struct`

### grid — Resource element grid

numeric 3-D array

Resource element grid, specified as an $N_{SC}$-by-$N_{SYM}$-by-$N_T$ numeric array. $N_{SC}$ must be a multiple of 12 REs per Resource Block, since number of resource blocks is $NRB = N_{SC} / 12$. $N_{SYM}$ must be a multiple of the number of SC-FDMA symbols in a subframe (14 for normal cyclic prefix and 12 for extended

cyclic prefix). $N_T$ is the number of antenna ports. `grid` defines the RE allocation across one or more subframes. Multiple subframes are defined by concatenation across the columns (second dimension).

Each antenna plane in `grid` is SC-FDMA modulated, resulting in the columns of `waveform`, as described in “Represent Resource Grids”.

Data Types: `double`
Complex Number Support: Yes

**`windowing` — Number of time-domain samples**
positive integer scalar | optional

Number of time-domain samples over which windowing and overlapping of sidelink SC-FDMA symbols is applied, specified as a positive integer.

If you specify `windowing` this value is returned in `info.Windowing` and any value provided in `ue.Windowing` is ignored.

Data Types: `double`

## Output Arguments

**`waveform` — Sidelink SC-FDMA modulated waveform**
numeric matrix

Sidelink SC-FDMA modulated waveform, returned as an $N_S$-by-$N_T$ numeric matrix, where $N_S$ is the number of the time-domain samples and $N_T$ is the number of transmission antennas.
$N_S = K \times 30720 / 2048 \times$ `Nfft`, where `Nfft` is the IFFT size and $K$ is the number of subframes in the `grid` input.

**`info` — Sidelink SC-FDMA modulated waveform information**
structure

Sidelink SC-FDMA modulated waveform information, returned as a parameter structure containing these fields:

**`SamplingRate` — Sampling rate**
positive numeric scalar

Sampling rate of the time-domain sidelink waveform, in Hz, returned as a positive numeric scalar. `SamplingRate` = `Nfft` × (30.72e6 / 2048).

**`Nfft` — Number of FFT points**
positive integer scalar

The number of FFT points, returned as a positive integer scalar. `Nfft` is a function of the number of resource blocks (*NRB*)

| ***NRB*** | **`Nfft`** |
|---|---|
| 6 | 128 |
| 15 | 256 |
| 25 | 512 |

| *NRB* | Nfft |
|---|---|
| 50 | 1024 |
| 75 | 2048 |
| 100 | 2048 |

In general, `Nfft` is the smallest power of 2 greater than or equal to (12 × *NRB*) / 0.85. Specifically, `Nfft` is the smallest FFT that spans all subcarriers and results in no more than 85% of bandwidth occupancy (12 × *NRB* / `Nfft`).

**`Windowing` — Number of time-domain samples**
positive integer scalar

Number of time-domain samples over which windowing and overlapping of sidelink SC-FDMA symbols is applied, returned as a positive integer scalar.

**`CyclicPrefixLengths` — Cyclic prefix length**
positive integer vector

Cyclic prefix length in symbols for each sidelink SC-FDMA symbol in a subframe, returned as an $N_{SYM}$-by-1 integer vector. $N_{SYM}$ is 14 for normal cyclic prefix and 12 for extended cyclic prefix.

The vector returned for `info.CyclicPrefixLengths` depends on the FFT size.

- When `info.Nfft = 2048`, then `CyclicPrefixLengths` is:
  - `[160 144 144 144 144 144 144 160 144 144 144 144 144 144]` for normal cyclic prefix
  - `[512 512 512 512 512 512 512 512 512 512 512 512]` for extended cyclic prefix
- For other values of `info.Nfft`, these element values in `CyclicPrefixLengths` are scaled by `info.Nfft` / 2048.

## More About

### Sidelink SC-FDMA Modulation

The sidelink SC-FDMA modulation processing in `lteSLSCFDMAModulate` performs IFFT calculation, half-subcarrier shifting, cyclic prefix insertions, and optional raised-cosine windowing and overlapping of adjacent sidelink SC-FDMA symbols. TS 36.211 specifies that for PSSCH (Section 9.3.6), PSCCH (9.4.6), PSDCH (9.5.6) and PSBCH (9.6.6), resource elements in the last SC-FDMA symbol within a subframe should be counted in the mapping process but not transmitted. Therefore, before performing the IFFT, the last SC-FDMA symbol of each subframe in the input resource grid is set to zero.

For sidelink SC-FDMA modulation, calling `lteSLSCFDMAModulate` on a multi-subframe array of resource grids is recommended.

- When the resource element grid input to `lteSLSCFDMAModulate` spans multiple subframes, the windowing and overlapping is applied between all adjacent SC-FDMA symbols, including the last symbol of the previous subframe and the first symbol of the next subframe. Multi-subframe modulation processing results in a waveform that does not have discontinuities between subframes.

- A time-domain waveform that concatenates individually modulated subframes has discontinuities at the start and end of each subframe. To avoid these discontinuities, the resulting multi-subframe time-domain waveform must be created by manually overlapping symbols at the subframe boundaries.
- If the value for windowing is zero, issues concerning concatenation of subframes before sidelink SC-FDMA modulation do not apply.

If `ue.Windowing` is absent, `info.Windowing` returns a default value chosen as a function of *NRB*. The chosen value is a compromise between:

- The effective duration of cyclic prefix, and therefore the channel delay spread tolerance
- The spectral characteristics of the transmitted signal, not considering any additional FIR filtering

# Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteSLSCFDMADemodulate` | `lteSLSCFDMAInfo` | `lteSLResourceGridSize`

# lteSLSCH

Sidelink shared channel

## Syntax

```
cw = lteSLSCH(ue,outlen,trblkin)
```

## Description

`cw = lteSLSCH(ue,outlen,trblkin)` returns the codeword column vector for the specified UE settings structure and output length. `lteSLSCH` applies the complete sidelink shared channel (SL-SCH) transport channel processing to the input data, `trblkin`.

For more information, see “Sidelink Shared Transport Channel Processing” on page 2-1132.

## Examples

### Create and Decode SL-SCH Codeword

Use the physical channel bit capacity information to configure the output codeword size for SL-SCH coding. Decode the resulting codeword and check for CRC errors.

```
ue = struct('NSLRB',50,'CyclicPrefixSL','Normal');
ue.PRBSet = (10:12)';
ue.Modulation = '16QAM';
ue.RV = 0;

[~,psschinfo] = ltePSSCHIndices(ue);
cwlength = psschinfo.G;

trblk = randi([0 1],100,1);
cw = lteSLSCH(ue,cwlength,trblk);
[rxtrblk,err] = lteSLSCHDecode(ue,length(trblk),cw);
err
```

```
err = logical
   0
```

The transport block is recovered with no error.

### Create SL-SCH Codeword Sequence

Create a cell array containing the redundancy version (RV) sequence of four codewords that is ready for transmission on the PSSCH.

Initialize a UE settings structure.

```
ue = struct('NSLRB',50,'CyclicPrefixSL','Normal');
ue.PRBSet = (10:12)';
ue.Modulation = '16QAM';
```

Use the physical channel bit capacity information to configure the output codeword size for SL-SCH coding. Create a transport block of information bits.

```
[~,psschinfo] = ltePSSCHIndices(ue);
cwlength = psschinfo.G;

trblk = randi([0 1],100,1);
```

Use a `for` loop to create a cell array containing the sequence of four SL-SCH codewords. `RV` = 0,2,3,1 for transmission on the PSSCH.

```
rvseq = [0 2 3 1];
for ii = 1:length(rvseq)
    ue.RV = rvseq(ii);
    cwseq = lteSLSCH(ue,cwlength,trblk);
    cwseqCell{ii} = cwseq;
end
```

Alternatively, the same cell array of SL-SCH codeword sequences can be created using an anonymous function handle.

```
rvseq = [0 2 3 1];

cwgenfn = @(rv)lteSLSCH(setfield(ue,'RV',rv),cwlength,trblk); %#ok<SFLD>

cwseqCell2 = arrayfun(cwgenfn,rvseq,'UniformOutput',false);
```

## Input Arguments

### ue — User equipment settings

structure

User equipment settings, specified as a parameter structure containing these fields:

#### SidelinkMode — Sidelink mode

`'D2D'` (default) | `'V2X'` | optional

Sidelink mode, specified as `'D2D'` or `'V2X'`.

Data Types: `char` | `string`

#### CyclicPrefixSL — Cyclic prefix length

`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

#### Modulation — Modulation type

`'QPSK'` | `'16QAM'`

Modulation type, specified as `'QPSK'` or `'16QAM'`.

Data Types: `char` | `string`

**`RV` — Redundancy version indicator**
0 | 1 | 2 | 3 | vector with element values from 0 to 3

Redundancy version indicator, specified as an integer scalar or vector with element values from 0 to 3.

Example: `[0 2 3 1]`, indicates the RV sequence order for transmission on the PSSCH.

Data Types: `double`

Data Types: `struct`

**`outlen` — Codeword length**
integer scalar

Codeword length, specified as an integer scalar. For more information, see “Sidelink Shared Transport Channel Processing” on page 2-1132.

Data Types: `double`

**`trblkin` — Transport block data bits**
bit vector

Transport block data bits, specified as a bit vector.

Data Types: `double`

## Output Arguments

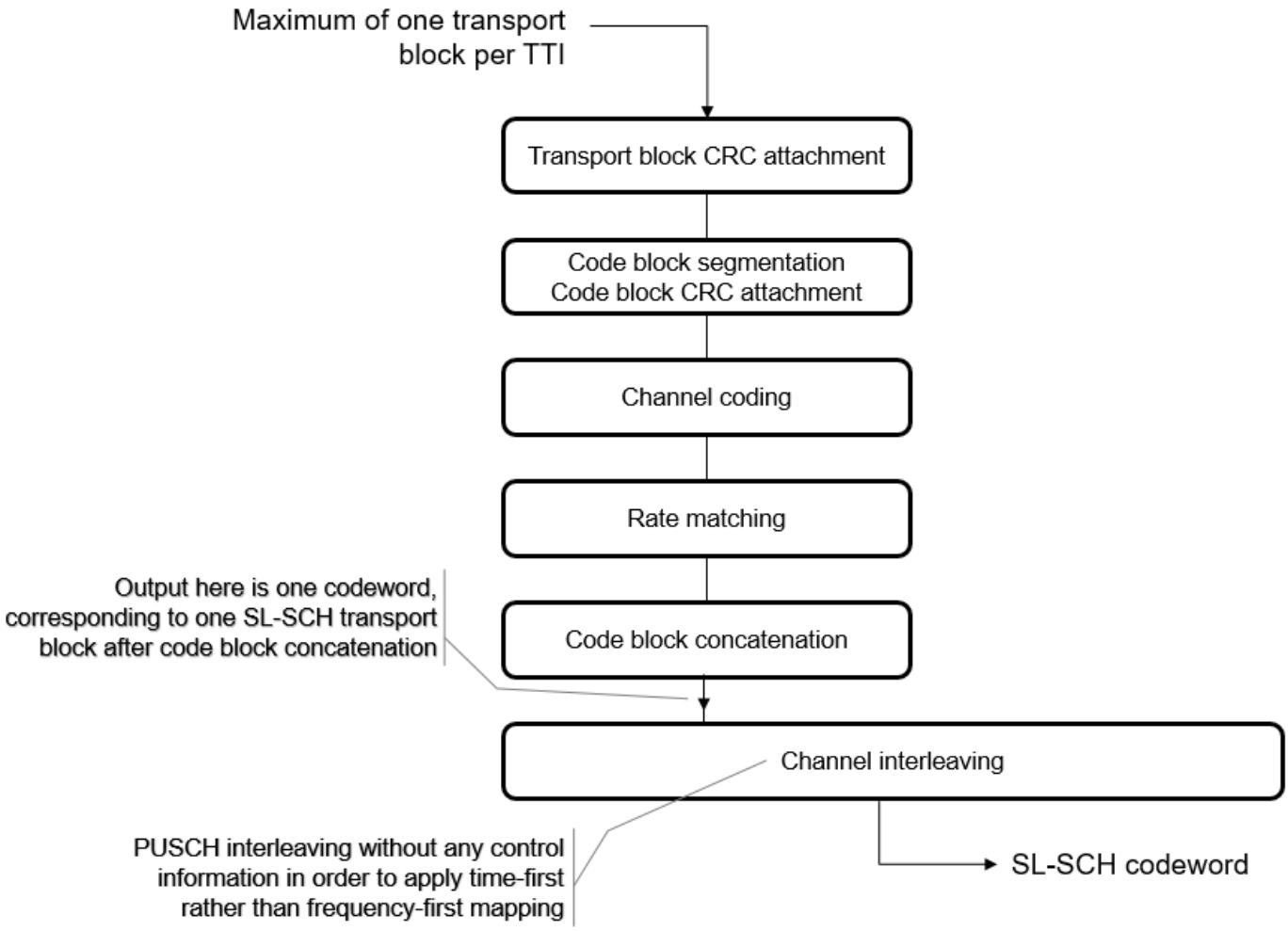
**`cw` — PSSCH codeword**
integer vector

PSSCH codeword, returned as an $M_{bit}$-by-1 integer vector. $M_{bit}$ is equal to `outlen` and is the number of bits transmitted on the physical sidelink shared channel in one subframe. `outlen` must be a multiple of the number of bits per symbol. For more information, see “Sidelink Shared Transport Channel Processing” on page 2-1132.

## More About

### Sidelink Shared Transport Channel Processing

The sidelink shared channel (SL-SCH) transport channel processing includes type-24A CRC calculation, code block segmentation (including type-24B CRC attachment, if present), turbo encoding, rate matching with redundancy version (RV), code block concatenation, and PUSCH interleaving. `lteSLSCH` generates this transport channel codeword as specified by TS 36.212, Section 5.4.2.




The SL-SCH transport channel codeword carrying the information bits of a single transport block is transmitted on the physical sidelink shared channel. Use the `ltePSSCH` and `ltePSSCHIndices` functions to generate the modulated symbols and populate the resource grid for transmission.

The length of the codeword output by `lteSLSCH` represents the bit capacity of the physical channel. For PSSCH, the input codeword length is $M_{\text{bits}} = N_{\text{RE}} \times N_{\text{bps}}$, where $N_{\text{bps}}$ is the number of bits per symbol. The PSSCH modulation is either QPSK (2 bits per symbol) or 16QAM (4 bits per symbol). The number of PSSCH resource elements ($N_{\text{RE}}$) in a subframe is $N_{\text{RE}} = N_{\text{PRB}} \times N_{\text{REperPRB}} \times N_{\text{SYM}}$ and includes symbols associated with the sidelink SC-FDMA guard symbol.

- $N_{\text{PRB}}$ is the number of physical resource blocks (PRB) used for transmission.
- $N_{\text{REperPRB}}$ is the number of resource elements in a PRB. Each PRB has 12 resource elements.
- $N_{\text{SYM}}$ is the number of SC-FDMA symbols in a PSSCH subframe, including symbols associated with the sidelink SC-FDMA guard symbol. $N_{\text{SYM}}$ is 12 for D2D normal cyclic prefix or 10 for D2D extended cyclic prefix and V2X.

For D2D sidelink, the SL-SCH codeword carrying the information bits of a single transport block is always transmitted four times on four consecutive PSSCH subframes using the fixed RV sequence, RV = 0,2,3,1. The transmission subframes are selected from a subset of the PSSCH subframe pool. There is no HARQ feedback involved in the process. For V2X, there can be either one or two transmissions

of a transport block using the RV sequence, RV = 0,2. For more information on the SL-SCH transmission and the sidelink HARQ process, see TS 36.321, Section 5.14.2.2.

# Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.321. “Evolved Universal Terrestrial Radio Access (E-UTRA); Medium Access Control (MAC) protocol Specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteSLSCHDecode` | `ltePSSCH`

# lteSLSCHDecode

Sidelink shared channel decoding

## Syntax

```
trblkout,blkcrc,stateout = lteSLSCHDecode(ue,trblklen,cwin)
[trblkout,blkcrc,stateout] = lteSLSCHDecode(ue,trblklen,cwin,statein)
```

## Description

`trblkout,blkcrc,stateout = lteSLSCHDecode(ue,trblklen,cwin)` returns a column vector of information bits,`trblkout`, decoded from the soft log-likelihood ratio (LLR) codeword data vector `cwin` for the specified UE settings structure and transport block length. Additional outputs contains the result from a block cyclic redundancy check, `blkcrc` and a structure containing the HARQ process decoding state, `stateout`.

The SL-SCH decoder processing includes PUSCH deinterleaving, rate recovery, turbo decoding, block concatenation, and CRC calculations. The SL-SCH decoder performs the inverse of the sidelink shared channel processing defined in TS 36.212 [1], Section 5.4.2. For more information, see “Sidelink Shared Transport Channel Processing” on page 2-1139.

`[trblkout,blkcrc,stateout] = lteSLSCHDecode(ue,trblklen,cwin,statein)` accepts an input structure specifying the initial HARQ process state that is used in support of HARQ soft combining.

The `stateout` array is normally reapplied via the `statein` argument of subsequent `lteSLSCHDecode` function calls, as part of a fixed sequence of HARQ retransmissions used by the SL-SCH. When a transport block is transmitted, it is always sent four times on four consecutive PSSCH subframes using the fixed RV sequence of {0,2,3,1}. The consecutive PSSCH subframes are selected from a subset of the PSSCH subframe pool. The `statein` and `stateout` variables allow this set of transmissions to be soft combined.

## Examples

### Decode SL-SCH Transport Channel

Encode and decode an information block using the SL-SCH transport channel.

Create a UE settings structure. Generate a 100-bit transport block and SL-SCH codeword.

```
ue = struct('CyclicPrefixSL','Normal','Modulation','16QAM','RV',0);
trblk = randi([0 1],100,1);
cw = lteSLSCH(ue,5760,trblk);
```

Decode the SL-SCH codeword.

```
rxtrblk = lteSLSCHDecode(ue,length(trblk),cw);
```

### Decode SL-SCH Transport Channel and Check CRC

Encode and decode an information block using the SL-SCH transport channel, and display the CRC error result.

Create a UE settings structure. Generate a 100-bit transport block and SL-SCH codeword.

```
ue = struct('CyclicPrefixSL','Normal','Modulation','16QAM','RV',0);
trblk = randi([0 1],100,1);
cw = lteSLSCH(ue,5760,trblk);
```

Decode the SL-SCH codeword and check for block errors.

```
[rxtrblk,err] = lteSLSCHDecode(ue,length(trblk),cw);
err
```

```
err = logical
   0
```

The decoded transport block has no errors.

### Decode SL-SCH Using HARQ Soft Combining

Use soft combining while decoding the sequence of four transmissions used to send every transport block on the SL-SCH. The rate matching and noise level are set so that successful decoding of the block requires multiple transmissions.

#### Initialize parameters

- Create a UE settings structure for the SL-SCH.
- Generate a transport block of 100 random bits.
- Create a local variable specifying an SL-SCH bit capacity of 288.
- Define the fixed redundancy version sequence used by the HARQ process.
- Clear the HARQ process decoding state.

```
ue = struct('CyclicPrefixSL','Normal','Modulation','QPSK');
trblk = randi([0 1],100,1);
bitcapacity = 288;
rvseq = [0 2 3 1];
decstate = [];
```

#### Transmit and recover the SL-SCH transport block

- Send the transport block four times.
- Display result of decoding successive transmissions.

```
for i = 1:4
    % Encode information bits with the next RV value.
    ue.RV = rvseq(i);
    cw = lteSLSCH(ue,bitcapacity,trblk);

    % Modulate the codeword and add noise.
```

```
    sym = awgn(lteSymbolModulate(cw,ue.Modulation),-4,'measured');
    softdata = lteSymbolDemodulate(sym,ue.Modulation);

    % Decode the current transmission and combine with decoding state.
    [rxtrblk,err,decstate] = lteSLSCHDecode(ue,length(trblk), ...
        softdata,decstate);
    X = ['Decoding error ', num2str(err), ' for transmission #', ...
        num2str(i), ' with RV ', num2str(ue.RV)];
    disp(X)
end
```

```
Decoding error 1 for transmission #1 with RV 0
Decoding error 1 for transmission #2 with RV 2
Decoding error 0 for transmission #3 with RV 3
Decoding error 0 for transmission #4 with RV 1
```

The soft-combined data is recovered without error on the third transmission.

## Input Arguments

### ue — User equipment settings

structure

User equipment settings, specified as a parameter structure containing these fields:

#### SidelinkMode — Sidelink mode

`'D2D'` (default) | `'V2X'` | optional

Sidelink mode, specified as `'D2D'` or `'V2X'`.

Data Types: `char` | `string`

#### CyclicPrefixSL — Cyclic prefix length

`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

#### Modulation — Modulation type

`'QPSK'` | `'16QAM'`

Modulation type, specified as `'QPSK'` or `'16QAM'`.

Data Types: `char` | `string`

#### RV — Redundancy version indicator

0 | 1 | 2 | 3 | vector with element values from 0 to 3

Redundancy version indicator, specified as an integer scalar or vector with element values from 0 to 3.

Example: `[0 2 3 1]`, indicates the RV sequence order for transmission on the PSSCH.

Data Types: `double`

#### NTurboDecIts — Number of turbo decoder iteration cycles

5 (default) | integer scalar from 1 to 30 | optional

Number of turbo decoder iteration cycles, specified as an integer scalar from 1 to 30.

Data Types: `double`

Data Types: `struct`

**`trblklen` — Transport block length**
positive integer scalar

Transport block length, specified as a positive integer scalar. `trblklen` defines the decoded transport block length.

Data Types: `double`

**`cwin` — LLR codeword data**
bit vector

LLR codeword data, specified as a soft bit vector.

Data Types: `double`

**`statein` — Decoder buffer state**
structure | optional

Decoder buffer state, specified as a structure. Use `statein` to input the current decoder buffer state for the transport block in an active HARQ process. `statein` can be an empty structure or a structure array with one or two elements. If nonempty, `statein`.`CBSBuffers` should contain a cell array of vectors representing the log-likelihood ratio (LLR) soft buffer states for the set of code blocks at the input to the turbo decoder, after explicit rate recovery. The updated buffer states after decoding are returned in the `CBSBuffers` field of `stateout`.

The `statein` array is normally generated and recycled from the `stateout` of previous calls to `lteSLSCHDecode`, as part of the fixed sequence of SL-SCH HARQ (re)transmissions.

The `statein` structure contains this field:

**`CBSBuffers` — LLR soft buffer states**
cell array of numeric vectors

LLR soft buffer states, specified as a cell array of numeric vectors. `CBSBuffers` contains the LLR soft buffer states for the set of code blocks associated with a single transport block. The LLR soft buffer states are positioned at the input to the turbo decoder. The states are available after the explicit rate recovery.

Data Types: `cell`

Data Types: `struct`

## Output Arguments

**`trblkout` — Decoded information bits**
integer column vector

Decoded information bits, returned as a column vector. The `trblkout` information bits are decoded from the soft log-likelihood ratio (LLR) codeword data vector, `cwin`.

**`blkcrc` — CRC failure check of block**
`true` | `false`

CRC failure check of block, returned as `true` or `false`.

- `blkcrc` = `false` indicates that the subframe was recovered with no block errors.
- `blkcrc` = `true` indicates a block error.

**`stateout` — Internal state of decoder**
structure

Internal state of decoder, returned as a structure containing these fields:

**`CBSBuffers` — LLR soft buffer states**
cell array of integer vectors

LLR soft buffer states, returned as a cell array of integer vectors. `CBSBuffers` contains the LLR soft buffer states for the set of code blocks associated with a single transport block. The LLR soft buffer states are positioned at the input to the turbo decoder. The states are available after the explicit rate recovery.

Data Types: `cell`

**`CBSCRC` — CRC decoding results of type-24B code block set**
integer array | empty array

CRC decoding results of type-24B code block set, returned as an integer array or empty array.

Data Types: `double`

**`BLKCRC` — CRC decoding error in type-24A transport block**
logical

CRC decoding error in type-24A transport block, returned as a logical.

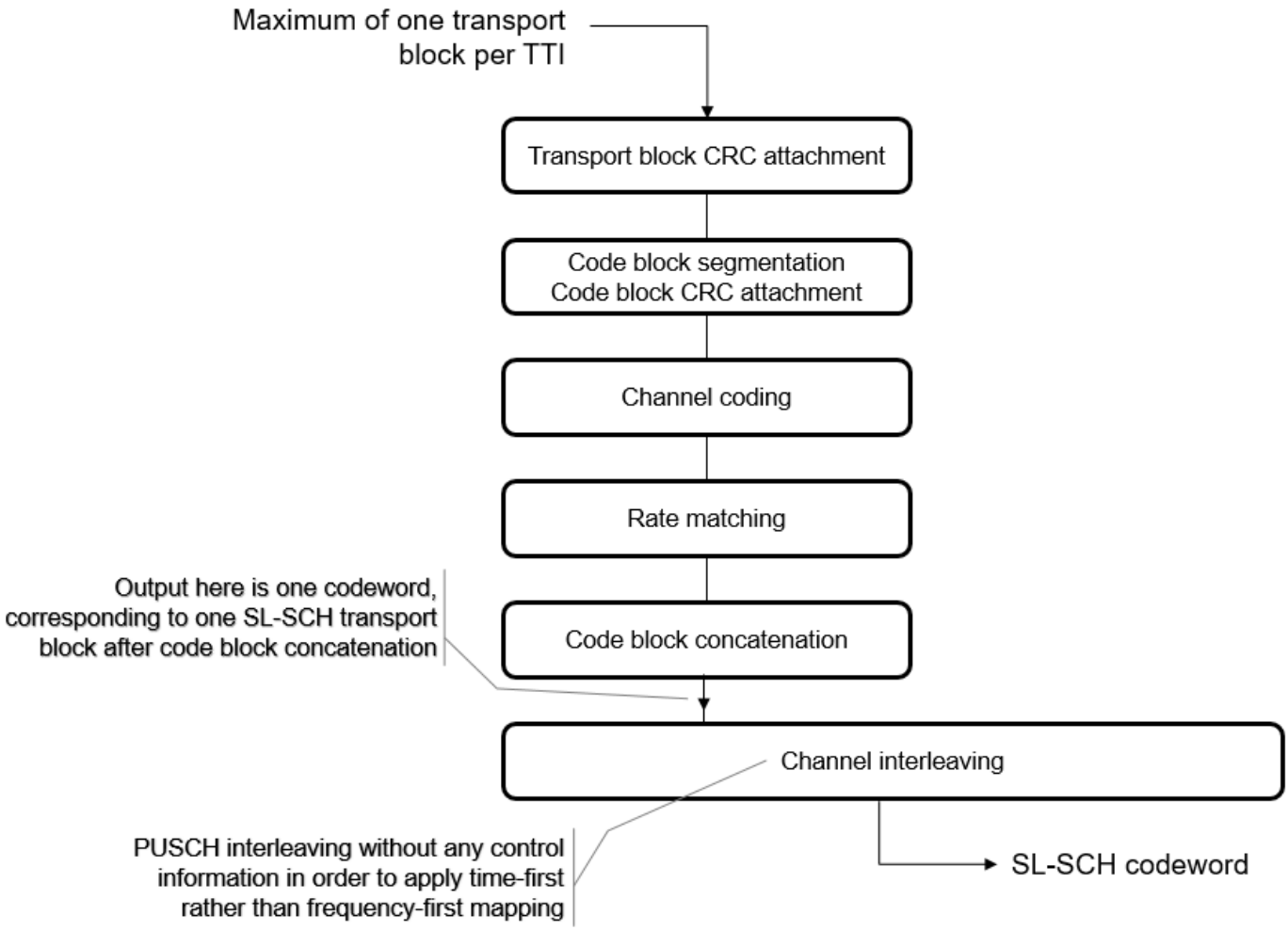
- `BLKCRC` = 0 indicates that the subframe was recovered with no block errors.
- `BLKCRC` = 1 indicates a block error.

Data Types: `logical`

## More About

### Sidelink Shared Transport Channel Processing

The sidelink shared channel (SL-SCH) transport channel processing includes type-24A CRC calculation, code block segmentation (including type-24B CRC attachment, if present), turbo encoding, rate matching with redundancy version (RV), code block concatenation, and PUSCH interleaving. `lteSLSCH` generates this transport channel codeword as specified by TS 36.212, Section 5.4.2.




The SL-SCH transport channel codeword carrying the information bits of a single transport block is transmitted on the physical sidelink shared channel. Use the `ltePSSCH` and `ltePSSCHIndices` functions to generate the modulated symbols and populate the resource grid for transmission.

The length of the codeword output by `lteSLSCH` represents the bit capacity of the physical channel. For PSSCH, the input codeword length is $M_{\text{bits}} = N_{\text{RE}} \times N_{\text{bps}}$, where $N_{\text{bps}}$ is the number of bits per symbol. The PSSCH modulation is either QPSK (2 bits per symbol) or 16QAM (4 bits per symbol). The number of PSSCH resource elements ($N_{\text{RE}}$) in a subframe is $N_{\text{RE}} = N_{\text{PRB}} \times N_{\text{REperPRB}} \times N_{\text{SYM}}$ and includes symbols associated with the sidelink SC-FDMA guard symbol.

- $N_{\text{PRB}}$ is the number of physical resource blocks (PRB) used for transmission.
- $N_{\text{REperPRB}}$ is the number of resource elements in a PRB. Each PRB has 12 resource elements.
- $N_{\text{SYM}}$ is the number of SC-FDMA symbols in a PSSCH subframe, including symbols associated with the sidelink SC-FDMA guard symbol. $N_{\text{SYM}}$ is 12 for D2D normal cyclic prefix or 10 for D2D extended cyclic prefix and V2X.

For D2D sidelink, the SL-SCH codeword carrying the information bits of a single transport block is always transmitted four times on four consecutive PSSCH subframes using the fixed RV sequence, RV = 0,2,3,1. The transmission subframes are selected from a subset of the PSSCH subframe pool. There is no HARQ feedback involved in the process. For V2X, there can be either one or two transmissions

of a transport block using the RV sequence, RV = 0,2. For more information on the SL-SCH transmission and the sidelink HARQ process, see TS 36.321, Section 5.14.2.2.

# Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.321. “Evolved Universal Terrestrial Radio Access (E-UTRA); Medium Access Control (MAC) protocol Specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteSLSCH` | `ltePSSCHDecode`

# lteSRS

Uplink sounding reference signal

## Syntax

```
seq = lteSRS(ue,chs)
[seq,info] = lteSRS(ue,chs)
```

## Description

`seq = lteSRS(ue,chs)` returns a complex matrix, `seq`, containing uplink sounding reference signal (SRS) values and information structure array given structures containing UE-specific settings, and signal transmission configuration settings. For more information, see “SRS Processing” on page 2-1147 and TS 36.213 [1], Section 8.2.

`[seq,info] = lteSRS(ue,chs)` also returns an SRS information structure array, `info`.

## Examples

### Generate Uplink SRS Values

This example generates SRS values for 1.4 MHz bandwidth using the default SRS configuration.

Set the signal transmission configuration, chs structure fields.

```
chs.BWConfig = 7;
chs.BW = 0;
chs.CyclicShift = 0;
chs.SeqGroup = 0;
chs.SeqIdx = 0;
chs.ConfigIdx = 7;
```

Set ue structure fields.

```
ue.DuplexMode = 'FDD';
ue.CyclicPrefixUL = 'Normal';
ue.NTxAnts = 1;
ue.NFrame = 0;
ue.NULRB = 6;
ue.NSubframe = 0;
```

Generate Uplink SRS resource element values.

```
srs = lteSRS(ue,chs);
srs(1:4)
```

```
ans = 4×1 complex

   0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
```

```
  -0.7071 - 0.7071i
```

### Generate SRS Symbols for Two Antennas

Generate the SRS symbols for two transmit antenna paths. Display the information structure.

Initialize UE-specific and channel configuration structures (`ue` and `chs`) for 3 MHz bandwidth and two antennas using the default SRS configuration. Generate SRS symbols and the information structure (`ind` and `info`).

```
ue.DuplexMode = 'FDD';
ue.CyclicPrefixUL = 'Normal';
ue.NFrame = 0;
ue.NULRB = 15;
ue.NSubframe = 0;

chs = struct();
chs.NTxAnts = 2;
chs.BWConfig = 7;
chs.BW = 0;
chs.CyclicShift = 0;
chs.ConfigIdx = 7;
chs.SeqIdx = 0;
chs.SeqGroup = 0;

[ind,info] = lteSRS(ue,chs);
```

Since there are two antennas, the SRS symbols are output as a two column vector and the `info` output structure contains two elements.

```
ind(1:6,:)
```

```
ans = 6×2 complex

   0.5000 - 0.5000i   0.5000 - 0.5000i
  -0.5000 + 0.5000i   0.5000 - 0.5000i
   0.5000 + 0.5000i   0.5000 + 0.5000i
  -0.5000 - 0.5000i   0.5000 + 0.5000i
  -0.5000 + 0.5000i  -0.5000 + 0.5000i
   0.5000 - 0.5000i  -0.5000 + 0.5000i
```

```
size(info)
```

```
ans = 1×2

     1     2
```

View the contents of the two `info` structure elements.

```
info(1)
```

```
ans = struct with fields:
       Alpha: 0
```

```
    SeqGroup: 0
      SeqIdx: 0
     RootSeq: -1
         NZC: -1
```

```
info(2)
```

```
ans = struct with fields:
       Alpha: 3.1416
    SeqGroup: 0
      SeqIdx: 0
     RootSeq: -1
         NZC: -1
```

## Input Arguments

### `ue` — UE-specific settings

structure

UE-specific settings, specified as a structure containing these following fields.

### `NULRB` — Number of uplink resource blocks

positive integer

Number of uplink resource blocks, specified as a positive integer.

Data Types: `double`

### `NSubframe` — Subframe number

0 (default) | optional | nonnegative integer

Subframe number, specified as a nonnegative integer.

Data Types: `double`

### `NTxAnts` — Number of transmission antennas

1 (default) | optional | 2 | 4

Number of transmission antennas, specified as 1, 2, or 4.

Data Types: `double`

### `CyclicPrefixUL` — Cyclic prefix length for uplink

`'Normal'` (default) | optional | `'Extended'`

Cyclic prefix length for uplink, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

### `NFrame` — Initial frame number

0 (default) | optional | nonnegative integer

Initial frame number, specified as a nonnegative integer.

Data Types: `double`

**`DuplexMode` — Duplexing mode**
`'FDD'` (default) | optional | `'TDD'`

Duplexing mode, specified as `'FDD'` or `'TDD'` to indicate the frame structure type of the generated waveform.

Example: `'TDD'`

Data Types: `char` | `string`

**`TDDConfig` — Uplink or downlink configuration**
0 (default) | integer from 0 to 6 | optional

Uplink or downlink configuration, specified as an integer from 0 to 6. Only required for `TDD` duplex mode.

Data Types: `double`

**`SSC` — Special subframe configuration**
0 (default) | integer from 0 to 9 | optional

Special subframe configuration, specified as an integer from 0 to 9. Only required for `TDD` duplex mode.

Data Types: `double`

**`CyclicPrefix` — Cyclic prefix length in the downlink**
`'Normal'` (default) | optional | `'Extended'`

Cyclic prefix length in the downlink, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

Data Types: `struct`

**`chs` — Signal transmission configuration**
structure

Signal transmission configuration, specified as a structure containing these fields.

**`NTxAnts` — Number of transmission antennas**
1 (default) | optional | 2 | 4

Number of transmission antennas, specified as 1, 2, or 4.

Data Types: `double`

**`BWConfig` — SRS bandwidth configuration**
7 (default) | integer from 0 to 7 | optional

SRS bandwidth configuration, specified as an integer from 0 to 7. ($C_{\mathrm{SRS}}$)

Data Types: `double`

**`BW` — UE-specific SRS bandwidth**
0 (default) | optional | 1 | 2 | 3

UE-specific SRS bandwidth, specified as an integer from 0 to 3. ($B_{\mathrm{SRS}}$)

Data Types: `double`

**`ConfigIdx` — Configuration index for UE-specific periodicity**
7 (default) | integer from 0 to 644 | optional

Configuration index for UE-specific periodicity, specified as a nonnegative integer from 0 to 644. This parameter contains the configuration index for UE-specific periodicity ($T_{\mathrm{SRS}}$) and subframe offset ($T_{\mathrm{offset}}$).

Data Types: `double`

**`CyclicShift` — UE-specific cyclic shift**
0 (default) | integer from 0 to 7 | optional

UE-specific cyclic shift, specified as an integer from 0 to 7. ($n_{SRS}^{cs}$)

Data Types: `double`

**`SeqGroup` — SRS sequence group number**
0 (default) | integer from 0 to 29 | optional

SRS sequence group number, specified as an integer from 0 to 29. ($u$)

Data Types: `double`

**`SeqIdx` — Base sequence number**
0 (default) | optional | 1

Base sequence number, specified as either 0 or 1. ($v$)

Data Types: `double` | `logical`

**`OffsetIdx` — SRS subframe offset**
0 (default) | optional | 1

SRS subframe offset choice for 2 ms SRS periodicity, specified as 0 or 1. Only required for `'TDD'` duplex mode. This parameter indexes the two SRS subframe offset entries in the row of TS 36.213 [1], Table 8.2-2 for the SRS configuration index specified by the `ConfigIdx` parameter.

Data Types: `double`

Data Types: `struct`

## Output Arguments

**`seq` — Uplink SRS values**
complex matrix

Uplink SRS values, returned as a complex matrix. The symbols for each antenna are in the columns of the matrix, `seq`. The symbols for each antenna are in the columns of `seq`, with the number of columns determined by the number of transmission antennas configured. For more information, see “SRS Processing” on page 2-1147.

Data Types: `double`

**`info` — Information related to SRS**
structure

Information related to SRS, returned as a structure array with elements corresponding to each transmit antenna and containing these fields.

**`Alpha` — Reference signal cyclic shift**
numeric scalar

Reference signal cyclic shift, returned as a numeric scalar. ($\alpha$)

Data Types: `double`

**`SeqGroup` — SRS sequence group number**
0,...,29

SRS sequence group number, returned as an integer from 0 to 29. ($u$)

Data Types: `double`

**`SeqIdx` — Base sequence number**
0 | 1

Base sequence number, returned as 0 or 1. ($v$)

Data Types: `double`

**`RootSeq` — Root Zadoff-Chu sequence index**
integer

Root Zadoff-Chu sequence index, returned as an integer. ($q$)

Data Types: `double`

**`NZC` — Zadoff-Chu sequence length**
integer

Zadoff-Chu sequence length, returned as an integer. ($N_{ZC}^{RS}$)

Data Types: `double`

Data Types: `struct`

## More About

### SRS Processing

As specified in TS 36.213, Section 8.2, a UE shall transmit the sounding reference symbol (SRS) on per serving cell SRS resources, based on two trigger types:

- trigger type 0 — periodic SRS from higher layer signalling
- trigger type 1 — aperiodic SRS from DCI formats 0/4/1A for FDD or TDD and from DCI formats 2B/2C/2D for TDD.

The parameter `chs.ConfigIdx` indexes Tables 8.2-1, 8.2-2, 8.2-4, and 8.2-5 defined in TS 36.213, Section 8.2. The applicable table and appropriate range of `chs.ConfigIdx` depends on the duplex mode and the SRS trigger type.

If type 0 triggered SRS transmission is intended, then:

- The valid range of `chs.ConfigIdx` ($I_{SRS}$) is from 0 to 636 for FDD (Table 8.2-1) and from 0 to 644 for TDD (Table 8.2-2).

If type 1 triggered SRS transmission is intended, then:

- `chs.ConfigIdx` indexes trigger type 1 UE-specific periodicity $T_{\mathrm{SRS},1}$ and subframe offset $T_{\mathrm{offset},1}$. The valid range of `chs.ConfigIdx` ($I_{SRS}$) is from 0 to 16 for FDD (Table 8.2-4) and from 0 to 24 for TDD (Table 8.2-5).
- Frequency hopping is not permitted. Therefore, set `chs.HoppingBW` to be greater than or equal to `BW`. ($b_{\mathrm{hop}} \geq B_{\mathrm{SRS}}$).

To control whether to call the `lteSRS` and `lteSRSIndices` functions in a subframe, use `info.IsSRSSubframe`, returned by `lteSRSInfo`.

UE-specific configurations determine how `lteSRS` and `lteSRSIndices` operate. When no SRS is scheduled, calling `lteSRS` or `lteSRSIndices` in a subframe:

- May generate an SRS depending on the cell-specific SRS subframe configuration.
- Returns an empty `seq` or `ind` vector, for a given UE-specific SRS configuration. Also, the `info` structure scalar fields are set to –1, and any undefined vector fields are empty.

For short-base reference sequences, used with SRS transmissions spanning 4 PRBs, the `lteSRS` function does not use Zadoff Chu sequences and it sets `info.RootSeq` and `info.NZC` to –1.

`lteSRSIndices` returns the UE-specific SRS periodicity, `info.UePeriod`, and subframe offset, `info.UeOffset`. These parameters are distinct from the cell-specific SRS periodicity and subframe offset that `lteSRSInfo` returns.

If `chs.NTxAnts` is not present, `ue.NTxAnts` is used. If neither is present, the function assumes one antenna. In `lteSRSIndices`, for SRS transmission on multiple antennas:

- When `chs.NTxAnts` is set to 2 or 4, the value of `info.Port` matches the position in the structure array (0,...,`NTxAnts` – 1).
- If `chs.NTxAnts` is set to 1, `lteSRSIndices` uses `info.Port` to indicate the port chosen by SRS transmit antenna selection. `info.Port` indicates the selected antenna port, 0 or 1.

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.213. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteSRSIndices` | `lteSRSInfo` | `lteCellRS` | `lteCSIRS` | `lteDMRS` | `ltePRS`

# lteSRSIndices

Uplink SRS resource element indices

## Syntax

```
ind = lteSRSIndices(ue,chs)
[ind,info] = lteSRSIndices(ue,chs)
[___] = lteSRSIndices(ue,chs,opts)
```

## Description

`ind = lteSRSIndices(ue,chs)` returns a column vector of resource element (RE) indices for the Uplink sounding reference signal (SRS) given structures with the UE-specific settings, and the signal transmission configuration settings. For more information, see “SRS Processing” on page 2-1158 and TS 36.213 [1], Section 8.2.

`[ind,info] = lteSRSIndices(ue,chs)` also returns an SRS information structure array, `info`.

`[___] = lteSRSIndices(ue,chs,opts)` formats the returned indices using options specified by `opts`.

This syntax supports output options from prior syntaxes.

## Examples

### Generate Uplink SRS Indices

This example creates SRS indices for 3 MHz bandwidth.

Set the signal transmission configuration, `chs` structure fields.

```
chs.NTxAnts = 1;
chs.BWConfig = 7;
chs.BW = 0;
chs.ConfigIdx = 7;
chs.TxComb = 0;
chs.HoppingBW = 0;
chs.FreqPosition = 0;
```

Set `ue` structure fields.

```
ue.DuplexMode = 'FDD';
ue.CyclicPrefixUL = 'Normal';
ue.NFrame = 0;
ue.NULRB = 15;
ue.NSubframe = 0;
```

Generate the Uplink SRS resource element indices.

```
srsIndices = lteSRSIndices(ue,chs);
srsIndices(1:4)
```

```
ans = 4x1 uint32 column vector

   2401
   2403
   2405
   2407
```

### Generate SRS Indices for Two Antennas

Generate the SRS indices for two transmit antenna paths. Display the information structure.

Initialize UE-specific and channel configuration structures (`ue` and `chs`) for 3 MHz bandwidth and two antennas. Generate SRS indices and the information structure (`ind` and `info`).

```
ue.DuplexMode = 'FDD';
ue.CyclicPrefixUL = 'Normal';
ue.NFrame = 0;
ue.NULRB = 15;
ue.NSubframe = 0;

chs.NTxAnts = 2;
chs.BWConfig = 7;
chs.BW = 0;
chs.ConfigIdx = 7;
chs.TxComb = 0;
chs.HoppingBW = 0;
chs.FreqPosition = 0;

[ind,info] = lteSRSIndices(ue,chs);
```

Since there are two antennas, the SRS indices are output as a two column vector and the `info` output structure contains two elements.

```
ind(1:10,:)

ans = 10x2 uint32 matrix

   2401   4921
   2403   4923
   2405   4925
   2407   4927
   2409   4929
   2411   4931
   2413   4933
   2415   4935
   2417   4937
   2419   4939


size(info)

ans = 1×2

     1     2
```

View the contents of the two `info` structure elements.

```
info(1)
```

```
ans = struct with fields:
         UePeriod: 10
         UeOffset: 0
           PRBSet: [4x1 double]
        FreqStart: 60
          KTxComb: 0
         BaseFreq: 60
          FreqIdx: 0
    HoppingOffset: 0
           NSRSTx: 0
             Port: 0
```

```
info(2)
```

```
ans = struct with fields:
         UePeriod: 10
         UeOffset: 0
           PRBSet: [4x1 double]
        FreqStart: 60
          KTxComb: 0
         BaseFreq: 60
          FreqIdx: 0
    HoppingOffset: 0
           NSRSTx: 0
             Port: 1
```

### Generate SRS Indices Varying Indexing Style

Generate the SRS indices for two transmit antenna paths. Display the information structure.

Initialize UE-specific and channel configuration structures (`ue` and `chs`) for 3 MHz bandwidth and two antennas. Generate SRS indices and the information structure (`ind` and `info`).

```
ue.DuplexMode = 'FDD';
ue.CyclicPrefixUL = 'Normal';
ue.NFrame = 0;
ue.NULRB = 15;
ue.NSubframe = 0;

chs.NTxAnts = 2;
chs.BWConfig = 7;
chs.BW = 0;
chs.ConfigIdx = 7;
chs.TxComb = 0;
chs.HoppingBW = 0;
chs.FreqPosition = 0;

[ind,info] = lteSRSIndices(ue,chs,{'sub'});
```

Using `'sub'` indexing style, the indices are output in `[subcarrier,symbol,antenna]` subscript form. View the midpoint of `ind` and observe the antenna index change.

```
size(ind)
```

```
ans = 1×2

    48     3
```

```
ind(22:27,:)
```

```
ans = 6x3 uint32 matrix

   103    14     1
   105    14     1
   107    14     1
    61    14     2
    63    14     2
    65    14     2
```

Since there are two antennas, the `info` output structure contains two elements. View the contents of the second `info` structure element.

```
size(info)
```

```
ans = 1×2

     1     2
```

```
info(2)
```

```
ans = struct with fields:
         UePeriod: 10
         UeOffset: 0
           PRBSet: [4x1 double]
        FreqStart: 60
          KTxComb: 0
         BaseFreq: 60
          FreqIdx: 0
    HoppingOffset: 0
           NSRSTx: 0
             Port: 1
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific settings, specified as a structure containing the following fields.

#### NULRB — Number of uplink resource blocks

positive integer

Number of uplink resource blocks, specified as a positive integer.

Data Types: `double`

**`NSubframe` — Number of subframes**
0 (default) | optional | nonnegative integer

Number of subframes, specified as a nonnegative integer.

Data Types: `double`

**`NTxAnts` — Number of transmission antennas**
1 (default) | optional | 2 | 4

Number of transmission antennas, specified as a 1, 2, or 4.

Data Types: `double`

**`CyclicPrefixUL` — Cyclic prefix length**
`'Normal'` (default) | optional | `'Extended'`

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

**`NFrame` — Initial frame number**
0 (default) | optional | nonnegative integer

Initial frame number, returned as a nonnegative integer.

Data Types: `double`

**`DuplexMode` — Duplexing mode**
`'FDD'` (default) | optional | `'TDD'`

Duplexing mode, specified as `'FDD'` or `'FDD'` to indicate the frame structure of the generated waveform.

Data Types: `char` | `string`

**`TDDConfig` — Uplink or downlink configuration**
0 (default) | optional | 0,...,6

Uplink or downlink configuration, returned as a nonnegative integer from 0 to 6. Only required for `'TDD'` duplex mode.

Data Types: `double`

**`SSC` — Special subframe configuration**
0 (default) | optional | 0,...,9

Special subframe configuration, returned as a nonnegative integer from 0 to 9. Only required for `'TDD'` duplex mode.

Data Types: `double`

**`CyclicPrefix` — Cyclic prefix length**
`'Normal'` (default) | optional | `'Extended'`

Cyclic prefix length, returned as `'Normal'` or `'Extended'`. Only required for `'TDD'` duplex mode.

Data Types: `char` | `string`

Data Types: `struct`

**`chs` — Signal transmission configuration**
structure

Signal transmission configuration, specified as a structure containing these fields.

**`NTxAnts` — Number of transmission antennas**
1 (default) | optional | 2 | 4

Number of transmission antennas, specified as a 1, 2, or 4.

Data Types: `double`

**`BWConfig` — Cell-specific SRS bandwidth configuration**
7 (default) | optional | 0,...,7

Cell-specific SRS bandwidth configuration, specified as a nonnegative integer from 0 to 7. ($C_{\mathrm{SRS}}$)

Data Types: `double`

**`BW` — UE-specific SRS bandwidth**
0 (default) | optional | 1 | 2 | 3

UE-specific SRS bandwidth, specified as a nonnegative integer from 0 to 3. ($B_{\mathrm{SRS}}$)

Data Types: `double`

**`ConfigIdx` — Configuration index for UE-specific periodicity**
7 (default) | optional | 0,...,644

Configuration index for UE-specific periodicity, specified as a nonnegative integer from 0 to 644. This parameter contains the configuration index for UE-specific periodicity ($T_{\mathrm{SRS}}$) and subframe offset ($T_{\mathrm{offset}}$).

Data Types: `double`

**`TxComb` — Transmission comb**
0 (default) | optional | 1

Transmission comb, specified as a 0 or 1. This parameter controls SRS positions. SRS is transmitted in six carriers per resource block on odd (1) and even (0) resource indices.

Data Types: `double` | `logical`

**`HoppingBW` — SRS frequency hopping configuration index**
0 (default) | optional | 1 | 2 | 3

SRS frequency hopping configuration index, specified as a nonnegative integer from 0 to 3. ($b_{\mathrm{hop}}$)

Data Types: `double`

**`FreqPosition` — Frequency-domain position**
0 (default) | optional | 0,...,23

Frequency-domain position, specified as a nonnegative integer from 0 to 23. ($n_{\mathrm{RRC}}$)

Data Types: `double`

**`CyclicShift` — UE-specific cyclic shift**
0 (default) | optional | 0,...,7

UE-specific cyclic shift, specified as a nonnegative integer from 0 to 7. This parameter applies only when `NTxAnts` is 4. ($n_{SRS}^{cs}$)

Data Types: `double`

**`NF4RachPreambles` — Number of RACH preamble frequency resources of format 4 in UpPTS**
0 (default) | optional | 0,...,6

Number of RACH preamble frequency resources of format 4 in "UpPTS" on page 2-1159, specified as a nonnegative integer from 0 to 6. Only required for `'TDD'` duplex mode.

Data Types: `double`

**`OffsetIdx` — SRS subframe offset**
0 (default) | optional | 1

SRS subframe offset choice for 2 ms SRS periodicity, specified as 0 or 1. Only required for `'TDD'` duplex mode. This parameter indexes the two SRS subframe offset entries in the row of TS 36.213 [1], Table 8.2-2 for the SRS configuration index specified by the `ConfigIdx` parameter.

Data Types: `double`

**`MaxUpPts` — Option to disable reconfiguration of sounding maximum bandwidth**
1 (default) | optional | 0

Option to disable reconfiguration of sounding maximum bandwidth, specified as 0 or 1. Only required for `'TDD'` duplex mode. Enables (1) or disables (0) reconfiguration of $m_{SRS,0}^{max}$ in "UpPTS" on page 2-1159. See TS 36.331 [2] for information on how the system information element *srs-MaxUpPts* applies to$m_{SRS,0}^{max}$ configurability.

Data Types: `double` | `logical`

Data Types: `struct`

**`opts` — Output format options for resource element indices**
character vector | cell array of character vectors | string array

Output format options for resource element indices, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string) :

| Category | Options | Description |
|---|---|---|
| Indexing style | `'ind'` (default) | The returned indices are in linear index style. |

| Category | Options | Description |
|---|---|---|
| | `'sub'` | The returned indices are in `[subcarrier,symbol,port]` subscript row style. |
| Index base | `'1based'` (default) | The returned indices are one-based. |
| | `'0based'` | The returned indices are zero-based. |

Example: `'ind 1based'`, `"ind 1based"`, `{'ind','1based'}`, or `["ind","1based"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `ind` — Antenna indices

numeric matrix

Antenna indices, returned as a numeric matrix. By default, the indices are returned in one-based linear indexing form that can directly index elements of a resource matrix. These indices are ordered according to SRS modulation symbols mapping. The `opts` input offers alternative indexing formats. The indices for each antenna are in the columns of `ind`, with the number of columns determined by the number of transmission antennas configured specified in `chs.NTxAnts`. For more information, see “SRS Processing” on page 2-1158.

Data Types: `uint32`

### `info` — Information related to SRS

structure array

Information related to SRS, returned as a structure array with elements corresponding to each transmit antenna and containing these fields.

#### `UePeriod` — UE-specific SRS periodicity

`2` | `5` | `10` | `20` | `40` | `80` | `160` | `320`

UE-specific SRS periodicity, in ms, returned as a positive integer.

Data Types: `double`

#### `UeOffset` — UE-specific SRS offset

0,...,319 | integer

UE-specific SRS offset, returned as an integer from 0 to 319.

Data Types: `double`

#### `PRBSet` — Physical resource block set

vector of integers

Physical resource block set, returned as a vector of integers. `PRBSet` specifies the PRBs occupied by the indices (zero-based).

Data Types: `double`

**`FreqStart` — Frequency-domain starting position**
numeric scalar

Frequency-domain starting position ($k_0$), returned as a numeric scalar. This argument is the zero-based subcarrier index of the lowest SRS subcarrier.

Data Types: `double`

**`KTxComb` — Offset to the frequency-domain starting position**
numeric scalar

Offset to the frequency-domain starting position ($k_{\mathrm{TC}}$), returned as a numeric scalar. This argument is a function of the transmission comb parameter.

Data Types: `double`

**`BaseFreq` — Base frequency-domain starting position**
numeric scalar

Base (cell-specific) frequency-domain starting position ($\bar{k}_0$), returned as a numeric scalar. This UE-specific SRS is offset as a function of the UE-specific SRS bandwidth value, $B_{\mathrm{SRS}}$. UE-specific SRS configuration cannot result in a frequency-domain starting position ($k_0$) lower than this value, given the cell-specific SRS bandwidth configuration value, $C_{\mathrm{SRS}}$.

Data Types: `double`

**`FreqIdx` — Frequency position index**
numeric vector

Frequency position index, returned as a numeric vector. This argument specifies the frequency position index ($n_b$) for each $b$ in the range $0,\ldots,B_{\mathrm{SRS}}$.

Data Types: `double`

**`HoppingOffset` — Offset term due to frequency hopping**
numeric vector

Offset term due to frequency hopping, returned as a numeric vector. This argument specifies the offset term due to frequency hopping ($F_b$), used in the calculation of $n_b$.

Data Types: `double`

**`NSRSTx` — Number of UE-specific SRS transmissions**
positive integer

Number of UE-specific SRS transmissions ($n_{\mathrm{SRS}}$), returned as a positive integer.

Data Types: `double`

**`Port` — Antenna port number used for transmission**
positive integer

Antenna port number used for transmission ($p$), returned as a positive integer.

Data Types: `double`

Data Types: `struct`

## More About

### SRS Processing

As specified in TS 36.213, Section 8.2, a UE shall transmit the sounding reference symbol (SRS) on per serving cell SRS resources, based on two trigger types:

- trigger type 0 — periodic SRS from higher layer signalling
- trigger type 1 — aperiodic SRS from DCI formats 0/4/1A for FDD or TDD and from DCI formats 2B/2C/2D for TDD.

The parameter `chs.ConfigIdx` indexes Tables 8.2-1, 8.2-2, 8.2-4, and 8.2-5 defined in TS 36.213, Section 8.2. The applicable table and appropriate range of `chs.ConfigIdx` depends on the duplex mode and the SRS trigger type.

If type 0 triggered SRS transmission is intended, then:

- The valid range of `chs.ConfigIdx` ($I_{SRS}$) is from 0 to 636 for FDD (Table 8.2-1) and from 0 to 644 for TDD (Table 8.2-2).

If type 1 triggered SRS transmission is intended, then:

- `chs.ConfigIdx` indexes trigger type 1 UE-specific periodicity $T_{\mathrm{SRS},1}$ and subframe offset $T_{\mathrm{offset},1}$. The valid range of `chs.ConfigIdx` ($I_{SRS}$) is from 0 to 16 for FDD (Table 8.2-4) and from 0 to 24 for TDD (Table 8.2-5).
- Frequency hopping is not permitted. Therefore, set `chs.HoppingBW` to be greater than or equal to `BW`. ($b_{\mathrm{hop}} \geq B_{\mathrm{SRS}}$).

To control whether to call the `lteSRS` and `lteSRSIndices` functions in a subframe, use `info.IsSRSSubframe`, returned by `lteSRSInfo`.

UE-specific configurations determine how `lteSRS` and `lteSRSIndices` operate. When no SRS is scheduled, calling `lteSRS` or `lteSRSIndices` in a subframe:

- May generate an SRS depending on the cell-specific SRS subframe configuration.
- Returns an empty `seq` or `ind` vector, for a given UE-specific SRS configuration. Also, the `info` structure scalar fields are set to –1, and any undefined vector fields are empty.

For short-base reference sequences, used with SRS transmissions spanning 4 PRBs, the `lteSRS` function does not use Zadoff Chu sequences and it sets `info.RootSeq` and `info.NZC` to –1.

`lteSRSIndices` returns the UE-specific SRS periodicity, `info.UePeriod`, and subframe offset, `info.UeOffset`. These parameters are distinct from the cell-specific SRS periodicity and subframe offset that `lteSRSInfo` returns.

If `chs.NTxAnts` is not present, `ue.NTxAnts` is used. If neither is present, the function assumes one antenna. In `lteSRSIndices`, for SRS transmission on multiple antennas:

- When `chs.NTxAnts` is set to 2 or 4, the value of `info.Port` matches the position in the structure array (0,...,`NTxAnts` – 1).
- If `chs.NTxAnts` is set to 1, `lteSRSIndices` uses `info.Port` to indicate the port chosen by SRS transmit antenna selection. `info.Port` indicates the selected antenna port, 0 or 1.

**UpPTS**

Uplink pilot time slot — the uplink part of the special subframe. This special subframe is only applicable for TDD operation. For more information, see “Frame Structure Type 2: TDD”.

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.213. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.331. “Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Resource Control (RRC); Protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteSRS` | `lteSRSInfo` | `lteCellRSIndices` | `lteCSIRSIndices` | `lteDMRSIndices` | `ltePRSIndices`

# lteSRSInfo

Uplink SRS information

## Syntax

```
info = lteSRSInfo(ue,chs)
```

## Description

`info = lteSRSInfo(ue,chs)` returns information related to the sounding reference signal (SRS) configuration determined by UE-specific settings, `ue`, and signal transmission configuration, `chs`. The information returned relates to the cell-specific SRS subframe configuration as described in TS 36.211[1], Section 5.5.3.3.

Information relating to a particular UE, such as UE-specific SRS configuration defined in TS 36.213[2], Section 8.2, is output by the `lteSRSIndices` and `lteSRS` functions. For a given configuration, if either of these components returns an empty vector, the SRS is not transmitted for that UE in the specified subframe.

**Note** `lteSRSIndices` and `lteSRS` may generate an SRS signal and indices even in a subframe that, based on the cell-specific SRS subframe configuration, is not an SRS subframe. Use the field `info.IsSRSSubframe` returned by `lteSRSInfo` to control whether to call the `lteSRSIndices` and `lteSRS` functions in a subframe.

## Examples

### Get Information Related to SRS

Adjust the length of the PUCCH to allow for SRS transmission using the shortened field.

The setup in this example is consistent with `Simultaneous-ACK/NACK-and-SRS` set to `'True'` as described in TS 36.213, Section 8.2. Generate `pucchSymbols`, using `ue.Shortened=1`.

```
ue = lteRMCUL('A1-1');
srs.SubframeConfig = 0;
srsInfo = lteSRSInfo(ue,srs);
ue.Shortened = srsInfo.IsSRSSubframe;
harqValues = [];

pucchSymbols = ltePUCCH1(ue,ue.PUSCH,harqValues);
```

For the default case, `Simultaneous-ACK/NACK-and-SRS` is `'False'`, and the PUCCH is transmitted with `ue.Shortened=0`.

## Input Arguments

### `ue` — UE-specific settings

structure

UE-specific settings, specified as a structure. `ue` contains the following fields.

### `NSubframe` — Subframe number

numeric scalar | optional

Subframe number, specified as a numeric scalar.

Data Types: `double`

### `DuplexMode` — Duplex mode

`'FDD'` (default) | `'TDD'` | optional

Duplex mode, specified as `'FDD'` or `'TDD'`.

Data Types: `char` | `string`

Data Types: `struct`

### `chs` — Signal transmission configuration

structure

Signal transmission configuration, specified as a structure. `chs` contains the following fields.

### `SubframeConfig` — SRS subframe configuration

0...15

SRS subframe configuration, specified as a nonnegative scalar integer from 0 through 15.

Data Types: `double`

Data Types: `struct`

## Output Arguments

### `info` — Information related to the SRS configuration

structure

Information related to the SRS configuration, returned as a structure. `info` contains the following fields.

### `CellPeriod` — Cell-specific SRS periodicity

1 | 2 | 5 | 10

Cell-specific SRS periodicity, in ms, returned as 1, 2, 5, or 10.

Data Types: `uint32`

### `CellOffset` — Cell-specific SRS offsets

0...9

Cell-specific SRS offsets, returned as a nonnegative scalar integer from 0 through 9.

Data Types: `int32`

**`IsSRSSubframe` — SRS subframe flag**
1 | 0

SRS subframe flag, returned as 1 or 0. Present only if `ue` contains `NSubframe`. If `NSubframe` satisfies the expression `mod(NSubframe,CellPeriod)==CellOffset`, this value is 1. Otherwise, this value is 0.

Data Types: `uint32`

Data Types: `struct`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. "Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.213. "Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteSRS` | `lteSRSIndices`

# lteSSS

Secondary synchronization signal

## Syntax

```
sss = lteSSS(enb)
```

## Description

`sss = lteSSS(enb)` returns a complex column vector containing the secondary synchronization signal (SSS) values for cell-wide settings in structure `enb`.

This signal is only defined for subframes 0 and 5; therefore, an empty vector is returned for other values of `NSubframe`. This allows this function and the corresponding indices function `lteSSSIndices` to be used to index the resource grid, as described in “Resource Grid Indexing”, for any subframe number, but the resource grid is only modified in subframes 0 and 5.

## Examples

### Generate SSS Values

Generate secondary synchronization signal (SSS) values for a physical layer cell identity of 1.

```
sss = lteSSS(struct('NCellID',1,'NSubframe',0));
sss(1:4)
```

```
ans = 4×1

     1
    -1
     1
     1
```

## Input Arguments

### `enb` — Cell-wide settings

structure

Cell-wide settings, specified as a structure. This structure can contain the following fields.

### `NCellID` — Physical layer cell identity

integer

Physical layer cell identity, specified as an integer.

Data Types: `double`

**`NSubframe` — Subframe number**
0 (default) | optional | integer

Subframe number, specified as an integer.

Data Types: `double`

Data Types: `struct`

## Output Arguments

**`sss` — Secondary synchronization signal**
complex column vector

Secondary synchronization signal (SSS), returned as a complex column vector. The vector contains the SSS values for cell-wide settings in the `enb` structure.

Data Types: `double`

# Version History

**Introduced in R2014a**

## See Also

`lteSSSIndices` | `ltePSS` | `lteSSSS`

# lteSSSIndices

SSS resource element indices

## Syntax

```
ind = lteSSSIndices(enb)
ind = lteSSSIndices(enb,port)
ind = lteSSSIndices(enb,port,opts)
```

## Description

`ind = lteSSSIndices(enb)` returns a column vector of resource element indices, port 0 oriented, given the parameter fields of structure, `enb`. It returns a column vector of resource element (RE) indices for the Secondary Synchronization Signal (SSS). By default, the indices are returned in 1-based linear indexing form that can directly index elements of a 3-D array representing the resource array. These indices are ordered as the SSS modulation symbols should be mapped. Alternative indexing formats can also be generated.

These indices are only defined for subframes 0 and 5; therefore, an empty vector is returned for other values of `NSubframe`. This allows this function and the corresponding sequence function, `lteSSS`, to be used to index the resource grid, as described in “Resource Grid Indexing”, for any subframe number. However, the resource grid is only modified in subframes 0 and 5.

`ind = lteSSSIndices(enb,port)` returns indices appropriate for an antenna port, `port`, which must be either 0, 1, 2, or 3.

`ind = lteSSSIndices(enb,port,opts)` formats the returned indices using options specified by `opts`.

## Examples

### Generate SSS RE Indices in Linear Form

Get 0-based SSS resource element indices in linear form for antenna port 0.

```
enb = lteRMCDL('R.4');
sssIndices = lteSSSIndices(enb,0,{'0based','ind'});
sssIndices(1:4)
```

```
ans = 4x1 uint32 column vector

   365
   366
   367
   368
```

### Generate SSS RE Indices in Subscript Form

Generate 0-based SSS resource element indices in subscript form for antenna port 0. In this case, a matrix is generated where each row has three columns representing subcarrier, symbol, and antenna port.

Generate 0-based SSS resource element indices in subscript form for antenna port 0.

```
enb = lteRMCDL('R.4');
sssIndices = lteSSSIndices(enb,0,{'0based','sub'});
sssIndices(1:4,:)
```

```
ans = 4x3 uint32 matrix

   5   5   0
   6   5   0
   7   5   0
   8   5   0
```

The first column of the output represents subcarrier. The second column represents symbol. The third column represents antenna port.

## Input Arguments

**enb — Cell-wide settings**
structure

Cell-wide settings, specified as a structure. It contains the following fields.

**NDLRB — Number of downlink resource blocks**
positive integer

Number of downlink resource blocks, specified as a positive integer in the interval [6, 110].

Data Types: `double`

**CyclicPrefix — Cyclic prefix length**
`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as`'Normal'` or `'Extended'`.

Data Types: `char` | `string`

**NSubframe — Subframe number**
0 (default) | nonnegative integer | optional

Subframe number, specified as a nonnegative integer.

Data Types: `double`

**DuplexMode — Duplex mode**
`'FDD'` (default) | `'TDD'` | optional

Duplex mode, specified as `'FDD'` or `'TDD'`.

Data Types: `char` | `string`

Data Types: `struct`

**`port` — Antenna port number**
nonnegative integer

Antenna port number, specified as a nonnegative integer.

Example: 2

Data Types: `double`

**`opts` — Output format options for resource element indices**
character vector | cell array of character vectors | string array

Output format options for resource element indices, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string) :

| Category | Options | Description |
|---|---|---|
| Indexing style | `'ind'` (default) | The returned indices are in linear index style. |
| | `'sub'` | The returned indices are in `[subcarrier,symbol,port]` subscript row style. |
| Index base | `'1based'` (default) | The returned indices are one-based. |
| | `'0based'` | The returned indices are zero-based. |

Example: `'ind 1based'`, `"ind 1based"`, `{'ind','1based'}`, or `["ind","1based"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

**`ind` — SSS resource element (RE) indices**
integer column vector | 3-column integer matrix

SSS resource element (RE) indices, returned as a column vector or 3-column integer matrix. By default, the indices are returned in 1-based linear indexing form that can directly index elements of a 3D array representing the resource array. These indices are ordered as the SSS modulation symbols should be mapped. Alternative indexing formats can be generated using the `opts` input.

Data Types: `uint32`

# Version History

**Introduced in R2014a**

## See Also

`lteSSS` | `ltePSSIndices` | `lteSSSSIndices`

**Topics**
“Resource Grid Indexing”

# lteSSSS

Secondary sidelink synchronization signal

## Syntax

```
s = lteSSSS(ue)
```

## Description

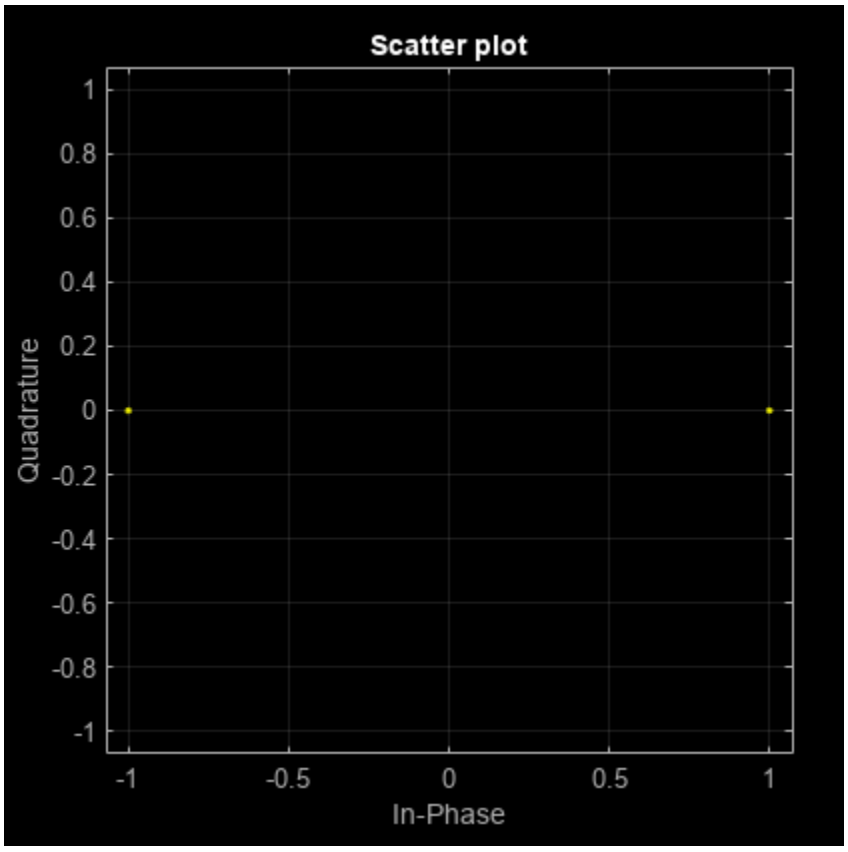
`s = lteSSSS(ue)` returns `s`, a column vector containing the secondary sidelink synchronization signal (SSSS) values for user equipment settings `ue`. For more information, see “Secondary Sidelink Synchronization Signal” on page 2-1171.

## Examples

### Generate SSSS

Generate and plot the BPSK sidelink secondary synchronization signal values for NSLID = 10.

```
ssss = lteSSSS(struct('NSLID',10));
scatterplot(ssss)
grid
```



### Generate All SSSS Sequences

Generate all possible SSSS sequences, contained as columns in a 124-by-336 matrix.

```
ssssfn = @(x)lteSSSS(struct('NSLID',x));
allssss = cell2mat(arrayfun(ssssfn,[0:335],'UniformOutput',false));
```

### Generate All SSSSs for V2X

Generate all possible SSSSs for V2X sidelink.

```
sfn = @(x)lteSSSS(struct('NSLID',x,'SidelinkMode','V2X'));
s = cell2mat(arrayfun(sfn,0:335,'UniformOutput',false));
```

## Input Arguments

### `ue` — User equipment settings

structure

User equipment settings, specified as a structure containing the following fields.

#### `SidelinkMode` — Sidelink mode

`'D2D'` (default) | `'V2X'` | optional

Sidelink mode, specified as `'D2D'` or `'V2X'`.

Data Types: `char` | `string`

#### `NSLID` — Physical layer sidelink synchronization identity

integer in the interval [0, 335]

Physical layer sidelink synchronization identity, specified as an integer in the interval [0, 335].

For more information, see “Secondary Sidelink Synchronization Signal” on page 2-1171.

Data Types: `double`

## Output Arguments

### `s` — SSSS values

complex-valued vector

SSSS values, returned as a 124-by-1 complex-valued vector. These values are created for the user equipment settings in the `ue` structure. For more information, see “Secondary Sidelink Synchronization Signal” on page 2-1171.

## More About

### Secondary Sidelink Synchronization Signal

The secondary sidelink synchronization signal (SSSS) is transmitted in the central 62 resource elements of two adjacent SC-FDMA symbols in a synchronization subframe. The same sequence of 62 complex values is repeated in each of the symbols, resulting in a 124-by-1 element vector returned by the `lteSSSS` function. The values of this sequence are ordered as they should be mapped into the resource elements of the adjacent symbols using `lteSSSSIndices`. If a terminal is transmitting SSSS then it should be sent every 40 ms for D2D sidelink or every 160 ms for V2X sidelink, with the exact subframe dependent on the RCC signaled subframe number offset (syncOffsetIndicator-r12). The SSSS is sent on antenna port 1020, along with the primary sidelink synchronization signal (PSSS). A synchronization subframe also contains the PSBCH, which is sent on antenna port 1010. The transmission power of the SSSS symbols should be the same as the PSBCH therefore they should be scaled by $\sqrt{72/62}$ in a subframe. No PSCCH or PSSCH transmission will occur in a sidelink subframe configured for synchronization purposes.

As specified in TS 36.211, Section 9.7, the SSSS identity assignment depends on the network coverage. The set of all $N_{\mathrm{ID}}^{\mathrm{SL}}$ is divided into two sets, id_net {0, ..., 167} and id_oon {168, ..., 335}, which are used by terminals that are in-network and out-of-network coverage, respectively. The sidelink physical layer cell identity number, $N_{\mathrm{ID}}^{\mathrm{SL}}$, corresponds to the `lteSSSS` input UE settings structure field `ue.NSLID`. Within each set, all identities result in the same SSSS. For an in-network terminal, the `ue.NSLID` value corresponds to the RRC sidelink synchronization signal identity (slssid-r12) associated with the cell.

### Secondary Sidelink Synchronization Signal Indexing

Use the indexing function, `lteSSSSIndices`, and the corresponding sequence function, `lteSSSS`, to populate the resource grid for the desired subframe number. The SSSS values are output by `lteSSSS`, ordered as they should be mapped, applying frequency-first mapping into the resource elements of the adjacent symbols using `lteSSSSIndices`. When indexing is zero-based, the SC-FDMA symbols used are {11,12} for normal cyclic prefix and {9, 10} for extended cyclic prefix.

**Note** The indicated symbol indices are based on TS 36.211, Section 9.7. However to align with the LTE Toolbox subframe orientation, these indices are expanded from symbol index per slot to symbol index per subframe.

For more information on mapping symbols to the resource element grid, see “Resource Grid Indexing”.

# Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteSSSSIndices` | `ltePSSS` | `lteSSS`

**Topics**

“Resource Grid Indexing”

# lteSSSSIndices

SSSS resource element indices

## Syntax

```
ind = lteSSSSIndices(ue)
ind = lteSSSSIndices(ue,opts)
```

## Description

`ind = lteSSSSIndices(ue)` returns a 124-by-1 complex column vector of resource element (RE) indices for the secondary sidelink synchronization signal (SSSS) values given the user equipment settings structure. By default, the indices are returned in one-based linear indexing form. You can use this form to directly index elements of a matrix representing the subframe resource grid for antenna port 1020. For more information, see “Secondary Sidelink Synchronization Signal Indexing” on page 2-1176.

`ind = lteSSSSIndices(ue,opts)` formats the returned indices using options specified by `opts`.

## Examples

### Generate SSSS Indices

Generate SSSS values and indices. Write the values into the SSSS resource elements in a synchronization subframe (extended cyclic prefix) and display an image of their locations.

Create a user equipment settings structure and a resource grid that has a 10 MHz bandwidth and extended cyclic prefix.

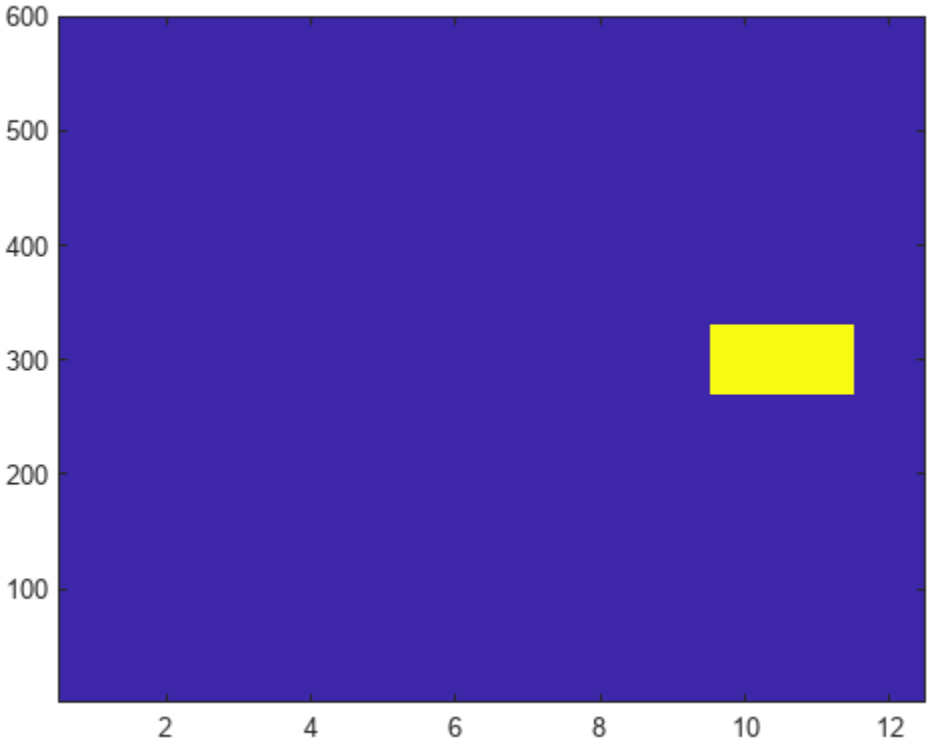
```
ue.NSLRB = 50;
ue.CyclicPrefixSL = 'Extended';
ue.NSLID = 1;
subframe = lteSLResourceGrid(ue);
```

Generate SSSS indices and display the first five indices. Load the SSSS symbols into the resource grid. Display an image showing the SSSS symbol locations.

