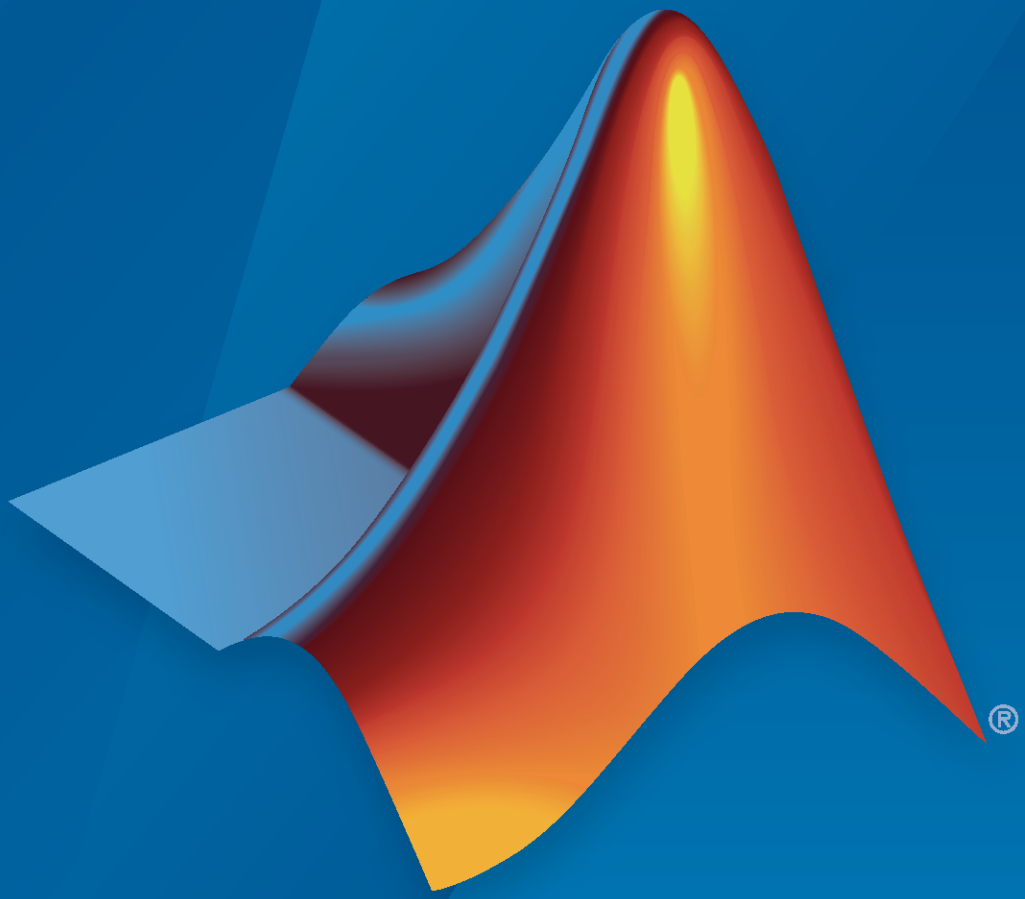
LTE Toolbox™

User's Guide



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™ User's Guide

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

1 LTE Physical Layer Concepts

2 Examples and Demos

3 LTE Physical Layer Examples

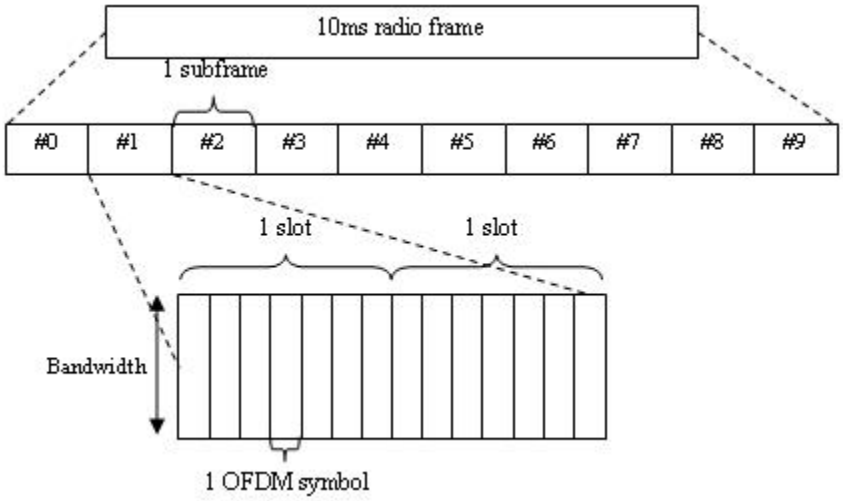
4 UMTS Concepts

LTE Physical Layer Concepts

FDD and TDD Duplexing

In this section...

The LTE Toolbox product can generate and manipulate signals for the duplexing arrangements specified in the LTE standard. In LTE, downlink and uplink transmissions are organized into radio frames of duration 10ms consisting of 10 consecutive subframes, each consisting of a number of consecutive OFDM symbols, as shown in the following figure.



Create Resource Grid for Cyclic Prefix Choice

This example shows how to create a resource grid for either normal or extended cyclic prefix. The number of OFDM symbols within one subframe is either 14 for normal cyclic prefix, or 12 for extended cyclic prefix.

Create the cell-wide settings structure.

```
enb.CyclicPrefix = 'Normal';
enb.NDLRB = 9;
enb.CellRefP = 1;
```

Get subframe resource grid dimensions.

```
dims = lteDLResourceGridSize(enb)
```

```
dims = 1×3

   108    14     1
```

Switch to extended cyclic prefix.

```
enb.CyclicPrefix = 'Extended';
dims = lteDLResourceGridSize(enb)
```

```
dims = 1×3

   108    12     1
```

The second dimension of the output of `lteDLResourceGridSize` is the number of symbols in the subframe.

Create Frame with CellRS in Subframes

This example shows how to create a frame containing the cell-specific reference signals (CellRS) in each subframe. The radio frame is represented in the LTE Toolbox™ product by the use of a succession of 10 cell-wide settings structures with the `NSubframe` field set from 0 through 9 in each case.

Initialize Cell-wide Setting Structure and Create Empty Resource Grid

Modify the `NDLRB` parameter to set the number of resource blocks. Modify `CellRefP` to set one transmit antenna port. Modify `NCellID` to set the cell ID. Specify normal cyclic prefix and antenna port zero.

```
enb.NDLRB = 6;
enb.CellRefP = 1;
enb.NCellID = 1;
enb.CyclicPrefix = 'Normal';
antenna = 0;
```

Create an empty resource grid to be populated with subframes.

```
txGrid = [];
```

Populate Resource Grid For Each Subframe

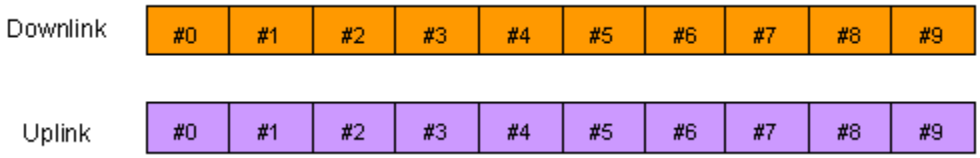
- Create an empty resource grid for each subframe and set the current subframe number.
- Generate cell-specific reference signal symbols and indices.
- Map the cell-specific reference signal to the grid and append the subframe to the grid to be transmitted.

```
for sf = 0:9
    subframe = lteDLResourceGrid(enb);
    enb.NSubframe = sf;
    cellRsSym = lteCellRS(enb,antenna);
    cellRsInd = lteCellRSIndices(enb,antenna);
    subframe(cellRsInd) = cellRsSym;
```

```
    txGrid = [txGrid subframe];
end
```

Frame Structure Type 1: FDD

In the FDD duplexing mode, all 10 subframes within a radio frame contain downlink or uplink subframes depending on the link direction.



The uplink and downlink transmitters have separate bandwidths in which to make their transmissions. Therefore, each can transmit at all times.

Within the LTE Toolbox product, you can create signals or indices for FDD duplexing mode simply by setting the `NSubframe` field of the cell-wide settings structure to the appropriate subframe number. Functions whose behavior depends on the duplexing mode have the `DuplexMode` field, which you can set to `'FDD'` or `'TDD'`. If you do not specify this field, `'FDD'` is used by default.

Generate PSS Indices for FDD Mode

This example shows how to generate the primary synchronization signal (PSS) indices in subframe 0 using the FDD duplexing mode.

First, create the cell-wide settings structure.

```
enb.CyclicPrefix = 'Normal';
enb.NDLRB = 9;
enb.NCellID = 1;
enb.NSubframe = 0;
enb.DuplexMode = 'FDD';
```

Next, create PSS indices, display size and the first five indices in subframe 0.

```
ind = ltePSSIndices(enb);
size(ind)
```

```
ans = 1×2

    62     1
```

```
ind(1:5)
```

```
ans = 5x1 uint32 column vector

   672
   673
   674
   675
```

```
    676
```

If the same call is made for subframe 1 instead, then the result is an empty matrix.

```
enb.NSubframe = 1;
```

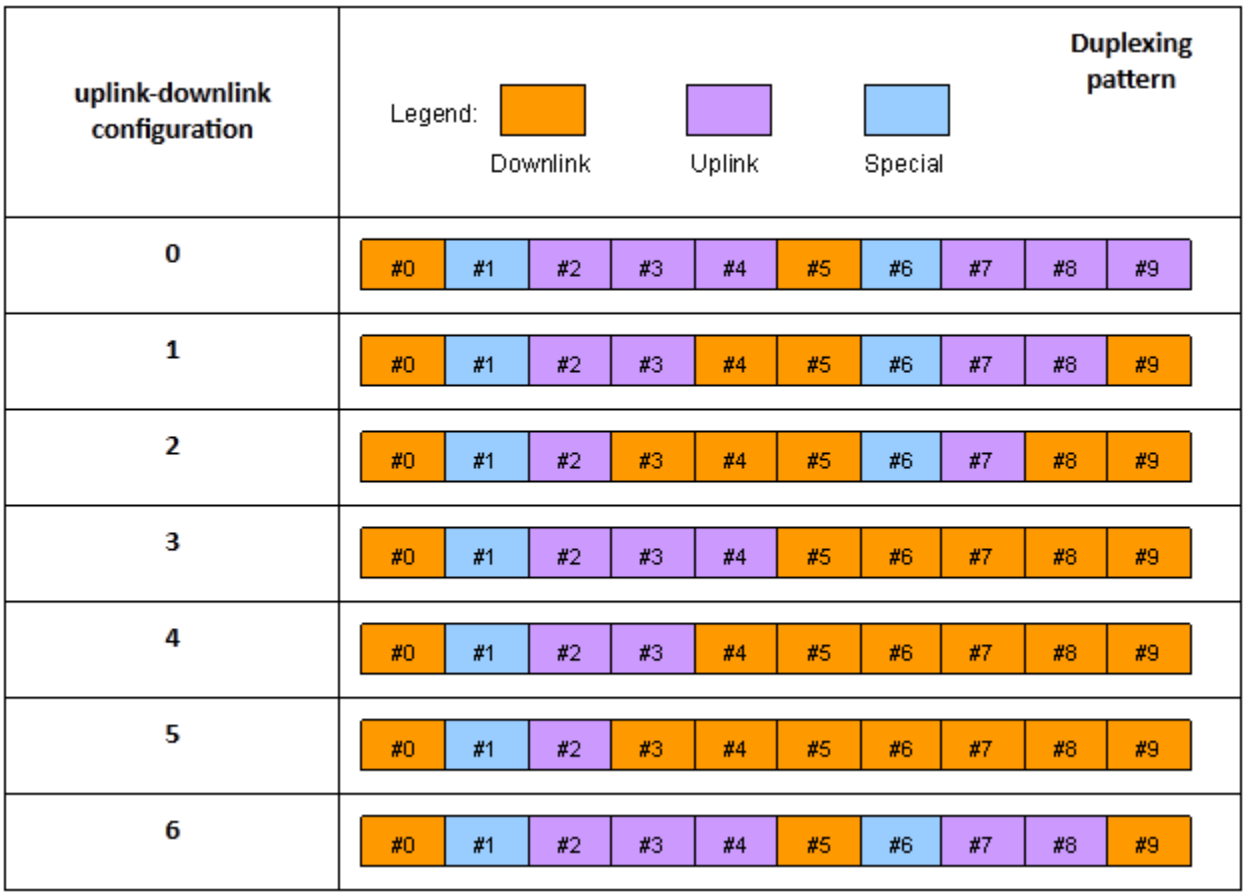
An empty matrix indicates that the PSS is not present in subframe 1. By calling the functions for indices and values for subframes 0 through to 9 by setting the `NSubframe` field, the appropriate transmissions across a radio frame can be formed.

```
ind = ltePSSIndices(enb)
```

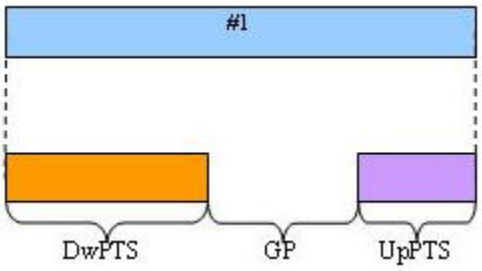
```
ind =

  0x1 empty uint32 column vector
```

Frame Structure Type 2: TDD

In the TDD duplexing mode, a single bandwidth is shared between uplink and downlink, with the sharing being performed by allotting different periods of time to uplink and downlink. In LTE, there are 7 different patterns of uplink-downlink switching, termed uplink-downlink configurations 0 through 6, as shown in the following figure.



The special subframe (subframe 1 in every uplink-downlink configuration, and subframe 6 in uplink-downlink configurations 0, 1, 2 and 6) contains a portion of downlink transmission at the start of the subframe (the Downlink Pilot Time Slot, DwPTS), a portion of unused symbols in the middle of the subframe (the Guard Period) and a portion of uplink transmission at the end of the subframe (the Uplink Pilot Time Slot, UpPTS), as shown in the following figure.



The lengths of DwPTS, GP, and UpPTS can take one of 10 combinations of values, termed special subframe configurations 0 through 9. The LTE standard, TS 36.211 Table 4.2-1, specifies the lengths in terms of the fundamental period of the OFDM modulation, but the lengths can be interpreted in terms of OFDM symbols as shown in the following table.

Configuration of special subframe (lengths of DwPTS/GP/UpPTS)						
Special Subframe Configuration	Normal cyclic prefix in downlink			Extended cyclic prefix in downlink		
	DwPTS	UpPTS		DwPTS	UpPTS	
		Normal cyclic prefix in uplink	Extended cyclic prefix in uplink		Normal cyclic prefix in uplink	Extended cyclic prefix in uplink
0	3	1	1	3	1	1
1	9			8		
2	10			9		
3	11			10		
4	12			3	2	2
5	3	2	2	8		
6	9			9		
7	10			5		
8	11			-	-	-
9	6			-	-	-

Thus, in effect, the special subframe is both a downlink subframe and an uplink subframe, with some restriction placed on the number of OFDM symbols that are occupied in each case.

To specify TDD operation, in the cell-wide settings structure, set the optional `DuplexMode` field to `'TDD'`. When you use this setting, functions that require `DuplexMode` also require that you specify the uplink-downlink configuration (0,...,6) in the `TDDConfig` field, the subframe number in the `NSubframe` field, and the special subframe configuration (0,...,9) in the `SSC` field.

Generate CellRS Indices for TDD Mode

This example shows how to create the subscripts for the positions of the cell-specific reference signal (CellRS) for antenna port 0 in subframe 6 for uplink-downlink configuration 2 and special subframe configuration 4 with extended cyclic prefix.

First, create the parameter structure.

```
enb.NDLRB        = 9;
enb.NCellID      = 1;
enb.DuplexMode   = 'TDD';
enb.NSubframe    = 6;
enb.TDDConfig    = 2;
enb.SSC          = 4;
enb.CyclicPrefix = 'Extended';
```

Next, create the cell-specific RS indices.

```
sub = lteCellRSIndices(enb,0,'sub')
```

```
sub = 18x3 uint32 matrix

    2    1    1
    8    1    1
   14    1    1
   20    1    1
   26    1    1
   32    1    1
   38    1    1
   44    1    1
   50    1    1
   56    1    1
    ⋮
```

The second column, which gives the OFDM symbol number (1-based) within the subframe, has values of 1 indicating that only the 1st OFDM symbol will contain cell-specific reference signals in this case. This is because the chosen subframe is a special subframe with DwPTS of length 3 and therefore the other cell-specific reference signal elements (in OFDM symbols 4, 7 and 10) which would be present in full downlink subframes are not generated.

To confirm this theory, change the duplex mode to FDD.

```
enb.DuplexMode   = 'FDD';
sub = lteCellRSIndices(enb,0,'sub');
unique(sub(:,2))
```

```
ans = 4x1 uint32 column vector

    1
    4
    7
   10
```

In this case, the switch over to FDD means that the now irrelevant fields, `TDDConfig` and `SSC`, are ignored.

Dimension Information Related to Duplexing

This example shows how to extract information from a parameter structure. To facilitate working with different duplexing arrangements, the LTE Toolbox™ product provides the `lteDuplexingInfo` information function. This function takes a cell-wide settings structure containing the fields mentioned in the preceding sections. It returns a structure that indicates the type of the current subframe and the number of symbols in the current subframe.

First, create a parameter structure.

```
enb.NDLRB        = 9;
enb.NCellID      = 1;
enb.DuplexMode   = 'TDD';
enb.NSubframe    = 6;
enb.TDDConfig    = 2;
enb.SSC          = 4;
enb.CyclicPrefix = 'Extended';
```

Next, extract the dimension information.

```
lteDuplexingInfo(enb)
```

```
ans = struct with fields:
     SubframeType: 'Special'
         NSymbols: 12
       NSymbolsDL: 3
    NSymbolsGuard: 7
       NSymbolsUL: 2
```

Finally, change the `NSubframe` property and extract the dimension information again.

```
enb.NSubframe    = 0;
lteDuplexingInfo(enb)
```

```
ans = struct with fields:
     SubframeType: 'Downlink'
         NSymbols: 12
       NSymbolsDL: 12
    NSymbolsGuard: 0
       NSymbolsUL: 0
```

This function provides direct access to the uplink-downlink configuration patterns via the `SubframeType` field and special subframe DwPTS, GP and UpPTS lengths via the `NSymbolsDL`, `NSymbolsGuard`, and `NSymbolsUL` fields.

Synchronization Signals (PSS and SSS)

In LTE, there are two downlink synchronization signals which are used by the UE to obtain the cell identity and frame timing.

- Primary synchronization signal (PSS)
- Secondary synchronization signal (SSS)

The division into two signals is aimed to reduce the complexity of the cell search process.

Cell Identity Arrangement

The physical cell identity, N_{ID}^{cell}, is defined by the equation:

$$N_{ID}^{CELL} = 3N_{ID}^{(1)} + N_{ID}^{(2)}$$

- $N_{ID}^{(1)}$ is the physical layer cell identity group (0 to 167).
- $N_{ID}^{(2)}$ is the identity within the group (0 to 2).

This arrangement creates 504 unique physical cell identities.

Synchronization Signals and Determining Cell Identity

The primary synchronization signal (PSS) is linked to the cell identity within the group ($N_{ID}^{(2)}$). The secondary synchronization signal (SSS) is linked to the cell identity group ($N_{ID}^{(1)}$) and the cell identity within the group ($N_{ID}^{(2)}$).

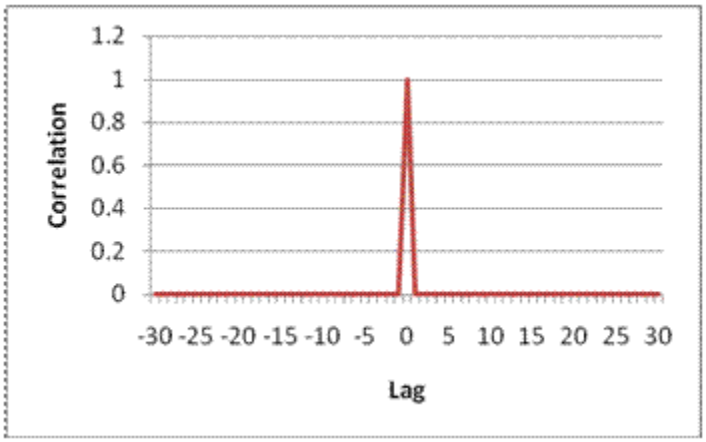
You can obtain $N_{ID}^{(2)}$ by successfully demodulating the PSS. The SSS can then be demodulated and combined with knowledge of $N_{ID}^{(2)}$ to obtain $N_{ID}^{(1)}$. Once you establish the values of $N_{ID}^{(1)}$ and $N_{ID}^{(2)}$, you can determine the cell identity (N_{ID}^{cell}).

Primary Synchronization Signal (PSS)

The primary synchronization signal (PSS) is based on a frequency-domain Zadoff-Chu sequence.

Zadoff-Chu Sequences

Zadoff-Chu sequences are a construction of Frank-Zadoff sequences defined by D. C. Chu in [1]. These codes have the useful property of having zero cyclic autocorrelation at all nonzero lags. When used as a synchronization code, the correlation between the ideal sequence and a received sequence is greatest when the lag is zero. When there is any lag between the two sequences, the correlation is zero. This property is illustrated in this figure.



PSS Generation

The PSS is a sequence of complex symbols, 62 symbols long.

The sequence $d_u(n)$ used for the PSS is generated according to these equations:

$$d_u(n) = e^{-j\frac{\pi un(n+1)}{63}}, \text{ for } n = 0, 1, \ldots, 30$$

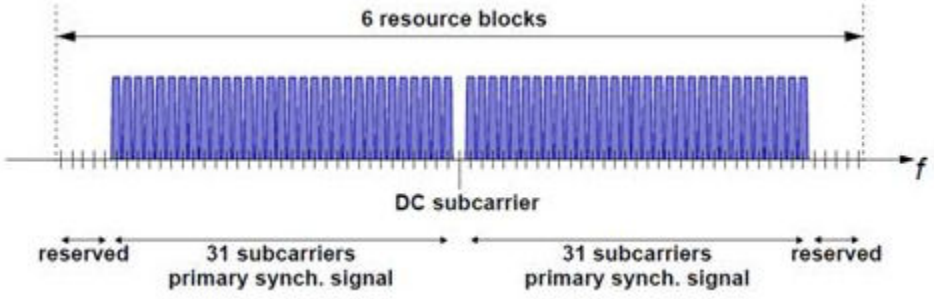
$$d_u(n) = e^{-j\frac{\pi u(n+1)(n+2)}{63}}, \text{ for } n = 31, 32, \ldots, 61$$

In the preceding equation, u is the Zadoff-Chu root sequence index and depends on the cell identity within the group $N_{ID}^{(2)}$.

$N_{ID}^{(2)}$	**Root index *u***
0	25
1	29
2	34

Mapping of the PSS

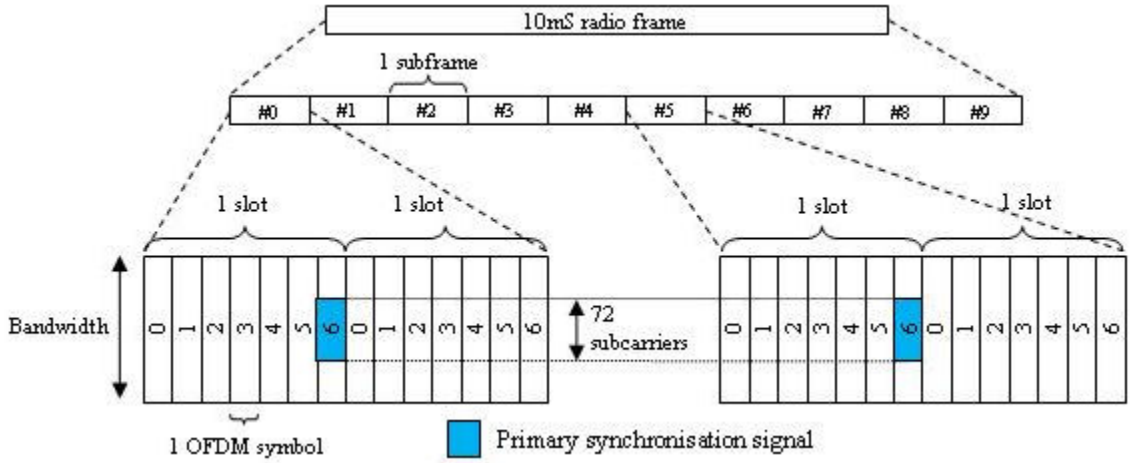
The PSS is mapped into the first 31 subcarriers either side of the DC subcarrier. Therefore, the PSS uses six resource blocks with five reserved subcarriers each side, as shown in this figure.



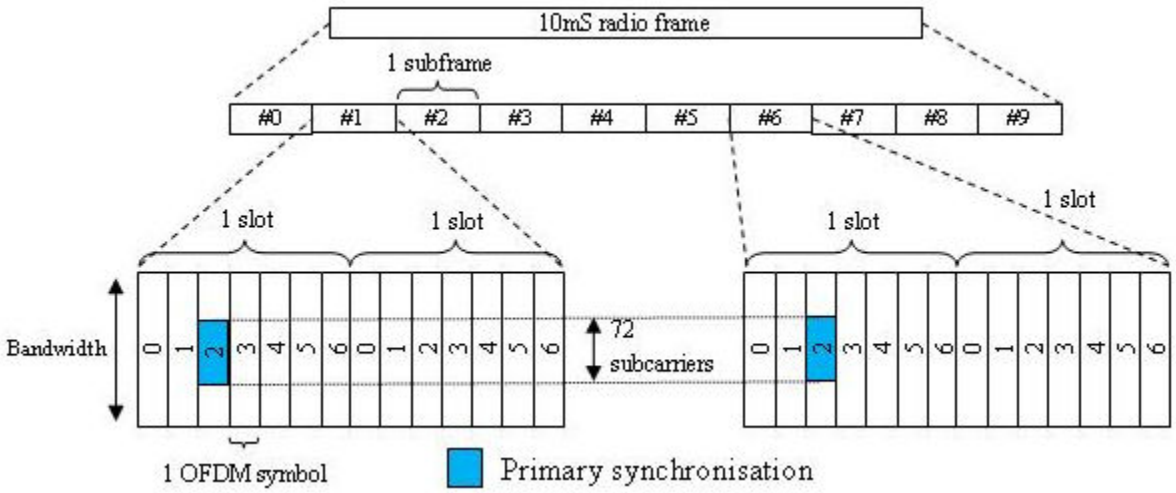
As the DC subcarrier contains no information in LTE this corresponds to mapping onto the middle 62 subcarriers within an OFDM symbol in a resource grid. $d(n)$ is mapped from lowest subcarrier to

highest subcarrier. The PSS is mapped to different OFDM symbols depending on which frame type is used. Frame type 1 is frequency division duplex (FDD), and frame type 2 is time division duplex (TDD).

- **FDD** — The PSS is mapped to the last OFDM symbol in slots 0 and 10, as shown in this figure.



- **TDD** — The PSS is mapped to the third OFDM symbol in subframes 1 and 6, as shown in this figure.



Secondary Synchronization Signal (SSS)

The secondary synchronization signal (SSS) is based on maximum length sequences (*m*-sequences).

M-Sequence Definition

An *m*-sequence is a pseudorandom binary sequence which can be created by cycling through every possible state of a shift register of length *m*, resulting in a sequence of length 2^m–1. Three *m*-sequences, each of length 31, are used to generate the synchronization signals denoted $\tilde{s}$, $\tilde{c}$ and $\tilde{z}$.

SSS Generation

Two binary sequences, each of length 31, are used to generate the SSS. Sequences $s_0^{(m_0)}$ and $s_1^{(m_1)}$ are different cyclic shifts of an m-sequence, $\tilde{s}$. The indices m_0 and m_1 are derived from the cell-identity group, $N_{ID}^{(2)}$ and determine the cyclic shift. The values can be read from table 6.11.2.1-1 in [2].

The two sequences are scrambled with a binary scrambling code ($c_0(n)$, $c_1(n)$), which depends on $N_{ID}^{(2)}$.

The second SSS sequence used in each radio frame is scrambled with a binary scrambling code ($z_1^{(m_0)}$, $z_1^{(m_1)}$) corresponding to the cyclic shift value of the first sequence transmitted in the radio frame.

Binary Sequence Generation

The sequences $s_0^{(m_0)}$ and $s_1^{(m_1)}$ are given by these equations:

$$s_0^{(m_0)}(n) = \tilde{s}((n + m_0) \bmod 31)$$

$$s_1^{(m_1)}(n) = \tilde{s}((n + m_1) \bmod 31)$$

$\tilde{s}$ is generated from the primitive polynomial $x^5 + x^2 + 1$ over the finite field GF(2).

$c_0(n)$ and $c_1(n$ are given by these equations:

$$c_0(n) = \tilde{c}\left(\left(n + N_{\mathrm{ID}}^{(2)}\right) \bmod 31\right)$$

$$c_1(n) = \tilde{c}\left(\left(n + N_{\mathrm{ID}}^{(2)} + 3\right) \bmod 31\right)$$

$\tilde{c}$ is generated from the primitive polynomial $x^5 + x^3 + 1$ over the finite field GF(2).

$z_1^{(m_0)}$ and $z_1^{(m_1)}$ are given by these equations:

$$z_1^{(m_0)}(n) = \tilde{z}((n + (m_0 \bmod 8)) \bmod 31)$$

$$z_1^{(m_1)}(n) = \tilde{z}((n + (m_1 \bmod 8)) \bmod 31)$$

$\tilde{z}$ is generated from the primitive polynomial $x^5 + x^4 + x^2 + x + 1$ over the finite field GF(2).

Mapping of the SSS

The scrambled sequences are interleaved to alternate the sequence transmitted in the first and second SSS transmission in each radio frame. This allows the receiver to determine the frame timing from observing only one of the two sequences; if the first SSS signal observed is in subframe 0 or subframe 5, synchronization can be achieved when the SSS signal is observed in subframe 0 or subframe 5 of the next frame.

As with PSS, the SSS is mapped to different OFDM symbols depending on which frame type is used:

- **FDD** — The SSS is transmitted in the same subframe as the PSS but one OFDM symbol earlier. The SSS is mapped to the same subcarriers (middle 72 subcarriers) as the PSS.
- **TDD** — The SSS is mapped to the last OFDM symbol in slots 1 and 11, which is three OFDM symbols before the PSS.

The SSS is constructed using different scrambling sequences when mapped to even and odd resource elements.

- Even resource elements:
  - Subframe 0: $d(2n) = s_0^{(m_0)}(n)c_0(n)$
  - Subframe 5: $d(2n) = s_1^{(m_1)}(n)c_0(n)$
- Odd resource elements:
  - Subframe 0: $d(2n+1) = s_1^{(m_1)}(n)c_1(n)z_1^{(m_0)}(n)$
  - Subframe 5: $d(2n+1) = s_0^{(m_0)}(n)c_1(n)z_1^{(m_1)}(n)$

$d(n)$ is mapped from lowest subcarrier to highest subcarrier.

References

[1] Chu, D. C. “Polyphase codes with good periodic correlation properties.” *IEEE Trans. Inf. Theory*. Vol. 18, Number 4, July 1972, pp. 531–532.

[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

`ltePSS` | `ltePSSIndices` | `lteSSS` | `lteSSSIndices` | `lteCellSearch` | `lteDLFrameOffset` | `lteDLResourceGrid` | `zadoffChuSeq`

Related Examples

- “Create Synchronization Signals” on page 3-2

Sounding Reference Signal (SRS)

In this section...

Sounding reference signals (SRS) are transmitted on the uplink and allow the network to estimate the quality of the channel at different frequencies.

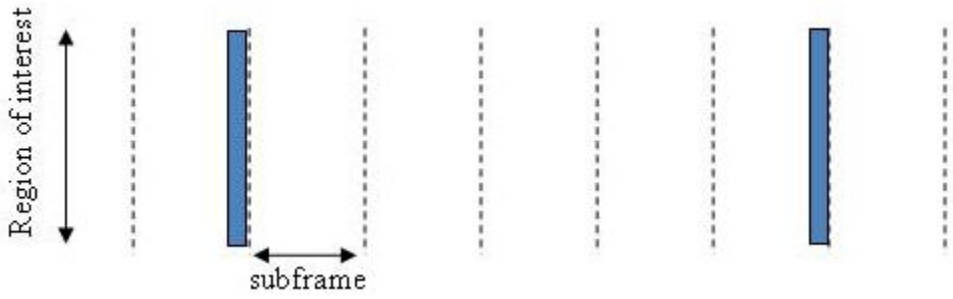
Sounding Reference Signals

The SRS is used by the base station to estimate the quality of the uplink channel for large bandwidths outside the assigned span to a specific UE. This measurement cannot be obtained with the DRS since these are always associated to the PUSCH or PUCCH and limited to the UE allocated bandwidth. Unlike the DRS associated with the physical uplink control and shared channels the SRS is not necessarily transmitted together with any physical channel. If the SRS is transmitted with a physical channel then it may stretch over a larger frequency band. The information provided by the estimates is used to schedule uplink transmissions on resource blocks of good quality.

SRS can be transmitted as often as every second subframe (2 ms) or as infrequent as every 16th frame (160ms). The SRS are transmitted on the last symbol of the subframe.

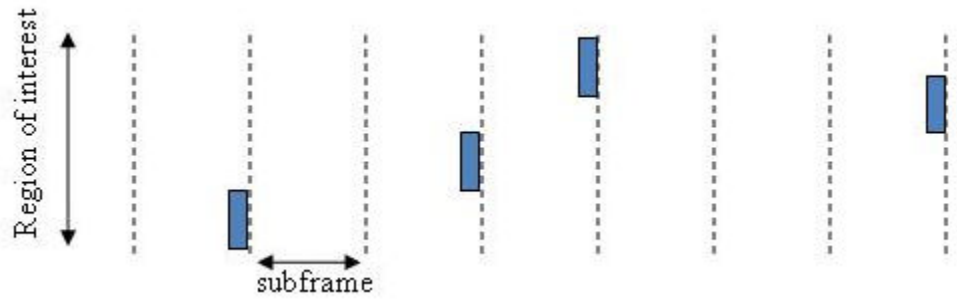
There are two methods of transmitting the SRS:

- **Wideband mode** — one single transmission of the SRS covers the bandwidth of interest. The channel quality estimate is obtained within a single SC-FDMA symbol. However, under poor channel conditions such as deep fade and high path loss, using this mode can result in a poor channel estimate.



Non-frequency-hopping SRS

- **Frequency-hopping mode** — the SRS transmission is split into a series of narrowband transmissions that will cover the whole bandwidth region of interest; this mode is the preferred method under poor channel conditions.



Frequency-hopping SRS

Sounding Reference Signals Generation

The sounding reference signals are generated using a base sequence denoted by $r_{u,v}^{SRS}(n)$. This base sequence is discussed further in “Base sequence” on page 1-15. This base sequence, used expressly to denote the SRS sequence, is defined by the following equation.

$$r_{u,v}^{SRS}(n) = r_{u,v}^{(\alpha)}(n)$$

It is desirable for the SRS sequences to have small power variations in time and frequency, resulting in high power amplifier efficiency and comparable channel estimation quality for all frequency components. Zadoff-Chu sequences are good candidates as they exhibit constant power in time and frequency. However the number of Zadoff-Chu sequences is limited which makes them unsuitable for use on their own. The generation and mapping of the SRS are discussed further in this section.

Base sequence

The sounding reference signals are defined by a cyclic shift, α, of a base sequence, r.