```
ind = lteSSSSIndices(ue);
ind(1:5)
```

```
ans = 5x1 uint32 column vector

   5670
   5671
   5672
   5673
   5674
```

```
subframe(ind) = lteSSSS(ue);
imagesc(100*abs(subframe))
axis xy
```




### Generate Zero-Based SSSS Indices

Generate SSSS indices using zero-based indexing style. Compare these indices to one-based indices.

Create a user equipment settings structure that has a 10 MHz bandwidth and extended cyclic prefix.

```
ue.NSLRB = 50;
ue.CyclicPrefixSL = 'Extended';
ue.NSLID = 1;
```

Generate SSSS zero-based indices and view the first five indices.

```
ind = lteSSSSIndices(ue,'0based');
s = size(ind)
```

```
s = 1×2

   124     1
```

```
ind(1:5)
```

```
ans = 5x1 uint32 column vector

   5669
   5670
   5671
   5672
   5673
```

Generate SSSS one-based indices and view the first five indices.

```
ind = lteSSSSIndices(ue,'1based');
s = size(ind)
```

```
s = 1×2

   124     1
```

```
ind(1:5)
```

```
ans = 5x1 uint32 column vector

   5670
   5671
   5672
   5673
   5674
```

For zero-based indexing, the first assigned index is one lower than the one-based indexing style.

## Input Arguments

### `ue` — User equipment settings

structure

User equipment settings, specified as a structure. `ue` contains the following fields.

#### `NSLRB` — Number of sidelink resource blocks

integer scalar from 6 to 110

Number of sidelink resource blocks, specified as an integer scalar from 6 to 110.

Example: `6`, which corresponds to a channel bandwidth of 1.4 MHz.

Data Types: `double`

#### `CyclicPrefixSL` — Cyclic prefix length

`'Normal'` (default) | `'Extended'` | optional

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

### `opts` — Output format options for resource element indices

character vector | cell array of character vectors | string array

Output format options for resource element indices, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string) :

| Category | Options | Description |
|---|---|---|
| Indexing style | `'ind'` (default) | The returned indices are in linear index style. |
| | `'sub'` | The returned indices are in `[subcarrier,symbol,port]` subscript row style. |
| Index base | `'1based'` (default) | The returned indices are one-based. |
| | `'0based'` | The returned indices are zero-based. |

Example: `'ind 1based'`, `"ind 1based"`, `{'ind','1based'}`, or `["ind","1based"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

## Output Arguments

### `ind` — SSSS resource element indices

integer column vector | three-column integer matrix

SSSS resource element indices, returned as an integer column vector or a three-column integer matrix. This output is generated using the cell-wide settings structure, `ue`. For more information, see “Secondary Sidelink Synchronization Signal Indexing” on page 2-1176.

Data Types: `uint32`

## More About

### Secondary Sidelink Synchronization Signal Indexing

Use the indexing function, `lteSSSSIndices`, and the corresponding sequence function, `lteSSSS`, to populate the resource grid for the desired subframe number. The SSSS values are output by `lteSSSS`, ordered as they should be mapped, applying frequency-first mapping into the resource elements of the adjacent symbols using `lteSSSSIndices`. When indexing is zero-based, the SC-FDMA symbols used are {11,12} for normal cyclic prefix and {9, 10} for extended cyclic prefix.

**Note** The indicated symbol indices are based on TS 36.211, Section 9.7. However to align with the LTE Toolbox subframe orientation, these indices are expanded from symbol index per slot to symbol index per subframe.

For more information on mapping symbols to the resource element grid, see “Resource Grid Indexing”.

### Secondary Sidelink Synchronization Signal

The secondary sidelink synchronization signal (SSSS) is transmitted in the central 62 resource elements of two adjacent SC-FDMA symbols in a synchronization subframe. The same sequence of 62

complex values is repeated in each of the symbols, resulting in a 124-by-1 element vector returned by the `lteSSSS` function. The values of this sequence are ordered as they should be mapped into the resource elements of the adjacent symbols using `lteSSSSIndices`. If a terminal is transmitting SSSS then it should be sent every 40 ms for D2D sidelink or every 160 ms for V2X sidelink, with the exact subframe dependent on the RCC signaled subframe number offset (syncOffsetIndicator-r12). The SSSS is sent on antenna port 1020, along with the primary sidelink synchronization signal (PSSS). A synchronization subframe also contains the PSBCH, which is sent on antenna port 1010. The transmission power of the SSSS symbols should be the same as the PSBCH therefore they should be scaled by $\sqrt{72/62}$ in a subframe. No PSCCH or PSSCH transmission will occur in a sidelink subframe configured for synchronization purposes.

As specified in TS 36.211, Section 9.7, the SSSS identity assignment depends on the network coverage. The set of all $N_{\mathrm{ID}}^{\mathrm{SL}}$ is divided into two sets, id_net {0, ..., 167} and id_oon {168, ..., 335}, which are used by terminals that are in-network and out-of-network coverage, respectively. The sidelink physical layer cell identity number, $N_{\mathrm{ID}}^{\mathrm{SL}}$, corresponds to the `lteSSSS` input UE settings structure field `ue.NSLID`. Within each set, all identities result in the same SSSS. For an in-network terminal, the `ue.NSLID` value corresponds to the RRC sidelink synchronization signal identity (slssid-r12) associated with the cell.

# Version History

**Introduced in R2016b**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteSSSS` | `ltePSSSIndices` | `lteSSSIndices`

**Topics**

“Resource Grid Indexing”

# lteSymbolDemodulate

Demodulation and symbol to bit conversion

## Syntax

```
out = lteSymbolDemodulate(in,mod)
out = lteSymbolDemodulate(in,mod,dec)
```

## Description

`out = lteSymbolDemodulate(in,mod)` returns a column vector containing bits resulting from soft constellation demodulation of complex values in vector `in`. The demodulation algorithm assumes the vector of received symbols are normalized to fall on constellation points as defined by `in`. `lteSymbolModulate` provides an output with the expected constellation scaling.

`out = lteSymbolDemodulate(in,mod,dec)` allows the decision mode, `dec`, to be specified as either `'Hard'` or `'Soft'`.

## Examples

### Demodulate Complex-Valued Symbols

Demodulate complex-valued symbols, specifying hard decision mode.

```
out = lteSymbolDemodulate([0.7 - 0.7i; -0.7 + 0.7i],'QPSK','Hard')
```

```
out = 4×1

     0
     1
     1
     0
```

## Input Arguments

### `in` — Input symbols to demodulate

numeric column vector

Input symbols to demodulate, specified as a column vector of complex numeric values. Demodulation is performed assuming the input constellation power normalization in accordance with TS 36.211, Section 7.1 [2], as follows:

- `1/sqrt(2)` for `'BPSK'` and `'QPSK'`
- `1/sqrt(10)` for `'16QAM'`
- `1/sqrt(42)` for `'64QAM'`
- `1/sqrt(170)` for `'256QAM'`

- `1/sqrt(682)` for `'1024QAM'`

Example: For `'BPSK'` and `'QPSK'` `[0.707 - 0.707i; -0.707 + 0.707i]`

Data Types: `double`
Complex Number Support: Yes

**`mod` — Modulation format**
`'BPSK'` | `'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'` | `'1024QAM'`

Modulation format, specified as `'BPSK'`, `'QPSK'`, `'16QAM'`, `'64QAM'`, `'256QAM'`, or `'1024QAM'`.

Data Types: `char` | `string`

**`dec` — Decision mode**
`'Hard'` | `'Soft'`

Decision mode, specified as `'Hard'` or `'Soft'`.

Data Types: `char` | `string`

## Output Arguments

**`out` — Demodulated output bits**
numeric column vector

Demodulated output bits, returned as a numeric column vector. This argument contains bits resulting from soft constellation demodulation of complex values vector, `in`.

`'Hard'` decision mode results in the output containing the bit sequences corresponding to the closest constellation points to the input.

`'Soft'` decision mode results in the output indicating the bit values using the sign (`-ve` for 0, `+ve` for 1). For `'Soft'` decision mode, the magnitude of the output gives a piecewise linear approximation to the log likelihood ratio (LLR) of the demodulated bits. The algorithm used for the LLR approximation is described in [1]. The returned LLRs are scaled such that for a input signal lying on the constellation points in the preceding description, the output values lie on the points with these magnitudes:

- `1` for `'BPSK'`
- `1/sqrt(2)` for `'QPSK'`
- `[1 3]/sqrt(10)` for `'16QAM'`
- `[1 3 5 7]/sqrt(42)` for `'64QAM'`
- `[1 3 5 7 9 11 13 15]/sqrt(170)` for `'256QAM'`
- `[1:2:31]/sqrt(682)` for `'1024QAM'`

Data Types: `double`

# Version History

**Introduced in R2014a**

## References

[1] Tosato, F., and Bisaglia, P. “Simplified soft-output demapper for binary interleaved COFDM with application to HIPERLAN/2.” *IEEE International Conference on Communications (ICC) 2002, Vol. 2*. pp. 664-668.

[2] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteSymbolModulate` | `lteLayerDemap` | `lteULDescramble`

# lteSymbolModulate

Symbol modulation

## Syntax

```
out = lteSymbolModulate(in,mod)
```

## Description

`out = lteSymbolModulate(in,mod)` maps the bit values, `in`, to complex modulation symbols with the modulation scheme specified in `mod`.

## Examples

### Generate QPSK Modulated Symbols

Map bit values to QPSK modulated symbols.

```
out = lteSymbolModulate([0; 1; 1; 0], 'QPSK')
```

```
out = 2×1 complex

   0.7071 - 0.7071i
  -0.7071 + 0.7071i
```

## Input Arguments

### `in` — Input bits

column vector | cell array

Input bits, specified as a column vector, where each bit is either `0` or `1`. The vector length must be a multiple of two for QPSK, four for 16-QAM, six for 64-QAM, eight for 256-QAM, or ten for 1024-QAM. The bit values must be 0 or 1.

Alternatively, specify `in` as a cell array containing one bit vector or a cell array containing two bit vectors.

Data Types: `double` | `cell`

### `mod` — Modulation scheme

`'BPSK'` | `'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'` | `'1024QAM'` | cell array

Modulation scheme, specified as `'BPSK'`, `'QPSK'`, `'16QAM'`, `'64QAM'`, `'256QAM'`, or `'1024QAM'`.

Alternately, you can specify `mod` as a cell array of one or two modulation schemes. The number of modulation schemes in `mod` cannot exceed the number of bit vectors specified by `in`. If `in` specifies two bit vectors and `mod` specifies one modulation scheme, the same modulation is used for both bit vectors.

Data Types: `char` | `cell`

## Output Arguments

### `out` — Complex modulated output symbols

column vector

Complex modulated output symbols, returned as a column vector. The symbols use the modulation scheme specified in `mod`.

Data Types: `double`
Complex Number Support: Yes

# Version History

**Introduced in R2014a**

## See Also

`lteSymbolDemodulate` | `lteLayerMap` | `lteDLPrecode` | `lteULScramble`

# lteTBS

Transport block size lookup

## Syntax

```
tbs = lteTBS(nprb)
tbs = lteTBS(nprb,itbs)
tbs = lteTBS(nprb,itbs,smnlayer)
```

## Description

Use this function to look up transport block sizes as defined in TS 36.213 [1], Section 7.1.7.2 tables. The tables are Release 15 compliant and contain 37 TBS values for each of the physical resource block entries. The first 27 table entries are the Release 8–Release 11 compatible TBSs. Entries 28-33 are Release 12 compatible.

`tbs = lteTBS(nprb)` returns the column of TS 36.213 [1], Table 7.1.7.2.1-1 for the number of physical resource blocks, `nprb`, specified. Table 7.1.7.2.1-1 is for transport blocks not mapped to two or more spatial multiplexing layers. The returned column vector, `tbs`, has 38 elements, corresponding to transport block size indices from 0 to 37.

`tbs = lteTBS(nprb,itbs)` uses an additional input, `itbs` (a vector of transport block size indices from 0 to 33) to restrict returned vector of values. A value in the `itbs` vector equal to –1, indicates a discontinuous transmission (DTX) and `lteTBS` produces a corresponding `tbs` value of 0.

`tbs = lteTBS(nprb,itbs,smnlayer)` uses an additional input, `smnlayer` to indicate the number of spatial multiplexing layers to which the transport block is mapped. This combines TS 36.213 [1], Table 7.1.7.2.1-1 with the appropriate spatial layer TBS translation table:

- For 2-layer spatial multiplexing, the function follows the rules in TS 36.213 [1], Section 7.1.7.2.2.
- For 3-layer spatial multiplexing, the function follows the rules in TS 36.213 [1], Section 7.1.7.2.4.
- For 4-layer spatial multiplexing, the function follows the rules in TS 36.213 [1], Section 7.1.7.2.5.

For transmission schemes that do not support spatial multiplexing (`'Port0','TxDiversity','Port5','Port8'`), set `smnlayer = 1`.

## Examples

### Generate Transport Block Sizes for Release 15

Generate the set of 38 transport block sizes from TS 36.213, Table 7.1.7.2.1-1 (Release 15), valid for a single PRB allocation.

```
tbs = lteTBS(1);
disp(tbs')
```

```
     16     24     32     40     56     72    328    104    120    136    144    176    208    22
```

### Generate Transport Block Sizes for Three Spatial Layers

Generate the set of 27 Release 8-Release 11 transport block sizes for a single PRB allocation and three spatial layers.

```
nprb = 1;
itbs = 0:26;
smnlayer = 3;
tbs = lteTBS(nprb,itbs,smnlayer);
disp(tbs')
```

```
     56     88    144    176    208    224    256    328    392    456    504    584    680    74
```

## Input Arguments

### `nprb` — Number of physical resource blocks

1,...,110

Number of physical resource blocks, specified as a positive scalar integer from 1 to 110.

Data Types: `double`

### `itbs` — Transport block size indices

numeric vector

Transport block size indices, specified as a numeric vector.

Data Types: `double`

### `smnlayer` — Number of spatial multiplexing layers to which transport block is mapped

1,...,4

Number of spatial multiplexing layers to which transport block is mapped, specified as a positive scalar integer from 1 to 4.

Data Types: `double`

## Output Arguments

### `tbs` — Transport block size or sizes

column vector | nonnegative integer

Transport block size or sizes, returned as a column vector of nonnegative integers from the transport block size table in TS 36.213, Section 7.1.7.2 [1]. The maximum length of this output is 38, corresponding to TBS indices from 0 to 37.

Data Types: `int32`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.213. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteDLSCH` | `ltePDSCH`

# lteTestModel

Downlink test model configuration structure

## Syntax

```
tm = lteTestModel(tmn,bw)
tm = lteTestModel(tmn,bw,ncellid,duplexmode)
tm = lteTestModel(tmcfg)
```

## Description

`tm = lteTestModel(tmn,bw)` returns the E-UTRA test model (E-TM) configuration structure for given test model number and bandwidth. The output structure, `tm`, contains the configuration parameters required to generate a given downlink E-TM waveform using the generator tool, `lteTestModelTool`. The field names and default values align with those defined in TS 36.141 [1], Section 6.1.

The `PDSCH` is a substructure relating to the physical channel configuration and contains the fields `NLayers`, `TxScheme`, and `Modulation`.

`tm = lteTestModel(tmn,bw,ncellid,duplexmode)` controls the configuration of the physical cell identity, `ncellid`, and duplex mode, `duplexmode`.

`tm = lteTestModel(tmcfg)` returns `tm`, a configuration structure for the test model partially (or wholly) defined by the input structure `tmcfg`. The input structure `tmcfg` can define any (or all) of the parameters or substructure parameters and the output structure `tm` retains the defined parameters. The undefined fields are given appropriate default values.

The `tm` structure can be used with the E-TM generator tool to generate a waveform.

## Examples

### Create Downlink E-TM Configuration Structure

Create the configuration structure for Test Model TS 36.141 E-TM 3.2, 20MHz.

Specify an E-TM 3.2 test model number and 20-MHz channel bandwidth. Create a test model configuration structure and view the contents.

```
tmn = '3.2';
bw  = '20MHz';

tm = lteTestModel(tmn, bw)
```

```
tm = struct with fields:
                    TMN: '3.2'
                     BW: '20MHz'
                 NDLRB: 100
              CellRefP: 1
               NCellID: 1
```

```
           CyclicPrefix: 'Normal'
                    CFI: 1
                     Ng: 'Sixth'
          PHICHDuration: 'Normal'
              NSubframe: 0
           TotSubframes: 10
              Windowing: 0
             DuplexMode: 'FDD'
                  PDSCH: [1x1 struct]
            CellRSPower: 0
               PSSPower: 2.4260
               SSSPower: 2.4260
              PBCHPower: 2.4260
            PCFICHPower: 0
     NAllocatedPDCCHREG: 180
             PDCCHPower: 1.1950
      PDSCHPowerBoosted: 2.4260
    PDSCHPowerDeboosted: -3
                   Nfft: []
```

```
tm.PDSCH
```

```
ans = struct with fields:
      TxScheme: 'Port0'
    Modulation: {'16QAM'  'QPSK'}
       NLayers: 1
```

## Input Arguments

### `tmn` — Test model number

`'1.1'` | `'1.2'` | `'2'` | `'2a'` | `'2b'` | `'3.1'` | `'3.1a'` | `'3.1b'` | `'3.2'` | `'3.3'`

Test model number, specified as a character vector or string scalar. Use double quotes for string. See TS 36.141 [1] for information on the test models.

Data Types: `char` | `string`

### `bw` — Channel bandwidth

`'1.4MHz'` | `'3MHz'` | `'5MHz'` | `'10MHz'` | `'15MHz'` | `'20MHz'` | `'9RB'` | `'11RB'` | `'27RB'` | `'45RB'` | `'64RB'` | `'91RB'`

Channel bandwidth, specified as a character vector or string scalar. Use double quotes for string. You can set the nonstandard bandwidths, `'9RB'`, `'11RB'`, `'27RB'`, `'45RB'`, `'64RB'`, and `'91RB'`, only when `tmn` is `'1.1'`. These nonstandard bandwidths specify custom test models.

Data Types: `char` | `string`

### `ncellid` — Physical layer cell identity

1 for standard bandwidths and 10 for non-standard bandwidths (default) | optional | integer from 0 to 503

Physical layer cell identity, specified as an integer from 0 to 503. If not specified, defaults to 1 for standard bandwidths and 10 for non-standard bandwidths.

Data Types: `double`

**duplexmode — Duplex mode**
'FDD' (default) | 'TDD' | optional

Duplex mode, specified as 'FDD' or 'TDD'.

Data Types: char | string

**tmcfg — Test model configuration**
scalar structure

Test model configuration, specified as a scalar structure. Refer to the output tm for structure fields. Define any or all listed parameters or substructure parameters in the input structure, tmcfg. The output structure, tm, retains the defined parameters and appropriate defaults are assigned for undefined fields.

Data Types: struct

## Output Arguments

**tm — E-UTRA test model (E-TM) configuration**
scalar structure

E-UTRA test model (E-TM) configuration, returned as a scalar structure. tm contains the following fields.

| Parameter Field | Values | Description |
|---|---|---|
| **TMN** | '1.1', '1.2', '2', '2a', '2b', '3.1', '3.1a', '3.1b' '3.2', '3.3' | Test model number |
| **BW** | '1.4MHz', '3MHz', '5MHz', '10MHz', '15MHz', '20MHz', '9RB', '11RB', '27RB', '45RB', '64RB', '91RB', | Channel bandwidth, in MHz, returned as a character vector. Nonstandard bandwidths '9RB', '11RB', '27RB', '45RB', '64RB', and '91RB' specify custom test models. |
| **NDLRB** | Nonnegative integer | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **CellRefP** | 1 | Number of cell-specific reference signal antenna ports. This argument is for informational purposes and is read-only. |
| **NCellID** | Integer from 0 to 503 | Physical layer cell identity |
| **CyclicPrefix** | 'Normal' | Cyclic prefix length. This argument is for informational purposes and is read-only. |
| **CFI** | 1, 2, or 3 | Control format indicator value |
| **Ng** | 'Sixth', 'Half', 'One', 'Two' | HICH group multiplier |
| **PHICHDuration** | 'Normal', 'Extended' | PHICH duration |
| **NSubframe** | 0 (default), nonnegative scalar integer | Subframe number. This argument is for informational purposes and is read-only. |

| Parameter Field | Values | Description |
|---|---|---|
| **TotSubframes** | Nonnegative scalar integer | Total number of subframes to generate |
| **Windowing** | Nonnegative scalar integer | Number of time-domain samples over which windowing and overlapping of OFDM symbols is applied |
| **Nfft** | Positive integer | Number of IFFT points used in the OFDM modulation |
| **DuplexMode** | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| **CellRSPower** | Numeric value | Cell-specific reference symbol power adjustment, in dB |
| **PDSCH** | Scalar structure | PDSCH transmission configuration substructure |
| **PSSPower** | Numeric value | Primary synchronization signal (PSS) symbol power adjustment, in dB |
| **SSSPower** | Numeric value | Secondary synchronization signal (SSS) symbol power adjustment, in dB |
| **PBCHPower** | Numeric value | PBCH symbol power adjustment, in dB |
| **PCFICHPower** | Numeric value | PCFICH symbol power adjustment, in dB |
| **NAllocatedPDCCHREG** | Nonnegative integer | Number of allocated PDCCH REGs. This argument is derived from `tmn` and `bw`. |
| **PDCCHPower** | Numeric value | PDCCH symbol power adjustment, in dB |
| **PDSCHPowerBoosted** | Numeric value | PDSCH symbol power adjustment, in dB, for the boosted physical resource blocks (PRBs) |
| **PDSCHPowerDeboosted** | Numeric value | PDSCH symbol power adjustment, in dB, for the de-boosted physical resource blocks (PRBs) |
| These fields are present only when `DuplexMode` is set to `'TDD'`. | | |
| **SSC** | Integer from 0 to 9<br><br>8 (default) | Special subframe configuration (SSC)<br><br>`SSC` enumerates the special subframe configuration. TS 36.211 [2], Section 4.2 specifies the special subframe configurations (lengths of DwPTS, GP, and UpPTS). |
| **TDDConfig** | Integer from 1 to 6<br><br>3 (default) | Uplink-downlink configuration<br><br>`TDDConfig` enumerates the subframe uplink-downlink configuration to be used in this frame. TS 36.211 [2], Section 4.2 specifies uplink-downlink configurations (uplink, downlink, and special subframe combinations). |

**PDSCH Substructure**

The substructure PDSCH relates to the physical channel configuration and contains these fields:

| Parameter Field | Values | Description |
|---|---|---|
| **`NLayers`** | `1` | Number of transmission layers, returned as `1`. This argument is for informational purposes and is read-only. |
| **`TxScheme`** | `'Port0'` | Transmission scheme. The E-TMs have a single antenna port. This argument is for informational purposes and is read-only. |
| **`Modulation`** | Cell array of one or two character vectors. Valid values of character vectors include: `'QPSK'`, `'16QAM'`, `'64QAM'`, `'256QAM'`, `'1024QAM'` | Modulation formats, specifying the modulation formats for boosted and deboosted PRBs. This argument is for informational purposes and is read-only. |

Data Types: `struct`

# Version History

**Introduced in R2014a**

**R2022b: IFFT size output**

The output structure `tm` includes the `Nfft` field, which contains the number of IFFT points used in the OFDM modulation.

## References

[1] 3GPP TS 36.141. “Evolved Universal Terrestrial Radio Access (E-UTRA); Base Station (BS) Conformance Testing.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteTestModelTool` | `lteRMCDL` | `lteRMCUL`

# lteTestModelTool

Generate downlink test model waveform

## Syntax

```
lteTestModelTool
[waveform,grid,tm] = lteTestModelTool(tmn,bw,ncellid,duplexmode)
[waveform,grid,tm] = lteTestModelTool(tm)
```

## Description

`lteTestModelTool` starts the **LTE Waveform Generator** app for the parameterization and generation of the E-UTRA test model (E-TM) waveforms.

`[waveform,grid,tm] = lteTestModelTool(tmn,bw,ncellid,duplexmode)` accepts inputs for the test model number and channel bandwidth for the generated waveform. Optionally, accepts inputs for the physical cell identity and duplex mode.

`[waveform,grid,tm] = lteTestModelTool(tm)` where a user-defined test model configuration structure is provided as an input.

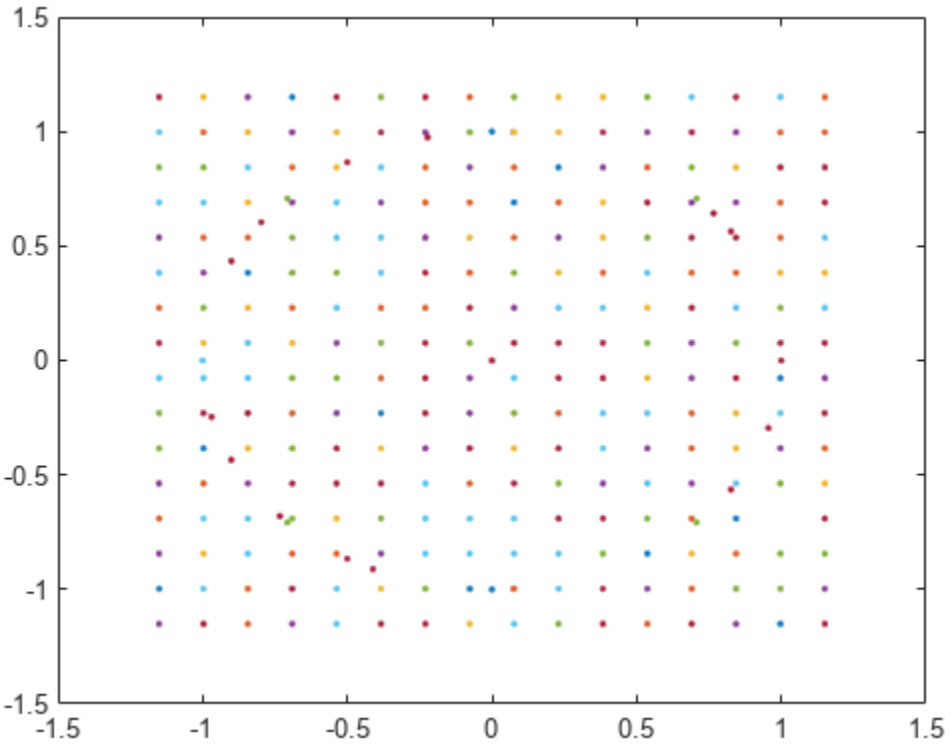
## Examples

### Generate Downlink E-TM 2a Waveform

Generate a time domain signal, `txWaveform`, and a 2-dimensional array of the Resource Elements, `txGrid`, for Test Model TS 36.141 E-TM 2a with 10MHz bandwidth. This is a 256QAM E-TM.

Specify test model number and bandwidth. Generate `txWaveform`. Plot the `txGrid` output.

```
[txWaveform,txGrid,tm] = lteTestModelTool('2a','10MHz');
plot(txGrid,'.')
```



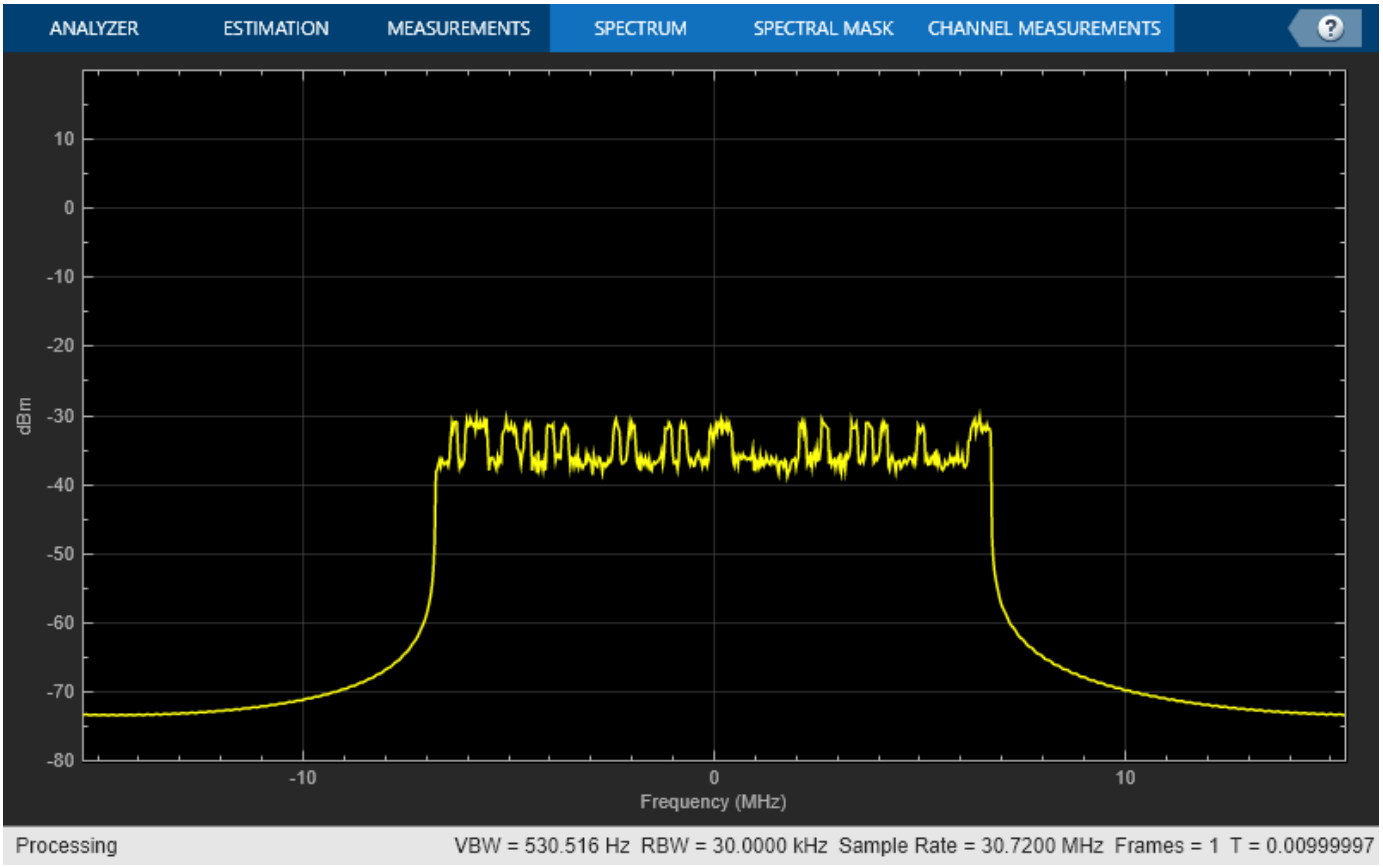

The plot of all the complex resource element symbols in the frame is dominated by the 256QAM PDSCH constellation.

### Generate Downlink Waveform Using Full E-TM Configuration Structure

Generate a time domain signal, `txWaveform`, and a 2-dimensional array of the Resource Elements, `txGrid`, for Test Model TS 36.141 E-TM 3.2 with 15MHz bandwidth.

Specify test model number and bandwidth for `tmCfg` configuration structure and create it. Generate `txWaveform`. View the waveform with a spectrum analyzer.

```
tmn = '3.2';
bw = '15MHz';
tmCfg = lteTestModel(tmn,bw);

[txWaveform,txGrid,tm] = lteTestModelTool(tmCfg);

saScope = spectrumAnalyzer(SampleRate = tm.SamplingRate);
saScope(txWaveform)
```




## Input Arguments

### `tmn` — Test model number

`'1.1'` | `'1.2'` | `'2'` | `'2a'` | `'2b'` | `'3.1'` | `'3.1a'` | `'3.1b'` | `'3.2'` | `'3.3'`

Test model number, specified as a character vector or string scalar. Use double quotes for string. For more information on these test model numbers, see TS 36.141 [1], Section 6.1.

Example: `'3.2'`

Data Types: `char` | `string`

### `bw` — Channel bandwidth

`'1.4MHz'` | `'3MHz'` | `'5MHz'` | `'10MHz'` | `'15MHz'` | `'20MHz'` | `'9RB'` | `'11RB'` | `'27RB'` | `'45RB'` | `'64RB'` | `'91RB'`

Channel bandwidth, specified as a character vector or string scalar. Use double quotes for string. You can set the nonstandard bandwidths, `'9RB'`,`'11RB'`,`'27RB'`,`'45RB'`,`'64RB'`, and `'91RB'`, only when `tmn` is `'1.1'`. These nonstandard bandwidths specify custom test models.

Example: `'15MHz'`

Data Types: `char` | `string`

### `ncellid` — Physical layer cell identity

1 or 10 (default) | optional | integer

Physical layer cell identity, specified as an integer. If you do not specify this argument, the default is 1 for standard bandwidths and 10 for non-standard bandwidths.

Example: 1

Data Types: `double`

**`duplexmode` — Duplex mode of the generated waveform**
`'FDD'` (default) | optional | `'TDD'`

Duplex mode of the generated waveform, specified as `'FDD'` or `'TDD'`. Optional.

Example: `'FDD'`

Data Types: `char` | `string`

**`tm` — User-defined test model configuration**
scalar structure

User-defined test model configuration, specified as a scalar structure. You can use `lteTestModel` to generate the various `tm` configuration structures as per TS 36.141, Section 6 [1]. This configuration structure then can be modified as per requirements and used to generate the `waveform`.

Data Types: `struct`

## Output Arguments

**`waveform` — Generated E-TM time-domain waveform**
numeric matrix

Generated E-TM time-domain waveform, returned as a *T*-by-*P* numeric matrix, where *P* is the number of antennas and *T* is the number of time-domain samples. TS 36.141 [1], Section 6 fixes $P = 1$, making `waveform` a *T*-by-1 column vector.

Data Types: `double`

**`grid` — Resource grid**
2-D numeric array

Resource grid, returned as a 2-D numeric array of resource elements for a number of subframes across a single antenna port. The number of subframes (10 for FDD and 20 for TDD), start from subframe zero, across a single antenna port, as specified in TS 36.141 [1], Section 6.1. Resource grids are populated as described in “Represent Resource Grids”.

Data Types: `double`

**`tm` — Test model configuration**
scalar structure

E-UTRA test model (E-TM) configuration, returned as a scalar structure. `tm` contains the following fields.

Test model configuration, returned as a scalar structure containing information about the OFDM modulated waveform as described in `lteOFDMInfo` and test model specific configuration parameters as described in `lteTestModel`. These fields are included in the output structure:

| Parameter Field | Values | Description |
|---|---|---|
| **TMN** | `'1.1'`, `'1.2'`, `'2'`, `'2a'`, `'2b'`, `'3.1'`, `'3.1a'`, `'3.1b'` `'3.2'`, `'3.3'` | Test model number |
| **BW** | `'1.4MHz'`, `'3MHz'`, `'5MHz'`, `'10MHz'`, `'15MHz'`, `'20MHz'`, `'9RB'`, `'11RB'`, `'27RB'`, `'45RB'`, `'64RB'`, `'91RB'`, | Channel bandwidth type, in MHz, returned as a character vector. Non-standard bandwidths, `'9RB'`, `'11RB'`, `'27RB'`, `'45RB'`, `'64RB'`, and `'91RB'`, specify custom test models. |
| **NDLRB** | Nonnegative integer | Number of downlink resource blocks ($N_{RB}^{DL}$) |
| **CellRefP** | 1 | Number of cell-specific reference signal antenna ports. This argument is for informational purposes and is read-only. |
| **NCellID** | Integer from 0 to 503 | Physical layer cell identity |
| **CyclicPrefix** | `'Normal'` | Cyclic prefix length. This argument is for informational purposes and is read-only. |
| **CFI** | 1, 2, or 3 | Control format indicator value |
| **Ng** | `'Sixth'`, `'Half'`, `'One'`, `'Two'` | HICH group multiplier |
| **PHICHDuration** | `'Normal'`, `'Extended'` | PHICH duration |
| **NSubframe** | 0 (default), nonnegative scalar integer | Subframe number<br><br>This argument is for informational purposes and is read-only. |
| **TotSubframes** | Nonnegative scalar integer | Total number of subframes to generate |
| **Windowing** | Nonnegative scalar integer | Number of time-domain samples over which windowing and overlapping of OFDM symbols is applied |
| **DuplexMode** | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |
| **CellRSPower** | Numeric value | Cell-specific reference symbol power adjustment, in dB |
| **PDSCH** | Scalar structure | PDSCH transmission configuration substructure |
| **PSSPower** | Numeric value | Primary synchronization signal (PSS) symbol power adjustment, in dB |
| **SSSPower** | Numeric value | Secondary synchronization signal (SSS) symbol power adjustment, in dB |
| **PBCHPower** | Numeric value | PBCH symbol power adjustment, in dB |
| **PCFICHPower** | Numeric value | PCFICH symbol power adjustment, in dB |

| Parameter Field | Values | Description |
|---|---|---|
| **NAllocatedPDCCHREG** | Nonnegative integer | Number of allocated PDCCH REGs. This argument is derived from `tmn` and `bw`. |
| **PDCCHPower** | Numeric value | PDCCH symbol power adjustment, in dB |
| **PDSCHPowerBoosted** | Numeric value | PDSCH symbol power adjustment, in dB, for the boosted physical resource blocks (PRBs) |
| **PDSCHPowerDeboosted** | Numeric value | PDSCH symbol power adjustment, in dB, for the de-boosted physical resource blocks (PRBs) |
| These fields are present only when `DuplexMode` is set to `'TDD'`. | | |
| **SSC** | Integer from 0 to 9<br><br>8 (default) | Special subframe configuration (SSC)<br><br>`SSC` enumerates the special subframe configuration. TS 36.211 [2], Section 4.2 specifies the special subframe configurations (lengths of DwPTS, GP, and UpPTS). |
| **TDDConfig** | Integer from 1 to 6<br><br>3 (default) | Uplink–downlink configuration<br><br>`TDDConfig` enumerates the subframe uplink-downlink configuration to be used in this frame. TS 36.211 [2], Section 4.2 specifies uplink-downlink configurations (uplink, downlink, and special subframe combinations). |
| **AllocatedPRB** | Numeric array | Allocated physical resource block list |
| **SamplingRate** | Numeric value | Sampling rate of the time-domain waveform |
| **Nfft** | Positive integer | Number of fast Fourier transform (FFT) points |

**PDSCH substructure**

The substructure PDSCH relates to the physical channel configuration and contains these fields:

| Parameter Field | Values | Description |
|---|---|---|
| **NLayers** | `1` | Number of transmission layers, returned as `1`. This argument is for informational purposes and is read-only. |
| **TxScheme** | `'Port0'` | Transmission scheme. The E-TMs have a single antenna port. This argument is for informational purposes and is read-only. |
| **Modulation** | Cell array of one or two character vectors. Valid values of character vectors include: `'QPSK'`, `'16QAM'`, `'64QAM'`, `'256QAM'`, `'1024QAM'` | Modulation formats, specifying the modulation formats for boosted and deboosted PRBs. This argument is for informational purposes and is read-only. |

Data Types: `struct`

# Version History

**Introduced in R2014a**

**R2019b: This function now opens the LTE Waveform Generator app**
*Behavior changed in R2019b*

In previous releases, the input-free syntaxes of this function opened the **LTE Test Model Generator** app. Starting in R2019b, input-free calls to this function open the **LTE Waveform Generator** app for an E-TM waveform.

## References

[1] 3GPP TS 36.141. “Evolved Universal Terrestrial Radio Access (E-UTRA); Base Station (BS) Conformance Testing.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

**Apps**
**LTE Waveform Generator**

**Functions**
`lteTestModel` | `lteDLConformanceTestTool` | `lteRMCDLTool` | `lteRMCULTool`

**Topics**
“Generate a Test Model”

# lteTurboDecode

Turbo decoding

## Syntax

```
out = lteTurboDecode(in)
out = lteTurboDecode(in,nturbodecits)
```

## Description

`out = lteTurboDecode(in)` returns the result of turbo decoding the input data `in`. The function can decode single data vectors or cell arrays of data vectors. In the case of cell array input, the output is a cell array containing the separately decoded input array vectors. The input data is assumed to be soft bit data that has been encoded with the parallel concatenated convolutional code (PCCC), as defined in TS 36.212 [1], Section 5.1.3.2. Each input data vector is assumed to be structured as three encoded parity streams concatenated in a block-wise fashion, `[S P1 P2]`, where `S` is the vector of systematic bits, `P1` is the vector of encoder 1 bits, and `P2` is the vector of encoder 2 bits. The decoder uses a default value of 5 iteration cycles. It returns the decoded bits in output vector `out` after performing turbo decoding using a sub-log-MAP (Max-Log-MAP) algorithm.

`out = lteTurboDecode(in,nturbodecits)` provides control over the number of turbo decoding iteration cycles via parameter `nturbodecits`. The `nturbodecits` is an optional parameter. If it is not provided, it uses the default value of 5 iteration cycles.

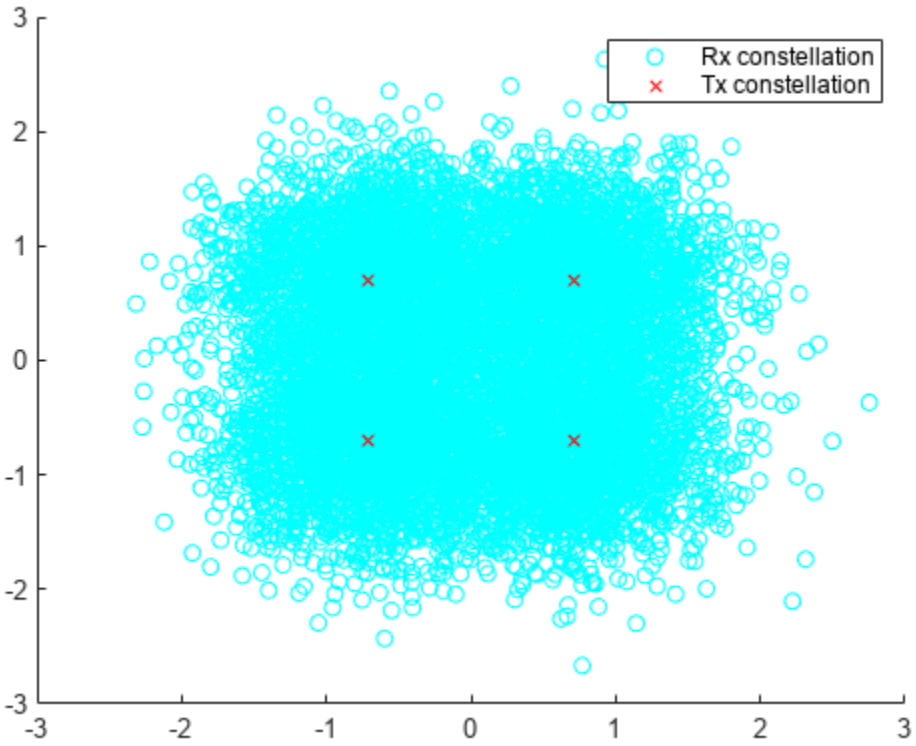
## Examples

### Turbo Decode Input Data

Perform turbo decoding of soft bits obtained from a noisy constellation.

Create a bit stream, turbo encode the bit stream, and modulate it. Create noise, add it to the modulated symbols. Display the transmitted and received symbols on a scatter plot.

```
txBits = randi([0 1],6144,1);
codedData = lteTurboEncode(txBits);
txSymbols = lteSymbolModulate(codedData,'QPSK');
noise = 0.5*complex(randn(size(txSymbols)),randn(size(txSymbols)));
rxSymbols = txSymbols + noise;

scatter(real(rxSymbols),imag(rxSymbols),'co');
hold on;

scatter(real(txSymbols),imag(txSymbols),'rx')
legend('Rx constellation','Tx constellation')
```




Demodulate the symbols and turbo decode soft bits. Compare the transmitted and recovered bits.

```
softBits = lteSymbolDemodulate(rxSymbols,'QPSK','Soft');
rxBits = lteTurboDecode(softBits);
numberErrors = sum(rxBits ~= int8(txBits))
```

```
numberErrors = 0
```

## Input Arguments

### `in` — Soft bit input data

numeric vector | numeric cell array of vectors

Soft bit input data, specified as a numeric vector or a cell array of vectors. The decoder expects the input bits to be encoded with the parallel concatenated convolutional code (PCCC), as defined in TS 36.212 [1], Section 5.1.3.

Data Types: `int8` | `double` | `cell`

### `nturbodecits` — Number of turbo decoding iteration cycles

5 (default) | optional | positive scalar integer (1...30)

Number of turbo decoder iteration cycles, specified as a positive scalar integer between 1 and 30. Optional.

Data Types: `double`

## Output Arguments

**`out` — Turbo decoded bits**
integer column vector | cell array of integer column vectors

Turbo decoded bits, returned as an integer column vector or a cell array of integer column vectors.

Data Types: `int8` | `cell`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteTurboEncode` | `lteRateRecoverTurbo` | `lteCodeBlockDesegment` | `lteDLSCHDecode` | `lteULSCHDecode` | `lteConvolutionalDecode`

# lteTurboEncode

Turbo encoding

## Syntax

```
out = lteTurboEncode(in)
```

## Description

`out = lteTurboEncode(in)` returns the result of turbo encoding the input data, `in`. Only a finite number of acceptable data vector lengths can be coded. For more information, see TS 36.212 [1], Table 5.1.3-3. Filler bits are supported through negative input values.

The encoder is a parallel concatenated convolutional code (PCCC) with two 8-state constituent encoders and a contention-free interleaver. The coding rate of turbo encoder is 1/3. The three encoded parity streams are concatenated block-wise to form the encoded output, `[S P1 P2]`, where `S` is the vector of systematic bits, `P1` is the vector of encoder 1 bits, and `P2` is the vector of encoder 2 bits. To support the correct processing of filler bits, negative input bit values are specially processed. They are treated as logical 0 at the input to both encoders but their negative values are passed directly through to the associated output positions in subblocks `S` and `P1`.

## Examples

### Turbo Encode Input Data

Perform turbo encoding for a cell array input.

```
bits = lteTurboEncode({ones(40,1),ones(6144,1)})
```

```
bits=1×2 cell array
    {132x1 int8}    {18444x1 int8}
```

## Input Arguments

### `in` — Input data

numeric vector | numeric cell array of vectors

Input data, specified as a numeric vector or a cell array of vectors.

Data Types: `int8` | `double` | `cell`

## Output Arguments

### `out` — Turbo encoded bits

integer column vector | cell array of integer column vectors

Turbo encoded bits, returned as an integer column vector or a cell array of integer column vectors. If the input is a cell array, the output is a cell array containing the separately encoded input array vectors.

Data Types: `int8` | `cell`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteTurboDecode` | `lteCodeBlockSegment` | `lteRateMatchTurbo` | `lteDLSCH` | `lteULSCH` | `lteConvolutionalEncode`

# lteTransmitDiversityDecode

Orthogonal space frequency block code decoding

## Syntax

```
[out,csi] = lteTransmitDiversityDecode(in,hest)
```

## Description

`[out,csi] = lteTransmitDiversityDecode(in,hest)` performs orthogonal space frequency block code (OSFBC) decoding of received symbols, `in`, and channel estimate, `hest`, returning the result in `out`.

## Examples

### Perform OSFBC decoding of PDSCH symbols

This example shows orthogonal space frequency block code (OSFBC) decoding of PDSCH symbols, using ideal channel estimates.

Generate a resource grid using multiple antennas to transmit a single PDSCH codeword.

```
enb = lteRMCDL('R.11');
enb.TotSubframes = 1;
[~,txGrid] = lteRMCDLTool(enb,[1;0;0;1]);
```

Extract the PDSCH symbols from this transmit grid

```
[ind,indInfo] = ltePDSCHIndices(enb,enb.PDSCH,enb.PDSCH.PRBSet);
pdschSym = txGrid(ind);
```

Create an ideal (identity) channel estimate

```
hest = permute(repmat(eye(enb.CellRefP),[1 1 indInfo.Gd]),[3 1 2]);
```

Deprecode the PDSCH symbols using the channel estimates

```
[out,csi] = lteTransmitDiversityDecode(pdschSym,hest);
```

## Input Arguments

### `in` — Received input symbols

numeric matrix

Received input symbols, specified as a numeric matrix. It has size *M*-by-*NRxAnts*, where *M* is the number of received symbols for each of *NRxAnts* receive antennas.

Data Types: `double`
Complex Number Support: Yes

**`hest` — Channel estimate**
3-D numeric array

Channel estimate, specified as a 3-D numeric array. It has size *M*-by-*NRxAnts*-by-*CellRefP*. *M* is the number of received symbols in `in`. *NRxAnts* is the number of receive antennas. *CellRefP* is the number of cell-specific reference signal antenna ports.

Data Types: `double`
Complex Number Support: Yes

## Output Arguments

**`out` — Decoded received symbols**
complex-valued numeric column vector

Decoded received symbols, returned as a complex-valued numeric column vector. It has size *M*-by-1, where *M* is the number of received symbols for each receive antenna.

Data Types: `double`
Complex Number Support: Yes

**`csi` — Soft channel state information**
numeric column vector

Soft channel state information (CSI), returned as a numeric column vector. It has size *M*-by-1, where *M* is the number of received symbols for each receive antenna. `csi` provides an estimate of the received RE gain for each received RE.

Data Types: `double`

# Version History
**Introduced in R2014a**

## See Also
`lteDLDeprecode`

# lteUCI3Decode

PUCCH format 3 transmission UCI decoding

## Syntax

```
ucibits = lteUCI3Decode(cw,n)
```

## Description

`ucibits = lteUCI3Decode(cw,n)` returns a column vector of decoded UCI bits, `ucibits`, resulting from decoding the soft bit column vector, `cw`. Where the output vector `ucibits` is expected to contain `n` bits. `ucibits` is empty if no HARQ-ACK bits are detected.

The decoder uses a maximum likelihood (ML) approach, assuming that `cw` has been demodulated using `ltePUCCH3Decode`, whose input had already been equalized to best restore the originally transmitted complex values. Specifically, this function assumes that `cw` is properly scaled to reflect a QPSK constellation (± sqrt(2)/2 amplitude for real and imaginary parts). If multiple decoded UCI bit vectors have a likelihood equal to the maximum, `ucibits` is a matrix where each column represents one of the equally likely bit vectors. If a minimum likelihood threshold is not met, `ucibits` is empty.

## Examples

### Encoding and decoding HARQ-ACK feedback for PUCCH Format 3

This example shows how to encode and decode an ACK using PUCCH format 3 transmission UCI decoding.

Create a Tx ACK vector. Encode the vector using PUCCH format 3. Convert the logical bits into soft data.

```
txAck = [1;0;0;1];

cw = lteUCI3Encode(txAck);

cw = (double(cw)-0.5)*sqrt(2.0);
```

Decode the received data using the PUCCH format 3 UCI decoder. Verify that the Rx ACK vector matches the Tx ACK vector.

```
rxAck = lteUCI3Decode(cw,length(txAck))
```

```
rxAck = 4x1 logical array

   1
   0
   0
   1
```

## Input Arguments

### `cw` — Soft bits to decode

numeric column vector

Soft bits to decode, specified as a numeric column vector.

Data Types: `int8` | `double`

### `n` — Number of bits to return

positive scalar integer (1...22)

Number of bits to return, specified as a positive scalar integer from 1 through 22.

Data Types: `double`

## Output Arguments

### `ucibits` — Concatenated HARQ-ACK bits, periodic CSI bits, and Scheduling Request (SR) bit

logical column vector

Concatenated HARQ-ACK bits, periodic CSI bits, and Scheduling Request (SR) bit, returned as a logical column vector. `ucibits` represents the $[a_0, a_1, \ldots a_{N-1}]$ bit sequence as described in TS 36.212 [1], Section 5.2.3.1. The number of bits returned, $N$, is defined by the input argument `n`.

`ucibits` is empty if no UCI bits are detected.

Data Types: `logical`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteUCI3Encode` | `ltePUCCH3Decode`

# lteUCI3Encode

PUCCH format 3 transmission UCI encoding

## Syntax

```
cw = lteUCI3Encode(ucibits)
```

## Description

`cw = lteUCI3Encode(ucibits)` returns a column vector of coded UCI bits, `cw`, resulting from processing of control information, `ucibits` for PUCCH format 3. The `ucibits` is a vector of concatenated HARQ-ACK bits and any appended periodic CSI bits and/or scheduling request (SR) bits.

The UCI processing is defined in TS 36.212 [1], Section 5.2.3.1, and consists of a (32,*O*) block code, where *O* is the number of bits in `ucibits`. The coded bit vector, `cw`, is 48 bits long.

## Examples

### Encode HARQ-ACK Feedback for PUCCH Format 3

Encode and decode HARQ-ACK feedback for PUCCH format 3.

Create a Tx ACK vector. Encode the vector using PUCCH format 3. Turn logical bits into 'LLR' data.

```
txAck = [1;0;0;1];
cw = lteUCI3Encode(txAck);
cw(cw == 0) = -1;
```

Decode the received data using the PUCCH format 3 UCI decoder. Verify that the Rx ACK vector matches the Tx ACK vector.

```
rxAck = lteUCI3Decode(cw,length(txAck))
```

```
rxAck = 4x1 logical array

   1
   0
   0
   1
```

## Input Arguments

**`ucibits` — Concatenated HARQ-ACK bits, periodic CSI bits, and Scheduling Request (SR) bit**
logical vector of length 1–22

Concatenated HARQ-ACK bits, periodic CSI bits, and Scheduling Request (SR) bit, specified as a logical vector containing from 1 to 22 bits. `ucibits` represents the [$a_0$, $a_1$, ... $a_{N-1}$] bit sequence as described in TS 36.212 [1], Section 5.2.3.1.

Data Types: `logical` | `double`

## Output Arguments

### `cw` — Coded UCI bits

48-by-1 integer column vector

Coded UCI bits, returned as a 48-by-1 integer column vector. This coded bit vector is 48 bits long.

Data Types: `int8`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteUCI3Decode` | `ltePUCCH3`

# lteUCIDecode

PUCCH format 2, 2a, and 2b transmission UCI decoding

## Syntax

```
ucibits = lteUCIDecode(cw,n)
```

## Description

`ucibits = lteUCIDecode(cw,n)` returns a vector of decoded UCI bits, `ucibits`, resulting from decoding the soft bit column vector, `cw`, where the output vector, `ucibits`, is expected to contain `n` bits. `ucibits` is a column vector of CQI/PMI or RI bits (UCI), representing the CQI/PMI or RI information fields described in TS 36.212, Section 5.2.3.3 [1]. `n` must be between 1 and 13. The decoder uses a maximum likelihood approach assuming that `cw` has been demodulated using `ltePUCCH2Decode` whose input had already been equalized to best restore the originally transmitted complex values. If multiple decoded UCI bit vectors have a likelihood equal to the maximum, UCIBITS will be a matrix where each column represents one of the equally likely bit vectors

## Examples

### Decode UCI Bits

Decode UCI bits representing RI=3 using N=2 bits. According to TS 36.212, Table 5.2.2.6-6 this maps to the set of input bits `[1;0]`.

```
cw = lteUCIEncode([1;0])
```

```
cw = 20x1 int8 column vector

   1
   1
   1
   1
   1
   1
   1
   1
   1
   1
      ⋮
```

```
softBits = double(cw)/sqrt(2);
decodedUciBits = lteUCIDecode(softBits, 2)
```

```
decodedUciBits = 2x1 logical array

   1
   0
```

The decoded UCI bits match the input bits.

## Input Arguments

**`cw` — Codeword of soft bits**
numeric column vector

Codeword of soft bits, specified as a numeric column vector.

Data Types: `logical` | `double`

**`n` — Number of bits**
1...11

Number of bits, specified as a scalar integer from 1 to 11.

Data Types: `double`

## Output Arguments

**`ucibits` — Decoded UCI bits**
logical column vector

Decoded UCI bits, returned as a logical column vector. UCI bits are CQI/PMI or RI information.

Data Types: `logical`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteUCIEncode` | `lteUCI3Encode` | `ltePUCCH2Decode`

# lteUCIEncode

PUCCH format 2, 2a, and 2b transmission UCI encoding

## Syntax

```
cw = lteUCIEncode(ucibits)
```

## Description

`cw = lteUCIEncode(ucibits)` returns a column vector of coded UCI bits, `cw`, resulting from processing of control information, `ucibits`. `ucibits` is a column vector of CQI/PMI or RI bits (UCI), representing the CQI/PMI or RI information fields described in TS 36.212, Section 5.2.3.3 [1]. `ucibits` should be a vector containing up to 13 bits. For PUCCH formats 2a and 2b with extended cyclic prefix, this vector should also contain the appended 1 or 2 HARQ-ACK bits for joint encoding.

The UCI processing is defined in TS 36.212, Section 5.2.3 [1], and consists of a (20,*A*) block code, where *A* is the number of bits in `ucibits`. The coded bit vector, `cw`, is 20 bits long.

## Examples

### Encode UCI Bits

Encode UCI bits representing RI=3 using two bits. According to TS 36.212, Table 5.2.2.6-6 this maps to the set of input bits [1; 0].

```
cw = lteUCIEncode([1;0])
```

```
cw = 20x1 int8 column vector

   1
   1
   1
   1
   1
   1
   1
   1
   1
   1
      ⋮
```

## Input Arguments

### `ucibits` — Control information bits

logical vector of length 1 to 13

Control information bits, specified as a logical vector of length 1 to 13. This vector contains the CQI/PMI or RI logical bits (UCI), representing the CQI/PMI or RI information fields. It should be up to

13 bits in length. For PUCCH formats 2a and 2b with extended cyclic prefix, this vector should also contain the appended 1 or 2 HARQ-ACK bits for joint encoding.

Data Types: `logical`

## Output Arguments

### `cw` — Coded UCI bits

20-by-1 integer column vector

Coded UCI bits, returned as a 20-by-1 integer column vector. The coded bit vector is 20 bits long.

Data Types: `int8`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteUCIDecode` | `lteUCI3Decode` | `ltePUCCH2`

# lteULChannelEstimate

PUSCH uplink channel estimation

## Syntax

```
[hest, noiseest] = lteULChannelEstimate(ue,chs,rxgrid)
[hest, noiseest] = lteULChannelEstimate(ue,chs,cec,rxgrid)
[hest, noiseest] = lteULChannelEstimate(ue,chs,cec,rxgrid,refgrid)
[hest, noiseest] = lteULChannelEstimate(ue,chs,rxgrid,refgrid)
```

## Description

`[hest, noiseest] = lteULChannelEstimate(ue,chs,rxgrid)` returns an estimate for the channel by averaging the least squares estimates of the reference symbols across time and copying these estimates across the allocated resource elements within the time frequency grid. It returns the estimated channel between each transmit and receive antenna and an estimate of the noise power spectral density. See “Algorithms” on page 2-1219.

`[hest, noiseest] = lteULChannelEstimate(ue,chs,cec,rxgrid)` returns the estimated channel using the method and parameters defined by the user in the channel estimator configuration `cec` structure.

`[hest, noiseest] = lteULChannelEstimate(ue,chs,cec,rxgrid,refgrid)` returns the estimated channel using the method and parameters defined by the channel estimation configuration structure and the additional information about the transmitted symbols found in `refgrid`.

When `cec.InterpType` is set to `'None'`, values in `refgrid` are treated as reference symbols and the resulting `hest` contains non-zero values in their locations.

`[hest, noiseest] = lteULChannelEstimate(ue,chs,rxgrid,refgrid)` returns the estimated channel using the estimation method as described in TS 36.101 [1], Annex F4. The method described utilizes extra channel information obtained through information of the transmitted symbols found in `refgrid`. This additional information allows for an improved estimate of the channel and is required for accurate EVM measurements. `rxgrid` and `refgrid` must only contain a whole subframe worth of SC-FDMA symbols.

## Examples

### Estimate Channel Characteristics for PUSCH

Use `lteULChannelEstimate` to estimate the channel characteristics for a received resource grid.

Initialize a UE configuration structure to RMC `A3-2`. Initialize the channel estimation configuration structure. Generate a transmission waveform. For the purpose of this example, we bypass the channel stage of the system model and copy `txWaveform` to `rxWaveform`.

```
ue = lteRMCUL('A3-2');
ue.TotSubframes = 1;
cec = struct('FreqWindow',7,'TimeWindow',1,'InterpType','cubic');
```

```
txWaveform = lteRMCULTool(ue,[1;0;0;1]);
rxWaveform = txWaveform;
```

Demodulate the SC-FDMA waveform and perform channel estimation operation on `rxGrid`.

```
rxGrid = lteSCFDMADemodulate(ue,rxWaveform);
hest = lteULChannelEstimate(ue,ue.PUSCH,cec,rxGrid);
```

## Input Arguments

### ue — UE-specific configuration

structure

UE-specific configuration, specified as a structure. `ue` can contain the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NULRB`** | Required | 6, 15, 25, 50, 75, 100 | Number of uplink resource blocks. ($N_{\mathrm{RB}}^{\mathrm{UL}}$) |
| **`NCellID`** | Required | Nonnegative scalar integer | Physical layer cell identity |
| **`NSubframe`** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **`CyclicPrefixUL`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length for uplink. |
| **`NTxAnts`** | Optional | 1 (default), 2, 4 | Number of transmission antennas. |
| **`Hopping`** | Optional | `'Off'` (default), `'Group'`, or `'Sequence'` | Frequency hopping method. |
| **`SeqGroup`** | Optional | 0 (default), integer from 0 to 29 | PUSCH sequence group assignment ($\Delta_{\mathrm{SS}}$).<br><br>Only used if `NDMRSID` or `NPUSCHID` is absent. |
| **`CyclicShift`** | Optional | 0 (default), integer from 0 to 7 | Number of cyclic shifts used for PUSCH DM-RS (yields $n_{DMRS}^{(1)}$). |
| **`NPUSCHID`** | Optional | 0 (default), nonnegative scalar integer from 0 to 509 | PUSCH virtual cell identity. If this field is not present, `NCellID` is used for group hopping sequence-shift pattern initialization.<br><br>See footnote. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NDMRSID** | Optional | 0 (default), nonnegative scalar integer from 0 to 509 | DM-RS identity for cyclic shift hopping ($n_{ID}^{csh\_DMRS}$). If this field is not present, NCellID is used for cyclic shift hopping initialization.<br><br>See footnote. |
| **1** The pseudorandom sequence generator for cyclic shift hopping is initialized according to NDMRSID, if present — otherwise it is initialized according to the cell identity NCellID and the sequence group assignment SeqGroup. Similarly, the sequence-shift pattern for group hopping is initialized according to NPUSCHID, if present — otherwise it is initialized according to NCellID and SeqGroup. | | | |

Data Types: struct

### chs — PUSCH channel settings

structure

PUSCH channel settings, specified as a structure that can contain the following fields. The parameter field PMI is only required if ue.NTxAnts is set to 2 or 4.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **PRBSet** | Required | Integer column vector or two-column matrix | Physical resource block set, specified as a 1-column or 2-column matrix. This parameter field contains the zero-based physical resource block (PRB) indices corresponding to the slot-wise resource allocations for this PUSCH.<br><br>If PRBSet is a column vector, the resource allocation is the same in both slots of the subframe. To specify differing PRBs for each slot in a subframe, use a 2-column matrix. The PRB indices are zero based. |
| **NLayers** | Optional | 1 (default), 2, 3, 4 | Number of transmission layers |
| **DynCyclicShift** | Optional | 0 (default), integer from 0 to 7 | Cyclic shift for DM-RS (yields $n_{DMRS}^{(2)}$). |
| **OrthoCover** | Optional | 'Off' (default), 'On' | Applies ('On'), or does not apply ('Off'), orthogonal cover sequence *w* (*Activate-DMRS-with OCC*). |
| The following field is required only when ue.NTxAnts is set to 2 or 4. | | | |
| **PMI** | Optional | nonnegative scalar integer (0,...,23)<br><br>0 (default) | Scalar precoder matrix indication (PMI) to be used during precoding<br><br>of the DRS reference symbols |

Data Types: struct

### rxgrid — Received resource element grid

3-D array

Received resource element grid, specified as an $N_{SC}$-by-$N_{Sym}$-by-$N_R$ array of complex symbols.

- $N_{SC}$ is the number of subcarriers
- $N_{Sym} = N_{SF} \times N_{SymPerSF}$
  - $N_{SF}$ is the total number of subframes. If $N_{SF}$ is greater than one, the correct region is extracted from the returned `hest` array. The location of the estimated subframe within `hest` is specified using the parameter field `cec.Window`.
  - $N_{SymPerSF}$ is the number of SC-FDMA symbols per subframe.
    - For normal cyclic prefix, each subframe contains 14 SC-FDMA symbols.
    - For extended cyclic prefix, each subframe contains 12 SC-FDMA symbols.
- $N_R$ is the number of receive antennas

Data Types: `double`
Complex Number Support: Yes

**`cec` — Channel estimator configuration**
structure

Channel estimator configuration, specified as a structure with these fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`FreqWindow`** | Required | Nonnegative scalar integer | Size of window in resource elements used to average over frequency during channel estimation<br><br>The window size must be either an odd number or a multiple of 12. |
| **`TimeWindow`** | Required | Nonnegative scalar integer | Size of window in resource elements used to average over time during channel estimation<br><br>The window size must be an odd number. |
| **`InterpType`** | Required | `'nearest'`, `'linear'`, `'natural'`, `'cubic'`, `'v4'`, `'none'`<br><br>See footnote. | Type of 2-D interpolation used during interpolation. For details, see `griddata`. Supported choices are shown in the following table.<br><br>**Value** — **Description**<br>`'nearest'` — Nearest neighbor interpolation<br>`'linear'` — Linear interpolation<br>`'natural'` — Natural neighbor interpolation<br>`'cubic'` — Cubic interpolation<br>`'v4'` — MATLAB 4 `griddata` method<br>`'none'` — Disables interpolation |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`PilotAverage`** | Optional | `'UserDefined'`(default), `'TestEVM'`<br><br>See footnote. | Type of pilot averaging |
| **`Reference`** | Optional | `'Antennas'` (default), `'Layers'`, `'None'`<br><br>See footnote. | Specifies point of reference (signals to internally generate) for channel estimation |
| The following field is required only when `rxgrid` contains more than one subframe. See footnote. | | | |
| **`Window`** | Optional | `'Left'`, `'Right'`, `'Centred'`, `'Centered'` | If more than one subframe is input this parameter is required to indicate the position of the subframe from *rxgrid* and *refgrid* containing the desired channel estimate. Only channel estimates for this subframe will be returned. For the `'Centred'` and `'Centered'` settings, the window size must be odd. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **1** | For `cec.InterpType = 'none'`, no interpolation is performed between pilot symbols and no virtual pilots are created. `hest` will contain channel estimates in the locations of transmitted reference symbols for each received antenna and all other elements of `hest` are zero. The averaging of pilot symbols estimates described by `cec.TimeWindow` and `cec.FreqWindow` are still performed. | | |
| **2** | The `'UserDefined'` pilot averaging uses a rectangular kernel of size `cec.FreqWindow`-by-`cec.TimeWindow` and performs a 2-D filtering operation upon the pilots. Pilots near the edge of the resource grid are averaged less as they have no neighbors outside of the grid. For `cec.FreqWindow` = 12×*X* (i.e. any multiple of 12) and `cec.TimeWindow` = 1 the estimator enters a special case where an averaging window of (12×*X*)-in-frequency is used to average the pilot estimates; the averaging is always applied across (12×*X*) subcarriers, even at the upper and lower band edges; therefore the first (6×*X*) symbols at the upper and lower band edge have the same channel estimate. This operation ensures that averaging is always done on 12 (or a multiple of 12) symbols. This provides the appropriate despreading operation required for the case multi-antenna transmission where the DM-RS signals associated with each antenna occupy the same time/frequency locations but use different orthogonal cover codes to allow them to be differentiated at the receiver. The `'TestEVM'` pilot averaging ignores other structure fields in `cec`, and follows the method described in TS 36.101, Annex F for the purposes of transmitter EVM testing. | | |
| **3** | Setting `cec.Reference` to `'Antennas'` uses the PUSCH DMRS after precoding onto the transmission antennas as the reference for channel estimation. In this case, the precoding matrix indicated in `chs.PMI` is used to precode the DMRS layers onto antennas, and the channel estimate, `hest`, is a matrix of size *M*-by-*N*-by-`NRxAnts`-by-`chs.NTxAnts`. Setting `cec.Reference` to `'Layers'` uses the PUSCH DMRS without precoding as the reference for channel estimation. The channel estimate, `hest`, is of size *M*-by-*N*-by-`NRxAnts`-by-`chs.NLayers`. Setting `cec.Reference` to `'None'` generates no internal reference signals, and the channel estimation can be performed on arbitrary known REs as given by the `refgrid` argument. This approach can be used to provide a `refgrid` containing the SRS signals created on all `NTxAnts`, allowing for full-rank channel estimation for the purposes of PMI selection when the PUSCH is transmitted with less than full rank. | | |
| **4** | When `rxgrid` contains more than one subframe, `cec.Window` provides control of the location of the subframe for which channel estimation is performed. This allows channel estimation for the subframe of interest to be aided by the presence of pilot symbols occupying the same resource block in subframes before and/or after that subframe. For example, if `rxgrid` contains five subframes, `'Left'` estimates the last first subframe in `rxgrid`, `'Centred'`/`'Centered'` estimates the third (middle) subframe, and `'Right'` estimates the last subframe. The parameter `ue.NSubframe` corresponds to the chosen subframe. So, with three subframes and `cec.Window = 'Right'`, `rxgrid` corresponds to subframes (`ue.NSubframe-2`, `ue.NSubframe-1`, `ue.NSubframe`). The `hest` output will be the same size as `rxgrid` and will correspond to the same subframe numbers. All locations other than the estimated subframe will contain zeros. | | |

Data Types: `struct`

**`refgrid` — Reference array of known transmitted data symbols in their correct locations**
3-D numeric array

Reference array of known transmitted data symbols in their correct locations, specified as an $N_{SC}$-by-$N_{Sym}$-by-$N_T$ array of complex symbols. All other locations, such as DM-RS Symbols and unknown data symbol locations, must be represented by a `NaN`. The first two dimensions of `rxgrid` and `refgrid` must be the same.

- $N_{SC}$ is the number of subcarriers.

- $N_{Sym} = N_{SF} \times N_{SymPerSF}$
  - $N_{SF}$ is the total number of subframes. If $N_{SF}$ is greater than one, the correct region is extracted from the returned `hest` array. The location of the estimated subframe within `hest` is specified using the parameter field `cec.Window`.
  - $N_{SymPerSF}$ is the number of SC-FDMA symbols per subframe.
    - For normal cyclic prefix, each subframe contains 14 SC-FDMA symbols.
    - For extended cyclic prefix, each subframe contains 12 SC-FDMA symbols.
- $N_T$ is the number of transmit antennas, `ue.NTxAnts`

For `cec.InterpType = 'None'`, values in `refgrid` are treated as reference symbols and the resulting `hest` contains non-zero values in their locations. A typical use for `refgrid` is to provide values of the SRS transmitted at some point during the time span of `rxgrid`. The SRS values can be used to enhance the channel estimation.

Data Types: `double`
Complex Number Support: Yes

## Output Arguments

### `hest` — Channel estimate between each transmit and receive antenna

4-D array

Channel estimate between each transmit and receive antenna, returned as an $N_{SC}$-by-$N_{Sym}$-by-$N_R$-by-$N_T$ array of complex symbols.

- $N_{SC}$ is the number of subcarriers.
- $N_{Sym}$ is the number of SC-FDMA symbols.
- $N_R$ is the number of receive antennas.
- $N_T$ is the number of transmit antennas, `ue.NTxAnts`.

Optionally, the channel estimator can be configured to use the DM-RS layers as the reference signal. In this case, the 4-D array is an $N_{SC}$-by-$N_{Sym}$-by-$N_R$-by-$N_{Layers}$ array of complex symbols, where $N_{Layers}$ is the number of transmission layers.

### `noiseest` — Noise estimate

numeric scalar

Noise estimate, returned as a numeric scalar. This output is the power spectral density of the noise present on the estimated channel response coefficients.

## Algorithms

The channel estimation algorithm is described in the following steps.

1. Extract the demodulation reference signals, or pilot symbols, for a transmit-receive antenna pair from the allocated physical resource blocks within the received subframe.
2. Average the least-squares estimates to reduce any unwanted noise from the pilot symbols.
3. Using the cleaned pilot symbol estimates, interpolate to obtain an estimate of the channel for the entire number of subframes passed into the function.

#### Least-Squares Estimation

The least-squares estimates of the reference signals are obtained by dividing the received pilot symbols by their expected value. The least-squares estimates are affected by any system noise. This noise needs to be removed or reduced to achieve a reasonable estimation of the channel at pilot symbol locations.

#### Noise Reduction and Interpolation

To minimize the effects of noise on the pilot symbol estimates, the least-squares estimates are averaged. This simple method produces a substantial reduction in the level of noise found on the pilot symbols. The pilot symbol averaging method uses an averaging window defined by the user. The averaging window size is measured in resource elements; any pilot symbols located within the window are used to average the value of the pilot symbol found at the center of the window.

Then, the averaged pilot symbol estimates are used to perform a 2-D interpolation across allocated physical resource blocks. The location of pilot symbols within the subframe is not ideally suited to interpolation. To account for this positioning, virtual pilots are created and placed out with the area of the current subframe. This placement allows complete and accurate interpolation to be performed.

**Note** The PUSCH channel estimator is only able to deal with contiguous allocation of resource blocks in time and frequency.

## Version History

**Introduced in R2013b**

## References

[1] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteEqualizeMIMO` | `lteEqualizeMMSE` | `lteEqualizeZF` | `lteSCFDMADemodulate` | `lteULFrameOffset` | `lteULPerfectChannelEstimate` | `griddata` | `lteEqualizeULMIMO`

# lteULChannelEstimateNPUSCH

NPUSCH channel estimation

## Syntax

```
[hEst,noiseEst] = lteULChannelEstimateNPUSCH(ue,chs,rxGrid)
[hEst,noiseEst] = lteULChannelEstimateNPUSCH(ue,chs,cec,rxGrid)
[hEst,noiseEst] = lteULChannelEstimateNPUSCH( ___ ,stateIn)
```

## Description

`[hEst,noiseEst] = lteULChannelEstimateNPUSCH(ue,chs,rxGrid)` estimates the channel between transmit and receive antennas for user equipment (UE) settings `ue`, channel transmission configuration `chs`, and received resource grid `rxGrid`. The function returns `hEst`, the estimated channel, and `noiseEst`, the estimated noise power spectral density.

The function calculates `hEst` and `noiseEst` by averaging least-squares estimates of the narrowband physical uplink shared channel (NPUSCH) demodulation reference signal (DRS) symbols over time and copying these symbols across the allocated resource elements (REs) within the time-frequency grid.

`[hEst,noiseEst] = lteULChannelEstimateNPUSCH(ue,chs,cec,rxGrid)` specifies `cec`, a structure containing the method and parameters to use for channel estimation.

`[hEst,noiseEst] = lteULChannelEstimateNPUSCH( ___ ,stateIn)` specifies `stateIn`, the initial encoder state for NPUSCH DRS symbol generation, in addition to any input argument combination from previous syntaxes..

## Examples

### Estimate Uplink Channel Characteristics

Perform NPUSCH channel estimation on a received resource grid.

Configure UE-specific settings.

```
ue = struct('NNCellID',0,'NBULSubcarrierSpacing','15kHz','NSlot',0);
```

Specify a channel transmission configuration.

```
chs = struct('NPUSCHFormat','Data','NRUsc',1,'NULSlots',16,'NRU',1, ...
    'NRep',1,'NBULSubcarrierSet',0,'Modulation','QPSK');
```

Configure the channel estimation type and parameters.

```
cec = struct('FreqWindow',7,'TimeWindow',1,'InterpType','cubic','PilotAverage','UserDefined');
```

Generate the NPUSCH DRS symbols and allocate them to the appropriate locations on a resource grid.

```
grid = lteNBResourceGrid(ue);
grid(lteNPUSCHDRSIndices(ue,chs)) = lteNPUSCHDRS(ue,chs);
```

Generate a waveform by performing single-carrier frequency-division multiple access (SC-FDMA) modulation on the NPUSCH DRS symbols.

```
waveform = lteSCFDMAModulate(ue,chs,grid);
```

Perform SC-FDMA demodulation, assuming that the received waveform matches the transmitted waveform.

```
rxGrid = lteSCFDMADemodulate(ue,chs,waveform);
```

Estimate the channel.

```
[hEst,noiseEst] = lteULChannelEstimateNPUSCH(ue,chs,cec,rxGrid);
```

## Input Arguments

### `ue` — UE-specific settings

structure

UE-specific settings, specified as a structure containing these fields.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`NBULSubcarrierSpacing`** | `'3.75kHz'`, `'15kHz'` | NB-IoT uplink subcarrier spacing<br><br>To set a subcarrier spacing of 3.75 kHz, specify this field as `'3.75kHz'`. To set a subcarrier spacing of 15 kHz, specify this field as `'15kHz'`. | `char`, `string` |
| **`NNCellID`** | Integer in the interval [0, 503] | Narrowband physical layer cell identity (PCI) | `double` |
| **`NFrame`** | `0` (default), nonnegative integer | Frame number | `double` |
| **`NSlot`** | Nonnegative integer | Slot number<br><br>When you specify the `NPUSCHFormat` field as `'Data'` and the `SeqGroupHopping` field as `'Off'` in the `chs` input, the function ignores this field. | `double` |

Data Types: `struct`

### `chs` — Channel transmission configuration

structure

Channel transmission configuration, specified as a structure containing these fields.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **NPUSCHFormat** | `'Data'`, `'Control'` | NPUSCH format<br><br>To indicate that the NPUSCH carries narrowband uplink shared channel (UL-SCH) data, specify this field as `'Data'`. To indicate that the NPUSCH carries uplink control information, specify this field as `'Control'`. | `char`, `string` |
| **NRUsc** | 1, 3, 6, 12 | Number of consecutive subcarriers in a resource unit (RU)<br><br>If you specify the `NPUSCHFormat` field as `'Control'` or the `NBULSubcarrierSpacing` field of the `ue` input as `'3.75kHz'`, then you must specify this field as `1` . | `double` |
| **NRep** | 1, 2, 4, 8, 16, 32, 64, 128 | Number of repetitions for a codeword | `double` |
| **NRU** | 1, 2, 3, 4, 5, 6, 8, 10 | Number of RUs | `double` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`NULSlots`** | 2, 4, 8, 16 | Number of slots per RU<br><br>If you specify the `NPUSCHFormat` field as `'Control'`, then you must specify this field as 4.<br><br>If you specify the `NPUSCHFormat` field as `'Data'`, then you must specify this field as:<br>• 16 when you specify the `NRUsc` field as 1<br>• 8 when you specify the `NRUsc` field as 3<br>• 4 when you specify the `NRUsc` field as 6<br>• 2 when you specify the `NRUsc` field as 12 | `double` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`BaseSeqIdx`** | Integer in the interval [0, 29]<br><br>Default depends on the value of the `NRUsc` field. | Multitone NPUSCH DRS base sequence index<br><br>• When you specify the `NRUsc` field as `3`, specify this field as an integer in the interval [0, 11]. If you do not specify this field, the function sets it to the value of `mod(ue.NNCellID, 12)`.<br>• When you specify the `NRUsc` field as `6`, specify this field as an integer in the interval [0, 13]. If you do not specify this field, the function sets it to the value of `mod(ue.NNCellID, 14)`.<br>• When you specify the `NRUsc` field as `12`, specify this field as an integer in the interval [0, 29]. If you do not specify this field, the function sets it to the value of `mod(ue.NNCellID, 30)`.<br>• When you specify the `NRUsc` field as any other value, the function does not use this field.<br><br>**Dependencies.** To enable this field, specify the `NRUsc` field as `3`, `6`, or `12`. | `double` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **SeqGroupHopping** | `'On'` (default), `'Off'` | To enable sequence-group hopping, specify this field as `'On'`. To disable sequence group hopping, specify this field as `'Off'`. For more information, see section 5.5.1.3 of [1]. | `char`, `string` |
| **SeqGroup** | `0` (default), integer in the interval [0, 29] | Sequence-group assignment for sequence shift pattern calculation<br><br>For more information, see section 10.1.4.1.3 of [1].<br><br>**Dependencies.** To enable this field, specify the `SeqGroupHopping` field as `'On'`. | `double` |
| **CyclicShift** | `0` (default), integer in the interval [0, 3] | Cyclic shift<br>• When you specify the `NRUsc` field as `3`, specify this field as an integer in the interval [0, 2].<br>• When you specify the `NRUsc` field as `6`, specify this field as an integer in the interval [0, 3].<br><br>**Dependencies.** To enable this field, specify the `NRUsc` field as `3` or `6`. | `double` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`NBULSubcarrierSet`** | Integer in the interval [0, 47], vector of integers in the interval [0, 11] | NB-IoT uplink subcarrier indices, in zero-based form<br><br>If you specify the `NPUSCHFormat` field as `'Control'`, specify this field as an integer in the interval [0, 11].<br><br>If you specify the `NPUSCHFormat` field as `'Data'` and the `NBULSubcarrierSpacing` field of the `ue` input as `'3.75kHz'`, specify this field as an integer in the interval [0, 47].<br><br>If you specify the `NPUSCHFormat` field as `'Data'` and the `NBULSubcarrierSpacing` field of the `ue` input as `'15kHz'`, specify this field as a vector of integers in the interval [0, 11]. | `double` |

Data Types: `struct`

### `rxGrid` — Received resource grid

complex-valued matrix

Received resource grid, specified as a complex-valued matrix of size *T*-by-*P*.

- *T* is the number of time-domain samples.
- *P* is the number of transmit antennas.

You can generate this input by performing SC-FDMA demodulation on a received resource grid using the `lteSCFDMADemodulate` function.

Data Types: `double`
Complex Number Support: Yes

### `cec` — Channel estimation configuration

structure

Channel estimation configuration, specified as a structure containing these fields.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **FreqWindow** | Positive odd integer, positive multiple of 12 | Size of window for frequency averaging, in resource elements | `double` |
| **TimeWindow** | Positive odd integer | Size of window for time averaging, in resource elements | `double` |
| **InterpType** | `'nearest'`, `'linear'`, `'natural'`, `'cubic'`, `'v4'`, `'none'` | Type of interpolation between pilot symbols, specified as one of these values.<br>• `'nearest'` — Use nearest neighbor interpolation<br>• `'linear'` — Use linear interpolation<br>• `'natural'` — Use natural neighbor interpolation<br>• `'cubic'` — Use cubic interpolation<br>• `'v4'` — Use the MATLAB 4 `griddata` method<br>• `'none'` — The function performs no interpolation between pilot symbols and does not create virtual pilots. The `hEst` output contains channel estimates in the locations of the transmitted NPUSCH DRS symbols for each receive antenna, and all other elements of `hEst` are `0`. The function still performs pilot symbol averaging in accordance with the values you specify for the `FreqWindow` and `TimeWindow` fields.<br>For more information, see the `griddata` function. | `char`, `string` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **PilotAverage** | `'TestEVM'`, `'UserDefined'` | Type of pilot averaging<br><br>If you specify this field as `'TestEVM'`, the function ignores any other fields you specify. In this case, the function performs pilot averaging according to the method set out in and Annex F of [2].<br><br>When you specify this field as `'UserDefined'`, the function performs pilot averaging with a rectangular kernel of size `FreqWindow`-by-`TimeWindow`. The function also performs a two-dimensional filtering operation on the pilots. The pilots near the edge of the resource grid either have no neighbors or a limited number of neighbors through the creation of virtual pilots. Consequently, these pilots are not averaged in the same way as pilots that are not near the edge of the resource grid. | `char`, `string` |

Data Types: `struct`

**`stateIn` — Encoder state**
`struct()` (default) | structure

Encoder state for NPUSCH DRS generation, specified as a structure. This input corresponds to the `stateIn` input of the `lteNPUSCHDRS` function. This input contains the internal state of each transport block in these fields.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **SlotIdx** | Integer in the interval [0, (`chs.NRU` × `chs.NULSlots` × `chs.NRep`) – 1] | Index of a slot within a bundle, in zero-based form | `double` |
| **InitNSlot** | Nonnegative integer | Slot number for scrambling sequence initialization | `double` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **InitNFrame** | Nonnegative integer | Frame number for scrambling sequence initialization | double |
| **EndOfBlk** | Logical 1 (true) or 0 (false) | To indicate that the transmission has reached the end of a transport block, specify this field as 1 (true). Otherwise, specify this field as 0 (false). | logical |
| **EndOfTx** | Logical 1 (true) or 0 (false) | To indicate that the transmission has reached the end of a bundle, specify this field as 1 (true). Otherwise, specify this field as 0 (false). | logical |
| **GhpNSlot** | Nonnegative integer | Slot number for the first slot in the RU<br><br>**Dependencies.** To enable this field, specify the NPUSCHFormat field as 'Data' and the NRUsc field as 1 in the chs input. | double |

Data Types: struct

## Output Arguments

### hEst — Channel estimate

complex-valued array

Channel estimate, returned as a complex-valued array of size *K*-by-*L*-by-*R*.

- *K* is the total number of subcarriers.
- *L* is the number of SC-FDMA symbols.
- *R* is the number of receive antennas.

Data Types: double
Complex Number Support: Yes

### noiseEst — Noise power spectral density

real-valued scalar

Noise power spectral density, returned as a real-valued scalar. This output represents the power spectral density of the noise present on the estimated channel response coefficients.

Data Types: double

# Version History

**Introduced in R2020a**

## References

[1] 3GPP TS 36.211. “Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. https://www.3gpp.org.

[2] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. https://www.3gpp.org.

## See Also

**Functions**

`griddata` | `lteEqualizeMMSE` | `lteEqualizeZF` | `lteNPUSCHDRS` | `lteSCFDMADemodulate` | `lteULChannelEstimate` | `lteULFrameOffsetNPUSCH` | `lteULPerfectChannelEstimate`

# lteULChannelEstimatePUCCH1

PUCCH format 1 uplink channel estimation

## Syntax

```
[hest,noiseest] = lteULChannelEstimatePUCCH1(ue,chs,rxgrid)
[hest,noiseest] = lteULChannelEstimatePUCCH1(ue,chs,cec,rxgrid)
[hest,noiseest] = lteULChannelEstimatePUCCH1(ue,chs,cec,rxgrid,refgrid)
[hest,noiseest] = lteULChannelEstimatePUCCH1(ue,chs,rxgrid,refgrid)
```

## Description

`[hest,noiseest] = lteULChannelEstimatePUCCH1(ue,chs,rxgrid)` returns an estimate for the channel by averaging the least squares estimates of the reference symbols across time and copying these across the allocated resource elements within the time frequency grid. `lteULChannelEstimatePUCCH1` returns `hest`, the estimated channel between each transmit and receive antenna and `noiseest`, an estimate of the noise power spectral density.

`[hest,noiseest] = lteULChannelEstimatePUCCH1(ue,chs,cec,rxgrid)` returns the estimated channel using the method and parameters defined by the user in the channel estimator configuration structure, `cec`.

`[hest,noiseest] = lteULChannelEstimatePUCCH1(ue,chs,cec,rxgrid,refgrid)` returns the estimated channel using the method and parameters defined by the channel estimation configuration structure and the additional information about the transmitted symbols found in `refgrid`. The `rxgrid` and `refgrid` inputs must have the same dimensions. For `cec.InterpType = 'None'`, values in `refgrid` are treated as reference symbols and the resulting `hest` will contain non-zero values in their locations.

`[hest,noiseest] = lteULChannelEstimatePUCCH1(ue,chs,rxgrid,refgrid)` returns the estimated channel using the estimation method as described in TS 36.101, Annex F4 [1]. The method described utilizes extra channel information obtained through information of the transmitted symbols found in `refgrid`. This additional information allows for an improved estimate of the channel and is required for accurate EVM measurements. `rxgrid` and `refgrid` must only contain a whole subframe worth of SC-FDMA symbols.

## Examples

### Estimate Channel Characteristics for PUCCH Format 1

Use the lteULChannelEstimatePUCCH1 function to estimate channel characteristics for PUCCH Format 1

Initialize a UE configuration structure, PUCCH settings, and create a resource grid.

```
ue = struct('NULRB',6,'NCellID',0,'NSubframe',0,'Hopping','Off');
ue.CyclicPrefixUL = 'Normal';
ue.NTxAnts = 1;
pucch1.ResourceIdx  = 0;
```

```
pucch1.DeltaShift   = 1;
pucch1.CyclicShifts = 0;
reGrid = lteULResourceGrid(ue);
reGrid(ltePUCCH1DRSIndices(ue,pucch1)) = ltePUCCH1DRS(ue,pucch1);
```

For the purpose of this example, we skip SC-FDMA modulation, channel and SC-FDMA demodulation stages of the system model and use `reGrid` as the received resource grid. Initialize the channel estimation configuration structure and perform channel estimation operation on `reGrid`.

```
cec = struct('FreqWindow',12,'TimeWindow',1,'InterpType','Cubic');
hest = lteULChannelEstimatePUCCH1(ue,pucch1,cec,reGrid);
```

## Input Arguments

### ue — UE-specific configuration settings

structure

UE-specific configuration settings, specified as a structure that can contain the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NULRB** | Required | Scalar integer from 6 to 110 | Number of uplink resource blocks. ($N_{RB}^{UL}$) |
| **NCellID** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **NSubframe** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **CyclicPrefixUL** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **NTxAnts** | Optional | 1 (default), 2, 4 | Number of transmission antennas. |
| **Hopping** | Optional | `'Off'` (default), `'Group'` | Frequency hopping method. |
| **NPUCCHID** | Optional | Integer from 0 to 503 | PUCCH virtual cell identity. If this field is not present, `NCellID` is used as the identity. |

Data Types: `struct`

### chs — PUCCH settings

structure

PUCCH channel settings, specified as a structure that can contain the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`ResourceIdx`** | Optional | 0 (default), integer from 0 to 2047 or vector of integers. | PUCCH resource indices, specified as an integer or a vector of integers. Values range from 0 to 2047. These indices determine the physical resource blocks, cyclic shift and orthogonal cover used for transmission. ($n^{(1)}_{PUCCH}$). Define one index for each transmission antenna. |
| **`ResourceSize`** | Optional | 0 (default), integer from 0 to 98. | Size of resource allocated to PUCCH format 2 ($N^{(2)}_{RB}$) |
| **`DeltaShift`** | Optional | 1 (default), 2, 3 | Delta shift, specified as 1, 2, or 3. ($\Delta_{\text{shift}}$) |
| **`DeltaOffset`** | Optional | 0 (default), 1, 2 | ($\Delta_{\text{offset}}$). Warning: The use of this parameter field is not advised. It applies only to 3GPP releases preceding v8.5.0. This parameter will be removed in a future release. |
| **`CyclicShifts`** | Optional | 0 (default), integer from 0 to 7 | Number of cyclic shifts used for format 1 in resource blocks (RBs) with a mixture of format 1 and format 2 PUCCH, specified as an integer from 0 to 7. ($N^{(1)}_{cs}$) |

Data Types: `struct`

**`rxgrid` — Received resource element grid**
3-D array

Received resource element grid, specified as an $N_{\text{SC}}$-by-$N_{\text{Sym}}$-by-$N_{\text{R}}$ array of complex symbols.

- $N_{\text{SC}}$ is the number of subcarriers
- $N_{\text{Sym}} = N_{\text{SF}} \times N_{\text{SymPerSF}}$
  - $N_{\text{SF}}$ is the total number of subframes. If $N_{\text{SF}}$ is greater than one, the correct region is extracted from the returned `hest` array. The location of the estimated subframe within `hest` is specified using the parameter field `cec.Window`.
  - $N_{\text{SymPerSF}}$ is the number of SC-FDMA symbols per subframe.
    - For normal cyclic prefix, each subframe contains 14 SC-FDMA symbols.
    - For extended cyclic prefix, each subframe contains 12 SC-FDMA symbols.
- $N_{\text{R}}$ is the number of receive antennas

Data Types: `double`
Complex Number Support: Yes

**cec — Channel estimator configuration**
structure

Channel estimator configuration, specified as a structure with these fields.

<table>
<tr><th>Parameter Field</th><th>Required or Optional</th><th>Values</th><th>Description</th></tr>
<tr><td>FreqWindow</td><td>Optional</td><td>Odd scalar integer or a multiple of 12</td><td>Size of window used to average over frequency, in resource elements (REs), specified as a scalar integer.</td></tr>
<tr><td>TimeWindow</td><td>Optional</td><td>Odd scalar integer</td><td>Size of window used to average over time, in resource elements (REs), specified as a scalar integer.</td></tr>
<tr><td>InterpType</td><td>Optional</td><td><code>'nearest'</code>, <code>'linear'</code>, <code>'natural'</code>, <code>'cubic'</code>, <code>'v4'</code>, <code>'none'</code><br><br>See footnote.</td><td>Type of 2-D interpolation used during interpolation. For details, see <code>griddata</code>. Supported choices are shown in the following table.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td><code>'nearest'</code></td><td>Nearest neighbor interpolation</td></tr>
<tr><td><code>'linear'</code></td><td>Linear interpolation</td></tr>
<tr><td><code>'natural'</code></td><td>Natural neighbor interpolation</td></tr>
<tr><td><code>'cubic'</code></td><td>Cubic interpolation</td></tr>
<tr><td><code>'v4'</code></td><td>MATLAB 4 <code>griddata</code> method</td></tr>
<tr><td><code>'none'</code></td><td>Disables interpolation</td></tr>
</table></td></tr>
<tr><td>PilotAverage</td><td>Optional</td><td><code>'UserDefined'</code>(default), <code>'TestEVM'</code><br><br>See footnote.</td><td>Type of pilot averaging</td></tr>
<tr><td colspan="4">The following parameter is required only if <code>rxgrid</code> contains more than one subframe. See footnote.</td></tr>
<tr><td>Window</td><td>Optional</td><td><code>'Left'</code>, <code>'Right'</code>, <code>'Centred'</code>, <code>'Centered'</code></td><td>If more than one subframe is input this parameter is required to indicate the position of the subframe from rxgrid and refgrid containing the desired channel estimate. Only channel estimates for this subframe will be returned. For the <code>'Centred'</code> and <code>'Centered'</code> settings, the window size must be odd.</td></tr>
</table>

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **1** | For `cec.InterpType = 'none'`, no interpolation is performed between pilot symbols and no virtual pilots are created. `hest` will contain channel estimates in the locations of transmitted reference symbols for each received antenna and all other elements of `hest` are zero. The averaging of pilot symbols estimates described by `cec.TimeWindow` and `cec.FreqWindow` are still performed. | | |
| **2** | The `'UserDefined'` pilot averaging uses a rectangular kernel of size `cec.FreqWindow`-by-`cec.TimeWindow` and performs a 2-D filtering operation upon the pilots. Pilots near the edge of the resource grid are averaged less as they have no neighbors outside of the grid. For `cec.FreqWindow` = 12×$X$ (i.e. any multiple of 12) and `cec.TimeWindow` = 1 the estimator enters a special case where an averaging window of (12×$X$)-in-frequency is used to average the pilot estimates; the averaging is always applied across (12×$X$) subcarriers, even at the upper and lower band edges; therefore the first (6×$X$) symbols at the upper and lower band edge have the same channel estimate. This operation ensures that averaging is always done on 12 (or a multiple of 12) symbols. This provides the appropriate despreading operation required for the case multi-antenna transmission where the DM-RS signals associated with each antenna occupy the same time/frequency locations but use different orthogonal cover codes to allow them to be differentiated at the receiver. The `'TestEVM'` pilot averaging ignores other structure fields in `cec`, and follows the method described in TS 36.101, Annex F for the purposes of transmitter EVM testing. | | |
| **3** | When `rxgrid` contains more than one subframe, `cec.Window` provides control of the location of the subframe for which channel estimation is performed. This allows channel estimation for the subframe of interest to be aided by the presence of pilot symbols occupying the same resource block in subframes before and/or after that subframe. For example, if `rxgrid` contains five subframes, `'Left'` estimates the last first subframe in `rxgrid`, `'Centred'`/`'Centered'` estimates the third (middle) subframe, and `'Right'` estimates the last subframe. The parameter `ue.NSubframe` corresponds to the chosen subframe. So, with three subframes and `cec.Window = 'Right'`, `rxgrid` corresponds to subframes (`ue.NSubframe-2`, `ue.NSubframe-1`, `ue.NSubframe`). The `hest` output will be the same size as `rxgrid` and will correspond to the same subframe numbers. All locations other than the estimated subframe will contain zeros. | | |

Data Types: `struct`

**`refgrid` — Reference array of known transmitted data symbols in their correct locations**
3-D numeric array

Reference array of known transmitted data symbols in their correct locations, specified as an $N_{SC}$-by-$N_{Sym}$-by-$N_T$ array of complex symbols. All other locations, such as DM-RS Symbols and unknown data symbol locations, must be represented by a `NaN`. The first two dimensions of `rxgrid` and `refgrid` must be the same.

- $N_{SC}$ is the number of subcarriers.
- $N_{Sym} = N_{SF} \times N_{SymPerSF}$
  - $N_{SF}$ is the total number of subframes. If $N_{SF}$ is greater than one, the correct region is extracted from the returned `hest` array. The location of the estimated subframe within `hest` is specified using the parameter field `cec.Window`.
  - $N_{SymPerSF}$ is the number of SC-FDMA symbols per subframe.
    - For normal cyclic prefix, each subframe contains 14 SC-FDMA symbols.
    - For extended cyclic prefix, each subframe contains 12 SC-FDMA symbols.
- $N_T$ is the number of transmit antennas, `ue.NTxAnts`

For `cec.InterpType = 'None'`, values in `refgrid` are treated as reference symbols and the resulting `hest` contains non-zero values in their locations. A typical use for `refgrid` is to provide values of the SRS transmitted at some point during the time span of `rxgrid`. The SRS values can be used to enhance the channel estimation.

Data Types: `double`
Complex Number Support: Yes

## Output Arguments

### `hest` — Channel estimate between each transmit and receive antenna

3-D array

Channel estimate between each transmit and receive antenna, returned as a $N_{SC}$-by-$N_{Sym}$-by-$N_R$. array of complex symbols. $N_{SC}$ is the total number of subcarriers, $N_{Sym}$ is the number of SC-FDMA symbols, and $N_R$ is the number of receive antennas.

### `noiseest` — Noise estimate

numeric scalar

Noise estimate, returned as a numeric scalar. This output is the power spectral density of the noise present on the estimated channel response coefficients.

## Algorithms

The channel estimation algorithm functions as described in the following steps.

1. Extract the PUCCH format 1 demodulation reference signals (DM-RS), or pilot symbols, for a transmit-receive antenna pair from the allocated physical resource blocks within the received subframe.
2. Average the least-squares estimates to reduce any unwanted noise from the pilot symbols.
3. Using the cleaned pilot symbol estimates, interpolate to obtain an estimate of the channel for the allocated subframe slot passed into the function.

#### Least-Squares Estimation

The least-squares estimates of the reference signals are obtained by dividing the received pilot symbols by their expected value. The least-squares estimates are affected by any system noise. This noise needs to be removed or reduced to achieve a reasonable estimation of the channel at pilot symbol locations.

#### Noise Reduction and Interpolation

To minimize the effects of noise on the pilot symbol estimates, the least-squares estimates are averaged. This simple method produces a substantial reduction in the level of noise found on the pilot symbols. The pilot symbol averaging method uses an averaging window defined by the user. The averaging window size is measured in resource elements; any pilot symbols located within the window are used to average the value of the pilot symbol found at the center of the window.

Then, the averaged pilot symbol estimates are used to perform a 2-D interpolation across the slot of the subframe that was allocated to the PUCCH format 1 data. The location of pilot symbols within the subframe is not ideally suited to interpolation. To account for this positioning, virtual pilots are

created and placed out with the area of the current subframe. This placement allows complete and accurate interpolation to be performed.

## Version History

**Introduced in R2013b**

## References

[1] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteSCFDMADemodulate` | `lteULFrameOffsetPUCCH1` | `lteULChannelEstimate` | `lteULPerfectChannelEstimate` | `griddata`

# lteULChannelEstimatePUCCH2

PUCCH format 2 uplink channel estimation

## Syntax

```
[hest,noiseest] = lteULChannelEstimatePUCCH2(ue,chs,rxgrid,rxack2)
[hest,noiseest] = lteULChannelEstimatePUCCH2(ue,chs,cec,rxgrid,rxack2)
[hest,noiseest] = lteULChannelEstimatePUCCH2(ue,chs,cec,rxgrid,rxack2,
refgrid)
[hest,noiseest] = lteULChannelEstimatePUCCH2(ue,chs,rxgrid,rxack2,refgrid)
```

## Description

`[hest,noiseest] = lteULChannelEstimatePUCCH2(ue,chs,rxgrid,rxack2)` returns an estimate for the channel by averaging the least squares estimates of the reference symbols across time and copying these across the allocated resource elements within the time frequency grid. It returns `hest`, the estimated channel between each transmit and receive antenna and `noiseest`, an estimate of the noise power spectral density.

`[hest,noiseest] = lteULChannelEstimatePUCCH2(ue,chs,cec,rxgrid,rxack2)` returns the estimated channel using the method and parameters defined by the user in the channel estimator configuration structure, `cec`.

`[hest,noiseest] = lteULChannelEstimatePUCCH2(ue,chs,cec,rxgrid,rxack2, refgrid)` returns the estimated channel using the method and parameters defined by the channel estimation configuration structure (`cec`), and the additional information about the transmitted symbols found in `refgrid`. The `rxgrid` and `refgrid` inputs must have the same dimensions. For `cec.InterpType = 'None'`, values in `refgrid` are treated as reference symbols and the resulting `hest` contains non-zero values in their locations.

`[hest,noiseest] = lteULChannelEstimatePUCCH2(ue,chs,rxgrid,rxack2,refgrid)` returns the estimated channel using the estimation method, as described in TS 36.101, Annex F4 [1]. The method described utilizes extra channel information obtained through information of the transmitted symbols found in `refgrid`. This additional information allows for an improved estimate of the channel and is required for accurate EVM measurements.

## Examples

### Estimate Channel Characteristics for PUCCH Format 2

Use the lteULChannelEstimatePUCCH2 function to estimate channel characteristics for PUCCH Format 2

Initialize a UE configuration structure, PUCCH settings, and create a resource grid. For the purpose of this example, we bypass the SC-FDMA modulation, channel and SC-FDMA demodulation stages of the system model and copy the `txGrid` to an `rxGrid`.

```
ue = struct('NULRB',6,'NCellID',0,'NSubframe',0,'Hopping','Off');
ue.CyclicPrefixUL = 'Normal';
```

```
ue.NTxAnts = 1;
pucch2.ResourceIdx = 0;
pucch2.ResourceSize = 0;
pucch2.CyclicShifts = 0;
txGrid = lteULResourceGrid(ue);
txAck = [1;1];
drsIndices = ltePUCCH2DRSIndices(ue,pucch2);

txGrid(drsIndices) = ltePUCCH2DRS(ue,pucch2,txAck);
rxGrid = txGrid;
```

The channel estimator uses the PUCCH Format 2 DRS to estimate the channel, so decode the hybrid ARQ indicators from the PUCCH Format 2 DM-RS. Initialize the channel estimation configuration structure and perform channel estimation operation on `rxGrid`.

```
rxAck = ltePUCCH2DRSDecode(ue,pucch2,length(txAck),rxGrid(drsIndices));
cec = struct('FreqWindow',12,'TimeWindow',1,'InterpType','cubic');
hest = lteULChannelEstimatePUCCH2(ue,pucch2,cec,rxGrid,rxAck);
```

## Input Arguments

### ue — UE-specific configuration settings

structure

UE-specific configuration settings, specified as a structure that can contain the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NULRB`** | Required | Scalar integer from 6 to 110 | Number of uplink resource blocks. ($N_{RB}^{UL}$) |
| **`NCellID`** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **`NSubframe`** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **`CyclicPrefixUL`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **`NTxAnts`** | Optional | 1 (default), 2, 4 | Number of transmission antennas. |
| **`Hopping`** | Optional | `'Off'` (default), `'Group'` | Frequency hopping method. |
| **`NPUCCHID`** | Optional | Integer from 0 to 503 | PUCCH virtual cell identity. If this field is not present, `NCellID` is used as the identity. |

Data Types: `struct`

### chs — PUCCH channel settings

structure

PUCCH channel settings, specified as a structure that can contain the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`ResourceIdx`** | Optional | 0 (default), integer from 0 to 1185 or vector of integers. | PUCCH resource indices which determine the physical resource blocks, cyclic shift, and orthogonal cover used for transmission. ($n^{(2)}_{PUCCH}$). Define one index for each transmission antenna. |
| **`ResourceSize`** | Optional | 0 (default), integer from 0 to 98. | Size of resource allocated to PUCCH format 2 ($N^{(2)}_{RB}$) |
| **`CyclicShifts`** | Optional | 0 (default), integer from 0 to 7 | Number of cyclic shifts used for format 1 in resource blocks (RBs) with a mixture of format 1 and format 2 PUCCH, specified as an integer from 0 to 7. ($N^{(1)}_{cs}$) |

Data Types: `struct`

**`rxgrid` — Received resource element grid**
3-D array

Received resource element grid, specified as an $N_{SC}$-by-$N_{Sym}$-by-$N_R$ array of complex symbols.

- $N_{SC}$ is the number of subcarriers
- $N_{Sym} = N_{SF} \times N_{SymPerSF}$
  - $N_{SF}$ is the total number of subframes. If $N_{SF}$ is greater than one, the correct region is extracted from the returned `hest` array. The location of the estimated subframe within `hest` is specified using the parameter field `cec.Window`.
  - $N_{SymPerSF}$ is the number of SC-FDMA symbols per subframe.
    - For normal cyclic prefix, each subframe contains 14 SC-FDMA symbols.
    - For extended cyclic prefix, each subframe contains 12 SC-FDMA symbols.
- $N_R$ is the number of receive antennas

Data Types: `double`
Complex Number Support: Yes

**`rxack2` — Hybrid ARQ indicators**
logical value

Hybrid ARQ indicators, specified as a row vector of either 1 or 2 indicators, decoded from the PUCCH Format 2 DRS. This is required as the channel estimator uses the PUCCH Format 2 DM-RS to estimate the channel. `rxack2` can be obtained for example by using the `lteULFrameOffsetPUCCH2` function.

Data Types: `logical`

**`cec` — Channel estimator configuration**
structure

Channel estimator configuration, specified as a structure with these fields.

<table>
<tr><th>Parameter Field</th><th>Required or Optional</th><th>Values</th><th>Description</th></tr>
<tr><td>FreqWindow</td><td>Optional</td><td>Odd scalar integer or a multiple of 12</td><td>Size of window used to average over frequency, in resource elements (REs), specified as a scalar integer.</td></tr>
<tr><td>TimeWindow</td><td>Optional</td><td>Odd scalar integer</td><td>Size of window used to average over time, in resource elements (REs), specified as a scalar integer.</td></tr>
<tr><td>InterpType</td><td>Optional</td><td>'nearest', 'linear', 'natural', 'cubic', 'v4', 'none'<br><br>See footnote.</td><td>Type of 2-D interpolation used during interpolation. For details, see griddata. Supported choices are shown in the following table.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>'nearest'</td><td>Nearest neighbor interpolation</td></tr>
<tr><td>'linear'</td><td>Linear interpolation</td></tr>
<tr><td>'natural'</td><td>Natural neighbor interpolation</td></tr>
<tr><td>'cubic'</td><td>Cubic interpolation</td></tr>
<tr><td>'v4'</td><td>MATLAB 4 griddata method</td></tr>
<tr><td>'none'</td><td>Disables interpolation</td></tr>
</table></td></tr>
<tr><td>PilotAverage</td><td>Optional</td><td>'UserDefined' (default), 'TestEVM'<br><br>See footnote.</td><td>Type of pilot averaging</td></tr>
<tr><td colspan="4">The following parameter is required only if rxgrid contains more than one subframe. See footnote.</td></tr>
<tr><td>Window</td><td>Optional</td><td>'Left', 'Right', 'Centred', 'Centered'</td><td>If more than one subframe is input this parameter is required to indicate the position of the subframe from rxgrid and refgrid containing the desired channel estimate. Only channel estimates for this subframe will be returned. For the 'Centred' and 'Centered' settings, the window size must be odd.</td></tr>
</table>

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **1** | For `cec.InterpType = 'none'`, no interpolation is performed between pilot symbols and no virtual pilots are created. `hest` will contain channel estimates in the locations of transmitted reference symbols for each received antenna and all other elements of `hest` are zero. The averaging of pilot symbols estimates described by `cec.TimeWindow` and `cec.FreqWindow` are still performed. | | |
| **2** | The `'UserDefined'` pilot averaging uses a rectangular kernel of size `cec.FreqWindow`-by-`cec.TimeWindow` and performs a 2-D filtering operation upon the pilots. Pilots near the edge of the resource grid are averaged less as they have no neighbors outside of the grid. For `cec.FreqWindow` = 12×$X$ (i.e. any multiple of 12) and `cec.TimeWindow` = 1 the estimator enters a special case where an averaging window of (12×$X$)-in-frequency is used to average the pilot estimates; the averaging is always applied across (12×$X$) subcarriers, even at the upper and lower band edges; therefore the first (6×$X$) symbols at the upper and lower band edge have the same channel estimate. This operation ensures that averaging is always done on 12 (or a multiple of 12) symbols. This provides the appropriate despreading operation required for the case multi-antenna transmission where the DM-RS signals associated with each antenna occupy the same time/frequency locations but use different orthogonal cover codes to allow them to be differentiated at the receiver. The `'TestEVM'` pilot averaging ignores other structure fields in `cec`, and follows the method described in TS 36.101, Annex F for the purposes of transmitter EVM testing. | | |
| **3** | When `rxgrid` contains more than one subframe, `cec.Window` provides control of the location of the subframe for which channel estimation is performed. This allows channel estimation for the subframe of interest to be aided by the presence of pilot symbols occupying the same resource block in subframes before and/or after that subframe. For example, if `rxgrid` contains five subframes, `'Left'` estimates the last first subframe in `rxgrid`, `'Centred'`/`'Centered'` estimates the third (middle) subframe, and `'Right'` estimates the last subframe. The parameter `ue.NSubframe` corresponds to the chosen subframe. So, with three subframes and `cec.Window = 'Right'`, `rxgrid` corresponds to subframes (`ue.NSubframe-2`, `ue.NSubframe-1`, `ue.NSubframe`). The `hest` output will be the same size as `rxgrid` and will correspond to the same subframe numbers. All locations other than the estimated subframe will contain zeros. | | |

Data Types: `struct`

**`refgrid` — Reference array of known transmitted data symbols in their correct locations**
3-D numeric array

Reference array of known transmitted data symbols in their correct locations, specified as an $N_{SC}$-by-$N_{Sym}$-by-$N_T$ array of complex symbols. All other locations, such as DM-RS Symbols and unknown data symbol locations, must be represented by a `NaN`. The first two dimensions of `rxgrid` and `refgrid` must be the same.

- $N_{SC}$ is the number of subcarriers.
- $N_{Sym} = N_{SF} \times N_{SymPerSF}$
  - $N_{SF}$ is the total number of subframes. If $N_{SF}$ is greater than one, the correct region is extracted from the returned `hest` array. The location of the estimated subframe within `hest` is specified using the parameter field `cec.Window`.
  - $N_{SymPerSF}$ is the number of SC-FDMA symbols per subframe.
    - For normal cyclic prefix, each subframe contains 14 SC-FDMA symbols.
    - For extended cyclic prefix, each subframe contains 12 SC-FDMA symbols.
- $N_T$ is the number of transmit antennas, `ue.NTxAnts`

For `cec.InterpType = 'None'`, values in `refgrid` are treated as reference symbols and the resulting `hest` contains non-zero values in their locations. A typical use for `refgrid` is to provide values of the SRS transmitted at some point during the time span of `rxgrid`. The SRS values can be used to enhance the channel estimation.

Data Types: `double`
Complex Number Support: Yes

## Output Arguments

### `hest` — Channel estimate between each transmit and receive antenna

3-D array

Channel estimate between each transmit and receive antenna, returned as an $N_{SC}$-by-$N_{Sym}$-by-$N_R$ array of complex symbols. $N_{SC}$ is the total number of subcarriers, $N_{Sym}$ is the number of SC-FDMA symbols, and $N_R$ is the number of receive antennas.

### `noiseest` — Noise estimate

numeric scalar

Noise estimate, returned as a numeric scalar. This output is the power spectral density of the noise present on the estimated channel response coefficients.

## Algorithms

The channel estimation algorithm functions as described in the following steps.

1. Extract the PUCCH format 2 demodulation reference signals (DM-RS), or pilot symbols, for a transmit-receive antenna pair from the allocated physical resource blocks within the received subframe.
2. Average the least-squares estimates to reduce any unwanted noise from the pilot symbols.
3. Using the cleaned pilot symbol estimates, interpolate to obtain an estimate of the channel for the allocated subframe slot.

#### Least-Squares Estimation

The least-squares estimates of the reference signals are obtained by dividing the received pilot symbols by their expected value. The least-squares estimates are affected by any system noise. This noise needs to be removed or reduced to achieve a reasonable estimation of the channel at pilot symbol locations.

#### Noise Reduction and Interpolation

To minimize the effects of noise on the pilot symbol estimates, the least-squares estimates are averaged. This simple method produces a substantial reduction in the level of noise found on the pilot symbols. The pilot symbol averaging method uses an averaging window defined by the user. The averaging window size is measured in resource elements; any pilot symbols located within the window are used to average the value of the pilot symbol found at the center of the window.

Then, the averaged pilot symbol estimates are used to perform a 2-D interpolation across the slot of the subframe that was allocated to the PUCCH format 2 data. The location of pilot symbols within the subframe is not ideally suited to interpolation. To account for this positioning, virtual pilots are

created and placed out with the area of the current subframe. This placement allows complete and accurate interpolation to be performed.

# Version History

**Introduced in R2013b**

## References

[1] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteSCFDMADemodulate` | `lteULFrameOffsetPUCCH2` | `lteULChannelEstimate` | `lteULPerfectChannelEstimate` | `griddata`

# lteULChannelEstimatePUCCH3

PUCCH format 3 uplink channel estimation

## Syntax

```
[hest,noiseest] = lteULChannelEstimatePUCCH3(ue,chs,rxgrid)
[hest,noiseest] = lteULChannelEstimatePUCCH3(ue,chs,cec,rxgrid)
[hest,noiseest] = lteULChannelEstimatePUCCH3(ue,chs,cec,rxgrid,refgrid)
[hest,noiseest] = lteULChannelEstimatePUCCH3(ue,chs,rxgrid,refgrid)
```

## Description

`[hest,noiseest] = lteULChannelEstimatePUCCH3(ue,chs,rxgrid)` returns an estimate for the channel by averaging the least squares estimates of the reference symbols across time and copying these across the allocated resource elements within the time frequency grid. It returns `hest`, the estimated channel between each transmit and receive antenna and `noiseest`, an estimate of the noise power spectral density.

`[hest,noiseest] = lteULChannelEstimatePUCCH3(ue,chs,cec,rxgrid)` returns the estimated channel using the method and parameters defined by the user in the channel estimator configuration structure, `cec`.

`[hest,noiseest] = lteULChannelEstimatePUCCH3(ue,chs,cec,rxgrid,refgrid)` returns the estimated channel using the method and parameters defined by the channel estimation configuration structure and the additional information about the transmitted symbols found in `refgrid`. `rxgrid` and `refgrid` must have the same dimensions. For `cec.InterpType = 'None'`, values in `refgrid` are treated as reference symbols and the resulting `hest` contains non-zero values in their locations.

`[hest,noiseest] = lteULChannelEstimatePUCCH3(ue,chs,rxgrid,refgrid)` returns the estimated channel using the estimation method, as described in TS 36.101, Annex F4 [1]. The method described utilizes extra channel information obtained through information of the transmitted symbols found in `refgrid`. This additional information allows for an improved estimate of the channel and is required for accurate EVM measurements. `rxgrid` and `refgrid` must have the same dimensions. `rxgrid` and `refgrid` must only contain a whole subframe worth of SC-FDMA symbols.

## Examples

### Estimate Channel Characteristics for PUCCH Format 3

Use the lteULChannelEstimatePUCCH3 function to estimate channel characteristics for PUCCH Format 3

Initialize a UE configuration structure, PUCCH settings, and create a resource grid. For the purpose of this example, we bypass the SC-FDMA modulation, channel and SC-FDMA demodulation stages of the system model and copy the `txGrid` to an `rxGrid`.

```
ue = struct('NULRB',6,'NCellID',0,'NSubframe',0,'Hopping','Off');
ue.CyclicPrefixUL = 'Normal';
```

```
ue.NTxAnts = 1;
pucch3 = struct('ResourceIdx',0);
txGrid = lteULResourceGrid(ue);
txGrid(ltePUCCH3DRSIndices(ue,pucch3)) = ltePUCCH3DRS(ue,pucch3);
rxGrid = txGrid;
```

Initialize the channel estimation configuration structure and perform channel estimation operation on `rxGrid`.

```
cec = struct('FreqWindow',12,'TimeWindow',1,'InterpType','Cubic');
hest = lteULChannelEstimatePUCCH3(ue,pucch3,cec,rxGrid);
```

## Input Arguments

### ue — UE-specific cell-wide settings

structure

UE-specific cell-wide settings, specified as a structure with the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NULRB`** | Required | Scalar integer from 6 to 110 | Number of uplink resource blocks. ($N_{RB}^{UL}$) |
| **`NCellID`** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **`NSubframe`** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **`CyclicPrefixUL`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **`NTxAnts`** | Optional | 1 (default), 2, 4 | Number of transmission antennas. |
| **`Hopping`** | Optional | `'Off'` (default), `'Group'` | Frequency hopping method. |
| **`Shortened`** | Optional | 0 (default), 1 | Option to shorten the subframe by omitting the last symbol, specified as 0 or 1. If `1`, the last symbol of the subframe is not used. For subframes with possible SRS transmission, set `Shortened` to 1 to maintain a standard compliant configuration. |
| **`NPUCCHID`** | Optional | Integer from 0 to 503 | PUCCH virtual cell identity. If this field is not present, `NCellID` is used as the identity. |

Data Types: `struct`

### chs — PUCCH channel settings

structure

PUCCH channel settings, specified as a structure with the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`ResourceIdx`** | Optional | 0 (default), integer from 0 to 549, or vector of integers. | PUCCH resource indices which determine the physical resource blocks, cyclic shift, and orthogonal cover used for transmission ($n^{(3)}_{PUCCH}$). Define one index for each transmission antenna. |

Data Types: `struct`

### `rxgrid` — Received resource element grid

3-D array

Received resource element grid, specified as an $N_{SC}$-by-$N_{Sym}$-by-$N_R$ array of complex symbols.

- $N_{SC}$ is the number of subcarriers
- $N_{Sym} = N_{SF} \times N_{SymPerSF}$
  - $N_{SF}$ is the total number of subframes. If $N_{SF}$ is greater than one, the correct region is extracted from the returned `hest` array. The location of the estimated subframe within `hest` is specified using the parameter field `cec.Window`.
  - $N_{SymPerSF}$ is the number of SC-FDMA symbols per subframe.
    - For normal cyclic prefix, each subframe contains 14 SC-FDMA symbols.
    - For extended cyclic prefix, each subframe contains 12 SC-FDMA symbols.
- $N_R$ is the number of receive antennas

Data Types: `double`
Complex Number Support: Yes

### `cec` — Channel estimator configuration

structure

Channel estimator configuration, specified as a structure that can contain the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`FreqWindow`** | Optional | Odd scalar integer or a multiple of 12 | Size of window used to average over frequency, in resource elements (REs), specified as a scalar integer. |
| **`TimeWindow`** | Optional | Odd scalar integer | Size of window used to average over time, in resource elements (REs), specified as a scalar integer. |

<table>
<tr><th>Parameter Field</th><th>Required or Optional</th><th>Values</th><th>Description</th></tr>
<tr><td>InterpType</td><td>Optional</td><td>'nearest', 'linear', 'natural', 'cubic', 'v4', 'none'<br><br>See footnote.</td><td>Type of 2-D interpolation used during interpolation. For details, see griddata. Supported choices are shown in the following table.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>'nearest'</td><td>Nearest neighbor interpolation</td></tr>
<tr><td>'linear'</td><td>Linear interpolation</td></tr>
<tr><td>'natural'</td><td>Natural neighbor interpolation</td></tr>
<tr><td>'cubic'</td><td>Cubic interpolation</td></tr>
<tr><td>'v4'</td><td>MATLAB 4 griddata method</td></tr>
<tr><td>'none'</td><td>Disables interpolation</td></tr>
</table></td></tr>
<tr><td>PilotAverage</td><td>Optional</td><td>'UserDefined'(default), 'TestEVM'<br><br>See footnote.</td><td>Type of pilot averaging</td></tr>
<tr><td colspan="4">The following parameter is required only if rxgrid contains more than one subframe. See footnote.</td></tr>
<tr><td>Window</td><td>Optional</td><td>'Left', 'Right', 'Centred', 'Centered'</td><td>If more than one subframe is input this parameter is required to indicate the position of the subframe from rxgrid and refgrid containing the desired channel estimate. Only channel estimates for this subframe will be returned. For the 'Centred' and 'Centered' settings, the window size must be odd.</td></tr>
</table>

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **1** | For `cec.InterpType = 'none'`, no interpolation is performed between pilot symbols and no virtual pilots are created. `hest` will contain channel estimates in the locations of transmitted reference symbols for each received antenna and all other elements of `hest` are zero. The averaging of pilot symbols estimates described by `cec.TimeWindow` and `cec.FreqWindow` are still performed. | | |
| **2** | The `'UserDefined'` pilot averaging uses a rectangular kernel of size `cec.FreqWindow`-by-`cec.TimeWindow` and performs a 2-D filtering operation upon the pilots. Pilots near the edge of the resource grid are averaged less as they have no neighbors outside of the grid. For `cec.FreqWindow` = 12×$X$ (i.e. any multiple of 12) and `cec.TimeWindow` = 1 the estimator enters a special case where an averaging window of (12×$X$)-in-frequency is used to average the pilot estimates; the averaging is always applied across (12×$X$) subcarriers, even at the upper and lower band edges; therefore the first (6×$X$) symbols at the upper and lower band edge have the same channel estimate. This operation ensures that averaging is always done on 12 (or a multiple of 12) symbols. This provides the appropriate despreading operation required for the case multi-antenna transmission where the DM-RS signals associated with each antenna occupy the same time/frequency locations but use different orthogonal cover codes to allow them to be differentiated at the receiver. The `'TestEVM'` pilot averaging ignores other structure fields in `cec`, and follows the method described in TS 36.101, Annex F for the purposes of transmitter EVM testing. | | |
| **3** | When `rxgrid` contains more than one subframe, `cec.Window` provides control of the location of the subframe for which channel estimation is performed. This allows channel estimation for the subframe of interest to be aided by the presence of pilot symbols occupying the same resource block in subframes before and/or after that subframe. For example, if `rxgrid` contains five subframes, `'Left'` estimates the last first subframe in `rxgrid`, `'Centred'`/`'Centered'` estimates the third (middle) subframe, and `'Right'` estimates the last subframe. The parameter `ue.NSubframe` corresponds to the chosen subframe. So, with three subframes and `cec.Window = 'Right'`, `rxgrid` corresponds to subframes (`ue.NSubframe-2`, `ue.NSubframe-1`, `ue.NSubframe`). The `hest` output will be the same size as `rxgrid` and will correspond to the same subframe numbers. All locations other than the estimated subframe will contain zeros. | | |

Data Types: `struct`

**`refgrid` — Reference array of known transmitted data symbols in their correct locations**
3-D numeric array

Reference array of known transmitted data symbols in their correct locations, specified as an $N_{SC}$-by-$N_{Sym}$-by-$N_T$ array of complex symbols. All other locations, such as DM-RS Symbols and unknown data symbol locations, must be represented by a `NaN`. The first two dimensions of `rxgrid` and `refgrid` must be the same.

- $N_{SC}$ is the number of subcarriers.
- $N_{Sym} = N_{SF} \times N_{SymPerSF}$
  - $N_{SF}$ is the total number of subframes. If $N_{SF}$ is greater than one, the correct region is extracted from the returned `hest` array. The location of the estimated subframe within `hest` is specified using the parameter field `cec.Window`.
  - $N_{SymPerSF}$ is the number of SC-FDMA symbols per subframe.
    - For normal cyclic prefix, each subframe contains 14 SC-FDMA symbols.
    - For extended cyclic prefix, each subframe contains 12 SC-FDMA symbols.
- $N_T$ is the number of transmit antennas, `ue.NTxAnts`

For `cec.InterpType = 'None'`, values in `refgrid` are treated as reference symbols and the resulting `hest` contains non-zero values in their locations. A typical use for `refgrid` is to provide values of the SRS transmitted at some point during the time span of `rxgrid`. The SRS values can be used to enhance the channel estimation.

Data Types: `double`
Complex Number Support: Yes

## Output Arguments

### `hest` — Channel estimate between each transmit and receive antenna

3-D array

Channel estimate between each transmit and receive antenna, returned as a $N_{SC}$-by-$N_{Sym}$-by-$N_R$. array of complex symbols. $N_{SC}$ is the total number of subcarriers, $N_{Sym}$ is the number of SC-FDMA symbols, and $N_R$ is the number of receive antennas.

### `noiseest` — Noise estimate

numeric scalar

Noise estimate, returned as a numeric scalar. This output is the power spectral density of the noise present on the estimated channel response coefficients.

## Algorithms

The channel estimation algorithm functions as described in the following steps.

1. Extract the PUCCH format 3 demodulation reference signals (DM-RS), or pilot symbols, for a transmit-receive antenna pair from the allocated physical resource blocks within the received subframe.
2. Average the least-squares estimates to reduce any unwanted noise from the pilot symbols.
3. Using the cleaned pilot symbol estimates, interpolate to obtain an estimate of the channel for the allocated subframe slot.

#### Least-Squares Estimation

The least-squares estimates of the reference signals are obtained by dividing the received pilot symbols by their expected value. The least-squares estimates are affected by any system noise. This noise needs to be removed or reduced to achieve a reasonable estimation of the channel at pilot symbol locations.

#### Noise Reduction and Interpolation

To minimize the effects of noise on the pilot symbol estimates, the least-squares estimates are averaged. This simple method produces a substantial reduction in the level of noise found on the pilot symbols. The pilot symbol averaging method uses an averaging window defined by the user. The averaging window size is measured in resource elements; any pilot symbols located within the window are used to average the value of the pilot symbol found at the center of the window.

Then, the averaged pilot symbol estimates are used to perform a 2-D interpolation across the slot of the subframe that was allocated to the PUCCH format 3 data. The location of pilot symbols within the subframe is not ideally suited to interpolation. To account for this positioning, virtual pilots are

created and placed out with the area of the current subframe. This placement allows complete and accurate interpolation to be performed.

## Version History

**Introduced in R2013b**

## References

[1] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteSCFDMADemodulate` | `lteULFrameOffsetPUCCH3` | `lteULChannelEstimate` | `lteULPerfectChannelEstimate` | `griddata`

# lteULDeprecode

SC-FDMA deprecoding

## Syntax

```
out = lteULDeprecode(in,nrb)
out = lteULDeprecode(in,n,resourcetype)
```

## Description

`out = lteULDeprecode(in,nrb)` performs SC-FDMA deprecoding of the complex modulation symbols `in` for PUSCH or NPUSCH configuration with a bandwidth of `nrb` resource blocks.

`out = lteULDeprecode(in,n,resourcetype)` performs SC-FDMA deprecoding of the complex modulation symbols `in` for PUSCH or NPUSCH configuration with a bandwidth of `n` resource blocks or subcarriers.

## Examples

### Deprecode Symbols After SC-FDMA Demodulation

Deprecode symbols after SC-FDMA demodulation and symbol extraction from the received resource grid.

Create an UL RMC configuration structure, resource grid, and bit stream.

```
rmc = lteRMCUL('A3-2');
[puschInd, info] = ltePUSCHIndices(rmc,rmc.PUSCH);
ueDim = lteULResourceGridSize(rmc);
bits = randi([0,1],info.G,rmc.PUSCH.NLayers);
```

Scramble bits, create modulated symbols, and perform UL precoding and resource mapping.

```
scrBits = lteULScramble(rmc,bits);
symbols = lteSymbolModulate(scrBits,rmc.PUSCH.Modulation);
precodedSymbols = lteULPrecode(symbols,rmc.NULRB);
grid = lteULResourceGrid(rmc);
grid(puschInd) = precodedSymbols;
```

Perform SC-FDMA modulation and demodulation.

```
[timeDomainSig,infoScfdma] = lteSCFDMAModulate(rmc,grid);
rxGrid = lteSCFDMADemodulate(rmc,timeDomainSig);
```

Extract PUSCH from grid and perform UL deprecoding.

```
rxPrecoded = rxGrid(puschInd);
dePrecodedSymbols = lteULDeprecode(rxPrecoded,rmc.NULRB);
```

## Input Arguments

### `in` — Complex modulation symbols

numeric matrix

Complex modulation symbols, specified as an $N_{Sym}$-by-$N_L$ matrix of complex symbols. $N_{Sym}$ is the number of symbols and $N_L$ is the number of layers.

Data Types: `double`
Complex Number Support: Yes

### `nrb` — Number of resource blocks

nonnegative integer

Number of resource blocks, specified as a nonnegative integer.

Data Types: `double`

### `n` — Number of resource blocks or subcarriers

nonnegative integer

Number of resource blocks or subcarriers, specified as a nonnegative integer.

#### Dependencies

If the `resourcetype` is `'PRB'`, then `n` is the number of resource blocks. If the `resourcetype` is `'Subcarrier'`, then `n` is the number of subcarriers.

Data Types: `double`

### `resourcetype` — Resource type

`'PRB'` | `'Subcarrier'`

Resource type, specified as `'PRB'` or `'Subcarrier'`.

Data Types: `char` | `string`

## Output Arguments

### `out` — Deprecoded PUSCH output symbols

numeric matrix

Deprecoded PUSCH output symbols, returned as an $N_{Sym}$-by-$N_L$ matrix of complex symbols. $N_{Sym}$ is the number of symbols, and $N_L$ is the number of layers.

The dimension and size of the input and output symbol matrices are the same.

# Version History

**Introduced in R2014a**

## See Also

`lteULPrecode` | `ltePUSCHDeprecode` | `lteLayerDemap` | `ltePUSCHDecode`

# lteULDescramble

PUSCH descrambling

## Syntax

```
out = lteULDescramble(ue,chs,in)
out = lteULDescramble(ue,in)
out = lteULDescramble(in,nsubframe,cellid,rnti)
```

## Description

`out = lteULDescramble(ue,chs,in)` performs PUSCH descrambling of the soft bit vector, `in`, or cell array in case of two codewords, according to UE-specific settings in the `ue` structure and UL-SCH related parameters in the `chs` structure. It performs PUSCH descrambling to undo the processing described in TS 36.212, Section 5.3.1 [1] and returns a soft bit vector or cell array of vectors, `out`. This syntax supports the descrambling of control information bits if they are present in the soft bits `in` in conjunction with information bits. The descrambling of the control information bits is done by establishing the correct locations of placeholder bits with the help of UL-SCH-related parameters present in `chs`. The descrambler skips the '*x*' placeholder bits to undo the processing defined in TS 36.212, Section 5.3.1 [1].

Multiple codewords can be parameterized by two different forms of the `chs` structure. Each codeword can be defined by separate elements of a 1-by-2 structure array, or the codeword parameters can be combined together in the fields of a single scalar, 1-by-1, structure. In the latter case, any scalar field values apply to both codewords and a scalar `NLayers` is the total number. For further details, see "UL-SCH Parameterization".

`out = lteULDescramble(ue,in)` performs PUSCH descrambling of the soft bit input, `in`, but takes only the UE-specific settings in the `ue` structure. The `in` input should contain only the scrambled data bits resulting in descrambling of transport data only. The `ue` structure must include the `NCellID`, `NSubframe`, and `RNTI` fields.

`out = lteULDescramble(in,nsubframe,cellid,rnti)` performs PUSCH descrambling of soft bits, `in`, for subframe number, `nsubframe`, cell identity, `cellid`, and specified radio network temporary identifier (RNTI), `rnti`. This syntax performs only block descrambling and expects the input, `in`, to contain only the scrambled data bits. If the `in` vector contains placeholder bits, they are not descrambled correctly because the placeholder bits are not skipped during the descrambling process. Thus, this function syntax descrambles only the transport data bits.

## Examples

### Scramble and Descramble PUSCH Vector

Perform scrambling and descrambling of vector `in`. The scrambled bits are modulated to QPSK symbols. Noise is added to these symbols, which are then demodulated to produce soft bits. These soft bits are finally descrambled.

```
in = ones(10,1);
ue = struct('NCellID',100,'NSubframe',0,'RNTI',61);
```

```
scrBits = lteULScramble(ue,in);
txSymbols = lteSymbolModulate(scrBits,'QPSK');
noise = 0.01*complex(randn(size(txSymbols)),randn(size(txSymbols)));
rxSymbols = txSymbols + noise;
softBits = lteSymbolDemodulate(rxSymbols,'QPSK','Soft');
descram = lteULDescramble(ue,softBits)
```

```
descram = 10×1

    0.7125
    0.7202
    0.7254
    0.7028
    0.6845
    0.7037
    0.7157
    0.7429
    0.7039
    0.6794
```

## Input Arguments

### `ue` — UE-specific settings

scalar structure

UE-specific settings, specified as a scalar structure that can contain the following fields.

### `NCellID` — Physical layer cell identity

scalar integer

Physical layer cell identity, specified as a scalar integer.

Data Types: `double`

### `NSubframe` — Subframe number

scalar integer

Subframe number, specified as a scalar integer.

Data Types: `double`

### `RNTI` — Radio network temporary identifier

numeric scalar

Radio network temporary identifier, 16-bit, specified as a numeric scalar.

Data Types: `double`

### `CyclicPrefixUL` — Cyclic prefix length

`'Normal'` (default) | optional | `'Extended'`

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

**`Shortened` — Shorten subframe flag**
0 (default) | optional | 1

Shorten subframe flag, specified as 0 or 1. If 1, the last symbol of the subframe is not used and rate matching is adjusted accordingly. This setting is required for subframes with possible SRS transmission.

Data Types: `logical` | `double`

Data Types: `struct`

**`chs` — UL-SCH channel-specific settings**
structure

UL-SCH channel-specific settings, specified as a structure that can contain the following fields.

**`Modulation` — Modulation scheme associated with each transport block**
`'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'`

Modulation scheme associated with each transport block, specified as `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`

Data Types: `char` | `string`

**`NLayers` — Number of transmission layers**
1 (default) | optional | 2 | 3 | 4

Number of transmission layers, total or per codeword, specified as 1, 2, 3, or 4.

Data Types: `double`

**`ORI` — Number of uncoded RI bits**
0 (default) | optional | nonnegative scalar integer

Number of uncoded RI bits, specified as a nonnegative scalar integer.

Data Types: `double`

**`OACK` — Number of uncoded HARQ-ACK bits**
0 (default) | optional | nonnegative scalar integer

Number of uncoded HARQ-ACK bits, specified as a nonnegative scalar integer.

Data Types: `double`

**`QdRI` — Number of coded RI symbols in UL-SCH**
0 (default) | optional | nonnegative scalar integer

Number of coded RI symbols in UL-SCH, specified as a nonnegative scalar integer. ($Q'\_RI$)

Data Types: `double`

**`QdACK` — Number of coded HARQ-ACK symbols in UL-SCH**
0 (default) | optional | nonnegative scalar integer

Number of coded HARQ-ACK symbols in UL-SCH, specified as a nonnegative scalar integer. ($Q'\_ACK$)

Data Types: `double`

Data Types: `struct`

**`in` — Soft bit input data**
numeric column vector | cell array of numeric column vectors

Soft bit input data, specified as a numeric column vector or cell array of numeric column vectors. This argument contains one or two vectors corresponding to the number of codewords to be scrambled.

Data Types: `double` | `cell`

**`nsubframe` — Subframe number**
scalar integer

Subframe number, specified as a scalar integer.

Data Types: `double`

**`cellid` — Physical layer cell identity**
scalar integer

Physical layer cell identity, specified as a scalar integer.

Data Types: `double`

**`rnti` — Radio network temporary identifier**
numeric scalar

Radio network temporary identifier, 16-bit, specified as a numeric scalar.

Data Types: `double`

## Output Arguments

**`out` — PUSCH descrambled output bits**
numeric column vector | cell array of numeric column vectors

PUSCH descrambled output bits, returned as a numeric column vector or cell array of numeric column vectors.

Data Types: `double`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteULScramble` | `lteSymbolDemodulate` | `ltePUSCHDecode`

# lteULFrameOffset

PUSCH DM-RS uplink subframe timing estimate

## Syntax

```
offset = lteULFrameOffset(ue,chs,waveform)
[offset,corr] = lteULFrameOffset(ue,chs,waveform)
```

## Description

`offset = lteULFrameOffset(ue,chs,waveform)` performs synchronization using PUSCH DM-RS signals for the time-domain waveform, `waveform`, given UE-specific settings, `ue`, and PUSCH configuration, `chs`.

The returned value `offset` indicates the number of samples from the start of the waveform, `waveform`, to the position in that waveform where the first subframe containing the DM-RS begins.

`offset` provides subframe timing; frame timing can be achieved by using `offset` with the subframe number, `ue.NSubframe`. This information is consistent with real-world operation, since the base station knows when, or in which subframe, to expect uplink transmissions.

`[offset,corr] = lteULFrameOffset(ue,chs,waveform)` also returns a complex matrix `corr`, which is the signal used to extract the timing offset.

## Examples

### Synchronize and SCFDMA Demodulate Delayed Transmission

Synchronization and demodulation of transmission which has been delayed by 5 samples.

Initialize waveform and insert a 5 sample delay.

```
ue = lteRMCUL('A3-2');
waveform = lteRMCULTool(ue,[1;0;0;1]);
tx = [zeros(5,1); waveform];
```

Determine `offset` and demodulate the waveform.

```
offset = lteULFrameOffset(ue,ue.PUSCH,tx)
```

```
offset = 5
```

```
rxGrid = lteSCFDMADemodulate(ue,tx(1+offset:end));
```

### View PUSCH Transmission Correlation Peaks

View the correlation peak for a delayed transmit waveform. The transmission contains PUSCH demodulation reference signal (DM-RS) symbols available for estimating the waveform timing.

### UE Configuration

Configure UE-specific settings and generate the transmit waveform.

```
ue = lteRMCUL('A3-2');
tx = lteRMCULTool(ue,[1;0;0;1]);
```

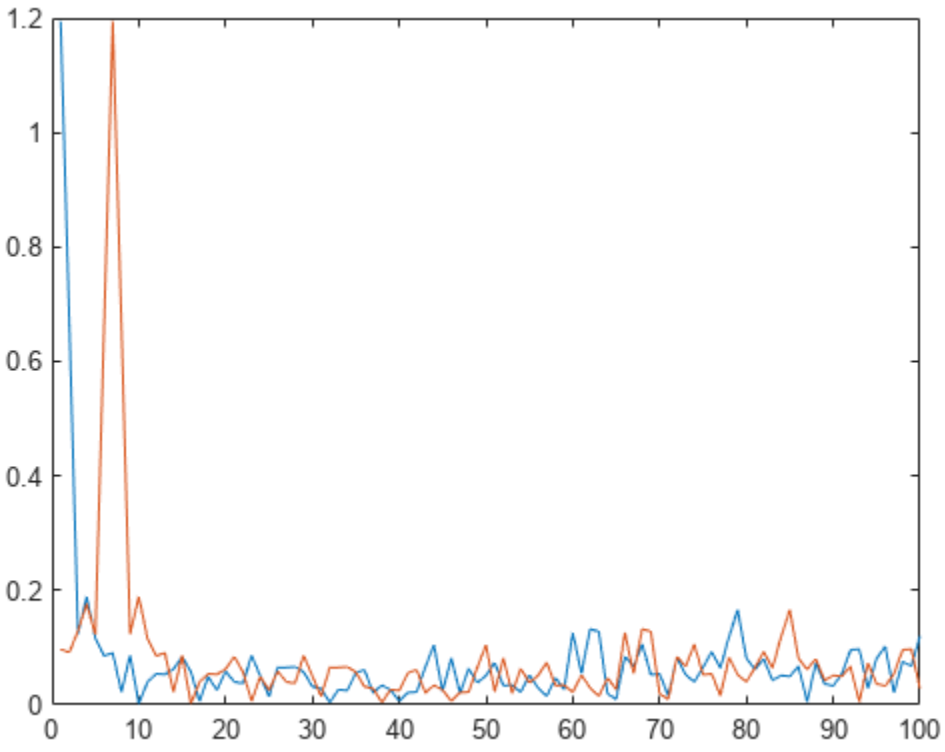
### Determine Offset

Calculate timing offset and return the correlations for the transmit waveform and for a delayed version of the transmit waveform.

```
[~,corr] = lteULFrameOffset(ue,ue.PUSCH,tx);
txDelayed = [zeros(6,1); tx];
[offset,corrDelayed] = lteULFrameOffset(ue,ue.PUSCH,txDelayed);
```

Plot the correlation data before and after delay is added. Zoom in on the *x*-axis to view correlation peaks.

```
plot(corr)
hold on
plot(corrDelayed)
hold off
xlim([0 100])
```




Correct the timing offset and demodulate the received waveform.

```
rxGrid = lteSCFDMADemodulate(ue,txDelayed(1+offset:end));
```

## Input Arguments

### ue — UE-specific settings

scalar structure

UE-specific settings, specified as a scalar structure with the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NULRB** | Required | Scalar integer from 6 to 110 | Number of uplink resource blocks. ($N_{\mathrm{RB}}^{\mathrm{UL}}$) |
| **NCellID** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **NSubframe** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **CyclicPrefixUL** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **NTxAnts** | Optional | 1 (default), 2, 4 | Number of transmission antennas. |
| **Hopping** | Optional | `'Off'` (default), `'Group'`, or `'Sequence'` | Frequency hopping method. |
| **SeqGroup** | Optional | 0 (default), integer from 0 to 29 | PUSCH sequence group assignment ($\Delta_{\mathrm{SS}}$).<br><br>Only used if `NDMRSID` or `NPUSCHID` is absent. |
| **CyclicShift** | Optional | 0 (default), integer from 0 to 7 | Number of cyclic shifts used for PUSCH DM-RS (yields $n_{DMRS}^{(1)}$). |
| **NPUSCHID** | Optional | 0 (default), nonnegative scalar integer from 0 to 509 | PUSCH virtual cell identity. If this field is not present, `NCellID` is used for group hopping sequence-shift pattern initialization.<br><br>See footnote. |
| **NDMRSID** | Optional | 0 (default), nonnegative scalar integer from 0 to 509 | DM-RS identity for cyclic shift hopping ($n_{ID}^{csh\_DMRS}$). If this field is not present, `NCellID` is used for cyclic shift hopping initialization.<br><br>See footnote. |
| **1** The pseudorandom sequence generator for cyclic shift hopping is initialized according to `NDMRSID`, if present — otherwise it is initialized according to the cell identity `NCellID` and the sequence group assignment `SeqGroup`. Similarly, the sequence-shift pattern for group hopping is initialized according to `NPUSCHID`, if present — otherwise it is initialized according to `NCellID` and `SeqGroup`. | | | |

Data Types: `struct`

### chs — PUSCH configuration

scalar structure

PUSCH configuration, specified as a scalar structure with the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **PRBSet** | Required | Integer column vector or two-column matrix | 0-based physical resource block indices (PRBs) for the slots of the current PUSCH resource allocation. As a column vector, the resource allocation is the same in both slots of the subframe. As a two-column matrix, it specifies different PRBs for each slot in a subframe. |
| **NLayers** | Optional | 1 (default), 2, 3, 4 | Number of transmission layers. |
| **DynCyclicShift** | Optional | 0 (default), integer from 0 to 7 | Cyclic shift for DM-RS (yields $n_{DMRS}^{(2)}$). |
| **OrthCover** | Optional | `'Off'` (default), `'On'` | Applies (`'On'`), or does not apply (`'Off'`), orthogonal cover sequence *w* (*Activate-DMRS-with OCC*). |
| The following field is required only when `ue.NTxAnts` is set to 2 or 4. | | | |
| **PMI** | Optional | 0 (default), nonnegative scalar integer from 0 to 23. | Scalar precoder matrix indication (PMI) to be used during precoding<br><br>See `lteULPMIInfo`. |

Data Types: `struct`

### `waveform` — Time-domain waveform

numeric matrix

Time-domain waveform, specified as a numeric matrix. `waveform` must be a $N_S$-by-$N_R$ matrix, where $N_S$ is the number of time-domain samples and $N_R$ is the number of receive antennas. `waveform` should be at least one subframe long and contain the DM-RS signals.

Generate `waveform` by SC-FDMA modulation of a resource matrix using the `lteSCFDMAModulate` function, or by using one of the channel model functions, `lteFadingChannel`, `lteHSTChannel`, or `lteMovingChannel`.

Data Types: `double`
Complex Number Support: Yes

## Output Arguments

### `offset` — Offset number of samples

scalar integer

Offset number of samples, returned as a scalar integer. This output is the number of samples from the start of the waveform to the position in that waveform where the first subframe containing the DM-RS begins. `offset` is computed by extracting the timing of the peak of the correlation between

`waveform` and internally generated reference waveforms containing DM-RS signals. The correlation is performed separately for each antenna and the antenna with the strongest correlation is used to compute `offset`.

**Note** `offset` is the position of `mod(max(abs(corr),`$L_{SF}$`))`, where $L_{SF}$ is the subframe length.

#### `corr` — Signal used to extract the timing offset

complex-valued numeric matrix

Signal used to extract the timing offset, returned as a complex-valued numeric matrix. `corr` has the same dimensions as `waveform`.

## Version History

**Introduced in R2014a**

## See Also

`lteFrequencyCorrect` | `lteFrequencyOffset` | `lteFadingChannel` | `lteMovingChannel` | `lteHSTChannel` | `lteSCFDMADemodulate`

# lteULFrameOffsetNPUSCH

Estimate NPUSCH DRS timing offset

## Syntax

```
[offset,corr] = lteULFrameOffsetNPUSCH(ue,chs,waveform)
[offset,corr] = lteULFrameOffsetNPUSCH(ue,chs,waveform,stateIn)
```

## Description

`[offset,corr] = lteULFrameOffsetNPUSCH(ue,chs,waveform)` performs slot synchronization on `waveform`, the input time-domain waveform, by using the narrowband physical uplink shared channel (NPUSCH) demodulation reference signal (DRS) symbols for user equipment (UE) settings `ue` and channel transmission configuration `chs`.

The function returns `offset`, the number of samples between the start of `waveform` and the sample within `waveform` at which the NPUSCH DRS symbols begin. The function also returns `corr`, the signal that the function uses to calculate `offset`.

The offset estimation process comprises these steps.

1. Extract the timing of the peak correlation between `waveform` and internally generated reference waveforms containing the NPUSCH DRS symbols.
2. Calculate the correlation for each antenna.
3. Compute the offset for the antenna with the strongest correlation.

`[offset,corr] = lteULFrameOffsetNPUSCH(ue,chs,waveform,stateIn)` specifies `stateIn`, the initial encoder state for NPUSCH DRS symbol generation.

## Examples

### Estimate NPUSCH DRS Timing Offset

Synchronize and demodulate a transmission containing NPUSCH DRS symbols.

Configure UE-specific settings.

```
ue = struct('NNCellID',0,'NBULSubcarrierSpacing','15kHz','NSlot',0);
```

Specify a channel transmission configuration.

```
chs = struct('NPUSCHFormat','Data','NRUsc',1,'NULSlots',16,'NRU',1, ...
    'NRep',1,'NBULSubcarrierSet',0,'Modulation','QPSK');
```

Generate the NPUSCH DRS symbols and allocate them to the appropriate locations on a resource grid.

```
grid = lteNBResourceGrid(ue);
grid(lteNPUSCHDRSIndices(ue,chs)) = lteNPUSCHDRS(ue,chs);
```

Generate a waveform by performing single-carrier frequency-division multiple access (SC-FDMA) modulation on the NPUSCH DRS symbols.

```
txWaveform = lteSCFDMAModulate(ue,chs,grid);
```

Create a received waveform by adding a time delay of 11 samples.

```
delay = 11;
waveform = [zeros(delay,1); txWaveform];
```

Specify an empty encoder state and calculate the timing offset in samples. Confirm that the result matches the added delay.

```
stateIn = struct();
[offset,corr] = lteULFrameOffsetNPUSCH(ue,chs,waveform,stateIn);
disp(isequal(offset,delay))
```

```
   1
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific settings, specified as a structure containing these fields.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **NBULSubcarrierSpacing** | `'3.75kHz'`, `'15kHz'` | NB-IoT uplink subcarrier spacing<br><br>To set a subcarrier spacing of 3.75 kHz, specify this field as `'3.75kHz'`. To set a subcarrier spacing of 15 kHz, specify this field as `'15kHz'`. | `char`, `string` |
| **Windowing** | Nonnegative even integer<br><br>Default depends on the value of the `NBULSubcarrierSpacing` field | Number of time-domain samples over which the function applies windowing and overlapping of SC-FDMA symbols | `double` |
| **NNCellID** | Integer in the interval [0, 503] | Narrowband physical layer cell identity (PCI) | `double` |
| **NFrame** | `0` (default), nonnegative integer | Frame number | `double` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **NSlot** | Nonnegative integer | Slot number<br><br>When you specify the `NPUSCHFormat` field as `'Data'` and the `SeqGroupHopping` field as `'Off'` in the `chs` input, the function ignores this field. | `double` |

Data Types: `struct`

### `chs` — Channel transmission configuration

structure

Channel transmission configuration, specified as a structure containing these fields.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **NPUSCHFormat** | `'Data'`, `'Control'` | NPUSCH format<br><br>To indicate that the NPUSCH carries narrowband uplink shared channel (UL-SCH) data, specify this field as `'Data'`. To indicate that the NPUSCH carries uplink control information, specify this field as `'Control'`. | `char`, `string` |
| **NRUsc** | 1, 3, 6, 12 | Number of consecutive subcarriers in a resource unit (RU)<br><br>If you specify the `NPUSCHFormat` field as `'Control'` or the `NBULSubcarrierSpacing` field of the `ue` input as `'3.75kHz'`, then you must specify this field as `1` . | `double` |
| **NRep** | 1, 2, 4, 8, 16, 32, 64, 128 | Number of repetitions for a codeword | `double` |
| **NRU** | 1, 2, 3, 4, 5, 6, 8, 10 | Number of RUs | `double` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`NULSlots`** | 2, 4, 8, 16 | Number of slots per RU<br><br>If you specify the `NPUSCHFormat` field as `'Control'`, then you must specify this field as 4.<br><br>If you specify the `NPUSCHFormat` field as `'Data'`, then you must specify this field as:<br>• 16 when you specify the `NRUsc` field as 1<br>• 8 when you specify the `NRUsc` field as 3<br>• 4 when you specify the `NRUsc` field as 6<br>• 2 when you specify the `NRUsc` field as 12 | `double` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **BaseSeqIdx** | Integer in the interval [0, 29]. Default depends on the value of the `NRUsc` field. | Multitone NPUSCH DRS base sequence index<br>• When you specify the `NRUsc` field as `3`, specify this field as an integer in the interval [0, 11]. If you do not specify this field, the function sets it to the value of `mod(ue.NNCellID, 12)`.<br>• When you specify the `NRUsc` field as `6`, specify this field as an integer in the interval [0, 13]. If you do not specify this field, the function sets it to the value of `mod(ue.NNCellID, 14)`.<br>• When you specify the `NRUsc` field as `12`, specify this field as an integer in the interval [0, 29]. If you do not specify this field, the function sets it to the value of `mod(ue.NNCellID, 30)`.<br>• When you specify the `NRUsc` field as any other value, the function does not use this field.<br>**Dependencies.** To enable this field, specify the `NRUsc` field as `3`, `6`, or `12`. | `double` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`SeqGroupHopping`** | `'On'` (default), `'Off'` | To enable sequence-group hopping, specify this field as `'On'`. To disable sequence group hopping, specify this field as `'Off'`. For more information, see section 5.5.1.3 of [1]. | `char`, `string` |
| **`SeqGroup`** | `0` (default), integer in the interval [0, 29] | Sequence-group assignment for sequence shift pattern calculation. For more information, see section 10.1.4.1.3 of [1].<br><br>**Dependencies.** To enable this field, specify the `SeqGroupHopping` field as `'On'`. | `double` |
| **`CyclicShift`** | `0` (default), integer in the interval [0, 3] | Cyclic shift<br>• When you specify the `NRUsc` field as `3`, specify this field as an integer in the interval [0, 2].<br>• When you specify the `NRUsc` field as `6`, specify this field as an integer in the interval [0, 3].<br><br>**Dependencies.** To enable this field, specify the `NRUsc` field as `3` or `6`. | `double` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **NBULSubcarrierSet** | Integer in the interval [0, 47], vector of integers in the interval [0, 11] | NB-IoT uplink subcarrier indices, in zero-based form, specified as one of these values:<br>• An integer in the interval [0, 11] when you specify the `NPUSCHFormat` field as `'Control'`<br>• An integer in the interval [0, 47] when you specify the `NPUSCHFormat` field as `'Data'` and the `NBULSubcarrierSpacing` fields of the `ue` input as `'3.75kHz'`<br>• A vector of integers in the interval [0, 11] when you specify the `NPUSCHFormat`field as `'Data'` and the `NBULSubcarrierSpacing` fields of the `ue` input as `'15kHz'`. | `double` |
| **Modulation** | `'BPSK'`, `'QPSK'` | Modulation type<br>To enable binary phase-shift keying (BPSK), specify this field as `'BPSK'`. To enable quadrature phase-shift keying (QPSK), specify this field as `'QPSK'`.<br>If you specify the `NPUSCHFormat` field as `'Control'`, then you must specify this field as `'BPSK'`. | `char`, `string` |
| **SlotIdx** | Integer in the interval [0, (`chs.NRU` × `chs.NULSlots` × `chs.NRep`) – 1] | Index of a slot within a bundle, in zero-based form | `double` |

Data Types: `struct`

### `waveform` — Time-domain waveform

complex-valued matrix

Time-domain waveform, specified as a complex-valued matrix of size *T*-by-*R*.