The base sequence, r, is represented in the following equation.

$$r_{u,v}^{(\alpha)} = e^{j\alpha n} r_{u,v}(n)$$

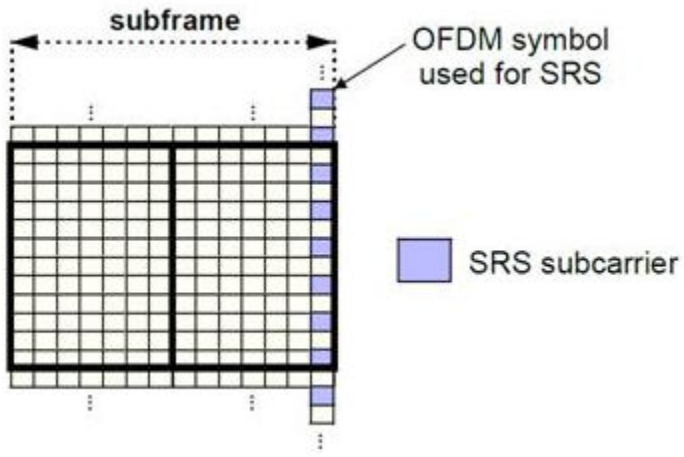
The preceding equation contains the following variables.

- $n = 0, \ldots, M_{SC}^{RS}$, where M_{SC}^{RS} is the length of the reference signal sequence.
- $U = 0, \ldots, 29$ is the base sequence group number.
- $V = 0, 1$ is the sequence number within the group and only applies to reference signals of length greater than 6 resource blocks.

A cyclic shift in the time domain (post IFFT in the OFDM modulation) is equivalent to a phase rotation in the frequency domain (pre-IFFT in the OFDM modulation). The base sequence is cyclic shifted to increase the total number of available sequences. For frequency non-selective channels over the 12 subcarriers of a resource block it is possible to achieve orthogonality between SRS generated from the same base sequence if $\alpha = 2\pi\frac{n_{SRS}}{8}$, where $n_{SRS}^{CS} = 0, 1, 2, 3, 4, 5, 6, 7$ (configured for each UE by higher layers) and assuming the SRS are synchronized in time.

The orthogonality can be exploited to transmit SRS at the same time, using the same frequency resources without mutual interference. Generally, SRS generated from different base sequences will not be orthogonal; however they will present low cross-correlation properties.

Similar to the uplink demodulation reference signals for the PUCCH and PUSCH, the SRS are time multiplexed. However, they are mapped to every second subcarrier in the last symbol of a subframe, creating a comb-like pattern, as illustrated in the following figure.



The minimum frequency span covered by the SRS in terms of bandwidth is 4 resource blocks and larger spans are covered in multiples of 4 resource blocks. This means that the minimum sequence length is 24. To maximize the number of available Zadoff-Chu sequences, a prime length sequence is needed. The minimum length sequence, 24, is not prime.

Therefore, Zadoff-Chu sequences are not suitable by themselves. Effectively, there are the following two types of base reference sequences:

- those with a sequence length ≥ 48 (spanning 8 or more resource blocks), which use a cyclic extension of Zadoff-Chu sequences
- those with a sequence length = 24 (spanning 4 resource blocks), which use a special QPSK sequence

Base sequences of length 48 and larger

For sequences of length 48 and larger (i.e. $M_{sc}^{RS} \geq 3N_{sc}^{RS}$), the base sequence is a repetition, with a cyclic offset of a Zadoff-Chu sequence of length N_{zc}^{RS}, where N_{zc}^{RS} is the largest prime such that $N_{zc}^{RS} < M_{sc}^{RS}$. Therefore, the base sequence will contain one complete length N_{zc}^{RS} Zadoff-Chu sequence plus a fractional repetition appended on the end. At the receiver the appropriate de-repetition can be done and the zero autocorrelation property will hold across the length N_{zc}^{RS} vector.

Base sequences of length 24

For sequences of length 24 (i.e. $M_{sc}^{RS} = 12, 24$), the sequences are a composition of unity modulus complex numbers drawn from a simulation generated table. These sequences have been found through computer simulation and are specified in the LTE specifications.

SRS Grouping

There are a total of 30 sequence groups, $u \in \{0, 1, \ldots, 29\}$, each containing one sequence for length less than or equal to 60. This corresponds to transmission bandwidths of 1,2,3,4 and 5 resource blocks. Additionally, there are two sequences (one for $v = 0$ or 1) for length ≥ 72; corresponding to transmission bandwidths of 6 resource blocks or more.

Note that not all values of m are allowed, where m is the number of resource blocks used for transmission. Only values for m that are the product of powers of 2, 3 and 5 are valid, as shown in the following equation.

$m = 2^{\alpha 0} \times 3^{\alpha 1} \times 5^{\alpha 2}$, where α_i are positive integers

The reason for this restriction is that the DFT sizes of the SC-FDMA precoding operation are limited to values which are the product of powers of 2, 3 and 5. The DFT operation can span more than one resource block, and since each resource block has 12 subcarriers, the total number of subcarriers fed to the DFT will be $12m$. Since the result of $12m$ has to be the product of powers of 2, 3 and 5 this implies that the number of resource blocks must themselves be the product of powers of 2, 3 and 5. Therefore values of m such as 7, 11, 14, 19, etc. are not valid.

For a given time slot, the uplink reference signal sequences to use within a cell are taken from one specific sequence group. If the same group is to be used for all slots then this is known as fixed assignment. On the other hand, if the group number u varies for all slots within a cell this is known as group hopping.

Fixed group assignment

When fixed group assignment is used, the same group number is used for all slots. The group number u is the same as for PUCCH and is a function of the cell identity number modulo 30.

$u = N_{ID}^{cell} mod 30$, with $N_{ID}^{cell} = 0, 1, \ldots, 503$

Group hopping

If group hopping is used, the pattern is applied to the calculation of the sequence group number.

This pattern is defined as the following equation.

$f_{gh}(n_s) = \sum_{i=0}^{7} c(8n_s + i) \cdot 2^i mod 30$

As shown in the preceding equation, this group hopping pattern is a function of the slot number n_s and is calculated making use of a pseudorandom binary sequence $c(k)$, generated using a length-30 Gold code. To generate the group hopping pattern, the PRBS generator is initialized with the following value at the start of each radio frame.

$$c_{init} = \left\lfloor \frac{N_{ID}^{cell}}{30} \right\rfloor$$

For SRS with group hopping, the group number, u, is given by the following equation.

$u = \left(f_{gh}(n_s) + N_{ID}^{cell} \text{mod} 30\right) \text{mod} 30$

See Also

`lteSRS` | `lteSRSIndices` | `lteSRSInfo` | `lteULResourceGrid`

Related Examples

- “Uplink Waveform Modeling Using SRS and PUCCH” on page 2-182

Resource Element Groups (REGs)

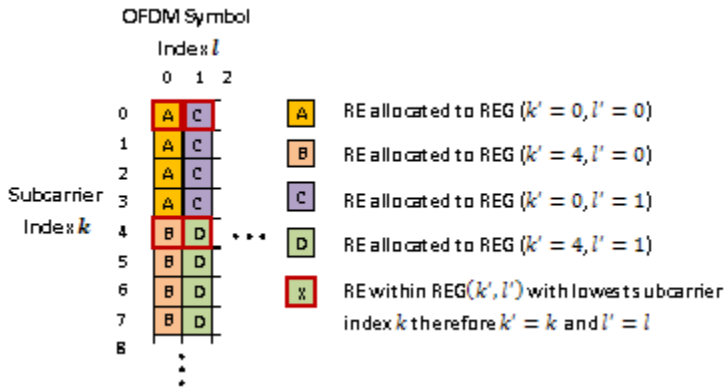
In this section...

Resource-element groups (REG) are used to define the mapping of control channels to resource elements (RE).

REGs are blocks of consecutive REs within the same OFDM symbol. The REGs within a subframe are located in the first four OFDM symbols and are identical in size and number for each corresponding subframe on every antenna port.

Resource Element Group Indexing

REGs are represented by an index pair (k', l'). The index k' is the subcarrier index of the RE within the REG with the lowest subcarrier index k. The index l' is the OFDM symbol index of the REG (l). This index pair is illustrated in the following figure.



Size and Location of REGs

The number of REs within a REG is such that a REG contains four REs which are not occupied by a cell specific reference signal on any antenna port in use.

All REs within a resource block in one of the first four OFDM symbols are allocated to a REG. Therefore the number of REs within each REG and the number of REGs within an OFDM symbol is affected by the number of cell-specific reference signals present on all antenna ports.

The number and location of cell specific reference signals are dependent on the number of antenna ports and the type of cyclic prefix used.

Antenna Port Configurations

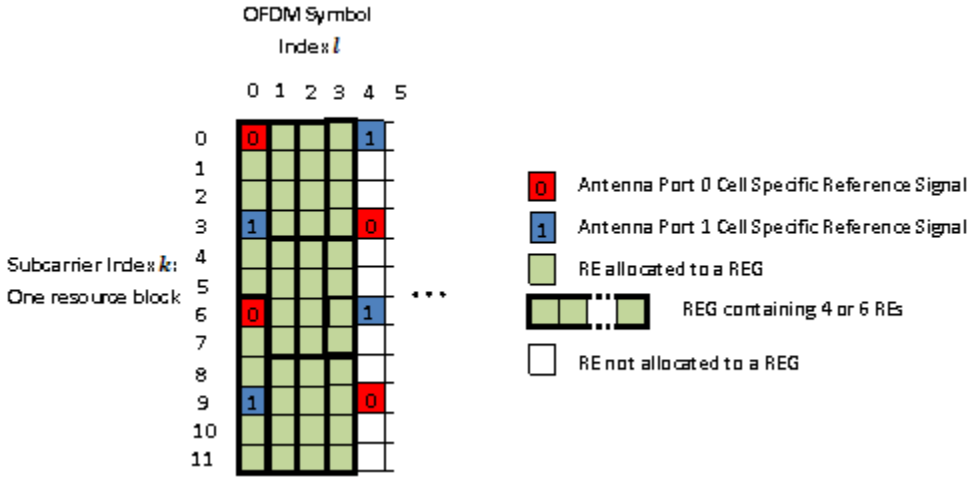
Each antenna port has a unique cell specific reference signal associated with it. As the REG arrangement is affected by cell specific reference signals, the REG arrangement for a one or two antenna port configuration or four antenna port configuration is different. The REG arrangement for each resource block within a subframe and for every antenna port is identical.

REG Arrangement with a Normal Cyclic Prefix

The REG arrangement for each antenna port configuration is described below for a normal cyclic prefix.

One or Two Antenna Port Configuration

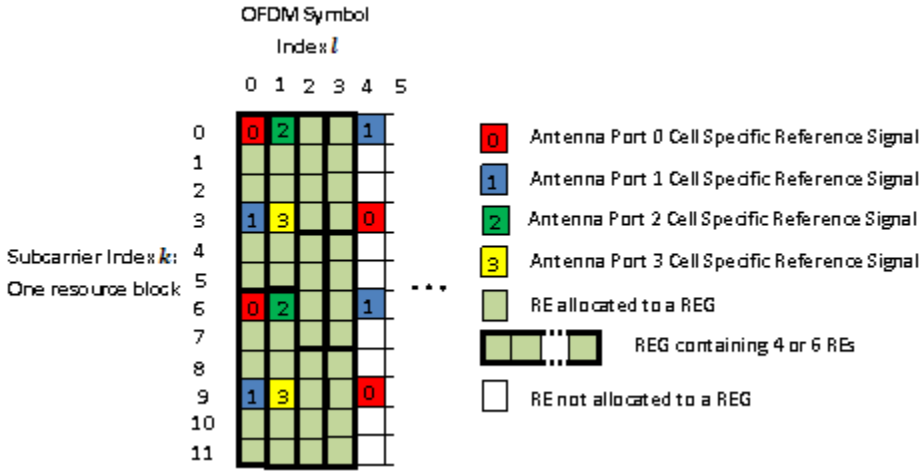
When antenna port 0 or ports 0 and 1 are used it is assumed the cell specific reference signal is present on both antenna ports 0 and 1. This leads to a REG arrangement for each resource block as shown in the following figure.



Cell-specific reference signals are present within the first OFDM symbol. As four REs not containing cell specific reference signals are required in a REG, the twelve REs in the first symbol are divided into two REGs, each containing six REs (two containing cell specific reference signals and four empty). In the second and third OFDM symbols no cell specific reference signal is present therefore the twelve REs in each symbol are divided between three REGs, each containing four REs.

Four Antenna Port Configuration

The REG arrangement in each resource block for four antenna port configuration is shown in the following figure.



The REG allocation within the first OFDM symbol is the same as for a one or two antenna port configuration. Four cell specific reference signals are present in the second OFDM symbol therefore eight REs are available for the mapping of control data. The twelve REs are divided into two REGs, each containing six REs. The third and fourth OFDM symbols contain no reference signals so three REGs are available.

REG Arrangement with an Extended Cyclic Prefix

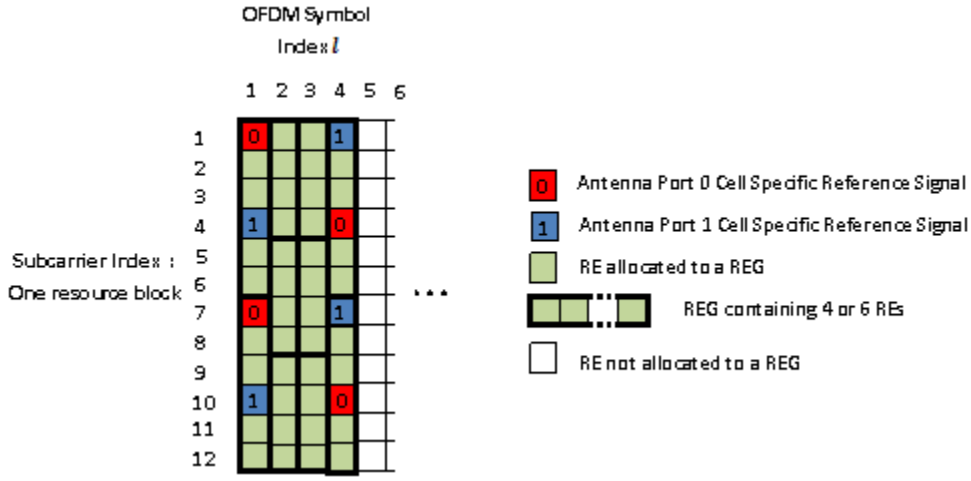
An extended cyclic prefix subframe contains twelve OFDM symbols as opposed to fourteen for a normal cyclic prefix. As the number of cell specific reference signals in a normal or extended cyclic prefix subframe is the same, the limited number of OFDM symbols in an extended cyclic prefix subframe requires the OFDM symbol spacing of the cell specific reference signals to be reduced compared to when using a standard cyclic prefix.

This reduction is spacing causes cell specific reference signals to be present within the fourth OFDM symbol of an extended cyclic prefix subframe whilst in a normal cyclic prefix subframe no cell specific reference signals are present. Therefore when an extended cyclic prefix is used two REGs, each containing six REs, are present in the fourth OFDM symbol.

The number of cell specific reference symbols within the first three OFDM symbols is identical for normal or extended cyclic prefix therefore the REG configurations are identical.

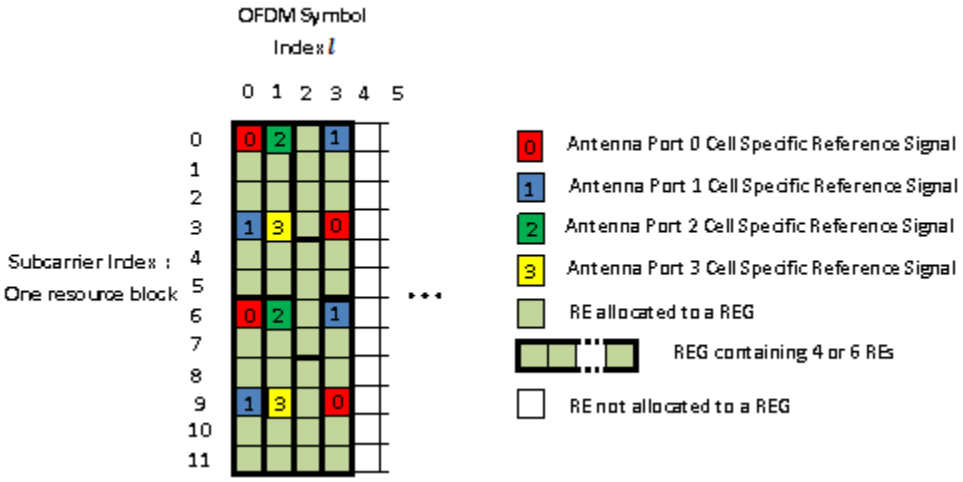
One or Two Antenna Port Configuration

The REG arrangement for a one or two antenna port configuration when using an extended cyclic prefix is shown in the following figure.



Four Antenna Port Configuration

The REG arrangement for a four antenna port configuration when using an extended cyclic prefix is shown in the following figure.



See Also

`ltePDCCH` | `ltePHICH` | `ltePCFICH`

More About

- “Represent Resource Grids”

Control Format Indicator (CFI) Channel

In this section...

When transmitting data on the downlink in an OFDM communication system, it is important to specify how many OFDM symbols are used to transmit the control channels so the receiver knows where to find control information. In LTE, the Control Format Indicator (CFI) value defines the time span, in OFDM symbols, of the Physical Downlink Control Channel (PDCCH) transmission (the control region) for a particular downlink subframe. The CFI is transmitted using the Physical Control Format Indicator Channel (PCFICH).

Control Format Indicator Values

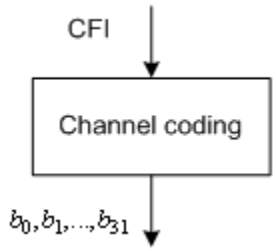
The CFI is limited to the value 1, 2, or 3. For bandwidths greater than ten resource blocks, the number of OFDM symbols used to contain the downlink control information is the same as the actual CFI value. Otherwise, the span of the downlink control information (DCI) is equal to *CFI*+1 symbols.

PCFICH Resourcing

The PCFICH is mapped in terms of Resource Element Groups (REGs) and is always mapped onto the first OFDM symbol. The number of REGs allocated to the PCFICH transmission is fixed to 4 i.e. 16 Resource Elements (REs). A PCFICH is only transmitted when the number of OFDM symbols for PDCCH is greater than zero.

CFI Channel Coding

The CFI value undergoes channel coding to form the PCFICH payload, as shown in the following figure.



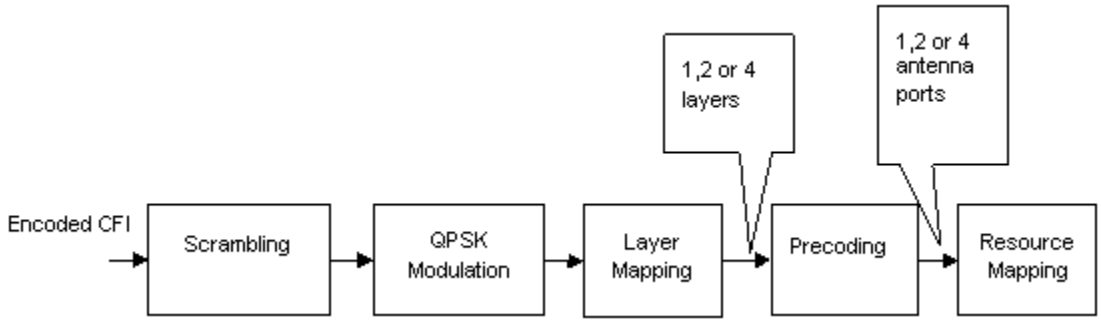
Using the following table contains the CFI codeword for each CFI value. Using these codewords corresponds to a block encoding rate of 1/16, changing a two bit CFI value to a 32 bit codeword.

CFI	CFI codeword $<b_0, b_1, \ldots, b_{31}>$
1	<0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1>
2	<1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0>

CFI	CFI codeword <b_0, b_1, ... , b_{31}>
3	<1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1>
4 (Reserved)	<0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0>

The PCFICH

The coded CFI is scrambled before undergoing QPSK modulation, layer mapping and precoding as shown in the following figure.



- “Scrambling” on page 1-24
- “Modulation” on page 1-24
- “Layer Mapping” on page 1-24
- “Precoding” on page 1-25
- “Mapping to Resource Elements” on page 1-27

Scrambling

The 32-bit coded CFI block undergoes a bit-wise exclusive-or (XOR) operation with a cell-specific scrambling sequence. The scrambling sequence is a pseudo-random sequence created using a length-31 Gold sequence generator. At the start of each subframe, it is initialized using the slot number within the radio frame, n_s, and the cell ID, N_{ID}^{cell}.

$$c_{init} = \left(\left\lfloor \frac{n_s}{2} \right\rfloor + 1\right) \times \left(2N_{ID}^{cell} + 1\right) \times 2^9 + N_{ID}^{cell}$$

Scrambling with a cell specific sequence serves the purpose of intercell interference rejection. When a UE descrambles a received bit stream with a known cell specific scrambling sequence, interference from other cells will be descrambled incorrectly and will only appear as uncorrelated noise.

Modulation

The scrambled bits are then QPSK modulated to create a block of complex-valued modulation symbols.

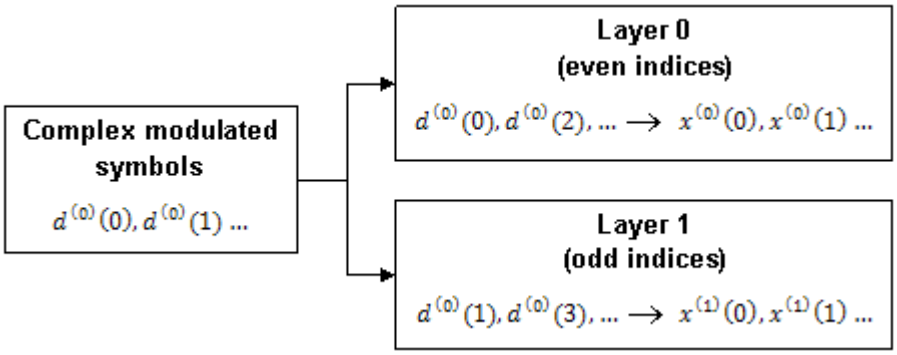
Layer Mapping

The complex symbols are mapped to one, two, or four layers depending on the number of transmit antennas used. The complex modulated input symbols, $d^{(0)}(i)$, are mapped onto v layers, $x^{(0)}(i), x^{(1)}(i), \ldots, x^{(v-1)}(i)$.

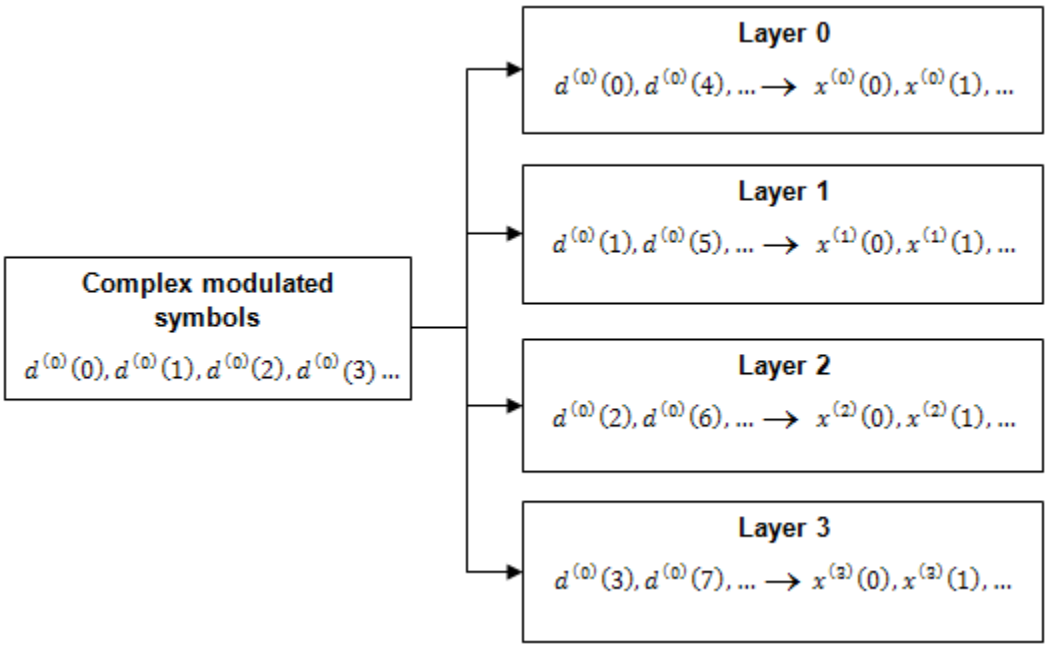
If a single antenna port is used, only one layer is used. Therefore, $x^{(0)}(i) = d^{(0)}(i)$.

If transmitter diversity is used, the input symbols are mapped to layers based on the number of layers.

- **Two Layers** — Even symbols are mapped to layer 0 and odd symbols are mapped to layer 1, as shown in this figure.



- **Four Layers** — The input symbols are mapped to layers sequentially, as shown in the following figure.



Precoding

- “Two Antenna Port Precoding” on page 1-26
- “Four Antenna Port Precoding” on page 1-26

The precoder takes a block from the layer mapper, $x^{(0)}(i), x^{(1)}(i), \ldots, x^{(v-1)}(i)$, and generates a sequence for each antenna port, $y^{(p)}(i)$. The variable p is the transmit antenna port number, and can assume values of $\{0\}$, $\{0,1\}$, or $\{0,1,2,3\}$.

For transmission over a single antenna port, no processing is carried out, as shown in this equation.

$$y^{(p)}(i) = x^{(0)}(i)$$

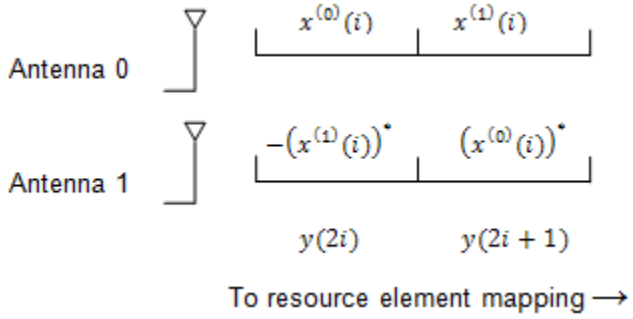
Precoding for transmit diversity is available on two or four antenna ports.

Two Antenna Port Precoding

An Alamouti scheme is used for precoding, which defines the relationship between input and output as shown in the following equation.

$$\begin{pmatrix} y^{(0)}(2i) \\ y^{(1)}(2i) \\ y^{(0)}(2i+1) \\ y^{(1)}(2i+1) \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & j & 0 \\ 0 & -1 & 0 & j \\ 0 & 1 & 0 & j \\ 1 & 0 & -j & 0 \end{pmatrix} \begin{pmatrix} \mathrm{Re}\{x^{(0)}(i)\} \\ \mathrm{Re}\{x^{(1)}(i)\} \\ \mathrm{Im}\{x^{(0)}(i)\} \\ \mathrm{Im}\{x^{(1)}(i)\} \end{pmatrix}$$

In the Alamouti scheme, two consecutive symbols, $x^{(0)}(i)$ and $x^{(1)}(i)$, are transmitted in parallel using two antennas with the following mapping, where the asterisk symbol (*) denotes the complex conjugate operation.



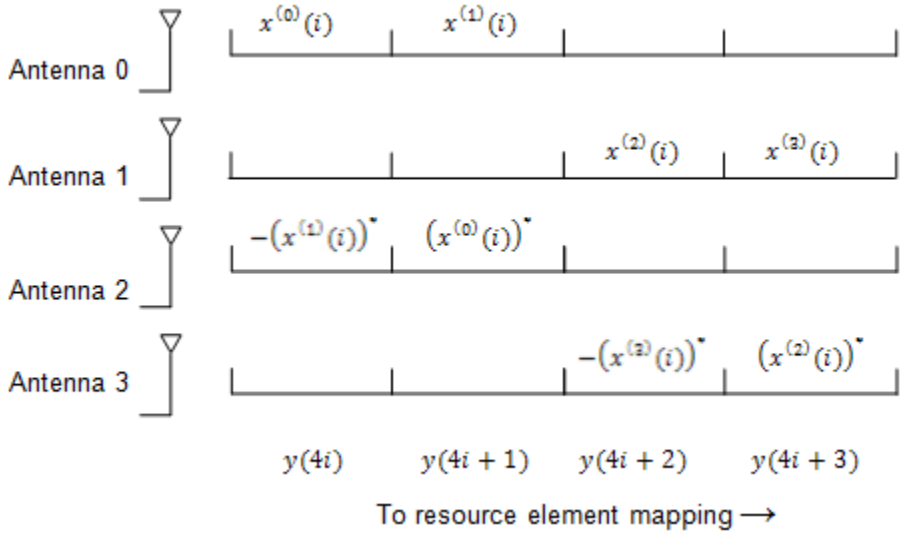
Since any two columns in the precoding matrix are orthogonal, the two symbols, $x^{(0)}(i)$ and $x^{(1)}(i)$, can be separated at the UE.

Four Antenna Port Precoding

Precoding for the four antenna port case defines the relationship between the input and output as shown in the following equation.

$$\begin{pmatrix} y^{(0)}(4i) \\ y^{(1)}(4i) \\ y^{(2)}(4i) \\ y^{(3)}(4i) \\ y^{(0)}(4i+1) \\ y^{(1)}(4i+1) \\ y^{(2)}(4i+1) \\ y^{(3)}(4i+1) \\ y^{(0)}(4i+2) \\ y^{(1)}(4i+2) \\ y^{(2)}(4i+2) \\ y^{(3)}(4i+2) \\ y^{(0)}(4i+3) \\ y^{(1)}(4i+3) \\ y^{(2)}(4i+3) \\ y^{(3)}(4i+3) \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 0 & j & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & j & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & j & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -j & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & j & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & j \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & j \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -j & 0 \end{pmatrix} \begin{pmatrix} \mathrm{Re}\{x^{(0)}(i)\} \\ \mathrm{Re}\{x^{(1)}(i)\} \\ \mathrm{Re}\{x^{(2)}(i)\} \\ \mathrm{Re}\{x^{(3)}(i)\} \\ \mathrm{Im}\{x^{(0)}(i)\} \\ \mathrm{Im}\{x^{(1)}(i)\} \\ \mathrm{Im}\{x^{(2)}(i)\} \\ \mathrm{Im}\{x^{(3)}(i)\} \end{pmatrix}$$

In this scheme, two consecutive symbols are transmitted in parallel in two symbol periods using four antennas with this mapping, where the asterisk symbol (*) denotes the complex conjugate operation.



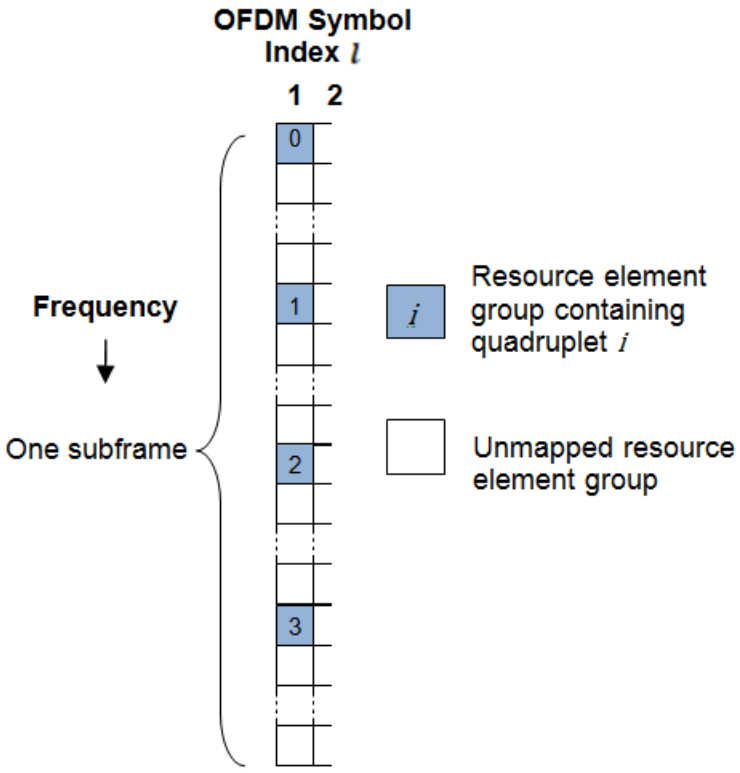
Mapping to Resource Elements

The complex valued symbols for each antenna are divided into quadruplets for mapping to resource elements. Each quadruplet is mapped to a Resource element Group (REG) within the first OFDM symbol. There are sixteen complex symbols to be mapped therefore four quadruplets are created.

The first quadruplet is mapped onto a REG with subcarrier index $k = \bar{k}$, given by the following equation.

$$\bar{k} = \left(\frac{N_{sc}^{RB}}{2}\right) \times \left(N_{ID}^{cell} mod 2N_{RB}^{DL}\right)$$

The subsequent three quadruplets are mapped to REGs spaced at intervals of $\left\lfloor N_{RB}^{DL}/2 \right\rfloor \times \left(N_{sc}^{RB}/2\right)$ from the first quadruplet and each other. This spreads the quadruplets, and hence the PCFICH, over the entire subframe as illustrated in the following figure.



See Also

`lteCFI` | `ltePCFICH` | `ltePCFICHInfo` | `ltePCFICHIndices` | `lteDLResourceGrid` | `lteSymbolModulate` | `lteSymbolDemodulate` | `ltePCFICHPRBS` | `lteLayerMap` | `lteLayerDemap` | `lteDLPrecode` | `lteDLDeprecode`

Related Examples

- “Model CFI and PCFICH” on page 3-4

HARQ Indicator (HI) Channel

In this section...

LTE uses a hybrid automatic repeat request (HARQ) scheme for error correction. The eNodeB sends a HARQ indicator to the UE to indicate a positive acknowledgement (ACK) or negative acknowledgement (NACK) for data sent using the uplink shared channel. The channel coded HARQ indicator codeword is transmitted through the Physical Hybrid Automatic Repeat Request Indicator Channel (PHICH).

HARQ Indicator

A HARQ indicator of ‘0’ represents a NACK and a ‘1’ represents an ACK.

PHICH Groups

Multiple PHICHs are mapped to the same set of resource elements (REs). This set of REs constitutes a PHICH group. The PHICHs within a PHICH group are separated through different orthogonal sequences.

A PHICH resource is identified by the index pair $\left(n_{PHICH}^{group}, n_{PHICH}^{seq}\right)$. The variable n_{PHICH}^{group} is the number of the PHICH group and the variable n_{PHICH}^{seq} is the orthogonal sequence index within the group. For more information about the orthogonal sequences, see “Scrambling” on page 1-31.

The number of PHICH groups varies based on whether the frame structure is type one, frequency division duplex (FDD), or type two, time division duplex (TDD).

Frame Structure Type 1: FDD

The number of PHICH groups is constant in all subframes and is given by the following equation.

$$N_{PHICH}^{group} = \begin{cases} \left\lceil N_g\left(N_{RB}^{DL}\right)/8 \right\rceil, & \text{for normal cyclic prefix} \\ 2 \times \left\lceil N_g\left(N_{RB}^{DL}\right)/8 \right\rceil, & \text{for extended cyclic prefix} \end{cases}$$

The set $N_g \in \left\{\frac{1}{6}, \frac{1}{2}, 1, 2\right\}$ is provided by higher layers and is a scaling factor to control the number of PHICH groups.

The index of the PHICH group n_{PHICH}^{group} ranges from 0 to n_{PHICH}^{group}–1.

Frame Structure Type 2: TDD

The number of PHICH groups varies depending on the number of the downlink subframe and the uplink/downlink time division duplex configuration. The number of groups is given by the expression

$m_i \times N_{PHICH}^{group}$. The variable n_{PHICH}^{group} is the number of PHICH groups for a frame structure type 1. The variable m_i is dependent on the subframe. The value for m_i for each uplink-downlink configuration and subframe number is given in the following table.

Uplink-downlink configuration	**Subframe number *i***									
	0	1	2	3	4	5	6	7	8	9
0	2	1	—	—	—	2	1	—	—	—
1	0	1	—	—	1	0	1	—	—	1
2	1	0	—	—	—	0	0	0	1	1
3	1	0	—	—	—	0	0	0	1	1
4	0	0	—	—	0	0	0	0	1	1
5	0	0	—	0	0	0	0	0	1	0
6	1	1	—	—	—	1	1	—	—	1

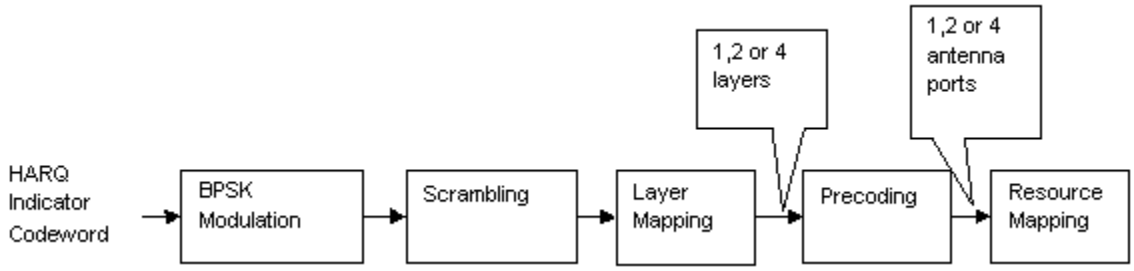
HARQ Indicator Channel Coding

The HARQ Indicator undergoes repetition coding to create a HARQ indicator codeword made up of three bits, $\langle b_0, b_1, b_2 \rangle$

HARQ Indicator	**HARQ Indicator Codeword** $\langle b_0, b_1, b_2 \rangle$
0 — Negative acknowledgement	$\langle 0, 0, 0 \rangle$
1 — Positive acknowledgement	$\langle 1, 1, 1 \rangle$

PHICH Processing

The HARQ Indicator codeword undergoes BPSK modulation, scrambling, layer mapping, precoding, and resource mapping as shown in block diagram in the following figure.



Modulation

The HARQ indicator codeword undergoes BPSK modulation resulting in a block of complex-valued modulated symbols, $z(0)$, $z(1)$, $z(2)$.

Scrambling

The block of modulated symbols is bitwise multiplied with an orthogonal sequence and a cell-specific scrambling sequence to create a sequence of symbols, $d(0), \ldots, d(M_{symb} - 1)$. The number of symbols, M_{symb}, is given by the equation $M_{symb} = 3 \times N_{SF}^{PHICH}$. The PHICH spreading factor, N_{SF}^{PHICH}, is 4 for a normal cyclic prefix and 2 for an extended cyclic prefix.

The orthogonal sequence allows multiple PHICHs to be mapped to the same set of resource elements.

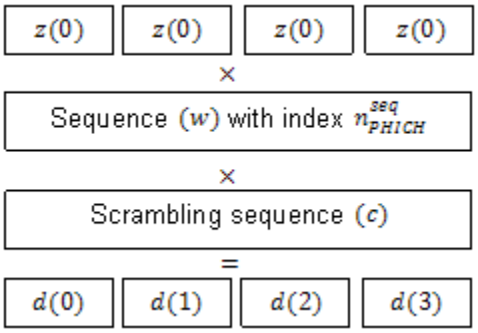
Scrambling with a cell-specific sequence serves the purpose of intercell interference rejection. When a UE descrambles a received bitstream with a known cell specific scrambling sequence, interference from other cells will be descrambled incorrectly and therefore only appear as uncorrelated noise.

The complex scrambled symbols, $d(0), \ldots, d(M_{symb} - 1)$, are created according to the following equation.

$$d(i) = w\left(i \bmod N_{SF}^{PHICH}\right) \times (1 - 2c(i)) \times z\left(\left\lfloor i/N_{SF}^{PHICH} \right\rfloor\right)$$

The first term, $w\left(i \bmod N_{SF}^{PHICH}\right)$, is the orthogonal sequence symbol with index N_{SF}^{PHICH}. The second term, $(1 - 2c(i))$, is the cell-specific scrambling sequence symbol. The third term, $z\left(\left\lfloor i/N_{SF}^{PHICH} \right\rfloor\right)$, is the modulated HARQ indicator symbol.

The three modulated symbols, $z(0)$, $z(1)$, $z(2)$, are repeated N_{SF}^{PHICH} times and scrambled to create a sequence of six or twelve symbols depending on whether a normal or extended cyclic prefix is used. When using a normal cyclic prefix, the first four scrambled symbols are created as shown in the following figure.



The variable w is an orthogonal scrambling sequence with index n_{PHICH}^{seq}. The sequences are given in the following table.

Sequence Index	**Orthogonal Sequence,** $w(0), \ldots, w\left(N_{SF}^{PHICH} - 1\right)$	
n_{PHICH}^{seq}	**Normal Cyclic Prefix,** $N_{SF}^{PHICH} = 4$	**Extended Cyclic Prefix,** $N_{SF}^{PHICH} = 2$
0	[+1 +1 +1 +1]	[+1 +1]
1	[+1 –1 +1 –1]	[+1 –1]
2	[+1 +1 –1 –1]	[+j +j]

Sequence Index	**Orthogonal Sequence,** $w(0), \ldots, w\left(N_{SF}^{PHICH}-1\right)$	
n_{PHICH}^{seq}	**Normal Cyclic Prefix,** $N_{SF}^{PHICH}=4$	**Extended Cyclic Prefix,** $N_{SF}^{PHICH}=2$
3	[+1 –1 –1 +1]	[+*j* –*j*]
4	[+*j* +*j* +*j* +*j*]	—
5	[+*j* –*j* +*j* –*j*]	—
6	[+*j* +*j* –*j* –*j*]	—
7	[+*j* –*j* –*j* +*j*]	—

The variable c is a cell-specific pseudo-random scrambling sequence created using a length-31 Gold sequence. The scrambling sequence is initialized using the slot number within the radio frame,n_s, and the cell ID, N_{ID}^{cell}.

$$c_{init} = (\lfloor n_s/2 \rfloor + 1) \times \left(2N_{ID}^{cell} + 1\right) \times 2^9 + N_{ID}^{cell}$$

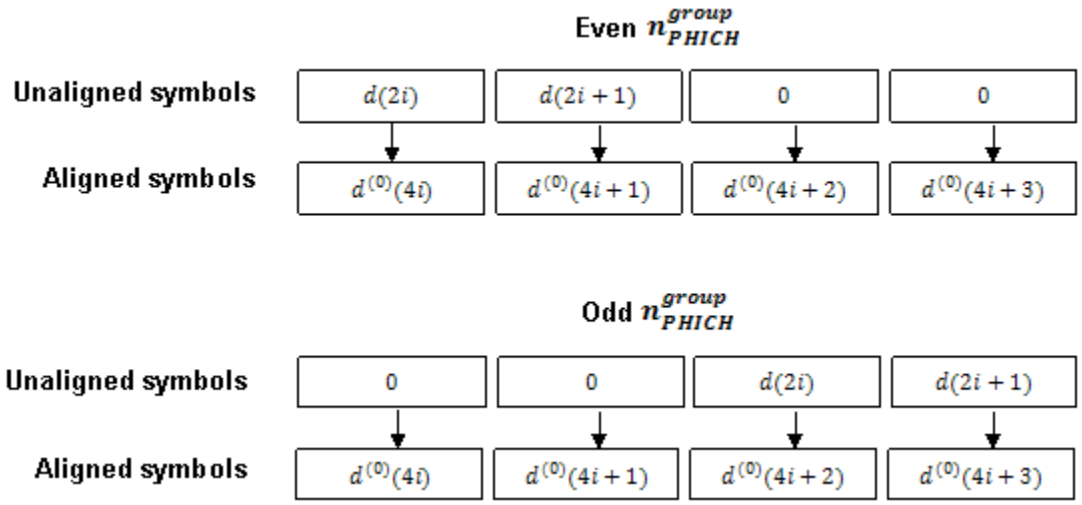
Resource Group Alignment

As resource element groups (REGs) contain four resource elements (each able to contain one symbol) the blocks of scrambled symbols are aligned to create blocks of four symbols.

In the case of a normal cyclic prefix, each of the original complex modulated symbols, $z(0)$, $z(1)$, $z(2)$, is represented by four scrambled symbols. Therefore, no alignment is required, as shown in the following equation.

$$d^{(0)}(i) = d(i)$$

In the case of an extended cyclic prefix each of the original complex modulated symbols, $z(0)$, $z(1)$, $z(2)$, is represented by two scrambled symbols. To create blocks of four symbols, zeros are added before or after blocks of two scrambled symbols depending on whether the PHICH index is odd or even. This allows two groups to be combined during the resource mapping stage and mapped to one REG. Groups of four symbols, $d^{(0)}$, are formed as shown in the following figure.



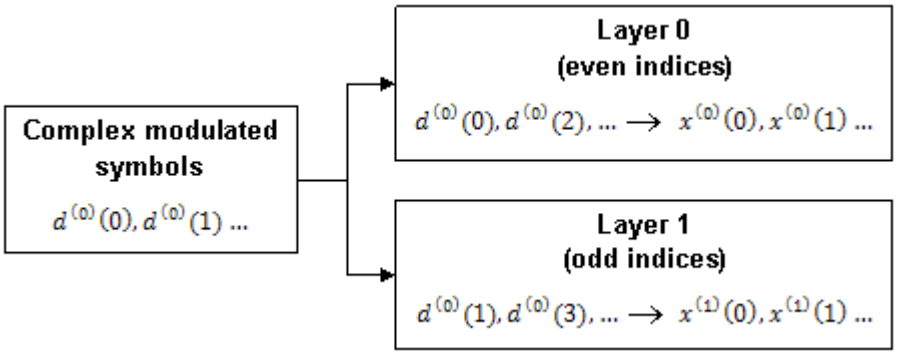
Layer Mapping

The complex symbols are mapped to one, two, or four layers depending on the number of transmit antennas used. The complex modulated input symbols, $d^{(0)}(i)$, are mapped onto v layers, $x^{(0)}(i), x^{(1)}(i), \ldots, x^{(v-1)}(i)$.

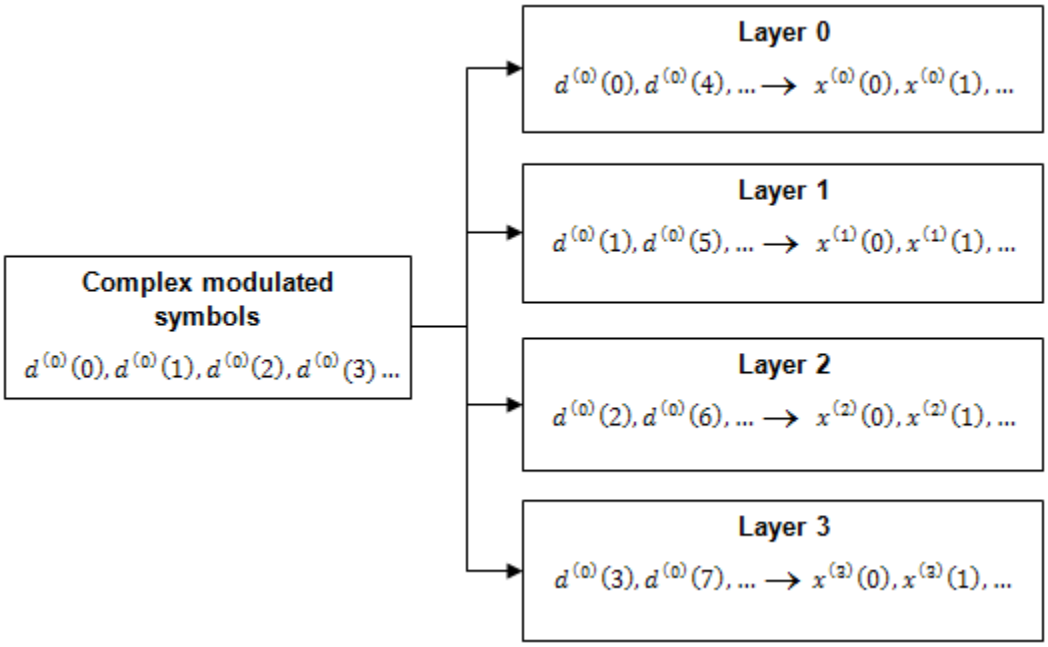
If a single antenna port is used, only one layer is used. Therefore, $x^{(0)}(i) = d^{(0)}(i)$.

If transmitter diversity is used, the input symbols are mapped to layers based on the number of layers.

- **Two Layers** — Even symbols are mapped to layer 0 and odd symbols are mapped to layer 1, as shown in this figure.



- **Four Layers** — The input symbols are mapped to layers sequentially, as shown in the following figure.



Precoding

The precoder takes a block from the layer mapper, $x^{(0)}(i), x^{(1)}(i), \ldots, x^{(v-1)}(i)$, and generates a sequence for each antenna port, $y^{(p)}(i)$. The variable p is the transmit antenna port number, and can assume values of {0}, {0,1}, or {0,1,2,3}.

For transmission over a single antenna port, no processing is carried out, as shown in this equation.

$$y^{(p)}(i) = x^{(0)}(i)$$

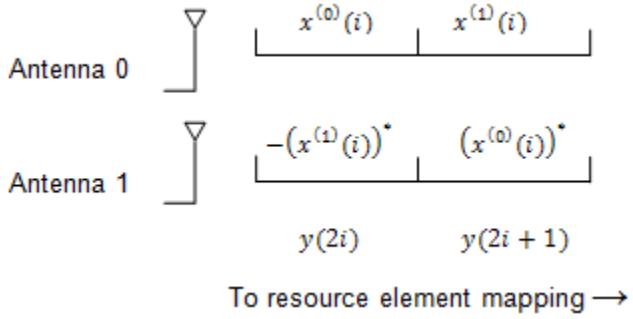
Precoding for transmit diversity is available on two or four antenna ports.

Two Antenna Port Precoding

An Alamouti scheme is used for precoding, which defines the relationship between input and output as shown in the following equation.

$$\begin{pmatrix} y^{(0)}(2i) \\ y^{(1)}(2i) \\ y^{(0)}(2i+1) \\ y^{(1)}(2i+1) \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & j & 0 \\ 0 & -1 & 0 & j \\ 0 & 1 & 0 & j \\ 1 & 0 & -j & 0 \end{pmatrix} \begin{pmatrix} \mathrm{Re}\{x^{(0)}(i)\} \\ \mathrm{Re}\{x^{(1)}(i)\} \\ \mathrm{Im}\{x^{(0)}(i)\} \\ \mathrm{Im}\{x^{(1)}(i)\} \end{pmatrix}$$

In the Alamouti scheme, two consecutive symbols, $x^{(0)}(i)$ and $x^{(1)}(i)$, are transmitted in parallel using two antennas with the following mapping, where the asterisk symbol (*) denotes the complex conjugate operation.



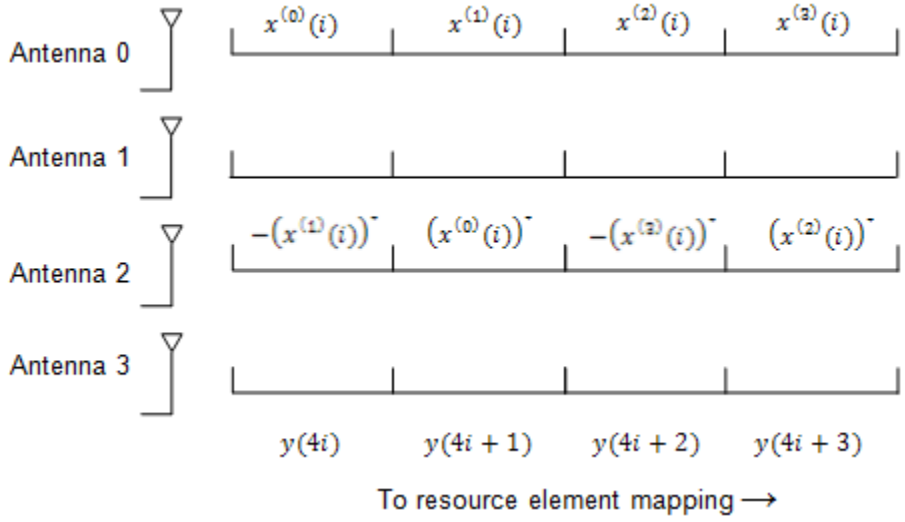
Since any two columns in the precoding matrix are orthogonal, the two symbols, $x^{(0)}(i)$ and $x^{(1)}(i)$, can be separated at the UE.

Four Antenna Port Precoding

Precoding for the four antenna port case depends on the index of the PHICH group, n_{PHICH}^{group}. If $n_{PHICH}^{group} + i$ is even for a normal cyclic prefix or if $\left\lfloor n_{PHICH}^{group}/2 \right\rfloor + i$ is even for an extended cyclic prefix, the relationship between the input and output is defined by the following equation.

$$\begin{pmatrix} y^{(0)}(4i) \\ y^{(1)}(4i) \\ y^{(2)}(4i) \\ y^{(3)}(4i) \\ y^{(0)}(4i+1) \\ y^{(1)}(4i+1) \\ y^{(2)}(4i+1) \\ y^{(3)}(4i+1) \\ y^{(0)}(4i+2) \\ y^{(1)}(4i+2) \\ y^{(2)}(4i+2) \\ y^{(3)}(4i+2) \\ y^{(0)}(4i+3) \\ y^{(1)}(4i+3) \\ y^{(2)}(4i+3) \\ y^{(3)}(4i+3) \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 0 & j & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & j & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & j & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -j & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & j & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & j \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & j \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -j & 0 \end{pmatrix} \begin{pmatrix} \mathrm{Re}\{x^{(0)}(i)\} \\ \mathrm{Re}\{x^{(1)}(i)\} \\ \mathrm{Re}\{x^{(2)}(i)\} \\ \mathrm{Re}\{x^{(3)}(i)\} \\ \mathrm{Im}\{x^{(0)}(i)\} \\ \mathrm{Im}\{x^{(1)}(i)\} \\ \mathrm{Im}\{x^{(2)}(i)\} \\ \mathrm{Im}\{x^{(3)}(i)\} \end{pmatrix}$$

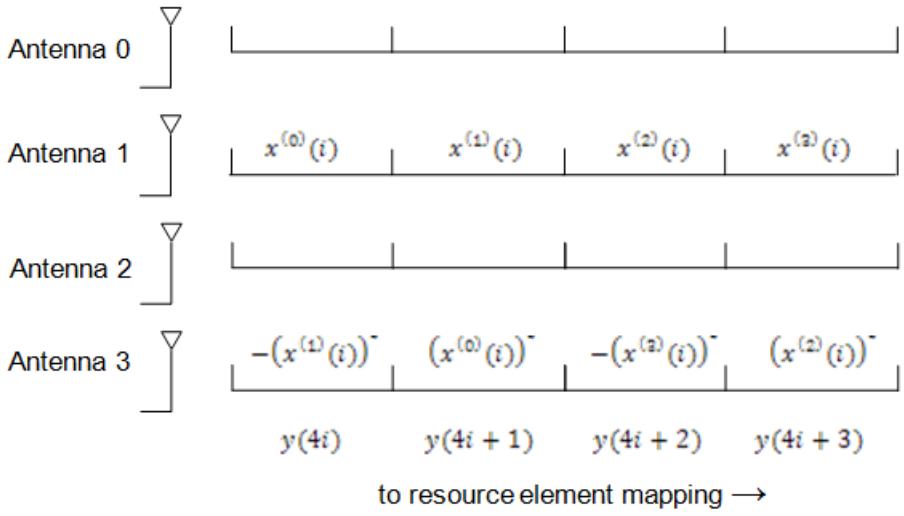
In this scheme, two consecutive symbols are transmitted in parallel in two symbol periods using four antennas with the following mapping, where the asterisk symbol (*) denotes the complex conjugate operation.



If $n_{PHICH}^{group} + i$ is odd for a normal cyclic prefix or if $\left\lfloor n_{PHICH}^{group}/2 \right\rfloor + i$ is odd for an extended cyclic prefix, the relationship between the input and output is defined by the following equation.

$$\begin{pmatrix} y^{(0)}(4i) \\ y^{(1)}(4i) \\ y^{(2)}(4i) \\ y^{(3)}(4i) \\ y^{(0)}(4i+1) \\ y^{(1)}(4i+1) \\ y^{(2)}(4i+1) \\ y^{(3)}(4i+1) \\ y^{(0)}(4i+2) \\ y^{(1)}(4i+2) \\ y^{(2)}(4i+2) \\ y^{(3)}(4i+2) \\ y^{(0)}(4i+3) \\ y^{(1)}(4i+3) \\ y^{(2)}(4i+3) \\ y^{(3)}(4i+3) \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & j & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & j & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & j & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -j & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & j & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & j \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & j \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -j & 0 \end{pmatrix} \begin{pmatrix} \mathrm{Re}\{x^{(0)}(i)\} \\ \mathrm{Re}\{x^{(1)}(i)\} \\ \mathrm{Re}\{x^{(2)}(i)\} \\ \mathrm{Re}\{x^{(3)}(i)\} \\ \mathrm{Im}\{x^{(0)}(i)\} \\ \mathrm{Im}\{x^{(1)}(i)\} \\ \mathrm{Im}\{x^{(2)}(i)\} \\ \mathrm{Im}\{x^{(3)}(i)\} \end{pmatrix}$$

In this scheme, two consecutive symbols are transmitted in parallel in two symbol periods using four antennas with the following mapping, where the asterisk symbol (*) denotes the complex conjugate operation.



Mapping to Resource Elements

PHICH Duration

The number of OFDM symbols used to carry the PHICH is configurable by the PHICH duration.

The PHICH duration is either normal or extended. A normal PHICH duration causes the PHICH to be present in only the first OFDM symbol. In general an extended PHICH duration causes the PHCH to

be present in the first three OFDM symbols but there are some exceptions. The PHICH is present in the first two OFDM symbols under the following exceptions.

- Within subframe 1 and 6 when frame structure type 2 (TDD) is used
- Within MBSFN subframes

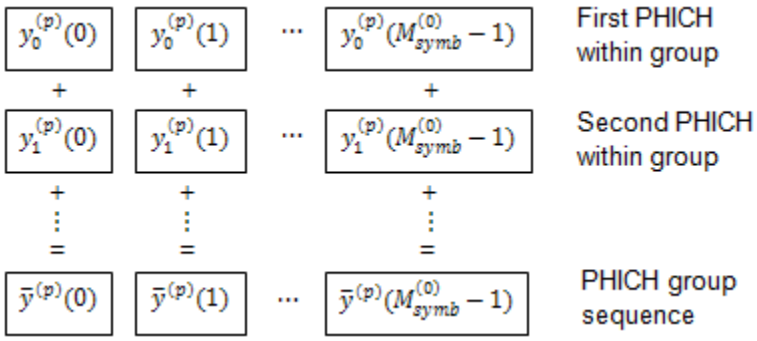
Relationship between CFI and PHICH Duration

Since the control format indicator (CFI) configures how many OFDM symbols are used for mapping the physical downlink control channel (PDCCH) and hence which OFDM symbols are available for the physical downlink shared channel (PDSCH), care must be taken when using an extended PHICH duration so the PHICH is not mapped into the same region as the PDSCH.

For example, when using an extended PHICH in subframe zero of a frame structure type 1 (FDD) 10MHz subframe the first three OFDM symbols will contain PHICH. Therefore, the CFI must be set to 3 so the PDSCH is not mapped to OFDM symbols 0, 1, or 2, and so does not overlap with the PHICH.

Combining PHICH Sequences

Corresponding elements of each PHICH sequence are summed to create the sequence for each PHICH group, $\left(\bar{y}^{(p)}\right)$. This process is illustrated in the following figure.



The variable $y_i^{(p)}(n)$ is the n-th element within PHICH i on antenna port p.

PHICH Mapping Units

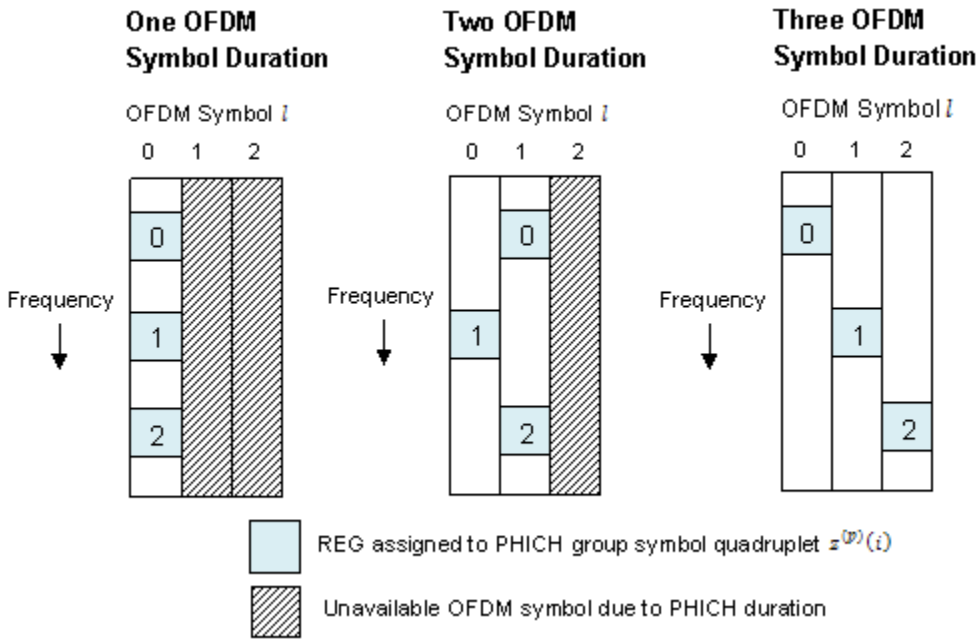
PHICHs are mapped to REGs using PHICH mapping units, $\tilde{y}_{m'}^{(p)}$, where m' is the index of the mapping unit. For a normal cyclic prefix, each PHICH group is mapped to a PHICH mapping unit. In the case of an extended cyclic prefix, two PHICH groups are mapped to one PHICH mapping unit. Due to the location of the padding zeros added during resource group alignment, when two consecutive groups are added, the zeros of one group overlap the data of the other.

Mapping to REGs

Each mapping unit contains twelve symbols. To map these twelve symbols to REGs, the mapping units are split into three groups of four symbols (quadruplets).

Each of the three symbol quadruplets, $z^{(p)}(i), i = \{0, 1, 2\}$, is mapped to a REG, (k', l'), so the PHICH is spread over all available OFDM symbols and resource blocks.

The OFDM symbol index, l', is set so adjacent quadruplets are spread amongst the available OFDM symbols, as illustrated in the following figure.



The subcarrier index, k'_i, of the REG is based upon the cell ID, N_{ID}^{cell}, and is chosen to spread the three symbol quadruplets over the entire bandwidth.

See Also

`ltePHICH` | `ltePHICHInfo` | `ltePHICHIndices` | `ltePHICHPRBS` | `lteDLResourceGrid` | `lteLayerMap` | `lteLayerDemap` | `lteDLPrecode` | `lteDLDeprecode` | `lteCRCEncode` | `lteCRCDecode` | `lteSymbolModulate` | `lteSymbolDemodulate`

Related Examples

- “Model HARQ Indicator and PHICH” on page 3-6

Downlink Control Channel

In this section...

Control signaling is required to support the transmission of the downlink and uplink transport channels (DL-SCH and UL-SCH). Control information for one or multiple UEs is contained in a Downlink scheduling Control Information (DCI) message and is transmitted through the Physical Downlink Control Channel (PDCCH). DCI messages contain the following information.

- DL-SCH resource allocation (the set of resource blocks containing the DL-SCH) and modulation and coding scheme, which allows the UE to decode the DL-SCH.
- Transmit Power Control (TPC) commands for the Physical Uplink Control Channel (PUCCH) and UL-SCH, which adapt the transmit power of the UE to save power
- Hybrid-Automatic Repeat Request (HARQ) information including the process number and redundancy version for error correction
- MIMO precoding information

DCI Message Formats

Depending on the purpose of DCI message, different DCI formats are defined. The DCI formats are given in the following list.

- **Format 0** — for transmission of uplink shared channel (UL-SCH) allocation
- **Format 1** — for transmission of DL-SCH allocation for Single Input Multiple Output (SIMO) operation
- **Format 1A** — for compact transmission of DL-SCH allocation for SIMO operation or allocating a dedicated preamble signature to a UE for random access
- **Format 1B** — for transmission control information of multiple-input multiple-output (MIMO) rank-1 based compact resource assignment
- **Format 1C** — for very compact transmission of PDSCH assignment
- **Format 1D** — same as *Format 1B*, but with additional information of power offset
- **Format 2** and **Format 2A**— for transmission of DL-SCH allocation for closed and open loop MIMO operation, respectively
- **Format 2B** — for the scheduling of dual layer transmission (antenna ports 7 & 8)
- **Format 2C** — for the scheduling of up to 8 layer transmission (antenna ports 7 to 14) using TM9
- **Format 2D** — for the scheduling of up to 8 layer transmission (antenna ports 7 to 14) using TM10
- **Format 3** and **Format 3A** — for transmission of TPC command for an uplink channel
- **Format 4** — for the scheduling of PUSCH with multi-antenna port transmission mode

In one subframe, multiple UE’s can be scheduled. Therefore, multiple DCI messages can be sent using multiple PDCCH’s.

PDCCH Restructuring

A PDCCH is transmitted on one or an aggregation of several consecutive control channel elements (CCEs). A CCE is a group of nine consecutive resource-element groups (REGs). The number of CCEs used to carry a PDCCH is controlled by the PDCCH format. A PDCCH format of 0, 1, 2, or 3 corresponds to 1, 2, 4, or 8 consecutive CCEs being allocated to one PDCCH.

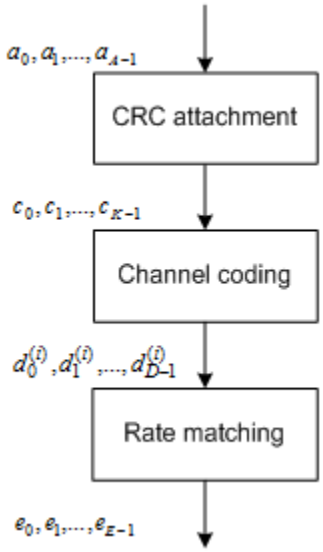
DCI Message Generation

The base station creates a DCI message based on a DCI format given in TS 36.212 [1], Section 5.3.3.1. Each field in a DCI message is mapped in order. Zeros may be appended to a DCI message to avoid ambiguous message lengths.

DCI Coding

- “CRC Attachment” on page 1-40
- “Channel Coding — Tail-Biting Convolutional Coding” on page 1-41
- “Rate Matching” on page 1-42

To form the PDCCH payload, the DCI undergoes coding as shown in the following figure.



CRC Attachment

A cyclic redundancy check (CRC) is used for error detection in DCI messages. The entire PDCCH payload is used to calculate a set of CRC parity bits. The PDCCH payload is divided by a cyclic generator polynomial to generate 16 parity bits. These parity bits are then appended to the end of the PDCCH payload.

As multiple PDCCHs relevant to different UEs can be present in one subframe, the CRC is also used to specify to which UE a PDCCH is relevant. This is done by scrambling the CRC parity bits with the corresponding Radio Network Temporary Identifier (RNTI) of the UE. The scrambled CRC is obtained by performing a bit-wise XOR operation between the 16-bit calculated PDCCH CRC and the 16-bit RNTI.

Different RNTI can be used to scramble the CRC. The following RNTI are some examples.

- A UE unique identifier; for example, a Cell-RNTI
- A paging indication identifier, or Paging-RNTI, if the PDCCH contains paging information
- A system information identifier, or system information-RNTI, if the PDCCH contains system information

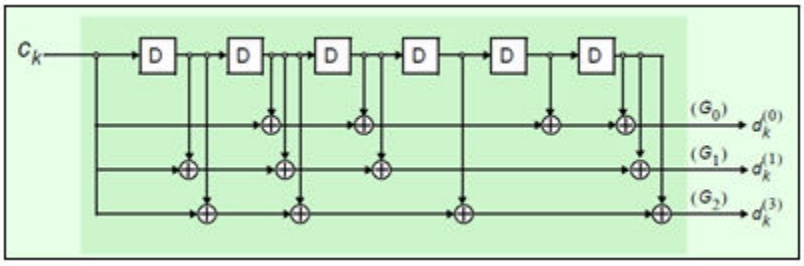
When encoding a format 0 DCI message, which contains the UE UL-SCH resource allocation, and the UE transmit antenna selection is configured and applicable, the RNTI scrambled CRC undergoes a bit-wise XOR operation with an antenna selection mask. This mask informs the UE transmit antenna on which port to transmit. The antenna selection masks are given in the following table.

UE transmit antenna selection	Antenna selection mask, $<x_0^{AS}, \ldots, x_{15}^{AS}>$
UE Port 0	<0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0>
UE Port 1	<0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1>

Channel Coding — Tail-Biting Convolutional Coding

The DCI message with the CRC attachment undergoes tail-biting convolutional coding as described in TS 36.212 [1], Section 5.1.3.1. Convolutional coding is a form of forward error correction and improves the channel capacity by adding carefully selected redundant information.

LTE uses a rate ⅓ tail-biting encoder with a constraint length, k, of 7. This means that one in three bits of the output contain useful information while the other two add redundancy. The structure of the convolutional encoder is shown in the following figure.



Each output stream of the coder is obtained by convolving the input with the impulse response of the encoder, as shown in the following equation.

$$d_k^{(i)} \rightarrow C_k * G_i$$

The impulse responses are called the generator sequences of the coder. For LTE, there are the following three generator sequences.

- G_0=133 (octal)
- G_1=171 (octal)
- G_2=165 (octal)

A standard convolutional encoder initializes its internal shift register to the all zeros state, and also ensures that the coder finishes in the all zeros state by padding the input sequence with k zeros at

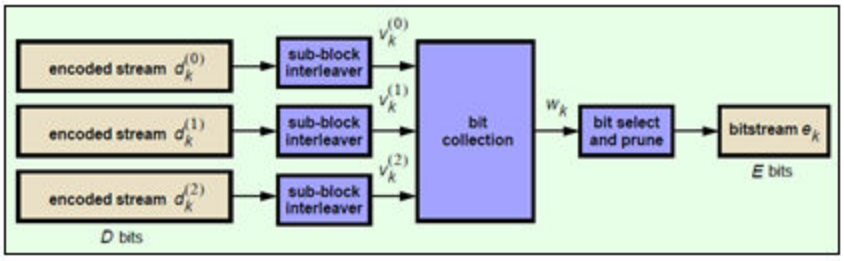
the end. Knowing the start and end states, which are all zeros, simplifies the design of the decoder, which is typically an implementation of the Viterbi algorithm.

A tail-biting convolutional coder initializes its internal shift register to the last k bits of the current input block, rather than to the all zeros state. Thus, the start and end states are the same, without the need to zero-pad the input block. Since the overhead of terminating the coder has been eliminated, the output block contains fewer bits than a standard convolutional coder. The drawback is that the decoder becomes more complicated because the initial state is unknown; however, the decoder does know the start and end states are the same.

Rate Matching

- “Sub-block Interleaver” on page 1-42
- “Bit Collection, Selection, and Transmission” on page 1-43

The rate matching block creates an output bitstream with a desired code rate. As the number of bits available for transmission depends on the available resources the rate matching algorithm is capable of producing any arbitrary rate. The three bitstreams from the tail-biting convolutional encoder are interleaved followed by bit collection to create a circular buffer. Bits are selected and pruned from the buffer to create an output bitstream with the desired code rate. The process is illustrated in the following figure.



Sub-block Interleaver

The three sub-block interleavers used in the rate matching block are identical. Interleaving is a technique to reduce the impact of burst errors on a signal as consecutive bits of data will not be corrupted.

The sub-block interleaver reshapes the encode bit sequence, row-by-row, to form a matrix with $C_{Subblock}^{TC} = 32$ columns and $R_{Subblock}^{TC}$ rows. The variable $R_{Subblock}^{TC}$ is determined by finding the minimum integer such that the number of encoded input bits is $D \le \left(R_{Subblock}^{TC} \times C_{Subblock}^{TC}\right)$. If $\left(R_{Subblock}^{TC} \times C_{Subblock}^{TC}\right) > D$, N_D <NULL>’s are appended onto the front of the encoded sequence. In this case, $N_D + D = \left(R_{Subblock}^{TC} \times C_{Subblock}^{TC}\right)$.