- *T* is the number of time-domain samples.
- *R* is the number of receive antennas.

You can generate this input by performing SC-FDMA modulation on a resource matrix using the `lteSCFDMAModulate` function. Alternatively, you can generate this input by using either of these channel model functions: `lteFadingChannel` or `lteMovingChannel`.

Data Types: `double`
Complex Number Support: Yes

### `stateIn` — Encoder state

`struct()` (default) | structure

Encoder state for NPUSCH DRS generation, specified as a structure. This input corresponds to the `stateIn` input of the `lteNPUSCHDRS` function. This input contains the internal state of each transport block in these fields.

| Field | Values | Description | Data Types |
|---|---|---|---|
| **`SlotIdx`** | Integer in the interval [0, (`chs.NRU` × `chs.NULSlots` × `chs.NRep`) – 1] | Index of a slot within a bundle, in zero-based form | `double` |
| **`InitNSlot`** | Nonnegative integer | Slot number for scrambling sequence initialization | `double` |
| **`InitNFrame`** | Nonnegative integer | Frame number for scrambling sequence initialization | `double` |
| **`EndOfBlk`** | Logical `1` (`true`) or `0` (`false`) | To indicate that the transmission has reached the end of a transport block, specify this field as `1` (`true`). Otherwise, specify this field as `0` (`false`). | `logical` |
| **`EndOfTx`** | Logical `1` (`true`) or `0` (`false`) | To indicate that the transmission has reached the end of a bundle, specify this field as `1` (`true`). Otherwise, specify this field as `0` (`false`). | `logical` |

| Field | Values | Description | Data Types |
|---|---|---|---|
| **GhpNSlot** | Nonnegative integer | Slot number for the first slot in the RU<br><br>**Dependencies.** To enable this field, specify the NPUSCHFormat field as 'Data' and the NRUsc field as 1 in the chs input. | double |

Data Types: struct

## Output Arguments

### offset — Timing offset

integer

Timing offset, returned as an integer. This output represents the offset, in samples, between the start of the waveform input and the sample within waveform at which the NPUSCH DRS symbols begin. The function returns this output as the value of max(abs(corr)) modulo slot length.

Data Types: double

### corr — Signal used to estimate timing offset

complex-valued matrix

Signal used to estimate timing offset, returned as a complex-valued matrix of the same dimensions as the waveform input.

Data Types: double
Complex Number Support: Yes

# Version History

**Introduced in R2020a**

## References

[1] 3GPP TS 36.211. “Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. https://www.3gpp.org.

## See Also

### Functions

lteFadingChannel | lteMovingChannel | lteNPUSCHDRS | lteSCFDMADemodulate | lteULChannelEstimateNPUSCH

# lteULFrameOffsetPUCCH1

PUCCH format 1 DM-RS uplink subframe timing estimate

## Syntax

```
offset = lteULFrameOffsetPUCCH1(ue,chs,waveform)
[offset,corr] = lteULFrameOffsetPUCCH1(ue,chs,waveform)
```

## Description

`offset = lteULFrameOffsetPUCCH1(ue,chs,waveform)` performs synchronization using PUCCH format 1 demodulation reference signals (DM-RS) for the time-domain waveform, `waveform`, given UE-specific settings, `ue`, and PUCCH format 1 configuration, `chs`.

The returned value `offset` indicates the number of samples from the start of the waveform `waveform` to the position in that waveform where the first subframe containing the DM-RS begins.

`offset` provides subframe timing. Frame timing can be achieved by using `offset` with the subframe number, `ue.NSubframe`. This behavior is consistent with real-world operation because the base station knows when, or in which subframe, to expect uplink transmissions.

`[offset,corr] = lteULFrameOffsetPUCCH1(ue,chs,waveform)` also returns a complex matrix `corr`, which is the signal used to extract the timing offset.

## Examples

### Synchronize and Demodulate Using PUCCH Format 1 DM-RS

Synchronize and demodulate a transmission that has been delayed by four samples using the PUCCH format 1 demodulation reference signal (DM-RS) symbols.

Initialize configuration structures (`ue` and `pucch1`).

```
ue = struct('NULRB',6,'NCellID',0,'NSubframe',0,'Hopping','Off');
ue.CyclicPrefixUL = 'Normal';
ue.NTxAnts = 1;

pucch1 = struct('ResourceIdx',0);
pucch1.CyclicShifts = 0;
pucch1.DeltaShift = 1;
pucch1.ResourceSize = 0;
```

On the transmit side, populate `reGrid`, generate `waveform`, and insert a delay of four samples.

```
reGrid = lteULResourceGrid(ue);
reGrid(ltePUCCH1DRSIndices(ue,pucch1)) = ltePUCCH1DRS(ue,pucch1);
waveform = lteSCFDMAModulate(ue,reGrid);
tx = [zeros(4,1); waveform];
```

On the receive side, perform synchronization using the PUCCH format 1 DM-RS symbols for the time-domain waveform and demodulate adjusting for the frame timing estimate. Show estimated frame timing offset.

```
fOffset = lteULFrameOffsetPUCCH1(ue,pucch1,tx)
```

```
fOffset = 4
```

```
rxGrid = lteSCFDMADemodulate(ue,tx(1+fOffset:end));
```

### View PUCCH Format 1 DM-RS Transmission Correlation Peaks

View the correlation peak for a delayed transmit waveform. The transmission contains PUCCH format 1 demodulation reference signal (DM-RS) symbols available for estimating the waveform timing.

#### UE Configuration

Configure UE-specific settings and channel transmission parameters.

```
ue = struct('NULRB',6,'NCellID',0,'NSubframe',0,'Hopping','Off');
ue.CyclicPrefixUL = 'Normal';
ue.NTxAnts = 1;
pucch1 = struct('ResourceIdx',0,'CyclicShifts',0, ...
    'DeltaShift',1,'ResourceSize',0);
```

#### Generate Transmit Waveform

On the transmit side, populate a resource grid and generate a waveform containing PUCCH1 DM-RS.

```
reGrid = lteULResourceGrid(ue);
reGrid(ltePUCCH1DRSIndices(ue,pucch1)) = ltePUCCH1DRS(ue,pucch1);
tx = lteSCFDMAModulate(ue,reGrid);
```

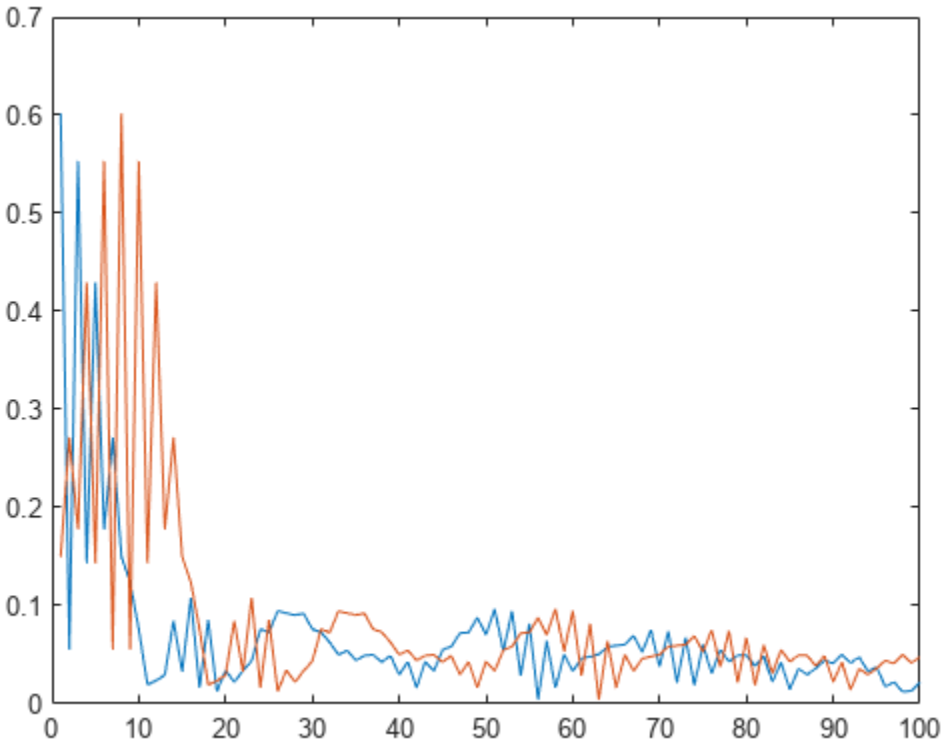
#### Waveform Reception

On the receive side, calculate timing offset using the PUCCH format 1 DM-RS symbols for the time-domain waveform. Estimate the correlations for the transmit waveform and for a delayed version of the transmit waveform.

```
[~,corr] = lteULFrameOffsetPUCCH1(ue,pucch1,tx);
txDelayed = [zeros(7,1); tx];
[offset,corrDelayed] = lteULFrameOffsetPUCCH1(ue,pucch1,txDelayed);
```

Plot the correlation data before and after delay is added. Zoom in on the *x*-axis to view correlation peaks.

```
plot(corr)
hold on
plot(corrDelayed)
hold off
xlim([0 100])
```




Correct the timing offset and demodulate the received waveform.

```
rxGrid = lteSCFDMADemodulate(ue,txDelayed(1+offset:end));
```

## Input Arguments

### ue — UE-specific settings

scalar structure

UE-specific settings, specified as a scalar structure with the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NULRB`** | Required | Scalar integer from 6 to 110 | Number of uplink resource blocks. ($N_{RB}^{UL}$) |
| **`NCellID`** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **`NSubframe`** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **`CyclicPrefixUL`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **`NTxAnts`** | Optional | 1 (default), 2, 4 | Number of transmission antennas. |
| **`Hopping`** | Optional | `'Off'` (default), `'Group'` | Frequency hopping method. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NPUCCHID** | Optional | Integer from 0 to 503 | PUCCH virtual cell identity. If this field is not present, `NCellID` is used as the identity. |

Data Types: `struct`

**`chs` — PUCCH format 1 configuration**
scalar structure

PUCCH format 1 configuration, specified as a scalar structure with the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **ResourceIdx** | Optional | 0 (default), integer from 0 to 2047 or vector of integers. | PUCCH resource indices, specified as an integer or a vector of integers. Values range from 0 to 2047. These indices determine the physical resource blocks, cyclic shift and orthogonal cover used for transmission. ($n^{(1)}_{PUCCH}$). Define one index for each transmission antenna. |
| **ResourceSize** | Optional | 0 (default), integer from 0 to 98. | Size of resource allocated to PUCCH format 2 ($N^{(2)}_{RB}$) |
| **DeltaShift** | Optional | 1 (default), 2, 3 | Delta shift, specified as 1, 2, or 3. ($\Delta_{\text{shift}}$) |
| **DeltaOffset** | Optional | 0 (default), 1, 2 | ($\Delta_{\text{offset}}$). Warning: The use of this parameter field is not advised. It applies only to 3GPP releases preceding v8.5.0. This parameter will be removed in a future release. |
| **CyclicShifts** | Optional | 0 (default), integer from 0 to 7 | Number of cyclic shifts used for format 1 in resource blocks (RBs) with a mixture of format 1 and format 2 PUCCH, specified as an integer from 0 to 7. ($N^{(1)}_{cs}$) |

Data Types: `struct`

**`waveform` — Time-domain waveform**
numeric matrix

Time-domain waveform, specified as a numeric matrix. `waveform` must be a $N_S$-by-$N_R$ matrix, where $N_S$ is the number of time-domain samples and $N_R$ is the number of receive antennas. `waveform` should be at least one subframe long and contain the DM-RS signals.

Generate `waveform` by SC-FDMA modulation of a resource matrix using `lteSCFDMAModulate` function, or by using one of the channel model functions (`lteFadingChannel`, `lteHSTChannel`, or `lteMovingChannel`).

Data Types: `double`
Complex Number Support: Yes

## Output Arguments

### `offset` — Number of samples from the start of the waveform to the position in that waveform where the first subframe begins

scalar integer

Number of samples from the start of the waveform to the position in that waveform where the first subframe containing the DM-RS begins, returned as a scalar integer. `offset` is computed by extracting the timing of the peak of the correlation between `waveform` and internally generated reference waveforms containing DM-RS signals. The correlation is performed separately for each antenna and the antenna with the strongest correlation is used to compute `offset`.

**Note** `offset` is the position of `mod(max(abs(corr),`$L_{SF}$`))`, where $L_{SF}$ is the subframe length.

### `corr` — Signal used to extract the timing offset

numeric matrix

Signal used to extract the timing offset, returned as a complex numeric matrix. `corr` has the same dimensions as `waveform`.

# Version History

**Introduced in R2014a**

## See Also

`lteULFrameOffset` | `lteULFrameOffsetPUCCH2` | `lteULFrameOffsetPUCCH3` | `lteFadingChannel` | `lteMovingChannel` | `lteHSTChannel` | `lteSCFDMADemodulate`

# lteULFrameOffsetPUCCH2

PUCCH format 2 DM-RS uplink subframe timing estimate

## Syntax

```
offset = lteULFrameOffsetPUCCH2(ue,chs,waveform,oack)
[offset,ack] = lteULFrameOffsetPUCCH2(ue,chs,waveform,oack)
[offset,ack,corr] = lteULFrameOffsetPUCCH2(ue,chs,waveform,oack)
```

## Description

`offset = lteULFrameOffsetPUCCH2(ue,chs,waveform,oack)` performs synchronization using PUCCH format 2 demodulation reference signals (DM-RS) for the time-domain waveform, `waveform`, given UE-specific settings, `ue`, PUCCH format 2 configuration `chs`, and the number of Hybrid ARQ indicators `oack`.

The returned value `offset` indicates the number of samples from the start of the waveform `waveform` to the position in that waveform where the first subframe containing the DM-RS begins.

`offset` provides subframe timing; frame timing can be achieved by using `offset` with the subframe number, `ueNSubframe`. This behavior is consistent with real-world operation because the base station knows when, in which subframe, to expect uplink transmissions.

`[offset,ack] = lteULFrameOffsetPUCCH2(ue,chs,waveform,oack)` also returns a vector `ack` of decoded PUCCH format 2 Hybrid ARQ indicators.

`[offset,ack,corr] = lteULFrameOffsetPUCCH2(ue,chs,waveform,oack)` also returns a complex matrix `corr`, which is used to extract the timing offset.

## Examples

### Synchronize and Demodulate Using PUCCH Format 2 DM-RS

This example performs synchronization and uses the PUCCH format 2 DM-RS when demodulating a transmission that has been delayed by 5 samples.

Initialize ue specific parameter structure, PUCCH2 structure, UL resource grid and txAck parameter.

```
ue.NULRB = 6;
ue.NCellID = 0;
ue.NSubframe = 0;
ue.Hopping = 'Off';
ue.CyclicPrefixUL = 'Normal';
ue.NTxAnts = 1;
pucch2.ResourceIdx = 0;
pucch2.ResourceSize = 0;
pucch2.CyclicShifts = 0;
rgrid = lteULResourceGrid(ue);
txAck = [1;1];
rgrid(ltePUCCH2DRSIndices(ue,pucch2)) = ltePUCCH2DRS(ue,pucch2,txAck);
```

Generate modulated waveform and add a five sample delay.

```
waveform = lteSCFDMAModulate(ue,rgrid);
tx = [zeros(5,1);waveform];
```

Use PUCCH format 2 DM-RS to estimate UL frame offset timing, then demodulate the waveform.

```
offset = lteULFrameOffsetPUCCH2(ue,pucch2,tx,length(txAck))
```

```
offset = 5
```

```
rxGrid = lteSCFDMADemodulate(ue,tx(1+offset:end));
```

### View PUCCH Format 2 H-ARQ Indicators

View the Hybrid ARQ indicators for a PUCCH format 2 transmission waveform. The transmission contains PUCCH format 2 demodulation reference signal (DM-RS) symbols available for estimating the waveform timing.

#### UE Configuration

Create configuration structures for ue and pucch2.

```
ue.NULRB = 6;
ue.NCellID = 0;
ue.NSubframe = 0;
ue.Hopping = 'Off';
ue.CyclicPrefixUL = 'Normal';
ue.NTxAnts = 1;

pucch2.ResourceIdx = 0;
pucch2.ResourceSize = 0;
pucch2.CyclicShifts = 0;
```

#### Generate Transmission Waveform

On the transmit side, populate a resource grid and generate a waveform containing PUCCH2 DM-RS.

```
reGrid = lteULResourceGrid(ue);
txAck = [0;1];
reGrid(ltePUCCH2DRSIndices(ue,pucch2)) = ltePUCCH2DRS(ue,pucch2,txAck);

tx = lteSCFDMAModulate(ue,reGrid);
```

#### Waveform Reception

On the receive side, calculate timing offset using the PUCCH2 DM-RS symbols for the time-domain waveform and return decoded PUCCH format 2 Hybrid ARQ indicators.

```
[offset,ack] = lteULFrameOffsetPUCCH2(ue,pucch2,tx,length(txAck));
ack
```

```
ack = 2x1 logical array

   0
   1
```

Correct the timing offset and demodulate the received waveform.

```
rxGrid = lteSCFDMADemodulate(ue,tx(1+offset:end));
```

### View PUCCH Format 2 DM-RS Transmission Correlation Peaks

View the correlation peak for a transmission waveform that has been delayed. The transmission contains PUCCH format 2 demodulation reference signal (DM-RS) symbols available for estimating the waveform timing.

#### UE Configuration

Create configuration structures for ue and pucch2.

```
ue.NULRB = 6;
ue.NCellID = 0;
ue.NSubframe = 0;
ue.Hopping = 'Off';
ue.CyclicPrefixUL = 'Normal';
ue.NTxAnts = 1;

pucch2.ResourceIdx = 0;
pucch2.ResourceSize = 0;
pucch2.CyclicShifts = 0;
```

#### Generate Transmission Waveform

On the transmit side, populate a resource grid and generate a waveform containing PUCCH2 DM-RS.

```
reGrid = lteULResourceGrid(ue);
txAck = [1;1];
reGrid(ltePUCCH2DRSIndices(ue,pucch2)) = ltePUCCH2DRS(ue,pucch2,txAck);

tx = lteSCFDMAModulate(ue,reGrid);
```

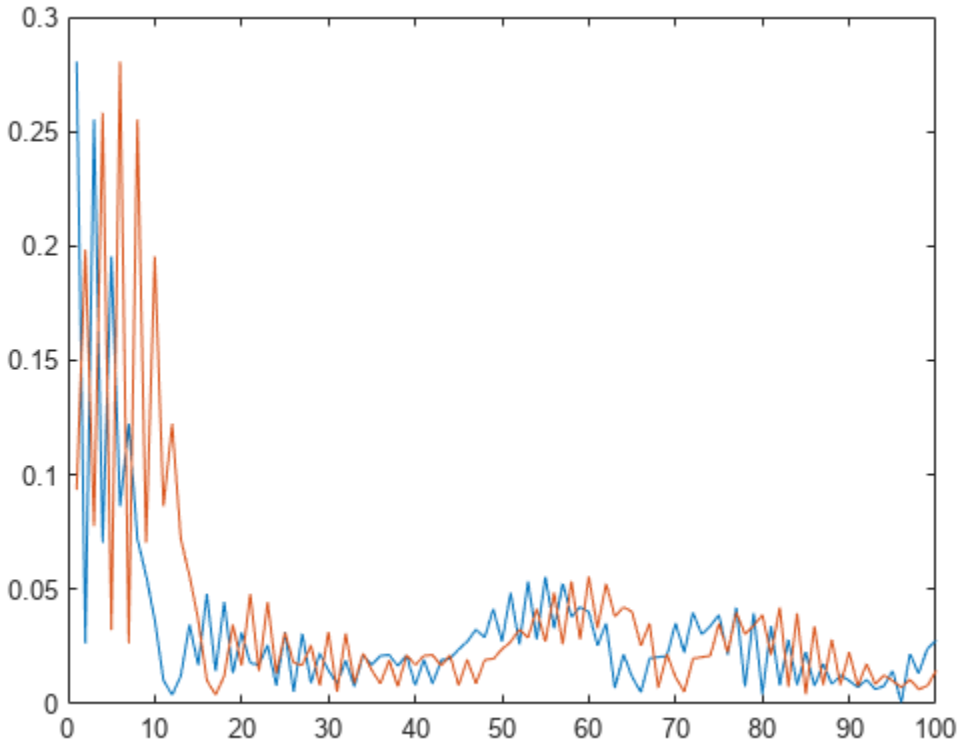
#### Waveform Reception

On the receive side, calculate timing offset using the PUCCH2 DM-RS symbols for the time-domain waveform and return the correlations for the transmit waveform and for a delayed version of the transmit waveform.

```
[~,ack,corr] = lteULFrameOffsetPUCCH2(ue,pucch2,tx,length(txAck));

txDelayed = [zeros(5,1); tx];
[offset,ack,corrDelayed] = lteULFrameOffsetPUCCH2(ue,pucch2,txDelayed,length(txAck));
```

Plot the correlation data before and after delay is added. Zoom in on the *x*-axis to view correlation peaks.

```
plot(corr)
hold on
plot(corrDelayed)
hold off
xlim([0 100])
```




Correct the timing offset and demodulate the received waveform.

```
rxGrid = lteSCFDMADemodulate(ue,txDelayed(1+offset:end));
```

## Input Arguments

### ue — UE-specific settings

scalar structure

UE-specific settings, specified as a scalar structure with the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NULRB`** | Required | Scalar integer from 6 to 110 | Number of uplink resource blocks. ($N_{RB}^{UL}$) |
| **`NCellID`** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **`NSubframe`** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **`CyclicPrefixUL`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **`NTxAnts`** | Optional | 1 (default), 2, 4 | Number of transmission antennas. |
| **`Hopping`** | Optional | `'Off'` (default), `'Group'` | Frequency hopping method. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NPUCCHID** | Optional | Integer from 0 to 503 | PUCCH virtual cell identity. If this field is not present, `NCellID` is used as the identity. |

Data Types: `struct`

### chs — PUCCH format 2 configuration

scalar structure

PUCCH format 2 configuration, specified as a scalar structure with the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **ResourceIdx** | Optional | 0 (default), integer from 0 to 1185 or vector of integers. | PUCCH resource indices which determine the physical resource blocks, cyclic shift, and orthogonal cover used for transmission. ($n^{(2)}_{PUCCH}$). Define one index for each transmission antenna. |
| **ResourceSize** | Optional | 0 (default), integer from 0 to 98. | Size of resource allocated to PUCCH format 2 ($N^{(2)}_{RB}$) |
| **CyclicShifts** | Optional | 0 (default), integer from 0 to 7 | Number of cyclic shifts used for format 1 in resource blocks (RBs) with a mixture of format 1 and format 2 PUCCH, specified as an integer from 0 to 7. ($N^{(1)}_{cs}$) |

Data Types: `struct`

### waveform — Time-domain waveform

numeric matrix

Time-domain waveform, specified as a numeric matrix. `waveform` must be a $N_S$-by-$N_R$ matrix, where $N_S$ is the number of time-domain samples and $N_R$ is the number of receive antennas. `waveform` should be at least one subframe long and contain the DM-RS signals.

Generate `waveform` by SC-FDMA modulation of a resource matrix using the `lteSCFDMAModulate` function, or by using one of the channel model functions, `lteFadingChannel`, `lteHSTChannel`, or `lteMovingChannel`.

Data Types: `double`
Complex Number Support: Yes

### oack — Number of uncoded Hybrid ARQ bits

1 | 2

Number of uncoded Hybrid ARQ bits expected, 1 (PUCCH format 2a) or 2 (PUCCH format 2b).

Data Types: `double`

## Output Arguments

### `offset` — Number of samples from the start of the waveform to the position in that waveform where the first subframe begins

scalar integer

Number of samples from the start of the waveform to the position in that waveform where the first subframe containing the DM-RS begins, returned as a scalar integer. `offset` is computed by extracting the timing of the peak of the correlation between `waveform` and internally generated reference waveforms containing DM-RS signals. The correlation is performed separately for each antenna and the antenna with the strongest correlation is used to compute `offset`. This process is repeated for either one or two Hybrid ARQ indicators combination as specified by the parameter `oack`. This correlation amounts to a maximum likelihood (ML) decoding of the Hybrid ARQ indicators, which are signaled on the PUCCH format 2 DM-RS.

**Note** `offset` is the position of `mod(max(abs(corr),`$L_{SF}$`))`, where $L_{SF}$ is the subframe length.

### `ack` — Decoded PUCCH format 2 Hybrid ARQ bits

numeric vector or matrix

Decoded PUCCH format 2 Hybrid ARQ bits, returned as a numeric vector or matrix. If multiple decoded Hybrid ARQ indicator vectors have a likelihood equal to the maximum, `ack` is a matrix where each column represents one of the equally likely Hybrid ARQ indicator vectors.

### `corr` — Signal used to extract the timing offset

numeric matrix

Signal used to extract the timing offset, returned as a complex numeric matrix. `corr` has the same dimensions as `waveform`.

# Version History

**Introduced in R2014a**

## See Also

`lteULFrameOffset` | `lteULFrameOffsetPUCCH1` | `lteULFrameOffsetPUCCH3` | `lteFadingChannel` | `lteMovingChannel` | `lteHSTChannel` | `lteSCFDMADemodulate`

# lteULFrameOffsetPUCCH3

PUCCH format 3 DM-RS uplink subframe timing estimate

## Syntax

```
offset = lteULFrameOffsetPUCCH3(ue,chs,waveform)
[offset corr] = lteULFrameOffsetPUCCH3(ue,chs,waveform)
```

## Description

`offset = lteULFrameOffsetPUCCH3(ue,chs,waveform)` performs synchronization using PUCCH format 3 demodulation reference signals (DM-RS) for the time-domain waveform, `waveform`, given UE-specific settings, `ue`, and PUCCH format 3 configuration, `chs`.

The returned value, `offset`, indicates the number of samples from the start of the waveform, `waveform`, to the position in that waveform where the first subframe containing the DM-RS begins.

`offset` provides subframe timing; frame timing can be achieved by using `offset` with the subframe number, `ue.NSubframe`. This behavior is consistent with real-world operation because the base station knows when, or in which subframe, to expect uplink transmissions.

`[offset corr] = lteULFrameOffsetPUCCH3(ue,chs,waveform)` also returns a complex-valued matrix `corr`, which is the signal used to extract the timing offset.

## Examples

### Synchronize and Demodulate Using PUCCH Format 3 DM-RS

Synchronize and demodulate a transmission that has been delayed by seven samples using the PUCCH format 3 demodulation reference signal (DM-RS) symbols.

Initialize configuration structures (`ue` and `pucch3`).

```
ue = struct('NULRB',6,'NCellID',0,'NSubframe',0,'Hopping','Off');
ue.CyclicPrefixUL = 'Normal';
ue.NTxAnts = 1;
ue.Shortened = 0;

pucch3 = struct('ResourceIdx',0);
```

On the transmit side, populate `reGrid`, generate `waveform`, and insert a delay of seven samples.

```
reGrid = lteULResourceGrid(ue);
reGrid(ltePUCCH3DRSIndices(ue,pucch3)) = ltePUCCH3DRS(ue,pucch3);
waveform = lteSCFDMAModulate(ue,reGrid);
tx = [zeros(7,1); waveform];
```

On the receive side, perform synchronization using the PUCCH format 3 DM-RS symbols for the time-domain waveform and demodulate adjusting for the frame timing estimate. Show estimated frame timing offset.

```
fOffset = lteULFrameOffsetPUCCH3(ue,pucch3,tx)

fOffset = 7

rxGrid = lteSCFDMADemodulate(ue,tx(1+fOffset:end));
```

### View PUCCH Format 3 DM-RS Transmission Correlation Peaks

View the correlation peak for a transmission waveform that has been delayed. The transmission contains PUCCH format 3 demodulation reference signal (DM-RS) symbols available for estimating the waveform timing.

#### UE Configuration

Create configuration structures for ue and pucch3.

```
ue = struct('NULRB',6,'NCellID',0,'NSubframe',0,'Hopping','Off');
ue.CyclicPrefixUL = 'Normal';
ue.NTxAnts = 1;
ue.Shortened = 0;

pucch3 = struct('ResourceIdx',0);
```

#### Generate Transmission Waveform

On the transmit side, populate a resource grid and generate a waveform containing PUCCH3 DM-RS.

```
reGrid = lteULResourceGrid(ue);
reGrid(ltePUCCH3DRSIndices(ue,pucch3)) = ltePUCCH3DRS(ue,pucch3);

tx = lteSCFDMAModulate(ue,reGrid);
```

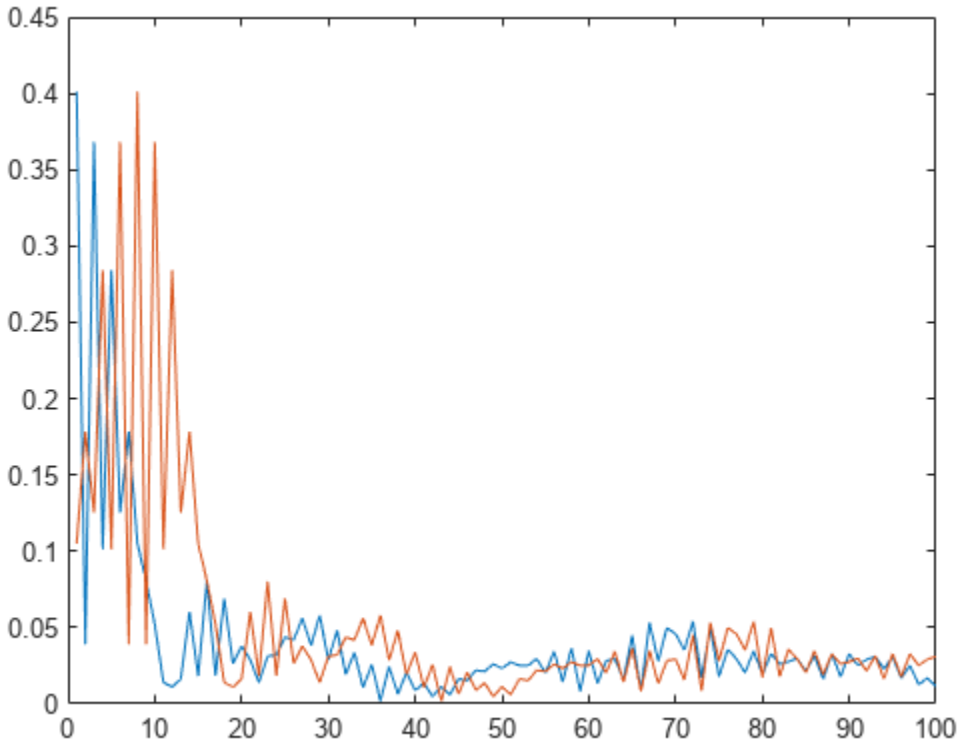
#### Waveform Reception

On the receive side, calculate timing offset using the PUCCH3 DM-RS symbols for the time-domain waveform and return the correlations for the transmit waveform and for a delayed version of the transmit waveform.

```
[~,corr] = lteULFrameOffsetPUCCH3(ue,pucch3,tx);

txDelayed = [zeros(7,1); tx];
[offset,corrDelayed] = lteULFrameOffsetPUCCH3(ue,pucch3,txDelayed);
```

Plot the correlation data before and after delay is added. Zoom in on the *x*-axis to view correlation peaks.

```
plot(corr)
hold on
plot(corrDelayed)
hold off
xlim([0 100])
```




Correct the timing offset and demodulate the received waveform.

```
rrxGrid = lteSCFDMADemodulate(ue,txDelayed(1+offset:end));
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific settings, specified as a structure with the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`NULRB`** | Required | Scalar integer from 6 to 110 | Number of uplink resource blocks. ($N_{RB}^{UL}$) |
| **`NCellID`** | Required | Integer from 0 to 503 | Physical layer cell identity |
| **`NSubframe`** | Required | 0 (default), nonnegative scalar integer | Subframe number |
| **`CyclicPrefixUL`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length |
| **`NTxAnts`** | Optional | 1 (default), 2, 4 | Number of transmission antennas. |
| **`Hopping`** | Optional | `'Off'` (default), `'Group'` | Frequency hopping method. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`Shortened`** | Optional | 0 (default), 1 | Option to shorten the subframe by omitting the last symbol, specified as 0 or 1. If `1`, the last symbol of the subframe is not used. For subframes with possible SRS transmission, set `Shortened` to 1 to maintain a standard compliant configuration. |
| **`NPUCCHID`** | Optional | Integer from 0 to 503 | PUCCH virtual cell identity. If this field is not present, `NCellID` is used as the identity. |

Data Types: `struct`

**`chs` — PUCCH format 3 configuration**
scalar structure

PUCCH format 3 configuration, specified as a scalar structure with the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`ResourceIdx`** | Optional | 0 (default), integer from 0 to 549, or vector of integers. | PUCCH resource indices which determine the physical resource blocks, cyclic shift, and orthogonal cover used for transmission ($n^{(3)}_{PUCCH}$). Define one index for each transmission antenna. |

Data Types: `struct`

**`waveform` — Time-domain waveform**
numeric matrix

Time-domain waveform, specified as a numeric matrix. `waveform` must be a $N_S$-by-$N_R$ matrix, where $N_S$ is the number of time-domain samples and $N_R$ is the number of receive antennas. `waveform` should be at least one subframe long and contain the DM-RS signals.

Generate `waveform` by SC-FDMA modulation of a resource matrix using the `lteSCFDMAModulate` function, or by using one of the channel model functions, `lteFadingChannel`, `lteHSTChannel`, or `lteMovingChannel`.

Data Types: `double`
Complex Number Support: Yes

## Output Arguments

**`offset` — Number of samples from the start of the waveform to the position in that waveform where the first subframe begins**
scalar integer

Number of samples from the start of the waveform to the position in that waveform where the first subframe begins, returned as a scalar integer. `offset` is computed by extracting the timing of the peak of the correlation between `waveform` and internally generated reference waveforms containing DM-RS signals. The correlation is performed separately for each antenna and the antenna with the strongest correlation is used to compute `offset`.

**Note** `offset` is the position of `mod(max(abs(corr),`$L_{SF}$`))`, where $L_{SF}$ is the subframe length.

### `corr` — Signal used to extract the timing offset

numeric matrix

Signal used to extract the timing offset, returned as a numeric matrix. `corr` has the same dimensions as `waveform`.

## Version History

**Introduced in R2014a**

## See Also

`lteULFrameOffset` | `lteULFrameOffsetPUCCH1` | `lteULFrameOffsetPUCCH2` | `lteFadingChannel` | `lteMovingChannel` | `lteHSTChannel` | `lteSCFDMADemodulate`

# lteULPMIInfo

PUSCH precoder matrix indication reporting information

## Syntax

```
info=lteULPMIInfo(ue,chs)
```

## Description

`info=lteULPMIInfo(ue,chs)` returns a structure `info` containing information related to precoder matrix indication (PMI) reporting.

You can use `info.NSubbands` to determine the correct size of the vector PMI required for closed-loop spatial multiplexing operation. PMI is a column vector with `info.NSubbands` rows. Currently, only wideband PMI reporting is defined by the standard. Thus, the number of subbands, `info.NSubbands`, is always 1. This field and `info.k` are provided for consistency with the downlink version of this function, `ltePMIInfo`.

## Examples

### Get PUSCH PMI Reporting Information

```
Get PMI reporting information for FRC A3-2.

ue = lteRMCUL('A3-2');
pmiInfo = lteULPMIInfo(ue, ue.PUSCH)

pmiInfo = struct with fields:
            k: 6
    NSubbands: 1
       MaxPMI: 0
```

## Input Arguments

### ue — UE-specific configuration

structure

UE-specific configuration, specified as a structure. `ue` can contain the following fields.

### NULRB — Number of uplink resource blocks

6 | 15 | 25 | 50 | 75 | 100

Number of uplink resource blocks, specified as a positive scalar integer.

Data Types: `double`

### NTxAnts — Number of transmission antennas

1 (default) | optional | 2 | 4

Number of transmission antennas, specified as a positive scalar integer. Optional. Valid values are 1, 2, and 4.

Data Types: `double`

Data Types: `struct`

**`chs` — PUSCH channel settings**
structure

PUSCH channel settings, specified as a structure with the following fields.

**`NLayers` — Number of transmission layers**
1 (default) | optional | 2 | 3 | 4

Number of transmission layers, specified as 1, 2, 3, or 4. Optional.

Data Types: `double`

Data Types: `struct`

## Output Arguments

**`info` — Information related to PMI reporting**
structure

Information related to PMI reporting, returned as a structure with these fields.

**`k` — Subband size**
scalar integer

Subband size, in resource blocks (RBs), returned as a scalar integer. This parameter is equal to `NULRB`.

Data Types: `double`

**`NSubbands` — Number of subbands for PMI reporting**
scalar integer

Number of subbands for PMI reporting, returned as a scalar integer. This parameter is equal to 1 for wideband PMI.

Data Types: `double`

**`MaxPMI` — Maximum permitted PMI value for the given configuration**
scalar integer

Maximum permitted PMI value for the given configuration, returned as a scalar integer. Valid PMI values range from 0 to `MaxPMI`.

Data Types: `double`

# Version History

**Introduced in R2014a**

## See Also

`lteULPMISelect` | `ltePUSCHPrecode` | `ltePUSCH`

# lteULPMISelect

PUSCH precoder matrix indication calculation

## Syntax

```
pmi = lteULPMISelect(ue,chs,hest,noiseest)
pmi = lteULPMISelect(ue,chs,hest,noiseest,refgrid)
pmi = lteULPMISelect(ue,chs,hest,noiseest,refgrid,cec)
```

## Description

`pmi = lteULPMISelect(ue,chs,hest,noiseest)` performs PUSCH precoder matrix indication (PMI) calculation for given UE-specific settings, `ue`, channel configuration structure, `chs`, channel estimate resource array, `hest`, and receiver noise variance, `noiseest`. The output, `pmi`, is a scalar containing the PMI selected for closed-loop transmission.

`hest` is a 4-D array of size *M*-by-*N*-by-`NRxAnts`-by-`NTxAnts`, where *M* is the number of subcarriers, *N* is the number of SC-FDMA symbols, `NRxAnts` is the number of receive antennas, and `NTxAnts` is the number of transmit antennas.

`noiseest` is a scalar, an estimate of the received noise power spectral density.

`pmi = lteULPMISelect(ue,chs,hest,noiseest,refgrid)` provides an additional input `refgrid`, a 3-D *M*-by-*N*-by-`NTxAnts` array containing known transmitted data symbols in their correct locations. All other locations i.e. DRS Symbols and unknown data symbol locations should be represented by a `NaN`. This is the same array as the additional `refgrid` input described for the `lteULChannelEstimate` function. For PMI selection the symbols in `refgrid` are ignored, but the non-`NaN` RE locations are used as RE locations at which to sample the channel estimate and perform PMI estimation. This approach can be used to provide a `refgrid` containing for example the SRS RE locations created on all `NTxAnts`, allowing for full-rank channel estimation for the purposes of PMI selection when the PUSCH is transmitted with less than full rank.

`pmi = lteULPMISelect(ue,chs,hest,noiseest,refgrid,cec)` accepts channel estimator configuration structure `cec` containing the field `Reference`.

`Reference = 'None'` will generate no internal reference signals, and the PMI estimation can be performed on arbitrary known REs as given by the `refgrid` argument. This approach can be used to provide a `refgrid` containing for example the SRS signals created on all `NTxAnts`, allowing for full-rank PMI estimation for the purposes of PMI selection when the PUSCH is transmitted with less than full rank. `Reference = 'Antennas'` or `Reference = 'Layers'` will use the PUSCH DMRS RE indices as reference locations for PMI estimation; additional references can still be provided in `refgrid`.

## Examples

### Calculate PUSCH PMI

This example creates an empty resource grid for RMC A3-2 and amend it for MIMO configuration.

Initialize ue specific parameter structure and create an empty resource grid for RMC A3-2 and amend it for MIMO configuration.

```
ue = lteRMCUL('A3-2');
ue.NTxAnts = 4;
ue.PUSCH.NLayers = 2;
rgrid = lteULResourceGrid(ue);
rgrid(ltePUSCHDRSIndices(ue,ue.PUSCH)) = ltePUSCHDRS(ue,ue.PUSCH);
```

Generate modulated waveform.

```
txWaveform = lteSCFDMAModulate(ue,rgrid);
```

Configure a fading channel.

```
chcfg.Seed = 100;
chcfg.DelayProfile = 'EPA';
chcfg.NRxAnts = 2;
chcfg.InitTime = 100;
chcfg.InitPhase = 'Random';
chcfg.ModelType = 'GMEDS';
chcfg.NTerms = 16;
chcfg.NormalizeTxAnts = 'On';
chcfg.NormalizePathGains = 'On';
chcfg.DopplerFreq = 50.0;
chcfg.MIMOCorrelation = 'Low';
chcfg.SamplingRate = 15360000;
```

Filter the transmit waveform through a fading channel and perform SC-FDMA demodulation.

```
rxWaveform = lteFadingChannel(chcfg,txWaveform);
rxSubframe = lteSCFDMADemodulate(ue,rxWaveform);
```

Estimate the corresponding channel and the noise power spectral density on the reference signal subcarriers.

```
cec = struct('FreqWindow',12,'TimeWindow',1,'InterpType','cubic');
cec.PilotAverage = 'UserDefined';
cec.Reference = 'Antennas';

[hest,noiseEst] = lteULChannelEstimate(ue,ue.PUSCH,cec,rxSubframe);
```

Use this estimate to calculate the precoder matrix indication (PMI).

```
pmi = lteULPMISelect(ue,ue.PUSCH,hest,noiseEst)
```

```
pmi = 4
```

## Input Arguments

### ue — UE-specific settings

scalar structure

UE-specific settings, specified as a scalar structure with the following fields.

### NULRB — Number of uplink (UL) resource blocks (RBs)

scalar integer

Number of uplink (UL) resource blocks (RBs), specified as a scalar integer.

Data Types: `double`

**`CyclicPrefixUL` — Cyclic prefix length**
`'Normal'` (default) | optional | `'Extended'`

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

**`NTxAnts` — Number of transmission antennas**
1 (default) | optional | 2 | 4

Number of transmission antennas, specified as 1, 2, or 4.

Data Types: `double`

Data Types: `struct`

**`chs` — Channel configuration structure**
scalar structure

Channel configuration structure, specified as a scalar structure with the following fields.

**`PRBSet` — Physical Resource Block indices**
numeric column matrix

Physical Resource Block indices, specified as a numeric column matrix. `PRBSet` can be a 1- or 2-column matrix, containing the 0-based Physical Resource Block indices (PRBs) corresponding to the resource allocations for this PUSCH.

Data Types: `double`

**`NLayers` — Number of transmission layers**
1 (default) | optional | 2 | 3 | 4

Number of transmission layers, specified as 1, 2, 3, or 4.

Data Types: `double`

Data Types: `struct`

**`hest` — Channel estimate**
4-D numeric array

Channel estimate, specified as a 4-D numeric array of size *M*-by-*N*-by-`NRxAnts`-by-`NTxAnts`. *M* is the number of subcarriers, *N* is the number of SC-FDMA symbols, `NRxAnts` is the number of receive antennas and `NTxAnts` is the number of transmit antennas.

Data Types: `double`
Complex Number Support: Yes

**`noiseest` — Receiver noise variance**
numeric scalar

Receiver noise variance, specified as a numeric scalar. It is an estimate of the received noise power spectral density.

Data Types: `double`

**`refgrid` — Transmitted data symbols**
3-D numeric array

Transmitted data symbols, specified as a 3-D numeric array. `refgrid` is an *M*-by-*N*-by-`NTxAnts` array containing known symbols in their correct locations.

Data Types: `double`
Complex Number Support: Yes

**`cec` — Channel estimator configuration**
scalar structure

Channel estimator configuration, specified as a scalar structure with the following fields.

**`Reference` — Point of reference (indices to internally generate) for PMI estimation**
`'Antennas'` (default) | optional | `'Layers'` | `'None'`

Point of reference (indices to internally generate) for PMI estimation. `Reference = 'None'` generates no internal reference signals, and the PMI estimation can be performed on arbitrary known REs as given by the `refgrid` argument. `Reference = 'Antennas'` or `Reference = 'Layers'` uses the PUSCH DMRS RE indices as reference locations for PMI estimation; additional references can still be provided in `refgrid`.

Data Types: `char` | `string`

Data Types: `struct`

## Output Arguments

**`pmi` — Precoder matrix indication selected for closed-loop transmission**
numeric scalar (0...23)

Precoder matrix indication selected for closed-loop transmission, returned as a numeric scalar between 0 and 23.

# Version History

**Introduced in R2014a**

## See Also

`ltePUSCH` | `ltePUSCHPrecode` | `lteULPMIInfo`

# lteULPerfectChannelEstimate

Uplink perfect channel estimation

## Syntax

```
hest = lteULPerfectChannelEstimate(ue,channel)
hest = lteULPerfectChannelEstimate(ue,channel,offset)

hest = lteULPerfectChannelEstimate(ue,chs,channel)
hest = lteULPerfectChannelEstimate(ue,chs,channel,offset)
```

## Description

`hest = lteULPerfectChannelEstimate(ue,channel)` performs perfect channel estimation for a system configuration given user-equipment-specific (UE-specific) settings `ue` and propagation channel configuration `channel`. The perfect channel estimates are produced only for fading channel models created using the `lteFadingChannel` function.

This function provides a perfect multiple-input-multiple-output (MIMO) channel estimate after single-carrier frequency-division multiple access (SC-FDMA) modulation. To obtain this estimate, the function sets the channel with the specified configuration and sends a set of known symbols through that channel for each transmit antenna in turn.

`hest = lteULPerfectChannelEstimate(ue,channel,offset)` performs perfect channel estimation for the timing and frequency offset specified by `offset`. Specifying `offset` guarantees that `hest` is the channel that results when the receiver is precisely synchronized.

`hest = lteULPerfectChannelEstimate(ue,chs,channel)` performs perfect channel estimation for channel transmission configuration `chs`. This syntax supports SC-FDMA for LTE, single-tone narrowband Internet of Things (NB-IoT), and multitone NB-IoT.

`hest = lteULPerfectChannelEstimate(ue,chs,channel,offset)` performs perfect channel estimation for the channel transmission configuration and the specified timing and frequency offset.

## Examples

### Perform Uplink Perfect Channel Estimation

Perform uplink perfect channel estimation for a chosen propagation channel configuration.

Initialize UE-specific settings, specifying fields appropriate for an LTE uplink configuration.

```
ue.NULRB = 6;
ue.CyclicPrefixUL = 'Normal';
ue.NTxAnts = 2;
ue.TotSubframes = 1;
```

Specify propagation channel conditions.

```
channel.Seed = 1;
channel.DelayProfile = 'EPA';
```

```
channel.NRxAnts = 4;
channel.DopplerFreq = 5.0;
channel.MIMOCorrelation = 'Low';
channel.InitPhase = 'Random';
channel.InitTime = 0.0;
channel.ModelType = 'GMEDS';
channel.NTerms = 16;
channel.NormalizeTxAnts = 'On';
channel.NormalizePathGains = 'On';
```

Perform uplink perfect channel estimation and display the dimension of the channel estimate array.

```
hest = lteULPerfectChannelEstimate(ue,channel);
disp(size(hest));
```

```
    72    14     4     2
```

### Perform Uplink Perfect Channel Estimation on Time Offset Waveform

Perform uplink perfect channel estimation on a time offset waveform passed through a fading channel.

#### Configuration Initialization

Initialize UE-specific settings by specifying fields appropriate for an LTE uplink configuration.

```
ue = lteRMCUL('A1-1','FDD',1);
ue.NULRB = 10;
ue.CyclicPrefixUL = 'Normal';
ue.NTxAnts = 4;
ue.TotSubframes = 1;
```

Specify the propagation channel configuration.

```
channel.Seed = 1;
channel.DelayProfile = 'EVA';
channel.NRxAnts = 2;
channel.DopplerFreq = 5.0;
channel.MIMOCorrelation = 'UplinkMedium';
channel.InitPhase = 'Random';
channel.InitTime = 0.0;
channel.ModelType = 'GMEDS';
channel.NTerms = 16;
channel.NormalizeTxAnts = 'On';
channel.NormalizePathGains = 'On';
```

#### Waveform Processing

Create a waveform and add samples for channel delay.

```
[txWaveform,txgrid,rmcCfg] = lteRMCULTool(ue,[1;0;0;1]);
txWaveform = [txWaveform; zeros(25,4)];
channel.SamplingRate = rmcCfg.SamplingRate;
```

Pass the waveform through a fading channel, generating time-domain receiver samples.

```
rxWaveform = lteFadingChannel(channel,txWaveform);
```

### Determine Timing Offset

Use the `lteULFrameOffset` function to estimate time offset.

```
offset = lteULFrameOffset(ue,ue.PUSCH,rxWaveform);
disp(offset);
```

```
     8
```

Modify the received waveform to account for the timing offset.

```
rxWaveform = rxWaveform(1+offset:end,:);
```

### Demodulation and Uplink Perfect Channel Estimation

Generate frequency-domain receiver data by demodulating the received time-domain waveform.

```
grid = lteSCFDMADemodulate(ue,rxWaveform);
```

Perform uplink perfect channel estimation with the specified time offset.

```
hest = lteULPerfectChannelEstimate(ue,channel,offset);
disp(size(hest));
```

```
   120    14     2     4
```

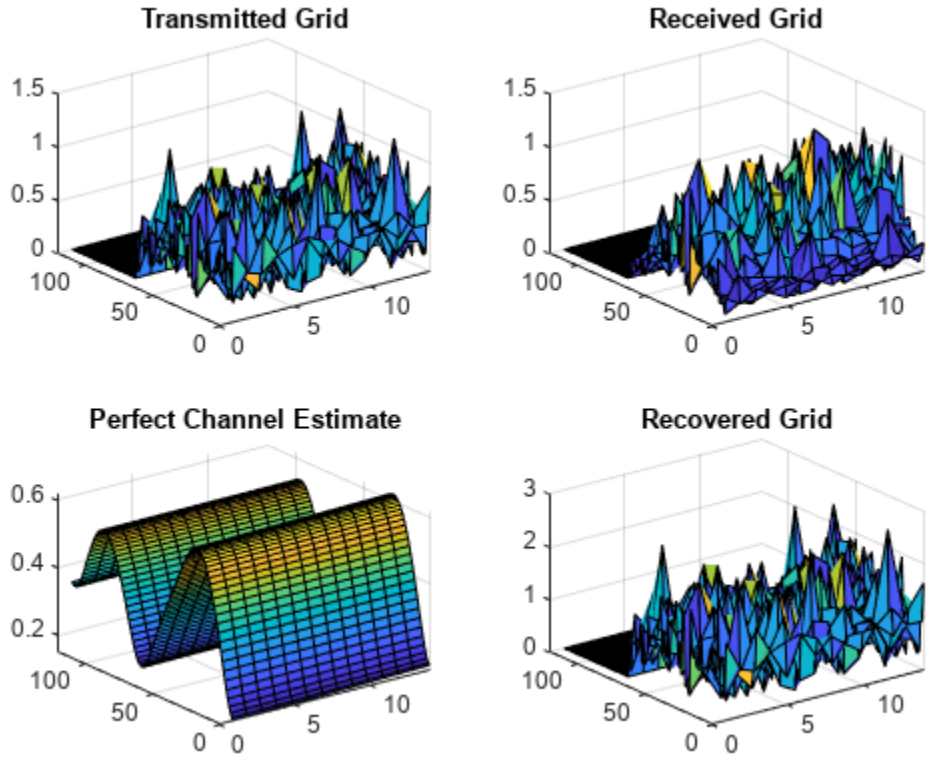
### Visualize Effect of Fading Channel

Plot resource element grids to show the impact of the fading channel on the transmitted signal and recovery of the signal using the perfect channel estimate.

The output channel estimate is a 4-D array. The input specified ten resource blocks leading to 120 subcarriers per symbol. Normal cyclic prefix results in 14 symbols per subframe. The third and fourth dimensions represent the two receive and four transmit antennas specified in the input configuration structures.

Comparing the transmitted grid to the recovered grid shows how equalization of the received grid with the perfect channel estimate recovers the transmission.

```
recoveredgrid = grid./hest;

subplot(2,2,1)
surf(abs(txgrid(:,:,1,1)))
title('Transmitted Grid')
subplot(2,2,2)
surf(abs(grid(:,:,1,1)))
title('Received Grid')
subplot(2,2,3)
surf(abs(hest(:,:,1,1)))
title('Perfect Channel Estimate')
subplot(2,2,4)
surf(abs(recoveredgrid(:,:,1,1)))
title('Recovered Grid')
```




### Perform Uplink Perfect Channel Estimation for NB-IoT configuration

Perform uplink perfect channel estimation for a chosen propagation channel configuration.

Initialize UE-specific settings, specifying fields appropriate for an NB-IoT uplink configuration.

```
ue.NBULSubcarrierSpacing = '15kHz';
ue.TotSlots = 10;
```

Specify propagation channel conditions.

```
channel.Seed = 5;
channel.DelayProfile = 'EPA';
channel.NRxAnts = 2;
channel.DopplerFreq = 5.0;
channel.MIMOCorrelation = 'Low';
channel.InitPhase = 'Random';
channel.InitTime = 0.0;
channel.ModelType = 'GMEDS';
channel.NTerms = 16;
channel.NormalizeTxAnts = 'On';
channel.NormalizePathGains = 'On';
```

Specify NPUSCH configuration information.

```
chs.NBULSubcarrierSet = 0;
chs.Modulation = 'QPSK';
```

```
chs.NULSlots = 2;
chs.NRU = 2;
chs.NRep = 1;
chs.SlotIdx = 0;
```

Perform uplink perfect channel estimation and display the dimension of the channel estimate array.

```
hest = lteULPerfectChannelEstimate(ue,chs,channel);
disp(size(hest));
```

```
    12    70     2
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific settings, specified as a structure. The fields you specify in `ue` determine whether the function performs channel estimation for an LTE or NB-IoT configuration. To indicate an LTE configuration, specify the `NULRB` field. To indicate an NB-IoT configuration, specify the `NBULSubcarrierSpacing` field. The `NTxAnts` field is required for both LTE and NB-IoT configurations. The other fields in `ue` are optional. The `CyclicPrefixUL` and `TotSubframes` fields are applicable only for an LTE configuration. The `TotSlots` field is applicable only for an NB-IoT configuration.

### NULRB — Number of uplink resource blocks

integer in the interval [6, 110]

Number of uplink resource blocks, $N_{RB}^{UL}$, specified as an integer in the interval [6, 110]. To perform channel estimation for an LTE configuration, you must specify this field.

Data Types: `double`

### CyclicPrefixUL — Cyclic prefix length

`'Normal'` (default) | `'Extended'`

Cyclic prefix length, specified as `'Normal'` or `'Extended'`. This field is optional.

#### Dependencies

This field applies only when you choose an LTE configuration by specifying the `NULRB` field.

Data Types: `char`

### NTxAnts — Number of transmit antennas

1 (default) | 2 | 4

Number of transmit antennas, $N_{TX}$, specified as 1, 2, or 4.

Data Types: `double`

### TotSubframes — Total number of subframes to generate

1 (default) | nonnegative integer

Total number of subframes to generate, specified as a nonnegative integer.

Data Types: `double`

**`NBULSubcarrierSpacing` — NB-IoT uplink subcarrier spacing**
`'3.75kHz'` | `'15kHz'`

NB-IoT uplink subcarrier spacing, specified as `'3.75kHz'` or `'15kHz'`. To set a subcarrier spacing of 3.75 kHz, specify `NBULSubcarrierSpacing` as `'3.75kHz'`. To set a subcarrier spacing of 15 kHz, specify `NBULSubcarrierSpacing` as `'15kHz'`.

To perform channel estimation for an NB-IoT configuration, you must specify this field. To indicate an LTE configuration, omit this field.

Data Types: `char`

**`TotSlots` — Total number of slots to generate**
1 (default) | nonnegative integer

Total number of slots to generate, specified as a nonnegative integer.

Data Types: `double`

Data Types: `struct`

**`channel` — Propagation channel configuration structure**
structure

Propagation channel configuration, specified as a structure. This argument must contain all the fields required to parameterize the fading channel model, that is, to call the `lteFadingChannel` function.

---

**Note** Before execution of the channel, `lteULPerfectChannelEstimate` sets the `SamplingRate` field internally to the sampling rate of the time domain waveform passed to the `lteFadingChannel` function for filtering. Therefore, this `channel` input does not require the `SamplingRate` field. If one is included, it is not used.

---

**`NRxAnts` — Number of receive antennas**
positive integer

Number of receive antennas, $N_{RX}$, specified as a positive integer.

Data Types: `double`

**`MIMOCorrelation` — Correlation between UE and eNodeB antennas**
`'Low'` | `'Medium'` | `'UplinkMedium'` | `'High'` | `'Custom'`

Correlation between UE and Evolved Node B (eNodeB) antennas, specified as one of these values:

- `'Low'` – No correlation between antennas
- `'Medium'` – Correlation level is applicable to tests defined in TS 36.101 [1]
- `'UplinkMedium'` – Correlation level is applicable to tests defined in TS 36.104 [2]
- `'High'` – Strong correlation between antennas
- `'Custom'` – Apply user-defined `TxCorrelationMatrix` and `RxCorrelationMatrix`

Data Types: `char` | `string`

**`NormalizeTxAnts` — Transmit antenna number normalization**
`'On'` (default) | `'Off'`

Transmit antenna number normalization, specified as `'On'` or `'Off'`. If you specify `NormalizeTxAnts` as `'On'`, `lteULPerfectChannelEstimate` normalizes the model output by $1/\sqrt{N_{TX}}$. Normalization by the number of transmit antennas ensures that the output power per receive antenna is unaffected by the number of transmit antennas. If you specify `NormalizeTxAnts` as `'Off'`, `lteULPerfectChannelEstimate` does not perform normalization. This field is optional.

Data Types: `char` | `string`

**`DelayProfile` — Delay profile model**
`'EPA'` | `'EVA'` | `'ETU'` | `'Custom'` | `'Off'`

Delay profile model, specified as `'EPA'`, `'EVA'`, `'ETU'`, `'Custom'`, or `'Off'`. For more information, see “Propagation Channel Models”.

Setting `DelayProfile` to `'Off'` switches off fading completely and implements a static MIMO channel model. In this case, the antenna geometry corresponds to the `MIMOCorrelation` and `NRxAnts` fields, and the number of transmit antennas. The temporal part of the model for each link between transmit and receive antennas consists of a single path with zero delay and constant unit gain.

Data Types: `char` | `string`

**`DopplerFreq` — Maximum Doppler frequency**
nonnegative scalar

Maximum Doppler frequency, in Hz, specified as a nonnegative scalar.

**Dependencies**

This field applies only when you specify the `DelayProfile` field as a value other than `'Off'`.

Data Types: `double`

**`SamplingRate` — Sampling rate of input signal**
nonnegative scalar

Sampling rate of input signal, specified as a nonnegative scalar.

**Dependencies**

This field applies only when you specify the `DelayProfile` field as a value other than `'Off'`.

Data Types: `double`

**`InitTime` — Fading process time offset**
nonnegative scalar

Fading process time offset, in seconds, specified as a nonnegative scalar.

**Dependencies**

This field applies only when you specify the `DelayProfile` field as a value other than `'Off'`.

Data Types: `double`

**`NTerms` — Number of oscillators used in fading path modeling**
16 (default) | power of two

Number of oscillators used in fading path modeling, specified as a power of two. This field is optional

**Dependencies**

This field applies only when you specify the `DelayProfile` field as a value other than `'Off'`.

Data Types: `double`

**`ModelType` — Rayleigh fading model type**
`'GMEDS'` (default) | `'Dent'`

Rayleigh fading model type, specified as `'GMEDS'` or `'Dent'`. To model Rayleigh fading using the generalized method of exact Doppler spread (GMEDS) described in [4], specify `ModelType` as `'GMEDS'`. To model Rayleigh fading using the modified Jakes fading model described in [3], specify `ModelType` as `'Dent'`. This field is optional.

**Note** Specifying `ModelType` as `'Dent'` is not recommended.

**Dependencies**

This field applies only when you specify the `DelayProfile` field as a value other than `'Off'`.

Data Types: `char` | `string`

**`NormalizePathGains` — Model output normalization indicator**
`'On'` (default) | `'Off'`

Model output normalization indicator, specified as `'On'` or `'Off'`. To normalize the model output such that the average power is unity, specify `NormalizePathGains` as `'On'`. To return the average output power as the sum of the powers of the taps of the delay profile, specify `NormalizePathGains` as `'Off'`. This field is optional.

**Dependencies**

This field applies only when you specify the `DelayProfile` field as a value other than `'Off'`.

Data Types: `char` | `string`

**`InitPhase` — Phase initialization for sinusoidal components of model**
`'Random'` (default) | real-valued scalar | 4-D array

Phase initialization for the sinusoidal components of the model, specified as one of these values:

- `'Random'` – Randomly initialize the phases according to the value you specify in the `Seed` field
- A real-valued scalar – Specify the single initial value of the phases of all components, in radians
- An $N$-by-$L$-by-$N_{TX}$-by-$N_{RX}$ array – Explicitly initialize the phase, in radians, of each component. In this case, $N$ is the number of phase initialization values per path and $L$ is the number of paths

**Note**

- When you specify `ModelType` as `'GMEDS'`, $N = 2 \times$`NTerms`.

- When you specify `ModelType` as `'Dent'`, $N$ = `NTerms`.

Data Types: `double` | `char` | `string`

**Seed — Random number generator seed**
real-valued scalar

Random number generator seed, specified as a real-valued scalar. To use a random seed, specify `Seed` as `0`.

**Note** Seed values in the interval $[0, 2^{31} - 1 - (K(K-1)/2)]$, where $K = N_{TX} \times N_{RX}$ and is the product of the number of transmit and receive antennas, are recommended. Seed values outside of this interval are not guaranteed to give distinct results.

**Dependencies**

This field applies only when you specify the `DelayProfile` field as a value other than `'Off'` and the `InitPhase` field as `'Random'`.

Data Types: `double`

**`AveragePathGaindB` — Average gains of the discrete paths**
real-valued vector

Average gains of the discrete paths, in dB, specified as a real-valued vector.

**Dependencies**

This field applies only when you specify the `DelayProfile` field as `'Custom'`.

Data Types: `double`

**`PathDelays` — Delays of discrete paths**
real-valued vector

Delays of the discrete paths, in seconds, specified as a real-valued vector.

**Dependencies**

This field applies only when you specify the `DelayProfile` field as `'Custom'`.

Data Types: `double`

**`TxCorrelationMatrix` — Correlation between each of the transmit antennas**
$N_{TX}$-by-$N_{TX}$ complex-valued matrix

Correlation between each of the transmit antennas, specified as an $N_{TX}$-by-$N_{TX}$ complex-valued matrix.

**Dependencies**

This field applies only when you specify the `MIMOCorrelation` field as `'Custom'`.

Data Types: `double`
Complex Number Support: Yes

**`RxCorrelationMatrix` — Correlation between each of the receive antennas**
$N_{RX}$-by-$N_{RX}$ complex-valued matrix

Correlation between each of the receive antennas, specified as an $N_{RX}$-by-$N_{RX}$ complex-valued matrix.

**Dependencies**

This field applies only when you specify the `MIMOCorrelation` field as `'Custom'`.

Data Types: `double`
Complex Number Support: Yes

Data Types: `struct`

**`offset` — Timing offset**
nonnegative integer

Timing offset, in samples, specified as a nonnegative integer. The timing offset is specified from the start of the output of the channel to the estimated SC-FDMA demodulation starting point. Specify the timing offset, when known, to obtain the perfect channel estimate as seen by a synchronized receiver. Use the `lteULFrameOffset` function to derive the value for `offset`.

Data Types: `double`

**`chs` — NPUSCH information**
structure

NPUSCH information, specified as a structure. For an NB-IoT configuration, you can set additional uplink-specific parameters by specifying the NB-IoT-specific fields in `chs`. Except for the `NBULSubcarrierSet` field, the fields in `chs` are applicable either when `ue.NBULSubcarrierSpacing` is `'3.75kHz'` or when `ue.NBULSubcarrierSpacing` is `'15kHz'` and `length(NBULSubcarrierSet)` is `1`.

**`NBULSubcarrierSet` — NB-IoT uplink subcarrier indices**
vector of nonnegative integers (default) | nonnegative integer

NB-IoT uplink subcarrier indices, specified as a vector of nonnegative integers in the interval [0, 11] or a nonnegative integer in the interval [0, 47]. The indices are in zero-based form. To use `lteULPerfectChannelEstimate` for a single-tone NB-IoT configuration, you must specify `NBULSubcarrierSet` as a scalar. If you do not specify `NBULSubcarrierSet`, `lteULPerfectChannelEstimate` returns an estimate for a multi-tone NB-IoT configuration by default.If you specify `ue.NBULSubcarrierSpacing` as `'15kHz'`, this field is required.

Data Types: `double`

**`Modulation` — Modulation type**
`'BPSK'` | `'QPSK'`

Modulation type, specified as `'BPSK'` or `'QPSK'`. For binary phase shift keying (BPSK), specify `Modulation` as `'BPSK'`. For quadrature phase shift keying (QPSK), specify `Modulation` as `'QPSK'`.

Data Types: `char`

**`NULSlots` — Number of slots per resource unit**
positive integer

Number of slots per resource unit (RU), specified as a positive integer. To use `lteULPerfectChannelEstimate` for a single-tone NB-IoT configuration, you must specify this field.

Data Types: `double`

**`NRU` — Number of RUs**
positive integer

Number of RUs, specified as a positive integer. To use `lteULPerfectChannelEstimate` for a single-tone NB-IoT configuration, you must specify this field.

Data Types: `double`

**`NRep` — Number of repetitions for codeword**
nonnegative integer

Number of repetitions for a codeword, specified as a nonnegative integer. To use `lteULPerfectChannelEstimate` for a single-tone NB-IoT configuration, you must specify this field.

Data Types: `double`

**`SlotIdx` — Relative slot index in an NPUSCH bundle**
`0` (default) | nonnegative integer

Relative slot index in an NPUSCH bundle, specified as a nonnegative integer. This field determines the zero-based relative slot index in a bundle of time slots for transmission of a transport block or control information bit. This field is optional.

Data Types: `double`

## Output Arguments

**`hest` — Perfect channel estimate**
complex-valued 4-D array

Perfect channel estimate, returned as an $N_{SC}$-by-$N_{SYM}$-by-$N_{RX}$-by-$N_{TX}$ complex-valued array, where $N_{SC}$ is the number of subcarriers and $N_{SYM}$ is the number of SC-FDMA symbols.

Data Types: `double`
Complex Number Support: Yes

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.101. “User Equipment (UE) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*.

[2] 3GPP TS 36.104. “Base Station (BS) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*.

[3] Dent, P., Bottomley, G. E., and Croft, T. “Jakes Fading Model Revisited.” *Electronics Letters*. Vol. 29, Number 13, 1993, pp. 1162–1163.

[4] Pätzold, M., Wang, C., and Hogstad, B. O. “Two New Sum-of-Sinusoids-Based Methods for the Efficient Generation of Multiple Uncorrelated Rayleigh Fading Waveforms.” *IEEE Transactions on Wireless Communications*. Vol. 8, Number 6, 2009, pp. 3122–3131.

## See Also

`lteULChannelEstimate` | `lteULChannelEstimatePUCCH1` | `lteULChannelEstimatePUCCH2` | `lteULChannelEstimatePUCCH3` | `lteDLPerfectChannelEstimate`

# lteULPrecode

SC-FDMA precoding

## Syntax

```
out = lteULPrecode(in,nrb)
out = lteULPrecode(in,n,resourceType)
```

## Description

`out = lteULPrecode(in,nrb)` performs SC-FDMA precoding of complex modulation symbols `in` for PUSCH or NPUSCH configuration with a bandwidth of `nrb` resource blocks.

`out = lteULPrecode(in,n,resourceType)` performs SC-FDMA precoding of the complex modulation symbols `in` for PUSCH or NPUSCH configuration with a bandwidth of `n` resource blocks or subcarriers.

## Examples

### Perform SC-FDMA Precoding on Complex Modulation Symbols

UL precoding is a step in the PUSCH processing chain. The chain includes scrambling, symbol mapping, UL precoding, RE mapping, and SC-FDMA modulation.

Create a UE-specific configuration structure, get PUSCH indices, and generate a bit stream sized according to configuration structure.

```
ue = lteRMCUL('A3-2');
[puschInd, info] = ltePUSCHIndices(ue,ue.PUSCH);
ueDim = lteULResourceGridSize(ue);
bits = randi([0,1],info.G,ue.PUSCH.NLayers);
```

Perform scrambling, symbol modulation, and UL precoding.

```
scrBits = lteULScramble(ue,bits);
symbols = lteSymbolModulate(scrBits,ue.PUSCH.Modulation);
precodedSymbols = lteULPrecode(symbols,ue.NULRB);
```

Generate resource mapping grid, populate the grid with the precoded symbols, and perform SC-FDMA modulation.

```
grid = lteULResourceGrid(ue);
grid(puschInd) = precodedSymbols;
[timeDomainSig,infoScfdma] = lteSCFDMAModulate(ue,grid);
```

### Three-Tone NB-IoT Uplink SC-FDMA Processing Chain

Generate an SC-FDMA modulated waveform for a three-tone NB-IoT uplink configuration by applying the SC-FDMA processing chain, comprising symbol mapping, UL precoding, and SC-FDMA modulation.

Specify an NB-IoT configuration with ten slots and a subcarrier spacing of 15 kHz.

```
NSlots = 10;
ue.NBULSubcarrierSpacing = '15kHz';
```

Set the subcarrier locations and generate random bits for transmission.

```
chs.NBULSubcarrierSet = 0:2;
bits = randi([0,1],7*NSlots*length(chs.NBULSubcarrierSet)*2,1);
```

Perform symbol modulation and generate precoded symbols.

```
symbols = lteSymbolModulate(bits,'QPSK');
precodedSymbols = lteULPrecode(symbols,length(chs.NBULSubcarrierSet),'Subcarrier');
```

Generate the narrowband resource array

```
grid = repmat(lteNBResourceGrid(ue),1,NSlots);
grid(chs.NBULSubcarrierSet + 1,:) = reshape(precodedSymbols,length(chs.NBULSubcarrierSet),7*NSlo
```

Generate the SC-FDMA modulated waveform for the specified configuration and display its size.

```
waveform = lteSCFDMAModulate(ue,chs,grid);
size(waveform)
```

```
ans = 1×2

        9600           1
```

### Single-Tone NB-IoT Uplink SC-FDMA Processing Chain

Generate an SC-FDMA modulated waveform for a single-tone NB-IoT uplink configuration by applying the SC-FDMA processing chain, comprising symbol mapping, UL precoding, and SC-FDMA modulation.

Specify an NB-IoT configuration with 16 slots and a subcarrier spacing of 3.75 kHz.

```
NSlots = 16;
ue.NBULSubcarrierSpacing = '3.75kHz';
```

Specify the channel transmission configuration.

```
chs = struct('NULSlots',4,'NRU',1,'NRep',4,'SlotIdx',0, ...
    'Modulation','BPSK','NBULSubcarrierSet',41);
```

Generate random bits for transmission, perform symbol modulation, and generate precoded symbols.

```
bits = randi([0,1],7*NSlots*length(chs.NBULSubcarrierSet),1);
symbols = lteSymbolModulate(bits,chs.Modulation);
precodedSymbols = lteULPrecode(symbols,length(chs.NBULSubcarrierSet),'Subcarrier');
```

Generate the narrowband resource array

```
grid = repmat(lteNBResourceGrid(ue),1,NSlots);
grid(chs.NBULSubcarrierSet+1,:) = reshape(precodedSymbols,length(chs.NBULSubcarrierSet),7*NSlots
```

Generate the SC-FDMA modulated waveform for the specified configuration and display its size.

```
waveform = lteSCFDMAModulate(ue,chs,grid);
size(waveform)
```

```
ans = 1×2

       61440           1
```

## Input Arguments

### `in` — Complex modulation symbols

complex-valued matrix

Complex modulation symbols, specified as an $N_{Sym}$-by-$N_L$ complex-valued matrix. $N_{Sym}$ is the number of symbols and $N_L$ is the number of layers.

Data Types: `double`
Complex Number Support: Yes

### `nrb` — Number of resource blocks

nonnegative integer

Number of resource blocks, specified as a nonnegative integer.

Data Types: `double`

### `n` — Number of resource blocks or subcarriers

nonnegative integer

Number of resource blocks or subcarriers, specified as a nonnegative integer.

#### Dependencies

If the `resourceType` input is `'PRB'`, then `n` is the number of resource blocks. If the `resourceType` is `'Subcarrier'`, then `n` is the number of subcarriers.

Data Types: `double`

### `resourceType` — Resource type

`'PRB'` | `'Subcarrier'`

Resource type, specified as `'PRB'` or `'Subcarrier'`.

Data Types: `char` | `string`

## Output Arguments

### `out` — Precoded PUSCH symbols

complex-valued matrix

Precoded PUSCH symbols, returned as an $N_{Sym}$-by-$N_L$ complex-valued matrix. $N_{Sym}$ is the number of symbols, and $N_L$ is the number of layers.

The dimension and size of the input and output symbol matrices are the same.

Data Types: `double`
Complex Number Support: Yes

# Version History

**Introduced in R2014a**

## See Also

`lteULDeprecode` | `lteLayerMap` | `ltePUSCHPrecode` | `ltePUSCH`

# lteULResourceGrid

Uplink subframe resource array

## Syntax

```
grid = lteULResourceGrid(ue)
grid = lteULResourceGrid(ue,p)
```

## Description

`grid = lteULResourceGrid(ue)` returns an empty resource array generated from the UE-specific settings structure `ue`. For more information on the resource grid and the multidimensional array used to represent the resource elements for one subframe across all configured antenna ports, see “Represent Resource Grids”.

`grid = lteULResourceGrid(ue,p)` returns a resource array, where `p` directly specifies the number of antenna planes in the array. In this case, `NTxAnts` is not required as a structure field of `ue`.

## Examples

### Create UL Resource Array

Create an empty resource array representing the resource elements for 10 MHz bandwidth.

```
reGrid = lteULResourceGrid(struct('NULRB',50,'CyclicPrefixUL','Normal','NTxAnts',1));
```

### Create Uplink Subframe Resource Array Using Optional Antenna Plane Input

Create an empty resource array that represents the uplink resource elements for 5 MHz bandwidth, one subframe, extended cyclic prefix, and four antenna ports.

```
cfg = struct('NULRB',25,'CyclicPrefixUL','Extended');
p = 4;
gridul = lteULResourceGrid(cfg,p);
size(gridul)
```

```
ans = 1×3

   300    12     4
```

## Input Arguments

**ue — UE-specific settings**
scalar structure

UE-specific settings, specified as a scalar structure with the following fields.

**`NULRB` — Number of uplink (UL) resource blocks (RBs)**
scalar integer from 6 to 110

Number of uplink (UL) resource blocks (RBs), specified as a scalar integer from 6 to 110.

Data Types: `double`

**`CyclicPrefixUL` — Cyclic prefix length**
`'Normal'` (default) | optional | `'Extended'`

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

**`NTxAnts` — Number of transmission antennas**
1 (default) | optional | 2 | 4

Number of transmission antennas, specified as 1, 2, or 4.

Data Types: `double`

Data Types: `struct`

**`p` — Number of antenna planes in the return array**
scalar integer

Number of antenna planes in the return array, specified as a scalar integer.

Data Types: `double`

## Output Arguments

**`grid` — Empty resource grid**
multidimensional numeric array

Empty resource grid, returned as an *N*-by-*M*-by-*P* numeric array. *N* is the number of subcarriers (12×`NULRB`). *M* is the number of SC-FDMA symbols in a subframe, 14 for normal cyclic prefix and 12 for extended cyclic prefix. *P* is the number of transmission antennas. `grid` is used to represent the resource elements for one subframe across all configured antenna ports.

Data Types: `double`

# Version History

**Introduced in R2014a**

## See Also

`lteULResourceGridSize` | `lteResourceGrid` | `lteDLResourceGrid` | `lteSCFDMAModulate`

# lteULResourceGridSize

Uplink subframe resource array size

## Syntax

```
d = lteULResourceGridSize(ue)
d = lteULResourceGridSize(ue,p)
```

## Description

`d = lteULResourceGridSize(ue)` returns the size of the uplink resource array generated from UE-specific settings `ue`. For more information on the resource grid and the multidimensional array used to represent the resource elements for one subframe across all configured antenna ports, see “Represent Resource Grids”.

`d = lteULResourceGridSize(ue,p)` also specifies the number of antenna planes in the array.