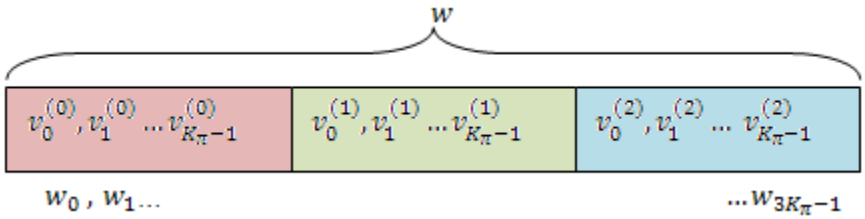
Inter-column permutation is performed on the matrix to reorder the columns as shown in the following pattern.

1, 17, 9, 25, 5, 21, 13, 29, 3, 19, 11, 27, 7, 23, 15, 31, 0, 16, 8, 24, 4, 20, 12, 28, 2, 18, 10, 26, 6, 22, 14, 30

The output of the block interleaver is the bit sequence read out column-by-column from the inter-column permutated matrix to create a stream $K_\Pi = \left(R_{Subblock}^{TC} \times C_{Subblock}^{TC}\right)$ bits long.

Bit Collection, Selection, and Transmission

The bit collection stage creates a virtual circular buffer by combining the three interleaved encoded bit streams, as shown in the following figure.

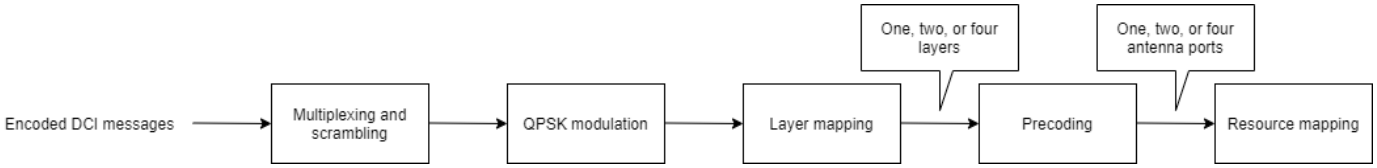


Bits are then selected and pruned from the circular buffer to create an output sequence length which meets the desired code rate. This is achieved by sequentially outputting the bits in the circular buffer from w_0 (looping back to w_0 after $w_{3K_\pi-1}$), discarding <NULL> bits, until the length of the output is x times the length of the input, creating a coding rate of $1/x$.

PDCCH Processing

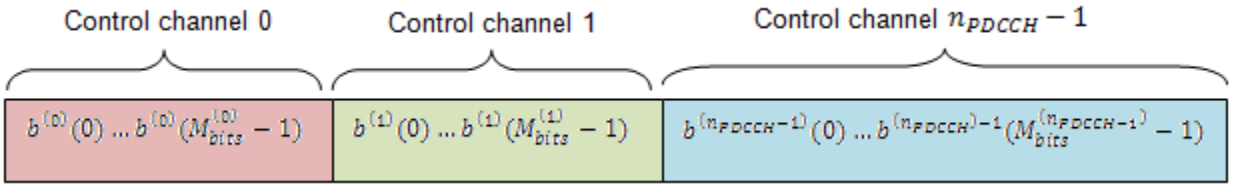
- "Multiplexing" on page 1-43
- "Matching PDCCHs to CCE Positions" on page 1-44
- "Scrambling" on page 1-45
- "Modulation" on page 1-46
- "Layer Mapping" on page 1-46
- "Precoding" on page 1-47
- "Mapping to Resource Elements" on page 1-49

The coded DCI messages for each control channel are multiplexed, scrambled, and undergo QPSK modulation, layer mapping, and precoding, as shown in this figure.



Multiplexing

The blocks of coded bits for each control channel are multiplexed in order to create a block of data, as shown in this figure.



The variable M_{bits}^{i} is the number of bits in the i^{th} control channel and n_{PDCCH} is the number of control channels.

Matching PDCCHs to CCE Positions

If necessary, `<NIL>` elements are inserted in the block of bits prior to scrambling to ensure PDCCHs start at particular CCE positions and the length of the block of bits matches the amount of REGs not assigned to PCFICH or PHICH.

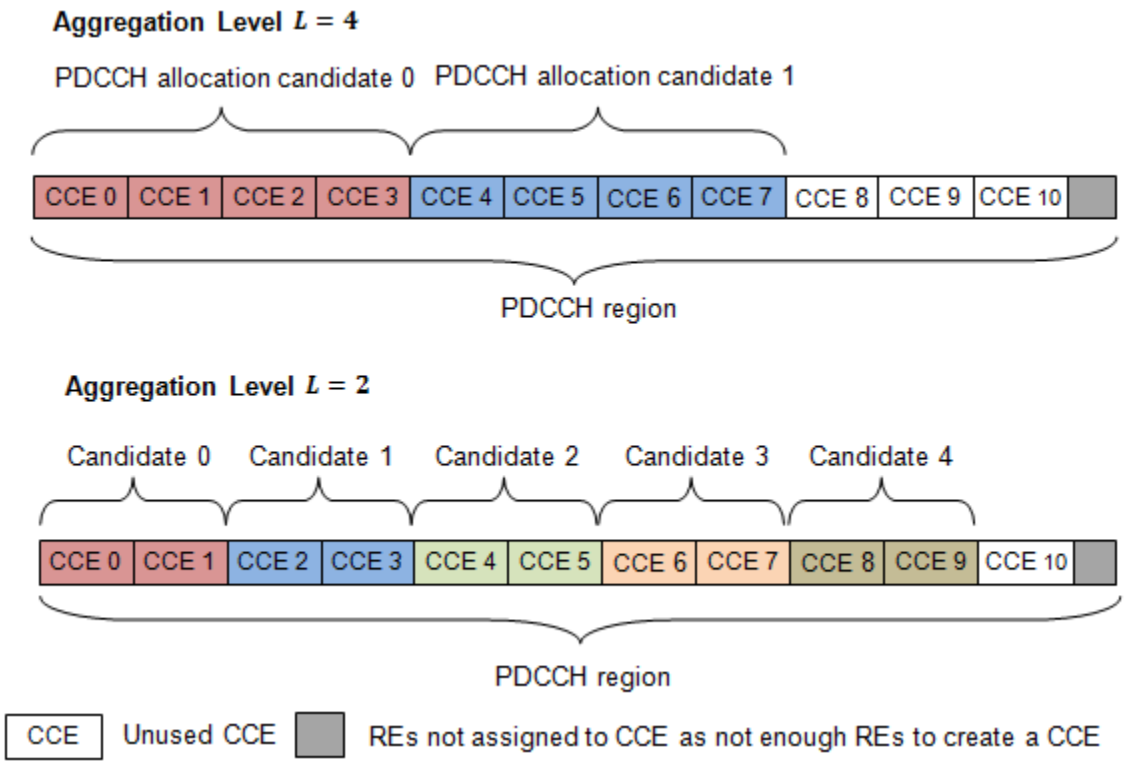
The PDCCH region consists of CCEs which could be allocated to a PDCCH. The configuration of how PDCCHs are mapped to CCEs is flexible.

Common and UE-specific PDCCHs are mapped to CCEs differently; each type has a specific set of search spaces associated with it. Each search space consists of a group of consecutive CCEs which could be allocated to a PDCCH called a PDCCH candidate. The CCE aggregation level is given by the PDCCH format and determines the number of PDCCH candidates in a search space. The number of candidates and size of the search space for each aggregation level is given in the following table.

Search space, $S_k^{(L)}$			**Number of PDCCH candidates, $M^{(L)}$**
Type	**Aggregation level**, L	**Size**, in CCEs	
UE-specific	1	6	6
	2	12	6
	4	8	2
	8	16	2
Common	4	16	4
	8	16	2

If the bandwidth is limited, not all candidates may be available because the PDCCH region is truncated.

A PDCCH can be mapped to any candidate within its suitable search space, as long as the allocated CCEs within the candidate do not overlap with a PDCCH already allocated. A simple example that shows the PDCCH candidates of two aggregation levels within a PDCCH region is shown in the following figure.



In this example, only 11 CCEs are available due to bandwidth constraints. The CCEs used to construct each PDCCH candidate are defined by the following equation.

$$L\{(Y_k + m) \bmod \lfloor N_{CCE,k}/L \rfloor\} + i$$

The preceding equation contains the following variables.

- $N_{CCE,k}$ — number of CCEs in a subframe, k
- m — number of PDCCH candidates in a given space, $m = 0, \ldots, M^{(L)} - 1$
- L — aggregation level
- i — an integer between 0 and L–1, $i = 0, \ldots, L - 1$

When a common search space is used, Y_k is 0. When a UE-specific search space is used, Y_k is given by the following equation.

$$Y_k = (AY_{k-1}) \bmod D$$

In the preceding equation, A is 39,827, D is 65,537, and Y_{-1} is the nonzero UE radio network temporary identifier.

Scrambling

This multiplexed block of bits undergoes a bit-wise exclusive-or (XOR) operation with a cell-specific scrambling sequence.

The scrambling sequence is pseudorandom, created using a length-31 Gold sequence generator and initialized using the slot number within the radio frame, n_s, and the cell ID, N_{ID}^{cell}, at the start of each subframe, as shown in the following equation.

$$c_{init} = \left\lfloor \frac{n_s}{2} \right\rfloor 2^9 + N_{ID}^{cell}$$

Scrambling serves the purpose of intercell interference rejection. When a UE descrambles a received bitstream with a known cell specific scrambling sequence, interference from other cells will be descrambled incorrectly, therefore only appearing as uncorrelated noise.

Modulation

The scrambled bits then undergo QPSK modulation to create a block of complex-valued modulation symbols.

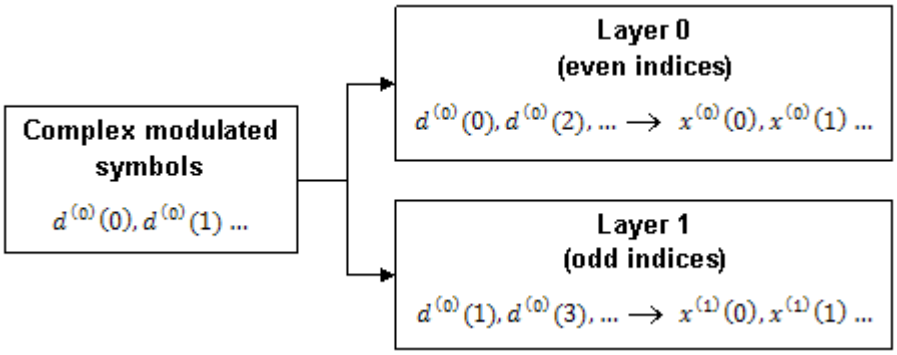
Layer Mapping

The complex symbols are mapped to one, two, or four layers depending on the number of transmit antennas used. The complex modulated input symbols, $d^{(0)}(i)$, are mapped onto v layers, $x^{(0)}(i), x^{(1)}(i), \ldots, x^{(v-1)}(i)$.

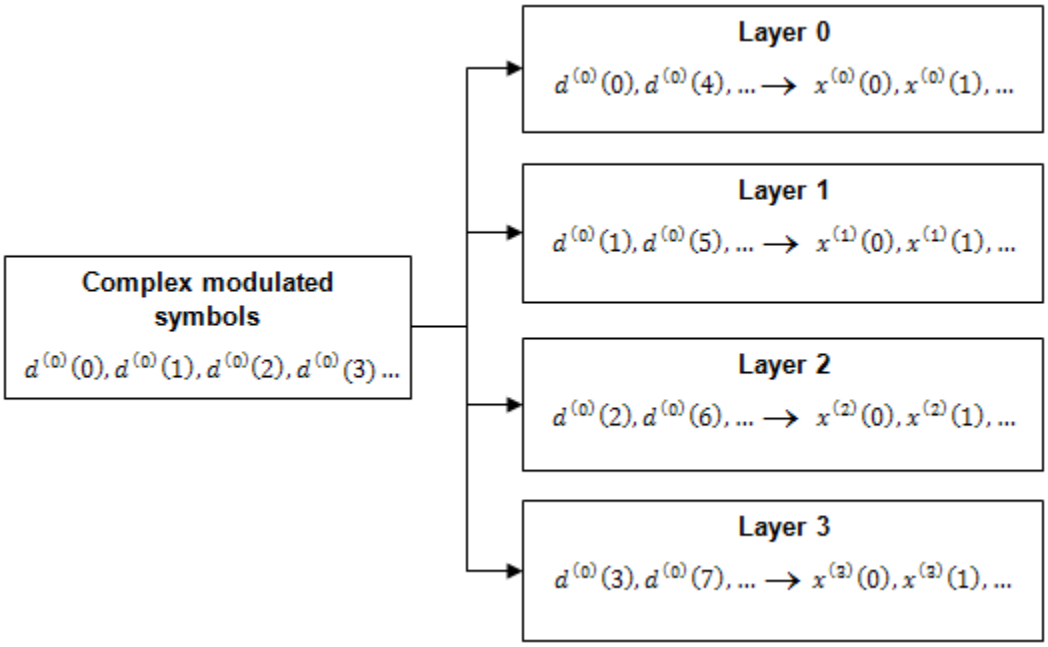
If a single antenna port is used, only one layer is used. Therefore, $x^{(0)}(i) = d^{(0)}(i)$.

If transmitter diversity is used, the input symbols are mapped to layers based on the number of layers.

- **Two Layers** — Even symbols are mapped to layer 0 and odd symbols are mapped to layer 1, as shown in this figure.



- **Four Layers** — The input symbols are mapped to layers sequentially, as shown in the following figure.



Precoding

- "Two Antenna Port Precoding" on page 1-47
- "Four Antenna Port Precoding" on page 1-48

The precoder takes a block from the layer mapper, $x^{(0)}(i), x^{(1)}(i), \ldots, x^{(v-1)}(i)$, and generates a sequence for each antenna port, $y^{(p)}(i)$. The variable p is the transmit antenna port number, and can assume values of {0}, {0,1}, or {0,1,2,3}.

For transmission over a single antenna port, no processing is carried out, as shown in this equation.

$$y^{(p)}(i) = x^{(0)}(i)$$

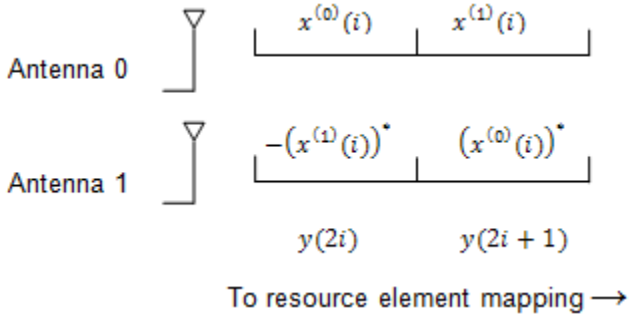
Precoding for transmit diversity is available on two or four antenna ports.

Two Antenna Port Precoding

An Alamouti scheme is used for precoding, which defines the relationship between input and output as shown in the following equation.

$$\begin{pmatrix} y^{(0)}(2i) \\ y^{(1)}(2i) \\ y^{(0)}(2i+1) \\ y^{(1)}(2i+1) \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & j & 0 \\ 0 & -1 & 0 & j \\ 0 & 1 & 0 & j \\ 1 & 0 & -j & 0 \end{pmatrix} \begin{pmatrix} \mathrm{Re}\{x^{(0)}(i)\} \\ \mathrm{Re}\{x^{(1)}(i)\} \\ \mathrm{Im}\{x^{(0)}(i)\} \\ \mathrm{Im}\{x^{(1)}(i)\} \end{pmatrix}$$

In the Alamouti scheme, two consecutive symbols, $x^{(0)}(i)$ and $x^{(1)}(i)$, are transmitted in parallel using two antennas with the following mapping, where the asterisk symbol (*) denotes the complex conjugate operation.



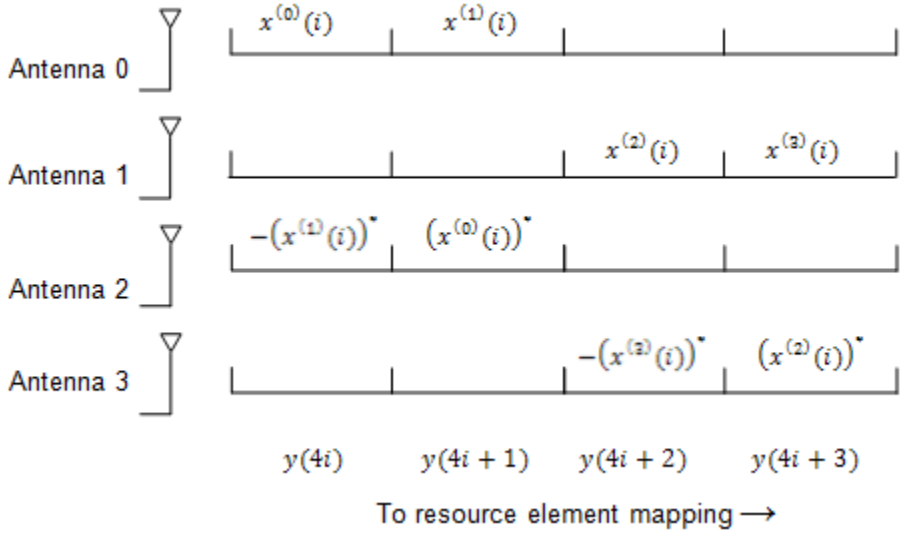
Since any two columns in the precoding matrix are orthogonal, the two symbols, $x^{(0)}(i)$ and $x^{(1)}(i)$, can be separated at the UE.

Four Antenna Port Precoding

Precoding for the four antenna port case defines the relationship between the input and output as shown in the following equation.

$$\begin{pmatrix} y^{(0)}(4i) \\ y^{(1)}(4i) \\ y^{(2)}(4i) \\ y^{(3)}(4i) \\ y^{(0)}(4i+1) \\ y^{(1)}(4i+1) \\ y^{(2)}(4i+1) \\ y^{(3)}(4i+1) \\ y^{(0)}(4i+2) \\ y^{(1)}(4i+2) \\ y^{(2)}(4i+2) \\ y^{(3)}(4i+2) \\ y^{(0)}(4i+3) \\ y^{(1)}(4i+3) \\ y^{(2)}(4i+3) \\ y^{(3)}(4i+3) \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 0 & j & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & j & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & j & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -j & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & j & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & j \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & j \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -j & 0 \end{pmatrix} \begin{pmatrix} \mathrm{Re}\{x^{(0)}(i)\} \\ \mathrm{Re}\{x^{(1)}(i)\} \\ \mathrm{Re}\{x^{(2)}(i)\} \\ \mathrm{Re}\{x^{(3)}(i)\} \\ \mathrm{Im}\{x^{(0)}(i)\} \\ \mathrm{Im}\{x^{(1)}(i)\} \\ \mathrm{Im}\{x^{(2)}(i)\} \\ \mathrm{Im}\{x^{(3)}(i)\} \end{pmatrix}$$

In this scheme, two consecutive symbols are transmitted in parallel in two symbol periods using four antennas with this mapping, where the asterisk symbol (*) denotes the complex conjugate operation.



Mapping to Resource Elements

- “Permutation” on page 1-49
- “Cyclic Shifting” on page 1-49
- “Mapping” on page 1-49

The complex valued symbols for each antenna are divided into quadruplets for mapping to resource elements. The sets of quadruplets then undergo permutation (interleaving) and cyclic shifting before being mapped to resource elements (REs) within resource-element groups (REGs).

Permutation

The blocks of quadruplets are interleaved as discussed in “Sub-block Interleaver” on page 1-42. However, instead of bits being interleaved, blocks of quadruplets are interleaved by substituting the term bit sequence with the term symbol-quadruplet sequence.

<NULL> symbols from the output of the interleaver are removed to form a sequence of interleaved quadruplets at each antenna, $w^{(p)}(i)$.

Cyclic Shifting

The interleaved sequence of quadruplets at each antenna is cyclically shifted, according to the following equation.

$$\overline{w}^{(p)}(i) = w^{(p)}(i)\left(\left(i + N_{ID}^{cell}\right)\mathrm{mod}M_{quad}\right)$$

In the preceding equation, the variable M_{quad} is the number of quadruplets, such that $M_{quad} = M_{symb}/4$, and N_{ID}^{cell} is the cell ID.

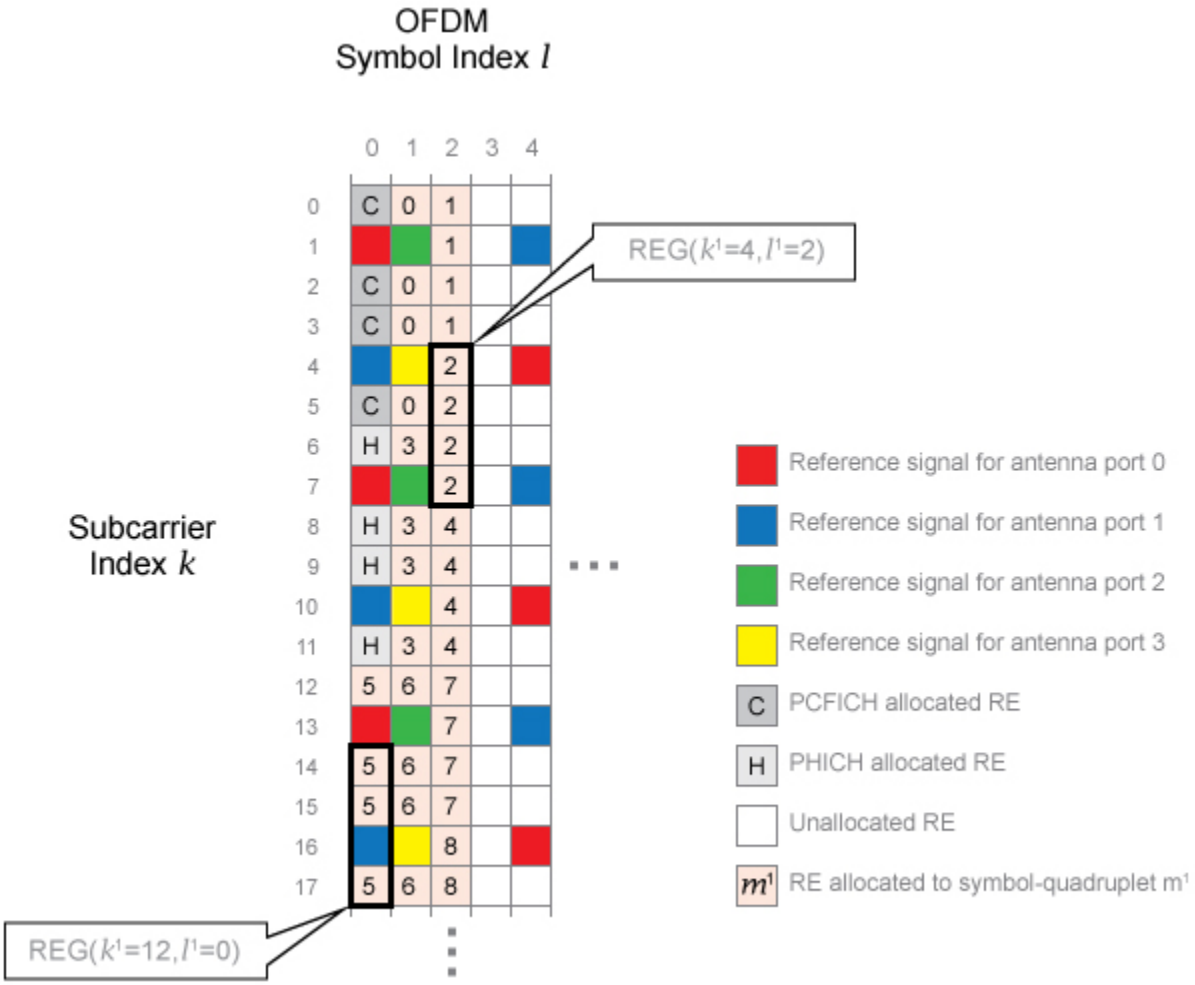
Mapping

The cyclic shifted symbol quadruplets are mapped to REGs that are not assigned to PCFICH or PHICH.

Each symbol-quadruplet is mapped to an unallocated REG in order, starting with the REG $(k' = 0, l' = 0)$. Symbol quadruplet $\overline{w}^{(p)}(m')$ maps to the REG m'. Then, the REG symbol index, l', is

incremented until all the REGs at subcarrier index $k' = 0$ have been allocated. Next, the REG subcarrier index, k', is incremented, and the process repeats. This mapping continues until all symbol quadruplets have been allocated REGs.

The mapping for an example resource grid is shown in the following figure.



Four transmit antenna ports and a control region size of three OFDM symbols are used to create the grid. In this example, the REG $(k' = 0, l' = 0)$ is allocated to PCFICH, so no symbol quadruplets are allocated to it. The symbol-quadruplets are first mapped to REG $(k' = 0, l' = 1)$, followed by $(k' = 0, l' = 2)$. Since there are no further REGs with $k' = 0$, the next REG allocated is REG $(k' = 4, l' = 2)$ because this REG has the lowest value of l' not already allocated. This process repeats to allocate all the symbol quadruplets to REGs.

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` | `lteDCIEncode` | `ltePDCCH` | `ltePDCCHIndices` | `ltePDCCHSpace` | `ltePDCCHInfo` | `ltePDCCHPRBS` | `ltePDCCHInterleave` | `ltePDCCHDeinterleave` | `ltePDCCHDecode` | `lteLayerMap` | `lteLayerDemap` | `lteDLPrecode` | `lteDLDeprecode` | `lteCRCEncode` | `lteCRCDecode` | `lteConvolutionalEncode` | `lteConvolutionalDecode` | `lteRateMatchConvolutional` | `lteRateRecoverConvolutional` | `lteSymbolModulate` | `lteSymbolDemodulate`

Related Examples

- “Model DCI and PDCCH” on page 3-9

Random Access Channel

In this section...

The Random Access Channel (RACH) is an uplink transmission used by the UE to initiate synchronization with the eNodeB.

RACH Coding

The relationship between RACH, a transport channel, and PRACH, a physical channel, as described by [2], is shown in the following table.

Transport Channel (TrCH)	Physical Channel
RACH	PRACH

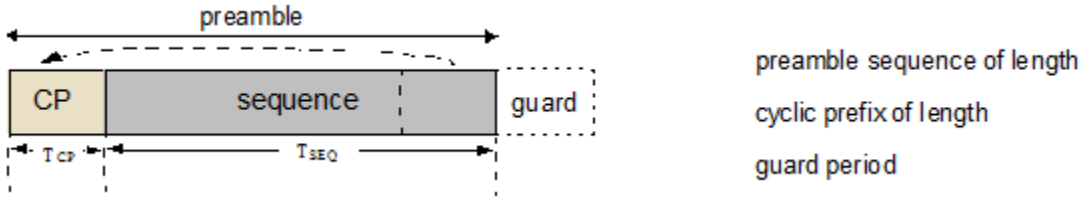
However, there are not actually any coding processes that take place to encode the RACH transport channel onto the input of the PRACH. Also, there is no logical channel which maps into the input of the RACH transport channel; the RACH originates in the MAC layer. The RACH effectively consists of a number of parameters within the MAC layer which ultimately control how and when the PRACH physical channel is generated.

The PRACH

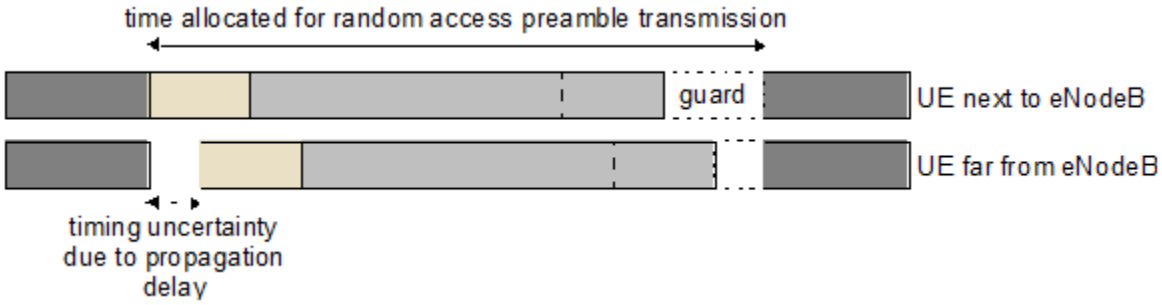
The PRACH transmission (the PRACH preamble) is an OFDM-based signal, but it is generated using a different structure from other uplink transmission; most notably it uses narrower subcarrier spacing and therefore is not orthogonal to the PUSCH, PUCCH and SRS, therefore those channels will suffer from some interference from the PRACH. However, the subcarrier spacing used by the PRACH is an integer submultiple of the spacing used for the other channels and therefore the PUSCH, PUCCH and SRS do not interfere on the PRACH.

PRACH preamble time structure

The PRACH preamble consists of a cyclic prefix, useful part of the sequence and then a guard period which is simply an unused portion of time up to the end of the last subframe occupied by the PRACH.



This guard period allows for timing uncertainty due to the UE to eNodeB distance.



Therefore the size of the guard period determines the cell radius, as any propagation delay exceeding the guard time would cause the random access preamble to overlap the following subframe at the eNodeB receiver.

The use of an OFDM transmission with cyclic prefix allows for an efficient frequency domain based receiver in the eNodeB to perform PRACH detection.

PRACH formats

There are five PRACH preamble formats which have different lengths for the cyclic prefix, useful part of the symbol, and guard period.

Preamble Format	T_{CP}	T_{SEQ}	Guard Period
0	$3{,}168\times T_s$	$24{,}576\times T_s$	$2{,}976\times T_s$
1	$21{,}024\times T_s$	$24{,}576\times T_s$	$15{,}840\times T_s$
2	$6{,}240\times T_s$	$2\times 24{,}576\times T_s$	$6{,}048\times T_s$
3	$21{,}024\times T_s$	$2\times 24{,}576\times T_s$	$21{,}984\times T_s$
4	$448\times T_s$	$4{,}096\times T_s$	$288\times T_s$

Note that Preamble Format 4 is only applicable for TDD in special subframes (subframe 1 or 6) and with Special Subframe Configuration that results in UpPTS with 2 symbols duration i.e. the Preamble Format 4 PRACH sits in UpPTS. Formats 2 and 3 have two repetitions of the nominal PRACH sequence which provides more total transmit energy and therefore allows for detection at lower SNRs. Also, Format 1 versus 0 and Format 3 versus 2 have a longer guard period, allowing for a larger cell size. The downside is that when the cyclic prefix time, sequence time and guard period are totaled up, some of the formats require multiple subframes for transmission.

Preamble Format	Number of Subframes
0	1
1	2
2	2
3	3
4	1

The penalty for using multiple subframes is a reduction in the capacity for normal uplink transmission.

PRACH Preamble Frequency Structure

As already mentioned, the PRACH uses a narrower subcarrier spacing that normal uplink transmission, specifically 1250 Hz for formats 0–3 and 7500 Hz for format 4. The ratio of the normal uplink subcarrier spacing to PRACH subcarrier spacing, K, is K=12 for formats 0–3 and K=2 for format 4.

The PRACH is designed to fit in the same bandwidth as 6 RBs of normal uplink transmission. For example, 72 subcarriers at 15,000 Hz spacing is 1.08 MHz. This makes it easy to schedule gaps in normal uplink transmission to allow for PRACH opportunities.

Therefore, there are $72\times K$ subcarriers for the PRACH, specifically 864 for formats 0–3 and 144 for format 4. As will be explained in the following subsection, the PRACH transmission for formats 0–3 uses 839 active subcarriers, and for format 4 uses 139 active subcarriers; the number of active subcarriers is denoted N_{ZC}.

As with normal uplink SC-FDMA transmission there is a half subcarrier (7500 Hz) shift, which for the PRACH is a $K/2$ subcarrier shift. A further subcarrier offset, φ (7 for formats 0–3 and 2 for format 4), centers the PRACH transmission within the 1.08 MHz bandwidth.

Preamble Format	$\varphi+K/2$	N_{ZC}	$72K-N_{ZC}-\varphi-K/2$
0–3	13	839	12
4	3	139	2

PRACH Subcarrier Content

The actual PRACH transmission is an OFDM-based reconstruction of a Zadoff-Chu sequence in the time domain. The OFDM modulator is used to position the Zadoff-Chu sequence in the frequency domain (i.e. to place the 6RBs of PRACH transmission in the 6 consecutive RBs starting from some particular physical resource block, denoted n_{PRB}^{RA} in the standard). If the output of the OFDM modulator in the time domain is to be a Zadoff-Chu sequence, the input to the OFDM modulator must be a Zadoff-Chu sequence in the frequency domain. Therefore, the active subcarriers, which total N_{ZC} in number, are set to the values of an N_{ZC}-point DFT of an N_{ZC}-sample Zadoff-Chu sequence.

PRACH Conformance Tests

Conformance tests for the PRACH, as defined in section 8.4 of [1], test the false alarm rate and detection rate of the PRACH in various environments. For a demonstration of how to perform the PRACH false alarm rate test specified in section 8.4.1, see “PRACH False Alarm Probability Conformance Test” on page 2-559. For a demonstration of how to perform the PRACH detection rate test specified in section 8.4.2, see “PRACH Detection Conformance Test” on page 2-554.

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.

See Also

`ltePRACH` | `ltePRACHInfo` | `ltePRACHDetect` | `zadoffChuSeq`

Related Examples

- “PRACH False Alarm Probability Conformance Test” on page 2-559

Uplink Control Channel Format 1

In this section...

The physical uplink control channel format 1 is a transmission channel used to carry information regarding scheduling requests in which the UE requests resources to transmit UL-SCH. It is also used to send acknowledgement responses and retransmission requests (ACK and NACK).

Uplink Control Information on PUCCH Format 1

Format 1 uplink control information (UCI) contains scheduling requests and acknowledgement responses or retransmission requests (ACK and NACK).

Channel Coding for UCI HARQ-ACK

The HARQ acknowledgement bits are received from higher layers. Depending on the number of codewords present, a HARQ acknowledgment consists of 1 or 2 information bits. A positive acknowledgement (ACK) is encoded as a binary 1, while a negative acknowledgement (NACK) is encoded as a binary 0. The HARQ-ACK bits are then processed, as required by the PUCCH.

Channel coding for UCI Scheduling Request

The scheduling request indication is received from higher layers. Zero information bits are used to request resources to transmit UL-SCH. However, the eNodeB knows when to expect a scheduling request from each UE within the cell. Therefore, if PUCCH energy is detected, the eNodeB will identify it as a scheduling request from the corresponding UE.

PUCCH Format 1, 1a, and 1b

The various PUCCH Format 1 messages used are identified by the type of control information they carry and the number of control bits they require per subframe. The three PUCCH format 1 types, their modulation schemes, and the number of information bits they use are shown in the following table.

PUCCH format	Modulation scheme	Number of bits per subframe, M_{bit}	Type of control information
1	n/a	n/a	Scheduling request
1a	BPSK	1	HARQ-ACK (1 bit)
1b	QPSK	2	HARQ-ACK (2 bits)

The bandwidth available during one subframe of a single resource block exceeds that needed for the control signaling information of a single user terminal. To make efficient use of the available

resources, the resource block can be shared by multiple user terminals. Even though the same RB is used for the PUCCH Formats 1, 1a and 1b, there is no possibility of intra-cell interference if different cyclic shifts of the same base reference sequence are used. Moreover, for PUCCH Formats 1, 1a and 1b, an extra degree of freedom is provided by applying an orthogonal cover code.

Format 1

A request for uplink resources can be made by means of the random access channel. However, due to the probability of collisions during high intensity periods, an alternative method is provided using the PUCCH Format 1.

Each UE in the cell is assigned a specific resource index mapping, a resource which can be used every nth frame to transmit a scheduling request. Therefore, if PUCCH energy is detected, the eNodeB will identify it as a scheduling request from the corresponding UE. Since each UE will have a specific resource allocated, there is no probability of a collision. However, the number of available PUCCH resources is reduced.

Format 1a and 1b

For transmission of the hybrid-ARQ acknowledgement, the HARQ ACK bits are used to generate a BPSK/QPSK symbol, depending on the number of codewords present. The modulated symbol is then used to generate the signal to be transmitted in each of the two PUCCH slots.