## Examples

### Get Uplink Subframe Resource Array Size

Configure UE-specific settings.

```
cfgul = struct(NULRB=6,NTxAnts=2,CyclicPrefixUL="Normal");
```

Get the uplink subframe resource array size.

```
d = lteResourceGridSize(cfgul);
```

Generate an uplink resource array of the appropriate size.

```
gridul = zeros(d);
```

### Get Uplink Subframe Resource Array Size for Specified Antenna Planes

Configure UE-specific settings.

```
ue = struct(NULRB=25,CyclicPrefixUL="Normal");
```

Get the uplink subframe resource array size for the specified configuration and four antenna planes.

```
p = 4;
d = lteResourceGridSize(ue,p);
```

Create a resource array of the appropriate size.

```
gridul = zeros(d);
```

## Input Arguments

### `ue` — UE-specific settings

structure

UE-specific settings, specified as a structure containing these fields.

### `NULRB` — Number of uplink RBs

integer in the interval [6, 110]

Number of uplink resource blocks (RBs), specified as an integer in the interval [6, 110].

Data Types: `double`

### `CyclicPrefixUL` — Cyclic prefix length

`"Normal"` (default) | `"Extended"`

Cyclic prefix length, specified as `"Normal"` or `"Extended"`.

Data Types: `char` | `string`

### `NTxAnts` — Number of transmission antennas

1 (default) | 2 | 4

Number of transmission antennas, specified as 1, 2, or 4. If you use the syntax containing the `p` input, the function ignores this field and uses the `p` input value instead.

Data Types: `double`

Data Types: `struct`

### `p` — Number of antenna planes in uplink resource array

positive integer

Number of antenna planes in the uplink resource array, specified as a positive integer.

Data Types: `double`

## Output Arguments

### `d` — Uplink resource array size

three-element row vector

Uplink resource array size, returned as a three-element row vector of the form [*N M P*].

- *N* is the number of subcarriers, which is given by $12 \times N_{RB}$, where $N_{RB}$ is the number of uplink RBs.
- *M* is the number of SC-FDMA symbols in a subframe: 14 for normal cyclic prefix and 12 for extended cyclic prefix.
- *P* is the number of transmission antennas.

Data Types: `double`

## Version History

**Introduced in R2014a**

## See Also

`lteULResourceGrid` | `lteResourceGridSize` | `lteDLResourceGridSize`

# lteULSCH

Uplink shared channel

## Syntax

```
[cwout,chinfo] = lteULSCH(ue,chs,trblkin)
[cwout,chinfo] = lteULSCH(ue,chs,trblkin,opts)
[cwout,chinfo] = lteULSCH(ue,chs,trblkin,cqi,ri,ack,opts)
```

## Description

`[cwout,chinfo] = lteULSCH(ue,chs,trblkin)` performs complete UL-SCH transport coding and UCI coding on the input information bits, `trblkin`, and returns the complete codewords in the output, `cwout`. It encodes both a single transport block or pair of blocks, contained in a cell array, for the case of spatial multiplexing schemes transmitting two codewords, represented by input `trblkin` without any UCI data. The lowest order information bit of `trblkin` should be mapped to the most significant bit of the transport block, as defined in TS 36.321 Section 6.1.1 [3]. The encoding process also includes the channel interleaving. The transport encoding includes type-24A CRC calculation, code block segmentation and type-24B CRC attachment, turbo encoding, rate matching, block concatenation, and channel interleaving. For more information, see TS 36.212 Sections 5.2.2.1 to 5.2.2.5 and 5.2.2.8 [2]. Parameter information relating to the underlying UL-SCH and UCI coding is available in structure `chinfo`.

The output `chinfo` is a structure containing information related to the UL-SCH coding process.

For multiple transport blocks, each structure in the array corresponds to one of the blocks. This output is also available from the `lteULSCHInfo` function.

`[cwout,chinfo] = lteULSCH(ue,chs,trblkin,opts)` allows for the merging of the input `chs` structure fields into `chinfo` at the output.

If the UL-SCH encoding is for a retransmission of a previously sent transport block, use the three “Init” fields, `'InitPRBSet'`, `'InitCyclicPrefixUL'`, and `'InitShortened'`. If any of these fields are absent, their values are assumed to be the same as the values for the associated current subframe fields, `'PRBSet'`, `'CyclicPrefixUL'`, and `'Shortened'`.

`opts` is an optional input parameter which enables the concatenation or merging of the `chs` input structure fields into the fields returned by `chinfo`. This parameter is useful for combining the high-level configuration parameters with the fine-grained ones used in the encoding process.

`opts` allows additional control of the contents and format of the `chinfo` output.

`[cwout,chinfo] = lteULSCH(ue,chs,trblkin,cqi,ri,ack,opts)` encodes and multiplexes the UCI input data, CQI, RI, and ACK, along with the information bits, `trblkin`, in the codeword, `cwout`. For more information, see TS 36.212 Sections 5.2.2.6 to 5.2.2.8 [2]. Any of the `trblkin`, `cqi`, `ri`, or `ack` vectors can be empty if that data is not present. If `trblkin` is empty, only UCI on UL-SCH/PUSCH is processed, according to TS 36.212 Section 5.2.4 [2]. The coding of the UCI can be controlled through the additional fields, `BetaACK`, `BetaCQI`, `BetaRI`, and `NBundled`, in the `chs` input structure. Setting `NBundled` to 0 disables the TDD HARQ-ACK bundling scrambling; therefore, it is off by default.

## Examples

### Generate UL-SCH Codewords

Create coded information bits for a 3 MHz, Uplink A3-3 FRC.

Create an UL RMC configuration structure. Initialize the optional fields for a ue-specific setting structure. Default settings are used if you don't initial optional fields. Create a transport block bit stream, `trBlk`.

```
rmc = lteRMCUL('A3-3');
ue.CyclicPrefixUL = 'Normal';
ue.Shortened = 0;
trBlk = randi([0,1],rmc.PUSCH.TrBlkSizes(1),1);
```

Generate UL-SCH codewords for the A3-3 FRC.

```
cw = lteULSCH(ue,rmc.PUSCH,trBlk);
```

## Input Arguments

### `ue` — UE-specific settings

structure

UE-specific settings, specified as a structure with the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`CyclicPrefixUL`** | Optional | `'Normal'` (default), `'Extended'` | Current cyclic prefix length |
| **`Shortened`** | Optional | 0 (default), 1 | Option to shorten the subframe by omitting the last symbol, specified as 0 or 1. If `1`, the last symbol of the subframe is not used. For subframes with possible SRS transmission, set `Shortened` to 1 to maintain a standard compliant configuration. |

### `chs` — Channel-specific transmission configuration

scalar structure

Channel-specific transmission configuration, specified as a structure that can contain the following parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`Modulation`** | Required | `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'` | Modulation type, specified as a character vector, cell array of character vectors, or string array. If blocks, each cell is associated with a transport block. |
| **`NLayers`** | Optional | 1 (default), 2, 3, 4 | Total number of transmission layers associated with the transport block or blocks. |
| **`RV`** | Required | Integer vector (0,1,2,3). A one or two column matrix (for one or two codewords). | Specifies the redundancy version for one or two codewords used in the initial subframe number, `NSubframe`. This parameter field is only for informational purposes and is read-only.<br><br>This parameter field is not required if `trblkin` is `[]`, which signifies that the UL-SCH is carrying only UCI and no transport data. |
| **`PRBSet`** | Required | Integer column vector or two-column matrix | 0-based physical resource block indices (PRBs) for the slots of the current PUSCH resource allocation. As a column vector, the resource allocation is the same in both slots of the subframe. As a two-column matrix, it specifies different PRBs for each slot in a subframe. |
| The following three `'Init'` fields should be used if the UL-SCH encoding is for a retransmission of a previously sent transport block. If any of these fields are absent, its value is assumed to be the same as the value for its associated current subframe field. | | | |
| **`InitPRBSet`** | Optional | 1- or 2-column integer matrix, `PRBSet` (default) | PRB indices used in the initial transmission PUSCH allocation. If this field is absent, its value is assumed to be the same as the value for the associated current subframe field, `PRBSet`. |
| **`InitCyclicPrefixUL`** | Optional | `'Normal'`, `'Extended'`, `CyclicPrefixUL` (default) | Cyclic prefix length of initial transmit subframe. This is the length used during the first transmission of this transport block. If this field is absent, its value is assumed to be the same as the value for the associated current subframe field, `CyclicPrefixUL`. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`InitShortened`** | Optional | 0, 1, `Shortened` (default) | Initial transmit subframe shortened flag. If 1, the initial transmit subframe was shortened for possible SRS. If this field is absent, its value is assumed to be the same as the value for the associated current subframe field, `Shortened`. |
| The coding of the UCI can be controlled through the following additional fields. | | | |
| **`BetaCQI`** | Optional | numeric scalar, 2.0 (default) | Modulation and coding scheme (MCS) offset for CQI and PMI bits |
| **`BetaRI`** | Optional | numeric scalar, 2.0 (default) | Modulation and coding scheme (MCS) offset for RI bits |
| **`BetaACK`** | Optional | numeric scalar, 2.0 (default) | Modulation and coding scheme (MCS) offset for HARQ-ACK bits. This field was previously named `BetaHI`; if this field is absent but `BetaHI` is present, it is used as before. |
| **`NBundled`** | Optional | 0 (default), 1, ..., 9 | TDD HARQ-ACK bundling scrambling sequence index. When set to 0, the function disables the TDD HARQ-ACK bundling scrambling. Therefore, it is off by default. |

**`trblkin` — Input transport blocks**
numeric vector | cell array of numeric vectors

Input transport blocks, specified as a numeric vector or a cell array of numeric vectors.

Data Types: `double` | `cell`

**`opts` — Options to control output contents and format**
character vector | cell array of character vectors | string array

Options to control output contents and format, specified as a character vector, cell array of character vectors, or string array. You may choose any of the option listed in this table. Use double quotes for string.

| Option | Values | Description |
|---|---|---|
| Channel parameter merging | `'nochsconcat'` (default) | Do not concatenate `chs` input structure into `chinfo`. |
| | `'chsconcat'` | Concatenate `chs` input structure into `chinfo`. |
| Output structure format | `'cwseparate'` (default) | Information values for each codeword are output in separate elements of the 1-by-*ncodewords* structure array `chinfo`. |

| Option | Values | Description |
|---|---|---|
| | `'cwcombined'` | Information values for each codeword are combined into the fields of a single scalar, or 1-by-1, structure. |

Data Types: `char` | `string` | `cell`

**`cqi` — CQI input data**
numeric vector

CQI input data, specified as a numeric vector. Part of the UCI data.

Data Types: `double`

**`ri` — RI input data**
numeric vector

RI input data, specified as a numeric vector. Part of the UCI data.

Data Types: `double`

**`ack` — HARQ-ACK input data**
numeric vector

HARQ-ACK input data, specified as a numeric vector. Part of the UCI data.

Data Types: `double`

## Output Arguments

**`cwout` — Complete output codewords**
integer column vector | cell array of integer column vectors

Complete output codewords, returned as an integer column vector or a cell array of integer column vectors.

Data Types: `int8` | `cell`

**`chinfo` — Parameter information relating to the underlying UL-SCH and UCI coding**
structure | structure array

Parameter information relating to the underlying UL-SCH and UCI coding, returned as a structure or a structure array. If two transport blocks are encoded, `chinfo` is a structure array of two elements, one for each block. Alternatively, you can create code block segmentation fields in this structure independently, by calling the `lteULSCHInfo` function. `chinfo` contains the following fields.

| Parameter Field | Description | Values | Data Type |
|---|---|---|---|
| **`C`** | Total number of code blocks | nonnegative scalar integer | `int32` |
| **`Km`** | Lower code block size (*K*–) | nonnegative scalar integer | `int32` |
| **`Cm`** | Number of code blocks of size `Km` (*C*–) | nonnegative scalar integer | `int32` |

| Parameter Field | Description | Values | Data Type |
|---|---|---|---|
| **Kp** | Upper code block size ($K+$) | nonnegative scalar integer | `int32` |
| **Cp** | Number of code blocks of size `Kp` ($C+$) | nonnegative scalar integer | `int32` |
| **F** | Number of filler bits in first block | nonnegative scalar integer | `int32` |
| **L** | Number of segment cyclic redundancy check (CRC) bits | nonnegative scalar integer | `int32` |
| **Bout** | Total number of bits in all segments | nonnegative scalar integer | `int32` |
| **G** | Number of coded and rate matched UL-SCH data bits | nonnegative scalar integer | `int32` |
| **Qm** | Number of bits per symbol | nonnegative scalar integer | `int32` |
| **Gd** | Number of coded and rate matched UL-SCH data symbols ($G'$) | nonnegative scalar integer | `int32` |
| **OCQI** | Number of uncoded channel quality information (CQI) bits | nonnegative scalar integer | `int32` |
| **ORI** | Number of uncoded symbols for RI | nonnegative scalar integer | `int32` |
| **OACK** | Number of uncoded symbols for ACK/NACK | nonnegative scalar integer | `int32` |
| **QdCQI** | Number of coded symbols for CQI ($Q'\_CQI$) | nonnegative scalar integer | `int32` |
| **QdRI** | Number of coded symbols for RI ($Q'\_RI$) | nonnegative scalar integer | `int32` |
| **QdACK** | Number of coded symbols for ACK/NACK ($Q'\_ACK$) | nonnegative scalar integer | `int32` |
| **NRE** | Number of resource elements (REs) used for PUSCH transmission | nonnegative scalar integer | `int32` |
| **NLayers** | Number of layers associated with one codeword | nonnegative scalar integer | `int32` |
| **Modulation** | Modulation scheme associated with one codeword | `'QPSK'`, `'16QAM'`, `'64QAM'` | `char` |
| **RV** | RV value associated with one codeword | scalar integer | `int32` |

## Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.104. “Evolved Universal Terrestrial Radio Access (E-UTRA); Base Station (BS) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[3] 3GPP TS 36.321. “Evolved Universal Terrestrial Radio Access (E-UTRA); Medium Access Control (MAC) protocol Specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteULSCHDecode` | `lteULSCHInterleave` | `lteULSCHInfo` | `ltePUSCH`

# lteULSCHDecode

Uplink shared channel decoding

## Syntax

```
[trblkout,blkcrc,stateout] = lteULSCHDecode(ue,chs,trblklen,cwin,statein)
```

## Description

`[trblkout,blkcrc,stateout] = lteULSCHDecode(ue,chs,trblklen,cwin,statein)` returns the information bits `trblkout` decoded from the input soft LLR codewords data `cwin`. The UL-SCH decoder includes channel deinterleaver, rate recovery, turbo decoding, block concatenation and CRC calculations. The function also returns the type-24A transport block CRC decoding result in `blkcrc` and the HARQ process decoding state in `stateout`. The initial HARQ process state can be input via the optional `statein` parameter. The function is capable of processing both a single codeword or pairs of codewords, contained in a cell array, for the case of spatial multiplexing schemes transmitting two codewords. The type of the return variable `trblkout` is the same as input `cwin`. If `cwin` is a cell array containing one or two codewords, `trblkout` returns a cell array of one or two transport blocks. If `cwin` is a vector of soft data, `trblkout` returns a vector too. If decoding a pair of codewords, pairs of modulation schemes and RV indicators are required to be defined in the associated parameter fields below. This function only decodes the information bits, but supports the presence of UCI data, CQI, RI, and HARQ-ACK, in the input codeword. UCI should be demultiplexed then decoded separately.

Strictly speaking, because all the fields in structure `ue` are optional, it is legal for this parameter to be an empty structure.

Multiple codewords can be parameterized by two different forms of the `chs` structure. Each codeword can be defined by separate elements of a 1-by-2 structure array, or the codeword parameters can be combined together in the fields of a single scalar, or 1-by-1, structure. Any scalar field values apply to both codewords and a scalar `NLayers` is the total number. See “UL-SCH Parameterization” for more details.

`trblklen` is an input vector (one or two elements in length) defining the transport block lengths that the input code blocks should be rate recovered and decoded to.

`cwin` is an input parameter containing the floating point soft LLR data of the codewords to be decoded. It can either be a single vector or a cell array containing one or two vectors. If the latter, then all rate matching calculations assume that the pair were transmitting on a single PUSCH, distributed across the total number of layers defined in `chs`, as per TS 36.211 [1].

`statein` is an optional input structure array (empty or one or two elements) which can input the current decoder buffer state for each transport block in an active HARQ process. If `statein` is not an empty array and it contains a non-empty field `CBSBuffers` then this field should contain a cell array of vectors representing the LLR soft buffer states for the set of code blocks at the input to the turbo decoder i.e. after explicit rate recovery. The updated buffer states after decoding are returned in the `CBSBuffers` field in the output parameter `stateout`. The `statein` array would normally be generated and recycled from the `stateout` of previous calls to lteULSCHDecode as part of a sequence of HARQ transmissions.

`trblkout` is the output parameter containing the decoded information bits. It is either a single vector or a cell array containing one or two vectors, depending on the class and dimensionality of `cwin`.

`blkcrc` is an output array (one or two elements) containing the result of the type-24A transport block CRC decoding for the transport blocks.

`stateout`, the final output parameter, is a one element structure array containing the internal state of each transport block decoder in the fields `CBSBuffers`, `CBSCRC`, `blkcrc`.

The `stateout` array would normally be reapplied via the `statein` variable of subsequent `lteULSCHDecode` function calls as part of a sequence of HARQ retransmissions.

## Examples

### Decode UL-SCH

Generate and decode 2 transmissions (RV=0 then RV=2) as part of a single codeword HARQ process for the A3-3 FRC.

Initialize one subframe of an FRC A3-3 transmission. Create a codeword with RV = 0. Turn logical bits into 'LLR' data. Initialize the decoder states for the first HARQ transmission. The returned `decState` contains the decoder buffer state for each transport block for an active HARQ process.

```
nsf = 1;
frc = lteRMCUL('A3-3');
trBlkLen = frc.PUSCH.TrBlkSizes(nsf);
trBlkData = randi([0,1],trBlkLen,1);

frc.PUSCH.RV = 0;
cw = lteULSCH(frc,frc.PUSCH,trBlkData);

cw(cw == 0) = -1;

decState = [];
[rxTrBlk,~,decState] = lteULSCHDecode(frc,frc.PUSCH,trBlkLen,cw,decState);
```

Create a second retransmitted codeword with RV = 2. Turn logical bits into 'LLR' data. The previous transmission decoder buffer state, `decState`, is used as part of the sequence of active HARQ transmissions.

```
frc.PUSCH.RV = 2;
cWord = lteULSCH(frc,frc.PUSCH,trBlkData);

cWord(cWord == 0) = -1;

rxTrBlk = lteULSCHDecode(frc,frc.PUSCH,trBlkLen,cWord,decState);
```

## Input Arguments

### ue — UE-specific settings

scalar structure

UE-specific settings, specified as a scalar structure with the following fields. Because all the fields in structure `ue` are optional, this parameter can be an empty structure.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`CyclicPrefixUL`** | Optional | `'Normal'` (default), `'Extended'` | Cyclic prefix length. |
| **`Shortened`** | Optional | 0 (default), 1 | Option to shorten the subframe by omitting the last symbol, specified as 0 or 1. If `1`, the last symbol of the subframe is not used. For subframes with possible SRS transmission, set `Shortened` to 1 to maintain a standard compliant configuration. |

Data Types: `struct`

**`chs` — UL-SCH related parameters**
scalar structure

UL-SCH related parameters, specified as a scalar structure with the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`Modulation`** | Required | `'QPSK'`, `'16QAM'`, `'64QAM'`, `'256QAM'` | Modulation scheme associated with each transport block, specified as a character vector, or if there are two blocks, as a cell array of character vectors or string array. |
| **`NLayers`** | Optional | 1 (default), 2, 3, 4 | Number of transmission layers.<br><br>(total or per codeword) |
| **`RV`** | Required | 0, 1, 2, 3 | Redundancy version indicators, specified as a vector of 1 or 2 indicators. |
| **`QdCQI`** | Optional | nonnegative scalar integer | Number of coded channel quality information (CQI) symbols (*Q'_CQI*) |
| **`QdRI`** | Optional | Nonnegative scalar integer<br><br>0 (default) | Number of coded symbols for RI (*Q'_RI*) |
| **`QdACK`** | Optional | nonnegative scalar integer<br><br>0 (default) | Number of coded ACK symbols (*Q'_ACK*) |
| **`NTurboDecIts`** | Optional | Scalar integer<br><br>5 (default) | Number of turbo decoder iteration cycles |

Data Types: `struct`

**`trblklen` — Transport block length**
numeric vector

Transport block length, specified as a numeric vector (one or two elements in length) defining the transport block lengths that the input code blocks should be rate recovered and decoded to.

Data Types: `double`

**`cwin` — Soft LLR codeword data**
numeric column vector | cell array of one or two column vectors

Soft LLR codeword data, specified as a numeric column vector or a cell array of one or two column vectors. This argument contains the floating point soft LLR data of the codeword or codewords to be decoded. It can either be a single vector or a cell array containing one or two vectors. If a cell array, all rate matching calculations assume that the pair were transmitting on a single PUSCH, distributed across the total number of layers defined in `chs`, as specified in [1].

Data Types: `int8` | `cell`

**`statein` — Initial HARQ process state**
optional | structure array

Initial HARQ process state, specified as a structure array. It can be empty or have one or two elements, which can input the current decoder buffer state for each transport block in an active HARQ process.

Data Types: `double`

## Output Arguments

**`trblkout` — Decoded information bits**
numeric vector | cell array

Decoded information bits, returned as a numeric vector or cell array. `trblkout` contains decoded transport data blocks. It is either a single vector or a cell array containing one or two vectors, depending on the class and dimensionality of `cwin`.

Data Types: `double` | `cell`

**`blkcrc` — Type 24A transport block CRC decoding result**
0 or 1

Type 24A transport block CRC decoding result, returned as 0 or 1.

Data Types: `logical`

**`stateout` — HARQ process decoding state**
structure | structure array

HARQ process decoding state, returned as a one-element structure array. It contains the internal state of each transport block decoder. It contains the following parameter fields.

| Parameter Field | Description | Values | Data Type |
|---|---|---|---|
| **CBSBuffers** | LLR soft buffer states for the set of code blocks associated with a single transport block. The buffers are positioned at the input to the turbo decoder, or after explicit rate recovery. | Cell array of numeric vectors | `cell` |
| **CBSCRS** | Type-24B code block set CRC decoding results | Numeric vector | `int8` |
| **BLKCRC** | Type-24A transport block CRC decoding error | One- or two-element numeric vector | `logical` |

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteULSCH` | `lteULSCHDeinterleave` | `ltePUSCHDecode` | `lteULSCHInfo`

# lteULSCHDeinterleave

UL-SCH deinterleaving

## Syntax

```
[cdata,ccqi,cri,cack] = lteULSCHDeinterleave(ue,chs,in)
```

## Description

`[cdata,ccqi,cri,cack] = lteULSCHDeinterleave(ue,chs,in)` returns the deinterleaved data vector `cdata`, encoded UCI vectors, `ccqi`,`cri`, and `cack`, or cell array of vectors, after performing the demultiplexing and UL-SCH channel deinterleaving to undo the processing described in TS 36.212, Sections 5.2.2.7 and 5.2.2.8 [1] for UE-specific settings, `ue`, and UL-SCH channel specific configuration, `chs`.

The function expects the input `in` to be multiplexed and interleaved as per defined in TS 36.212, Sections 5.2.2.7 and 5.2.2.8 [1]. This input can be a vector or a cell array of vectors, deinterleaved and demultiplexed separately, and the outputs are of the same form. The size of the coded CQI symbols and codeword number with it is multiplexed, to correctly perform the demultiplexing, are deduced using the channel specific structure `chs` via the `Modulation` and `QdCQI` parameters. The presence or absence of `ccqi` in the transmission is signaled via `QdCQI` parameter with nonzero (number of coded CQI symbols) or zero value, respectively.

Multiple codewords can be parameterized by two different forms of the `chs` structure. Each codeword can be defined by separate elements of a 1-by-2 structure array, or the codeword parameters can be combined together in the fields of a single scalar, or 1-by-1, structure. Any scalar field values apply to both codewords and a scalar `NLayers` is the total number. See “UL-SCH Parameterization” for more details.

## Examples

### Interleave and Deinterleave UL-SCH

Perform back-to-back interleaving and deinterleaving of a vector of interleaver input bit indices.

Create UE-specific and propagation channel configuration structures.

```
ue.CyclicPrefixUL = 'Normal';
ue.Shortened = 0;
chs.Modulation = 'QPSK';
chs.NLayers = 1;
chs.QdCQI = 0;
chs.QdRI = 0;
chs.QdACK = 0;
```

There are 288 PUSCH QPSK symbols in two RBs and two bits per symbol for QPSK.

```
cdata = randi([0 1],2*288,1);
size(cdata)
```

```
ans = 1×2

   576     1
```

```
interleaved = lteULSCHInterleave(ue,chs,cdata);
deinterleaved = lteULSCHDeinterleave(ue,chs,interleaved);
size(deinterleaved)
```

```
ans = 1×2

   576     1
```

The deinterleaved output is the same size as the data prior to interleaving.

## Input Arguments

### `ue` — UE-specific settings

structure

UE-specific settings, specified as a structure with the following fields.

#### `CyclicPrefixUL` — Cyclic prefix length

`'Normal'` (default) | optional | `'Extended'`

Cyclic prefix length, specified as `'Normal'` or `'Extended'`.

Data Types: `char` | `string`

#### `Shortened` — Shorten subframe flag

0 (default) | optional | 1

Shorten subframe flag, specified as 0 or 1. Optional. If 1, the last symbol of the subframe is not used. It should be set if the current subframe contains a possible SRS transmission.

Data Types: `logical` | `double`

Data Types: `struct`

### `chs` — UL-SCH related parameters

structure

UL-SCH related parameters, specified as a structure with the following fields.

#### `Modulation` — Modulation format

`'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'` | cell array of character vectors | string array

Modulation format, specified as `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`. Use double quotes for string. It there are two blocks, use a cell array of character vectors or a string array.

Data Types: `char` | `string`

#### `NLayers` — Number of transmission layers

1 (default) | optional | 2 | 3 | 4

Number of transmission layers, total or per codeword, specified as a positive scalar integer. Optional.

Data Types: `double`

**`QdCQI` — Number of coded symbols for CQI**
0 (default) | optional | nonnegative scalar integer

Number of coded symbols for CQI, specified as a nonnegative scalar integer. Optional. (*Q'_CQI*)

Data Types: `double`

**`QdRI` — Number of coded symbols for RI**
0 (default) | optional | nonnegative scalar integer

Number of coded symbols for RI, specified as a nonnegative scalar integer. Optional. (*Q'_RI*)

Data Types: `double`

**`QdACK` — Number of coded symbols for ACK/NACK**
0 (default) | optional | nonnegative scalar integer

Number of coded symbols for ACK/NACK, specified as a nonnegative scalar integer. Optional. (*Q'_ACK*)

Data Types: `double`

Data Types: `struct`

**`in` — Input data**
column vector | cell array of column vectors

Input data specified as a column vector or a cell array of column vectors.

Data Types: `double` | `cell`

## Output Arguments

**`cdata` — Deinterleaved data**
column vector | cell array of column vectors

Deinterleaved data, returned as a column vector or cell array of column vectors.

Data Types: `double` | `cell`

**`ccqi` — Encoded UCI**
vector | cell array of vectors

Encoded UCI, returned as a vector or cell array of vectors.

Data Types: `double` | `cell`

**`cri` — Encoded UCI**
vector | cell array of vectors

Encoded UCI, returned as a vector or cell array of vectors.

Data Types: `double` | `cell`

**`cack` — Encoded UCI**
vector | cell array of vectors

Encoded UCI, returned as a vector or cell array of vectors.

Data Types: `double` | `cell`

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteULSCHInterleave` | `lteULSCH` | `lteULSCHInfo` | `lteACKDecode` | `lteCQIDecode` | `lteRIDecode` | `lteRateRecoverTurbo`

# lteULSCHInfo

UL-SCH coding information

## Syntax

```
info = lteULSCHInfo(ue,chs,blklen)
info = lteULSCHInfo(ue,chs,blklen,opts)
info = lteULSCHInfo(ue,chs,blklen,ocqi,ori,oack)
info = lteULSCHInfo(ue,chs,blklen,ocqi,ori,oack,opts)
```

## Description

`info = lteULSCHInfo(ue,chs,blklen)` provides information related to the entire UL-SCH coding process, for UL-SCH data without UCI. It returns a structure array with fields covering the transport channel (TrCH) encoding and UCI multiplexing. When UCI is present, it includes the coded symbol capacities given UCI sizes, PUSCH resource allocations, and *Beta* offset values, which can be useful in a number of UL-SCH- and PUSCH-related functions. These symbol capacities are calculated from the *Q*' formulae in TS 36.212, Sections 5.2.2.6 and 5.2.4.1 [1]. The one- or two-element vector, `blklen`, defines the length of the transmitted transport blocks.

By default, in the case of multiple transport blocks or codewords, each structure in the array corresponds to one of the blocks. Note that the `NLayers`, `Modulation`, and `RV` fields at the output contain only the value for the associated codeword and therefore have a different form to those given in the input. In the case of `NLayers` the output returns the number of layers per codeword where the input field represents the total number of transmission layers across all codewords.

If the UL-SCH encoding is for a retransmission of a previously sent transport block, use the three “Init” fields, `'InitPRBSet'`, `'InitCyclicPrefixUL'`, and `'InitShortened'`. If any of these fields are absent, their values are assumed to be the same as the values for the associated current subframe fields, `'PRBSet'`, `'CyclicPrefixUL'`, and `'Shortened'`.

`info = lteULSCHInfo(ue,chs,blklen,opts)` formats the output through options specified by `opts`. The optional parameter `opts` allows for the merging of the input `chs` structure fields into `info` at the output.

`info = lteULSCHInfo(ue,chs,blklen,ocqi,ori,oack)` supports the multiplexing of both transport and UCI data, CQI, RI, and HARQ-ACK, or UCI only. The number of uncoded UCI bits is given by `ocqi`, `ori` and `oack` respectively. Any of the data length parameters can be zero if the associated data is not present. The coding of the UCI can be controlled through the additional `BetaACK`, `BetaCQI`, and `BetaRI` fields in the `chs` input structure.

`info = lteULSCHInfo(ue,chs,blklen,ocqi,ori,oack,opts)` supports the multiplexing of both transport and UCI data (CQI, RI, HARQ-ACK) or UCI only.

## Examples

### Obtain UL-SCH Information for One Transport Block

Obtain information for UL-SCH coding of a single transport block of length 6712 bits.

Create a PUSCH configuration structure. Initialize the optional fields for a ue-specific setting structure. Default settings are used if you don't initial optional fields. View the UL-SCH information.

```
pusch.Modulation = 'QPSK';
pusch.NLayers = 1;
pusch.PRBSet = [0:74].';
ue.CyclicPrefixUL = 'Normal';
ue.Shortened = 0;
blkLen = 6712;
info = lteULSCHInfo(ue,pusch,blkLen)
```

```
info = struct with fields:
             C: 2
            Km: 3328
            Cm: 0
            Kp: 3392
            Cp: 2
             F: 0
             L: 24
          Bout: 6784
             G: 21600
            Qm: 2
            Gd: 10800
          OCQI: 0
           ORI: 0
          OACK: 0
         QdCQI: 0
          QdRI: 0
         QdACK: 0
           NRE: 10800
       NLayers: 1
    Modulation: 'QPSK'
```

### Obtain ULSCH Info for Transport Blocks in Structure Array

Obtain information for UL-SCH coding of two transport blocks (codewords) with UCI (3 bit RI, 2 bit HARQ-ACK). Each element in the output array corresponds to a codeword.

Create a PUSCH configuration structure and an empty UE structure.

```
pusch.Modulation = {'QPSK' '16QAM'};
pusch.NLayers = 3;
pusch.PRBSet = [0:74].';
ue = struct();
```

Specify the number of CQI, RI, and HARQ-ACK bits

```
OCQI = 0;
ORI = 3;
OACK = 2;
blkLen = [6712 6712];
```

View the UL-SCH information

```
info = lteULSCHInfo(ue,pusch,blkLen ,OCQI,ORI,OACK)
```

```
info=1×2 struct array with fields:
    C
    Km
    Cm
    Kp
    Cp
    F
    L
    Bout
    G
    Qm
    Gd
    OCQI
    ORI
    OACK
    QdCQI
    QdRI
    QdACK
    NRE
    NLayers
    Modulation
      ⋮
```

### Obtain ULSCH Info for Transport Blocks in Scalar Structure

Obtain information in a single scalar structure for the UL-SCH coding of two transport blocks with UCI, specifying 3-bit RI and 2-bit HARQ-ACK.

Create a PUSCH configuration structure and an empty UE structure.

```
pusch.Modulation={'QPSK' '16QAM'};
pusch.NLayers = 3;
pusch.PRBSet = [0:74].';
ue = struct();
```

Specify the number of CQI, RI, and HARQ-ACK bits.

```
OCQI = 0;
ORI = 3;
OACK = 2;
blkLen = [6712 6712];
```

View the UL-SCH information. Most fields in the structure contain two elements corresponding to each codeword.

```
info = lteULSCHInfo(ue,pusch,blkLen,OCQI,ORI,OACK,'cwcombined')
```

```
info = struct with fields:
             C: [2 2]
            Km: [3328 3328]
            Cm: [0 0]
            Kp: [3392 3392]
            Cp: [2 2]
             F: [0 0]
```

```
          L: [24 24]
       Bout: [6784 6784]
          G: [21590 86360]
         Qm: [2 4]
         Gd: [10795 21590]
       OCQI: 0
        ORI: 3
       OACK: 2
      QdCQI: [0 0]
       QdRI: [5 5]
      QdACK: [4 4]
        NRE: [10800 21600]
    NLayers: [1 2]
Modulation: {'QPSK'  '16QAM'}
```

## Input Arguments

### ue — UE-specific configuration settings

structure

UE-specific configuration settings, specified as a structure that can contain the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`CyclicPrefixUL`** | Optional | `'Normal'` (default), `'Extended'` | Current cyclic prefix length |
| **`Shortened`** | Optional | 0 (default), 1 | Option to shorten the subframe by omitting the last symbol, specified as 0 or 1. If 1, the last symbol of the subframe is not used. For subframes with possible SRS transmission, set `Shortened` to 1 to maintain a standard compliant configuration. |

### chs — Channel-specific transmission configuration

structure

Channel-specific transmission configuration, specified as a structure that can contain the following parameter fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`Modulation`** | Required | `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'` | Modulation type, specified as a character vector, cell array of character vectors, or string array. If blocks, each cell is associated with a transport block. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NLayers** | Optional | 1 (default), 2, 3, 4 | Total number of transmission layers associated with the transport block or blocks. |
| **PRBSet** | Required | Integer column vector or two-column matrix | 0-based physical resource block indices (PRBs) for the slots of the current PUSCH resource allocation. As a column vector, the resource allocation is the same in both slots of the subframe. As a two-column matrix, it specifies different PRBs for each slot in a subframe. |
| **RV** | Required | Integer vector (0,1,2,3). A one or two column matrix (for one or two codewords). | Specifies the redundancy version for one or two codewords used in the initial subframe number, `NSubframe`. This parameter field is only for informational purposes and is read-only. |
| The following three `'Init'` fields should be used if the UL-SCH encoding is for a retransmission of a previously sent transport block. If any of these fields are absent, its value is assumed to be the same as the value for its associated current subframe field. | | | |
| **InitPRBSet** | Optional | 1- or 2-column integer matrix, `PRBSet` (default) | PRB indices used in the initial transmission PUSCH allocation. If this field is absent, its value is assumed to be the same as the value for the associated current subframe field, `PRBSet`. |
| **InitCyclicPrefixUL** | Optional | `'Normal'`, `'Extended'`, `CyclicPrefixUL` (default) | Cyclic prefix length of initial transmit subframe. This is the length used during the first transmission of this transport block. If this field is absent, its value is assumed to be the same as the value for the associated current subframe field, `CyclicPrefixUL`. |
| **InitShortened** | Optional | 0, 1, `Shortened` (default) | Initial transmit subframe shortened flag. If 1, the initial transmit subframe was shortened for possible SRS. If this field is absent, its value is assumed to be the same as the value for the associated current subframe field, `Shortened`. |
| The coding of the UCI can be controlled through the following additional fields. | | | |
| **BetaCQI** | Optional | numeric scalar, 2.0 (default) | Modulation and coding scheme (MCS) offset for CQI and PMI bits |
| **BetaRI** | Optional | numeric scalar, 2.0 (default) | Modulation and coding scheme (MCS) offset for RI bits |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **BetaACK** | Optional | numeric scalar, 2.0 (default) | Modulation and coding scheme (MCS) offset for HARQ-ACK bits. This field was previously named `BetaHI`; if this field is absent but `BetaHI` is present, it is used as before. |

**`blklen` — Length of transmitted transport blocks**
numeric vector

Length of the transmitted transport blocks, specified as a one or two element numeric vector.

Data Types: `double`

**`opts` — Output formatting options**
character vector | cell array of character vectors | string array

Output formatting options, specified as a character vector, cell array of character vectors, or string array. For convenience, you can specify several options as a single character vector or string scalar by a space-separated list of values placed inside the quotes. Values for `opts` when specified as a character vector include (use double quotes for string):

| Option | Values | Description |
|---|---|---|
| Channel parameter merging | `'nochsconcat'` (default) | Do not concatenate `chs` input structure into `info`. |
| | `'chsconcat'` | Concatenate `chs` input structure into `info`. |
| Output structure format | `'cwseparate'` (default) | Information values for each codeword are output in separate elements of the 1-by-ncodewords structure array `info`. |
| | `'cwcombined'` | Information values for each codeword are combined into the fields of a single scalar, or 1-by-1, structure. |

Example: `'chsconcat cwcombined'`, `"chsconcat cwcombined"`, `{'chsconcat','cwcombined'}`, or `["chsconcat","cwcombined"]` specify the same formatting options.

Data Types: `char` | `string` | `cell`

**`ocqi` — Number of uncoded CQI bits**
numeric scalar

Number of uncoded CQI bits, specified as a numeric scalar.

Data Types: `double`

**`ori` — Number of uncoded RI bits**
numeric scalar

Number of uncoded RI bits, specified as a numeric scalar.

Data Types: `double`

### `oack` — Number of uncoded HARQ-ACK bits

numeric scalar

Number of uncoded HARQ-ACK bits, specified as a numeric scalar.

Data Types: `double`

## Output Arguments

### `info` — UL-SCH information

structure | structure array

UL-SCH information, returned as a structure or a structure array. If two transport blocks are encoded, `info` is a structure array of two elements, one for each block. , It contains the following parameter fields.

| Parameter Field | Description | Values | Data Type |
|---|---|---|---|
| **`C`** | Total number of code blocks | nonnegative scalar integer | `int32` |
| **`Km`** | Lower code block size ($K-$) | nonnegative scalar integer | `int32` |
| **`Cm`** | Number of code blocks of size `Km` ($C-$) | nonnegative scalar integer | `int32` |
| **`Kp`** | Upper code block size ($K+$) | nonnegative scalar integer | `int32` |
| **`Cp`** | Number of code blocks of size `Kp` ($C+$) | nonnegative scalar integer | `int32` |
| **`F`** | Number of filler bits in first block | nonnegative scalar integer | `int32` |
| **`L`** | Number of segment cyclic redundancy check (CRC) bits | nonnegative scalar integer | `int32` |
| **`Bout`** | Total number of bits in all segments | nonnegative scalar integer | `int32` |
| **`G`** | Number of coded and rate matched UL-SCH data bits | nonnegative scalar integer | `int32` |
| **`Qm`** | Number of bits per symbol | nonnegative scalar integer | `int32` |
| **`Gd`** | Number of coded and rate matched UL-SCH data symbols ($G'$) | nonnegative scalar integer | `int32` |
| **`OCQI`** | Number of uncoded channel quality information (CQI) bits | nonnegative scalar integer | `int32` |
| **`ORI`** | Number of uncoded symbols for RI | nonnegative scalar integer | `int32` |
| **`OACK`** | Number of uncoded symbols for ACK/NACK | nonnegative scalar integer | `int32` |

| Parameter Field | Description | Values | Data Type |
|---|---|---|---|
| **QdCQI** | Number of coded symbols for CQI (*Q'_CQI*) | nonnegative scalar integer | int32 |
| **QdRI** | Number of coded symbols for RI (*Q'_RI*) | nonnegative scalar integer | int32 |
| **QdACK** | Number of coded symbols for ACK/NACK (*Q'_ACK*) | nonnegative scalar integer | int32 |
| **NRE** | Number of resource elements (REs) used for PUSCH transmission | nonnegative scalar integer | int32 |
| **NLayers** | Number of layers associated with one codeword | nonnegative scalar integer | int32 |
| **Modulation** | Modulation scheme associated with one codeword | 'QPSK', '16QAM', '64QAM' | char |
| **RV** | RV value associated with one codeword | scalar integer | int32 |

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.212. "Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

lteULSCH | lteULSCHDecode | lteULSCHInterleave | ltePUSCHDecode | lteCQIEncode | lteRIEncode | lteACKEncode

# lteULSCHInterleave

UL-SCH interleaving

## Syntax

```
out = lteULSCHInterleave(ue,chs,cdata)
out = lteULSCHInterleave(ue,chs,cdata,ccqi,cri,cack)
```

## Description

`out = lteULSCHInterleave(ue,chs,cdata)` performs the UL-SCH channel interleaving on input `cdata` containing encoded transport channel (TrCH) data without UCI. It performs the UL-SCH data and UCI multiplexing and interleaving as defined in TS 36.212 Sections 5.2.2.7 and 5.2.2.8 [1]. This input can be a vector or a cell array of vectors, interleaved separately, and the output is of the same form.

Multiple codewords can be parameterized by two different forms of the `chs` structure. Each codeword can be defined by separate elements of a 1-by-2 structure array, or the codeword parameters can be combined together in the fields of a single scalar, or 1-by-1, structure. Any scalar field values apply to both codewords and a scalar `NLayers` is the total number. See “UL-SCH Parameterization” for more details.

`out = lteULSCHInterleave(ue,chs,cdata,ccqi,cri,cack)` is as above except it also supports UL-SCH channel interleaving on both `cdata` and encoded UCI in `ccqi`, `cri` and `cack`. If any of these inputs are cell arrays, the output has the same form and any vector inputs are interleaved into the first cell of the output only. Any of the input cells or arrays can be empty if the associated input is not transmitted on one or more codewords.

## Examples

### PUSCH Interleave

Interleave two PUSCH RBs worth of bits for QPSK modulation. Considering the REs reserved for PUSCH DM-RS, there are 144 REs available for PUSCH data per RB. Therefore, two RBs contain 288 PUSCH symbols. This results in 2*288 bits to QPSK modulate after interleaving.

Initialize UE specific and UL-SCH related parameter structures. Generate data for QPSK modulation of PUSCH symbols in two RBs. For QPSK, there are two bits per symbol. Perform interleaving and symbol modulation.

```
ue.CyclicPrefixUL = 'Normal';
ue.Shortened = 0;

chs.Modulation = 'QPSK';
chs.NLayers = 1;

numRB = 2;
numREperRB = 144;
bitsPerSymbol = 2;
```

```
numBits = numRB * numREperRB * bitsPerSymbol;
cdata = randi([0 1], numBits, 1);

interleaved = lteULSCHInterleave(ue, chs, cdata);
symbols = lteSymbolModulate(interleaved, 'QPSK');
```

## Input Arguments

### ue — UE-specific settings

structure

UE-specific settings, specified as a structure with the following fields.

#### CyclicPrefixUL — Cyclic prefix length

'Normal' (default) | optional | 'Extended'

Cyclic prefix length, specified as 'Normal' or 'Extended'.

Data Types: char | string

#### Shortened — Shorten subframe

0 (default) | optional | 1

Shorten subframe, specified as 0 or 1. If 1, the last symbol of the subframe is not used. It should be set if the current subframe contains a possible SRS transmission.

Data Types: logical

Data Types: struct

### chs — UL-SCH related parameters

scalar structure

UL-SCH related parameters, specified as a scalar structure with the following fields.

#### Modulation — Modulation format

'QPSK' | '16QAM' | '64QAM' | '256QAM' | cell array of these character vectors. | string array

Modulation format, specified as 'QPSK', '16QAM', '64QAM', or '256QAM'. Use double quotes for string. It there are two blocks, use a cell array of character vectors or a string array. Each element of the arrays is associated with a transport block.

Data Types: char | string

#### NLayers — Number of transmission layers (total or per codeword)

1 (default) | optional | 2 | 3 | 4

Number of transmission layers (total or per codeword), specified as 1, 2, 3, or 4.

Data Types: double

Data Types: struct

### cdata — Encoded TrCH data

column vector | cell array of column vectors

Encoded TrCH data, specified as a column vector or a cell array of column vectors.

Data Types: `double` | `cell`

**`ccqi` — Encoded CQI**
vector

Encoded CQI, specified as a vector.

Data Types: `double`

**`cri` — Encoded RI**
vector

Encoded RI, specified as a vector.

Data Types: `double`

**`cack` — Encoded ACK**
vector

Encoded ACK, specified as a vector.

Data Types: `double`

## Output Arguments

**`out` — Interleaved UL-SCH output**
numeric column vector | cell array of numeric column vectors

Interleaved UL-SCH output, returned as a numeric column vector or a cell array of numeric column vectors.

Data Types: `double` | `cell`

# Version History
**Introduced in R2014a**

## References

[1] 3GPP TS 36.212. “Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also
`lteULSCHDeinterleave` | `lteULSCH` | `lteULSCHInfo` | `lteACKEncode` | `lteCQIEncode` | `lteRIEncode` | `lteRateMatchTurbo`

# lteULScramble

PUSCH scrambling

## Syntax

```
out = lteULScramble(in,nsubframe,cellid,rnti)
out = lteULScramble(ue,in)
```

## Description

`out = lteULScramble(in,nsubframe,cellid,rnti)` performs PUSCH scrambling of bit vector, `in`, for subframe number, `nsubframe`, cell identity, `cellid`, and specified RNTI, `rnti`. It performs PUSCH scrambling according to TS 36.211, Section 5.3.1 [1]. Placeholder bits, denoted by *x*, are represented by –1 in the input vector or cell array of vectors. Repetition placeholder bits, *y*, are represented by –2. This function substitutes these placeholders as part of its scrambling operation.

`in` is a vector or a cell array containing one or two vectors corresponding to the number of codewords to be scrambled.

`out = lteULScramble(ue,in)` performs PUSCH scrambling of the `in` according to UE-specific settings in structure, `ue`.

## Examples

### Perform PUSCH Scrambling

Perform PUSCH scrambling for NCellID=100 and RNTI=61.

```
in = ones(10,1);
bits = lteULScramble(struct('NCellID',100,'NSubframe',0,'RNTI',61),in)
```

```
bits = 10x1 int8 column vector

   0
   1
   0
   0
   0
   1
   0
   0
   1
   1
```

## Input Arguments

### `in` — Bit input data

numeric column vector | cell array of numeric column vectors

Bit input data, specified as a numeric column vector or cell array of numeric column vectors. This argument contains one or two vectors corresponding to the number of codewords to be scrambled.

Data Types: `double` | `cell`
Complex Number Support: Yes

**`nsubframe` — Subframe number**
numeric scalar

Subframe number, specified as a numeric scalar.

Data Types: `double`

**`cellid` — Physical layer cell identity**
numeric scalar

Physical layer cell identity, specified as a numeric scalar.

Data Types: `double`

**`rnti` — Radio Network Temporary Identifier (16-bit)**
numeric scalar

Radio Network Temporary Identifier (16-bit). Specified as a numeric scalar.

Data Types: `double`

**`ue` — UE-specific settings**
structure

UE-specific settings, specified as a structure with the following fields.

**`NCellID` — Physical layer cell identity**
numeric scalar

Physical layer cell identity, specified as a numeric scalar.

Data Types: `double`

**`NSubframe` — Subframe number**
numeric scalar

Subframe number, specified as a numeric scalar.

Data Types: `double`

**`RNTI` — Radio Network Temporary Identifier (16-bit)**
numeric scalar

Radio Network Temporary Identifier (16-bit). Specified as a numeric scalar.

Data Types: `double`

Data Types: `struct`

## Output Arguments

**`out` — PUSCH scrambled output bits**
numeric column vector | cell array of numeric column vectors

PUSCH scrambled output bits, returned as a numeric column vector or a cell array of numeric column vectors.

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.211. “Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`lteULDescramble` | `lteSymbolModulate` | `ltePUSCH`

# lteWarning

Enable and disable warnings for LTE Toolbox

## Syntax

```
lteWarning('OFF',msgname)
lteWarning('ON',msgname)
```

## Description

`lteWarning('OFF',msgname)` and `lteWarning('ON',msgname)` disable and enable the display of any warning messages of type `msgname`, specific to the LTE Toolbox.

## Examples

### Toggle LTE Function Warnings

Disable and enable default value warnings for optional parameters.

Turn on the warning about default values.

```
lteWarning('on','DefaultValue')
bch = lteBCH(struct('CellRefP',1),ones(24,1));
```

```
Warning: Using default value for parameter field CyclicPrefix (Normal)
```

Notice a warning about default values is displayed when the same command is run to generate a BCH.

Turn off the warning about default values.

```
lteWarning('off','DefaultValue')
bch = lteBCH(struct('CellRefP',1),ones(24,1));
```

## Input Arguments

### `msgname` — Message name

`'defaultValue'` | `'deprecated'`

Message name, specified as `'defaultValue'` or `'deprecated'`.

| Option | Warning Message |
|---|---|
| `'defaultValue'` | This warning occurs when an optional parameter value or parameter structure field is not defined and the default value is used instead. |

| Option | Warning Message |
| --- | --- |
| `'deprecated'` | This warning occurs for deprecated functionality of the LTE Toolbox. For example, it occurs when the user calls a function using a deprecated syntax option. This warning indicates that the functionality may be removed in a later release. |

Data Types: `char`

# Version History

**Introduced in R2014a**

## See Also

`warning`

# umtsDownlinkReferenceChannels

UMTS downlink measurement channel definition

## Syntax

```
config = umtsDownlinkReferenceChannels(rc)
config = umtsDownlinkReferenceChannels(rc,modulation)
```

## Description

`config = umtsDownlinkReferenceChannels(rc)` uses the input reference channel, `rc`, to produce a downlink reference channel definition structure, `config`. The configuration parameters required by `umtsDownlinkWaveformGenerator` to generate a downlink reference channel waveform are included in `config`.

For all syntaxes, `umtsDownlinkReferenceChannels` uses the input, `rc`, to initialize a configuration data structure compliant with one of the reference channels defined in the following 3GPP standards:

- Downlink W-CDMA reference measurement channel (RMC) waveforms, as defined in TS 25.101, Annex A3 [1]
- HSDPA fixed reference channel (FRC) H-Set waveforms, as defined in TS 25.101, Annex A7 [1]
- Downlink test model waveforms, as defined in TS 25.141, Section 6.1.1 [2]

`config = umtsDownlinkReferenceChannels(rc,modulation)` gives you the option of changing the default modulation scheme when `rc` specifies initialization of an FRC H-Set configuration. See the table of valid H-Set/modulation combinations in the description of the `modulation` input.

## Examples

### UMTS Downlink Reference Channel Initialization

Initialize a `'QPSK'` `'RMC12.2kbps'` reference channel.

Generate the configuration structure, `rmcStruct`

```
rc = 'RMC12.2kbps';
modulation = 'QPSK';
rmcStruct = umtsDownlinkReferenceChannels(rc, modulation);
```

The output from `umtsDownlinkReferenceChannels` provides the input required to generate the desired UMTS waveform corresponding to these settings.

Examine the `DPCH` field in `rmcStruct`. This field has a nested structure defining this physical channel for the `'RMC12.2kbps'` reference channel with `'QPSK'` modulation.

```
rmcStruct
```

```
rmcStruct = struct with fields:
                TotFrames: 1
```

```
    PrimaryScramblingCode: 0
               FilterType: 'RRC'
         OversamplingRatio: 4
           NormalizedPower: 'Off'
                      DPCH: [1x1 struct]
                    PCCPCH: [1x1 struct]
                    SCCPCH: [1x1 struct]
                    PCPICH: [1x1 struct]
                    SCPICH: [1x1 struct]
                      PSCH: [1x1 struct]
                      SSCH: [1x1 struct]
                      PICH: [1x1 struct]
                     HSDPA: [1x1 struct]
                      OCNS: [1x1 struct]
```

```
rmcStruct.DPCH
```

```
ans = struct with fields:
                        Enable: 'On'
                    SlotFormat: 11
                  SpreadingCode: 6
                   NMulticodes: 1
    SecondaryScramblingCode: 1
                TimingOffset: 0
                       Power: 0
                     TPCData: 0
                        TFCI: 0
                  DataSource: 'CCTrCH'
                      CCTrCH: [1x1 struct]
```

```
rmcStruct.DPCH.CCTrCH
```

```
ans = struct with fields:
              Name: 'DCH'
    DTXPosition: 'fixed'
              TrCH: [1x2 struct]
```

```
rmcStruct.DPCH.CCTrCH.TrCH(1)
```

```
ans = struct with fields:
                    Name: 'DTCH'
                     CRC: '16'
              CodingType: 'conv3'
                     RMA: 256
                     TTI: 20
              DataSource: 'PN9-ITU'
    ActiveDynamicPart: 1
            DynamicPart: [1x1 struct]
```

```
rmcStruct.DPCH.CCTrCH.TrCH(1).DynamicPart
```

```
ans = struct with fields:
       BlockSize: 244
    BlockSetSize: 244
```

```
rmcStruct.DPCH.CCTrCH.TrCH(2)

ans = struct with fields:
                 Name: 'DCCH'
                  CRC: '12'
           CodingType: 'conv3'
                  RMA: 256
                  TTI: 40
           DataSource: 'PN9-ITU'
    ActiveDynamicPart: 1
          DynamicPart: [1x1 struct]


rmcStruct.DPCH.CCTrCH.TrCH(2).DynamicPart

ans = struct with fields:
       BlockSize: 100
    BlockSetSize: 100
```

## Input Arguments

### `rc` — Reference channel configuration

character vector | string scalar

Reference channel configuration, specified as a character vector or string scalar. `rc` identifies which RMC, H-Set, or test model to configure. Values for `rc` when specified as a character vector include (for string scalar use double quotes):

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`rc`** | Required | Reference Measurement Channels:<br><br>`'RMC0kbps'`, `'RMC12.2kbps'`, `'RMC64kbps'`, `'RMC144kbps'`, `'RMC384kbps'` | Reference channel identifying the W-CDMA downlink RMC configuration, as defined in TS 25.101, Annex A3 [1]. |
| | | Fixed Reference Channel H-Sets:<br><br>`'H-Set1'`, `'H-Set2'`, `'H-Set3'`, `'H-Set4'`, `'H-Set5'`, `'H-Set6'`, `'H-Set7'`, `'H-Set8'`, `'H-Set10'`, `'H-Set12'`. | Reference channel identifying the HSDPA and HSPA+ FRC H-Set configuration, as defined in TS 25.101, Annex A7 [1]. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| | | Test Models:<br><br>'TM1_4DPCH', 'TM1_8DPCH', 'TM1_16DPCH', 'TM1_32DPCH', 'TM1_64DPCH', 'TM2_3DPCH', 'TM3_4DPCH', 'TM3_8DPCH', 'TM3_16DPCH', 'TM3_32DPCH', 'TM4', 'TM5_4DPCH_4HSPDSCH', 'TM5_6DPCH_2HSPDSCH', 'TM5_14DPCH_4HSPDSCH', 'TM5_30DPCH_8HSPDSCH', 'TM6_4DPCH_4HSPDSCH', 'TM6_30DPCH_8HSPDSCH' | Reference channel identifying the test model physical channel configuration, as defined in TS 25.141, Section 6.1.1 [2]. |

Data Types: `char` | `string`

**`modulation` — Modulation scheme when FRC H-Set is configured**
character vector | string scalar

Modulation scheme when FRC H-Set is configured, specified as a character vector or string scalar. This argument applies only when `rc` specifies an FRC H-Set configuration. The table identifies valid H-Set/Modulation combinations as character vectors (use double quotes for string). When `modulation` is not specified, the default value is applied.

| Valid Combinations | `modulation` | | | Default modulation (if not specified) |
|---|---|---|---|---|
| `rc` | `'QPSK'` | `'16QAM'` | `'64QAM'` | |
| 'H-Set1' | ✓ | ✓ | — | 'QPSK' |
| 'H-Set2' | ✓ | ✓ | — | 'QPSK' |
| 'H-Set3' | ✓ | ✓ | — | 'QPSK' |
| 'H-Set4' | ✓ | — | — | 'QPSK' |
| 'H-Set5' | ✓ | — | — | 'QPSK' |
| 'H-Set6' | ✓ | ✓ | — | 'QPSK' |
| 'H-Set7' | ✓ | — | — | 'QPSK' |
| 'H-Set8' | — | — | ✓ | '64QAM' |
| 'H-Set10' | ✓ | ✓ | — | 'QPSK' |
| 'H-Set12' | ✓ | — | — | 'QPSK' |

Data Types: `char` | `string`

## Output Arguments

**`config` — Definition of the channels included for the waveform generator**
structure

#### Top-Level Parameters and Substructures

Definition of the channels included for the waveform generator, returned as a structure.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`TotFrames`** | Required | Nonnegative scalar integer | Total number of frames to be generated |
| **`PrimaryScramblingCode`** | Required | Scalar integer from 0 to 511 | Primary scrambling code index |
| **`FilterType`** | Required | `'RRC'` (default), or `'Off'` | Enable the RRC filter |
| **`OversamplingRatio`** | Required | Nonnegative scalar integer | Oversampling ratio |
| **`NormalizedPower`** | Required | Float (-inf to +inf) or `'Off'` to disable power normalization | Overall waveform power in dBW relative to 1 ohm |
| **`DPCH`** | Optional | Not present, single structure, or structure array | See DPCH Substructure. |
| **`PCCPCH`** | Optional | Not present or single structure | See PCCPCH Substructure. |
| **`SCCPCH`** | Optional | Not present or single structure | See SCCPCH Substructure. |
| **`PCPICH`** | Optional | Not present or single structure | See PCPICH Substructure. |
| **`SCPICH`** | Optional | Not present or single structure | See SCPICH Substructure. |
| **`PSCH`** | Optional | Not present or single structure | See PSCH Substructure. |
| **`SSCH`** | Optional | Not present or single structure | See SSCH Substructure. |
| **`PICH`** | Optional | Not present or single structure | See PICH Substructure. |
| **`HSDPA`** | Optional | Not present or single structure | See HSDPA Substructure |
| **`OCNS`** | Optional | Not present or single structure | See OCNS Substructure. |

#### DPCH Substructure

Include the `DPCH` substructure in the `config` structure to add dedicated physical channels to the output structure. The `DPCH` substructure contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **Enable** | Required | `'On'`, `'Off'` | Enable or disable the channel by setting `Enable` to `'On'` or `'Off'`, respectively. |
| **SlotFormat** | Required | Nonnegative integer | DPCH slot format number, specified as a nonnegative integer in the interval [0, 16]. |
| **SpreadingCode** | Required | Nonnegative integer | DPCH spreading code, specified as a nonnegative integer in the interval [0, 512]. For multicode transmission, `SpreadingCode` is the first DPCH code. |
| **NMulticodes** | Required | Positive integer | Number of DPCHs, specified as 1, 2, 3, 4, 5, or 6. |
| **SecondaryScramblingCode** | Required | Nonnegative integer | DPCH secondary scrambling code index, specified as a nonnegative integer in the interval [0, 15]. |
| **TimingOffset** | Required | Nonnegative integer | The timing offset in terms of the number of chips (x256Tchip), specified as a nonnegative integer in the interval [0, 149]. |
| **Power** | Required | Float, `-inf`, `inf` | Channel power in dB, specified as a float, `-inf`, or `inf`. |
| **TPCData** | Required | Binary scalar, binary vector | Transmit Power Control data, specified as a binary scalar or a vector with binary entries. |
| **TFCI** | Required | Nonnegative integer | Transport Format Combination Indicator (TFCI), specified as a nonnegative integer in the interval [0, 1023]. |
| **DataSource** | Required | Binary scalar, binary vector, character vector, cell array, or string scalar | DPCH data source, specified as a binary scalar, a vector with binary entries, a character vector, a cell array, or a string scalar.<br><br>When specifying `DataSource` as a cell array, use standard PN sequences and a seed value: {PN, seed}. PN options for character vector or cell array are `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, and `'PN23'`. If no seed is specified, the shift register is initialized with all ones.<br><br>To enable transport channel coding, specify `DataSource` as `'CCTrCH'`. |
| **CCTrCH** | Optional | Structure | See CCTrCH Substructure. |

**CCTrCH Substructure**

Include a `CCTrCH` substructure instance individually for `DPCH`, `PCCPCH`, and/or `SCCPCH` substructures. Separate instances of a coded composite transport channel are added to the output structures of the

DPCH, P-CCPCH, and/or S-CCPCH physical channel definitions. When the `CCTrCH` substructure is included, it contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`Name`** | Optional | Character vector, string scalar<br><br>Default depends on the physical channel specified | Name assigned to the CCTrCH, specified as a character vector or a string scalar. Functions do not use the `Name` field. Therefore, you can redefine the content with no consequence. |
| **`DTXPosition`** | Required | `'fixed'`, `'flexible'` | Specifies the DTX position, specified as `'fixed'` or `'flexible'`. |
| **`TrCH`** | Required | Structure, structure array | Transport channels in the CCTrCH, specified as a structure or a structure array. |
| **`TrCH.Name`** | Required | Character vector, string scalar<br><br>Default depends on the physical channel specified | Name assigned to the TrCH, specified as a character vector or a string scalar. Functions do not use the `Name` field. Therefore, you can redefine the content with no consequence. |
| **`TrCH.CRC`** | Required | Character vector, string scalar | Cyclic redundancy check (CRC) polynomial specifier, specified as one of these values: `'0'`, `'8'`, `'12'`, `'16'`, or `'24'`. |
| **`TrCH.TTI`** | Required | Positive integer | Transmission time interval (TTI) in milliseconds, specified as 10, 20, 40, or 80. |
| **`TrCH.CodingType`** | Required | `'turbo'`, `'conv2'`, `'conv3'` | Channel coding type and rate, specified as `'turbo'`, `'conv2'`, or `'conv3'`. |
| **`TrCH.RMA`** | Required | Positive integer | Rate matching attribute value, specified as a positive integer in the interval [1, 256]. |
| **`TrCH.DataSource`** | Required | Binary scalar, binary vector, character vector, cell array, or string scalar | Transport channel data source, specified as a binary scalar, a vector with binary entries, a cell array, or a string scalar.<br><br>When defined as a cell array use standard PN sequences and a seed value: {PN, seed}. PN options for character vector or cell array are `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, and `'PN23'`.<br><br>If no seed is specified, the shift register is initialized with all ones. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| | | Examples for setting the `DataSource` field include:<br>• ...`CCTrCH.TrCh(1).DataSource = [1 0 0 1]` generates a sequence of transport blocks by looping the vector [1 0 0 1].<br>• ...`CCTrCH.TrCh(1).DataSource = 'PN9'` generates a transport channel data block with random seed = 511.<br>• ...`CCTrCH.TrCh(1).DataSource = {'PN9',5}` generates a transport channel data block with seed = 5. | |
| **`TrCH.ActiveDynamicPart`** | Required | Positive integer, vector | Active dynamic part, specified as a positive integer or a vector whose entries are positive integers in the interval [1, `length(DynamicPart)`]. |
| | | The `ActiveDynamicPart` field indicates the `DynamicPart` array index for the active transport format `(BlockSize, BlockSetSize)` from available combinations defined in `DynamicPart`. The selected transport format is used for data transmission in the current TTI. | |
| **`TrCH.DynamicPart`** | Required | Structure, structure array | Size of each transport block, specified as a structure or a structure array. |
| | | The `DynamicPart` fields, `BlockSize` and `BlockSetSize`, define the size of each transport block and the total bits per transport block set. As a pair `(BlockSize, BlockSetSize)` describe a transport format set. `DynamicPart` defines one or multiple transport format sets. | |
| **`TrCH.DynamicPart.BlockSize`** | Required | Positive integer | Transport block length, specified as a positive integer. |
| **`TrCH.DynamicPart.BlockSetSize`** | Required | Integer, multiple of `BlockSize` | Total number of bits in the transport block set. Implementation does not support multiple transport blocks, so by definition `BlockSize` is equal to `BlockSetSize`. |

**Note** When configuring the output structure to transmit the RMC 0kbps, as defined in TS 25.101, Section A.3.0 [1], a transport channel CRC is defined for transmission. The standard indicates DTCH transport block size = 0 and transport block set size = 0. Our implementation requires signalling transmission of a transport block to transmit a CRC. In the `umtsDownlinkWaveformGenerator`, one transport block of size zero is signaled by setting either `BlockSize or BlockSetSize` to `'0'`.

In our implementation, setting both `BlockSize` and `BlockSetSize` to zero signals transmission of zero transport blocks and a transport block size of zero and causes a transmission with no CRC.

#### PCCPCH Substructure

Include the `PCCPCH` substructure in the `config` structure to add the primary common control physical channel to the output structure. The `PCCPCH` substructure contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`Enable`** | Required | `'On'`, `'Off'` | Enable or disable the channel by specifying `Enable` as `'On'` or `'Off'`, respectively. |
| **`Power`** | Required | Float, `–inf`, `inf` | PCCPCH power in dB, specified as a float, `-inf`, or `inf`. |
| **`DataSource`** | Required | Binary scalar, binary vector, character vector, cell array, or string scalar | PCCPCH data source, specified as a binary scalar, a vector with binary entries, a character vector, a cell array, or a string scalar.<br><br>When specifying `DataSource` as a cell array, use standard PN sequences and a seed value: {PN, seed}. PN options for character vector or cell array are `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, and `'PN23'`. If no seed is specified, the shift register is initialized with all ones.<br><br>To enable BCH transport channel coding, specify `DataSource` as `'CCTrCH'`. |
| **`CCTrCH`** | Optional | Structure | See `CCTrCH` Substructure. |

#### SCCPCH Substructure

Include the `SCCPCH` substructure in the `config` structure to add the secondary common control physical channel to the output structure. The `SCCPCH` substructure contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`Enable`** | Required | `'On'`, `'Off'` | Enable or disable the channel by specifying `Enable` as `'On'` or `'Off'`, respectively. |
| **`SlotFormat`** | Required | Nonnegative integer | SCCPCH slot format number, specified as a nonnegative integer in the interval [0, 17]. |
| **`SpreadingCode`** | Required | Nonnegative integer<br><br>Valid range depends on slot format | SCCPCH spreading code, specified as a nonnegative integer in the interval [0, 255]. |
| **`SecondaryScramblingCode`** | Required | Nonnegative integer | SCCPCH secondary scrambling code index, specified as a nonnegative integer in the interval [0, 15]. |
| **`TimingOffset`** | Required | Nonnegative integer | Timing offset in terms of the number of chips (x256Tchip), specified as a nonnegative integer in the interval [0, 149]. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **Power** | Required | Float, -inf, inf | SCCPCH power in dB, specified as a float, -inf, or inf. |
| **TFCI** | Required | Nonnegative integer | Transport format combination indicator, specified as a nonnegative integer in the interval [0, 1023]. |
| **DataSource** | Required | Scalar, vector, character vector, cell array, or string scalar | SCCPCH data source, specified as a binary scalar, a vector with binary entries, a character vector, a cell array, or a string scalar.<br><br>When defined as a cell array use standard PN sequences and a seed value: {PN, seed}. PN options for character vector or cell array are 'PN9-ITU', 'PN9', 'PN11', 'PN15', and 'PN23'. If no seed is specified, the shift register is initialized with all ones.<br><br>To enable PCH/FACH transport channel coding, specify DataSource as 'CCTrCH'. |
| **CCTrCH** | Optional | Structure | See CCTrCH Substructure. |

#### PCPICH Substructure

Include the PCPICH substructure in the config structure to add the primary common pilot channel to the output structure. The PCPICH substructure contains the following fields.

| Parameter Field | Required or Optional | Values / Ranges / Notes | Description |
|---|---|---|---|
| **Enable** | Required | 'On', 'Off' | Enable or disable the channel by specifying Enable as 'On' or 'Off', respectively. |
| **Power** | Required | Float, –inf, inf | PCPICH power in dB, specified as a float, –inf, or inf. |

#### SCPICH Substructure

Include the SCPICH substructure in the config structure to add the secondary common pilot channel to the output structure. The SCPICH substructure contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **Enable** | Required | 'On', 'Off' | Enable or disable the channel by specifying Enable as 'On' or 'Off', respectively. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **SpreadingCode** | Required | Nonnegative integer | SCPICH spreading code, specified as a nonnegative integer in the interval [0, 255]. |
| **SecondaryScramblingCode** | Required | Nonnegative integer | SCPICH secondary scrambling code index, specified as a nonnegative integer in the interval [0, 15]. |
| **Power** | Required | Float, `-inf`, `inf` | SCPICH power in dB, specified as a float, `-inf`, or `inf`. |

#### PSCH Substructure

Include the `PSCH` substructure in the `config` structure to add the physical shared channel to the output structure. The `PSCH` substructure contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **Enable** | Required | `'On'`, `'Off'` | Enable or disable the channel by specifying `Enable` as `'On'` or `'Off'`, respectively. |
| **Power** | Required | Float, `-inf`, `inf` | PSCH power in dB, specified as a float, `-inf`, or `inf`. |

#### SSCH Substructure

Include the `SSCH` substructure in the `config` structure to add the secondary synchronization channel to the output structure. The `SSCH` substructure contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **Enable** | Required | `'On'`, `'Off'` | Enable or disable the channel by specifying `Enable` as `'On'` or `'Off'`, respectively. |
| **Power** | Required | Float, `-inf`, `inf` | SSCH power in dB, specified as a float, `-inf`, or `inf`. |

#### PICH Substructure

Include the `PICH` substructure in the `config` structure to add the page indicator channel to the output structure. The `PICH` substructure contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **Enable** | Required | 'On', 'Off' | Enable or disable the channel by specifying Enable as 'On' or 'Off', respectively. |
| **SpreadingCode** | Required | Nonnegative integer | PICH spreading code, specified as a nonnegative integer in the interval [0, 255]. |
| **TimingOffset** | Required | Nonnegative integer | Timing offset in terms of the number of chips (x256Tchip), specified as a nonnegative integer in the interval [0, 149]. |
| **Power** | Required | Float, –inf, inf | PICH power in dB, specified as a float, –inf, or inf. |
| **DataSource** | Required | Scalar, vector, character vector, cell array, or string scalar | PICH data source, specified as a binary scalar, a vector with binary entries, a character vector, a cell array, or a string scalar.<br><br>When defined as a cell array use standard PN sequences and a seed value: {PN, seed}. PN options for character vector or cell array are 'PN9-ITU', 'PN9', 'PN11', 'PN15', and 'PN23'. If no seed is specified, the shift register is initialized with all ones.<br><br>To use paging data, specify DataSource as 'PagingData'. |
| **Np** | Required | Positive integer | Number of paging indicators per frame, specified as one of the values 18, 36, 72, 144. |

#### HSDPA Substructure

To add high-speed downlink packet access (HSDPA) information and channels to the output structure, include the HSDPA substructure in the config structure. The HSDPA substructure contains the following fields.

| Parameter Field | Required or Optional | Values / Ranges / Notes | Description |
|---|---|---|---|
| **Enable** | Required | 'On', 'Off' | Enable or disable the HSDPA channels (HS-PDSCHs and HS-SCCH) by specifying Enable as 'On' or 'Off', respectively. |
| **CodeGroup** | Required | Positive integer | Number of channelization codes used simultaneously for HS-PDSCHs, specified as a positive integer in the interval [1, 16]. |

| Parameter Field | Required or Optional | Values / Ranges / Notes | Description |
|---|---|---|---|
| **`CodeOffset`** | Required | Nonnegative integer | Offset to the first channelization code to use for HS-PDSCHs, specified as a nonnegative integer in the interval [0, 15]. |
| **`Modulation`** | Required | `'QPSK'`, `'16QAM'`, `'64QAM'` | Symbol modulation, specified as one of the values `'QPSK'`, `'16QAM'`, or `'64QAM'`. |
| **`VirtualBufferCapacity`** | Required | Positive integer | Number of soft channel bits (or soft metric location) in a HARQ process for the H-Sets as defined in TS 36.101 Annex A.7. Specify `VirtualBufferCapacity` as a positive integer. The number of soft channel bits depends on the UE category as specified in TS 25.306 Section 5.1. The value of this parameter must match the number of soft channel bits in a HARQ process used in the test device or decoding software. |
| **`InterTTIDistance`** | Required | Positive integer | Transmission time interval in subframes. This interval is the distance between different HARQ transmissions to the same UE.<br>• A value of 1 indicates continuous HSDPA transmissions in every subframe to the UE under test.<br>• A value larger than 1 indicates the presence of gap subframes with no data transmission to the UE under test. |
| **`NHARQProcesses`** | Required | Positive integer | Total number of HARQ processes, specified as a positive integer in the interval [1, 8]. |

| Parameter Field | Required or Optional | Values / Ranges / Notes | Description |
|---|---|---|---|
| **`XrvSequence`** | Required | Nonnegative integer, vector of nonnegative integers | Redundancy and constellation version coding sequence, specified as a nonnegative integer, or a vector whose entries are nonnegative integers, in the interval [0, 7]. `XrvSequence` encodes the redundancy version parameters (*r*,*s*) and constellation version as defined in TS 25.212 Section 4.6. The encoding includes the constellation version only if the modulation scheme is 16QAM/64QAM. The values are used by each HARQ process for each transmission.<br>• A scalar indicates a single transmission.<br>• A vector indicates retransmissions. The new data indicator bit signalled by HS-SCCH stays the same and the redundancy version changes to the value encoded in the next element of `XrvSequence`.<br>When a HARQ process completes all transmissions corresponding to the `XrvSequence`, the new data indicator bit toggles between 0 and 1 indicating a new transmission. For more information, see TS 25.321 Section 11.6.1.3.<br>For sequences used for HSDPA H-Sets, see TS 25.101 Section 9. |
| **`UEId`** | Required | Nonnegative integer | UE identity, specified as a nonnegative integer in the interval [0, $2^{16}$ – 1]. |
| **`TransportBlockSizeId`** | Required | Nonnegative integer | Transport block size index ($x_{tbs}$) signaled on the HS-SCCH as defined in TS 25.212 Section 4.6. The calculation is based on the `HSDSCH.BlockSize` parameter used for transmission as defined in TS 25.321 Section 9.2.3 Annex A. Specify `TransportBlockSizeID` as a nonnegative integer in the interval [0, 63]. |
| **`HSSCCHSpreadingCode`** | Required | Nonnegative integer | HS-SCCH spreading code, specified as a nonnegative integer in the interval [0, 127]. |
| **`SecondaryScramblingCode`** | Required | Nonnegative integer | Secondary scrambling code index for HS-PDSCH and HS-SCCH channels, specified as a nonnegative integer in the interval [0, 15]. |

| Parameter Field | Required or Optional | Values / Ranges / Notes | Description |
|---|---|---|---|
| **HSPDSCHPower** | Required | Float, –inf, inf | HS-PDSCH power in dB, specified as a float, –inf, or inf. |
| **HSSCCHPower** | Required | Floatinf, inf | HS-SCCH power in dB, specified as a float, –inf, or inf. |
| **DataSource** | Required | Scalar, vector, character vector, cell array, or string scalar | HSDPA data source, specified as a binary scalar, a vector with binary entries, a character vector, a cell array, or a string scalar.<br><br>When specifying DataSource as a cell array, use standard PN sequences and a seed value in the form {PN, seed}. PN options for character vector or cell array are 'PN9-ITU', 'PN9', 'PN11', 'PN15', and 'PN23'. If no seed is specified, the shift register is initialized with all ones.<br><br>To enable HS-DSCH transport channel coding, specify DataSource as 'HSDSCH'. |
| **HSDSCH** | Optional | Not present or a structure | HS-DSCH transport channel configuration, specified as a structure. |
| The following fields are required only if the HSDSCH substructure is present. | | | |
| **HSDSCH.BlockSize** | Required | Nonnegative integer | Transport block size, specified as a nonnegative integer. |
| **HSDSCH.DataSource** | Required | Scalar, vector, character vector, cell array, or string scalar | HS-DSCH transport data source, specified as a binary scalar, a vector with binary entries, a cell array, or a string scalar.<br><br>When defined as a cell array use standard PN sequences and a seed value: {PN, seed}. PN options for character vector or cell array are 'PN9-ITU', 'PN9', 'PN11', 'PN15', and 'PN23'.<br><br>If no seed is specified, the shift register is initialized with all ones. |

In the generator, the HSPDA functionality creates continuous HS-PDSCH and HS-SCCH transmissions. This functionality supports the HSPDA H-Set fixed reference channels where a multi-HARQ reference transmission sequence is defined. The multi-HARQ reference transmission sequence is masked with the same RNTI, directed at a single UE specified by the UEId parameter. The NHARQProcesses and InterTTIDistance parameters define the reference transmission frequency to the UE. Any gaps between the reference subframes are filled with additional HS-PDSCH/HS-SCCH subframes. These subframes are masked with a complementary RNTI, directed at a different UE defined as xor(UEId,65535). The NHARQProcesses parameter gives the numbers of HARQ processes used in the reference transmission. The number of gap subframes between each transport

block transmission or retransmission for different HARQ processes is `InterTTIDistance-1`. Due to the HARQ ACK-NACK feedback signaling requirements, the gap between the transmissions of the same HARQ process should be no less than six subframes.