Demodulation Reference Signals on PUCCH Format 1

Demodulation reference signals associated with the PUCCH format 1 are used by the base station to perform channel estimation and allow for coherent demodulation of the received signal.

These reference signals are time-multiplexed with data, whereas in the downlink there is both time and frequency multiplexing. This multiplexing is performed to maintain the single-carrier nature of the SC-FDMA signal, which ensures that all data carriers are contiguous.

DRS Generation

- “Base Sequence” on page 1-58
- “DRS Grouping” on page 1-59

The demodulation reference signals are generated using a base sequence denoted by $r_{u,v}(n)$, which is discussed further in “Base Sequence” on page 1-58. More specifically, r^{PUCCH} is used to denote the PUCCH format 1 DRS sequence and is defined by the following equation.

$$r^{PUCCH}\left(m'N_{RS}^{PUCCH}M_{SC}^{RS}+mM_{SC}^{RS}+n\right)=\overline{w}(m)r_{u,v}^{(\alpha)}(n)$$

It is desired that the DRS sequences have small power variations in time and frequency, resulting in high power amplifier efficiency and comparable channel estimation quality for all frequency components. Zadoff-Chu sequences are good candidates, since they exhibit constant power in time and frequency. However, there are a limited number of Zadoff-Chu sequences; therefore, they are not suitable on their own.

The generation and mapping of the DRS associated with the PUCCH format 1 are discussed further in the following sections.

Base Sequence

The demodulation reference signals are defined by a cyclic shift, α, of a base sequence, r.

The base sequence, r, is represented in the following equation.

$$r_{u,v}^{(\alpha)} = e^{j\alpha n} r_{u,v}(n)$$

The preceding equation contains the following variables.

- $n = 0, \ldots, M_{SC}^{RS}$, where M_{SC}^{RS} is the length of the reference signal sequence.
- $U = 0, \ldots, 29$ is the base sequence group number.
- $V = 0, 1$ is the sequence number within the group and only applies to reference signals of length greater than 6 resource blocks.

A phase rotation in the frequency domain (pre-IFFT in the OFDM modulation) is equivalent to a cyclic shift in the time domain (post IFFT in the OFDM modulation). For frequency non-selective channels over the 12 subcarriers of a resource block, it is possible to achieve orthogonality between DRS generated from the same base sequence if $\alpha = {m\pi}/{6}$ for $m = 0, 1, \ldots, 11$, and assuming the DRS are synchronized in time.

The orthogonality can be exploited to transmit DRS at the same time, using the same frequency resources without mutual interference. In the PUCCH format 1 case, an extra degree of freedom can be achieved by applying an orthogonal cover code, $\overline{w}(m)$. Generally, DRS generated from different base sequences will not be orthogonal; however, they will present low cross-correlation properties.

To maximize the number of available Zadoff-Chu sequences, a prime length sequence is needed. The minimum sequence length in the UL is 12, the number of subcarriers in a resource block, which is not prime.

Therefore, Zadoff-Chu sequences are not suitable by themselves. There are effectively the following two types of base reference sequences.

- those with a sequence length ≥ 36 (spanning 3 or more resource blocks), which use a cyclic extension of Zadoff-Chu sequences
- those with a sequence length ≤ 36 (spanning 2 resource blocks), which use a special QPSK sequence

Base sequences of length ≥ three resource blocks

For sequences of length 3 resource blocks and larger (i.e. $M_{sc}^{RS} \geq 3N_{sc}^{RS}$), the base sequence is a repetition, with a cyclic offset of a Zadoff-Chu sequence of length N_{zc}^{RS}, where N_{zc}^{RS} is the largest prime such that $N_{zc}^{RS} < M_{sc}^{RS}$. Therefore, the base sequence will contain one complete length N_{zc}^{RS} Zadoff-Chu sequence plus a fractional repetition appended on the end. At the receiver the appropriate de-repetition can be done and the zero autocorrelation property will hold across the length N_{zc}^{RS} vector.

Base sequences of length ≤ three resource blocks

For sequences shorter than three resource blocks (i.e. $M_{SC}^{RS} = 12, 24$), the sequences are a composition of unity modulus complex numbers drawn from a simulation generated table. These sequences have been found through computer simulation and are specified in the LTE specifications.

DRS Grouping

There are a total of 30 sequence groups, $u \in \{0, 1, ..., 29\}$, each containing one sequence for length less than or equal to 60. This corresponds to transmission bandwidths of 1,2,3,4 and 5 resource blocks. Additionally, there are two sequences (one for $v = 0$ or 1) for length ≥ 72; corresponding to transmission bandwidths of 6 resource blocks or more.

Note that not all values of m are allowed, where m is the number of resource blocks used for transmission. Only values for m that are the product of powers of 2, 3 and 5 are valid, as shown in the following equation.

$$m = 2^{\alpha 0} \times 3^{\alpha 1} \times 5^{\alpha 2}, \text{ where } \alpha_i \text{ are positive integers}$$

The reason for this restriction is that the DFT sizes of the SC-FDMA precoding operation are limited to values which are the product of powers of 2, 3 and 5. The DFT operation can span more than one resource block, and since each resource block has 12 subcarriers, the total number of subcarriers fed to the DFT will be $12m$. Since the result of $12m$ has to be the product of powers of 2, 3 and 5 this implies that the number of resource blocks must themselves be the product of powers of 2, 3 and 5. Therefore values of m such as 7, 11, 14, 19, etc. are not valid.

For a given time slot, the uplink reference signal sequences to use within a cell are taken from one specific sequence group. If the same group is to be used for all slots then this is known as fixed assignment. On the other hand, if the group number u varies for all slots within a cell this is known as group hopping.

Fixed Group Assignment

When fixed group assignment is used, the same group number is used for all slots. For PUCCH, the group number is a function of the cell identity number modulo 30, as shown in the following equation.

$$u = N_{ID}^{cell} mod 30, \text{ with } N_{ID}^{cell} = 0, 1, ..., 503$$

Group Hopping

If group hopping is used, the pattern is applied to the calculation of the sequence group number. This pattern is the same for both the PUCCH and PUSCH.

This pattern is defined as the following equation.

$$f_{gh}(n_s) = \sum_{i=0}^{7} c(8n_s + i) \cdot 2^i mod 30$$

As shown in the preceding equation, this group hopping pattern is a function of the slot number n_s and is calculated making use of a pseudorandom binary sequence $c(k)$, generated using a length-30 Gold code. To generate the group hopping pattern, the PRBS generator is initialized with the following value at the start of each radio frame.

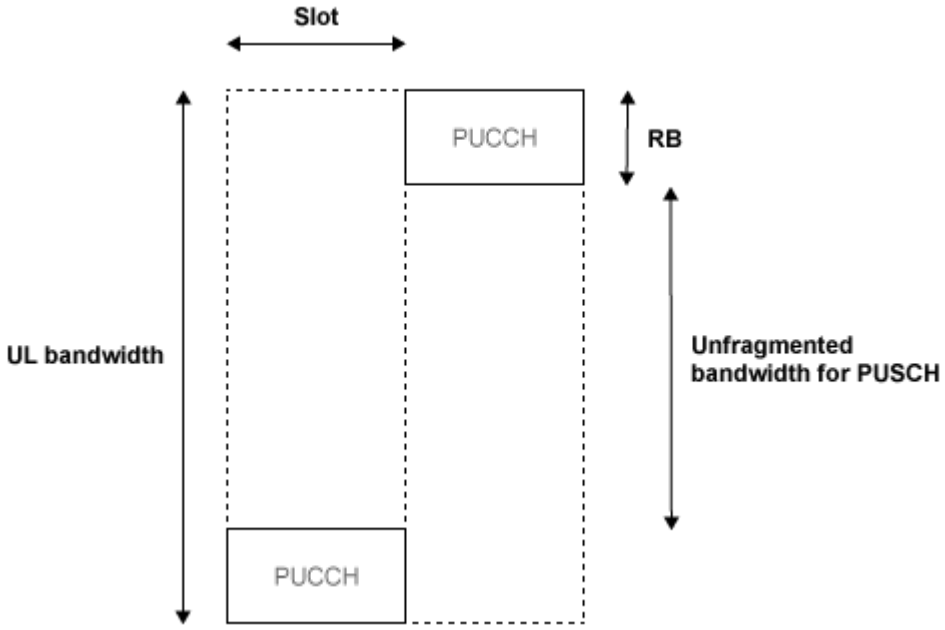
$$c_{init} = \left\lfloor \frac{N_{ID}^{cell}}{30} \right\rfloor$$

For PUCCH with group hopping, the group number, u, is given by the following equation.

$$u = \left(f_{gh}(n_s) + \left(\left(N_{ID}^{cell} \text{mod} 30\right) + \Delta_{ss}\right)\right) \text{mod} 30$$

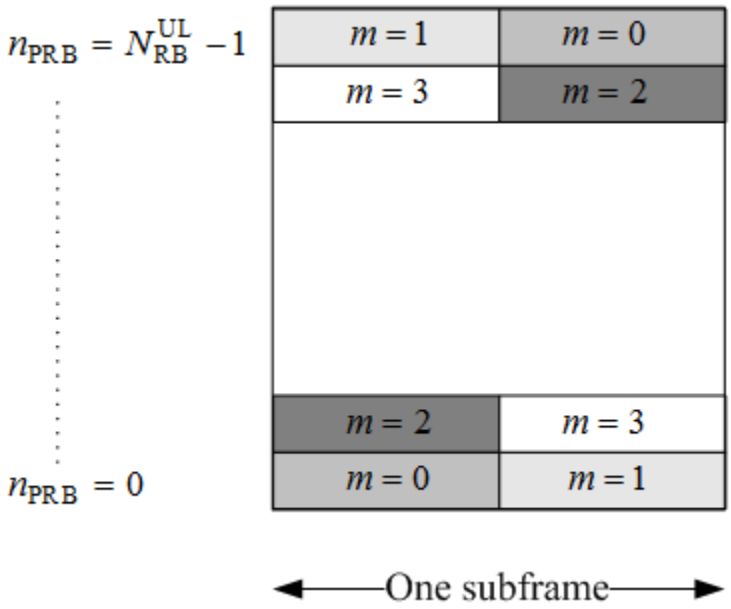
PUCCH Format 1 Resource Element Mapping

The resource blocks assigned to L1/L2 control information within a subframe are located at the edges of the total available cell bandwidth. A frequency hopping pattern is used where the lower end of the available UL spectrum is used in the first slot of the subframe and the higher end on the second; this adds a level of frequency diversity.

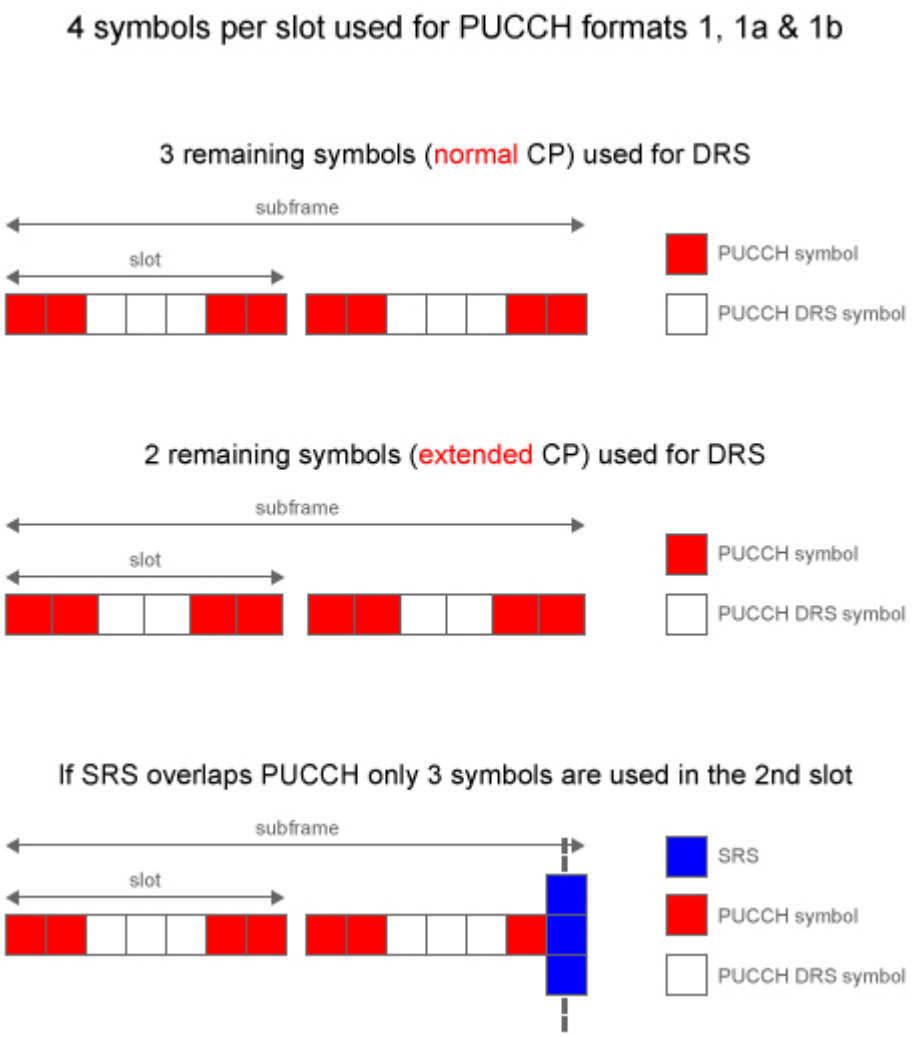


Bandwidth edges are used so that a large unfragmented portion of the spectrum remains to allocate to the PUSCH. If this spectrum was fragmented by multiple PUCCHs it would not be possible to allocate a number of contiguous RBs to a UE, hence the single carrier nature of SC-FDMA would be lost.

There is a single index, m, derived from the PUCCH resource index and other parameters that specifies the location of the PUCCH in time/frequency. When m is 0, the PUCCH occupies the lowest RB in the first slot and the highest RB in the second slot of a subframe. When m is 1, the opposite corners are used—the highest RB in the first slot and the lowest RB in the second slot. As m increases further, the allocated resource blocks move in towards the band center as shown in the following figure.



PUCCH formats 1, 1a, and 1b make use of four SC-FDMA symbols per slot. If normal cyclic prefix is used, the remaining 3 symbols, 2 for extended cyclic prefix, are used for PUCCH demodulation reference signal (DRS). If the sounding reference signal (SRS) overlaps the PUCCH symbols, only three symbols are used in the second slot of the subframe. The mapping of the symbols is illustrated in the following figure.



See Also

`ltePUCCH1` | `ltePUCCH1Indices` | `ltePUCCH1DRS` | `ltePUCCH1DRSIndices` | `lteULResourceGrid`

Related Examples

- “Model PUCCH Format 1” on page 3-12
- “PUCCH1a ACK Missed Detection Probability Conformance Test” on page 2-539
- “PUCCH1a Multi User ACK Missed Detection Probability Conformance Test” on page 2-533

Uplink Control Channel Format 2

In this section...

The Physical Uplink Control Channel (PUCCH) Format 2 is a transmission channel used to carry information regarding channel status reports as well as Hybrid Automatic Repeat request (HARQ) acknowledgements.

Uplink Control Information on PUCCH Format 2

The UE uses PUCCH format 2 control information to relay an estimate of the channel properties to the base station in order to aid channel dependent scheduling. Channel status reports include channel quality indication (CQI), rank indication (RI), and precoder matrix indication (PMI).

- **CQI** — represents the recommended modulation scheme and coding rate that should be used for the downlink transmission.
- **RI** — provides information about the rank of the channel, which is used to determine the optimal number of layers that should be used for the downlink transmission (only used for spatial multiplexed systems).
- **PMI** — provides information about which precoding matrix to use (only used in closed loop spatial multiplexing systems).

HARQ-ACK can also be transmitted with channel status information. Two forms of channel coding exist—one for the CQI alone and another for the combination of CQI with HARQ-ACK.

Channel Coding for UCI CQI Indication

The CQI codewords are coded using a (20,A) block code and are a linear combination of the 13 basis sequences denoted by $M_{i,n}$ and defined by the following equation.

$$b_i = \sum_{n=0}^{A-1} (a_n \cdot M_{i,n}) \bmod 2, \text{ where } i = 0, 1, 2, \ldots, B-1$$

The values of the basis sequence, $M_{i,n}$, for a (20,A) block code are given in the following table.

i	$M_{i,0}$	$M_{i,1}$	$M_{i,2}$	$M_{i,3}$	$M_{i,4}$	$M_{i,5}$	$M_{i,6}$	$M_{i,7}$	$M_{i,8}$	$M_{i,9}$	$M_{i,10}$	$M_{i,11}$	$M_{i,12}$
0	1	1	0	0	0	0	0	0	0	0	1	1	0
1	1	1	1	0	0	0	0	0	0	1	1	1	0
2	1	0	0	1	0	0	1	0	1	1	1	1	1
3	1	0	1	1	0	0	0	0	1	0	1	1	1

i	$M_{i,0}$	$M_{i,1}$	$M_{i,2}$	$M_{i,3}$	$M_{i,4}$	$M_{i,5}$	$M_{i,6}$	$M_{i,7}$	$M_{i,8}$	$M_{i,9}$	$M_{i,10}$	$M_{i,11}$	$M_{i,12}$
4	1	1	1	1	0	0	0	1	0	0	1	1	1
5	1	1	0	0	1	0	1	1	1	0	1	1	1
6	1	0	1	0	1	0	1	0	1	1	1	1	1
7	1	0	0	1	1	0	0	1	1	0	1	1	1
8	1	1	0	1	1	0	0	1	0	1	1	1	1
9	1	0	1	1	1	0	1	0	0	1	1	1	1
10	1	0	1	0	0	1	1	1	0	1	1	1	1
11	1	1	1	0	0	1	1	0	1	0	1	1	1
12	1	0	0	1	0	1	0	1	1	1	1	1	1
13	1	1	0	1	0	1	0	1	0	1	1	1	1
14	1	0	0	0	1	1	0	1	0	0	1	0	1
15	1	1	0	0	1	1	1	1	0	1	1	0	1
16	1	1	1	0	1	1	1	0	0	1	0	1	1
17	1	0	0	1	1	1	0	0	1	0	0	1	1
18	1	1	0	1	1	1	1	1	0	0	0	0	0
19	1	0	0	0	0	1	1	0	0	0	0	0	0

Together, the CQI, PMI, and RI form the channel status report. These indications can be brought together in a range of different configurations depending on the transmission mode of the terminal. Therefore, the total number of bits used to report the channel status condition can change, depending on the transmission format. The bit widths for the various wideband and UE selected sub-band reports can be found in sections 5.2.3.3.1 and 5.2.3.3.2 of [1], respectively.

Channel Coding for UCI CQI and HARQ-ACK

When HARQ acknowledgement responses are transmitted with the channel status report in a subframe, a different method is used. Using normal cyclic prefix length, the CQI is block coded as shown in the preceding section with the one or two HARQ-ACK bits appended to the end of the coded CQI sequence. These are coded separately to produce an 11th complex symbol which is transmitted with the PUCCH DRS for formats 2a and 2b.

When extended cyclic prefix is used, the CQI and HARQ bits are coded together. The channel quality indication is multiplexed with the one or two HARQ bits and the bits are block coded as shown in the preceding section. Bits 21 and 22 are coded separately to produce an 11th complex symbol which is transmitted with the PUCCH DRS.

PUCCH Format 2

The three PUCCH format 2 types, their modulation schemes, and the number of information bits they use are shown in the following table.

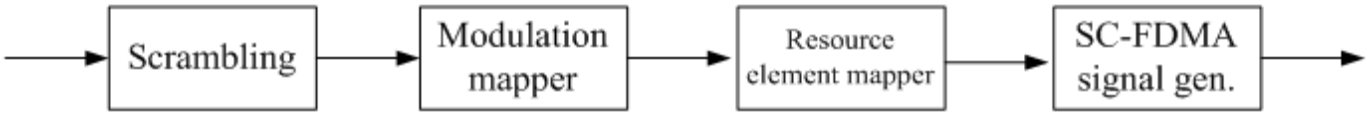
PUCCH format	Modulation scheme	Number of bits per subframe, M_{bit}	Type of control information
2	QPSK	20	Channel status reports

PUCCH format	Modulation scheme	Number of bits per subframe, M_{bit}	Type of control information
2a	QPSK + BPSK	21	Channel status reports and HARQ-ACK (1 bit)
2b	QPSK + BPSK	22	Channel status reports and HARQ-ACK (2 bits)

Format 2, 2a, and 2b

- "Scrambling" on page 1-65
- "Modulation" on page 1-65
- "Resource Element Mapping" on page 1-65

The block diagram for PUCCH format 2, 2a, and 2b is shown in the following figure.



Scrambling

A block of 20 coded UCI bits undergoes a bit-wise XOR operation with a cell-specific scrambling sequence.

The scrambling sequence is pseudorandom, created using a length-31 Gold sequence generator and initialized using the slot number within the radio frame, n_s, and the cell ID, N_{ID}^{cell}, at the start of each subframe, as shown in the following equation.

$$c_{init} = \left\lfloor \frac{n_s}{2} \right\rfloor 2^9 + N_{ID}^{cell}$$

Scrambling serves the purpose of intercell interference rejection. When a base station descrambles a received bitstream with a known cell specific scrambling sequence, interference from other cells will be descrambled incorrectly, therefore only appearing as uncorrelated noise.

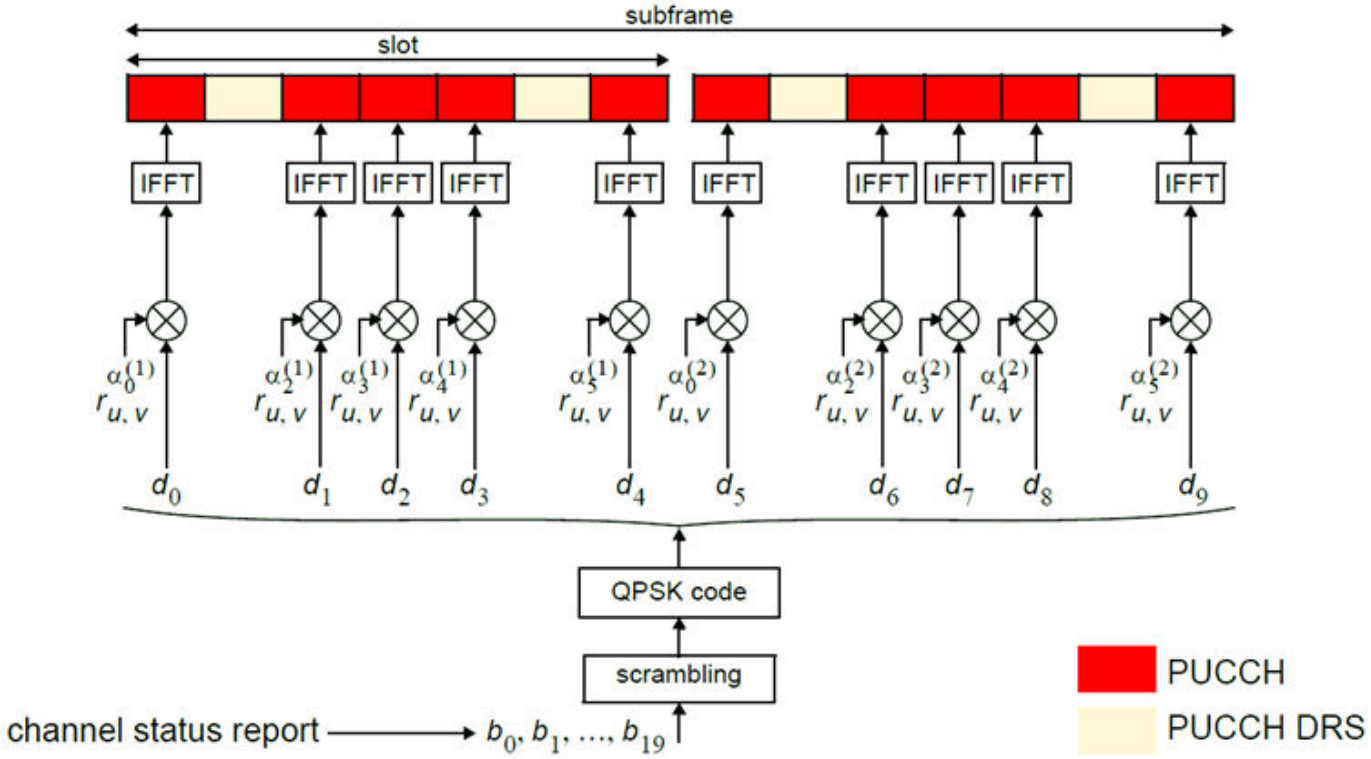
Modulation

The scrambled bits are then QPSK modulated resulting in a block of complex-valued modulation symbols. Each complex-valued symbol is multiplied with a cyclically shifted length 12 sequence.

Suppose 21 or 22 bits are available. In the case of HARQ-ACK transmission, these are coded separately to produce an 11th complex symbol that is transmitted with the PUCCH DRS for formats 2a and 2b. As in PUCCH Formats 1, 1a, and 1b, a hopping pattern is applied to the cyclic shift to randomize intercell interference. PUCCH Formats 2a and 2b are supported for normal cyclic prefix only.

Resource Element Mapping

The final stage in the PUCCH format 2 processing involves mapping to resource elements. The complete processing chain for normal cyclic prefix, including the position occupied by the PUCCH format 2 in a subframe and in each slot, is shown in the following figure.



The cyclic shifted sequence applied to randomize intercell interference is denoted here by $r_{u,v}$. For extended cyclic prefix, where there are only six SC-FDMA symbols per slot, the mapping to resources changes slightly. In this case, only one reference signal is transmitted per slot, and the signal occupies the third symbol in each slot.

Demodulation Reference Signals on PUCCH Format 2

Demodulation reference signals associated with the PUCCH format 2 are used by the base station to perform channel estimation and allow for coherent demodulation of the received signal.

These reference signals are time-multiplexed with data, whereas in the downlink there is both time and frequency multiplexing. This multiplexing is performed to maintain the single-carrier nature of the SC-FDMA signal, which ensures that all data carriers are contiguous.

DRS Generation

- “Base Sequence” on page 1-67
- “DRS Grouping” on page 1-68

The demodulation reference signals are generated using a base sequence denoted by $r_{u,v}(n)$, which is discussed further in “Base Sequence” on page 1-67. More specifically, r^{PUCCH} is used to denote the PUCCH format 2 DRS sequence and is defined by the following equation.

$$r^{PUCCH}\left(m'N_{RS}^{PUCCH}M_{SC}^{RS}+mM_{SC}^{RS}+n\right)=\overline{w}(m)r_{u,v}^{(\alpha)}(n)$$

It is desired that the DRS sequences have small power variations in time and frequency, resulting in high power amplifier efficiency and comparable channel estimation quality for all frequency

components. Zadoff-Chu sequences are good candidates, since they exhibit constant power in time and frequency. However, there are a limited number of Zadoff-Chu sequences; therefore, they are not suitable on their own.

The generation and mapping of the DRS associated with the PUCCH format 2 are discussed further in the following sections.

Base Sequence

The demodulation reference signals are defined by a cyclic shift, α, of a base sequence, r.

The base sequence, r, is represented in the following equation.

$$r_{u,v}^{(\alpha)} = e^{j\alpha n} r_{u,v}(n)$$

The preceding equation contains the following variables.

- $n = 0, \ldots, M_{SC}^{RS}$, where M_{SC}^{RS} is the length of the reference signal sequence.
- $U = 0, \ldots, 29$ is the base sequence group number.
- $V = 0, 1$ is the sequence number within the group and only applies to reference signals of length greater than 6 resource blocks.

A phase rotation in the frequency domain (pre-IFFT in the OFDM modulation) is equivalent to a cyclic shift in the time domain (post IFFT in the OFDM modulation). For frequency non-selective channels over the 12 subcarriers of a resource block, it is possible to achieve orthogonality between DRS generated from the same base sequence if $\alpha = {m\pi}/{6}$ for $m = 0, 1, \ldots, 11$, and assuming the DRS are synchronized in time.

To maximize the number of available Zadoff-Chu sequences, a prime length sequence is needed. The minimum sequence length in the UL is 12, the number of subcarriers in a resource block, which is not prime.

Therefore, Zadoff-Chu sequences are not suitable by themselves. There are effectively the following two types of base reference sequences.

- those with a sequence length ≥ 36 (spanning 3 or more resource blocks), which use a cyclic extension of Zadoff-Chu sequences
- those with a sequence length ≤ 36 (spanning 2 resource blocks), which use a special QPSK sequence

Base sequences of length ≥ three resource blocks

For sequences of length 3 resource blocks and larger (i.e. $M_{sc}^{RS} \geq 3N_{sc}^{RS}$), the base sequence is a repetition, with a cyclic offset of a Zadoff-Chu sequence of length N_{zc}^{RS}, where N_{zc}^{RS} is the largest prime such that $N_{zc}^{RS} < M_{sc}^{RS}$. Therefore, the base sequence will contain one complete length N_{zc}^{RS} Zadoff-Chu sequence plus a fractional repetition appended on the end. At the receiver the appropriate de-repetition can be done and the zero autocorrelation property will hold across the length N_{zc}^{RS} vector.

Base sequences of length ≤ three resource blocks

For sequences shorter than three resource blocks (i.e. $M_{SC}^{RS} = 12, 24$), the sequences are a composition of unity modulus complex numbers drawn from a simulation generated table. These sequences have been found through computer simulation and are specified in the LTE specifications.

DRS Grouping

There are a total of 30 sequence groups, $u \in \{0, 1, ..., 29\}$, each containing one sequence for length less than or equal to 60. This corresponds to transmission bandwidths of 1,2,3,4 and 5 resource blocks. Additionally, there are two sequences (one for $v = 0$ or 1) for length ≥ 72; corresponding to transmission bandwidths of 6 resource blocks or more.

Note that not all values of m are allowed, where m is the number of resource blocks used for transmission. Only values for m that are the product of powers of 2, 3 and 5 are valid, as shown in the following equation.

$$m = 2^{\alpha 0} \times 3^{\alpha 1} \times 5^{\alpha 2}, \text{ where } \alpha_i \text{ are positive integers}$$

The reason for this restriction is that the DFT sizes of the SC-FDMA precoding operation are limited to values which are the product of powers of 2, 3 and 5. The DFT operation can span more than one resource block, and since each resource block has 12 subcarriers, the total number of subcarriers fed to the DFT will be $12m$. Since the result of $12m$ has to be the product of powers of 2, 3 and 5 this implies that the number of resource blocks must themselves be the product of powers of 2, 3 and 5. Therefore values of m such as 7, 11, 14, 19, etc. are not valid.

For a given time slot, the uplink reference signal sequences to use within a cell are taken from one specific sequence group. If the same group is to be used for all slots then this is known as fixed assignment. On the other hand, if the group number u varies for all slots within a cell this is known as group hopping.

Fixed Group Assignment

When fixed group assignment is used, the same group number is used for all slots. For PUCCH, the group number is a function of the cell identity number modulo 30, as shown in the following equation.

$$u = N_{ID}^{cell} mod 30, \text{ with } N_{ID}^{cell} = 0, 1, ..., 503$$

Group Hopping

If group hopping is used, the pattern is applied to the calculation of the sequence group number. This pattern is the same for both the PUCCH and PUSCH.

This pattern is defined as the following equation.

$$f_{gh}(n_s) = \sum_{i=0}^{7} c(8n_s + i) \cdot 2^i mod 30$$

As shown in the preceding equation, this group hopping pattern is a function of the slot number n_s and is calculated making use of a pseudorandom binary sequence $c(k)$, generated using a length-30 Gold code. To generate the group hopping pattern, the PRBS generator is initialized with the following value at the start of each radio frame.

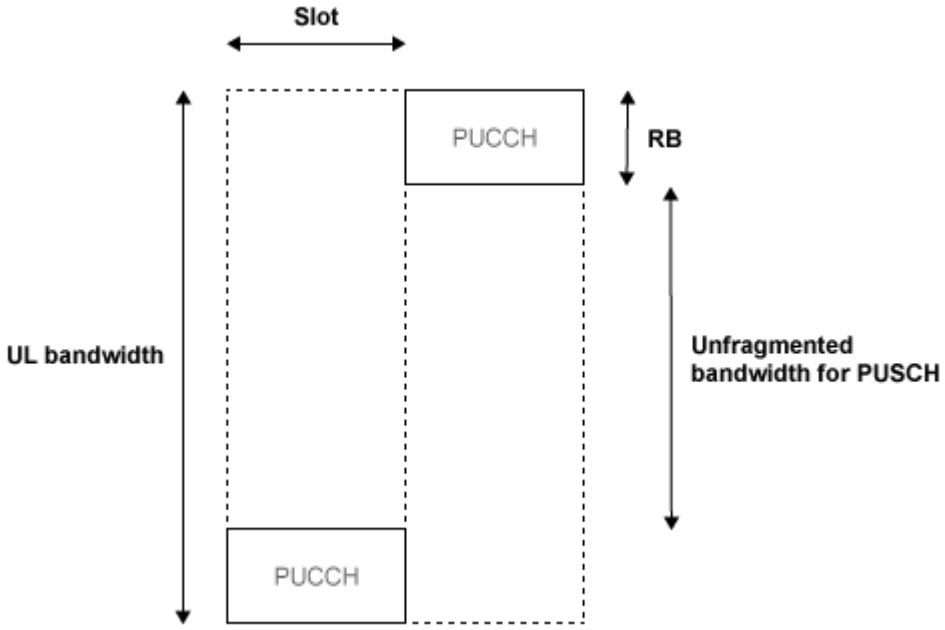
$$c_{init} = \left\lfloor \frac{N_{ID}^{cell}}{30} \right\rfloor$$

For PUCCH with group hopping, the group number, u, is given by the following equation.

$$u = \left(f_{gh}(n_s) + \left(\left(N_{ID}^{cell} \text{mod} 30\right) + \Delta_{ss}\right)\right) \text{mod} 30$$

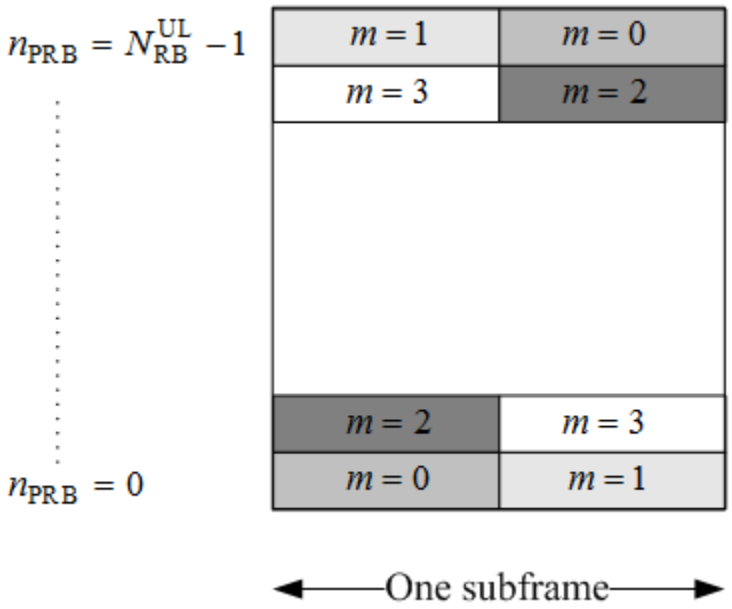
PUCCH Format 2 Resource Element Mapping

The resource blocks assigned to L1/L2 control information within a subframe are located at the edges of the total available cell bandwidth. A frequency hopping pattern is used where the lower end of the available UL spectrum is used in the first slot of the subframe and the higher end on the second; this adds a level of frequency diversity.



Bandwidth edges are used so that a large unfragmented portion of the spectrum remains to allocate to the PUSCH. If this spectrum was fragmented by multiple PUCCHs it would not be possible to allocate a number of contiguous RBs to a UE, hence the single carrier nature of SC-FDMA would be lost.