The `HSDPA.DataSource` parameter controls the data transmitted on the reference PDSCH and HS-SCCH physical channels. If `HSDPA.DataSource` is set to `'HSDSCH'`, the reference PDSCH data comes from an HS-DSCH transport channel and the HS-SCCH channel carries the associated control information. In this case, the source to the HS-DSCH transport channel is parameterized by the fields in the `HSDSCH` substructure data (transport block size and data stream). This reference data is also used to fill the non-reference gap subframes:

- The gap HS-PDSCH subframes are filled with the same HS-DSCH encoded data used for reference transmission. The encoded data is scrambled according to the subframe.
- The gap HS-SCCH subframes are filled with the encoded control information using the complimentary RNTI.

The HS-SCCH transmission is aligned with the scrambling boundary. The HS-PDSCH transmission begins `2×Tslot = 5120` chips after the start of the HS-SCCH (see TS 25.211 Section 7.8). To fill the first two slots in the generated waveform, the HS-PDSCH wraps around for the last subframe.

The `virtualBufferCapacity` parameter must match the number of soft channel bits in a HARQ process used in the test device or decoding software.

#### OCNS Substructure

Include the `OCNS` substructure in the `config` structure to add orthogonal channel noise source information to the output structure. The `OCNS` substructure contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`Enable`** | Required | `'On'`, `'Off'` | Enable or disable the channel by specifying `Enable` as `'On'` or `'Off'`, respectively. |
| **`Power`** | Required | Float, `–inf`, or `inf` | OCNS overall power in dB, specified as a float, `-inf`, or `inf`. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`OCNSType`** | Required | Character vector or string scalar | If `OCNS` is enabled, `OCNSType` specifies which OCNS configuration to use. The `OCNS` substructure and `OCNSType` field are used to generate:<br>• DPCHs, defined as OCNS channels in TS 25.101.<br>• DPCHs, HS-PDSCHs, and HS-SCCHs, defined for test models in TS 25.141, Section 6.<br>For RMCs and H-Sets, specify `OCNSType` as one of these values: `'RMC_16DPCH'`, `'H-Set_6DPCH'`, `'H-Set_4DPCH'`<br>For Test Model DPCH and HS-PDSCH/HS-SCCH sets, specify `OCNSType` as one of these values: `'TM1_4DPCH'`, `'TM1_8DPCH'`, `'TM1_16DPCH'`, `'TM1_32DPCH'`, `'TM1_64DPCH'`, `'TM2_3DPCH'`, `'TM3_4DPCH'`, `'TM3_8DPCH'`, `'TM3_16DPCH'`, `'TM3_32DPCH'`, `'TM5_4DPCH_4HSPDSCH'`, `'TM5_6DPCH_2HSPDSCH'`, `'TM5_14DPCH_4HSPDSCH'`, `'TM5_30DPCH_8HSPDSCH'`, `'TM6_4DPCH_4HSPDSCH'`, `'TM6_30DPCH_8HSPDSCH'`.<br>For test model generation, set the corresponding channel configuration `Enable` field to `'Off'`. |

# Version History

**Introduced in R2015a**

## References

[1] 3GPP TS 25.101. “Universal Mobile Telecommunications System (UMTS); User Equipment (UE) Radio Transmission and Reception (FDD).” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 25.141. “Universal Mobile Telecommunications System (UMTS); Base Station (BS) Conformance Testing (FDD).” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[3] 3GPP TS 25.211. “Universal Mobile Telecommunications System (UMTS); Physical channels and mapping of transport channels onto physical channels (FDD).” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[4] 3GPP TS 25.212. “Universal Mobile Telecommunications System (UMTS); Multiplexing and channel coding (FDD).” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[5] 3GPP TS 25.306. “Universal Mobile Telecommunications System (UMTS); UE Radio Access capabilities.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[6] 3GPP TS 25.321. “Universal Mobile Telecommunications System (UMTS); Medium Access Control (MAC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[7] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`umtsDownlinkWaveformGenerator` | `umtsUplinkReferenceChannels` | `umtsUplinkWaveformGenerator`

**Topics**

“Downlink Reference Channel and Waveform Generation Parameter Structures”

# umtsDownlinkWaveformGenerator

UMTS downlink waveform generation

## Syntax

```
waveform = umtsDownlinkWaveformGenerator(config)
```

## Description

`waveform = umtsDownlinkWaveformGenerator(config)` returns the Universal Mobile Telecommunications Service (UMTS) downlink waveform, `waveform`, defined by the configuration structure, `config`. This function supports Wideband Code Division Multiple Access (W-CDMA), High-Speed Downlink Packet Access (HSDPA), and Evolved High-Speed Packet Access (HSPA+) waveform generation. The top-level parameters and lower-level substructures of `config` characterize the waveform and channel properties of the `umtsDownlinkWaveformGenerator` function output. The `config` input is generated using the `umtsDownlinkReferenceChannels` function; `config` includes top-level parameters and substructures to describe the different channels to include in the waveform. The top-level parameters of `config` are: `TotFrames`, `PrimaryScramblingCode`, `FilterType`, `OversamplingRatio`, and `NormalizedPower`. To enable the specific channels, you can add associated substructures: `DPCH`, `PCCPCH`, `SCCPCH`, `PCPICH`, `SCPICH`, `PSCH`, `SSCH`, `PICH`, `HSDPA`, and `OCNS`.

---

**Note** Include an interfering downlink W-CDMA noise source by initializing the `OCNS` substructure. Specify the orthogonal channel noise source (OCNS) parameters using the appropriate 3GPP definition,

- RMC OCNS channels are defined in TS 25.101, Table C.6 [1]
- H-Set OCNS channels are defined in TS 25.141, Tables C.13, and C.13A [2]
- Test model DPCHs and HS-PDSCH/HS-SCCH channels are defined in TS 25.141, Section 6.1.1 [2]

---

## Examples

### UMTS Downlink Waveform Generation

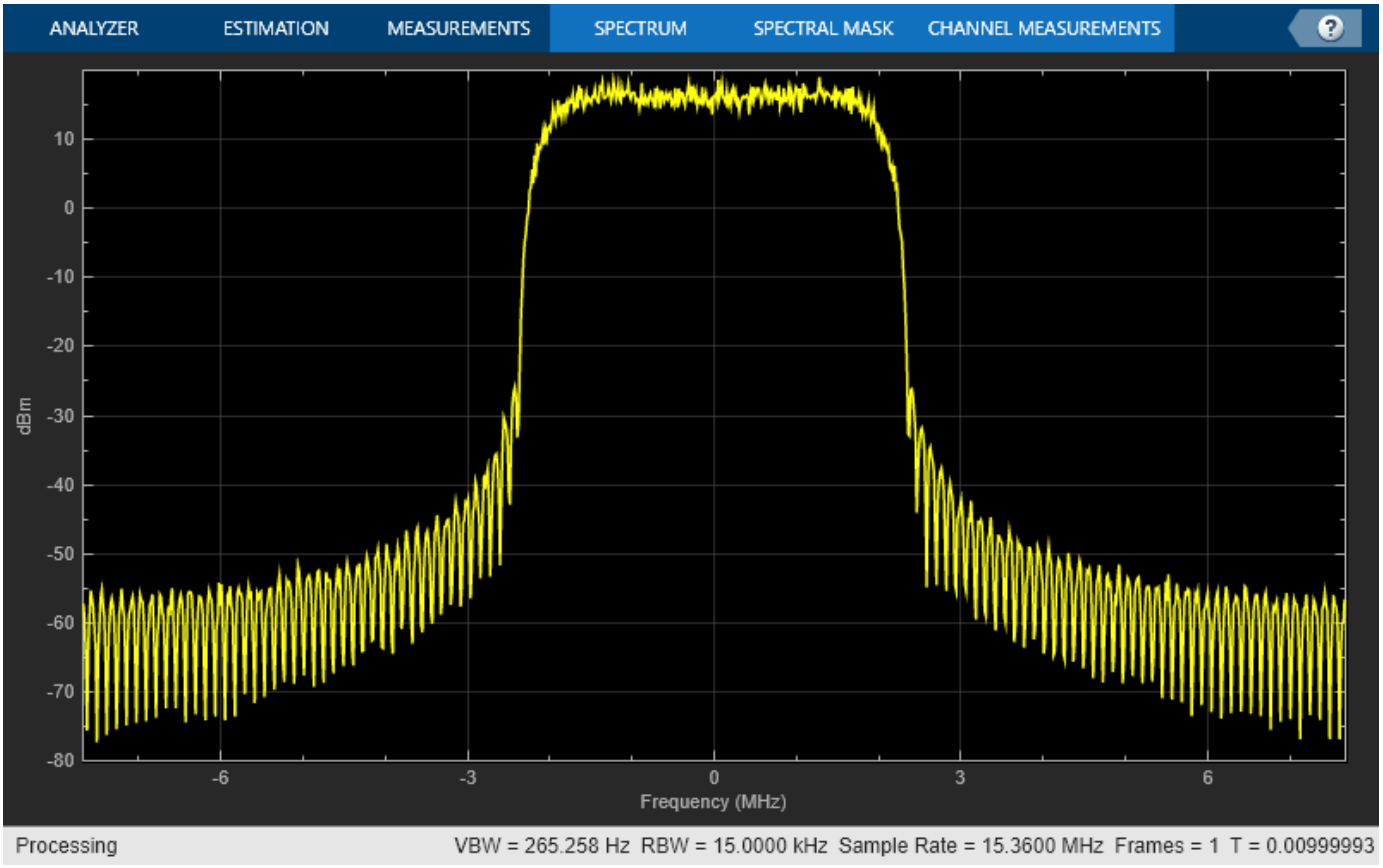
Initialize a `'QPSK'`, `'H-Set1'` FRC reference channel and generate the UMTS waveform that corresponds to these settings.

Generate the configuration structure, `frcStruct`.

```
rc = 'H-Set1';
modulation = 'QPSK';
frcStruct = umtsDownlinkReferenceChannels(rc, modulation);
```

Generate the desired waveform using `frcStruct` as the input to the waveform generation function. Create a spectrum analyzer object sampling at `chiprate` x `OversamplingRatio`. Plot the waveform.

```
waveform = umtsDownlinkWaveformGenerator(frcStruct);
saScope = spectrumAnalyzer(SampleRate=3.84e6*frcStruct.OversamplingRatio);
saScope(waveform);
```




## Input Arguments

### `config` — Definition of the channels included by waveform generator

structure

#### Top-Level Parameters and Substructures

Definition of the channels included by the waveform generator, specified as a structure.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`TotFrames`** | Required | Nonnegative integer | Total number of 10 ms frames to be generated, specified as a nonnegative integer. The default is 1. |
| **`PrimaryScramblingCode`** | Required | Nonnegative integer | Primary scrambling code index, specified as a nonnegative integer in the interval [0, 511]. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **FilterType** | Required | 'RRC' (default), 'Off' | Enable or disable the RRC filter by setting FilterType to 'RRC' or 'off'. respectively. |
| **OversamplingRatio** | Required | Nonnegative integer | Oversampling ratio, specified as a nonnegative integer. |
| **NormalizedPower** | Required | Float, –inf, inf,'Off' | Overall waveform power in dBW relative to 1 ohm, specified as a float, –inf, inf, or 'Off'. Setting NormalizedPower to 'Off' disables power normalization. |
| **DPCH** | Optional | Not present, structure, or structure array | See DPCH Substructure. |
| **PCCPCH** | Optional | Not present or structure | See PCCPCH Substructure. |
| **SCCPCH** | Optional | Not present or structure | See SCCPCH Substructure. |
| **PCPICH** | Optional | Not present or structure | See PCPICH Substructure. |
| **SCPICH** | Optional | Not present or structure | See SCPICH Substructure. |
| **PSCH** | Optional | Not present or structure | See PSCH Substructure. |
| **SSCH** | Optional | Not present or structure | See SSCH Substructure. |
| **PICH** | Optional | Not present or structure | See PICH Substructure. |
| **HSDPA** | Optional | Not present or structure | See HSDPA Substructure. |
| **OCNS** | Optional | Not present or structure | See OCNS Substructure. |

#### DPCH Substructure

To add dedicated physical channels (DPCHs) to the output structure, include the DPCH substructure in the config structure. The DPCH substructure contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **Enable** | Required | 'On', 'Off' | Enable or disable the channel by setting Enable to 'On' or 'Off', respectively. |
| **SlotFormat** | Required | Nonnegative integer | DPCH slot format number, specified as a nonnegative integer in the interval [0, 16]. |
| **SpreadingCode** | Required | Nonnegative integer | DPCH spreading code, specified as a nonnegative integer in the interval [0, 512]. For multicode transmission, SpreadingCode is the first DPCH code. |
| **NMulticodes** | Required | Positive integer | Number of DPCHs, specified as 1, 2, 3, 4, 5, or 6. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **SecondaryScramblingCode** | Required | Nonnegative integer | DPCH secondary scrambling code index, specified as a nonnegative integer in the interval [0, 15]. |
| **TimingOffset** | Required | Nonnegative integer | The timing offset in terms of the number of chips (x256Tchip), specified as a nonnegative integer in the interval [0, 149]. |
| **Power** | Required | Float, `-inf`, `inf` | Channel power in dB, specified as a float, `-inf`, or `inf`. |
| **TPCData** | Required | Binary scalar, binary vector | Transmit Power Control data, specified as a binary scalar or a vector with binary entries. |
| **TFCI** | Required | Nonnegative integer | Transport Format Combination Indicator (TFCI), specified as a nonnegative integer in the interval [0, 1023]. |
| **DataSource** | Required | Binary scalar, binary vector, character vector, cell array, or string scalar | DPCH data source, specified as a binary scalar, a vector with binary entries, a character vector, a cell array, or a string scalar.<br><br>When specifying `DataSource` as a cell array, use standard PN sequences and a seed value: {PN, seed}. PN options for character vector or cell array are `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, and `'PN23'`. If no seed is specified, the shift register is initialized with all ones.<br><br>To enable transport channel coding, specify `DataSource` as `'CCTrCH'`. |
| **CCTrCH** | Optional | Structure | See CCTrCH Substructure. |

**CCTrCH Substructure**

Include a `CCTrCH` substructure instance individually for `DPCH`, `PCCPCH`, and/or `SCCPCH` substructures. Separate instances of a coded composite transport channel (CCTrCH) are added to the output structures of the DPCH, P-CCPCH, and/or S-CCPCH physical channel definitions. When the `CCTrCH` substructure is included, it contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **Name** | Optional | Character vector, string scalar<br><br>Default depends on the physical channel specified | Name assigned to the CCTrCH, specified as a character vector or a string scalar. Functions do not use the `Name` field. Therefore, you can redefine the content with no consequence. |
| **DTXPosition** | Required | `'fixed'`, `'flexible'` | Specifies the DTX position, specified as `'fixed'` or `'flexible'`. |
| **TrCH** | Required | Structure, structure array | Transport channels in the CCTrCH, specified as a structure or a structure array. |
| **TrCH.Name** | Required | Character vector, string scalar<br><br>Default depends on the physical channel specified | Name assigned to the TrCH, specified as a character vector or a string scalar. Functions do not use the `Name` field. Therefore, you can redefine the content with no consequence. |
| **TrCH.CRC** | Required | Character vector, string scalar | Cyclic redundancy check (CRC) polynomial specifier, specified as one of these values: `'0'`, `'8'`, `'12'`, `'16'`, or `'24'`. |
| **TrCH.TTI** | Required | Positive integer | Transmission time interval (TTI) in milliseconds, specified as 10, 20, 40, or 80. |
| **TrCH.CodingType** | Required | `'turbo'`, `'conv2'`, `'conv3'` | Channel coding type and rate, specified as `'turbo'`, `'conv2'`, or `'conv3'`. |
| **TrCH.RMA** | Required | Positive integer | Rate matching attribute value, specified as a positive integer in the interval [1, 256]. |
| **TrCH.DataSource** | Required | Binary scalar, binary vector, character vector, cell array, or string scalar | Transport channel data source, specified as a binary scalar, a vector with binary entries, a cell array, or a string scalar.<br><br>When defined as a cell array use standard PN sequences and a seed value: {PN, seed}. PN options for character vector or cell array are `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, and `'PN23'`.<br><br>If no seed is specified, the shift register is initialized with all ones. |
| | | Examples for setting the `DataSource` field include:<br><br>• ...`CCTrCH.TrCh(1).DataSource = [1 0 0 1]` generates a sequence of transport blocks by looping the vector [1 0 0 1].<br>• ...`CCTrCH.TrCh(1).DataSource = 'PN9'` generates a transport channel data block with random seed = 511.<br>• ...`CCTrCH.TrCh(1).DataSource = {'PN9',5}` generates a transport channel data block with seed = 5. | |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **TrCH.ActiveDynamicPart** | Required | Positive integer, vector | Active dynamic part, specified as a positive integer or a vector whose entries are positive integers in the interval [1, `length(DynamicPart)`]. |
| | | The `ActiveDynamicPart` field indicates the `DynamicPart` array index for the active transport format `(BlockSize, BlockSetSize)` from available combinations defined in `DynamicPart`. The selected transport format is used for data transmission in the current TTI. | |
| **TrCH.DynamicPart** | Required | Structure, structure array | Size of each transport block, specified as a structure or a structure array. |
| | | The `DynamicPart` fields, `BlockSize` and `BlockSetSize`, define the size of each transport block and the total bits per transport block set. As a pair `(BlockSize, BlockSetSize)` describe a transport format set. `DynamicPart` defines one or multiple transport format sets. | |
| **TrCH.DynamicPart.BlockSize** | Required | Positive integer | Transport block length, specified as a positive integer. |
| **TrCH.DynamicPart.BlockSetSize** | Required | Integer, multiple of `BlockSize` | Total number of bits in the transport block set. Implementation does not support multiple transport blocks, so by definition `BlockSize` is equal to `BlockSetSize`. |

**Note** When configuring the output structure to transmit the RMC 0kbps as defined in TS 25.101, Section A.3.0 [1], a transport channel CRC is defined for transmission. The standard indicates DTCH transport block size = 0 and transport block set size = 0. Our implementation requires signaling transmission of a transport block to transmit a CRC. In the `umtsDownlinkWaveformGenerator`, one transport block of size zero is signaled by setting either `BlockSize or BlockSetSize` to `'0'`. Setting both `BlockSize` and `BlockSetSize` to `'0'` signals `'0'` transport block of size `'0'` and no CRC is transmitted.

#### PCCPCH Substructure

To add the primary common control physical channel (PCCPCH) to the output structure, include the `PCCPCH` substructure in the `config` structure. The `PCCPCH` substructure contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **Enable** | Required | `'On'`, `'Off'` | Enable or disable the channel by specifying `Enable` as `'On'` or `'Off'`, respectively. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **Power** | Required | Float, –inf, inf | PCCPCH power in dB, specified as a float, -inf, or inf. |
| **DataSource** | Required | Binary scalar, binary vector, character vector, cell array, or string scalar | PCCPCH data source, specified as a binary scalar, a vector with binary entries, a character vector, a cell array, or a string scalar.<br><br>When specifying DataSource as a cell array, use standard PN sequences and a seed value: {PN, seed}. PN options for character vector or cell array are 'PN9-ITU', 'PN9', 'PN11', 'PN15', and 'PN23'. If no seed is specified, the shift register is initialized with all ones.<br><br>To enable BCH transport channel coding, specify DataSource as 'CCTrCH'. |
| **CCTrCH** | Optional | Structure | See CCTrCH Substructure. |

#### SCCPCH Substructure

To add the secondary common control physical channel (SCCPCH) to the output structure, include the SCCPCH substructure in the config structure. The SCCPCH substructure contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **Enable** | Required | 'On', 'Off' | Enable or disable the channel by specifying Enable as 'On' or 'Off', respectively. |
| **SlotFormat** | Required | Nonnegative integer | SCCPCH slot format number, specified as a nonnegative integer in the interval [0, 17]. |
| **SpreadingCode** | Required | Nonnegative integer<br><br>Valid range depends on slot format | SCCPCH spreading code, specified as a nonnegative integer in the interval [0, 255]. |
| **SecondaryScramblingCode** | Required | Nonnegative integer | SCCPCH secondary scrambling code index, specified as a nonnegative integer in the interval [0, 15]. |
| **TimingOffset** | Required | Nonnegative integer | Timing offset in terms of the number of chips (x256Tchip), specified as a nonnegative integer in the interval [0, 149]. |
| **Power** | Required | Float, -inf, inf | SCCPCH power in dB, specified as a float, -inf, or inf. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **TFCI** | Required | Nonnegative integer | Transport format combination indicator, specified as a nonnegative integer in the interval [0, 1023]. |
| **DataSource** | Required | Scalar, vector, character vector, cell array, or string scalar | SCCPCH data source, specified as a binary scalar, a vector with binary entries, a character vector, a cell array, or a string scalar.<br><br>When defined as a cell array use standard PN sequences and a seed value: {PN, seed}. PN options for character vector or cell array are `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, and `'PN23'`. If no seed is specified, the shift register is initialized with all ones.<br><br>To enable PCH/FACH transport channel coding, specify `DataSource` as `'CCTrCH'`. |
| **CCTrCH** | Optional | Structure | See `CCTrCH` Substructure. |

#### PCPICH Substructure

To add the primary common pilot channel to the output structure, include the `PCPICH` substructure in the `config` structure. The `PCPICH` substructure contains the following fields.

| Parameter Field | Required or Optional | Values / Ranges / Notes | Description |
|---|---|---|---|
| **Enable** | Required | `'On'`, `'Off'` | Enable or disable the channel by specifying `Enable` as `'On'` or `'Off'`, respectively. |
| **Power** | Required | Float, `-inf`, `inf` | PCPICH power in dB, specified as a float, `-inf`, or `inf`. |

#### SCPICH Substructure

To add the secondary common pilot channel (SCPICH) to the output structure, include the `SCPICH` substructure in the `config` structure. The `SCPICH` substructure contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **Enable** | Required | `'On'`, `'Off'` | Enable or disable the channel by specifying `Enable` as `'On'` or `'Off'`, respectively. |
| **SpreadingCode** | Required | Nonnegative integer | SCPICH spreading code, specified as a nonnegative integer in the interval [0, 255]. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **SecondaryScramblingCode** | Required | Nonnegative integer | SCPICH secondary scrambling code index, specified as a nonnegative integer in the interval [0, 15]. |
| **Power** | Required | Float, `-inf`, `inf` | SCPICH power in dB, specified as a float, `-inf`, or `inf`. |

#### PSCH Substructure

To add the physical shared channel (PSCH) to the output structure, include the `PSCH` substructure in the `config` structure. The `PSCH` substructure contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **Enable** | Required | `'On'`, `'Off'` | Enable or disable the channel by specifying `Enable` as `'On'` or `'Off'`, respectively. |
| **Power** | Required | Float, `-inf`, `inf` | PSCH power in dB, specified as a float, `-inf`, or `inf`. |

#### SSCH Substructure

To add the secondary synchronization channel (SSCH) to the output structure, include the `SSCH` substructure in the `config` structure. The `SSCH` substructure contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **Enable** | Required | `'On'`, `'Off'` | Enable or disable the channel by specifying `Enable` as `'On'` or `'Off'`, respectively. |
| **Power** | Required | Float, `-inf`, `inf` | SSCH power in dB, specified as a float, `-inf`, or `inf`. |

#### PICH Substructure

To add the page indicator channel (PICH) to the output structure, include the `PICH` substructure in the `config` structure. The `PICH` substructure contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **Enable** | Required | `'On'`, `'Off'` | Enable or disable the channel by specifying `Enable` as `'On'` or `'Off'`, respectively. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **SpreadingCode** | Required | Nonnegative integer | PICH spreading code, specified as a nonnegative integer in the interval [0, 255]. |
| **TimingOffset** | Required | Nonnegative integer | Timing offset in terms of the number of chips (x256Tchip), specified as a nonnegative integer in the interval [0, 149]. |
| **Power** | Required | Float, `-inf`, `inf` | PICH power in dB, specified as a float, `-inf`, or `inf`. |
| **DataSource** | Required | Scalar, vector, character vector, cell array, or string scalar | PICH data source, specified as a binary scalar, a vector with binary entries, a character vector, a cell array, or a string scalar.<br><br>When defined as a cell array use standard PN sequences and a seed value: {PN, seed}. PN options for character vector or cell array are `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, and `'PN23'`. If no seed is specified, the shift register is initialized with all ones.<br><br>To use paging data, specify `DataSource` as `'PagingData'`. |
| **Np** | Required | Positive integer | Number of paging indicators per frame, specified as one of the values 18, 36, 72, 144. |

#### HSDPA Substructure

To add high-speed downlink packet access (HSDPA) information and channels to the output structure, include the `HSDPA` substructure in the `config` structure. The `HSDPA` substructure contains the following fields.

| Parameter Field | Required or Optional | Values / Ranges / Notes | Description |
|---|---|---|---|
| **Enable** | Required | `'On'`, `'Off'` | Enable or disable the HSDPA channels (HS-PDSCHs and HS-SCCH) by specifying `Enable` as `'On'` or `'Off'`, respectively. |
| **CodeGroup** | Required | Positive integer | Number of channelization codes used simultaneously for HS-PDSCHs, specified as a positive integer in the interval [1, 16]. |
| **CodeOffset** | Required | Nonnegative integer | Offset to the first channelization code to use for HS-PDSCHs, specified as a nonnegative integer in the interval [0, 15]. |

| Parameter Field | Required or Optional | Values / Ranges / Notes | Description |
|---|---|---|---|
| **`Modulation`** | Required | `'QPSK'`, `'16QAM'`, `'64QAM'` | Symbol modulation, specified as one of the values `'QPSK'`, `'16QAM'`, or `'64QAM'`. |
| **`VirtualBufferCapacity`** | Required | Positive integer | Number of soft channel bits (or soft metric location) in a HARQ process for the H-Sets as defined in TS 36.101 Annex A.7. Specify `VirtualBufferCapacity` as a positive integer. The number of soft channel bits depends on the UE category as specified in TS 25.306 Section 5.1. The value of this parameter must match the number of soft channel bits in a HARQ process used in the test device or decoding software. |
| **`InterTTIDistance`** | Required | Positive integer | Transmission time interval in subframes. This interval is the distance between different HARQ transmissions to the same UE.<br>• A value of 1 indicates continuous HSDPA transmissions in every subframe to the UE under test.<br>• A value larger than 1 indicates the presence of gap subframes with no data transmission to the UE under test. |
| **`NHARQProcesses`** | Required | Positive integer | Total number of HARQ processes, specified as a positive integer in the interval [1, 8]. |

| Parameter Field | Required or Optional | Values / Ranges / Notes | Description |
|---|---|---|---|
| **`XrvSequence`** | Required | Nonnegative integer, vector of nonnegative integers | Redundancy and constellation version coding sequence, specified as a nonnegative integer, or a vector whose entries are nonnegative integers, in the interval [0, 7]. `XrvSequence` encodes the redundancy version parameters (*r*,*s*) and constellation version as defined in TS 25.212 Section 4.6. The encoding includes the constellation version only if the modulation scheme is 16QAM/64QAM. The values are used by each HARQ process for each transmission.<br>• A scalar indicates a single transmission.<br>• A vector indicates retransmissions. The new data indicator bit signalled by HS-SCCH stays the same and the redundancy version changes to the value encoded in the next element of `XrvSequence`.<br>When a HARQ process completes all transmissions corresponding to the `XrvSequence`, the new data indicator bit toggles between 0 and 1 indicating a new transmission. For more information, see TS 25.321 Section 11.6.1.3.<br>For sequences used for HSDPA H-Sets, see TS 25.101 Section 9. |
| **`UEId`** | Required | Nonnegative integer | UE identity, specified as a nonnegative integer in the interval [0, $2^{16}$ – 1]. |
| **`TransportBlockSizeId`** | Required | Nonnegative integer | Transport block size index ($x_{tbs}$) signaled on the HS-SCCH as defined in TS 25.212 Section 4.6. The calculation is based on the `HSDSCH.BlockSize` parameter used for transmission as defined in TS 25.321 Section 9.2.3 Annex A. Specify `TransportBlockSizeID` as a nonnegative integer in the interval [0, 63]. |
| **`HSSCCHSpreadingCode`** | Required | Nonnegative integer | HS-SCCH spreading code, specified as a nonnegative integer in the interval [0, 127]. |
| **`SecondaryScramblingCode`** | Required | Nonnegative integer | Secondary scrambling code index for HS-PDSCH and HS-SCCH channels, specified as a nonnegative integer in the interval [0, 15]. |

| Parameter Field | Required or Optional | Values / Ranges / Notes | Description |
|---|---|---|---|
| **HSPDSCHPower** | Required | Float, –inf, inf | HS-PDSCH power in dB, specified as a float, –inf, or inf. |
| **HSSCCHPower** | Required | Floatinf, inf | HS-SCCH power in dB, specified as a float, –inf, or inf. |
| **DataSource** | Required | Scalar, vector, character vector, cell array, or string scalar | HSDPA data source, specified as a binary scalar, a vector with binary entries, a character vector, a cell array, or a string scalar.<br><br>When specifying DataSource as a cell array, use standard PN sequences and a seed value in the form {PN, seed}. PN options for character vector or cell array are 'PN9-ITU', 'PN9', 'PN11', 'PN15', and 'PN23'. If no seed is specified, the shift register is initialized with all ones.<br><br>To enable HS-DSCH transport channel coding, specify DataSource as 'HSDSCH'. |
| **HSDSCH** | Optional | Not present or a structure | HS-DSCH transport channel configuration, specified as a structure. |
| The following fields are required only if the HSDSCH substructure is present. | | | |
| **HSDSCH.BlockSize** | Required | Nonnegative integer | Transport block size, specified as a nonnegative integer. |
| **HSDSCH.DataSource** | Required | Scalar, vector, character vector, cell array, or string scalar | HS-DSCH transport data source, specified as a binary scalar, a vector with binary entries, a cell array, or a string scalar.<br><br>When defined as a cell array use standard PN sequences and a seed value: {PN, seed}. PN options for character vector or cell array are 'PN9-ITU', 'PN9', 'PN11', 'PN15', and 'PN23'.<br><br>If no seed is specified, the shift register is initialized with all ones. |

In the generator, the HSPDA functionality creates continuous HS-PDSCH and HS-SCCH transmissions. This functionality supports the HSPDA H-Set fixed reference channels where a multi-HARQ reference transmission sequence is defined. The multi-HARQ reference transmission sequence is masked with the same RNTI, directed at a single UE specified by the UEId parameter. The NHARQProcesses and InterTTIDistance parameters define the reference transmission frequency to the UE. Any gaps between the reference subframes are filled with additional HS-PDSCH/HS-SCCH subframes. These subframes are masked with a complementary RNTI, directed at a different UE defined as xor(UEId,65535). The NHARQProcesses parameter gives the numbers of HARQ processes used in the reference transmission. The number of gap subframes between each transport

block transmission or retransmission for different HARQ processes is `InterTTIDistance-1`. Due to the HARQ ACK-NACK feedback signaling requirements, the gap between the transmissions of the same HARQ process should be no less than six subframes.

The `HSDPA.DataSource` parameter controls the data transmitted on the reference PDSCH and HS-SCCH physical channels. If `HSDPA.DataSource` is set to `'HSDSCH'`, the reference PDSCH data comes from an HS-DSCH transport channel and the HS-SCCH channel carries the associated control information. In this case, the source to the HS-DSCH transport channel is parameterized by the fields in the `HSDSCH` substructure data (transport block size and data stream). This reference data is also used to fill the non-reference gap subframes:

- The gap HS-PDSCH subframes are filled with the same HS-DSCH encoded data used for reference transmission. The encoded data is scrambled according to the subframe.
- The gap HS-SCCH subframes are filled with the encoded control information using the complimentary RNTI.

The HS-SCCH transmission is aligned with the scrambling boundary. The HS-PDSCH transmission begins `2×Tslot = 5120` chips after the start of the HS-SCCH (see TS 25.211 Section 7.8). To fill the first two slots in the generated waveform, the HS-PDSCH wraps around for the last subframe.

The `virtualBufferCapacity` parameter must match the number of soft channel bits in a HARQ process used in the test device or decoding software.

#### OCNS Substructure

To add orthogonal channel noise source (OCNS) information to the output structure, include the `OCNS` substructure in the `config` structure. The `OCNS` substructure contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`Enable`** | Required | `'On'`, `'Off'` | Enable or disable the channel by specifying `Enable` as `'On'` or `'Off'`, respectively. |
| **`Power`** | Required | Float, `–inf`, or `inf` | OCNS overall power in dB, specified as a float, `-inf`, or `inf`. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`OCNSType`** | Required | Character vector or string scalar | If `OCNS` is enabled, `OCNSType` specifies which OCNS configuration to use. The `OCNS` substructure and `OCNSType` field are used to generate:<br>• DPCHs, defined as OCNS channels in TS 25.101.<br>• DPCHs, HS-PDSCHs, and HS-SCCHs, defined for test models in TS 25.141, Section 6.<br>For RMCs and H-Sets, specify `OCNSType` as one of these values: `'RMC_16DPCH'`, `'H-Set_6DPCH'`, `'H-Set_4DPCH'`<br>For Test Model DPCH and HS-PDSCH/HS-SCCH sets, specify `OCNSType` as one of these values: `'TM1_4DPCH'`, `'TM1_8DPCH'`, `'TM1_16DPCH'`, `'TM1_32DPCH'`, `'TM1_64DPCH'`, `'TM2_3DPCH'`, `'TM3_4DPCH'`, `'TM3_8DPCH'`, `'TM3_16DPCH'`, `'TM3_32DPCH'`, `'TM5_4DPCH_4HSPDSCH'`, `'TM5_6DPCH_2HSPDSCH'`, `'TM5_14DPCH_4HSPDSCH'`, `'TM5_30DPCH_8HSPDSCH'`, `'TM6_4DPCH_4HSPDSCH'`, `'TM6_30DPCH_8HSPDSCH'`.<br>For test model generation, set the corresponding channel configuration `Enable` field to `'Off'`. |

## Output Arguments

**`waveform` — Modulated baseband waveform containing the UMTS physical channels**
complex vector array

Modulated baseband waveform containing the UMTS physical channels, returned as a complex vector array, sampled at (3.84 × `config.OversamplingRatio`) MHz.

Data Types: `double`

# Version History

**Introduced in R2015a**

## References

[1] 3GPP TS 25.101. “Universal Mobile Telecommunications System (UMTS); User Equipment (UE) Radio Transmission and Reception (FDD).” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 25.141. “Universal Mobile Telecommunications System (UMTS); Base Station (BS) Conformance Testing (FDD).” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[3] 3GPP TS 25.211. “Universal Mobile Telecommunications System (UMTS); Physical channels and mapping of transport channels onto physical channels (FDD).” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[4] 3GPP TS 25.212. “Universal Mobile Telecommunications System (UMTS); Multiplexing and channel coding (FDD).” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[5] 3GPP TS 25.306. “Universal Mobile Telecommunications System (UMTS); UE Radio Access capabilities.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[6] 3GPP TS 25.321. “Universal Mobile Telecommunications System (UMTS); Medium Access Control (MAC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[7] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`umtsDownlinkReferenceChannels` | `umtsUplinkReferenceChannels` | `umtsUplinkWaveformGenerator`

### Topics

“Downlink Reference Channel and Waveform Generation Parameter Structures”

# umtsUplinkReferenceChannels

UMTS uplink measurement channel definition

## Syntax

```
config = umtsUplinkReferenceChannels(rc)
```

## Description

`config = umtsUplinkReferenceChannels(rc)` returns a structure containing the configuration parameters for the UMTS uplink reference channel defined by `rc`. The output structure, `config`, contains the configuration parameters required by `umtsUplinkWaveformGenerator` to generate an uplink reference channel waveform. `umtsUplinkReferenceChannels` uses, `rc`, to initialize a configuration data structure that is compliant with one of the reference channels defined in the following 3GPP standards:

- Uplink RMC configurations are defined in TS 25.101, Annex A.2 [1].
- Uplink E-DPDCH FRC configurations are as defined in TS 25.141, Annex 10 [2].

## Examples

### UMTS Uplink Reference Channel Initialization

Initialize a `'RMC12.2kbps'` reference channel.

Generate the configuration structure, `config`.

```
rc = 'RMC12.2kbps';
config = umtsUplinkReferenceChannels(rc);
```

The output from `umtsUplinkReferenceChannels` provides the input required to generate the desired UMTS waveform corresponding to these settings.

Examine the `DPDCH` field in `config`. This field uses a nested structure to define this physical channel for the `'RMC12.2kbps'` reference channel.

```
config
```

```
config = struct with fields:
            TotFrames: 1
       ScramblingCode: 1
           FilterType: 'RRC'
    OversamplingRatio: 4
      NormalizedPower: 'Off'
                DPDCH: [1x1 struct]
                DPCCH: [1x1 struct]
                HSUPA: [1x1 struct]
              HSDPCCH: [1x1 struct]
```

```
config.DPDCH
```

```
ans = struct with fields:
             Enable: 'On'
         SlotFormat: 2
    CodeCombination: 64
              Power: 0
         DataSource: 'CCTrCH'
             CCTrCH: [1x1 struct]
```

```
config.DPDCH.CCTrCH
```

```
ans = struct with fields:
    Name: 'DCH'
    TrCH: [1x2 struct]
```

```
config.DPDCH.CCTrCH.TrCH(1)
```

```
ans = struct with fields:
                Name: 'DTCH'
                 CRC: '16'
          CodingType: 'conv3'
                 RMA: 256
                 TTI: 20
          DataSource: 'PN9-ITU'
    ActiveDynamicPart: 1
          DynamicPart: [1x1 struct]
```

```
config.DPDCH.CCTrCH.TrCH(1).DynamicPart
```

```
ans = struct with fields:
       BlockSize: 244
    BlockSetSize: 244
```

```
config.DPDCH.CCTrCH.TrCH(2)
```

```
ans = struct with fields:
                Name: 'DCCH'
                 CRC: '12'
          CodingType: 'conv3'
                 RMA: 256
                 TTI: 40
          DataSource: 'PN9-ITU'
    ActiveDynamicPart: 1
          DynamicPart: [1x1 struct]
```

```
config.DPDCH.CCTrCH.TrCH(2).DynamicPart
```

```
ans = struct with fields:
       BlockSize: 100
    BlockSetSize: 100
```

## Input Arguments

### `rc` — Reference channel configuration

character vector | string scalar

Reference channel configuration, specified as a character vector or string scalar. `rc` identifies which RMC or E-DPDCH FRC to configure. Values for `rc` when specified as a character vector include (for string scalar use double quotes):

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`rc`** | Required | Reference measurement channels:<br><br>`'RMC12.2kbps'`, `'RMC64kbps'`, `'RMC144kbps'`, `'RMC384kbps'` | Reference channel identifying the W-CDMA uplink RMC configuration set-up as defined in TS 25.101, Annex A.2 [1]. |
| | E-DPDCH Fixed Reference Channels:<br><br>`'FRC1'`, `'FRC2'`, `'FRC3'`, `'FRC4'`, `'FRC5'`, `'FRC6'`, `'FRC7'`, `'FRC8'` | Reference channel identifying the E-DPDCH FRC configuration as defined in TS 25.141, Annex A.10 [2]. |

**Note** Additional standards-based reference channels can be configured by executing `lteUplinkReferenceChannels` and then adjusting parameters to match configurations defined in TS 25.141 [2]. For example:

- To generate the HS-DPCCH RMC, use `'RMC12.2kbps'` and set `HSDPCCH.Enable = 'On'`.
- To generate the 12.2 kbps RMC defined in TS 25.141 [2], use `'RMC12.2kbps'`. Using this value the function initializes `config` to generate the TS 25.101 [1] 12.2 kbps RMC). After `config` is generated, adjust the DPDCH and DPCCH parameters to align with the settings in TS 25.141 [2].

Data Types: `char` | `string`

## Output Arguments

### `config` — Definition of the channels included for the waveform generator

structure

#### Top-Level Parameters and Substructures

Definition of the channels included for the waveform generator, returned as a structure.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`TotFrames`** | Required | Positive scalar integer | Total number of frames to be generated |
| **`ScramblingCode`** | Required | Scalar integer<br><br>$0 - \left(2^{24} - 1\right)$ | Scrambling code index used by UE |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **FilterType** | Required | 'RRC', or 'Off' | Enable the RRC filter |
| **OversamplingRatio** | Required | Positive scalar integer | Oversampling ratio |
| **NormalizedPower** | Required | Float (-inf to +inf) or 'Off' to disable power normalization | Overall waveform power in dBW relative to 1 ohm |
| **DPDCH** | Optional | Not present or single structure | See DPDCH Substructure. |
| **DPCCH** | Optional | Not present or single structure | See DPCCH Substructure. |
| **HSUPA** | Optional | Not present or single structure | See HSUPA Substructure. |
| **HSDPCCH** | Optional | Not present or single structure | See HSDPCCH Substructure. |

#### DPDCH Substructure

Include the DPDCH substructure in the config structure to add the dedicated physical data channel to the output structure. The DPDCH substructure contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **Enable** | Required | 'On', 'Off' | Enable or disable the channel by specifying Enable as 'On' or 'Off', respectively. |
| **SlotFormat** | Required | Nonnegative integer | DPDCH slot format number, specified as 0, 1, 2, 3, 4, 5, or 6. |
| **CodeCombination** | Required | Nonnegative integer, vector | Valid spreading factors, specified as a power of two or a vector of powers of two in the interval [4, 256]. |
| **Power** | Required | Float, –inf, inf | Channel power in dB, specified as a float, –inf, or inf. |
| **DataSource** | Required | Scalar, vector, character vector, cell array, string scalar | DPDCH data source, specified as a scalar, vector, cell array, or string scalar.<br><br>When defined as a cell array, use standard PN sequences and a seed value: {PN, seed}. PN options for character vector or cell array are 'PN9-ITU', 'PN9', 'PN11', 'PN15', and 'PN23'. If no seed is specified, the shift register is initialized with all ones.<br><br>To enable transport channel coding, specify DataSource as 'CCTrCH'. |
| **CCTrCH** | Optional | Structure | See CCTrCH Substructure. |

#### CCTrCH Substructure

The `CCTrCH` substructure is associated with the DPDCH physical channel definition substructures. The `CCTrCH` substructure contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`Name`** | Optional | Character vector, string scalar<br><br>Default depends on the physical channel specified | Name assigned to the CCTrCH, specified as a character vector or a string scalar. Functions do not use the `Name` field. Therefore, you can redefine the content with no consequence. |
| **`TrCH`** | Required | Structure, structure array | Transport channels in the CCTrCH, specified as a structure or a structure array. |
| **`TrCH.Name`** | Required | Character vector or string scalar<br><br>Default depends on the physical channel specified | Name assigned to the TrCH, specified as a character vector or a string scalar. Functions do not use the `Name` field. Therefore, you can redefine the content with no consequence. |
| **`TrCH.CRC`** | Required | Character vector, string scalar | Cyclic redundancy check (CRC) polynomial specifier, specified as one of these values: `'0'`, `'8'`, `'12'`, `'16'`, or `'24'`. |
| **`TrCH.TTI`** | Required | Positive integer | Transmission Time Interval (TTI) in ms, specified as 10, 20, 40, or 80. |
| **`TrCH.CodingType`** | Required | `'turbo'`, `'conv2'`, `'conv3'` | Channel coding type and rate, specified as `'turbo'`, `'conv2'`, or `'conv3'`. |
| **`TrCH.RMA`** | Required | Positive integer | Rate matching attribute value, specified as a positive integer in the interval [1, 256]. |
| **`TrCH.DataSource`** | Required | Binary scalar, binary vector, character vector, cell array, or string scalar | Transport channel data source, specified as a binary scalar, a vector with binary entries, a cell array, or a string scalar.<br><br>When defined as a cell array use standard PN sequences and a seed value: {PN, seed}. PN options for character vector or cell array are `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, and `'PN23'`. If no seed is specified, the shift register is initialized with all ones. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| | | Examples for setting the `DataSource` field include:<br>• ...`CCTrCH.TrCh(1).DataSource = [1 0 0 1]`, generates a sequence of transport blocks by looping the vector [1 0 0 1].<br>• ...`CCTrCH.TrCh(1).DataSource = 'PN9'`, generates a physical channel data block with random seed = 511.<br>• ...`CCTrCH.TrCh(1).DataSource = {'PN9',5}`, generates a physical channel data block with seed = 5. | |
| **`TrCH.ActiveDynamicPart`** | Required | Positive integer, vector | Active dynamic part, specified as a positive integer or a vector whose entries are positive integers in the interval [1, `length(DynamicPart)`]. |
| | | The `ActiveDynamicPart` field indicates the `DynamicPart` array index for the active transport format `(BlockSize, BlockSetSize)` from available combinations defined in `DynamicPart`. The selected transport format is used for data transmission in the current TTI. | |
| **`TrCH.DynamicPart`** | Required | Structure, structure array | Size of each transport block, specified as a structure or a structure array. |
| | | The `DynamicPart` fields, `BlockSize` and `BlockSetSize`, define the size of each transport block and the total bits per transport block set. As a pair `(BlockSize, BlockSetSize)` describe a transport format set. `DynamicPart` defines one or multiple transport format sets. | |
| **`TrCH.DynamicPart.BlockSize`** | Required | Positive integer | Transport block length, specified as a positive integer. |
| **`TrCH.DynamicPart.BlockSetSize`** | Required | Integer, multiple of `BlockSize` | Total number of bits in the transport block set. Implementation does not support multiple transport blocks, so by definition `BlockSize` is equal to `BlockSetSize`. |

#### DPCCH Substructure

Include the `DPCCH` substructure in the `config` structure to add the dedicated physical control channel to the output structure. The `DPCCH` substructure contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`Enable`** | Required | `'On'`, `'Off'` | Enable or disable the channel by setting `Enable` to `'On'` or `'Off'`, respectively. |
| **`SlotFormat`** | Required | Nonnegative integer | DPCCH slot format number, specified as 0, 1, 2, 3, 4, or 5. |
| **`Power`** | Required | Float, `-inf`, `inf` | DPCCH power in dB, specified as a float, `-inf`, or `inf`. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`TPCData`** | Required | Binary scalar, binary vector | Transmit power control data, specified as a binary scalar or a vector with binary entries. |
| **`TFCI`** | Required | Nonnegative integer | Transport format combination indicator, specified as a nonnegative integer in the interval [0, 1023]. |
| **`FBIData`** | Required | Binary scalar, binary vector | Feedback information data, specified as a binary scalar or a vector with binary entries. |

#### HSUPA Substructure

Include the `HSUPA` substructure in the `config` structure to add the high speed uplink packet access information and channels to the output structure. The `HSUPA` substructure contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`Enable`** | Required | `'On'`, `'Off'` | Enable or disable the channel by specifying `Enable` as `'On'` or `'Off'`, respectively. |
| **`CodeCombination`** | Required | Positive integer, vector | Valid one-code combinations for BPSK modulation are: 2, 4, 8, 16, 32, 64, 128, and 256.<br><br>Valid two-code combinations for BPSK modulation are [2 2] and [4 4].<br><br>The valid four-code combination for BPSK and 4PAM modulation is [2 2 4 4]. |
| **`EDPDCHPower`** | Required | Float, `–inf`, `inf` | E-DPDCH channel power in dB, specified as a float, `–inf`, or `inf`. |
| **`EDPCCHPower`** | Required | Float, `–inf`, `inf` | E-DPCCH channel power in dB, specified as a float, `–inf`, or `inf`. |
| **`RSNSequence`** | Required | Vector | Retransmission sequence numbers, specified as a vector whose entries are 0, 1, 2, or 3. The length of this vector determines the number of retransmissions. |
| **`ETFCI`** | Required | Nonnegative integer | E-TFCI value, specified as a nonnegative integer in the interval [0, 127]. |
| **`HappyBit`** | Required | 0 or 1 | Happy bit, specified as 0 or 1. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **DataSource** | Required | Scalar, vector, character vector, cell array, or string scalar | E-DPDCH data source, specified as a binary scalar, a vector with binary entries, a character vector, a cell array, or a string scalar.<br><br>When specifying `DataSrouce` as a cell array, use standard PN sequences and a seed value: {PN, seed}. PN options for character vector or cell array are `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, and `'PN23'`. If no seed is specified, the shift register is initialized with all ones.<br><br>To enable transport channel coding, specify `DataSource` as `'EDCH'`. |
| **EDCH** | Required | Structure | Enhanced dedicated channel (EDCH), specified as a structure. |
| **EDCH.BlockSize** | Required | Nonnegative integer | Transport block size, specified as a nonnegative integer. |
| **EDCH.TTI** | Required | 2, 10 | Transmission Time Interval (TTI), in ms, specified as 2 or 10. |
| **EDCH.Modulation** | Required | `'BPSK'`, `'4PAM'` | Modulation scheme, specified as `'BPSK'` or `'4PAM'`. |
| **EDCH.DataSource** | Required | Scalar, vector, character vector, cell array, or string scalar | E-DCH transport data source, specified as a binary scalar, a vector with binary entries, a character vector, a cell array, or a string scalar.<br><br>When specifying `DataSource` as a cell array, use standard PN sequences and a seed value: {PN, seed}. PN options for character vector or cell array are `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, and `'PN23'`. If no seed is specified, the shift register is initialized with all ones. |

#### HSDPCCH Substructure

Include `HSDPCCH` substructure in `config` structure to add the high speed dedicated physical control channel to the output structure. The `HSDPCCH` substructure contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **Enable** | Required | `'On'`, `'Off'` | Enable or disable the channel by specifying `Enable` as `'On'` or `'Off'`, respectively. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`Power`** | Required | Float, `-inf`, `inf` | HS-DPCCH channel power in dB, specified as a float, `-inf`, or `inf`. |
| **`CQI`** | Required | Nonnegative integer, vector | CQI values, specified as a nonnegative integer or a vector whose entries are nonnegative integers in the interval [0, 30]. |
| **`HARQACK`** | Required | Nonnegative integer, vector | HARQACK messages, specified as a nonnegative integer or a vector whose entries are nonnegative integers in the interval [0, 3]. |
| **`UEMIMO`** | Required | 0, 1 | Flag to indicate MIMO mode, specified as 0 or 1. |

# Version History

**Introduced in R2015a**

## References

[1] 3GPP TS 25.101. "Universal Mobile Telecommunications System (UMTS); User Equipment (UE) Radio Transmission and Reception (FDD)." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 25.141. "Universal Mobile Telecommunications System (UMTS); Base Station (BS) Conformance Testing (FDD)." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`umtsUplinkWaveformGenerator` | `umtsDownlinkReferenceChannels` | `umtsDownlinkWaveformGenerator`

**Topics**

"Uplink Reference Channel and Waveform Generation Parameter Structures"

# umtsUplinkWaveformGenerator

UMTS uplink waveform generation

## Syntax

```
waveform = umtsUplinkWaveformGenerator(config)
```

## Description

`waveform = umtsUplinkWaveformGenerator(config)` returns the Universal Mobile Telecommunications Service (UMTS) uplink waveform defined by the configuration structure, `config`. This function supports Wideband Code Division Multiple Access (W-CDMA), High-Speed Uplink Packet Access (HSUPA), and Evolved High-Speed Uplink Packet Access (HSPA+) waveform generation. The top-level parameters and lower-level substructures of `config` characterize the waveform and channel properties of the `umtsUplinkWaveformGenerator` function output. The `config` input is generated using the `umtsUplinkReferenceChannels` function; `config` includes top-level parameters and substructures to describe the different channels to include in the waveform. The top-level parameters of `config` are: `TotFrames`, `ScramblingCode`, `FilterType`, `OversamplingRatio`, and `NormalizedPower`. To enable the specific channels, you can add associated substructures: `DPDCH`, `DPCCH`, `HSUPA`, and `HSDPCCH`.

## Examples

### UMTS Uplink Waveform Generation

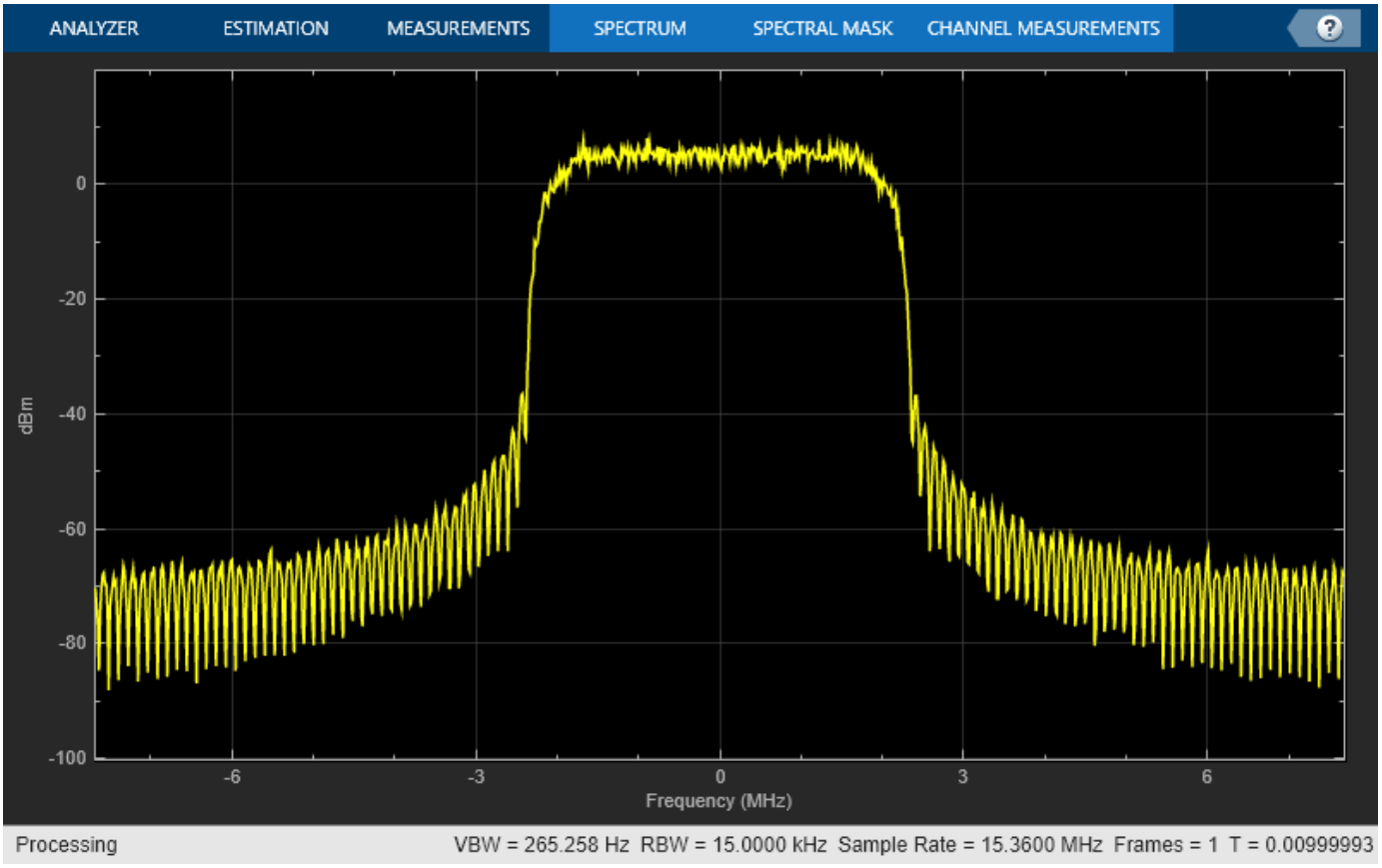
Initialize an `'RMC384kbps'` reference channel and generate the UMTS waveform that corresponds to these settings.

Generate the configuration structure, `config`.

```
rc = 'RMC384kbps';
config = umtsUplinkReferenceChannels(rc);
```

Generate the desired waveform using `config` as the input to the waveform generation function. Create a spectrum analyzer object sampling at `chiprate` x `OversamplingRatio`. Plot the waveform.

```
waveform = umtsUplinkWaveformGenerator(config);
saScope = spectrumAnalyzer(SampleRate=3.84e6*config.OversamplingRatio);
saScope(waveform);
```




## Input Arguments

### `config` — Definition of the channels included for the waveform generator

structure

#### Top-Level Parameters and Substructures

Definition of the channels included by the waveform generator, specified as a structure.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`TotFrames`** | Required | Positive integer | Total number of frames to be generated, specified as a positive integer. |
| **`ScramblingCode`** | Required | Nonnegative integer | Scrambling code index used by user equipment (UE), specified as a nonnegative integer in the interval [0, $2^{24}$–1]. |
| **`FilterType`** | Required | `'RRC'` (default), or `'Off'` | Enable or disable the RRC Filter by specifying `FilterType` as `'RRC'` or `'Off'`, respectively. |
| **`OversamplingRatio`** | Required | Positive integer | Oversampling ratio, specified as a positive integer. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **NormalizedPower** | Required | Float, –inf, inf, 'Off' | Overall waveform power in dBW relative to 1 ohm, specified as a float, –inf, inf, or 'Off'. Disable power normalization by specifying NormalizedPower as 'Off'. |
| **DPDCH** | Optional | Not present or structure | See DPDCH Substructure. |
| **DPCCH** | Optional | Not present or structure | See DPCCH Substructure. |
| **HSUPA** | Optional | Not present or structure | See HSUPA Substructure. |
| **HSDPCCH** | Optional | Not present or structure | See HSDPCCH Substructure. |

#### DPDCH Substructure

To add the dedicated physical data channel (DPDCH) to the output structure, include the DPDCH substructure in the config structure. The DPDCH substructure contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **Enable** | Required | 'On', 'Off' | Enable or disable the channel by specifying Enable as 'On' or 'Off', respectively. |
| **SlotFormat** | Required | Nonnegative integer | DPDCH slot format number, specified as 0, 1, 2, 3, 4, 5, or 6. |
| **CodeCombination** | Required | Nonnegative integer, vector | Valid spreading factors, specified as a power of two or a vector of powers of two in the interval [4, 256]. |
| **Power** | Required | Float, –inf, inf | Channel power in dB, specified as a float, –inf, or inf. |
| **DataSource** | Required | Scalar, vector, character vector, cell array, string scalar | DPDCH data source, specified as a scalar, vector, cell array, or string scalar.<br><br>When defined as a cell array, use standard PN sequences and a seed value: {PN, seed}. PN options for character vector or cell array are 'PN9-ITU', 'PN9', 'PN11', 'PN15', and 'PN23'. If no seed is specified, the shift register is initialized with all ones.<br><br>To enable transport channel coding, specify DataSource as 'CCTrCH'. |
| **CCTrCH** | Optional | Structure | See CCTrCH Substructure. |

#### CCTrCH Substructure

The CCTrCH substructure is associated with the DPDCH physical channel definition substructures. The CCTrCH substructure contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **Name** | Optional | Character vector, string scalar<br><br>Default depends on the physical channel specified | Name assigned to the CCTrCH, specified as a character vector or a string scalar. Functions do not use the `Name` field. Therefore, you can redefine the content with no consequence. |
| **TrCH** | Required | Structure, structure array | Transport channels in the CCTrCH, specified as a structure or a structure array. |
| **TrCH.Name** | Required | Character vector or string scalar<br><br>Default depends on the physical channel specified | Name assigned to the TrCH, specified as a character vector or a string scalar. Functions do not use the `Name` field. Therefore, you can redefine the content with no consequence. |
| **TrCH.CRC** | Required | Character vector, string scalar | Cyclic redundancy check (CRC) polynomial specifier, specified as one of these values: `'0'`, `'8'`, `'12'`, `'16'`, or `'24'`. |
| **TrCH.TTI** | Required | Positive integer | Transmission Time Interval (TTI) in ms, specified as 10, 20, 40, or 80. |
| **TrCH.CodingType** | Required | `'turbo'`, `'conv2'`, `'conv3'` | Channel coding type and rate, specified as `'turbo'`, `'conv2'`, or `'conv3'`. |
| **TrCH.RMA** | Required | Positive integer | Rate matching attribute value, specified as a positive integer in the interval [1, 256]. |
| **TrCH.DataSource** | Required | Binary scalar, binary vector, character vector, cell array, or string scalar | Transport channel data source, specified as a binary scalar, a vector with binary entries, a cell array, or a string scalar.<br><br>When defined as a cell array use standard PN sequences and a seed value: {PN, seed}. PN options for character vector or cell array are `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, and `'PN23'`. If no seed is specified, the shift register is initialized with all ones. |
| | | Examples for setting the `DataSource` field include:<br><br>• ...`CCTrCH.TrCh(1).DataSource = [1 0 0 1]`, generates a sequence of transport blocks by looping the vector [1 0 0 1].<br>• ...`CCTrCH.TrCh(1).DataSource = 'PN9'`, generates a physical channel data block with random seed = 511.<br>• ...`CCTrCH.TrCh(1).DataSource = {'PN9',5}`, generates a physical channel data block with seed = 5. | |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **TrCH.ActiveDynamicPart** | Required | Positive integer, vector | Active dynamic part, specified as a positive integer or a vector whose entries are positive integers in the interval [1, `length(DynamicPart)`]. |
| | | The `ActiveDynamicPart` field indicates the `DynamicPart` array index for the active transport format `(BlockSize, BlockSetSize)` from available combinations defined in `DynamicPart`. The selected transport format is used for data transmission in the current TTI. | |
| **TrCH.DynamicPart** | Required | Structure, structure array | Size of each transport block, specified as a structure or a structure array. |
| | | The `DynamicPart` fields, `BlockSize` and `BlockSetSize`, define the size of each transport block and the total bits per transport block set. As a pair `(BlockSize, BlockSetSize)` describe a transport format set. `DynamicPart` defines one or multiple transport format sets. | |
| **TrCH.DynamicPart.BlockSize** | Required | Positive integer | Transport block length, specified as a positive integer. |
| **TrCH.DynamicPart.BlockSetSize** | Required | Integer, multiple of `BlockSize` | Total number of bits in the transport block set. Implementation does not support multiple transport blocks, so by definition `BlockSize` is equal to `BlockSetSize`. |

#### DPCCH Substructure

To add the dedicated physical control channel (DPCCH) to the output structure, include the `DPCCH` substructure in the `config` structure. The `DPCCH` substructure contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **Enable** | Required | `'On'`, `'Off'` | Enable or disable the channel by setting `Enable` to `'On'` or `'Off'`, respectively. |
| **SlotFormat** | Required | Nonnegative integer | DPCCH slot format number, specified as 0, 1, 2, 3, 4, or 5. |
| **Power** | Required | Float, `-inf`, `inf` | DPCCH power in dB, specified as a float, `-inf`, or `inf`. |
| **TPCData** | Required | Binary scalar, binary vector | Transmit power control data, specified as a binary scalar or a vector with binary entries. |
| **TFCI** | Required | Nonnegative integer | Transport format combination indicator, specified as a nonnegative integer in the interval [0, 1023]. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **FBIData** | Required | Binary scalar, binary vector | Feedback information data, specified as a binary scalar or a vector with binary entries. |

#### HSUPA Substructure

To add the high-speed uplink packet access (HSUPA) information and channels to the output structure, include the `HSUPA` substructure in the `config` structure. The `HSUPA` substructure contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **Enable** | Required | `'On'`, `'Off'` | Enable or disable the channel by specifying `Enable` as `'On'` or `'Off'`, respectively. |
| **CodeCombination** | Required | Positive integer, vector | Valid one-code combinations for BPSK modulation are: 2, 4, 8, 16, 32, 64, 128, and 256.<br><br>Valid two-code combinations for BPSK modulation are [2 2] and [4 4].<br><br>The valid four-code combination for BPSK and 4PAM modulation is [2 2 4 4]. |
| **EDPDCHPower** | Required | Float, `–inf`, `inf` | E-DPDCH channel power in dB, specified as a float, `–inf`, or `inf`. |
| **EDPCCHPower** | Required | Float, `–inf`, `inf` | E-DPCCH channel power in dB, specified as a float, `–inf`, or `inf`. |
| **RSNSequence** | Required | Vector | Retransmission sequence numbers, specified as a vector whose entries are 0, 1, 2, or 3. The length of this vector determines the number of retransmissions. |
| **ETFCI** | Required | Nonnegative integer | E-TFCI value, specified as a nonnegative integer in the interval [0, 127]. |
| **HappyBit** | Required | 0 or 1 | Happy bit, specified as 0 or 1. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **DataSource** | Required | Scalar, vector, character vector, cell array, or string scalar | E-DPDCH data source, specified as a binary scalar, a vector with binary entries, a character vector, a cell array, or a string scalar.<br><br>When specifying `DataSrouce` as a cell array, use standard PN sequences and a seed value: {PN, seed}. PN options for character vector or cell array are `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, and `'PN23'`. If no seed is specified, the shift register is initialized with all ones.<br><br>To enable transport channel coding, specify `DataSource` as `'EDCH'`. |
| **EDCH** | Required | Structure | Enhanced dedicated channel (EDCH), specified as a structure. |
| **EDCH.BlockSize** | Required | Nonnegative integer | Transport block size, specified as a nonnegative integer. |
| **EDCH.TTI** | Required | 2, 10 | Transmission Time Interval (TTI), in ms, specified as 2 or 10. |
| **EDCH.Modulation** | Required | `'BPSK'`, `'4PAM'` | Modulation scheme, specified as `'BPSK'` or `'4PAM'`. |
| **EDCH.DataSource** | Required | Scalar, vector, character vector, cell array, or string scalar | E-DCH transport data source, specified as a binary scalar, a vector with binary entries, a character vector, a cell array, or a string scalar.<br><br>When specifying `DataSource` as a cell array, use standard PN sequences and a seed value: {PN, seed}. PN options for character vector or cell array are `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, and `'PN23'`. If no seed is specified, the shift register is initialized with all ones. |

#### HSDPCCH Substructure

Include `HSDPCCH` substructure in `config` structure To add the high speed dedicated physical control channel (HS-DPCCH) to the output structure. The `HSDPCCH` substructure contains the following fields.

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **Enable** | Required | `'On'`, `'Off'` | Enable or disable the channel by specifying `Enable` as `'On'` or `'Off'`, respectively. |

| Parameter Field | Required or Optional | Values | Description |
|---|---|---|---|
| **`Power`** | Required | Float, `-inf`, `inf` | HS-DPCCH channel power in dB, specified as a float, `-inf`, or `inf`. |
| **`CQI`** | Required | Nonnegative integer, vector | CQI values, specified as a nonnegative integer or a vector whose entries are nonnegative integers in the interval [0, 30]. |
| **`HARQACK`** | Required | Nonnegative integer, vector | HARQACK messages, specified as a nonnegative integer or a vector whose entries are nonnegative integers in the interval [0, 3]. |
| **`UEMIMO`** | Required | 0, 1 | Flag to indicate MIMO mode, specified as 0 or 1. |

## Output Arguments

**`waveform` — Modulated baseband waveform containing the UMTS physical channels**
complex vector array

Modulated baseband waveform containing the UMTS physical channels, returned as a complex vector array, sampled at (3.84 × `config.OversamplingRatio`) MHz.

Data Types: `double`

# Version History

**Introduced in R2015a**

## References

[1] 3GPP TS 25.101. “Universal Mobile Telecommunications System (UMTS); User Equipment (UE) Radio Transmission and Reception (FDD).” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

[2] 3GPP TS 25.141. “Universal Mobile Telecommunications System (UMTS); Base Station (BS) Conformance Testing (FDD).” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

`umtsUplinkReferenceChannels` | `umtsDownlinkReferenceChannels` | `umtsDownlinkWaveformGenerator`

**Topics**
“Uplink Reference Channel and Waveform Generation Parameter Structures”

# getPathFilters

Get path filter impulse response for 3-D MIMO fading channel

## Syntax

```
pathFilters = getPathFilters(lte3d)
```

## Description

`pathFilters = getPathFilters(lte3d)` returns path filter impulse responses for the 3-D multi-input/multi-output (MIMO) fading channel object specified by `lte3d`. You can use this information, along with the `pathGains` output argument returned by the channel object, to reconstruct a perfect channel estimate.

## Examples

### Reconstruct Channel Impulse Response Using 3-D Channel Path Filters

Reconstruct the channel impulse response and perform timing offset estimation using path filters of an `lte3DChannel` System object.

Configure a channel for delay profile CDL-B from TR 38.901 Section 7.7.1, with 1000 ns delay spread, 2 transmit antennas, and 1 receive antenna.

```
lte3d = lte3DChannel.makeCDL('CDL-B',1000e-9);
lte3d.Seed = 11;
lte3d.TransmitAntennaArray.Size = [1 1 2];
lte3d.ReceiveAntennaArray.Size = [1 1 1];
```

Create an LTE waveform for reference measurement channel (RMC) R.10 (10MHz, QPSK, R=1/3, 2 CRS ports).

```
rmc = lteRMCDL('R.10');
rmc.TotSubframes = 1;
data = [1; 0; 0; 1];
[txWaveform,~,txInfo] = lteRMCDLTool(rmc,data);
lte3d.SampleRate = txInfo.SamplingRate;
```

Pass the waveform through the channel.

```
[rxWaveform,pathGains] = lte3d(txWaveform);
```

Perform timing offset estimation using cell-specific reference signals.

```
corrcfg.CellRS = 'On';
offset = lteDLFrameOffset(rmc,rxWaveform,corrcfg)
```

```
offset = 15
```

Obtain the path filters used in channel filtering.

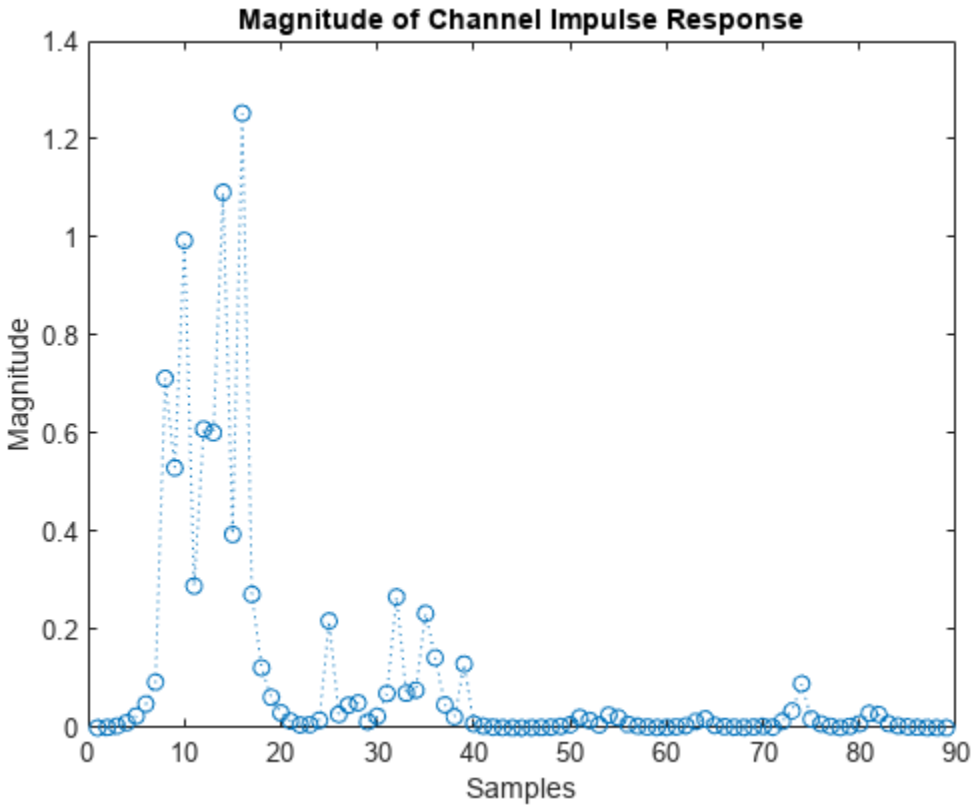
```
pathFilters = getPathFilters(lte3d);
```

Reconstruct the channel impulse response using the path filters and path gains. Take the average of path gains across all time samples (first dimension). Construct the impulse response, h, for each transmit and receive antenna. Sum the responses for each transmit antenna.

```
[~,Np,P,R] = size(pathGains);
Nh = size(pathFilters,1);
h = zeros([Nh P R]);
pathGains = permute(mean(pathGains,1),[2 3 4 1]);
for np = 1:Np
    h = h + pathFilters(:,np) .* pathGains(np,:,:);
end
h = permute(sum(h,2),[1 3 2]);
mag = abs(h);
```

Plot the magnitude of the channel impulse response.

```
plot(mag,'o:')
title('Magnitude of Channel Impulse Response')
xlabel('Samples')
ylabel('Magnitude')
```




Estimate the timing offset by finding the peak of the impulse response magnitude.

```
offset_ref = find(mag==max(mag)) - 1
```

```
offset_ref = 15
```

## Input Arguments

### `lte3d` — MIMO fading channel

`lte3DChannel` System object

MIMO fading channel, specified as an `lte3DChannel` System object. This object implements the TR 36.873 link-level MIMO fading channel.

## Output Arguments

### `pathFilters` — Path filter impulse response

$N_h$-by-$N_p$ real matrix

Path filter impulse response, returned as an $N_h$-by-$N_p$ real matrix, where:

- $N_h$ is the number of impulse response samples.
- $N_p$ is the number of paths.

Each column of the matrix contains the filter impulse response for each path of the delay profile.

Data Types: `double`

# Version History

**Introduced in R2018a**

## See Also

`lte3DChannel`

# info

Get characteristic information about 3-D MIMO fading channel

## Syntax

```
channelInfo = info(lte3d)
```

## Description

`channelInfo = info(lte3d)` returns characteristic information about the 3-D multi-input/multi-output (MIMO) fading channel object specified by `lte3d`.

## Examples

### Get Characteristic Information About MIMO Fading Channel

Create an `lte3DChannel` System object.

```
lte3d = lte3DChannel('PathDelays',[0 500e-9], ...
    'AveragePathGains',[-13.4 3.0], ...
    'AnglesAoD',[-178.1 -4.2], ...
    'AnglesAoA',[51.3 -152.7], ...
    'AnglesZoD',[50.2 93.2], ...
    'AnglesZoA',[125.4 91.3], ...
    'NumStrongestClusters',1);
```

To get characteristic information about the channel, call the `info` function on the object.

```
channelInfo = info(lte3d)
```

```
channelInfo = struct with fields:
    KFactorFirstCluster: -Inf
           ClusterTypes: {'SubclusteredNLOS'  'SubclusteredNLOS'  'SubclusteredNLOS'  'NLOS'}
             PathDelays: [5.0000e-07 5.0500e-07 5.1000e-07 0]
       AveragePathGains: [-0.0103 -2.2288 -3.9897 -13.4000]
              AnglesAoD: [-4.2000 -4.2000 -4.2000 -178.1000]
              AnglesAoA: [-152.7000 -152.7000 -152.7000 51.3000]
              AnglesZoD: [93.2000 93.2000 93.2000 50.2000]
              AnglesZoA: [91.3000 91.3000 91.3000 125.4000]
    NumTransmitAntennas: 8
        NumInputSignals: 8
     NumReceiveAntennas: 2
       NumOutputSignals: 2
     ChannelFilterDelay: 7
    MaximumChannelDelay: 23
```

## Input Arguments

### `lte3d` — MIMO fading channel

`lte3DChannel` System object

MIMO fading channel, specified as an `lte3DChannel` System object. This object implements the TR 36.873 link-level MIMO fading channel.

## Output Arguments

### `channelInfo` — Characteristic information about MIMO fading channel

structure

Characteristic information about MIMO fading channel, returned as a structure containing the following fields:

| Parameter Field | Value | Description |
|---|---|---|
| **`PathDelays`** | Numeric row vector | Delays of discrete channel paths for each cluster, returned in seconds. These values include the effects of `DelaySpread` scaling, and `KFactor` scaling (when enabled). |
| **`ClusterTypes`** | Cell array of character vectors | Type of each cluster in the delay profile, returned as a cell array of character vectors. Cluster types can be `'LOS'`, `'SubclusteredNLOS'`, or `'NLOS'`. The PathDelays, AveragePathGains, AnglesAoA, AnglesAoD, AnglesZoA, and AnglesZoD properties define the delay profile. |
| **`AveragePathGains`** | Numeric row vector | Average path gains of the discrete path or clusters in dB. These values include the effect of K-factor scaling if enabled. For more information, see the `KFactor` property. |
| **`AnglesAoD`** | Numeric row vector | Azimuth of departure angles of the clusters in degrees. |
| **`AnglesAoA`** | Numeric row vector | Azimuth of arrival angles of the clusters in degrees. |
| **`AnglesZoD`** | Numeric row vector | Zenith of departure angles of the clusters in degrees. |
| **`AnglesZoA`** | Numeric row vector | Zenith of arrival angles of the clusters in degrees. |

| Parameter Field | Value | Description |
|---|---|---|
| **KFactorFirstCluster** | Numeric scalar | K-factor of first cluster of delay profile in dB. If the first cluster of the delay profile follows a Laplacian instead of a Rician distribution, `KFactorFirstCluster` is `-Inf`. |
| **NumTransmitAntennas** | Numeric scalar | Number of transmit antennas. |
| **NumInputSignals** | Numeric scalar | Number of input signals. |
| **NumReceiveAntennas** | Numeric scalar | Number of receive antennas. |
| **NumOutputSignals** | Numeric scalar | Number of output signals. |
| **ChannelFilterDelay** | Numeric scalar | Channel filter delay in samples. |

**Note**

- The step of splitting the strongest clusters into subclusters, described in TR 36.873 [1], Section 7.3, requires sorting of the clusters by their average power. Therefore if the `NumStrongestClusters` property is nonzero, the fields of the information structure are sorted by average power. That is, the `AveragePathGains`, `ClusterTypes`, `PathDelays`, `AnglesAoD`, `AnglesAoA`, `AnglesZoD`, and `AnglesZoA` fields are presented in descending order of the average gain.
- If the `HasLOSCluster` property is set to `true`, the NLOS (Laplacian) part of that cluster, and the LOS cluster, are not necessarily next to each other. However, the `KFactorFirstCluster` field still indicates the appropriate K-factor scaling.

# Version History

**Introduced in R2018a**

## References

[1] 3GPP TR 36.873. “Study on 3D channel model for LTE.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. URL: https://www.3gpp.org.

## See Also

`lte3DChannel`

# Apps

# LTE Throughput Analyzer

Generate throughput curves for physical downlink shared channel (PDSCH) conformance test analysis

## Description

The **LTE Throughput Analyzer** app performs PDSCH demodulation performance testing. TS 36.101 [1], Annex A.3 specifies RMCs for UE performance testing.

The app also performs analysis and testing for custom user-defined measurement channels settings. For an example, see “LTE Throughput Analyzer User-Defined Testing” on page 3-8. This approach can also be used for simulating transmission modes 7–10, specifically, when the transmission scheme (`TxScheme`) is `'Port5'`, `'Port7-8'`, `'Port8'`, or `'Port7-14'` where DM-RS based channel estimation is required for PDSCH demodulation. In this case, the precoding matrix `W` is randomly defined per subframe according to TS 36.101 [1], Table 8.3.1-1 for FDD and Table 8.3.2-1 for TDD.

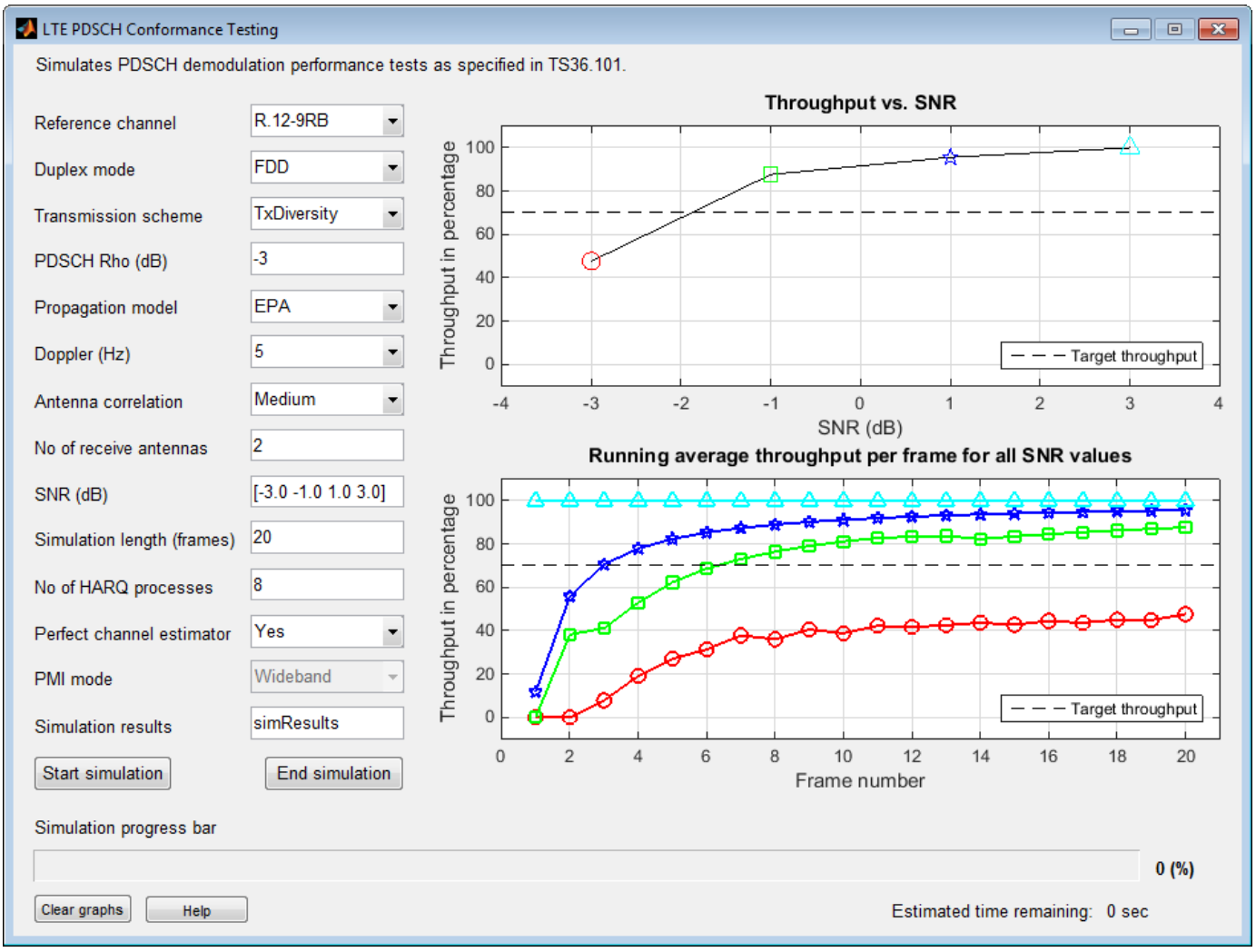
### Dialog Box Inputs and Outputs

In the **LTE PDSCH Conformance Testing** user interface, you can set these parameters:

| Parameter (Equivalent Field) | Values | Description |
|---|---|---|
| **Reference channel** (RC) | `'R0'` (default), `'R1'`, `'R2'`, `'R3'`, `'R4'`, `'R5'`, `'R6'`, `'R7'`, `'R8'`, `'R9'`, `'R10'`, `'R11'`, `'R12'`, `'R13'`, `'R14'`, `'R6-27RB'`, `'R12-9RB'`, `'R11-45RB'`, `User defined` | Reference measurement channel (RMC) number or type, as specified in TS 36.101, Annex A.3.<br>• To facilitate the transmission of system information blocks (SIB), normally no user data is scheduled on subframe 5. However, `'R.31-3A'` and `'R.31-4'` are sustained data rate RMCs and have user data in subframe 5.<br>• `'R.6-27RB'`, `'R.12-9RB'`, and `'R.11-45RB'` are custom RMCs configured for non-standard bandwidths that maintain the same code rate as the standardized versions defined in TS 36.101, Annex A.3.<br>To define your own reference channel, select `User defined`. The **User-defined configuration** dialog box opens. For **Configuration structure variable name**, type the name of an RC parameter structure variable in the MATLAB workspace.<br>The tool expects this variable to be present in the MATLAB base workspace. Create the basic configuration structure with the function `lteRMCDL` by choosing a closely matched RMC and modifying to meet your requirements. Use this approach to simulate transmission modes 7–10. Specifically, when `TxScheme = 'Port5'`, `'Port7-8'`, `'Port8'`, or `'Port7-14'`, where DM-RS based channel estimation is required for PDSCH demodulation. In this case, the precoding matrix, `W`, is randomly defined per subframe according to TS 36.101, Table 8.3.1-1, or Table 8.3.2-1. |
| **Duplex mode** (`DuplexMode`) | `'FDD'` (default), `'TDD'` | Duplexing mode, specified as either:<br>• `'FDD'` for Frequency Division Duplex<br>• `'TDD'` for Time Division Duplex |

<table>
<tr><th>Parameter (Equivalent Field)</th><th>Values</th><th>Description</th></tr>
<tr><td>Transmission scheme (TxScheme)</td><td>'Port0', 'TxDiversity', 'CDD', 'SpatialMux', 'MultiUser', 'Port5', 'Port7-8', 'Port8', 'Port7-14'.</td><td>PDSCH transmission scheme, specified as one of the following options.
<table>
<tr><th>Transmission scheme</th><th>Description</th></tr>
<tr><td>'Port0'</td><td>Single antenna port, port 0</td></tr>
<tr><td>'TxDiversity'</td><td>Transmit diversity</td></tr>
<tr><td>'CDD'</td><td>Large delay cyclic delay diversity scheme</td></tr>
<tr><td>'SpatialMux'</td><td>Closed loop spatial multiplexing</td></tr>
<tr><td>'MultiUser'</td><td>Multi-user MIMO</td></tr>
<tr><td>'Port5'</td><td>Single-antenna port, port 5</td></tr>
<tr><td>'Port7-8'</td><td>Single-antenna port, port 7, when NLayers = 1. Dual layer transmission, ports 7 and 8, when NLayers = 2.</td></tr>
<tr><td>'Port8'</td><td>Single-antenna port, port 8</td></tr>
<tr><td>'Port7-14'</td><td>Up to eight layer transmission, ports 7–14</td></tr>
</table></td></tr>
<tr><td>PDSCH Rho (dB) (Rho)</td><td>0 (default), numeric scalar</td><td>PDSCH resource element power allocation, in dB</td></tr>
<tr><td>Propagation Model (DelayProfile)</td><td>'Off', 'EPA' (default), 'EVA', 'ETU', 'HST'</td><td>Delay profile model. For more information, see “Propagation Channel Models”.</td></tr>
<tr><td>Doppler (Hz) (DopplerFreq)</td><td>'5', '70', '300', '750'</td><td>Maximum Doppler frequency, in Hz.</td></tr>
<tr><td>Antenna Correlation (MIMOCorrelation)</td><td>'Low', 'Medium', 'High'</td><td>Correlation between UE and eNodeB antennas</td></tr>
<tr><td>No of receive antennas (NRxAnts)</td><td>Nonnegative scalar integer</td><td>Number of receive antennas</td></tr>
<tr><td>SNR (dB)</td><td>Numeric vector</td><td>SNR values, in dB</td></tr>
<tr><td>Simulation length (frames)</td><td>Positive scalar integer</td><td>Simulation length, in frames</td></tr>
<tr><td>Number of HARQ processes (NHARQProcesses)</td><td>1, 2, 3, 4, 5, 6, 7, or 8</td><td>Number of HARQ processes per component carrier</td></tr>
<tr><td>Perfect channel estimator</td><td>'Yes', 'No'</td><td>Channel estimator provides a perfect channel estimate when setting is 'Yes'. For more information, see lteDLPerfectChannelEstimate.</td></tr>
</table>

| Parameter (Equivalent Field) | Values | Description |
|---|---|---|
| **PMI mode** (`PMIMode`) | `'Wideband'` (default), `'Subband'` | PMI reporting mode. `PMIMode='Wideband'` corresponds to PUSCH reporting Mode 1-2 or PUCCH reporting Mode 1-1 (PUCCH Report Type 2) and `PMIMode='Subband'` corresponds to PUSCH reporting Mode 3-1. |
| **Simulation results** | Variable name beginning with an alphabetical character and containing alphanumeric characters. | Simulation results output variable name. When you click **Generate waveform**, a new variable with this name is created in the MATLAB workspace. |




## Open the LTE Throughput Analyzer App

- MATLAB Toolstrip: On the **Apps** tab, under **Signal Processing and Communications**, select the **LTE Throughput Analyzer** app icon.

- MATLAB command prompt: Enter `lteThroughputAnalyzer`.

## Examples

### Perform 4-by-2 Transmit Diversity Conformance Test

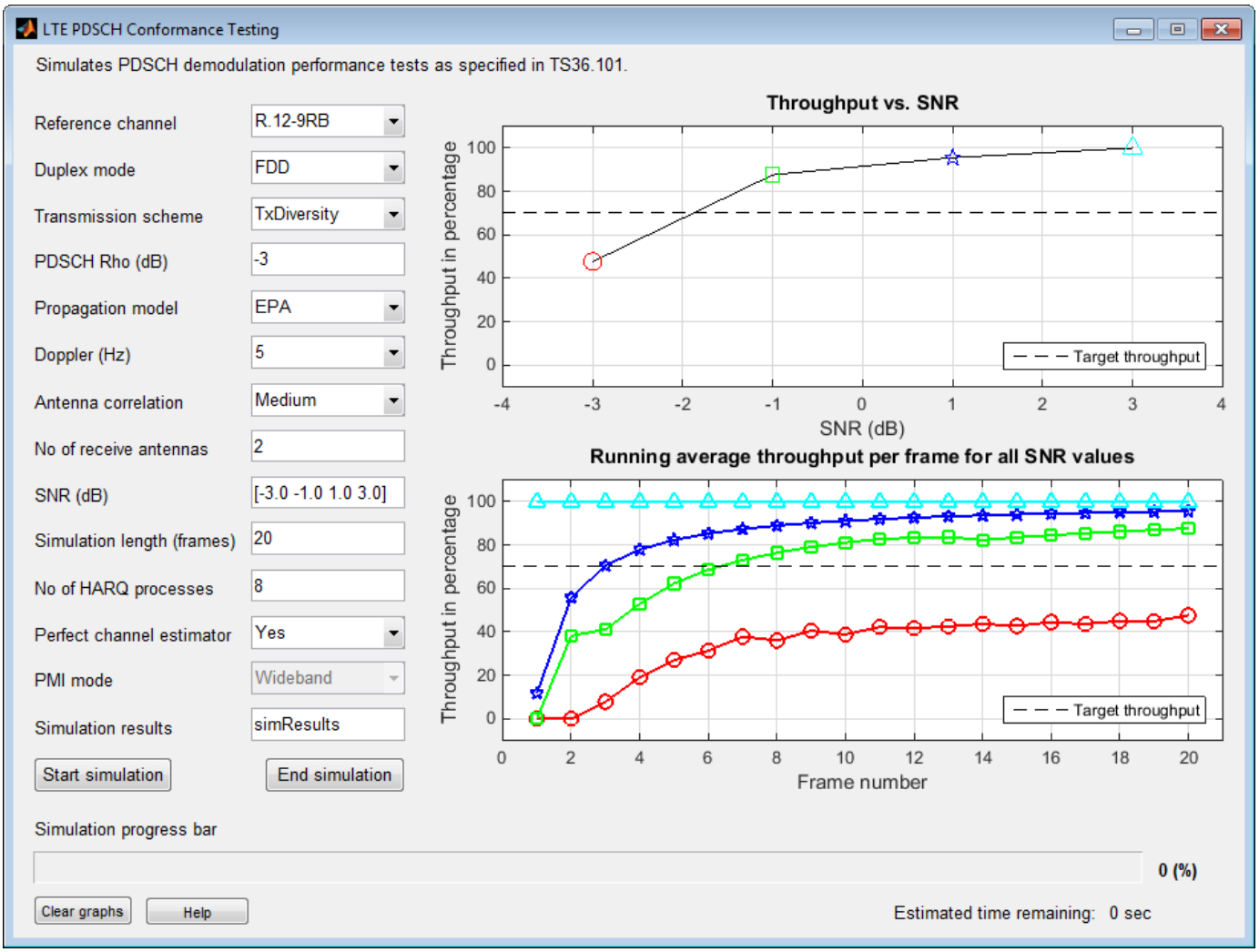
Use the **LTE Throughput Analyzer** app to run a conformance test for a single codeword RMC R.12-9RB for the transmit diversity transmission scheme with EPA-5 fading.

Open the **LTE Throughput Analyzer** app.

Adjust default runtime parameter settings:

- Set **Reference channel** to `R.12-9RB`.
- For **SNR (dB)**, enter `[-3.0 -1.0 1.0 3.0]`.
- For **Simulation length (frames)**, enter `20`.

Select **Start simulation**. The app provides the **Estimated time remaining**. When the simulation finishes, the dialog box shows performance curves.




The simulation result for a 20-frame run is displayed in the MATLAB Command Window.

```
Result for -3 dB SNR
Throughput: 47.65%

Result for -1 dB SNR
Throughput: 87.65%

Result for 1 dB SNR
Throughput: 95.59%

Result for 3 dB SNR
Throughput: 100.00%
```

In addition, the `simResults` variable now appears in the MATLAB workspace. View its contents.

```
simResults
```

```
simResults =

1x4 struct array with fields:
```

```
throughput
tpPerFrame
rawBER
```

### LTE Throughput Analyzer User-Defined Testing

Open the LTE throughput analyzer app and run a user-defined measurement channel. Define a custom measurement channel. You can select any RMC and change any settings, though care must be taken not to define an invalid configuration.

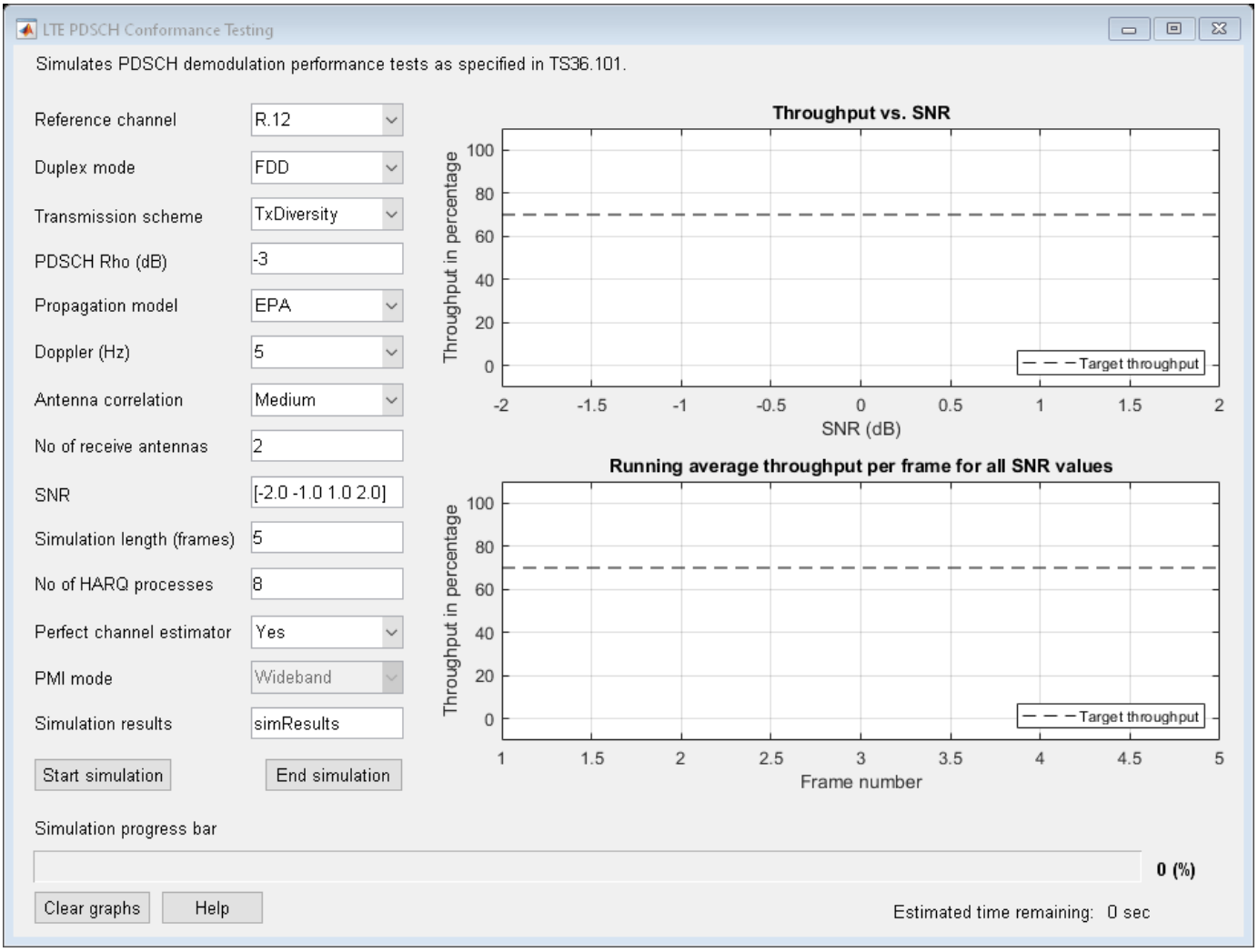
For this example, start with an R.3 RMC, and adjust the number of resource blocks from 50 to 30.

```
cmc = lteRMCDL('R.3');
cmc.NDLRB = 30
```

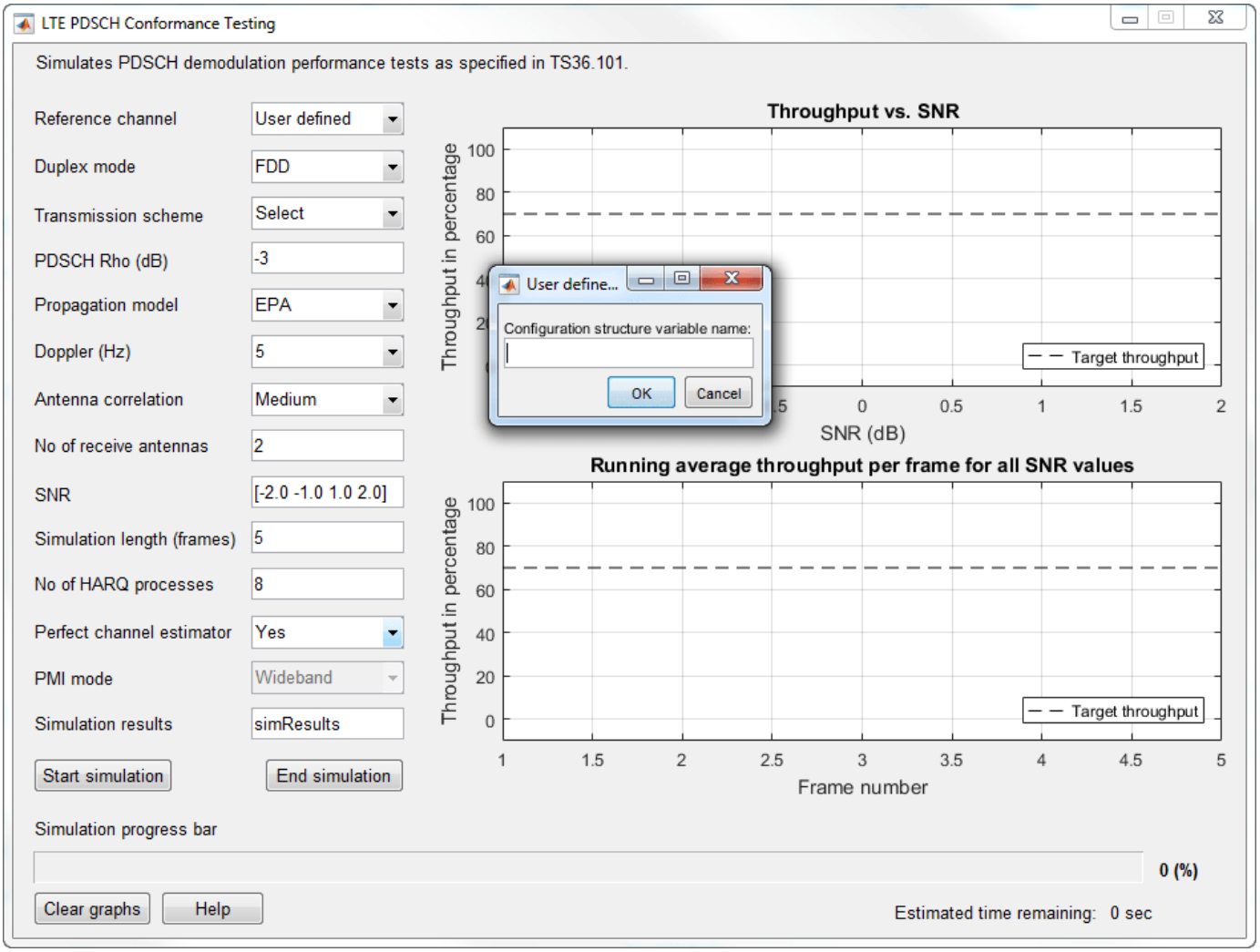
```
cmc = struct with fields:
                RC: 'R.3'
             NDLRB: 30
           CellRefP: 1
            NCellID: 0
       CyclicPrefix: 'Normal'
                CFI: 2
        PCFICHPower: 0
                 Ng: 'Sixth'
      PHICHDuration: 'Normal'
              HISet: [112x3 double]
         PHICHPower: 0
             NFrame: 0
          NSubframe: 0
       TotSubframes: 10
          Windowing: 0
         DuplexMode: 'FDD'
               PDSCH: [1x1 struct]
    OCNGPDCCHEnable: 'Off'
     OCNGPDCCHPower: 0
    OCNGPDSCHEnable: 'Off'
     OCNGPDSCHPower: 0
          OCNGPDSCH: [1x1 struct]
               Nfft: []
```

Open the LTE throughput analyzer app.

```
lteThroughputAnalyzer
```




Choose the `Reference channel` dropdown menu and select `User defined`.




At the prompt, enter the custom measurement channel configuration structure name, `cmc`.

To tun this user-defined configuration, click `Start simulation`.

- “Analyze Throughput for PDSCH Demodulation Performance Test”

# Version History

**Introduced in R2014a**

## References

[1] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

## See Also

**Apps**
**LTE Waveform Generator**

**Functions**
`lteRMCDL`

**Topics**
“Analyze Throughput for PDSCH Demodulation Performance Test”

# LTE Waveform Generator

Create, impair, visualize, and export LTE waveforms

## Description

The **LTE Waveform Generator** app enables you to create, impair, visualize, and export LTE waveforms.

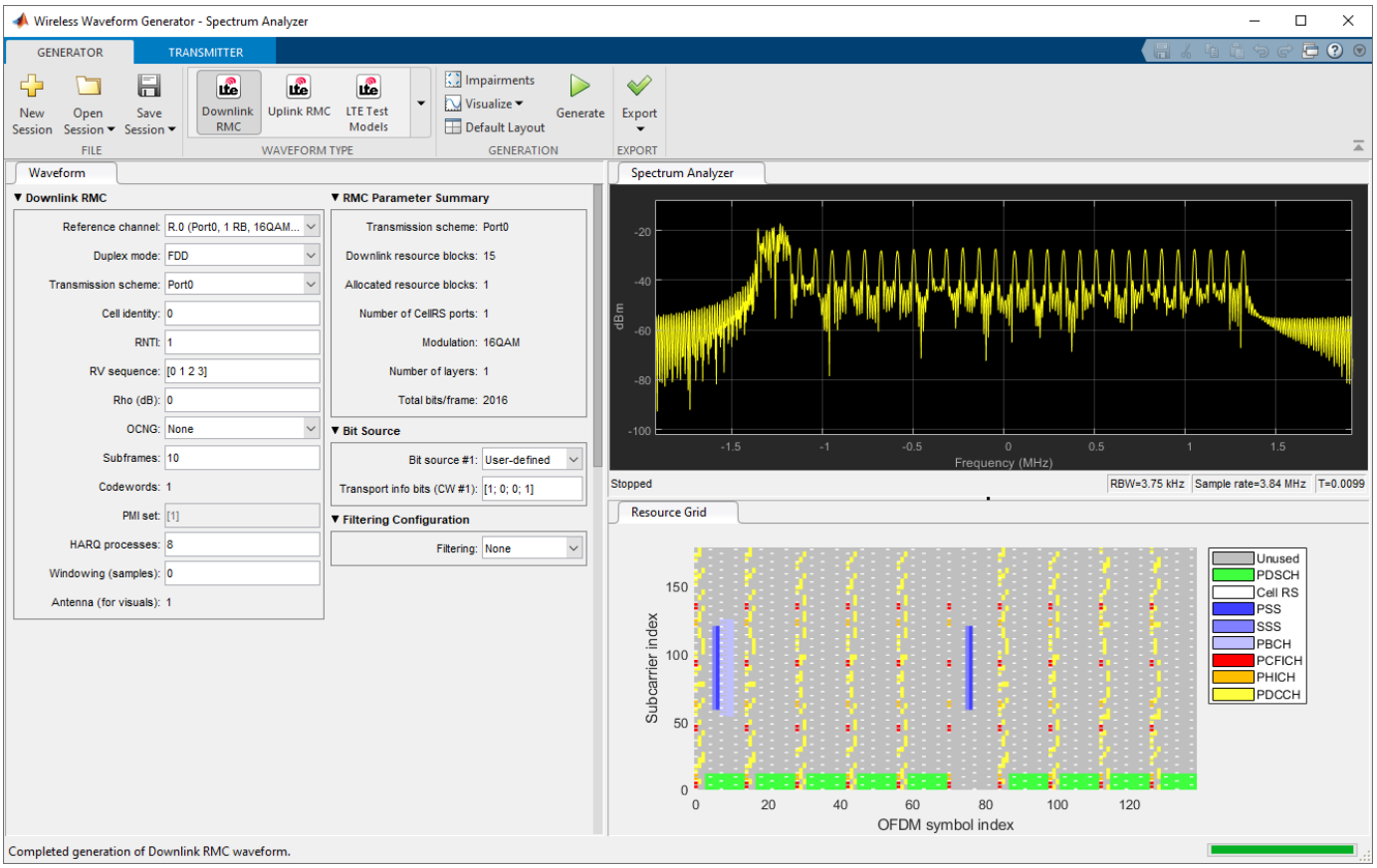
The app provides these capabilities by using the **Wireless Waveform Generator** app configured for LTE waveform generation. Using the app, you can:

- Generate LTE test model (E-TM) waveforms, as defined in section 6 of TS 36.141 [1].
- Generate LTE downlink reference measurement channel (RMC) waveforms for user equipment (UE) performance testing, as defined in Annex A.3 of TS 36.101 [2].
- Generate LTE uplink RMC waveforms for base station (BS) performance testing, as defined in Annex A of TS 36.104 [3].
- Export the LTE waveform to your workspace or to a `.mat` or a `.bb` file.
- Export LTE waveform generation parameters to a runnable MATLAB script or a Simulink block.
  - Use the exported script to generate your waveform without the app from the command line.
  - Use the exported block as a waveform source in a Simulink model. For more information, see Waveform From Wireless Waveform Generator App.
- Visualize the LTE waveform in constellation diagram, spectrum analyzer, OFDM grid, 3D spectrogram, time scope, and complementary cumulative distribution function (CCDF) plots.
- Distort the LTE waveform by adding RF impairments, such as AWGN, phase offset, frequency offset, DC offset, IQ imbalance, and memoryless cubic nonlinearity.
- Generate an LTE waveform that you can transmit using a connected radio or lab test instrument.
  - To transmit a waveform by using an SDR, connect one of the supported SDRs (ADALM-Pluto, USRP™, USRP embedded series, and Xilinx® Zynq-based radios) to your computer and have the associated add-on installed. For more information, see “Transmit Using SDR”.
  - To transmit a waveform by using a lab test instrument, the connected lab test instrument must:
    - Support the TCP/IP interface
    - Use one of these drivers — AgRfSigGen, RsRfSigGen, AgRfSigGen_SCPI, or RsRfSigGen_SCPI
    - Be supported by the `rfsiggen` function

    For more information, see “Quick-Control RF Signal Generator Requirements” (Instrument Control Toolbox). This feature requires “Instrument Control Toolbox”.
  - To transmit your waveforms over the air at full radio device rates, use the Wireless Testbench™ software and connect a supported radio to your computer. For a list of radios that support full device rates, see “Supported Radio Devices” (Wireless Testbench). This feature requires “Wireless Testbench”. For an example, see “Transmit App-Generated Wireless Waveform Using Radio Transmitters” on page 3-15.

To create, impair, visualize, and export waveforms other than LTE waveforms, you must reconfigure the app. For a full list of features, see the **Wireless Waveform Generator** app.

For more information, see “Create Waveforms Using Wireless Waveform Generator App”.




# Open the LTE Waveform Generator App

MATLAB Toolstrip: On the **Apps** tab, under **Signal Processing and Communications**, click the app icon. 


MATLAB Command Prompt: Enter `wirelessWaveformGenerator`. This command opens the **Wireless Waveform Generator** app. To configure the app for LTE waveform generation, in the **Waveform Type** section, select one of the options under **LTE (4G)**.

## Examples

### App-Based LTE Waveform Generation

This example shows how to generate LTE test model (E-TM) and reference measurement channel (RMC) waveforms by using the **LTE Waveform Generator** app.

### Open LTE Waveform Generator App

On the **Apps** tab of the MATLAB® toolstrip, select the **LTE Waveform Generator** app icon under **Signal Processing and Communications**. This section opens the **Wireless Waveform Generator** app configured for LTE waveform generation.

### Select LTE Waveform

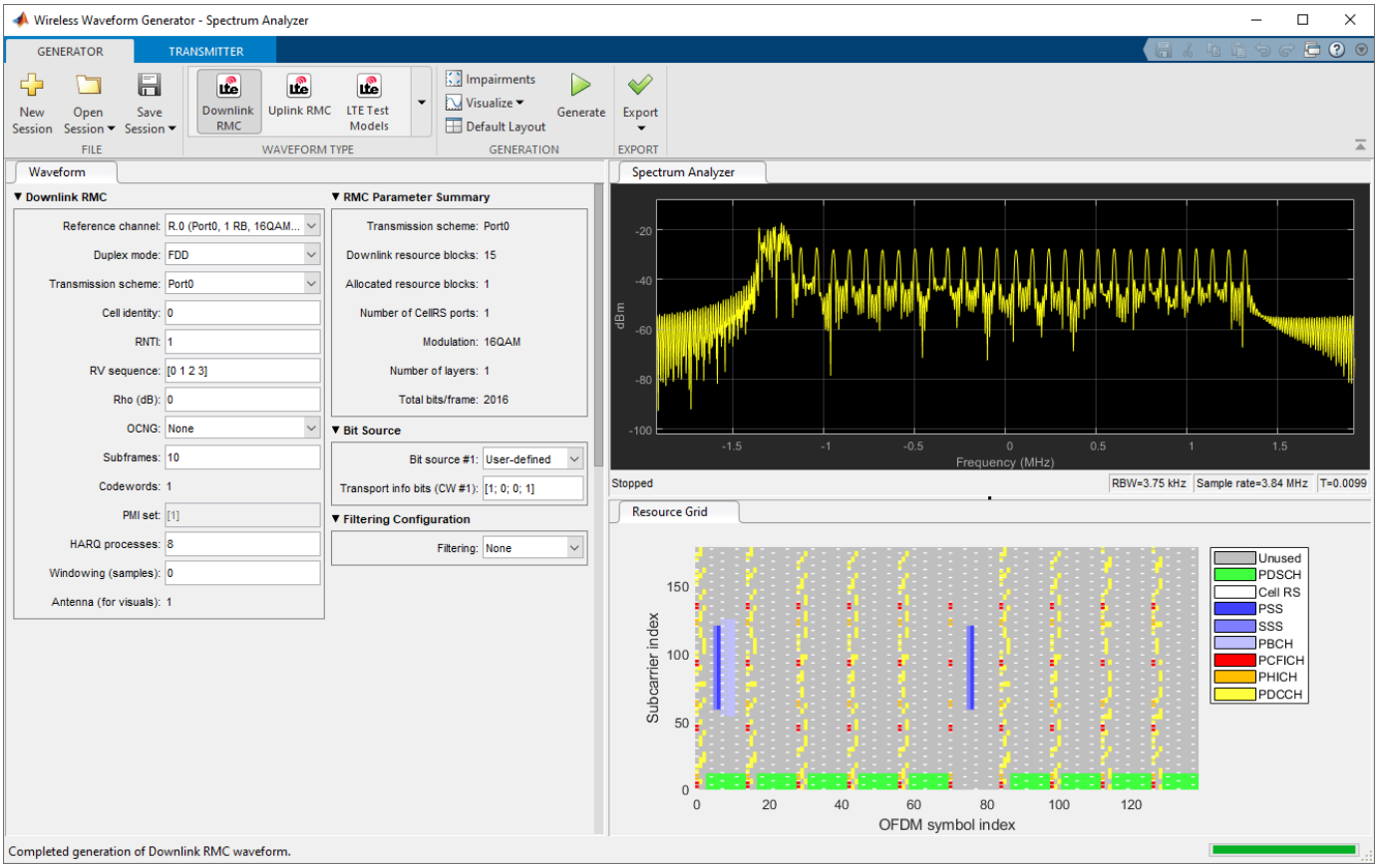
Choose the waveform you want to generate by selecting one of the options under **LTE (4G)** in the **Waveform Type** section of the app toolstrip. The app supports these waveforms.

- `Downlink RMC`
- `Uplink RMC`
- `Test Models`

### Generate LTE Waveform

Set the parameters for the selected waveform by specifying options in the **Waveform** tab on the left pane of the app. Add impairments and select visualization tools by specifying options in the **Generation** section of the app toolstrip. To visualize the waveform, click **Generate**.

For example, this figure shows the visualization results of a downlink RMC waveform with default parameters.



### Export Generated Waveform

You can export the generated waveform and its parameters by clicking **Export**. You can export the waveform to:

- A MATLAB script with a `.m` extension, which you can run to generate the waveform without the app
- A file with a `.bb` or `.mat` extension
- Your MATLAB workspace as a structure
- A Simulink® block, which you can use to generate the waveform in a Simulink model without the app

### Transmit LTE Waveform

This feature requires “Instrument Control Toolbox”™ software. To transmit a generated waveform, click the **Transmitter** tab on the app toolstrip and configure the instruments. You can use any instrument supported by the `rfsiggen` (Instrument Control Toolbox) function.

### Transmit App-Generated Wireless Waveform Using Radio Transmitters

Use the NI™ USRP™ N310, USRP N320, USRP N321, USRP X310, and USRP X410 radio transmitters, available in the **Wireless Waveform Generator** app, to transmit an app-generated waveform over the air (requires Wireless Testbench™). These radio transmitters enable you to transmit up to 2 GB of contiguous data over the air at full radio device rate.

### Introduction

The Wireless Waveform Generator app is an interactive tool for creating, impairing, visualizing, and transmitting waveforms. Using a radio transmitter available in the app, you can transmit your generated waveform repeatedly over the air. You can also export the waveform generation and transmission parameters to a runnable MATLAB script. Configure these radio transmitters to transmit an OFDM waveform. The same process applies for all waveform types that you can generate with the app.

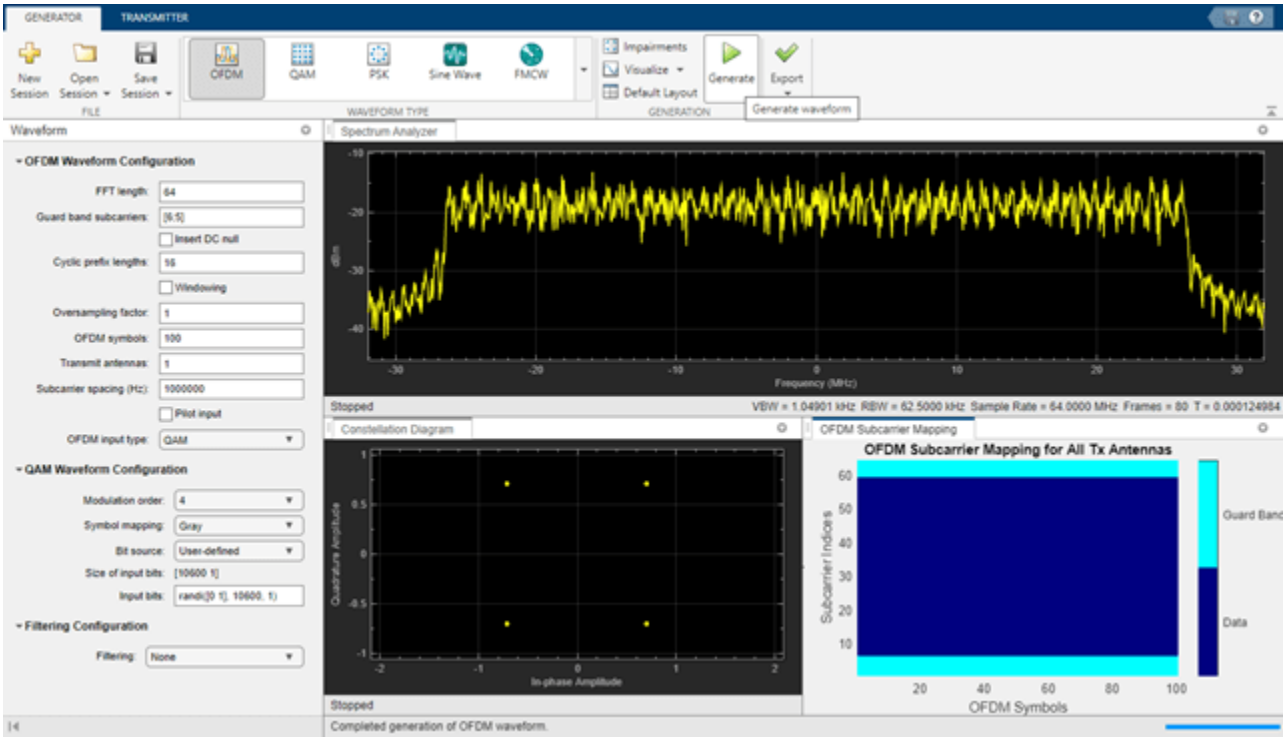
### Set Up for Radio Transmission

To use the radio transmitters in the app, you must install the Wireless Testbench Support Package for NI USRP Radios add-on, and set up your radio outside the app. For more information, see “Connect and Set Up NI USRP Radios” (Wireless Testbench).

### Generate Waveform for Transmission

Open the **Wireless Waveform Generator** app by clicking the app icon on the **Apps** tab, under **Signal Processing and Communications**. Alternatively, enter `wirelessWaveformGenerator` at the MATLAB command prompt.

In the **Waveform Type** section, select an OFDM waveform by clicking **OFDM**. In the **Waveform** pane of the app, specify the parameters of **OFDM Waveform Configuration**, **QAM Waveform Configuration**, and **Filtering Configuration** for the selected waveform. Then, generate the configuration by clicking **Generate** in the app toolstrip.



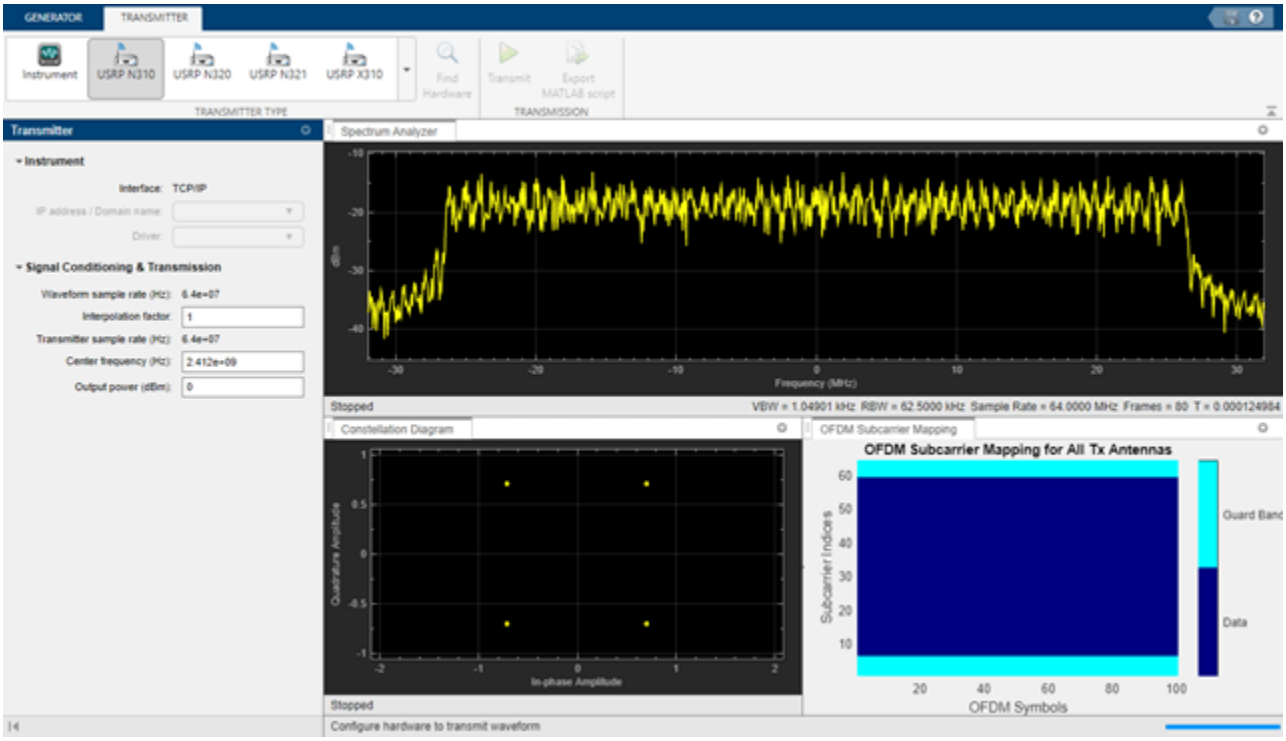

### Configure Radio Transmitter

Select the **Transmitter** tab from the app toolstrip. In the transmitter gallery, select a radio transmitter.

In the **Waveform** pane of the app, select the name of a radio setup configuration that you saved using the Radio Setup wizard. For more information, see “Connect and Set Up NI USRP Radios” (Wireless Testbench).

Set the **Center frequency**, **Gain**, and **Antennas** configuration parameters. The app automatically sets the waveform sample rate based on the waveform that you generated earlier. The radio transmitter uses onboard data buffering to ensure contiguous data transmission at up to the full hardware sample rate. If necessary, to achieve the specified sample rate, the radio uses a Farrow rate converter. Use this list as a reference when setting the sample rate:

- **USRP N310**: 120,945 Hz to 76.8 MHz, or one of: 122.88 MHz, 125 MHz, or 153.6 MHz
- **USRP N320**: 196,851 Hz to 125 MHz, or one of: 200 MHz, 245.76, MHz or 250 MHz
- **USRP N321**: 196,851 Hz to 125 MHz, or one of: 200 MHz, 245.76 MHz, or 250 MHz
- **USRP X310**: 181,418 Hz to 100 MHz, or one of: 184.32 MHz or 200 MHz
- **USRP X410**: 241,890 Hz to 125 MHz, or one of: 245.76 MHz or 250 MHz




### Transmit Waveform

To transmit the waveform continuously, click **Transmit**. To end the continuous transmission, click **Stop transmission**. To export the waveform generation and transmission parameters to a runnable MATLAB script, click **Export MATLAB script**.

# Version History

**Introduced in R2019a**

## References

[1] 3GPP TS 36.141. “Base Station (BS) conformance testing.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. https://www.3gpp.org.

[2] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. https://www.3gpp.org.

[3] 3GPP TS 36.104. “Evolved Universal Terrestrial Radio Access (E-UTRA); Base Station (BS) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. https://www.3gpp.org.

## See Also

**Apps**
**Wireless Waveform Generator**

**Topics**
“Create Waveforms Using Wireless Waveform Generator App”

# System Objects

# lte3DChannel

Filter signal through 3-D MIMO fading channel

## Description

The `lte3DChannel` System object filters an input signal through the TR 36.873 link-level multiple-input/multiple-output (MIMO) fading channel to obtain the channel-impaired signal. The object implements these channel processing steps defined in TR 36.873 [1], Section 7.3:

- Step 7: Adding ray offset angles
- Step 8: Coupling of rays
- Step 9: Generating cross-polarization power ratios (XPRs)
- Step 10: Drawing random initial phases
- Step 11: Generating channel coefficients for each cluster

To filter an input signal using the TR 36.873 link-level MIMO fading channel:

1. Create the `lte3DChannel` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

# Creation

## Syntax

```
lte3d = lte3DChannel
lte3d = lte3DChannel(Name,Value)

lte3d = lte3DChannel.makeCDL(DelayProfile)
lte3d = lte3DChannel.makeCDL(DelayProfile,DelaySpread)
lte3d = lte3DChannel.makeCDL(DelayProfile,DelaySpread,KFactor)
```

### Description

`lte3d = lte3DChannel` creates a TR 36.873 link-level MIMO System object.

`lte3d = lte3DChannel(Name,Value)` creates the object with properties set by using one or more name-value pairs. Enclose the property name inside quotes, followed by the specified value. Unspecified properties take default values.

Example: `lte3d = lte3DChannel('PathDelays',2e-6,'HasLOSCluster',true,'KFactorFirstCluster',12)` creates the channel object with a path delay of two microseconds, the LOS cluster of the delay profile enabled, and a K-factor of 12 dB for the first cluster of the delay profile.

`lte3d = lte3DChannel.makeCDL(DelayProfile)` creates the object with the specified CDL delay profile from TR 38.901 [2] Section 7.7.1, and a delay spread of 30 ns.

`lte3d = lte3DChannel.makeCDL(DelayProfile,DelaySpread)` creates the object with the specified CDL delay profile and delay spread.

`lte3d = lte3DChannel.makeCDL(DelayProfile,DelaySpread,KFactor)` creates the object with the specified CDL delay profile, delay spread, and K-factor scaling.

**Input Arguments**

**`DelayProfile` — Delay profile**
`'CDL-A'` | `'CDL-B'` | `'CDL-C'` | `'CDL-D'` | `'CDL-E'`

Delay profile, specified as one of `'CDL-A'`, `'CDL-B'`, `'CDL-C'`, `'CDL-D'`, or `'CDL-E'`.

**`DelaySpread` — Delay spread in ns**
`30` (default) | numeric scalar

Delay spread in ns, specified as a numeric scalar.

Data Types: `double`

**`KFactor` — K-factor scaling**
numeric scalar

K-factor scaling, specified as a numeric scalar. K-factor scaling applies only when you specify `DelayProfile` as `'CDL-D'` or `'CDL-E'`.

Data Types: `double`

## Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

**`PathDelays` — Discrete path delays in seconds**
`0` (default) | numeric scalar | row vector

Discrete path delays in seconds, specified as a numeric scalar or row vector. `AveragePathGains` and `PathDelays` must have the same size.

Data Types: `double`

**`AveragePathGains` — Average path gains in dB**
`0` (default) | numeric scalar | row vector

Average path gains in dB, specified as a numeric scalar or row vector. `AveragePathGains` and `PathDelays` must have the same size.

Data Types: `double`

### `AnglesAoA` — Azimuth of arrival angle in degrees

`0` (default) | numeric scalar | row vector

Azimuth of arrival angle in degrees, specified as a numeric scalar or row vector. The vector elements specify the angles for each cluster.

Data Types: `double`

### `AnglesAoD` — Azimuth of departure angle

`0` (default) | numeric scalar | row vector

Azimuth of departure angle in degrees, specified as a numeric scalar or row vector. The vector elements specify the angles for each cluster.

Data Types: `double`

### `AnglesZoA` — Zenith of arrival angle

`0` (default) | numeric scalar | row vector

Zenith of arrival angle in degrees, specified as a numeric scalar or row vector. The vector elements specify the angles for each cluster.

Data Types: `double`

### `AnglesZoD` — Zenith of departure angle

`0` (default) | numeric scalar | row vector

Zenith of departure angle in degrees, specified as a numeric scalar or row vector. The vector elements specify the angles for each cluster.

Data Types: `double`

### `HasLOSCluster` — Line of sight cluster of the delay profile

`false` (default) | `true`

Line of sight (LOS) cluster of the delay profile, specified as `false` or `true`. The `PathDelays`, `AveragePathGains`, `AnglesAoA`, `AnglesAoD`, `AnglesZoA`, and `AnglesZoD` properties define the delay profile. To enable the LOS cluster of the delay profile, set this property to `true`.

Data Types: `logical`

### `KFactorFirstCluster` — K-factor of first cluster of delay profile in dB

`13.3` (default) | numeric scalar

K-factor of the first cluster of the delay profile in dB, specified as a numeric scalar.

#### Dependencies

To enable this property, set `HasLOSCluster` to `true`.

Data Types: `double`

### `AngleSpreads` — Cluster-wise RMS angle spreads

`[5.0 11.0 3.0 3.0]` (default) | row vector

Cluster-wise root mean square (RMS) angle spreads, in degrees, for scaling ray offset angles within a cluster. Specify this property as a row vector of the form $[C_{\mathrm{AoD}}\ C_{\mathrm{AoA}}\ C_{\mathrm{ZoD}}\ C_{\mathrm{ZoA}}]$, where:

- $C_{AoD}$ is the cluster-wise RMS azimuth spread of departure angles within a cluster
- $C_{AoA}$ is the cluster-wise RMS azimuth spread of arrival angles within a cluster
- $C_{ZoD}$ is the cluster-wise RMS zenith spread of departure angles within a cluster
- $C_{ZoA}$ is the cluster-wise RMS zenith spread of arrival angles within a cluster

Data Types: `double`

**XPR — Cross-polarization power ratio in dB**
`10` (default) | numeric scalar

Cross-polarization power ratio, in dB, specified as a numeric scalar.

**Dependencies**

To enable this property, set `HasLOSCluster` to `true`.

Data Types: `double`

**`CarrierFrequency` — Carrier frequency in Hz**
`4e9` (default) | numeric scalar

Carrier frequency in Hz, specified as a numeric scalar.

Data Types: `double`

**`MaximumDopplerShift` — Maximum Doppler shift in Hz**
`5` (default) | nonnegative numeric scalar

Maximum Doppler shift in Hz, specified as a nonnegative numeric scalar. This property applies to all channel paths. When the maximum Doppler shift is set to 0, the channel remains static for the entire input. To generate a new channel realization, reset the object by calling the `reset` function.

Data Types: `double`

**`UTDirectionOfTravel` — User terminal direction of travel in degrees**
`[0; 90]` (default) | two-element column vector

User terminal (UT) direction of travel in degrees, specified as a two-element column vector. The vector elements specify the azimuth and the elevation components: `[azimuth; elevation]`.

Data Types: `double`

**`SampleRate` — Sample rate of input signal in Hz**
`30.72e6` (default) | positive numeric scalar

Sample rate of input signal in Hz, specified as a positive numeric scalar.

Data Types: `double`

**`TransmitAntennaArray` — Transmit antenna array characteristics**
structure

Transmit antenna array characteristics, specified as a structure that contains the following fields:

| Parameter Field | Values | Description |
|---|---|---|
| **Size** | `[2 2 2]` (default),<br><br>row vector | Size of antenna array, specified as a row vector of the form [M N P].<br><br>• M and N are the number of rows and columns in the antenna array, respectively.<br>• P is the number of polarizations (1 or 2).<br><br>The antenna array elements are mapped to the input waveform channels (columns) in the order that a 3-D array of size M-by-N-by-P is linearly indexed across the first dimension to the last.<br><br>For example, an antenna array of size `[4 8 2]` has the first M = 4 channels mapped to the first column of the first polarization angle. The next M = 4 antennas are mapped to the next column, and so on. Following this pattern, the first M×N = 32 channels are mapped to the first polarization angle of the complete antenna array. Similarly, the remaining 32 channels are mapped to the second polarization angle of the complete antenna array. |
| | | For an antenna array with multiple panels, specify the size as a row vector of the form [M N P $M_g$ $N_g$], where $M_g$ and $N_g$ are the number of row and column array panels, respectively.<br><br>The antenna array elements are mapped panel-wise to the waveform channels in the order that a 5-D array of size M-by-N-by-P-by-$M_g$-by-$N_g$ is linearly indexed across the first dimension to the last. Subsequent sets of M×N×P= 64 channels are mapped to consecutive panels, taking panel rows first, then panel columns. |
| **ElementSpacing** | `[0.5 0.5]` (default),<br><br>row vector | Element spacing in wavelengths, specified as a row vector of the form [$\lambda_v$ $\lambda_h$], representing the vertical and horizontal element spacing. |
| | | For an antenna array with multiple panels, specify the spacing as a row vector of the form [$\lambda_v$ $\lambda_h$ $dg_v$ $dg_h$], where $dg_v$ and $dg_h$ are the vertical and horizontal panel spacing, respectively. |
| **PolarizationAngles** | `[45 -45]` (default),<br><br>row vector | Polarization angles in degrees, specified as a row vector of the form [θ ρ]. Polarization angles apply only when the number of polarizations is 2. |
| **Orientation** | `[0; 0; 0]` (default),<br><br>column vector | Mechanical orientation of the array, in degrees, specified as a column vector of the form [α; β; γ] describing bearing, downtilt, and slant. The default value indicates that the broadside direction of the array points to the positive x-axis. |

| Parameter Field | Values | Description |
|---|---|---|
| **`Element`** | `'36.873'` (default),<br><br>`'isotropic'` | Antenna element radiation pattern. See TR 36.873 [1], Section 7.1.1. |
| **`PolarizationModel`** | `'Model-2'` (default),<br><br>`'Model-1'` | Model that determines the radiation field patterns based on a defined radiation power pattern. See TR 36.873 [1], Section 7.1.1. |

Data Types: `struct`

**`ReceiveAntennaArray` — Receive antenna array characteristics**
structure

Receive antenna array characteristics, specified as a structure that contains the following fields:

| Parameter Field | Values | Description |
|---|---|---|
| **`Size`** | `[2 2 2]` (default),<br><br>row vector | Size of antenna array, specified as a row vector of the form [M N P].<br>• M and N are the number of rows and columns in the antenna array.<br>• P is the number of polarizations (1 or 2).<br><br>The antenna array elements are mapped to the input waveform channels (columns) in the order that a 3-D array of size M-by-N-by-P is linearly indexed across the first dimension to the last.<br><br>For example, an antenna array of size `[4 8 2]` has the first M = 4 channels mapped to the first column of the first polarization angle. The next M = 4 antennas are mapped to the next column, and so on. Following this pattern, the first M×N = 32 channels are mapped to the first polarization angle of the complete antenna array. Similarly, the remaining 32 channels are mapped to the second polarization angle of the complete antenna array. |
| | | For an antenna array with multiple panels, you can specify the size as a row vector of the form [M N P $M_g$ $N_g$], where $M_g$ and $N_g$ are the number of row and column array panels, respectively.<br><br>The antenna array elements are mapped panel-wise to the waveform channels in the order that a 5-D array of size M-by-N-by-P-by-$M_g$-by-$N_g$ is linearly indexed across the first dimension to the last. Subsequent sets of M×N×P= 64 channels are mapped to consecutive panels, taking panel rows first, then panel columns. |

| Parameter Field | Values | Description |
|---|---|---|
| **ElementSpacing** | `[0.5 0.5]` (default),<br><br>row vector | Element spacing in wavelengths, specified as a a row vector of the form [$\lambda_v$ $\lambda_h$] representing the vertical and horizontal element spacing, respectively. |
| | | For an antenna array with multiple panels, you can specify the spacing as a row vector of the form [$\lambda_v$ $\lambda_h$ $dg_v$ $dg_h$], where $dg_v$ and $dg_h$ are the vertical and horizontal panel spacing, respectively. |
| **PolarizationAngles** | `[0 90]` (default),<br><br>row vector | Polarization angles in degrees, specified as a row vector of the form [θ ρ]. Polarization angles apply only when the number of polarizations is 2. |
| **Orientation** | `[0; 0; 0]` (default),<br><br>column vector | Mechanical orientation of the array, in degrees, specified as a column vector of the form [α; β; γ] describing bearing, downtilt, and slant, respectively. The default value indicates that the broadside direction of the array points to the positive x-axis. |
| **Element** | `'isotropic'` (default),<br><br>`'36.873'` | Antenna element radiation pattern. See TR 36.873 [1], Section 7.1.1. |
| **PolarizationModel** | `'Model-2'` (default),<br><br>`'Model-1'` | Model that determines the radiation field patterns based on a defined radiation power pattern. See TR 36.873 [1], Section 7.1.1. |

Data Types: `structure`

**`SampleDensity` — Number of time samples per half wavelength**
64 (default) | `Inf` | numeric scalar

Number of time samples per half wavelength, specified as a numeric scalar. The `SampleDensity` and `MaximumDopplerShift` properties control the coefficient generation sampling rate, *Fcg*:

*Fcg* = `MaximumDopplerShift` × 2 × `SampleDensity`.

Setting `SampleDensity` to `Inf` assigns *Fcg* the value of the `SampleRate` property.

Data Types: `double`

**`NormalizePathGains` — Normalize path gains**
`true` (default) | `false`

Normalize path gains, specified as `true` or `false`. Use this property to normalize the fading processes. When this property is set to `true`, the total power of the path gains, averaged over time, is 0 dB. When this property is set to `false`, the path gains are not normalized. The `AveragePathGains` property specifies the average powers of the path gains.

Data Types: `logical`

**`InitialTime` — Start time of fading process in seconds**
`0.0` (default) | numeric scalar

Start time of fading process in seconds, specified as a numeric scalar.

**Tunable:** Yes

Data Types: `double`

### `NumStrongestClusters` — Number of strongest clusters to split into subclusters

`0` (default) | numeric scalar

Number of strongest clusters to split into subclusters, specified as a numeric scalar. See TR 36.873 [1], Section 7.3, Step 11.

Data Types: `double`

### `ClusterDelaySpread` — Cluster delay spread in seconds

`3.90625e-9` (default) | nonnegative scalar

Cluster delay spread in seconds, specified as a nonnegative scalar. Use this property to specify the delay offset between subclusters for clusters split into subclusters. See TR 36.873 [1], Section 7.3, Step 11.

#### Dependencies

To enable this property, set `NumStrongestClusters` to a value greater than zero.

Data Types: `double`

### `RandomStream` — Source of random number stream

`'mt19937ar with seed'` (default) | `'Global stream'`

Source of random number stream, specified as one of the following:

- `'mt19937ar with seed'` — The object uses the mt19937ar algorithm for normally distributed random number generation. Calling the `reset` function resets the filters and reinitializes the random number stream to the value of the `Seed` property.
- `'Global stream'` — The object uses the current global random number stream for normally distributed random number generation. Calling the `reset` function resets only the filters.

### `Seed` — Initial seed of mt19937ar random number stream

`73` (default) | nonnegative numeric scalar

Initial seed of mt19937ar random number stream, specified as a nonnegative numeric scalar.

#### Dependencies

To enable this property, set `RandomStream` to `'mt19937ar with seed'`. When calling the `reset` function, the seed reinitializes the mt19937ar random number stream.

Data Types: `double`

### `ChannelFiltering` — Filter input signal

`true` (default) | `false`

Filter input signal, specified as `true` or `false`. When this property is set to `false`, the object takes no input signal, and the path gains and sample times are the only outputs. In this case, the `NumTimeSamples` property controls the duration of the fading process realization at a sampling rate given by the `SampleRate` property.

Data Types: `logical`

#### `NumTimeSamples` — Number of time samples

`30720` (default) | positive integer

Number of time samples, specified as a positive integer. Use this property to set the duration of the fading process realization.

**Tunable:** Yes

##### Dependencies

To enable this property, set `ChannelFiltering` to `false`.

Data Types: `double`

#### `NormalizeChannelOutputs` — Normalize channel outputs by the number of receive antennas

`true` (default) | `false`

Normalize channel outputs by the number of receive antennas, specified as `true` or `false`.

##### Dependencies

To enable this property, set `ChannelFiltering` to `true`.

Data Types: `double`

## Usage

## Syntax

```
signalOut = lte3d(signalIn)
[signalOut,pathGains] = lte3d(signalIn)
[signalOut,pathGains,sampleTimes] = lte3d(signalIn)

pathGains = lte3d()
[pathGains,sampleTimes] = lte3d()
```

### Description

`signalOut = lte3d(signalIn)` filters the input signal through a TR 36.873 link-level MIMO fading channel System object `lte3d` and returns the channel-impaired signal.

`[signalOut,pathGains] = lte3d(signalIn)` also returns the MIMO channel path gains of the underlying fading process.

`[signalOut,pathGains,sampleTimes] = lte3d(signalIn)` also returns the sample times of the channel snapshots of `pathGains` (first-dimension elements).

`pathGains = lte3d()` returns only the path gains. In this case, the `NumTimeSamples` property determines the duration of the fading process. The object acts as a source of path gains without filtering an input signal.

To use this syntax, you must set the `ChannelFiltering` property of `lte3d` to `false`.

`[pathGains,sampleTimes] = lte3d()` also returns the sample times. The object acts as a source of the path gains and sample times without filtering an input signal.

To use this syntax, you must set the `ChannelFiltering` property of `lte3d` to `false`.

### Input Arguments

#### `signalIn` — Input signal

complex scalar | vector | $N_S$-by-$N_T$ matrix

Input signal, specified as a complex scalar, vector, or $N_S$-by-$N_T$ matrix, where:

- $N_S$ is the number of samples.
- $N_T$ is the number of transmit antennas.

Data Types: `single` | `double`
Complex Number Support: Yes

### Output Arguments

#### `signalOut` — Output signal

complex scalar | vector | $N_S$-by-$N_R$ matrix

Output signal, returned as a complex scalar, vector, or $N_S$-by-$N_R$ matrix, where:

- $N_S$ is the number of samples.
- $N_R$ is the number of receive antennas.

The output signal data type is of the same precision as the input signal data type.

Data Types: `single` | `double`
Complex Number Support: Yes

#### `pathGains` — MIMO channel path gains of fading process

$N_{CS}$-by-$N_P$-by-$N_T$-by-$N_R$ complex matrix

MIMO channel path gains of the fading process, returned as a $N_{CS}$-by-$N_P$-by-$N_T$-by-$N_R$ complex matrix, where:

- $N_{CS}$ is the number of channel snapshots, controlled by the `SampleDensity` property.
- $N_P$ is the number of paths, given by the size of the `PathDelays` property.
- $N_T$ is the number of transmit antennas.
- $N_R$ is the number of receive antennas.

The path gains data type is of the same precision as the input signal data type.

Data Types: `single` | `double`
Complex Number Support: Yes

#### `sampleTimes` — Sample times of channel snapshots

$N_{CS}$-by-1 column vector

Sample times of channel snapshots, returned as an $N_{CS}$-by-1 column vector, where $N_{CS}$ is the number of channel snapshots, controlled by the `SampleDensity` property.

Data Types: `double`

## Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

## Specific to `lte3DChannel`

| | |
|---|---|
| displayChannel | Visualize and explore 3-D MIMO fading channel model characteristics |
| getPathFilters | Get path filter impulse response for 3-D MIMO fading channel |
| info | Get characteristic information about 3-D MIMO fading channel |

## Common to All System Objects

| | |
|---|---|
| step | Run System object algorithm |
| clone | Create duplicate System object |
| isLocked | Determine if System object is in use |
| release | Release resources and allow changes to System object property values and input characteristics |
| reset | Reset internal states of System object |

## Examples

### Transmission over 3-D Channel with Delay Profile CDL-D

Transmit an LTE waveform through a 3-D channel with delay profile CDL-D from TR 38.901 Section 7.7.1.

Define the transmission waveform configuration structure, initialized to reference measurement channel (RMC) R.50, TDD (10MHz, QPSK, R=1/3, 1 layer, 8 CSI-RS ports), and one subframe.

```
rmc = lteRMCDL('R.50','TDD');
rmc.TotSubframes = 1;
data = [1; 0; 0; 1];
[txWaveform,~,txInfo] = lteRMCDLTool(rmc,data);
```

Define the channel configuration structure using an `lte3DChannel` System object. Use delay profile CDL-D from TR 38.901 Section 7.7.1, a delay spread of 10 ns, and UT velocity of 15 km/h:

```
v = 15.0;                    % UT velocity in km/h
fc = 4e9;                    % carrier frequency in Hz
c = physconst('lightspeed'); % speed of light in m/s
fd = (v*1000/3600)/c*fc;     % UT max Doppler frequency in Hz

lte3d = lte3DChannel.makeCDL('CDL-D',10e-9);
lte3d.CarrierFrequency = fc;
lte3d.MaximumDopplerShift = fd;
lte3d.SampleRate = txInfo.SamplingRate;
```

Configure the transmit array as [M N P] = [2 2 2], representing a 2-by-2 antenna array (M=2, N=2) and P=2 polarization angles. Configure the receive antenna array as [M N P] = [1 1 2], representing a single pair of cross-polarized co-located antennas.

```
lte3d.TransmitAntennaArray.Size = [2 2 2];
lte3d.ReceiveAntennaArray.Size = [1 1 2];
```

Call the 3-D channel object on the input waveform.

```
rxWaveform = lte3d(txWaveform);
```

### Explore Effect of SampleDensity Property on Channel Output

Plot channel output and path gain snapshots for various sample density values while using an `lte3DChannel` System object.

Configure a 3-D channel for SISO operation and delay profile CDL-B from TR 38.901 Section 7.7.1. Set the maximum Doppler shift to 300 Hz and the channel sampling rate to 10 kHz.

```
lte3d = lte3DChannel.makeCDL('CDL-B');
lte3d.MaximumDopplerShift = 300.0;
lte3d.SampleRate = 10e3;
lte3d.Seed = 19;
```

Configure transmit and receive antenna arrays.

```
lte3d.TransmitAntennaArray.Size = [1 1 1];
lte3d.ReceiveAntennaArray.Size = [1 1 1];
```

Create an input waveform with a length of 40 samples.

```
T = 40;
in = ones(T,1);
```

Plot the step response of the channel (displayed as lines) and the corresponding path gain snapshots (displayed circles) for various values of the `SampleDensity` property. The sample density property controls how often the channel snapshots are taken relative to the Doppler frequency.

- When `SampleDensity = Inf`, a channel snapshot is taken for every input sample.
- When `SampleDensity = X`, a channel snapshot is taken at a rate of `Fcs = 2*X*MaximumDopplerShift`.

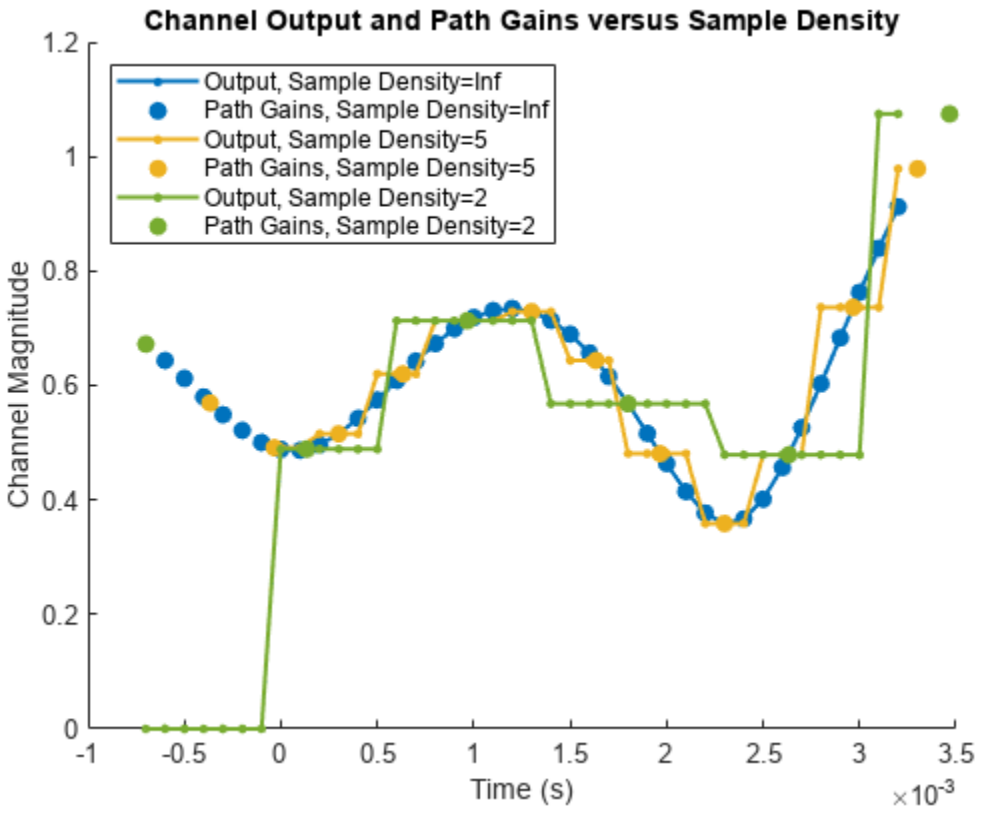
The `lte3DChannel` object applies the channel snapshots to the input waveform by means of zero-order hold interpolation. The object takes an extra snapshot beyond the end of the input. Some of the final output samples use this extra value to minimize the interpolation error. The channel output contains a transient (and a delay) due to the filters that implement the path delays.

```
s = [Inf 5 2]; % sample densities

legends = {};
figure; hold on;
SR = lte3d.SampleRate;

for i = 1:length(s)

    % call channel with chosen sample density
    release(lte3d); lte3d.SampleDensity = s(i);
    [out,pathgains,sampletimes] = lte3d(in);
```

```
    chInfo = info(lte3d); tau = chInfo.ChannelFilterDelay;

    % plot channel output against time
    t = lte3d.InitialTime + ((0:(T-1)) - tau).' / SR;
    h = plot(t,abs(out),'o-'); h.MarkerSize = 2; h.LineWidth = 1.5;
    desc = ['Sample Density=' num2str(s(i))];
    legends = [legends ['Output, ' desc]];
    disp([desc ', Ncs=' num2str(length(sampletimes))]);

    % plot path gains against sample times
    h2 = plot(sampletimes - tau/SR,abs(sum(pathgains,2)),'o');
    h2.Color = h.Color; h2.MarkerFaceColor = h.Color;
    legends = [legends ['Path Gains, ' desc]];

end
```

```
Sample Density=Inf, Ncs=40
Sample Density=5, Ncs=13
Sample Density=2, Ncs=6
```

```
xlabel('Time (s)');
title('Channel Output and Path Gains versus Sample Density');
ylabel('Channel Magnitude');
legend(legends,'Location','NorthWest');
```



### Waveform Spectrum of 3-D Channel Filtered LTE OFDM Modulation

Display waveform spectrum of an LTE OFDM modulation waveform passed through a 40-by-2 channel using the `lte3DChannel` System object.

Create a resource grid for 40 antennas.

```
enb.NDLRB = 25;
enb.CyclicPrefix = 'Normal';
grid = lteDLResourceGrid(enb,40);

Fill the grid with QPSK symbols and perform LTE OFDM modulation.

grid(:) = lteSymbolModulate(randi([0 1],numel(grid)*2,1),'QPSK');
[txWaveform,txInfo] = lteOFDMModulate(enb,grid);
```

Create an `lte3DChannel` System object with specific properties.

```
lte3d = lte3DChannel('PathDelays',[0 500e-9], ...
    'AveragePathGains',[-13.4 3.0], ...
    'AnglesAoD',[-178.1 -4.2], ...
    'AnglesAoA',[51.3 -152.7], ...
    'AnglesZoD',[50.2 93.2], ...
    'AnglesZoA',[125.4 91.3], ...
    'NumStrongestClusters',1, ...
    'SampleRate',txInfo.SamplingRate);
```

Configure transmit and receive antenna arrays.

```
lte3d.TransmitAntennaArray.Size = [10 2 2];
lte3d.ReceiveAntennaArray.Size = [1 1 2];
```

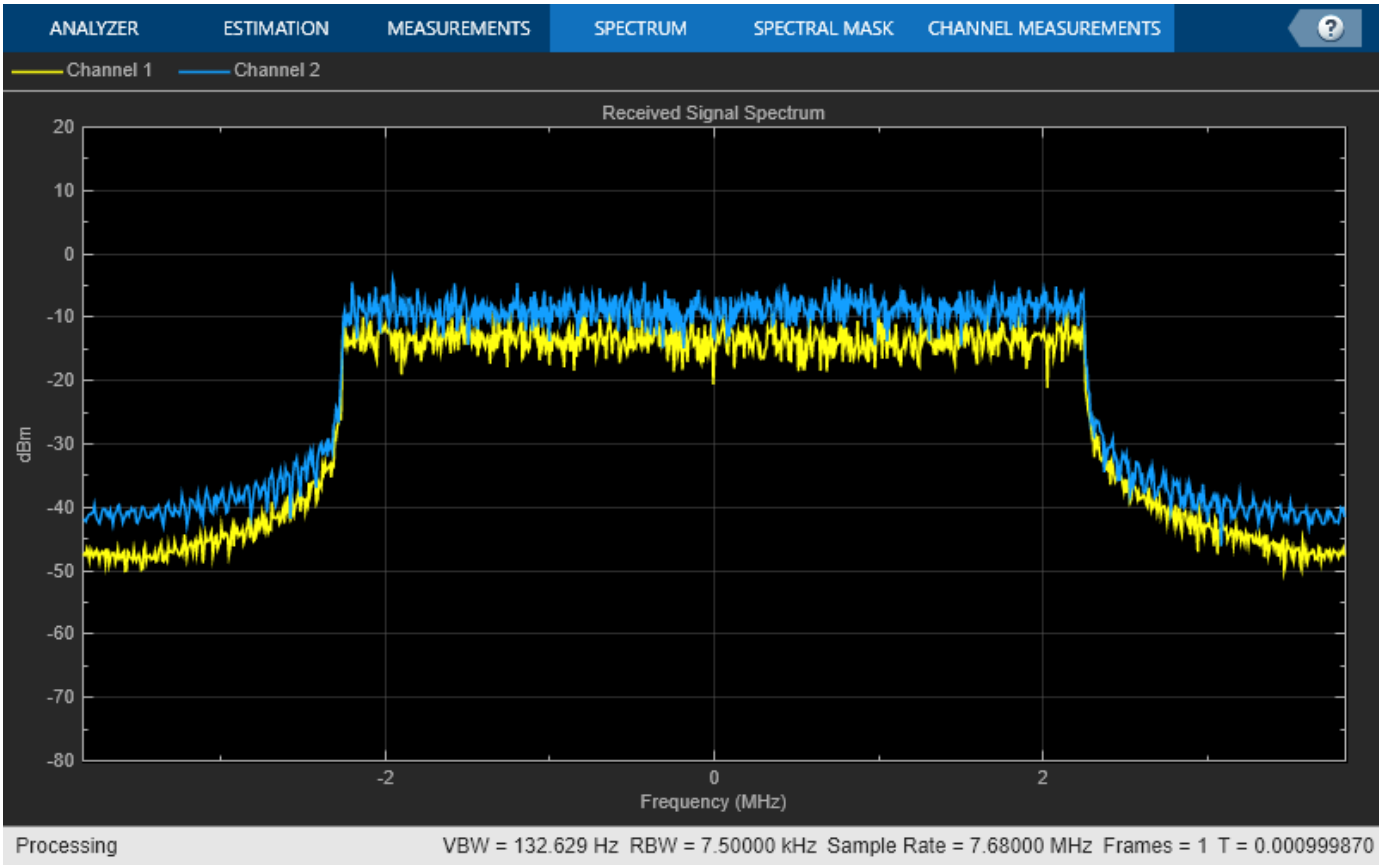
The antenna array elements are mapped to the waveform channels (columns) using linear indexing of `TransmitAntennaArray.Size` or `ReceiveAntennaArray.Size` across the first dimension to the last. See the `TransmitAntennaArray` or `ReceiveAntennaArray` properties of the `lte3DChannel` System object for more details.