There is a single index, m, derived from the PUCCH resource index and other parameters that specifies the location of the PUCCH in time/frequency. When m is 0, the PUCCH occupies the lowest RB in the first slot and the highest RB in the second slot of a subframe. When m is 1, the opposite corners are used—the highest RB in the first slot and the lowest RB in the second slot. As m increases further, the allocated resource blocks move in towards the band center as shown in the following figure.



The resource for PUCCH formats 2, 2a, and 2b is indicated by a single scalar value called resource index. From this value, the cyclic shift can be determined. Time and frequency allocated resources are not derived from this value. However, higher layers are in full control of when and where control information is transmitted.

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

`ltePUCCH2` | `ltePUCCH2Indices` | `ltePUCCH2DRS` | `ltePUCCH2DRSIndices` | `lteULResourceGrid`

Related Examples

- "Model PUCCH Format 2" on page 3-14

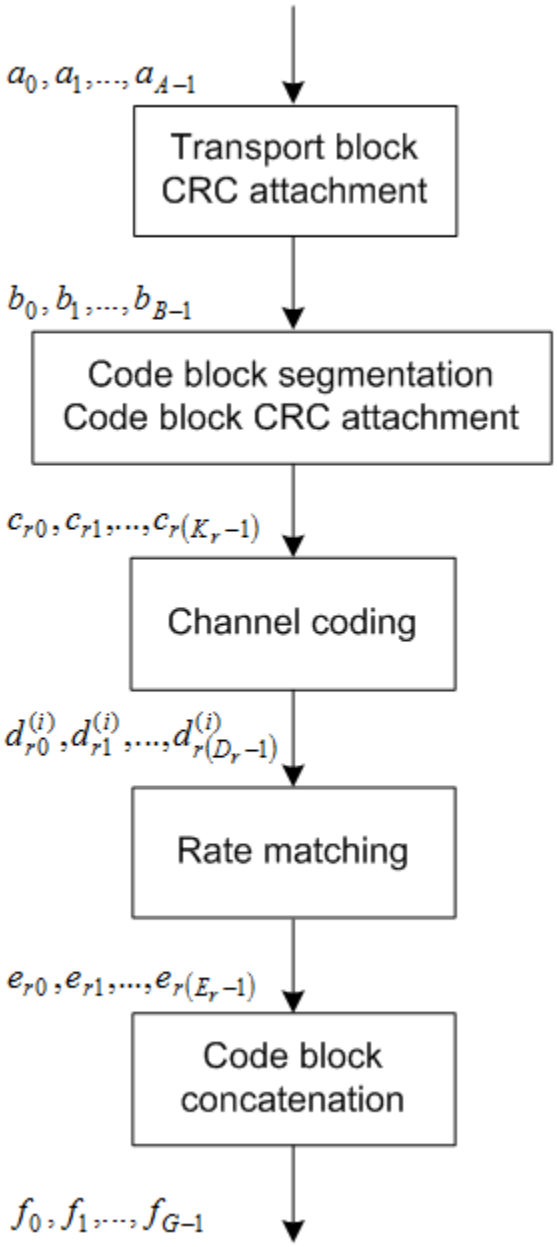
Downlink Shared Channel

In this section...

The physical downlink shared channel (PDSCH) is used to transmit the downlink shared channel (DL-SCH). The DL-SCH is the transport channel used for transmitting downlink data (a transport block).

DL-SCH Coding

To create the PDSCH payload, a transport block of length A, denoted by $a_0, a_1, \ldots, a_{A-1}$, undergoes transport block CRC attachment, code block segmentation and code block CRC attachment, channel coding, rate matching and code block concatenation. The coding steps are illustrated in this block diagram.



Transport Block CRC Attachment

A cyclic redundancy check (CRC) is used for error detection in transport blocks. The entire transport block is used to calculate the CRC parity bits. The transport block is divided by a cyclic generator polynomial, described as g_{CRC24A} in section 5.1.1 of [1], to generate 24 parity bits. These parity bits are then appended to the end of the transport block.

Code Block Segmentation and CRC Attachment

The input block of bits to the code segmentation block is denoted by $b_0, b_1, \ldots, b_{B-1}$, where $B = A + 24$. In LTE, a minimum and maximum code block size is specified, so the block sizes are compatible with the block sizes supported by the turbo interleaver.

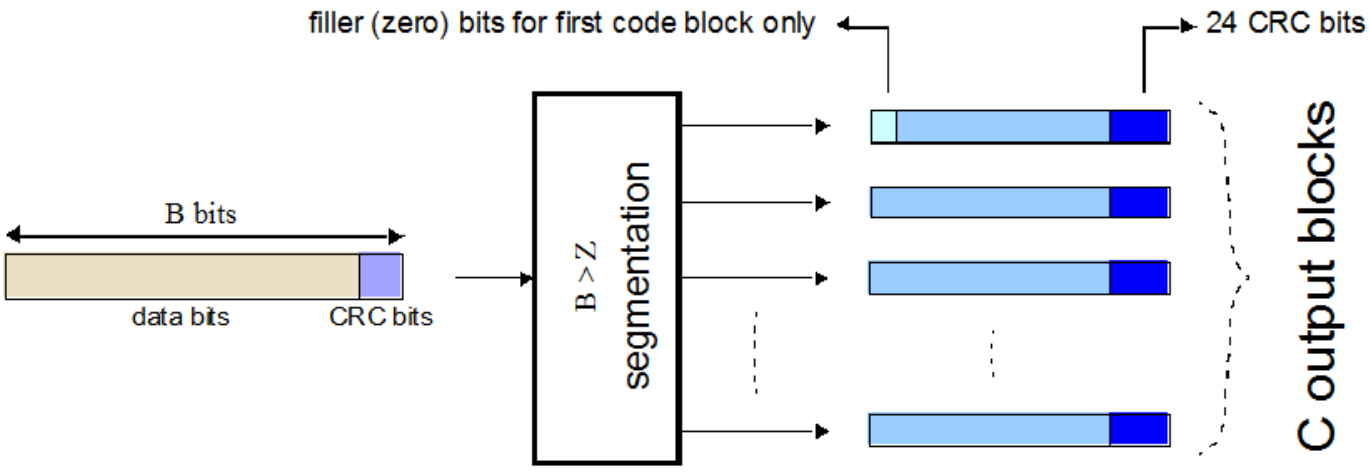
- The minimum code block size is 40 bits
- The maximum code block size, denoted by Z, is 6144 bits

If the length of the input block, B, is greater than the maximum code block size, the input block is segmented.

When the input block is segmented, it is divided into $C = [B/(Z - L)]$ smaller blocks, where L is 24. Therefore, $C = [B/6120]$ code blocks.

Each code block has a 24-bit CRC attached to the end, calculated as described in “Transport Block CRC Attachment” on page 1-72, but the generator polynomial, described as g_{CRC24B} in section 5.1.1 of [1] is used.

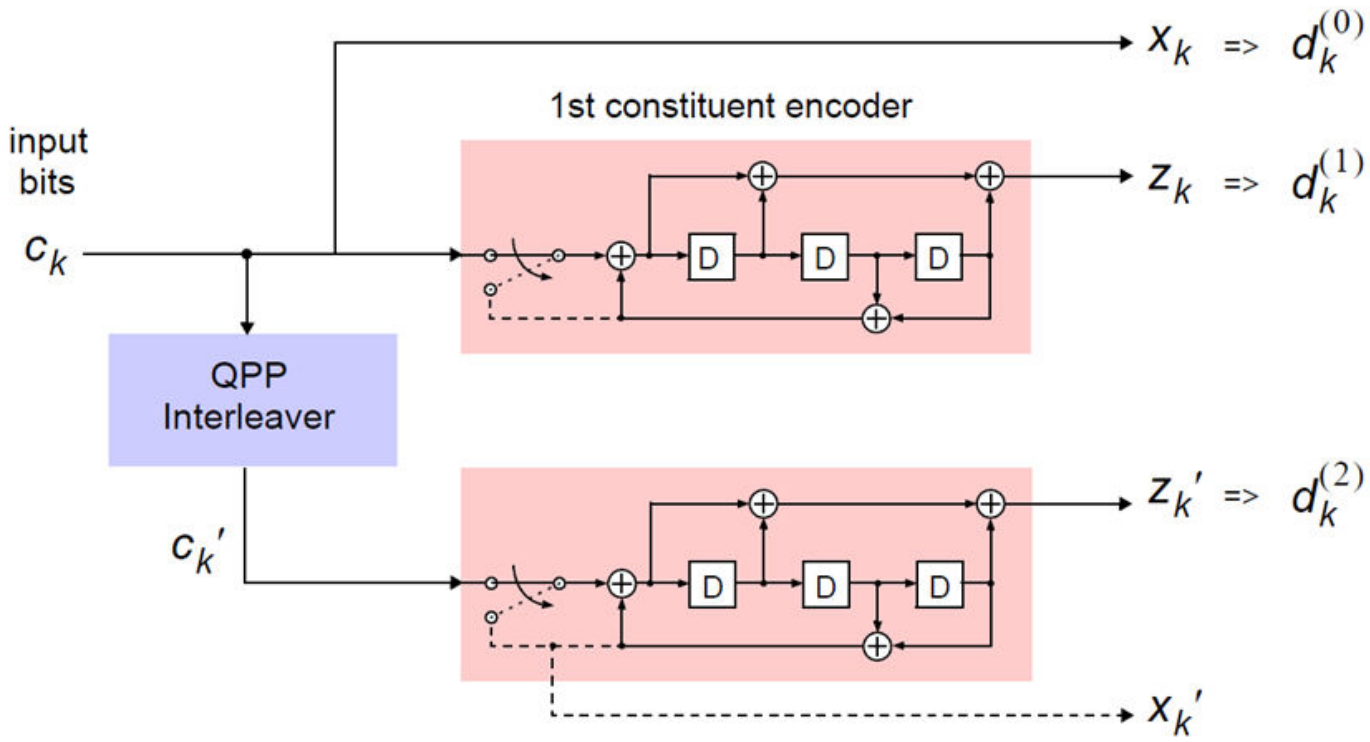
If required, the algorithm appends filler bits to the start of the segment so that the code block sizes match a set of valid turbo interleaver block sizes. The process is shown in this figure.



If no segmentation is needed, only one code block is produced. If B is less than the minimum size, filler bits (zeros) are added to the beginning of the code block to achieve a total of 40 bits.

Channel Coding

The code blocks undergo turbo coding. Turbo coding is a form of forward error correction that improves the channel capacity by adding redundant information. The turbo encoder scheme used is a Parallel Concatenated Convolutional Code (PCCC) with two recursive convolutional coders and a “contention-free” Quadratic Permutation Polynomial (QPP) interleaver, as shown in this figure.



The output of the encoder is three streams, $d_k^{(0)}$, $d_k^{(1)}$, and $d_k^{(2)}$, to achieve a code rate of 1/3.

Constituent Encoders

The input to the first constituent encoder is the input bit stream to the turbo coding block. The input to the second constituent encoder is the output of the QPP interleaver, a permutation of the input sequence.

Each encoder outputs two sequences, called the systematic sequence and the parity sequence. The systematic sequences are denoted by (x_k, x'_k) and the parity sequences are denoted by (z_k, z'_k). Only one of the systematic sequences (x_k) is used as an output because the other (x'_k) is simply a permutation of the chosen systematic sequence. The transfer function for each constituent encoder is given by the following equation.

$$G(D) = \left[1, \frac{g_1(D)}{g_0(D)}\right]$$

The first element, 1, represents the systematic output transfer function. The second element, $\left(\frac{g_1(D)}{g_0(D)}\right)$, represents the recursive convolutional output transfer function.

$$g_0(D) = 1 + D^2 + D^3$$

$$g_1(D) = 1 + D + D^3$$

The output for each sequence can be calculated using the transfer function.

The encoder is initialized with all zeros. If the code block to be encoded is the 0-th and filler bits (F) are used, the input to the encoder (c_k) is set to zero and the output (x_k) and (z_k) set to <NULL> for $k = 0, ..., F - 1$.

Trellis Termination for Turbo Encoder

In a normal convolutional coder, the coder is driven to an all zeros state upon termination by appending zeros to the end of the input data stream. Since the decoder knows the start and end state of the encoder, it can decode the data. Driving a recursive coder to an all zeros state using this method is not possible. To overcome this problem, trellis termination is used.

Upon termination, the tail bits are fed back to the input of each encoder using a switch. The first three tail bits are used to terminate each encoder.

The QPP Interleaver

The role of the interleaver is to spread the information bits such that in the event of a burst error, the two code streams are affected differently, allowing data to still be recovered.

The output of the interleaver is a permutation of the input data, as shown in these equations.

$$c'_i = c_{\Pi(i)}, i = 0, 1, ..., (K - 1)$$

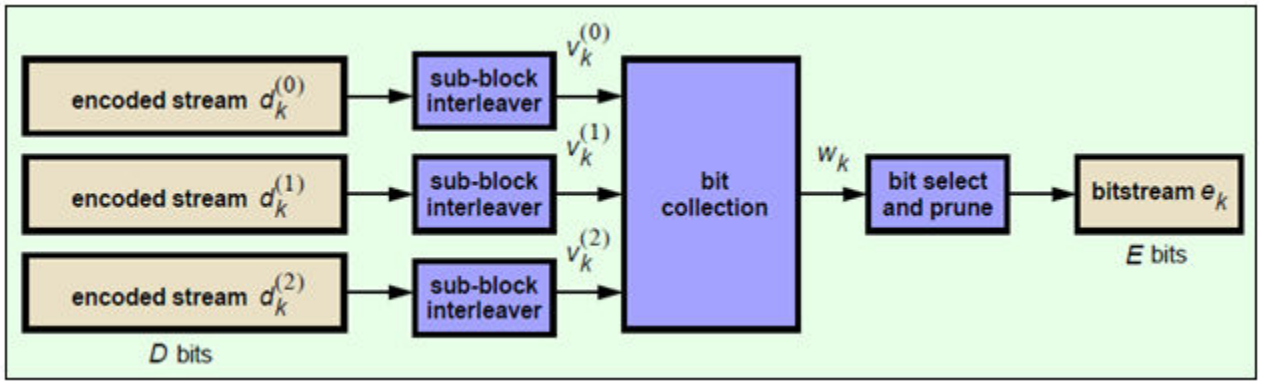
$$\Pi(i) = \left(f_1 i + f_2 i^2\right) \mathrm{mod} K$$

The variable K is the input length. The variables f_1 and f_2 are coefficients chosen depending on K, in table 5.1.3-3 of [1]. For example, K=40, f_1=3, and f_2=10, yields the following sequence.

$$\Pi(i) = 0, 13, 6, 19, 12, 25, 18, 31, 24, 37, 30, 3, 36, 9, 2, 15, 8, 21, 14, 27, 20, 33, 26, 39, 32, 5, 38, ...$$

Rate Matching

The rate matching block creates an output bitstream with a desired code rate. Since the number of bits available for transmission depends on the available resources, the rate matching algorithm is capable of producing any rate. The three bitstreams from the turbo encoder are interleaved, followed by bit collection, to create a circular buffer. Bits are selected and pruned from the buffer to create an output bitstream with the desired code rate. The process is illustrated in this figure.



Sub-block Interleaver

The three sub-block interleavers used in the rate matching block are identical. An interleaver permutes the bits it receives. This makes burst errors easier to correct by preventing consecutive corrupt bits.

The sub-block interleaver reshapes the encoded bit sequence, row-by-row, to form a matrix with $C_{Subblock}^{TC} = 32$ columns and $R_{Subblock}^{TC}$ rows. The variable $R_{Subblock}^{TC}$ is determined by finding the minimum integer such that the number of encoded input bits is $D \le \left(R_{Subblock}^{TC} \times C_{Subblock}^{TC}\right)$. If $\left(R_{Subblock}^{TC} \times C_{Subblock}^{TC}\right) > D$, N_D <NULL>'s are appended to the front of the encoded sequence. In this case, $N_D + D = \left(R_{Subblock}^{TC} \times C_{Subblock}^{TC}\right)$.

For blocks $d_k^{(0)}$ and $d_k^{(1)}$, inter-column permutation is performed on the matrix to reorder the columns as shown in this pattern.

0, 16, 8, 24, 4, 20, 12, 28, 2, 18, 10, 26, 6, 22, 14, 30, 1, 17, 9, 25, 5, 21, 13, 29, 3, 19, 11, 27, 7, 23, 15, 31

The output of the block interleaver for blocks $d_k^{(0)}$ and $d_k^{(1)}$ is the bit sequence read out column-by-column from the inter-column permuted matrix to create a stream $K_\Pi = \left(R_{Subblock}^{TC} \times C_{Subblock}^{TC}\right)$ bits long.

For block $d_k^{(2)}$, the elements of the matrix are permuted separately based on the permutation pattern shown above, but modified to create a permutation which is a function of the variables $R_{Subblock}^{TC}$, $C_{Subblock}^{TC}$, k, and K_Π. This process creates three interleaved bitstreams.

$$d_k^{(0)} \rightarrow v_k^{(0)}$$

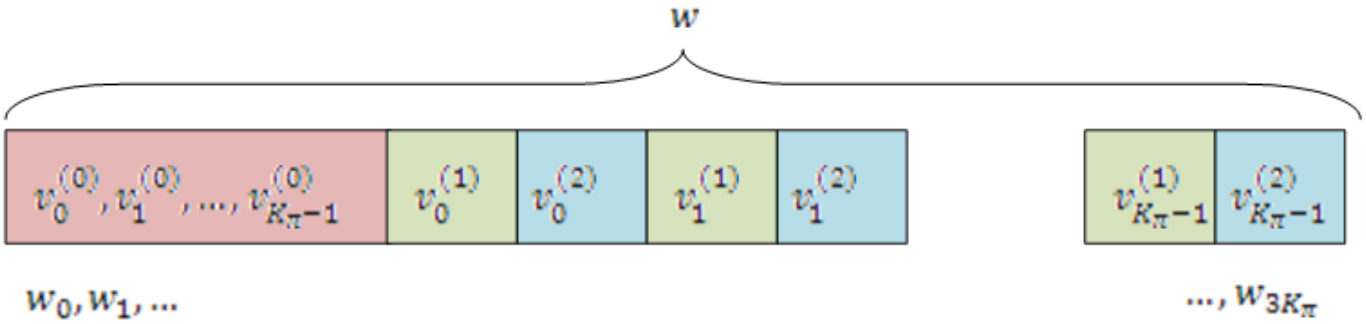
$$d_k^{(1)} \rightarrow v_k^{(1)}$$

$$d_k^{(2)} \rightarrow v_k^{(2)}$$

Bit Collection, Selection, and Transmission

The bit collection stage creates a virtual circular buffer by combining the three interleaved encoded bit streams.

The sequences $v_k^{(1)}$ and $v_k^{(2)}$ are combined by interlacing successive values from each sequence. This combination is then appended to the end of $v_k^{(0)}$ to create the circular buffer w_k shown in this figure.



Interlacing allows equal levels of protection for each parity sequence.

The algorithm then selects and prunes bits from the circular buffer to create an output sequence length that meets the desired code rate.

The Hybrid Automatic Repeat Request (HARQ) error correction scheme is incorporated into the rate-matching algorithm of LTE. For any desired code rate the coded bits are output serially from the circular buffer from a starting location, given by the redundancy version (RV), wrapping around to the beginning of the buffer if the end of the buffer is reached. `NULL` bits are discarded. Different RVs, and hence starting points, allow for the retransmission of selected data. The ability to select different starting points enables the following two main methods of recombining data at the receiver in the HARQ process.

- Chase combining — retransmissions contain the same data and parity bit.
- Incremental redundancy — retransmissions contain different information, so the receiver gains knowledge on each retransmission.

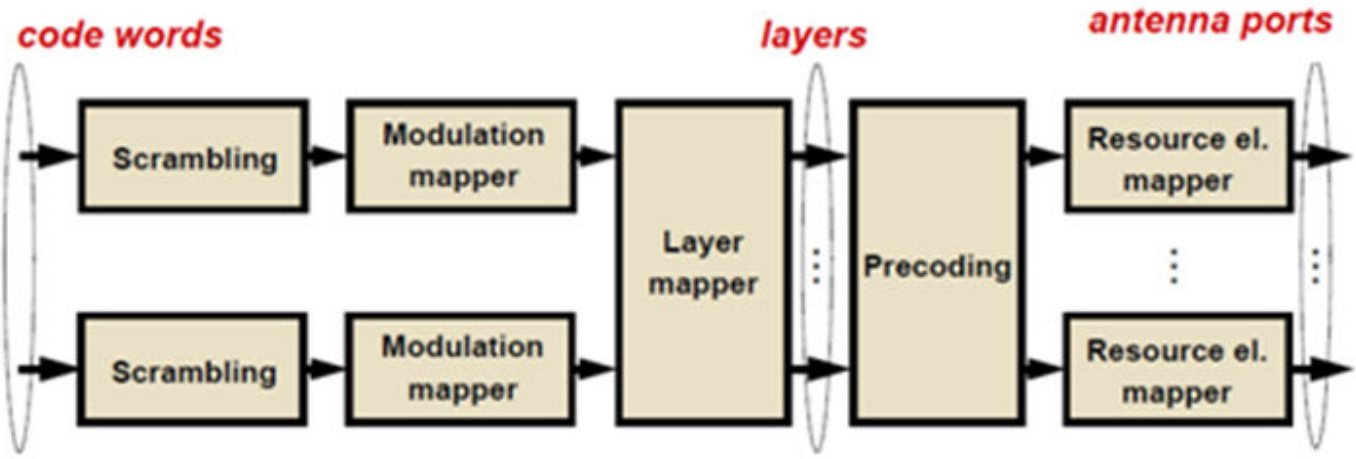
Code Block Concatenation

At this stage, the rate-matched code blocks are combined again. This is done by sequentially concatenating the blocks to create the output of the channel coding, f_k for $k = 0, \ldots, G-1$.

PDSCH Processing

- “PDSCH Scrambling” on page 1-78
- “Modulation” on page 1-78
- “Layer Mapping” on page 1-79
- “Precoding” on page 1-82
- “Valid Codeword, Layer and Precoding Scheme Combinations” on page 1-87
- “Mapping to Resource Elements” on page 1-88

One or two coded transport blocks (codewords) can be transmitted simultaneously on the PDSCH, depending on the precoding on page 1-82 scheme used. The DL-SCH codewords undergo scrambling, modulation, layer mapping, precoding and resource element mapping as shown in this figure.



PDSCH Scrambling

The codewords are bit-wise multiplied with an orthogonal sequence and a UE-specific scrambling sequence to create the following sequence of symbols for each codeword, q.

$$\tilde{b}^{(q)}(0), \ldots, \tilde{b}^{(q)}\left(M_{bit}^{(q)} - 1\right)$$

The variable $M_{bit}^{(q)}$ is the number of bits in codeword q.

The scrambling sequence is pseudo-random, created using a length-31 Gold sequence generator and initialized at the start of each subframe using four parameters:

- the slot number within the radio network temporary identifier associated with the PDSCH transmission, n_{RNTI}
- the cell ID, N_{ID}^{cell}
- the slot number within the radio frame, n_s
- the codeword index, $q = \{0, 1\}$.

The initial scrambler state is given by this equation.

$$c_{init} = n_{RNTI} \times 2^{14} + q \times 2^{13} + \left\lfloor \frac{n_s}{2} \right\rfloor \times 2^9 + N_{ID}^{cell}$$

Scrambling with a cell-specific sequence serves the purpose of intercell interference rejection. When a UE descrambles a received bitstream with a known cell-specific scrambling sequence, interference from other cells will be descrambled incorrectly and therefore only appear as uncorrelated noise.

Modulation

The scrambled codewords undergo QPSK, 16-QAM, 64-QAM, or 256-QAM modulation. This choice creates flexibility to allow the scheme to maximize the data transmitted depending on the channel conditions.

Layer Mapping

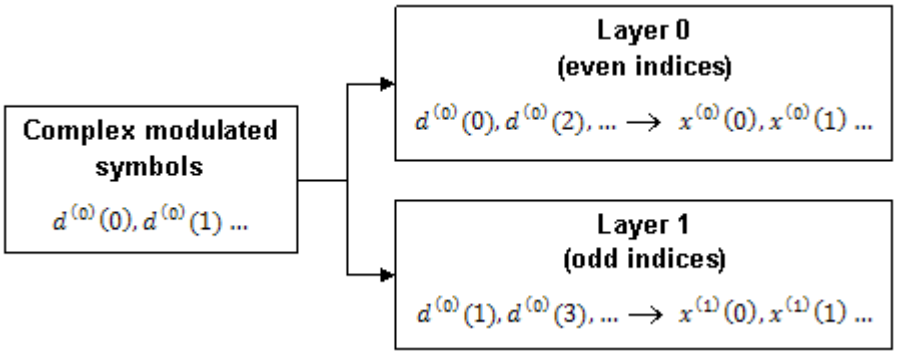
The complex symbols are mapped to one, two, or four layers depending on the number of transmit antennas used. The complex modulated input symbols, $d(i)$, are mapped onto v layers, $x^{(0)}(i), x^{(1)}(i), \ldots, x^{(v-1)}(i)$.

If a single antenna port is used, only one layer is used. Therefore, $x^{(0)}(i) = d^{(0)}(i)$.

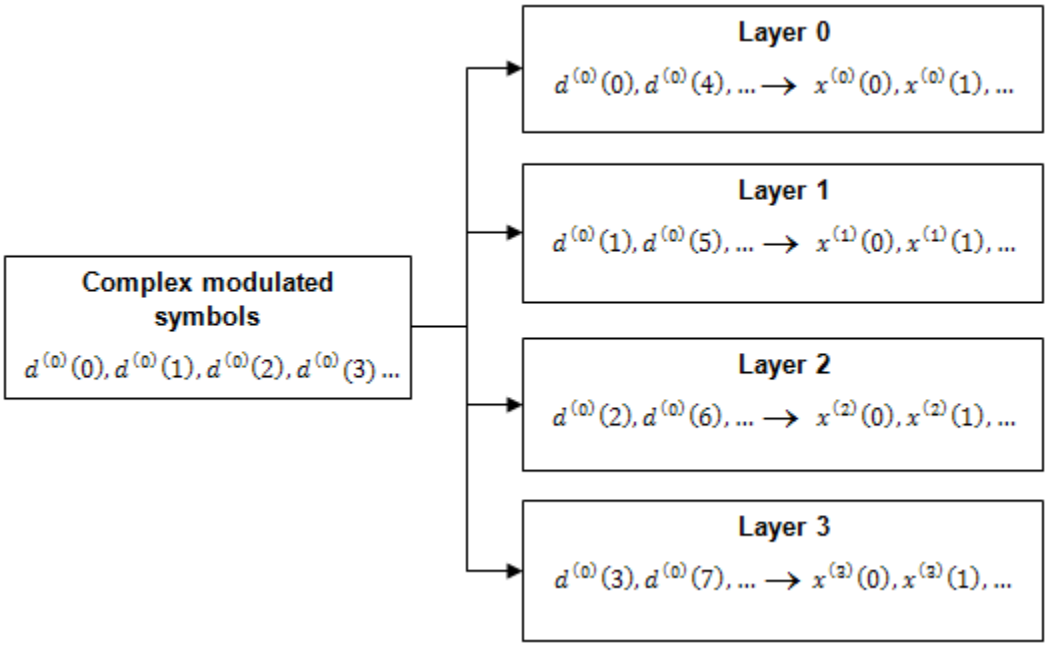
Layer Mapping for Transmit Diversity

If transmitter diversity is used, the input symbols are mapped to layers based on the number of layers.

- **Two Layers** — Even symbols are mapped to layer 0 and odd symbols are mapped to layer 1, as shown in this figure.



- **Four Layers** — The input symbols are mapped to layers sequentially, as shown in this figure.



If the total number of input symbols is not an integer multiple of four, two null symbols are appended to the end. Since the original number of symbols is always an integer multiple of two, the new total is an integer multiple of four.

Layer Mapping for Spatial Multiplexing

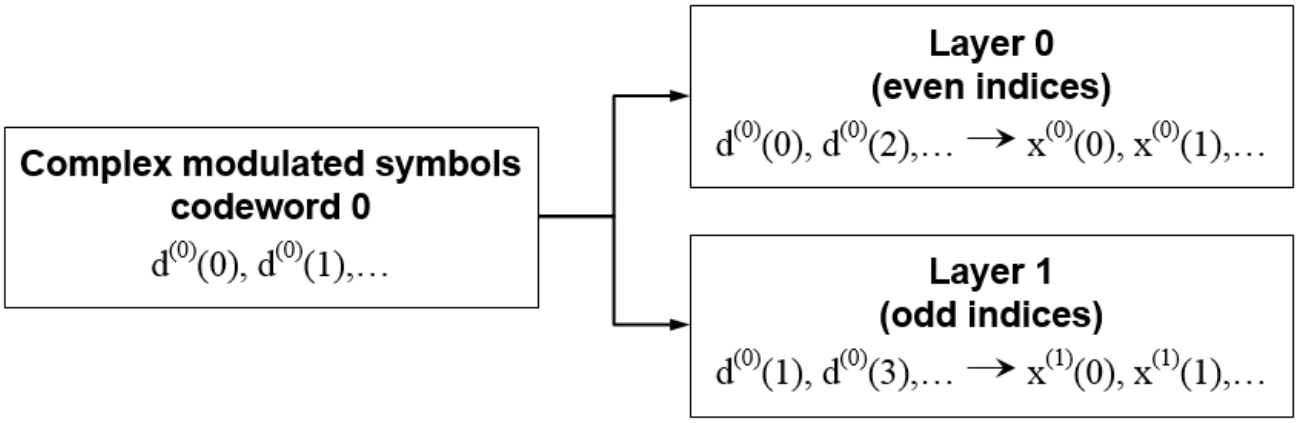
In the case of spatial multiplexing, the number of layers used is always less than or equal to the number of antenna ports used for transmission of the physical channel.

One Layer

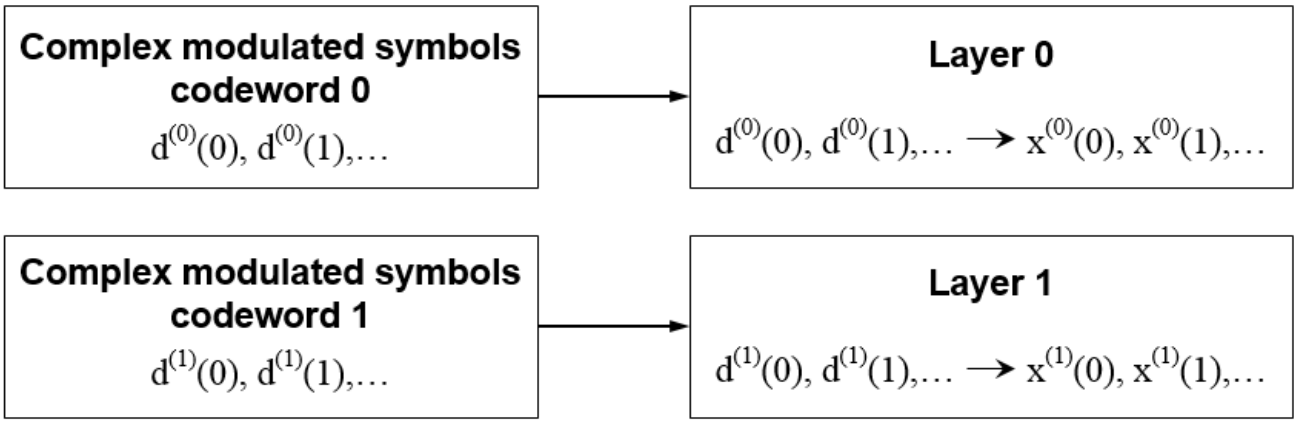
If one layer is used, $x^{(0)}(i) = d^{(0)}(i)$.

Two Layers

- **One Codeword** — Even symbols are mapped to layer 0 and odd symbols are mapped to layer 1, as shown in the following figure.

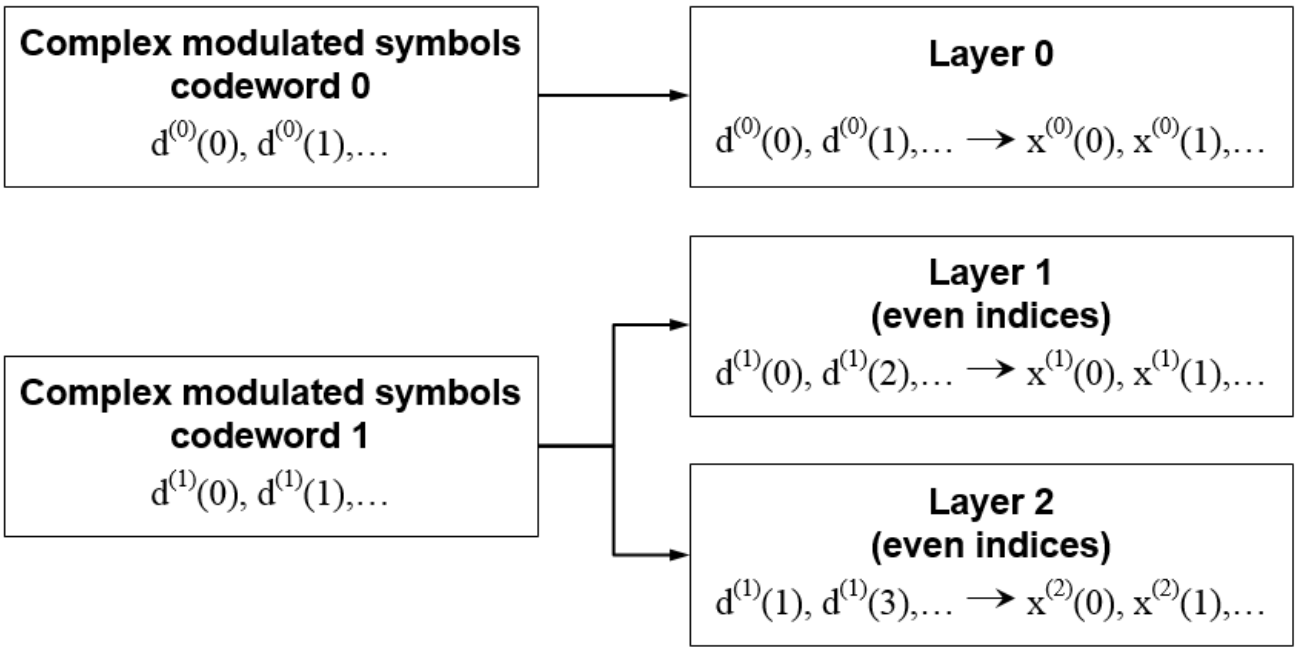


- **Two Codewords** — Codeword 0 is mapped to layer 0 and codeword 1 is mapped to layer 1, as shown in the following figure.