Pass the LTE OFDM modulation waveform through the 40-by-2 3-D channel.

```
rxWaveform = lte3d(txWaveform);
```

Plot the received waveform spectrum.

```
analyzer = spectrumAnalyzer(SampleRate=lte3d.SampleRate);
analyzer.Title = 'Received Signal Spectrum';
analyzer(rxWaveform);
```




## Version History

**Introduced in R2018a**

## References

[1] 3GPP TR 36.873. “Study on 3D channel model for LTE.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*. URL: https://www.3gpp.org.

[2] 3GPP TR 38.901. “Study on channel model for frequencies from 0.5 to 100 GHz.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

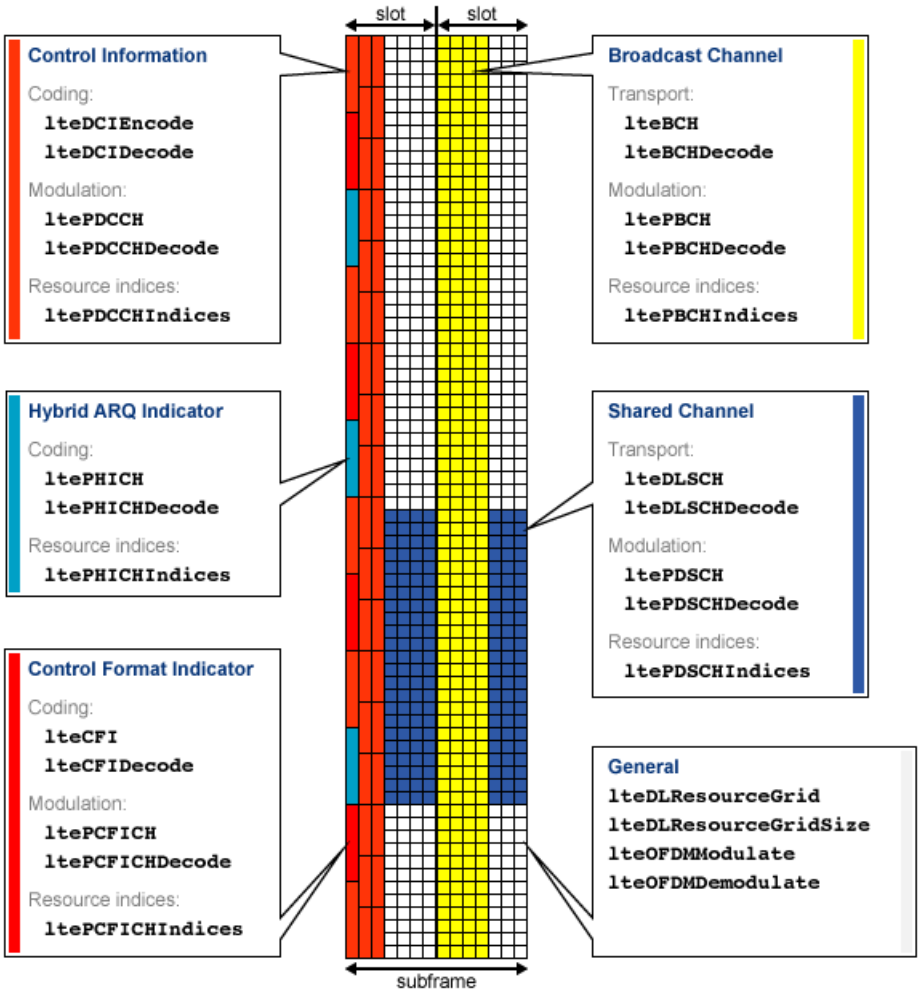
## See Also

`lteFadingChannel`

# Resource Grid and Block Diagrams

# Downlink Physical Channels Grid

The downlink physical channels, their associated functions, and their locations on the resource grid are shown in the following figure.

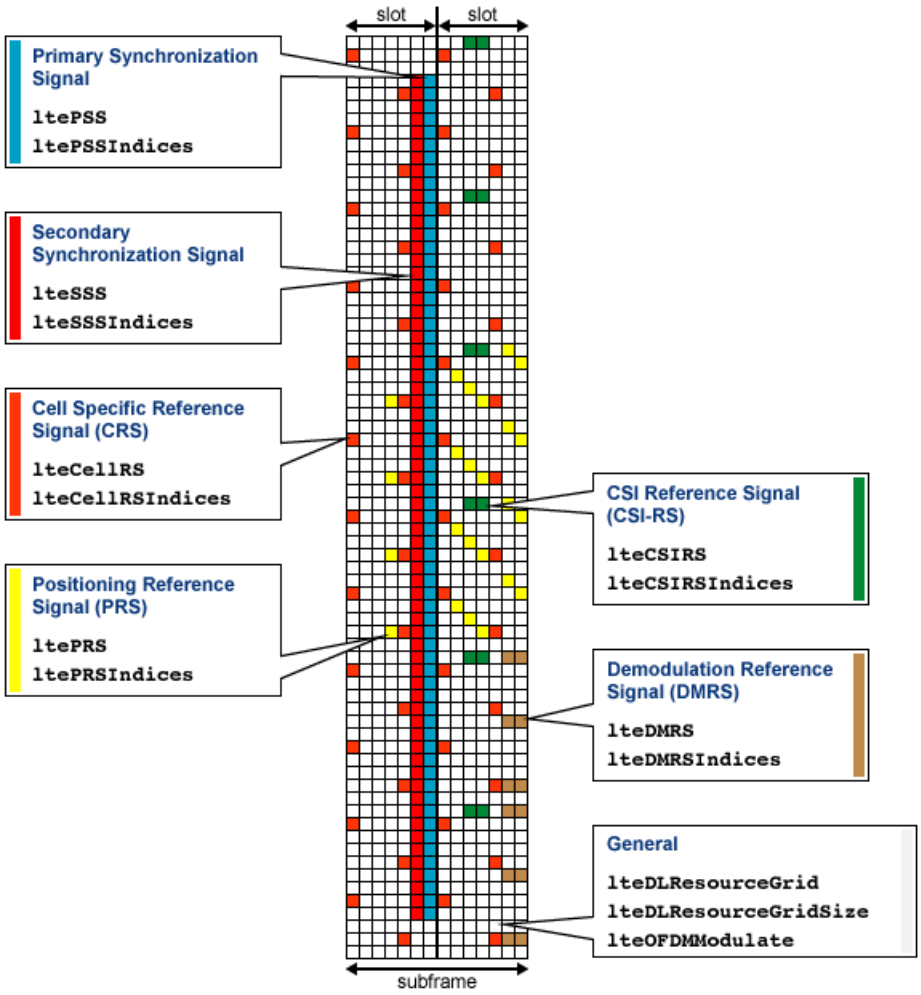



- Control Information
  - Coding
    - `lteDCIEncode`
    - `lteDCIDecode`
  - Modulation
    - `ltePDCCH`
    - `ltePDCCHDecode`
  - Resource Indices — `ltePDCCHIndices`
- Hybrid ARQ Indicator

  - Coding
    - `ltePHICH`
    - `ltePHICHDecode`
  - Resource Indices — `ltePHICHIndices`
- Control Format Indicator
  - Coding
    - `lteCFI`
    - `lteCFIDecode`
  - Modulation
    - `ltePCFICH`
    - `ltePCFICHDecode`
  - Resource Indices — `ltePCFICHIndices`
- Broadcast Channel
  - Transport
    - `lteBCH`
    - `lteBCHDecode`
  - Modulation
    - `ltePBCH`
    - `ltePBCHDecode`
  - Resource Indices — `ltePBCHIndices`
- Shared Channel
  - Transport
    - `lteDLSCH`
    - `lteDLSCHDecode`
  - Modulation
    - `ltePDSCH`
    - `ltePDSCHDecode`
  - Resource Indices — `ltePDSCHIndices`
- General
  - `lteDLResourceGrid`
  - `lteDLResourceGridSize`
  - `lteOFDMModulate`
  - `lteOFDMDemodulate`

# Downlink Physical Signals Grid

The downlink physical signals, their associated functions, and their locations on the resource grid are shown in the following figure.

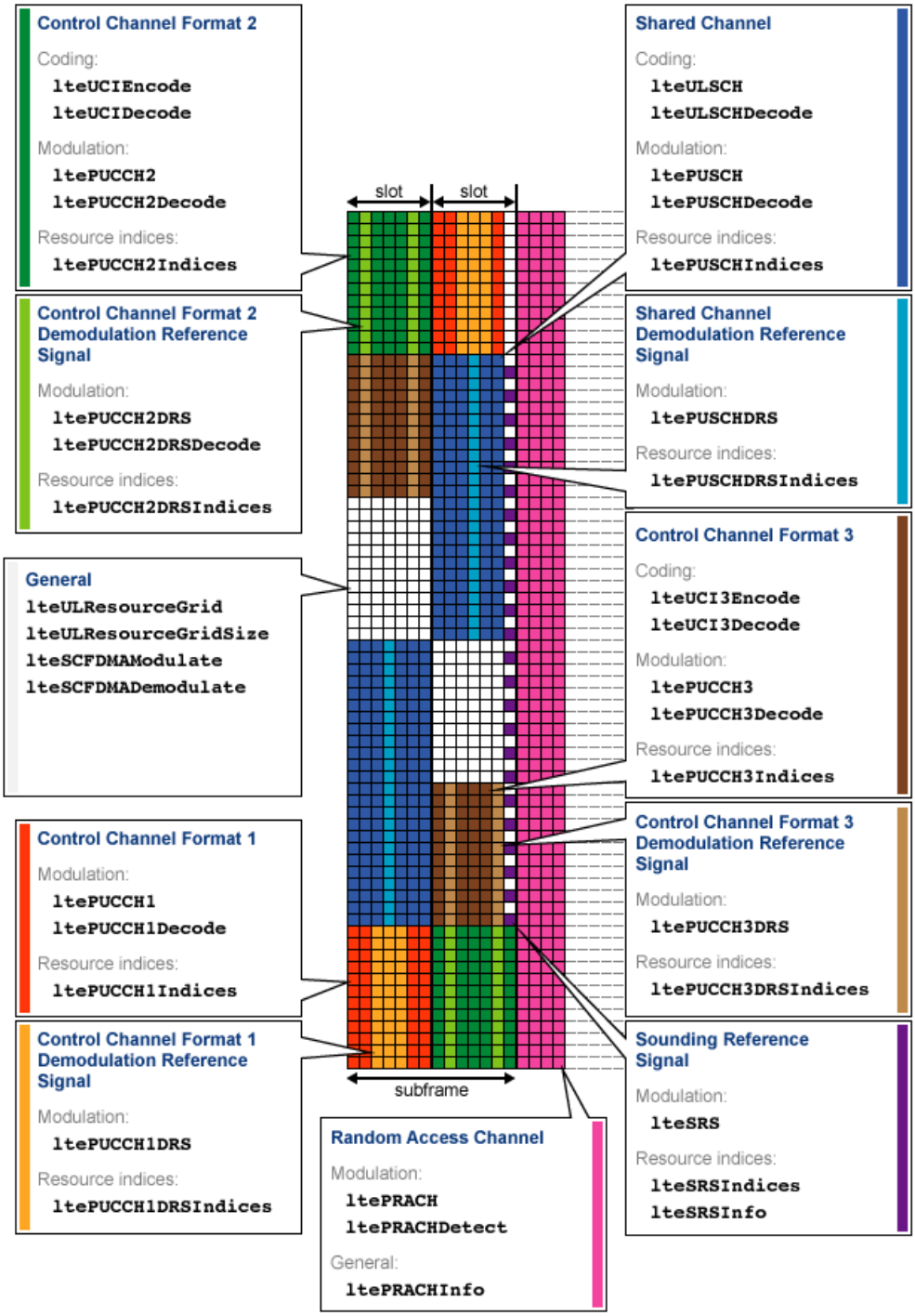



- Primary Synchronization Signal (PSS)
  - `ltePSS`
  - `ltePSSIndices`
- Secondary Synchronization Signal (SSS)
  - `lteSSS`
  - `lteSSSIndices`
- Cell-specific Reference Signal (CRS)
  - `lteCellRS`
  - `lteCellRSIndices`

- Positioning Reference Signal (PRS)
  - `ltePRS`
  - `ltePRSIndices`
- Channel State Information Reference Signal (CSI-RS)
  - `lteCSIRS`
  - `lteCSIRSIndices`
- Demodulation Reference Signal (DMRS)
  - `lteDMRS`
  - `lteDMRSIndices`
- General
  - `lteDLResourceGrid`
  - `lteDLResourceGridSize`
  - `lteOFDMModulate`

# Uplink Physical Channels and Signals Grid

The uplink physical channels and signals, their associated functions, and their locations on the resource grid are shown in the following figure.

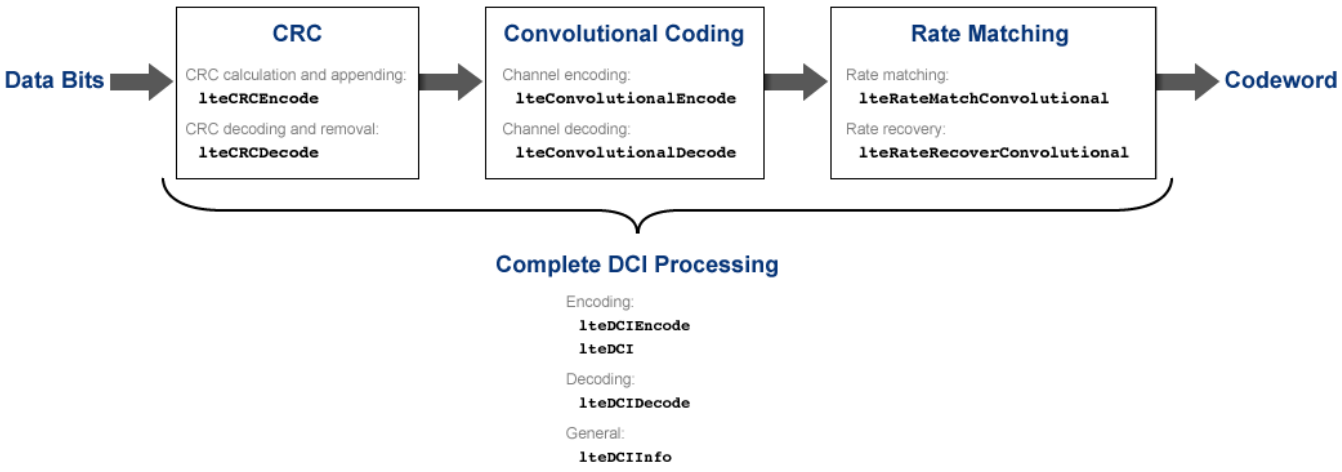


- Control Channel Format 1
  - Modulation
    - `ltePUCCH1`
    - `ltePUCCH1Decode`
  - Resource Indices — `ltePUCCH1Indices`
- Control Channel Format 2
  - Coding
    - `lteUCIEncode`
    - `lteUCIDecode`
  - Modulation
    - `ltePUCCH2`
    - `ltePUCCH2Decode`
  - Resource Indices — `ltePUCCH2Indices`
- Control Channel Format 3
  - Coding
    - `lteUCI3Encode`
    - `lteUCI3Decode`
  - Modulation
    - `ltePUCCH3`
    - `ltePUCCH3Decode`
  - Resource Indices — `ltePUCCH3Indices`
- Random Access Channel
  - Modulation
    - `ltePRACH`
    - `ltePRACHDetect`
  - General — `ltePRACHInfo`
- Shared Channel
  - Transport
    - `lteULSCH`
    - `lteULSCHDecode`
  - Modulation
    - `ltePUSCH`
    - `ltePUSCHDecode`
  - Resource Indices
    - `ltePUSCHIndices`

- Control Channel Format 1 Demodulation Reference Signal
  - Modulation — `ltePUCCH1DRS`
  - Resource Indices — `ltePUCCH1DRSIndices`
- Control Channel Format 2 Demodulation Reference Signal
  - Modulation
    - `ltePUCCH2DRS`
    - `ltePUCCH2DRSDecode`
  - Resource Indices — `ltePUCCH2DRSIndices`
- Control Channel Format 3 Demodulation Reference Signal
  - Modulation — `ltePUCCH3DRS`
  - Resource Indices — `ltePUCCH3DRSIndices`
- Sounding Reference Signal
  - Modulation — `lteSRS`
  - Resource Indices — `lteSRSIndices`
  - General — `lteSRSInfo`
- Shared Channel Demodulation Reference Signal
  - Modulation — `ltePUSCHDRS`
  - Resource Indices — `ltePUSCHDRSIndices`
- General
  - `lteULResourceGrid`
  - `lteULResourceGridSize`
  - `lteSCFDMAModulate`
  - `lteSCFDMADemodulate`

# DCI Processing Functions

The complete downlink control information process and associated low-level and mid-level DCI functions are shown in the following block diagram.




- Cyclic redundancy check (CRC)
  - CRC calculation and appending — `lteCRCEncode`
  - CRC decoding and removal — `lteCRCDecode`
- Convolutional channel coding
  - Channel encoding — `lteConvolutionalEncode`
  - Channel decoding — `lteConvolutionalDecode`
- Rate matching and recovery
  - Rate matching — `lteRateMatchConvolutional`
  - Rate recovery — `lteRateRecoverConvolutional`
- Complete DCI processing
  - Encoding
    - `lteDCIEncode`
    - `lteDCI`
  - Decoding — `lteDCIDecode`
  - General — `lteDCIInfo`

## See Also

## Related Examples

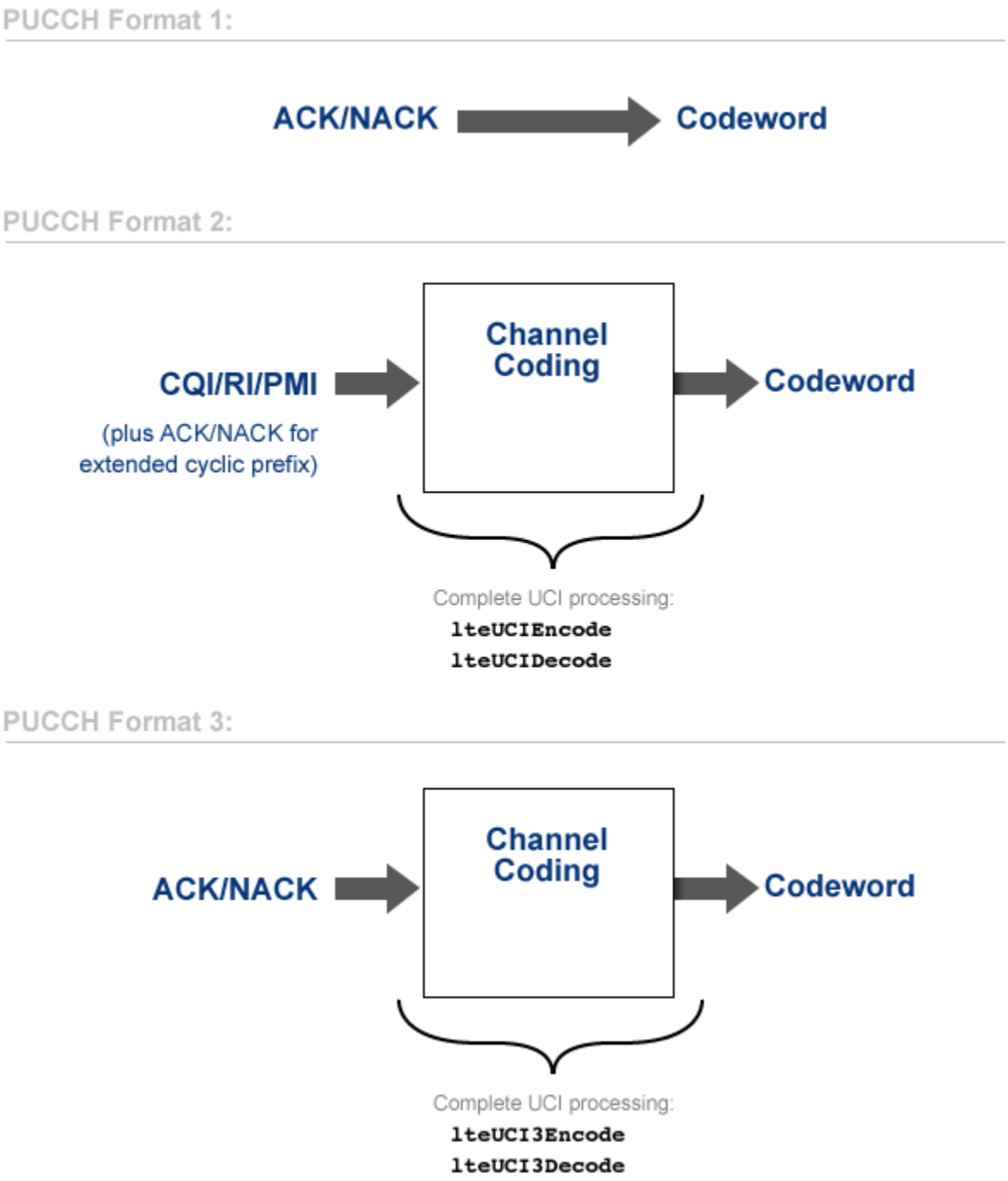
- “Model DCI and PDCCH”

## More About

- “Downlink Control Channel”

# UCI Processing Functions

The uplink control information process for PUCCH format 1, 2, and 3 and associated mid-level UCI functions are shown in the following block diagram.




- PUCCH format 2 complete UCI processing

  - `lteUCIEncode`
  - `lteUCIDecode`
- PUCCH format 3 complete UCI processing
  - `lteUCI3Encode`
  - `lteUCI3Decode`

## See Also

## Related Examples

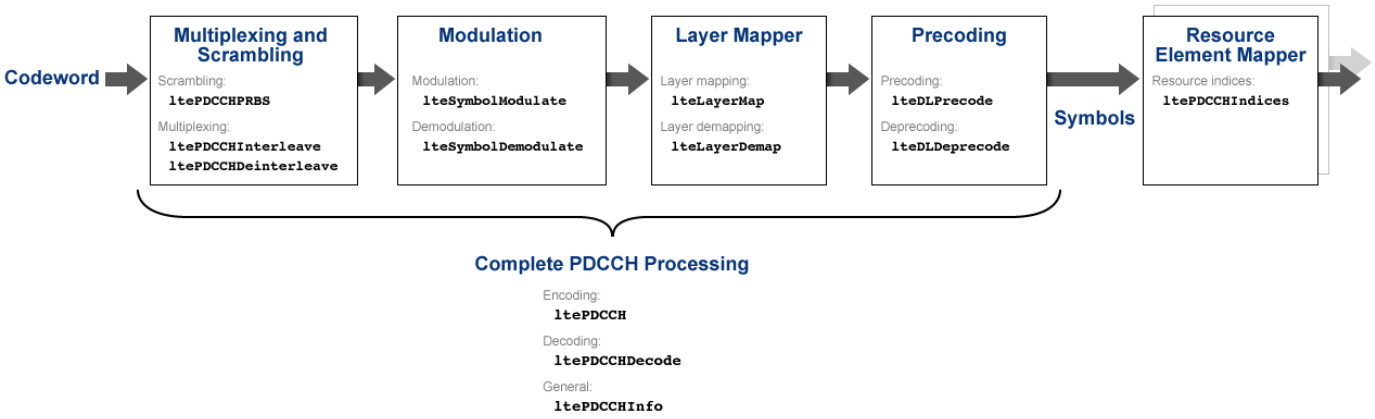
- “Model PUCCH Format 2”

## More About

- “Uplink Control Channel Format 1”
- “Uplink Control Channel Format 2”

# PDCCH Processing Functions

The complete physical downlink control channel process and associated low-level and mid-level PDCCH functions are shown in the following block diagram.




- Multiplexing and scrambling
  - Scrambling — `ltePDCCHPRBS`
  - Multiplexing
    - `ltePDCCHInterleave`
    - `ltePDCCHDeinterleave`
- Symbol modulation and demodulation
  - Modulation — `lteSymbolModulate`
  - Demodulation — `lteSymbolDemodulate`
- Layer mapper and de-mapper
  - Layer mapping — `lteLayerMap`
  - Layer demapping — `lteLayerDemap`
- Precoding and deprecoding
  - Precoding — `lteDLPrecode`
  - Deprecoding — `lteDLDeprecode`
- Resource element mapper
  - Resource indices — `ltePDCCHIndices`
- Complete PDCCH processing
  - Encoding — `ltePDCCH`
  - Decoding — `ltePDCCHDecode`
  - General — `ltePDCCHInfo`

## See Also

## Related Examples

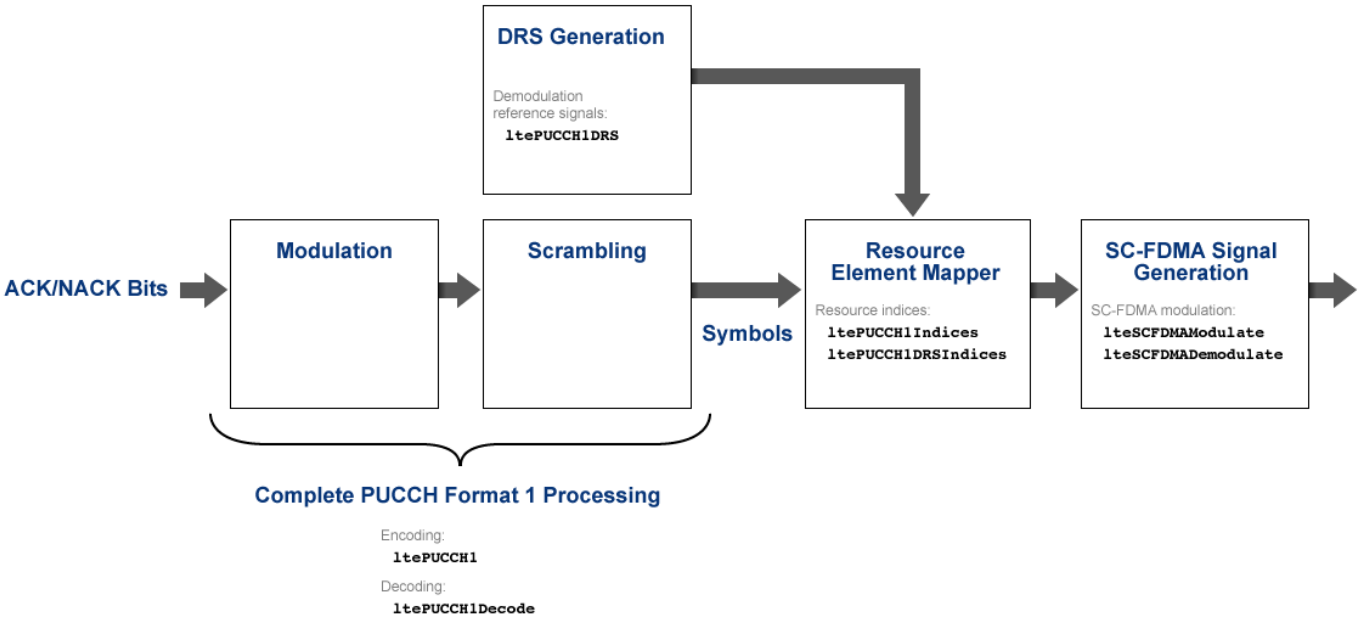
- “Model DCI and PDCCH”

## More About

- “Downlink Control Channel”

# PUCCH Format 1 Processing Functions

The complete physical uplink control channel format 1 process and associated low-level and mid-level PUCCH format 1 functions are shown in the following block diagram.




- Demodulation reference signal (DRS) generation
  - Demodulation reference signals — `ltePUCCH1DRS`
- Resource element mapper
  - Resource indices — `ltePUCCH1Indices`
  - DRS resource indices — `ltePUCCH1DRSIndices`
- SC-FDMA signal generation
  - SC-FDMA modulation — `lteSCFDMAModulate`
  - SC-FDMA demodulation — `lteSCFDMADemodulate`
- Complete PUCCH format 1 processing
  - Encoding — `ltePUCCH1`
  - Decoding — `ltePUCCH1Decode`

## See Also

## Related Examples

- “Model PUCCH Format 1”
- “PUCCH1a ACK Missed Detection Probability Conformance Test”

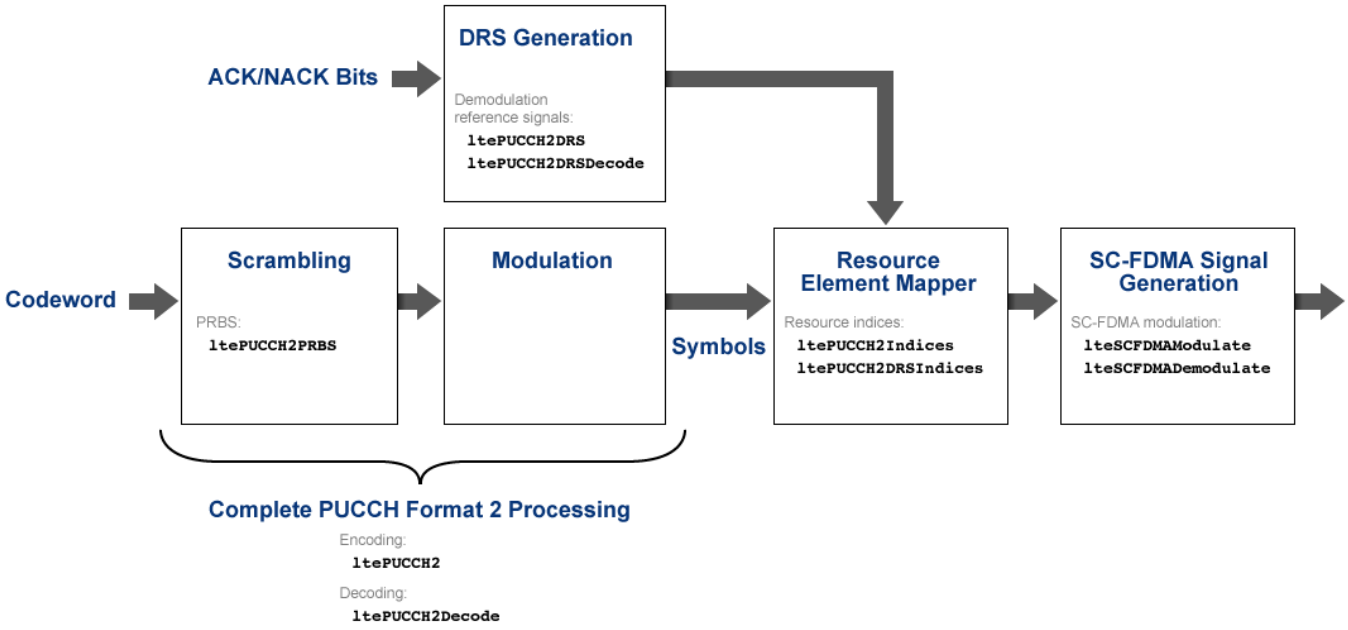
- “PUCCH1a Multi User ACK Missed Detection Probability Conformance Test”

## More About

- “Uplink Control Channel Format 1”

# PUCCH Format 2 Processing Functions

The complete physical uplink control channel format 2 process and associated low-level and mid-level PUCCH format 2 functions are shown in the following block diagram.




- Scrambling
  - Pseudo-random binary sequence (PRBS) — `ltePUCCH2PRBS`
- Demodulation reference signal (DRS) generation
  - Demodulation reference signals — `ltePUCCH2DRS`
  - Demodulation reference signal decoding — `ltePUCCH2DRSDecode`
- Resource element mapper
  - Resource indices — `ltePUCCH2Indices`
  - DRS resource indices — `ltePUCCH2DRSIndices`
- SC-FDMA signal generation
  - SC-FDMA modulation — `lteSCFDMAModulate`
  - SC-FDMA demodulation — `lteSCFDMADemodulate`
- Complete PUCCH format 2 processing
  - Encoding — `ltePUCCH2`
  - Decoding — `ltePUCCH2Decode`

## See Also

## Related Examples

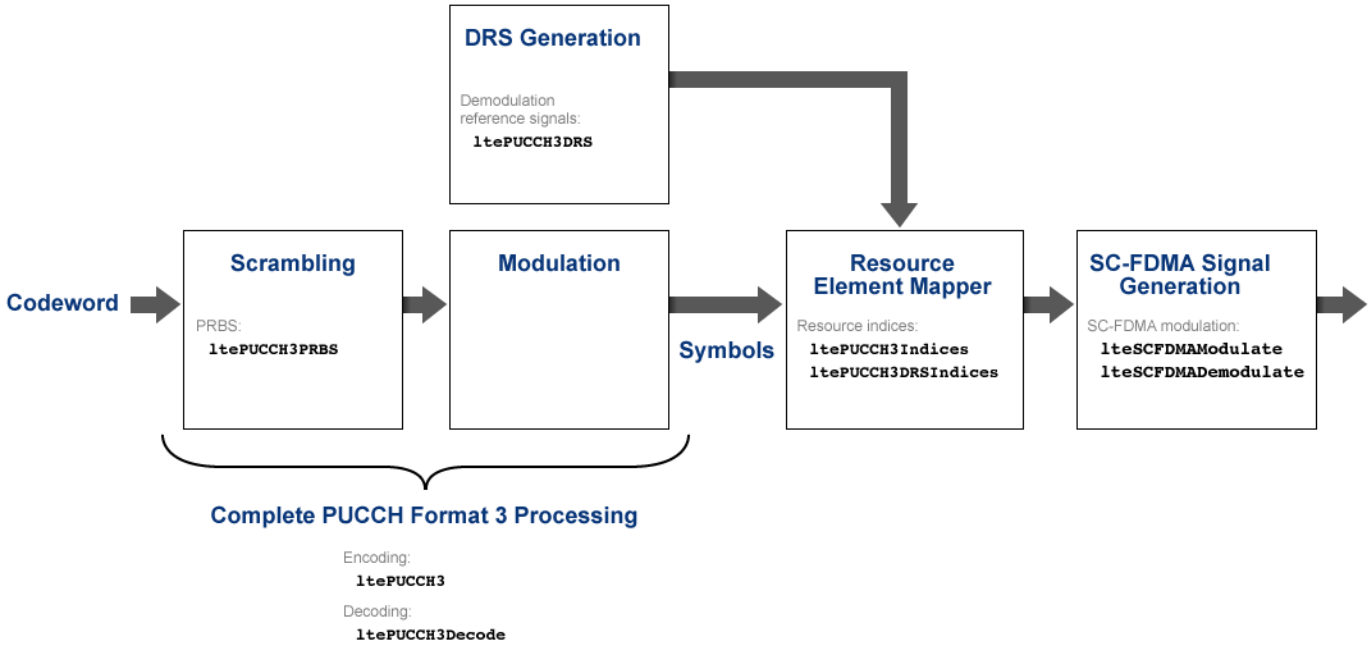
- “Model PUCCH Format 2”

## More About

- “Uplink Control Channel Format 2”

# PUCCH Format 3 Processing Functions

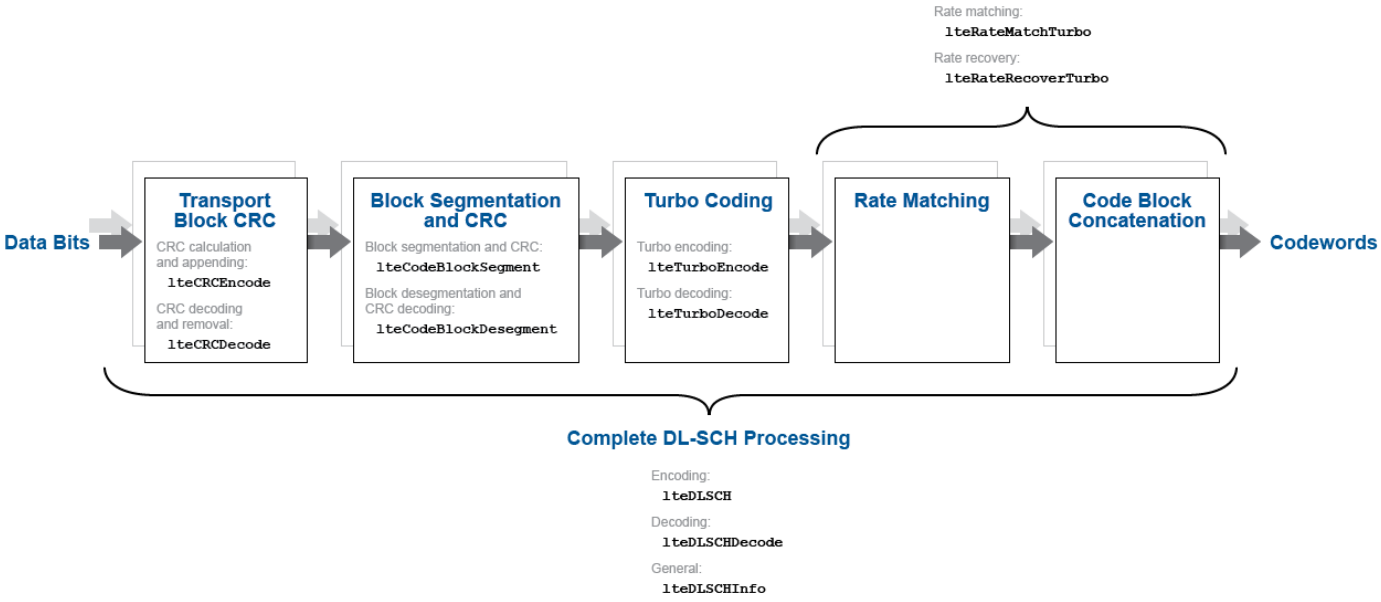
The complete physical uplink control channel format 3 process and associated low-level and mid-level PUCCH format 3 functions are shown in the following block diagram.




- Scrambling
  - Pseudo-random binary sequence (PRBS) — `ltePUCCH3PRBS`
- Demodulation reference signal (DRS) generation
  - Demodulation reference signals — `ltePUCCH3DRS`
- Resource element mapper
  - Resource indices — `ltePUCCH3Indices`
  - DRS resource indices — `ltePUCCH3DRSIndices`
- SC-FDMA signal generation
  - SC-FDMA modulation — `lteSCFDMAModulate`
  - SC-FDMA demodulation — `lteSCFDMADemodulate`
- Complete PUCCH format 3 processing
  - Encoding — `ltePUCCH3`
  - Decoding — `ltePUCCH3Decode`

# DL-SCH Processing Functions

The complete downlink shared channel process and associated low-level and mid-level DL-SCH functions are shown in the following block diagram.




- Transport block cyclic redundancy check (CRC)
  - CRC calculation and appending — `lteCRCEncode`
  - CRC decoding and removal — `lteCRCDecode`
- Block segmentation and CRC
  - Block segmentation and CRC attachment — `lteCodeBlockSegment`
  - Block desegmentation and CRC decoding — `lteCodeBlockDesegment`
- Turbo encoding and decoding
  - `lteTurboEncode`
  - `lteTurboDecode`
- Rate matching and recovery
  - Rate matching — `lteRateMatchTurbo`
  - Rate recovery — `lteRateRecoverTurbo`
- Complete DL-SCH processing
  - Encoding — `lteDLSCH`
  - Decoding — `lteDLSCHDecode`
  - General — `lteDLSCHInfo`

## See Also

## Related Examples

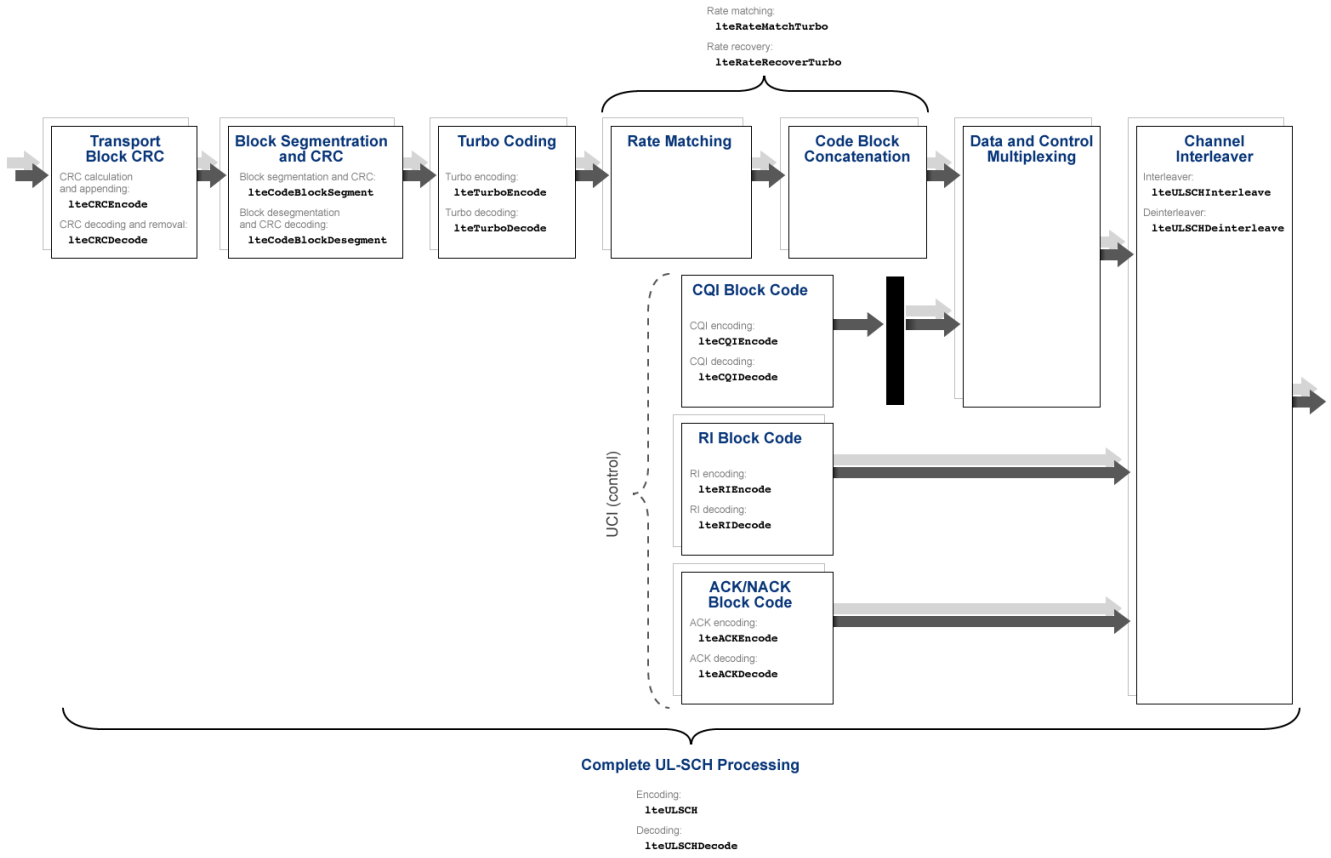
- “Model DL-SCH and PDSCH”
- “DL-SCH HARQ Modeling”

## More About

- “Downlink Shared Channel”

# UL-SCH Processing Functions

The complete uplink shared channel process and associated low-level and mid-level UL-SCH functions are shown in the following block diagram.




- Transport block cyclic redundancy check (CRC)
  - CRC calculation and appending — `lteCRCEncode`
  - CRC decoding and removal — `lteCRCDecode`
- Block segmentation and CRC
  - Block segmentation and CRC attachment — `lteCodeBlockSegment`
  - Block desegmentation and CRC decoding — `lteCodeBlockDesegment`
- Turbo encoding and decoding
  - `lteTurboEncode`
  - `lteTurboDecode`
- Rate matching and recovery
  - Rate matching — `lteRateMatchTurbo`
  - Rate recovery — `lteRateRecoverTurbo`

- Uplink control information (UCI)
  - Channel quality information (CQI) block code
    - CQI encoding — `lteCQIEncode`
    - CQI decoding — `lteCQIDecode`
  - Rank indicator (RI) block code
    - RI encoding — `lteRIEncode`
    - RI decoding — `lteRIDecode`
  - Acknowledgement (ACK) or Negative acknowledgement (NACK) block code
    - ACK encoding — `lteACKEncode`
    - ACK decoding — `lteACKDecode`
- Channel interleaver
  - Interleaver — `lteULSCHInterleave`
  - Deinterleaver — `lteULSCHDeinterleave`
- Complete UL-SCH processing
  - Encoding — `lteULSCH`
  - Decoding — `lteULSCHDecode`

## See Also

## Related Examples

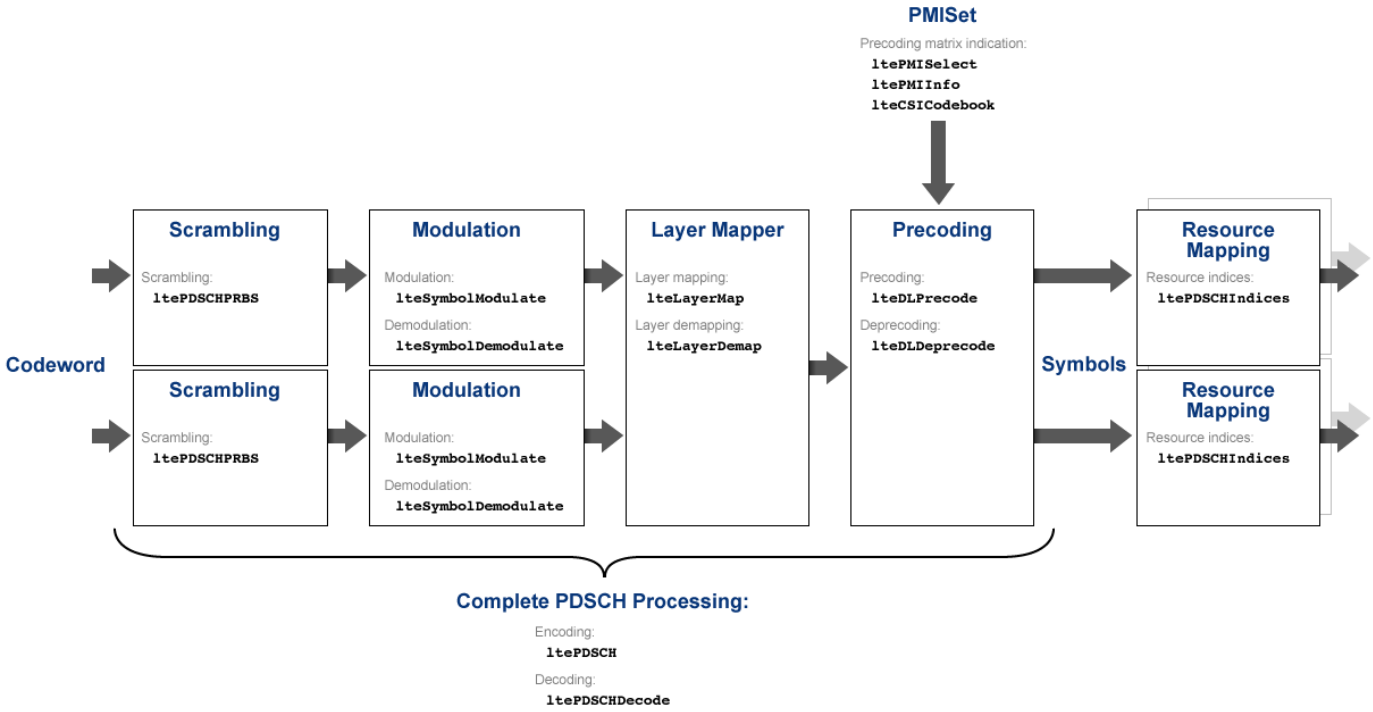
- “Model UL-SCH and PUSCH”

## More About

- “Uplink Shared Channel”

# PDSCH Processing Functions

The complete physical downlink shared channel process and associated low-level and mid-level PDSCH functions are shown in the following block diagram.




- Scrambling — `ltePDSCHPRBS`
- Symbol modulation and demodulation
  - Modulation — `lteSymbolModulate`
  - Demodulation — `lteSymbolDemodulate`
- Layer mapping
  - Layer mapping — `lteLayerMap`
  - Layer demapping — `lteLayerDemap`
- Precoding and deprecoding
  - Precoding — `lteDLPrecode`
  - Deprecoding — `lteDLDeprecode`
- Downlink precoding matrix indication (PMI)
  - `ltePMISelect`
  - `ltePMIInfo`
  - `lteCSICodebook`
- Resource mapping

    - Resource indices — `ltePDSCHIndices`
- Complete PDSCH processing
    - Encoding — `ltePDSCH`
    - Decoding — `ltePDSCHDecode`

## See Also

## Related Examples

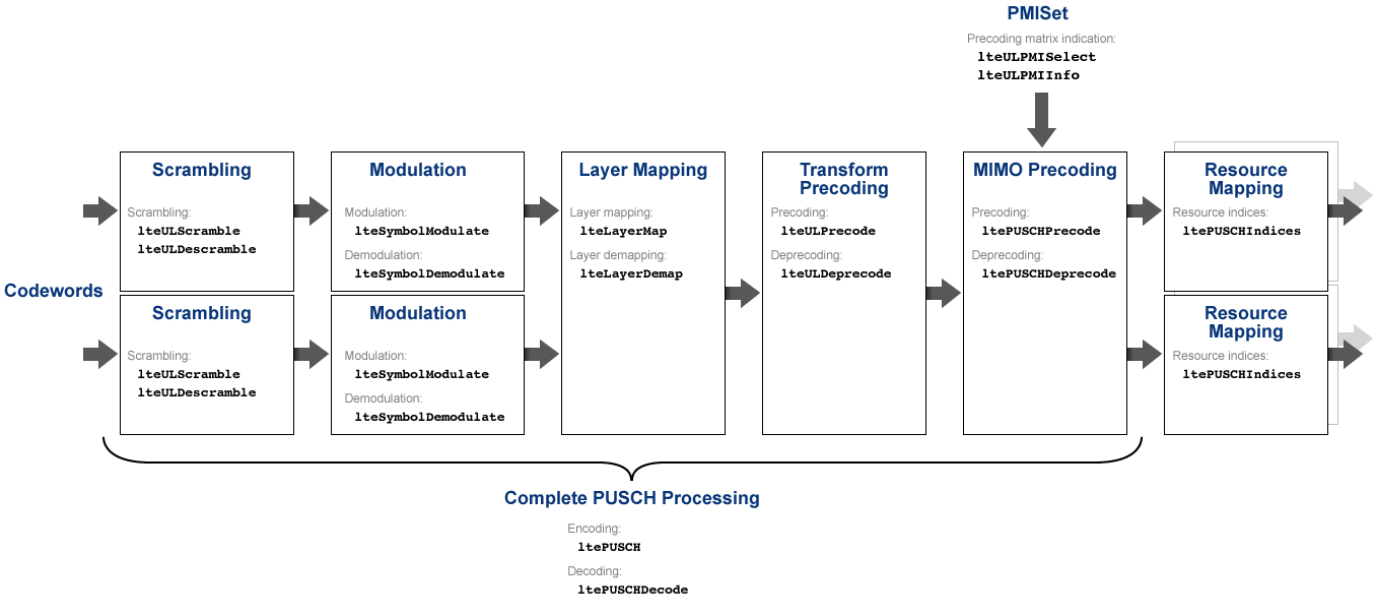
- “Model DL-SCH and PDSCH”
- “PDSCH Bit Error Rate Curve Generation”

## More About

- “Downlink Shared Channel”

# PUSCH Processing Functions

The complete physical uplink shared channel process and associated low-level and mid-level PUSCH functions are shown in the following block diagram.




- Scrambling
  - Scrambling — `lteULScramble`
  - Descrambling — `lteULDescramble`
- Symbol modulation and demodulation
  - Modulation — `lteSymbolModulate`
  - Demodulation — `lteSymbolDemodulate`
- Layer mapping
  - Layer mapping — `lteLayerMap`
  - Layer demapping — `lteLayerDemap`
- Transform precoding and deprecoding
  - Transform precoding — `lteULPrecode`
  - Transform deprecoding — `lteULDeprecode`
- Multiple-input multiple-output (MIMO) precoding and deprecoding
  - MIMO precoding — `ltePUSCHPrecode`
  - MIMO deprecoding — `ltePUSCHDeprecode`
- Uplink (UL) precoding matrix indication (PMI)
  - `lteULPMISelect`

    - `lteULPMIInfo`
- Resource mapping
    - Resource indices — `ltePUSCHIndices`
- Complete PUSCH processing
    - Encoding — `ltePUSCH`
    - Decoding — `ltePUSCHDecode`

## See Also

## Related Examples

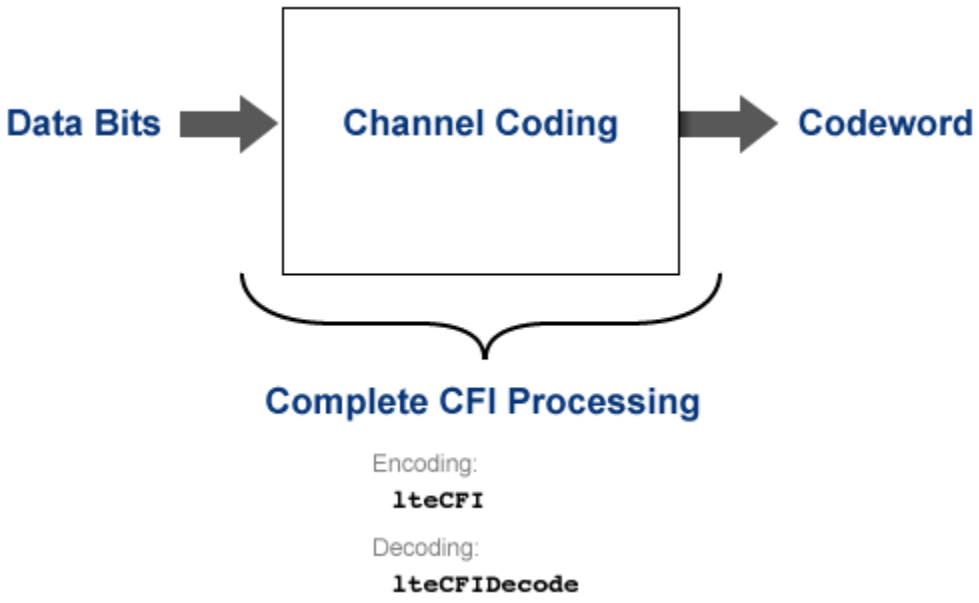
- “Model UL-SCH and PUSCH”

## More About

- “Uplink Shared Channel”

# CFI Processing Functions

The complete control format information process and associated low-level and mid-level CFI functions are shown in the following block diagram.




- Complete CFI processing
  - Encoding — `lteCFI`
  - Decoding — `lteCFIDecode`

## See Also

## Related Examples

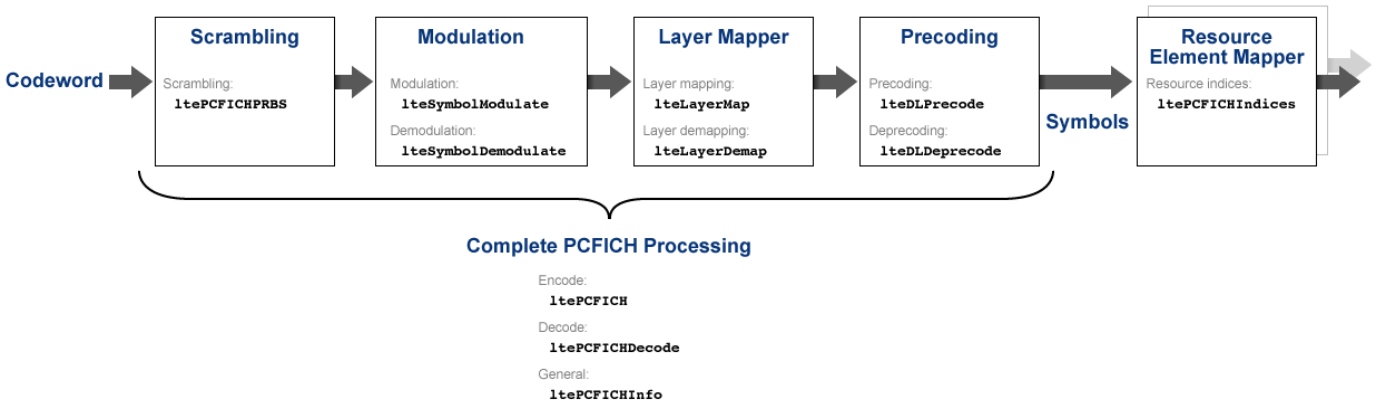
- “Model CFI and PCFICH”

## More About

- “Control Format Indicator (CFI) Channel”

# PCFICH Processing Functions

The complete physical downlink control format indicator channel process and associated low-level and mid-level PCFICH functions are shown in the following block diagram.




- Scrambling — `ltePCFICHPRBS`
- Symbol modulation and demodulation
  - Modulation — `lteSymbolModulate`
  - Demodulation — `lteSymbolDemodulate`
- Layer mapper and demapper
  - Layer mapping — `lteLayerMap`
  - Layer demapping — `lteLayerDemap`
- Precoding and deprecoding
  - Precoding — `lteDLPrecode`
  - Deprecoding — `lteDLDeprecode`
- Resource element mapper
  - Resource indices — `ltePCFICHIndices`
- Complete PCFICH processing
  - Encoding — `ltePCFICH`
  - Decoding — `ltePCFICHDecode`
  - General — `ltePCFICHInfo`

## See Also

## Related Examples

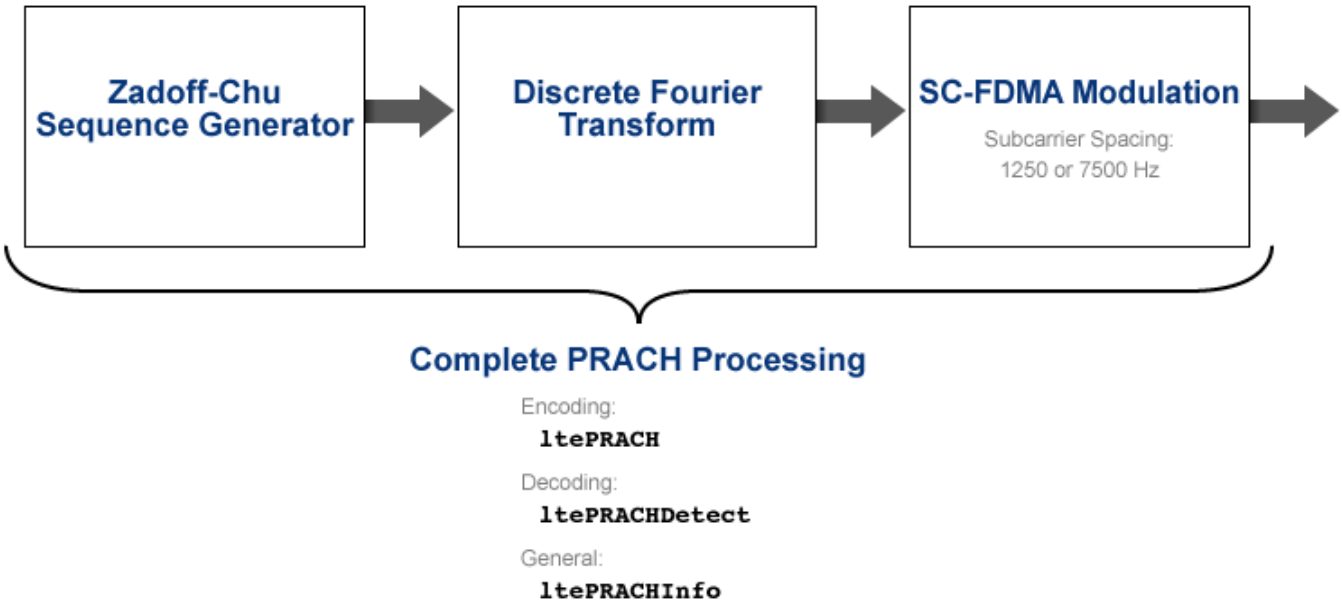
- “Model CFI and PCFICH”

## More About

- “Control Format Indicator (CFI) Channel”

# PRACH Processing Functions

The random access channel process and associated PRACH functions are shown in the following block diagram.




- Complete PRACH processing
  - Encoding — `ltePRACH`
  - Decoding — `ltePRACHDetect`
  - General — `ltePRACHInfo`

## See Also

## Related Examples

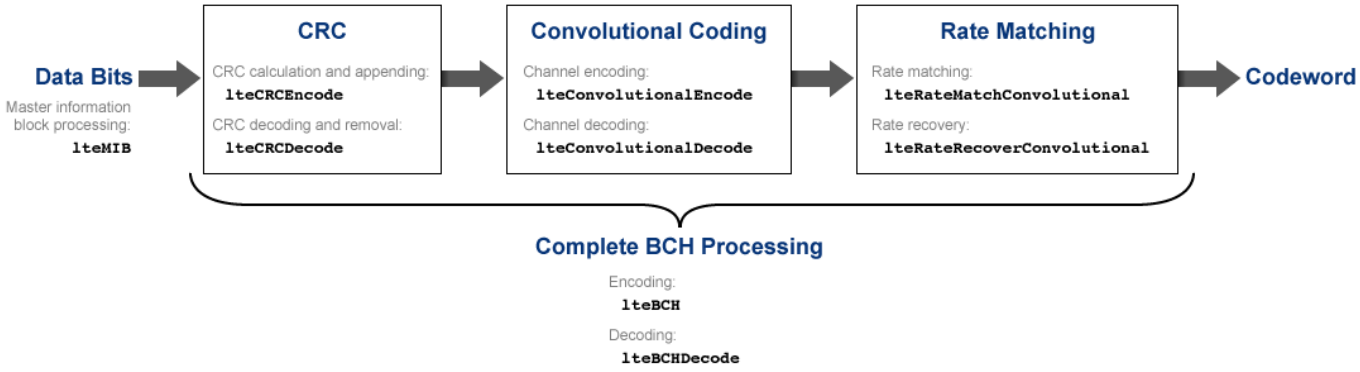
- “PRACH False Alarm Probability Conformance Test”
- “PRACH Detection Conformance Test”

## More About

- “Random Access Channel”

# BCH Processing Functions

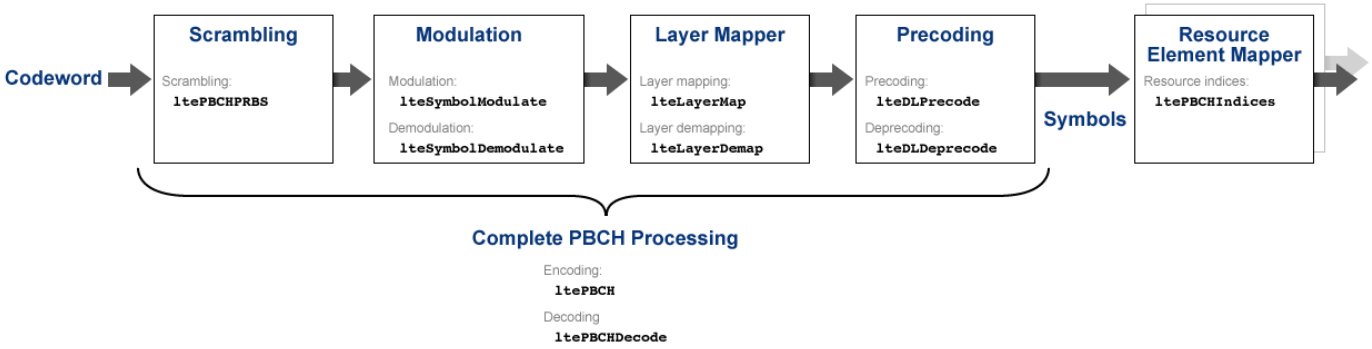
The complete broadcast channel process and associated low-level and mid-level BCH functions are shown in the following block diagram.




- Cyclic redundancy check (CRC)
  - CRC calculation and appending — `lteCRCEncode`
  - CRC decoding and removal — `lteCRCDecode`
- Convolutional channel encoding and decoding
  - `lteConvolutionalEncode`
  - `lteConvolutionalDecode`
- Rate matching and recovery
  - Rate matching — `lteRateMatchConvolutional`
  - Rate recovery — `lteRateRecoverConvolutional`
- MIB processing — `lteMIB`
- Complete BCH processing
  - Encoding — `lteBCH`
  - Decoding — `lteBCHDecode`

# PBCH Processing Functions

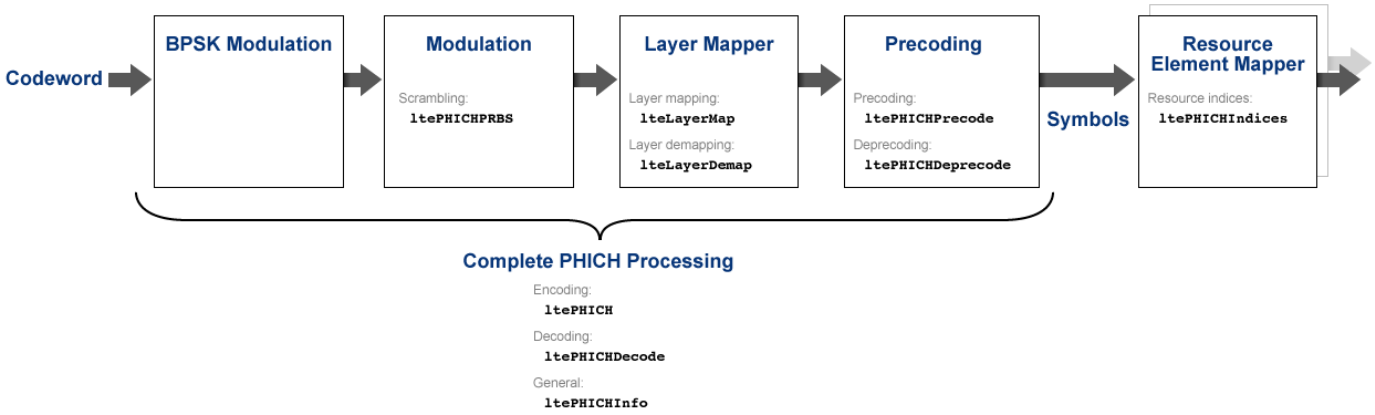
The complete physical broadcast channel process and associated low-level and mid-level PBCH functions are shown in the following block diagram.




- Scrambling — `ltePBCHPRBS`
- Symbol modulation and demodulation
  - Modulation — `lteSymbolModulate`
  - Demodulation — `lteSymbolDemodulate`
- Layer mapper and demapper
  - Layer mapping — `lteLayerMap`
  - Layer demapping — `lteLayerDemap`
- Precoding and deprecoding
  - Precoding — `lteDLPrecode`
  - Deprecoding — `lteDLDeprecode`
- Resource element mapper
  - Resource indices — `ltePBCHIndices`
- Complete PBCH processing
  - Encoding — `ltePBCH`
  - Decoding — `ltePBCHDecode`

# PHICH Processing Functions

The complete physical hybrid automatic repeat request (HARQ) indicator channel process and associated low-level and mid-level PHICH functions are shown in the following block diagram.




- Scrambling — `ltePHICHPRBS`
- Layer mapper and demapper
  - Layer mapping — `lteLayerMap`
  - Layer demapping — `lteLayerDemap`
- Precoding and deprecoding
  - Precoding — `ltePHICHPrecode`
  - Deprecoding — `ltePHICHDeprecode`
- Resource element mapper
  - Resource indices — `ltePHICHIndices`
- Complete PBCH processing
  - Encoding — `ltePHICH`
  - Decoding — `ltePHICHDecode`
  - General — `ltePHICHInfo`

## See Also

## Related Examples

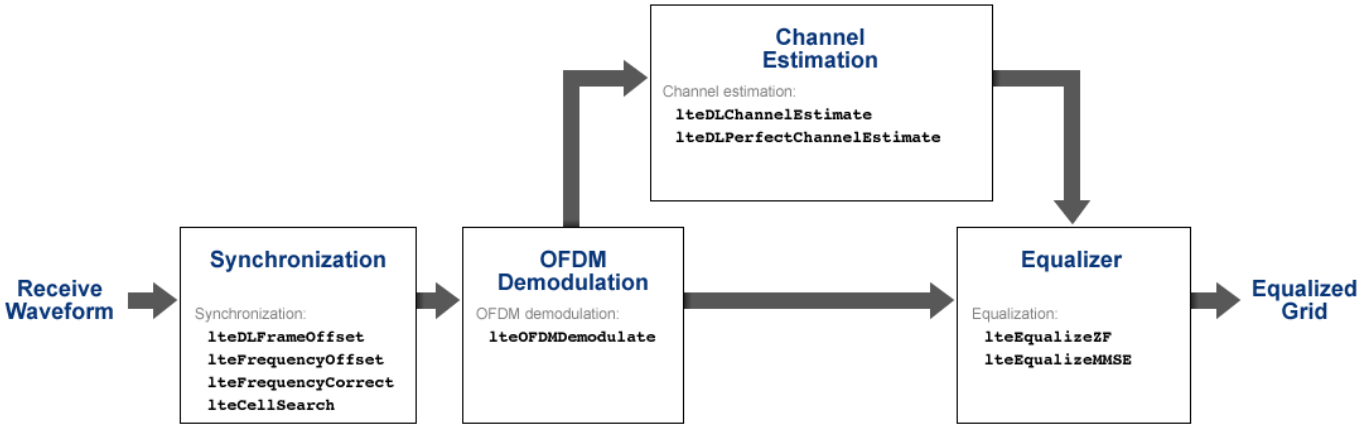
- “Model HARQ Indicator and PHICH”

## More About

- “HARQ Indicator (HI) Channel”

# Downlink Receiver Functions

The complete downlink (DL) receiver process and associated functions are shown in the following block diagram.




- Synchronization
  - `lteDLFrameOffset`
  - `lteFrequencyOffset`
  - `lteFrequencyCorrect`
  - `lteCellSearch`
- OFDM demodulation — `lteOFDMDemodulate`
- Channel estimation
  - `lteDLChannelEstimate`
  - `lteDLPerfectChannelEstimate`
- Equalization
  - `lteEqualizeZF`
  - `lteEqualizeMMSE`

## See Also

## Related Examples

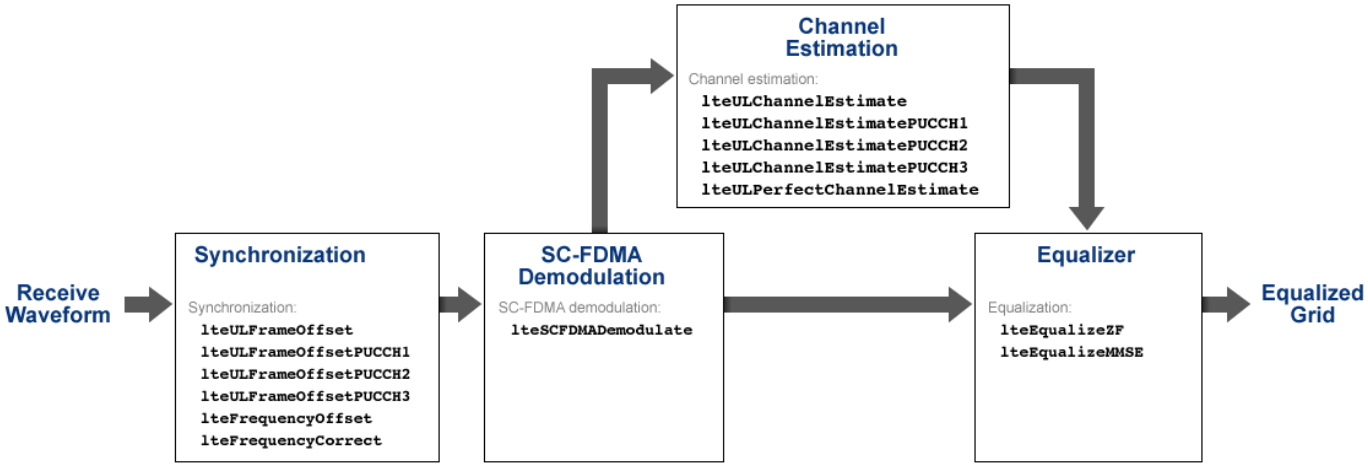
- “LTE Downlink Channel Estimation and Equalization”

## More About

- “Channel Estimation”

# Uplink Receiver Functions

The complete uplink (UL) receiver process and associated functions are shown in the following block diagram.




- Synchronization
  - `lteULFrameOffset`
  - `lteULFrameOffsetPUCCH1`
  - `lteULFrameOffsetPUCCH2`
  - `lteULFrameOffsetPUCCH3`
  - `lteFrequencyOffset`
  - `lteFrequencyCorrect`
- SC-FDMA demodulation — `lteSCFDMADemodulate`
- Channel estimation
  - `lteULChannelEstimate`
  - `lteULChannelEstimatePUCCH1`
  - `lteULChannelEstimatePUCCH2`
  - `lteULChannelEstimatePUCCH3`
  - `lteULPerfectChannelEstimate`
- Equalization
  - `lteEqualizeZF`
  - `lteEqualizeMMSE`

## See Also

## Related Examples

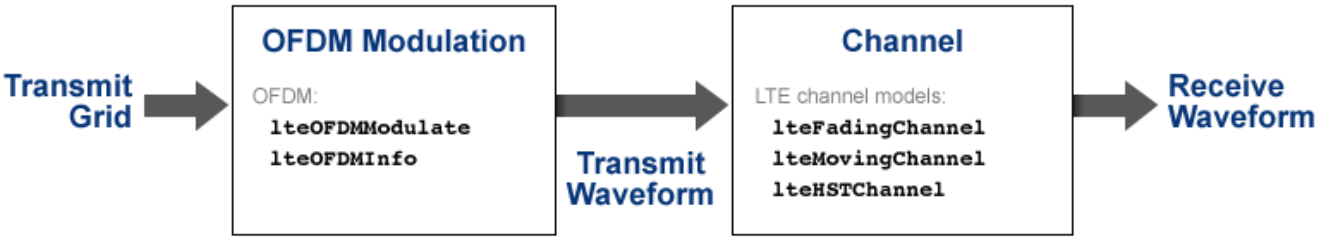
- “PUCCH2 CQI BLER Conformance Test”

## More About

- “Channel Estimation”

# OFDM Modulation and Propagation Channel Models

The orthogonal frequency-division multiplexing (OFDM) modulation process, propagation channel models, and their associated functions are shown in the following block diagram.




- OFDM modulation
  - `lteOFDMModulate`
  - `lteOFDMInfo`
- LTE propagation channel models
  - `lteFadingChannel`
  - `lteMovingChannel`
  - `lteHSTChannel`

## See Also

## Related Examples

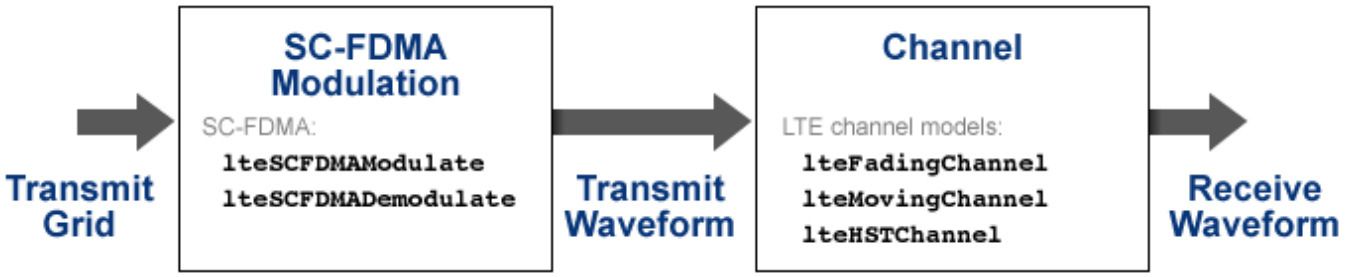
- “Simulate Propagation Channels”
- “Find Channel Impulse Response”

## More About

- “Propagation Channel Models”

# SC-FDMA Modulation and Propagation Channel Models

The single-carrier frequency-division multiple access (SC-FDMA) modulation process, propagation channel models, and their associated functions are shown in the following block diagram.




- SC-FDMA modulation
  - `lteSCFDMAModulate`
  - `lteSCFDMADemodulate`
- LTE propagation channel models
  - `lteFadingChannel`
  - `lteMovingChannel`
  - `lteHSTChannel`

## See Also

## Related Examples

- “Simulate Propagation Channels”
- “Find Channel Impulse Response”

## More About

- “Propagation Channel Models”