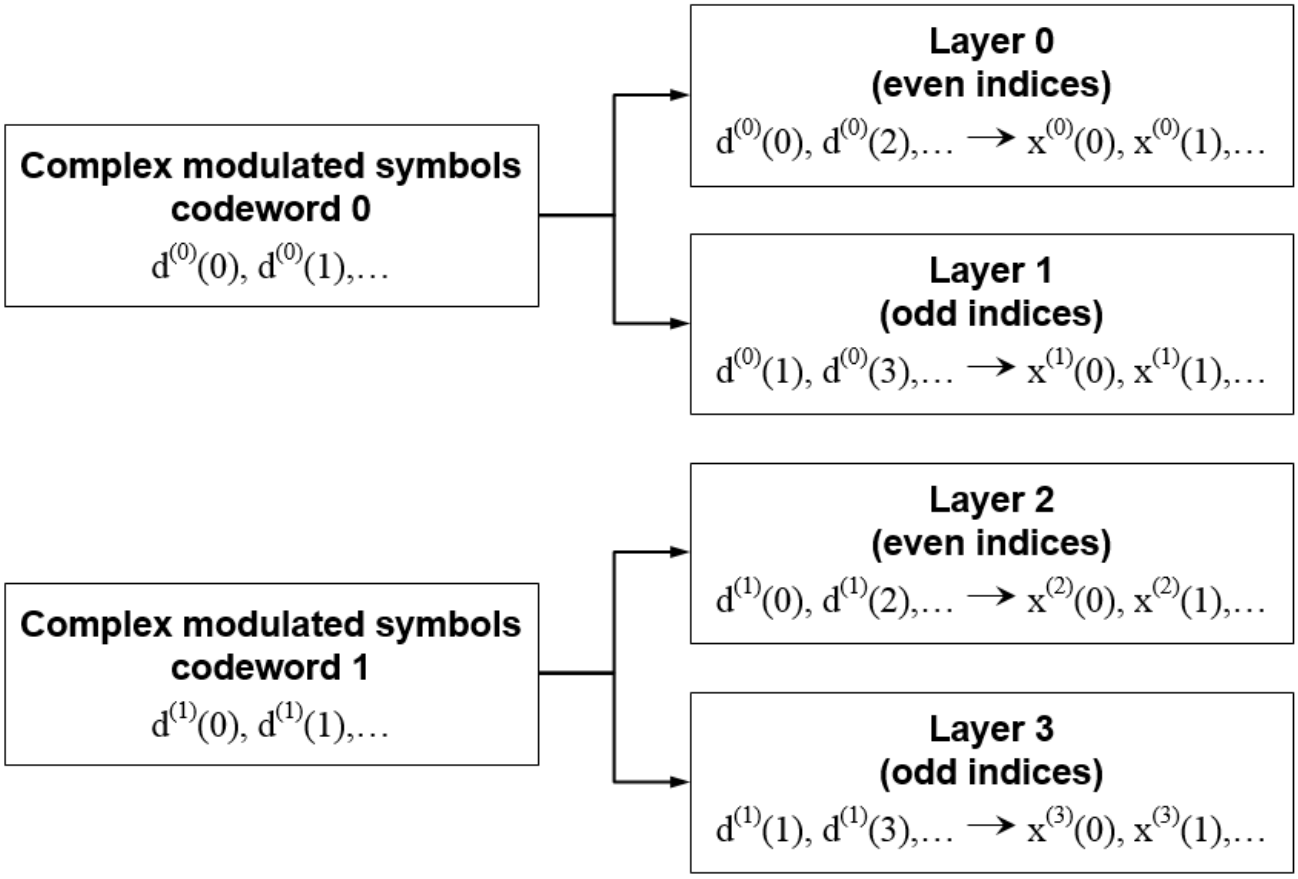
Three Layers

In the case of three layers, codeword 0 is mapped to layer 0 and even symbols within codeword 1 are mapped to layer 1 and odd symbols within codeword 1 are mapped to layer 2.



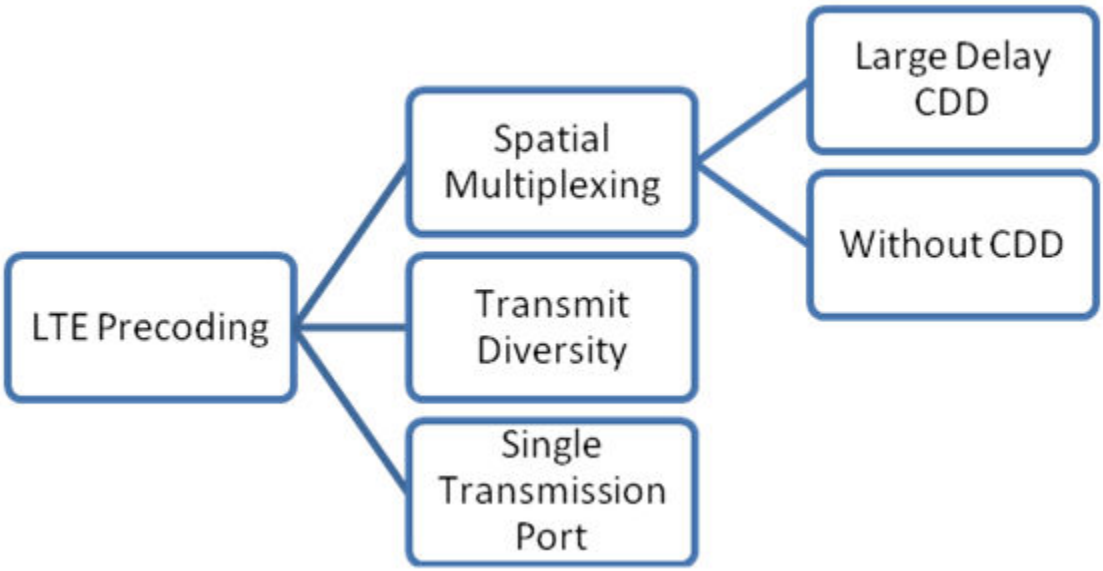
Four Layers

In the case of four layers, two codewords must be used. In this case, the even symbols of codeword 0 are mapped to layer 0, and the odd symbols are mapped to layer 1. The even symbols of codeword 1 are mapped to layer 2, and the odd symbols are mapped to layer 3.



Precoding

Three types of precoding are available in LTE for the PDSCH: spatial multiplexing, transmit diversity, and single antenna port transmission. Within spatial multiplexing, there are two schemes: precoding with large delay cyclic delay diversity (CDD), also known as open loop spatial multiplexing, and precoding without CDD, also known as closed loop spatial multiplexing. The various types of precoding are illustrated in this tree diagram.



The precoder takes a block from the layer mapper, $x^{(0)}(i), x^{(1)}(i), \ldots, x^{(v-1)}(i)$, and generates a sequence for each antenna port, $y^{(p)}(i)$. The variable p is the transmit antenna port number, and can assume values of {0}, {0,1}, or {0,1,2,3}.

Single Antenna Port Precoding

For transmission over a single antenna port, no processing is carried out, as shown in this equation.

$$y^{(p)}(i) = x^{(0)}(i)$$

Precoding for Large Delay CDD Spatial Multiplexing

CDD operation applies a cyclic shift, which is a delay of N_{FFT}/v samples to each antenna, where N_{FFT} is the size of the OFDM FFT. The use of CDD improves the robustness of performance by randomizing the channel frequency response, reducing the probability of deep fading.

Precoding with CDD for spatial multiplexing is defined by the following equation.

$$\begin{pmatrix} y^{(0)}(i) \\ \vdots \\ y^{(p-1)}(i) \end{pmatrix} = W(i) \times D(i) \times U \times \begin{pmatrix} x^{(0)}(i) \\ \vdots \\ x^{(v-1)}(i) \end{pmatrix}$$

Values of the precoding matrix, $W(i)$, of size $P \times v$, are selected from a codebook configured by the eNodeB and user equipment. The precoding codebook is described in “Spatial Multiplexing Precoding Codebook” on page 1-84. Every group of symbols at index i across all available layers can use a different precoding matrix, if required. The supporting matrices, $D(i)$and U, are given for various numbers of layers in the following table.

Number of layers, υ	U	$D(i)$
2	$\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & e^{-j2\pi/2}\end{pmatrix}$	$\begin{pmatrix}1 & 0\\ 0 & e^{-j2\pi i/2}\end{pmatrix}$
3	$\frac{1}{\sqrt{3}}\begin{pmatrix}1 & 1 & 1\\ 1 & e^{-j2\pi/3} & e^{-j4\pi/3}\\ 1 & e^{-j4\pi/3} & e^{-j8\pi/3}\end{pmatrix}$	$\begin{pmatrix}1 & 0 & 0\\ 0 & e^{-j2\pi i/3} & 0\\ 0 & 0 & e^{-j4\pi i/3}\end{pmatrix}$
4	$\frac{1}{2}\begin{pmatrix}1 & 1 & 1 & 1\\ 1 & e^{-j2\pi/4} & e^{-j4\pi/4} & e^{-j6\pi/4}\\ 1 & e^{-j4\pi/4} & e^{-j8\pi/4} & e^{-j12\pi/4}\\ 1 & e^{-j6\pi/4} & e^{-j12\pi/4} & e^{-j18\pi/4}\end{pmatrix}$	$\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & e^{-j2\pi i/4} & 0 & 0\\ 0 & 0 & e^{-j4\pi i/4} & 0\\ 0 & 0 & 0 & e^{-j6\pi i/4}\end{pmatrix}$

Precoding for Spatial Multiplexing without CDD

Precoding for spatial multiplexing without CDD is defined by the following equation.

$$\begin{pmatrix} y^{(0)}(i) \\ \vdots \\ y^{(p-1)}(i) \end{pmatrix} = W(i) \times \begin{pmatrix} x^{(0)}(i) \\ \vdots \\ x^{(v-1)}(i) \end{pmatrix}$$

Values of the precoding matrix, $W(i)$, of size $P \times v$, are selected from a codebook configured by the eNodeB and user equipment. Every group of symbols at index i across all available layers can use a different precoding matrix if required. For more information on the precoding codebook, see “Spatial Multiplexing Precoding Codebook” on page 1-84.

Spatial Multiplexing Precoding Codebook

The precoding matrices for antenna ports {0,1} are given in the following table.

Codebook index	Number of layers, υ	
	1	**2**
0	$\frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}$	$\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}$
1	$\frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix}$	$\frac{1}{2}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$
2	$\frac{1}{\sqrt{2}}\begin{pmatrix}1\\j\end{pmatrix}$	$\frac{1}{2}\begin{pmatrix}1 & 1\\ j & -j\end{pmatrix}$
3	$\frac{1}{\sqrt{2}}\begin{pmatrix}1\\-j\end{pmatrix}$	—

The precoding matrices for antenna ports {0,1,2,3} are given in the following table.

Codebook index	u_n	Number of layers, v			
		1	2	3	4
0	$u_0 = [1\ -1\ -1\ -1]^T$	$W_0^{\{1\}}$	$\frac{W_0^{\{14\}}}{\sqrt{2}}$	$\frac{W_0^{\{124\}}}{\sqrt{3}}$	$\frac{W_0^{\{1234\}}}{2}$
1	$u_1 = [1\ -j\ 1\ j]^T$	$W_1^{\{1\}}$	$\frac{W_1^{\{12\}}}{\sqrt{2}}$	$\frac{W_1^{\{123\}}}{\sqrt{3}}$	$\frac{W_1^{\{1234\}}}{2}$
2	$u_2 = [1\ 1\ -1\ 1]^T$	$W_2^{\{1\}}$	$\frac{W_2^{\{12\}}}{\sqrt{2}}$	$\frac{W_2^{\{123\}}}{\sqrt{3}}$	$\frac{W_2^{\{3214\}}}{2}$
3	$u_3 = [1\ j\ 1\ -j]^T$	$W_3^{\{1\}}$	$\frac{W_3^{\{12\}}}{\sqrt{2}}$	$\frac{W_3^{\{123\}}}{\sqrt{3}}$	$\frac{W_3^{\{3214\}}}{2}$
4	$u_4 = \left[1\ \frac{-1-j}{\sqrt{2}}\ -j\ \frac{1-j}{\sqrt{2}}\right]^T$	$W_4^{\{1\}}$	$\frac{W_4^{\{14\}}}{\sqrt{2}}$	$\frac{W_4^{\{124\}}}{\sqrt{3}}$	$\frac{W_4^{\{1234\}}}{2}$
5	$u_5 = \left[1\ \frac{1-j}{\sqrt{2}}\ j\ \frac{-1-j}{\sqrt{2}}\right]^T$	$W_5^{\{1\}}$	$\frac{W_5^{\{14\}}}{\sqrt{2}}$	$\frac{W_5^{\{124\}}}{\sqrt{3}}$	$\frac{W_5^{\{1234\}}}{2}$
6	$u_6 = \left[1\ \frac{1+j}{\sqrt{2}}\ -j\ \frac{-1+j}{\sqrt{2}}\right]^T$	$W_6^{\{1\}}$	$\frac{W_6^{\{13\}}}{\sqrt{2}}$	$\frac{W_6^{\{134\}}}{\sqrt{3}}$	$\frac{W_6^{\{1324\}}}{2}$
7	$u_7 = \left[1\ \frac{-1+j}{\sqrt{2}}\ j\ \frac{1+j}{\sqrt{2}}\right]^T$	$W_7^{\{1\}}$	$\frac{W_7^{\{13\}}}{\sqrt{2}}$	$\frac{W_7^{\{134\}}}{\sqrt{3}}$	$\frac{W_7^{\{1324\}}}{2}$
8	$u_8 = [1\ -1\ 1\ 1]^T$	$W_8^{\{1\}}$	$\frac{W_8^{\{12\}}}{\sqrt{2}}$	$\frac{W_8^{\{124\}}}{\sqrt{3}}$	$\frac{W_8^{\{1234\}}}{2}$
9	$u_9 = [1\ -j\ -1\ -j]^T$	$W_9^{\{1\}}$	$\frac{W_9^{\{14\}}}{\sqrt{2}}$	$\frac{W_9^{\{134\}}}{\sqrt{3}}$	$\frac{W_9^{\{1234\}}}{2}$
10	$u_{10} = [1\ 1\ 1\ -1]^T$	$W_{10}^{\{1\}}$	$\frac{W_{10}^{\{13\}}}{\sqrt{2}}$	$\frac{W_{10}^{\{123\}}}{\sqrt{3}}$	$\frac{W_{10}^{\{1324\}}}{2}$
11	$u_{11} = [1\ j\ -1\ j]^T$	$W_{11}^{\{1\}}$	$\frac{W_{11}^{\{13\}}}{\sqrt{2}}$	$\frac{W_{11}^{\{134\}}}{\sqrt{3}}$	$\frac{W_{11}^{\{1324\}}}{2}$
12	$u_{12} = [1\ -1\ -1\ 1]^T$	$W_{12}^{\{1\}}$	$\frac{W_{12}^{\{12\}}}{\sqrt{2}}$	$\frac{W_{12}^{\{123\}}}{\sqrt{3}}$	$\frac{W_{12}^{\{1324\}}}{2}$

Codebook index	u_n	Number of layers, v			
		1	2	3	4
13	$u_{13} = [1\ -1\ 1\ -1]^T$	$W_{13}^{\{1\}}$	$\frac{W_{13}^{\{13\}}}{\sqrt{2}}$	$\frac{W_{13}^{\{123\}}}{\sqrt{3}}$	$\frac{W_{13}^{\{1324\}}}{2}$
14	$u_{14} = [1\ 1\ -1\ -1]^T$	$W_{14}^{\{1\}}$	$\frac{W_{14}^{\{13\}}}{\sqrt{2}}$	$\frac{W_{14}^{\{123\}}}{\sqrt{3}}$	$\frac{W_{14}^{\{3214\}}}{2}$
15	$u_{15} = [1\ 1\ 1\ 1]^T$	$W_{15}^{\{1\}}$	$\frac{W_{15}^{\{12\}}}{\sqrt{2}}$	$\frac{W_{15}^{\{123\}}}{\sqrt{3}}$	$\frac{W_{15}^{\{1234\}}}{2}$

The precoding matrix, $W_n^{\{s\}}$, is the matrix defined by the columns in set $\{s\}$ by the following equation.

$$W_n = I - 2u_n u_n^H / u_n^H u_n$$

In the preceding equation, I is a 4-by-4 identity matrix. The vector u_n is given in the preceding table.

Precoding for Transmit Diversity

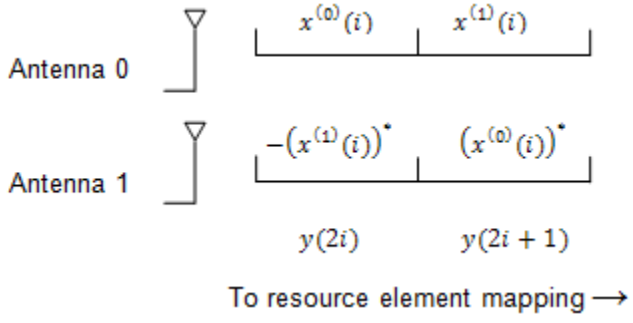
Precoding for transmit diversity is available on two or four antenna ports.

Two Antenna Ports

An Alamouti scheme is used for precoding, which defines the relationship between input and output as shown in the following equation.

$$\begin{pmatrix} y^{(0)}(2i) \\ y^{(1)}(2i) \\ y^{(0)}(2i+1) \\ y^{(1)}(2i+1) \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & j & 0 \\ 0 & -1 & 0 & j \\ 0 & 1 & 0 & j \\ 1 & 0 & -j & 0 \end{pmatrix} \begin{pmatrix} \mathrm{Re}\{x^{(0)}(i)\} \\ \mathrm{Re}\{x^{(1)}(i)\} \\ \mathrm{Im}\{x^{(0)}(i)\} \\ \mathrm{Im}\{x^{(1)}(i)\} \end{pmatrix}$$

In the Alamouti scheme, two consecutive symbols, $x^{(0)}(i)$ and $x^{(1)}(i)$, are transmitted in parallel using two antennas with the following mapping, where the asterisk symbol (*) denotes the complex conjugate operation.



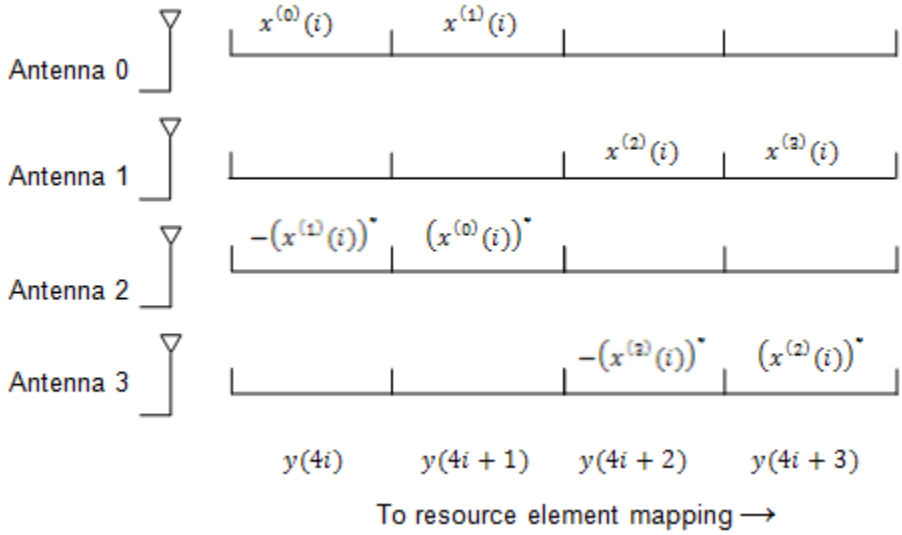
Since any two columns in the precoding matrix are orthogonal, the two symbols, $x^{(0)}(i)$ and $x^{(1)}(i)$, can be separated at the UE.

Four Antenna Ports

Precoding for the four antenna port case defines the relationship between the input and output as shown in the following equation.

$$\begin{pmatrix} y^{(0)}(4i) \\ y^{(1)}(4i) \\ y^{(2)}(4i) \\ y^{(3)}(4i) \\ y^{(0)}(4i+1) \\ y^{(1)}(4i+1) \\ y^{(2)}(4i+1) \\ y^{(3)}(4i+1) \\ y^{(0)}(4i+2) \\ y^{(1)}(4i+2) \\ y^{(2)}(4i+2) \\ y^{(3)}(4i+2) \\ y^{(0)}(4i+3) \\ y^{(1)}(4i+3) \\ y^{(2)}(4i+3) \\ y^{(3)}(4i+3) \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 0 & j & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & j & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & j & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -j & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & j & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & j \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & j \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -j & 0 \end{pmatrix} \begin{pmatrix} \mathrm{Re}\{x^{(0)}(i)\} \\ \mathrm{Re}\{x^{(1)}(i)\} \\ \mathrm{Re}\{x^{(2)}(i)\} \\ \mathrm{Re}\{x^{(3)}(i)\} \\ \mathrm{Im}\{x^{(0)}(i)\} \\ \mathrm{Im}\{x^{(1)}(i)\} \\ \mathrm{Im}\{x^{(2)}(i)\} \\ \mathrm{Im}\{x^{(3)}(i)\} \end{pmatrix}$$

In this scheme, two consecutive symbols are transmitted in parallel in two symbol periods using four antennas with this mapping, where the asterisk symbol (*) denotes the complex conjugate operation.



Valid Codeword, Layer and Precoding Scheme Combinations

The valid numbers of codewords and layers for each precoding scheme, as described in previous sections, are summarized in these tables.

Single Antenna Port

Codewords	Layers
1	1

Transmit Diversity

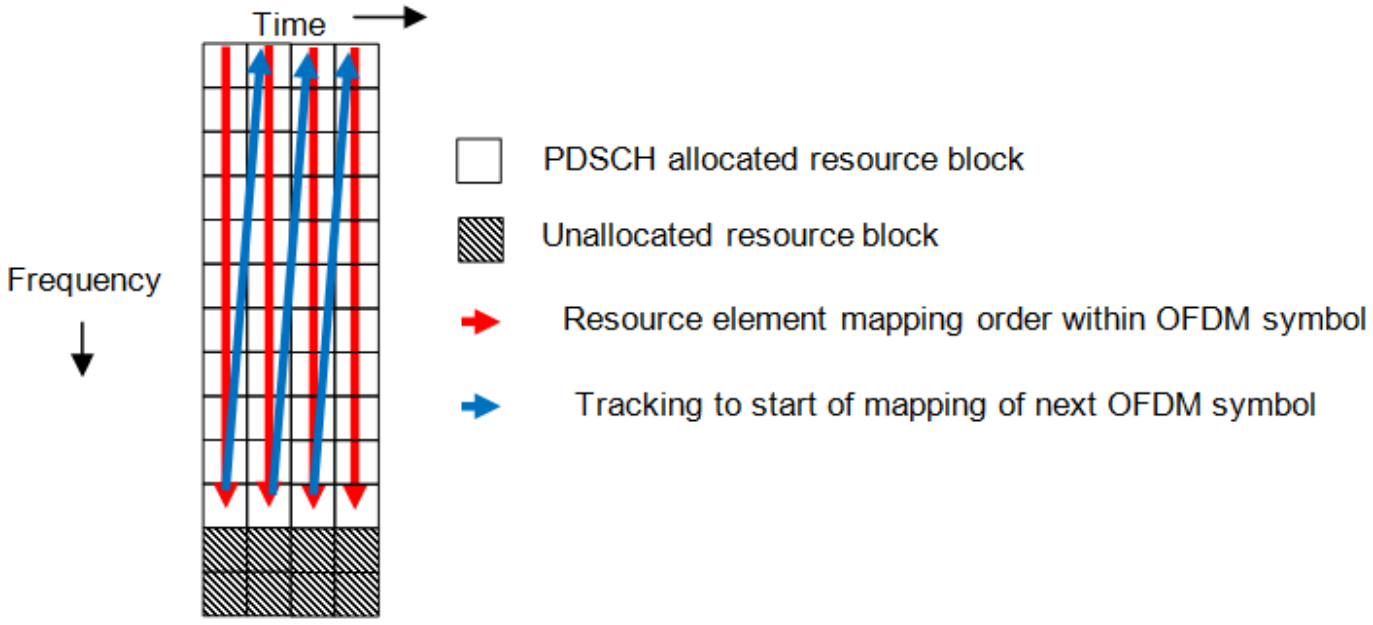
Codewords	Layers
1	2
1	4

Spatial Multiplexing

Codewords	Layers
1	1
1	2
2	2
2	3
2	4

Mapping to Resource Elements

For each of the antenna ports used for transmission of the PDSCH, the block of complex valued symbols, $y^{(p)}(i)$, are mapped in sequence to resource elements not occupied by the PCFICH, PHICH, PDCCH, PBCH, or synchronization and reference signals. The number of resource elements mapped to is controlled by the number of resource blocks allocated to the PDSCH. The symbols are mapped by increasing the subcarrier index and mapping all available REs within allocated resource blocks for each OFDM symbol as shown in this figure.



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

`lteDLSCH` | `ltePDSCH` | `ltePDSCHIndices` | `lteDLResourceGrid` | `lteDLSCHInfo` | `lteDLSCHDecode` | `lteLayerMap` | `lteLayerDemap` | `lteDLPrecode` | `lteDLDeprecode` | `lteTurboEncode` | `lteTurboDecode` | `ltePDSCHPRBS` | `lteCRCEncode` | `lteCRCDecode` | `lteCodeBlockSegment` | `lteCodeBlockDesegment` | `lteRateMatchTurbo` | `lteRateRecoverTurbo`

Related Examples

- “Model DL-SCH and PDSCH” on page 3-16

More About

- “Interleaving”

Uplink Shared Channel

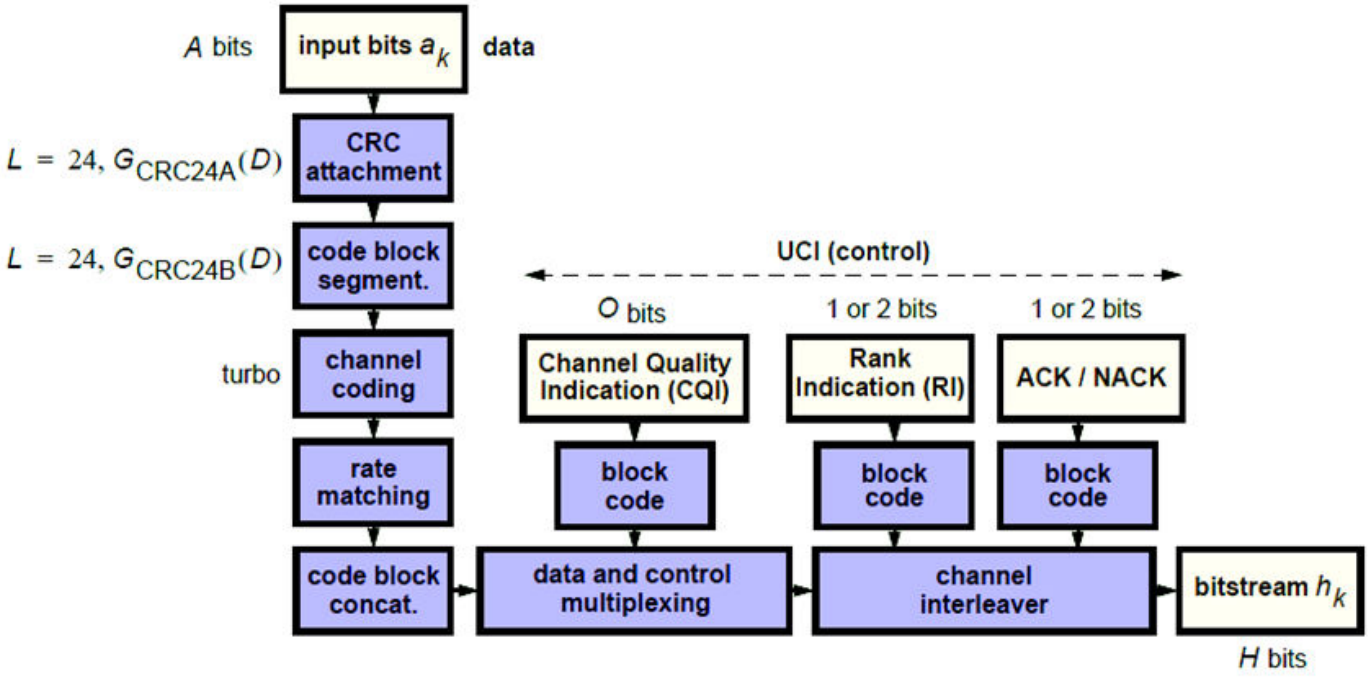
In this section...

The physical uplink shared channel is used to transmit the uplink shared channel (UL-SCH) and L1 and L2 control information. The UL-SCH is the transport channel used for transmitting uplink data (a transport block). L1 and L2 control signalling can carry HARQ acknowledgements for received DL-SCH blocks, channel quality reports, and scheduling requests.

UL-SCH Coding

To create the PUSCH payload a transport block of length A, denoted as $a_0, a_1, \ldots, a_{A-1}$, undergoes transport block coding. 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. This processing is described in TS 36.212 [1], Sections 5.2.2.1 to 5.2.2.5 and 5.2.2.8.

The transport block coding and control information coding and multiplexing steps are illustrated in the following block diagram.



It is possible for the PUSCH to carry only control information and no data. In this case, only the control information coding and multiplexing chain are followed as per the preceding diagram.

Transport Block CRC Attachment

A cyclic redundancy check (CRC) is used for error detection in transport blocks. The entire transport block is used to calculate the CRC parity bits. The transport block is divided by a cyclic generator polynomial, described as g_{CRC24A} in section 5.1.1 of [1], to generate 24 parity bits. These parity bits are then appended to the end of the transport block.

Code Block Segmentation and CRC Attachment

The input block of bits to the code segmentation block is denoted by $b_0, b_1, \ldots, b_{B-1}$, where $B = A + 24$. In LTE, a minimum and maximum code block size is specified, so the block sizes are compatible with the block sizes supported by the turbo interleaver.

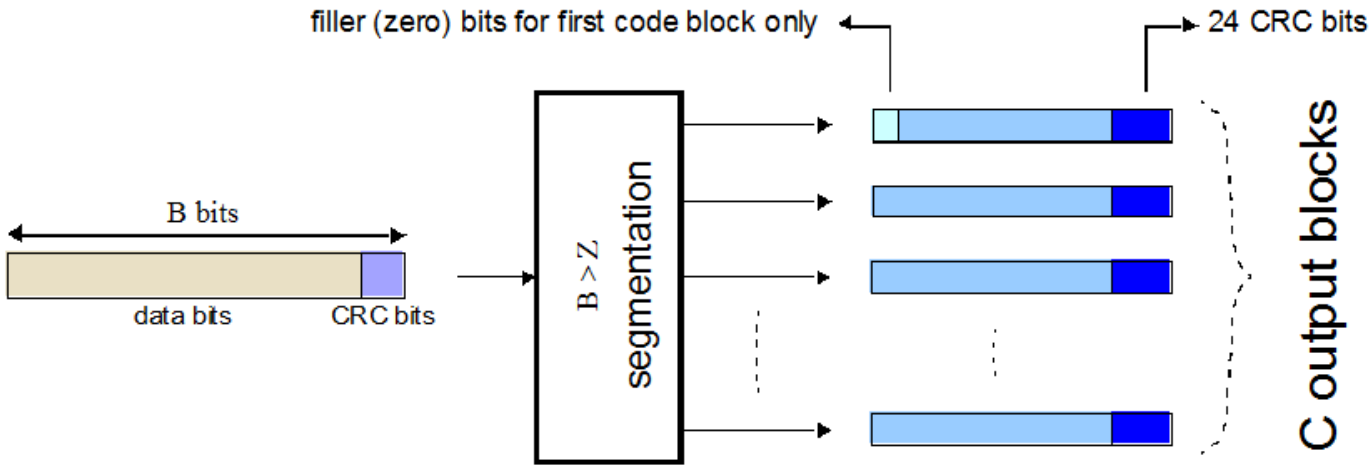
- The minimum code block size is 40 bits
- The maximum code block size, denoted by Z, is 6144 bits

If the length of the input block, B, is greater than the maximum code block size, the input block is segmented.

When the input block is segmented, it is divided into $C = [B/(Z - L)]$ smaller blocks, where L is 24. Therefore, $C = [B/6120]$ code blocks.

Each code block has a 24-bit CRC attached to the end, calculated as described in “Transport Block CRC Attachment” on page 1-72, but the generator polynomial, described as g_{CRC24B} in section 5.1.1 of [1] is used.

If required, the algorithm appends filler bits to the start of the segment so that the code block sizes match a set of valid turbo interleaver block sizes. The process is shown in this figure.

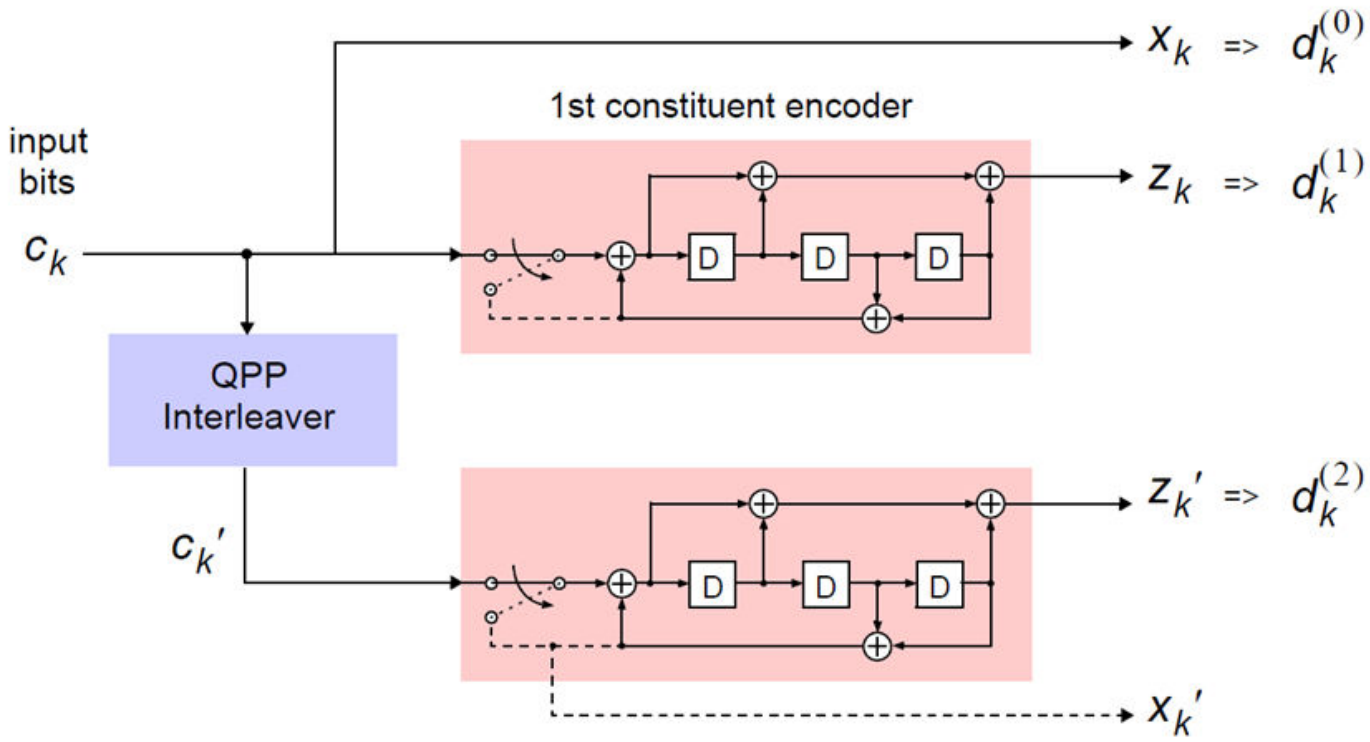


If no segmentation is needed, only one code block is produced. If *B* is less than the minimum size, filler bits (zeros) are added to the beginning of the code block to achieve a total of 40 bits.

Channel Coding

- “Constituent Encoders” on page 1-93
- “Trellis Termination for Turbo Encoder” on page 1-94
- “The QPP Interleaver” on page 1-94

The code blocks undergo turbo coding. Turbo coding is a form of forward error correction that improves the channel capacity by adding redundant information. The turbo encoder scheme used is a Parallel Concatenated Convolutional Code (PCCC) with two recursive convolutional coders and a “contention-free” Quadratic Permutation Polynomial (QPP) interleaver, as shown in this figure.



The output of the encoder is three streams, $d_k^{(0)}$, $d_k^{(1)}$, and $d_k^{(2)}$, to achieve a code rate of 1/3.

Constituent Encoders

The input to the first constituent encoder is the input bit stream to the turbo coding block. The input to the second constituent encoder is the output of the QPP interleaver, a permutation of the input sequence.

Each encoder outputs two sequences, called the systematic sequence and the parity sequence. The systematic sequences are denoted by (x_k, x'_k) and the parity sequences are denoted by (z_k, z'_k). Only one of the systematic sequences (x_k) is used as an output because the other (x'_k) is simply a permutation of the chosen systematic sequence. The transfer function for each constituent encoder is given by the following equation.

$$G(D) = \left[1, \frac{g_1(D)}{g_0(D)}\right]$$

The first element, 1, represents the systematic output transfer function. The second element, $\left(\frac{g_1(D)}{g_0(D)}\right)$, represents the recursive convolutional output transfer function.

$$g_0(D) = 1 + D^2 + D^3$$

$$g_1(D) = 1 + D + D^3$$

The output for each sequence can be calculated using the transfer function.

The encoder is initialized with all zeros. If the code block to be encoded is the 0-th and filler bits (F) are used, the input to the encoder (c_k) is set to zero and the output (x_k) and (z_k) set to <NULL> for $k = 0, ..., F - 1$.

Trellis Termination for Turbo Encoder

A standard convolutional coder initializes its internal registers to an all zeros state and ensures the coder finishes in an all zeros state by padding the end of the input sequence with k zeros. Since the decoder knows the start and end state of the encoder, it can decode the data. Driving a recursive coder to an all zeros state using this method is not possible. To overcome this problem, trellis termination is used.

Upon termination, the tail bits are fed back to the input of each encoder using a switch. The first three tail bits are used to terminate each encoder.

The QPP Interleaver

The role of the interleaver is to spread the information bits such that in the event of a burst error, the two code streams are affected differently, allowing data to still be recovered.

The output of the interleaver is a permutation of the input data, as shown in these equations.

$$c'_i = c_{\Pi(i)}, i = 0, 1, ..., (K - 1)$$

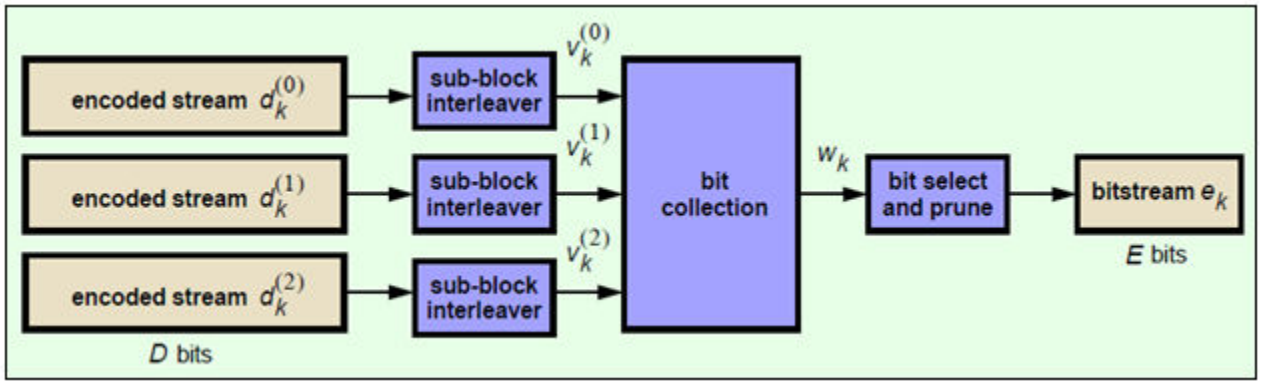
$$\Pi(i) = \left(f_1 i + f_2 i^2\right) \mathrm{mod} K$$

The variable K is the input length. The variables f_1 and f_2 are coefficients chosen depending on K, in table 5.1.3-3 of [1]. For example, K=40, f_1=3, and f_2=10, yields the following sequence.

$$\Pi(i) = 0, 13, 6, 19, 12, 25, 18, 31, 24, 37, 30, 3, 36, 9, 2, 15, 8, 21, 14, 27, 20, 33, 26, 39, 32, 5, 38, ...$$

Rate Matching

The rate matching block creates an output bitstream with a desired code rate. Since the number of bits available for transmission depends on the available resources, the rate matching algorithm is capable of producing any rate. The three bitstreams from the turbo encoder are interleaved, followed by bit collection, to create a circular buffer. Bits are selected and pruned from the buffer to create an output bitstream with the desired code rate. The process is illustrated in this figure.



Sub-block Interleaver

The three sub-block interleavers used in the rate matching block are identical. An interleaver permutes the bits it receives. This makes burst errors easier to correct by preventing consecutive corrupt bits.

The sub-block interleaver reshapes the encoded bit sequence, row-by-row, to form a matrix with $C_{Subblock}^{TC} = 32$ columns and $R_{Subblock}^{TC}$ rows. The variable $R_{Subblock}^{TC}$ is determined by finding the minimum integer such that the number of encoded input bits is $D \le \left(R_{Subblock}^{TC} \times C_{Subblock}^{TC}\right)$. If $\left(R_{Subblock}^{TC} \times C_{Subblock}^{TC}\right) > D$, N_D <NULL>'s are appended to the front of the encoded sequence. In this case, $N_D + D = \left(R_{Subblock}^{TC} \times C_{Subblock}^{TC}\right)$.

For blocks $d_k^{(0)}$ and $d_k^{(1)}$, inter-column permutation is performed on the matrix to reorder the columns as shown in this pattern.

0, 16, 8, 24, 4, 20, 12, 28, 2, 18, 10, 26, 6, 22, 14, 30, 1, 17, 9, 25, 5, 21, 13, 29, 3, 19, 11, 27, 7, 23, 15, 31

The output of the block interleaver for blocks $d_k^{(0)}$ and $d_k^{(1)}$ is the bit sequence read out column-by-column from the inter-column permuted matrix to create a stream $K_\Pi = \left(R_{Subblock}^{TC} \times C_{Subblock}^{TC}\right)$ bits long.

For block $d_k^{(2)}$, the elements of the matrix are permuted separately based on the permutation pattern shown above, but modified to create a permutation which is a function of the variables $R_{Subblock}^{TC}$, $C_{Subblock}^{TC}$, k, and K_Π. This process creates three interleaved bitstreams.

$$d_k^{(0)} \rightarrow v_k^{(0)}$$

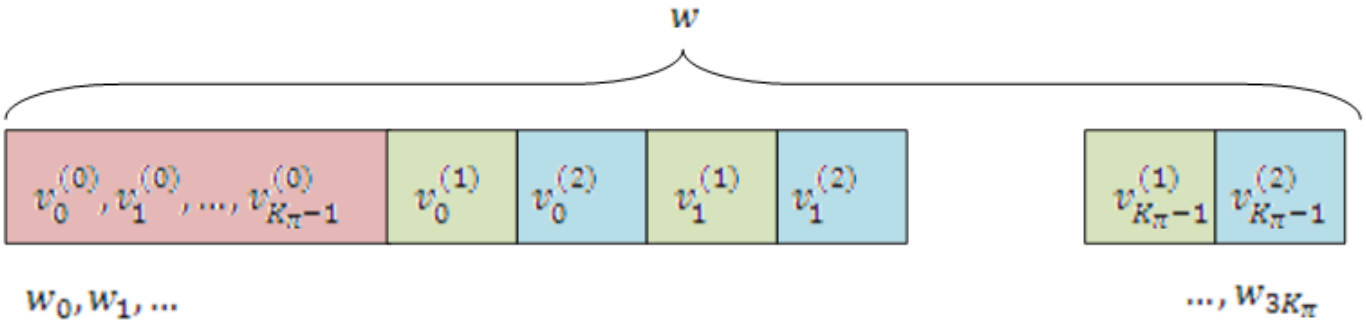
$$d_k^{(1)} \rightarrow v_k^{(1)}$$

$$d_k^{(2)} \rightarrow v_k^{(2)}$$

Bit Collection, Selection, and Transmission

The bit collection stage creates a virtual circular buffer by combining the three interleaved encoded bit streams.

The sequences $v_k^{(1)}$ and $v_k^{(2)}$ are combined by interlacing successive values from each sequence. This combination is then appended to the end of $v_k^{(0)}$ to create the circular buffer w_k shown in this figure.



Interlacing allows equal levels of protection for each parity sequence.

The algorithm then selects and prunes bits from the circular buffer to create an output sequence length that meets the desired code rate.

The Hybrid Automatic Repeat Request (HARQ) error correction scheme is incorporated into the rate-matching algorithm of LTE. For any desired code rate the coded bits are output serially from the circular buffer from a starting location, given by the redundancy version (RV), wrapping around to the beginning of the buffer if the end of the buffer is reached. `NULL` bits are discarded. Different RVs, and hence starting points, allow for the retransmission of selected data. The ability to select different starting points enables the following two main methods of recombining data at the receiver in the HARQ process.

- Chase combining — retransmissions contain the same data and parity bit.
- Incremental redundancy — retransmissions contain different information, so the receiver gains knowledge on each retransmission.

Code Block Concatenation

At this stage, the rate-matched code blocks are combined again. This is done by sequentially concatenating the blocks to create the output of the channel coding, f_k for $k = 0, \ldots, G-1$.

Channel Coding of Control Information with UL-SCH Data

- “HARQ-ACK Information” on page 1-97
- “Rank Indicator” on page 1-97
- “Channel Quality Information and Precoder Matrix Indicator” on page 1-98

Control information arrives at the coder in the form of channel quality information (CQI), precoder matrix indication (PMI), rank indication (RI), and HARQ-Indicator (HI). Different control information coding rates are achieved by allocating a different number of coded symbols for transmission. When control information is transmitted on the PUSCH, the channel coding for HI, RI, and CQI is done independently.

The transmission mode determines the bit widths assigned to the various types control information; the corresponding widths for the transmission modes can be found in TS 36.212 [1], Section 5.2.2.6.1-4.

The following sections describe uplink control information on PUSCH with UL-SCH data.

HARQ-ACK Information

The number of coded symbols, *Q*, used by the UE to transmit the HARQ acknowledgement bits is determined using the number of HARQ bits (1 or 2 depending on the number of codewords present), the scheduled PUSCH bandwidth expressed as a number of subcarriers, the number of SC-FDMA symbols per subframe for the initial PUSCH transmission, and information obtained from the initial PDCCH for the same transport block.

Each positive acknowledgement (ACK) is encoded as a binary `1` and negative acknowledgement (NACK) is encoded as a binary `0`. If the HARQ-ACK consists of 1-bit of information, $[o_0^{ACK}]$, corresponding to 1 codeword, then it is first encoded according to the following table.

Q_m	Encoded HARQ-ACK
2	$[o_0^{ACK}$ y]
4	$[o_0^{ACK}$ y x x]
6	$[o_0^{ACK}$ y x x x x]

In the preceding table, *x* and *y* are placeholders used to scramble the HARQ-ACK bits in such a way as to maximize the Euclidean distance of the modulation symbols carrying the HARQ information.

If the HARQ-ACK consists of 2 bits of information, $[o_0^{ACK}\ o_1^{ACK}]$, where o_0^{ACK} and o_1^{ACK} correspond to the first and second codeword, respectively, and $o_2^{ACK} = (o_0^{ACK} + o_1^{ACK})\text{mod}2$, then they are encoded according to the following table.

Q_m	Encoded HARQ-ACK
2	$[o_0^{ACK}\ o_1^{ACK}\ o_2^{ACK}\ o_0^{ACK}\ o_1^{ACK}\ o_2^{ACK}]$
4	$[o_0^{ACK}\ o_1^{ACK}$ x x $\ o_2^{ACK}\ o_0^{ACK}$ x x $o_1^{ACK}\ o_2^{ACK}$ x x]
6	$[o_0^{ACK}\ o_1^{ACK}$ x x x x $\ o_2^{ACK}\ o_0^{ACK}$ x x x x $\ o_1^{ACK}\ o_2^{ACK}$ x x x x]

Rank Indicator

The bit widths, *Q*, (1 or 2 information bits) for rank indication feedback for PDSCH transmissions are determined using the maximum number of layers according to the corresponding eNodeB antenna configuration and UE category. If RI consists of 1 information bit, $[o_0^{RI}]$, then it is first encoded according to the following table.

Q_m	Encoded RI
2	$[o_0^{RI}$ y]
4	$[o_0^{RI}$ y x x]
6	$[o_0^{RI}$ y x x x x]

In the preceding table, *x* and *y* are placeholders used to scramble the HARQ-ACK bits in such a way as to maximize the Euclidean distance of the modulation symbols carrying the HARQ information.

If the RI consists of 2 information bits, $[o_0^{RI}\ o_1^{RI}]$, then they are first encoded according to the following table.

Q_m	Encoded RI
2	$[o_0^{RI}\ o_1^{RI}\ o_2^{RI}\ o_0^{RI}\ o_1^{RI}\ o_2^{RI}]$
4	$[o_0^{RI}\ o_1^{RI}$ x x $o_2^{RI}\ o_0^{RI}$ x x $o_1^{RI}\ o_2^{RI}$ x x]
6	$[o_0^{RI}\ o_1^{RI}$ x x x x $o_2^{RI}\ o_0^{RI}$ x x x x $o_1^{RI}\ o_2^{RI}$ x x x x]

Channel Quality Information and Precoder Matrix Indicator

The number of coded symbols, Q, used for channel quality information is determined from the number of CQI bits present, the number of CRC bits, the scheduled PUSCH bandwidth expressed as a number of subcarriers, and information obtained from the PDCCH for the same transport block.

If the payload size is greater than 11 bits, the CQI bit sequence undergoes CRC attachment, convolution channel coding, and rate matching. If the payload size is less than or equal to 11 bits, the channel coding of the CQI is performed using the following steps.

- The CQI bits are coded using a (32,O) block code. The codewords of the (32,O) block code use the following equation.

  $$b_i = \sum_{n=0}^{O-1} (o_n \cdot M_{i,n}) \bmod 2$$

  The codewords of the (32,O) block code are a linear combination of the 11 basis sequences denoted in the following table.

i	$M_{i,0}$	$M_{i,1}$	$M_{i,2}$	$M_{i,3}$	$M_{i,4}$	$M_{i,5}$	$M_{i,6}$	$M_{i,7}$	$M_{i,8}$	$M_{i,9}$	$M_{i,10}$
0	1	1	0	0	0	0	0	0	0	0	1
1	1	1	1	0	0	0	0	0	0	1	1
2	1	0	0	1	0	0	1	0	1	1	1
3	1	0	1	1	0	0	0	0	1	0	1
4	1	1	1	1	0	0	0	1	0	0	1
5	1	1	0	0	1	0	1	1	1	0	1
6	1	0	1	0	1	0	1	0	1	1	1
7	1	0	0	1	1	0	0	1	1	0	1
8	1	1	0	1	1	0	0	1	0	1	1
9	1	0	1	1	1	0	1	0	0	1	1
10	1	0	1	0	0	1	1	1	0	1	1
11	1	1	1	0	0	1	1	0	1	0	1
12	1	0	0	1	0	1	0	1	1	1	1
13	1	1	0	1	0	1	0	1	0	1	1
14	1	0	0	0	1	1	0	1	0	0	1

i	$M_{i,0}$	$M_{i,1}$	$M_{i,2}$	$M_{i,3}$	$M_{i,4}$	$M_{i,5}$	$M_{i,6}$	$M_{i,7}$	$M_{i,8}$	$M_{i,9}$	$M_{i,10}$
15	1	1	0	0	1	1	1	1	0	1	1
16	1	1	1	0	1	1	1	0	0	1	0
17	1	0	0	1	1	1	0	0	1	0	0
18	1	1	0	1	1	1	1	1	0	0	0
19	1	0	0	0	0	1	1	0	0	0	0
20	1	0	1	0	0	0	1	0	0	0	1
21	1	1	0	1	0	0	0	0	0	1	1
22	1	0	0	0	1	0	0	1	1	0	1
23	1	1	1	0	1	0	0	0	1	1	1
24	1	1	1	1	1	0	1	1	1	1	0
25	1	1	0	0	0	1	1	1	0	0	1
26	1	0	1	1	0	1	0	0	1	1	0
27	1	1	1	1	0	1	0	1	1	1	0
28	1	0	1	0	1	1	1	0	1	0	0
29	1	0	1	1	1	1	1	1	1	0	0
30	1	1	1	1	1	1	1	1	1	1	1
31	1	0	0	0	0	0	0	0	0	0	0

The output sequence is obtained by a circular repetition of the CQI/PMI block, as shown in the following equation.

$$q_i = b_{(i \bmod B)}$$

In the preceding equation, the variable B is 32.

Data and Control Multiplexing

The control and transport data multiplexing is performed such that HARQ-ACK information is present in both slots and is mapped to resources around the demodulation reference signals. The mapping is important, as it assumes the channel estimation around the DRS are of better quality. Thus, the integrity of the HARQ information is maintained.

Channel Interleaver

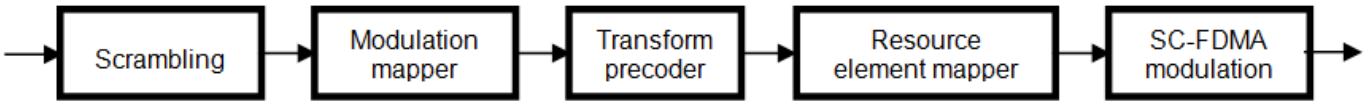
The channel interleaver implements a time-first mapping of modulation symbols onto the transmit waveform while ensuring that HARQ information is present on both slots and is mapped to resources around the DRS.

Channel Coding of Control Information without UL-SCH Data

When control data is sent on the PUSCH without UL-SCH data the following coding process can be identified: channel coding of control information, mapping and channel interleaving. The fundamental change is the number of bits used to transmit the control information. After determining the bit widths for the various pieces of coded control information, channel coding and rate matching is then performed as per section 1.6 of this document. The coded channel quality information is remapped into columns of symbols before being interleaved with the coded HI and RI.

PUSCH Processing

The Physical Uplink Shared Channel (PUSCH) carries uplink shared channel data and control information. The processing chain for the PUSCH includes scrambling, modulation mapping, precoding, resource element mapping and Single Carrier – Frequency Division Multiple Access (SC-FDMA) modulation. This processing chain is illustrated in the following figure.



Scrambling

The transport codeword is bit-wise multiplied with an orthogonal sequence and a UE-specific scrambling sequence to create the following sequence of symbols for each codeword, q.

$$\tilde{b}^{(q)}(0), \ldots, \tilde{b}^{(q)}\left(M_{bit}^{(q)} - 1\right)$$

The variable $M_{bit}^{(q)}$ is the number of bits transmitted on the PUSCH in one subframe q.

The scrambling sequence is pseudorandom, created using a length-31 Gold sequence generator and initialized using the slot number within the radio network temporary identifier associated with the PUSCH transmission, n_{RNTI}, the cell ID, N_{ID}^{cell}, the slot number within the radio frame, n_s, and the codeword index, $q = \{0, 1\}$, at the start of each subframe.

$$c_{init} = n_{RNTI} \times 2^{14} + q \times 2^{13} + \left\lfloor \frac{n_s}{2} \right\rfloor \times 2^9 + N_{ID}^{cell}$$

Scrambling with a cell-specific sequence serves the purpose of intercell interference rejection. When a base station descrambles a received bit stream with a known cell specific scrambling sequence, interference from other cells will be descrambled incorrectly and therefore only appear as uncorrelated noise.

Modulation

The scrambled codeword undergoes QPSK, 16QAM, or 64QAM modulation to generate complex valued symbols. This choice provides the flexibility to allow the scheme to maximize the data transmitted depending on the channel conditions.

Precoding

The PUSCH precoding is not the same as the in the downlink (multi-antenna) precoding. The block of complex valued symbols, $d(0), \ldots, d(M_{symb} - 1)$, is divided into M_{symb}/M_{sc}^{PUSCH} sets. Each set, which has size M_{sc}^{PUSCH}, corresponds to one SC-FDMA symbol. A Discrete Fourier Transform is then applied

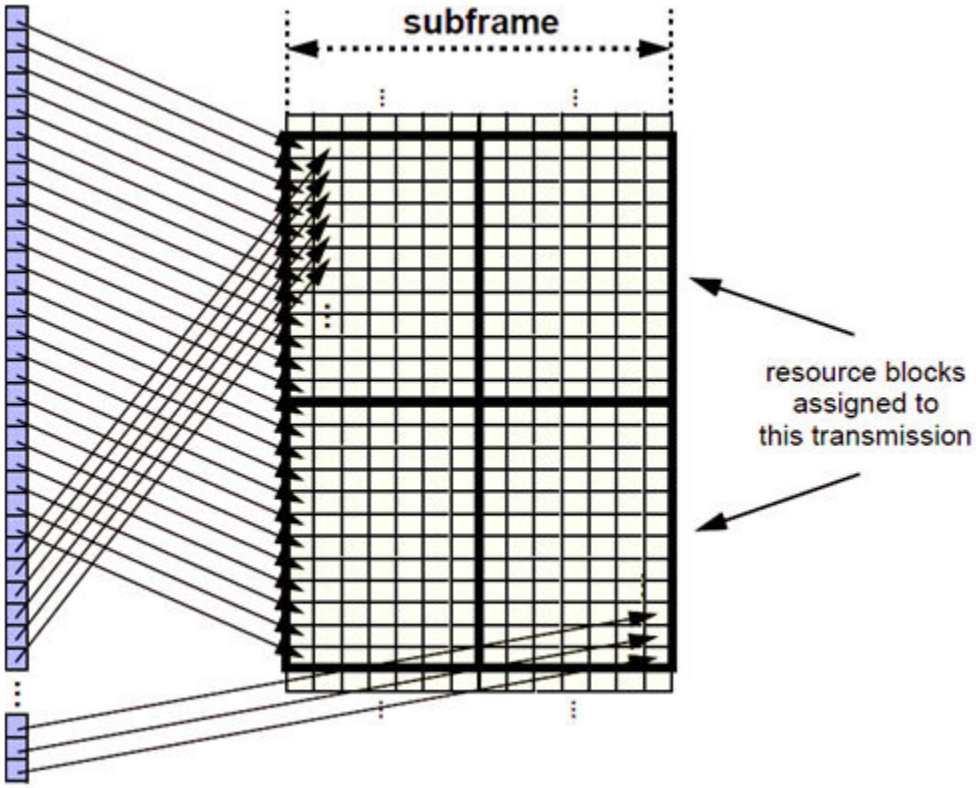
to each set, essentially precoding part of the SC-FDMA modulation. The size of the DFT, which is the value of M_{sc}^{PUSCH}, must have a prime that is a product of 2, 3, or 5, thereby fulfilling the following equation.

$$M_{\mathrm{sc}}^{\mathrm{PUSCH}} = N_{\mathrm{sc}}^{\mathrm{RB}} \times 2^{\alpha 2} \times 3^{\alpha 3} \times 5^{\alpha 5} \leq N_{\mathrm{sc}}^{\mathrm{RB}} N_{\mathrm{RB}}^{\mathrm{UL}}$$

In the preceding equation, α_2, α_3, and α_5 are a set of nonnegative integers.

Mapping to Resource Elements

The final stage in the PUSCH processing is to map the symbols to the allocated physical resource elements. The allocation sizes are limited to values whose prime factors are 2, 3 and 5; this limit is imposed by the precoding stage. The symbols are mapped in increasing order beginning with subcarriers, then SC-FDMA symbols. SC-FDMA symbols carrying DRS or SRS are avoided during the mapping process. An example of the order of mapping the output of the precoding stage to the allocated resource blocks is shown in the following figure.



Demodulation Reference Signals (DM-RS) on the PUSCH

- “DRS Generation” on page 1-102
- “DRS Resource Mapping” on page 1-104

Demodulation reference signals associated with the PUSCH are used by the base station to perform channel estimation and allow for coherent demodulation of the received signal.

These reference signals are time-multiplexed with data, whereas in the downlink there is both time and frequency multiplexing. This multiplexing is performed to maintain the single-carrier nature of the SC-FDMA signal, which ensures that all data carriers are contiguous.

DRS Generation

- "Base Sequence" on page 1-102
- "DRS Grouping" on page 1-103

The demodulation reference signals are generated using a base sequence denoted by $r_{u,v}(n)$, which is discussed further in "Base Sequence" on page 1-102. More specifically, r^{PUSCH} is used to denote the PUSCH DRS sequence and is defined by the following equation.

$$r^{PUSCH}\left(mM_{SC}^{RS}+n\right)=r_{u,v}^{(\alpha)}(n)$$

It is desired that the DRS sequences have small power variations in time and frequency, resulting in high power amplifier efficiency and comparable channel estimation quality for all frequency components. Zadoff-Chu sequences are good candidates, since they exhibit constant power in time and frequency. However, there are a limited number of Zadoff-Chu sequences; therefore, they are not suitable on their own.

The generation and mapping of the DRS associated with the PUSCH are discussed further in the following sections.

Base Sequence

The demodulation reference signals are defined by a cyclic shift, α, of a base sequence, r.

The base sequence, r, is represented in the following equation.

$$r_{u,v}^{(\alpha)}=e^{j\alpha n}r_{u,v}(n)$$

The preceding equation contains the following variables.

- $n=0,\ldots,M_{SC}^{RS}$, where M_{SC}^{RS} is the length of the reference signal sequence.
- $U=0,\ldots,29$ is the base sequence group number.
- $V=0,1$ is the sequence number within the group and only applies to reference signals of length greater than 6 resource blocks.

A phase rotation in the frequency domain (pre-IFFT in the OFDM modulation) is equivalent to a cyclic shift in the time domain (post IFFT in the OFDM modulation). For frequency non-selective channels over the 12 subcarriers of a resource block, it is possible to achieve orthogonality between DRS generated from the same base sequence if $\alpha=m\pi/6$ for $m=0,1,\ldots,11$, and assuming the DRS are synchronized in time.

The orthogonality can be exploited to transmit DRS at the same time, using the same frequency resources without mutual interference. Generally, DRS generated from different base sequences will not be orthogonal; however they will present low cross-correlation properties.

To maximize the number of available Zadoff-Chu sequences, a prime length sequence is needed. The minimum sequence length in the UL is 12, the number of subcarriers in a resource block, which is not prime.

Therefore, Zadoff-Chu sequences are not suitable by themselves. There are effectively the following two types of base reference sequences.

- those with a sequence length ≥ 36 (spanning 3 or more resource blocks), which use a cyclic extension of Zadoff-Chu sequences
- those with a sequence length ≤ 36 (spanning 2 resource blocks), which use a special QPSK sequence

Base sequences of length ≥ three resource blocks

For sequences of length 3 resource blocks and larger (i.e. $M_{sc}^{RS} \geq 3N_{sc}^{RS}$), the base sequence is a repetition, with a cyclic offset of a Zadoff-Chu sequence of length N_{zc}^{RS}, where N_{zc}^{RS} is the largest prime such that $N_{zc}^{RS} < M_{sc}^{RS}$. Therefore, the base sequence will contain one complete length N_{zc}^{RS} Zadoff-Chu sequence plus a fractional repetition appended on the end. At the receiver the appropriate de-repetition can be done and the zero autocorrelation property will hold across the length N_{zc}^{RS} vector.

Base sequences of length ≤ three resource blocks

For sequences shorter than three resource blocks (i.e. $M_{sc}^{RS} = 12, 24$), the sequences are a composition of unity modulus complex numbers drawn from a simulation generated table. These sequences have been found through computer simulation and are specified in the LTE specifications.

DRS Grouping

There are a total of 30 sequence groups, $u \in \{0, 1, \ldots, 29\}$, each containing one sequence for length less than or equal to 60. This corresponds to transmission bandwidths of 1,2,3,4 and 5 resource blocks. Additionally, there are two sequences (one for $v = 0$ or 1) for length ≥ 72; corresponding to transmission bandwidths of 6 resource blocks or more.

Note that not all values of m are allowed, where m is the number of resource blocks used for transmission. Only values for m that are the product of powers of 2, 3 and 5 are valid, as shown in the following equation.

$m = 2^{\alpha 0} \times 3^{\alpha 1} \times 5^{\alpha 2}$, where α_i are positive integers

The reason for this restriction is that the DFT sizes of the SC-FDMA precoding operation are limited to values which are the product of powers of 2, 3 and 5. The DFT operation can span more than one resource block, and since each resource block has 12 subcarriers, the total number of subcarriers fed to the DFT will be $12m$. Since the result of $12m$ has to be the product of powers of 2, 3 and 5 this implies that the number of resource blocks must themselves be the product of powers of 2, 3 and 5. Therefore values of m such as 7, 11, 14, 19, etc. are not valid.

For a given time slot, the uplink reference signal sequences to use within a cell are taken from one specific sequence group. If the same group is to be used for all slots then this is known as fixed assignment. On the other hand, if the group number u varies for all slots within a cell this is known as group hopping.

Fixed Group Assignment

When fixed group assignment is used, the same group number is used for all slots. For PUSCH, the group number is a function of the cell identity number modulo 30 and is signaled by higher layers through the parameter Δ_{ss}, as shown in the following equation.

$$u = \left(\left(N_{ID}^{cell} mod 30\right) + \Delta_{ss}\right) mod 30, \text{ with } \Delta_{ss} \in \{0, 1, \ldots, 29\}$$

This method allows the same u to be used in different cells.

Group Hopping

If group hopping is used, the pattern is applied to the calculation of the sequence group number. This pattern is the same for both the PUCCH and PUSCH.

This pattern is defined as the following equation.

$$f_{gh}(n_s) = \sum_{i=0}^{7} c(8n_s + i) \cdot 2^i mod 30$$

As shown in the preceding equation, this group hopping pattern is a function of the slot number n_s and is calculated making use of a pseudorandom binary sequence $c(k)$, generated using a length-30 Gold code. To generate the group hopping pattern, the PRBS generator is initialized with the following value at the start of each radio frame.

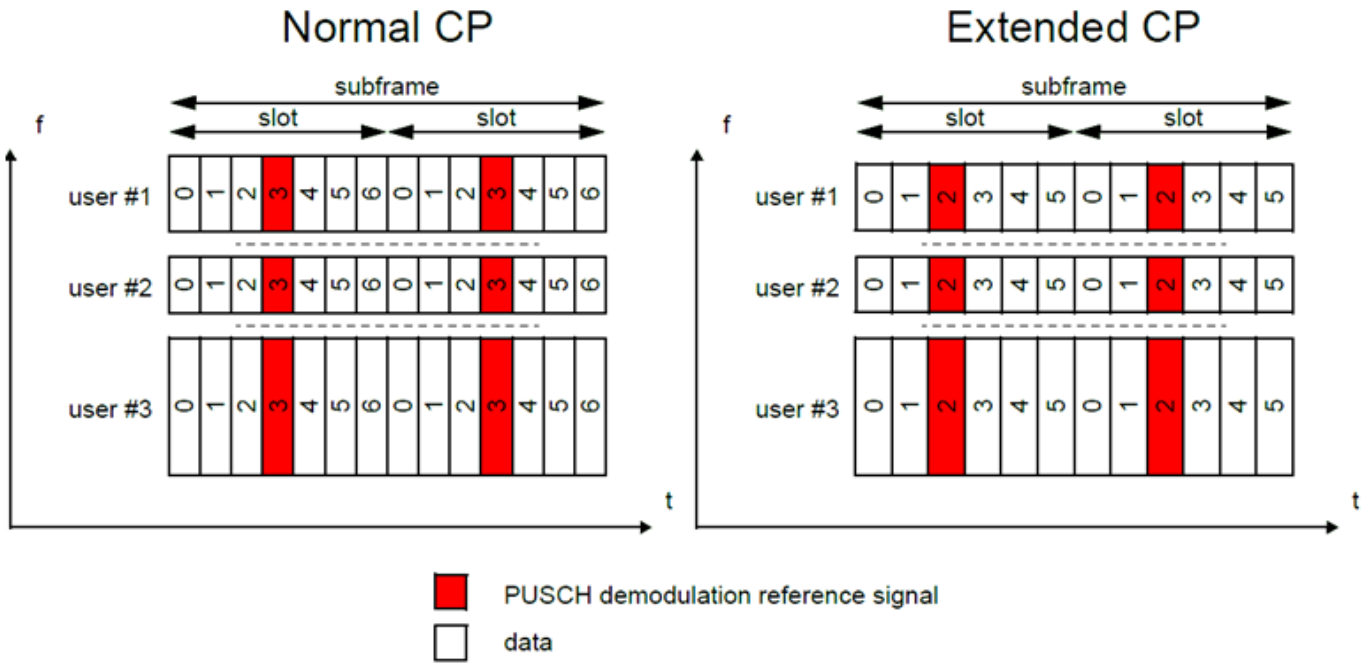
$$c_{init} = \left\lfloor \frac{N_{ID}^{cell}}{30} \right\rfloor$$

For PUSCH with group hopping, the group number, u, is given by the following equation.

$$u = \left(f_{gh}(n_s) + \left(\left(N_{ID}^{cell} \text{mod} 30\right) + \Delta_{ss}\right)\right) \text{mod} 30$$

DRS Resource Mapping

The PUSCH demodulation reference signal is mapped to the 4th SC-FDMA symbol of the slot during normal cyclic prefix and to every 3rd SC-FDMA slot during extended cyclic prefix. This resource mapping is shown in the following figure.



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` | `lteULSCHInfo` | `ltePUSCH` | `ltePUSCHIndices` | `ltePUSCHDRS` | `ltePUSCHDRSIndices` | `lteULResourceGrid`

Related Examples

- “Model UL-SCH and PUSCH” on page 3-20

Propagation Channel Models

In this section...

The LTE Toolbox product provides a set of channel models for the test and verification of UE and eNodeB radio transmission and reception as defined in [1] and [2]. The following channel models are available in the LTE Toolbox product.

- Multipath fading propagation conditions
- High speed train conditions
- Moving propagation conditions

Multipath Fading Propagation Conditions

The multipath fading channel model specifies the following three delay profiles.

- Extended Pedestrian A model (EPA)
- Extended Vehicular A model (EVA)
- Extended Typical Urban model (ETU)

These three delay profiles represent a low, medium, and high delay spread environment, respectively. The multipath delay profiles for these channels are shown in the following tables.

EPA Delay Profile

Excess tap delay (ns)	Relative power (dB)
0	0.0
30	–1.0
70	–2.0
90	–3.0
110	–8.0
190	–17.2
410	–20.8

EVA Delay Profile

Excess tap delay (ns)	Relative power (dB)
0	0.0
30	–1.5
150	–1.4
310	–3.6
370	–0.6
710	–9.1
1090	–7.0
1730	–12.0
2510	–16.9

ETU Delay Profile

Excess tap delay (ns)	Relative power (dB)
0	–1.0
50	–1.0
120	–1.0
200	0.0
230	0.0
500	0.0
1600	–3.0
2300	–5.0
5000	–7.0

All the taps in the preceding tables have a classical Doppler spectrum. In addition to multipath delay profile, a maximum Doppler frequency is specified for each multipath fading propagation condition, as shown in the following table.

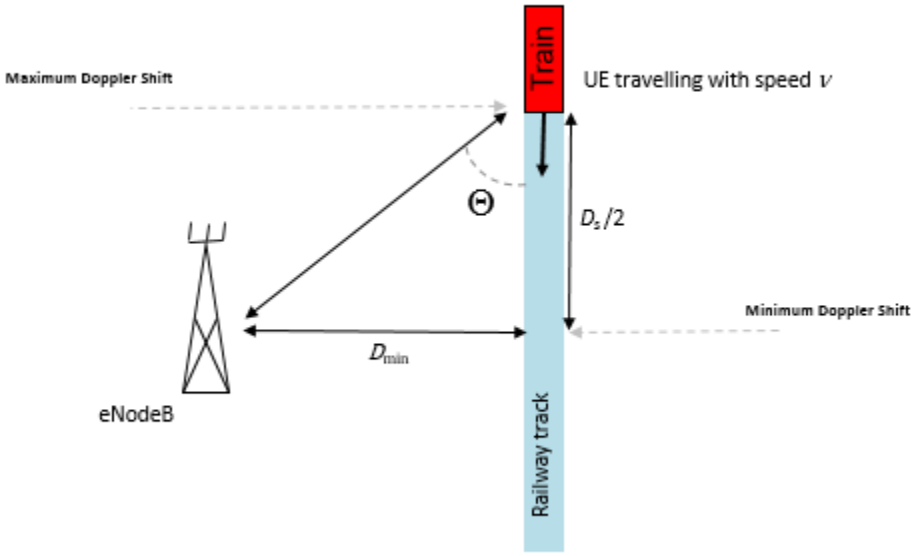
Channel model	Maximum Doppler frequency
EPA 5Hz	5 Hz
EVA 5Hz	5 Hz
EVA 70Hz	70 Hz
ETU 70Hz	70 Hz
ETU 300Hz	300 Hz

In the case of MIMO environments, a set of correlation matrices is introduced to model the correlation between UE and eNodeB antennas. These correlation matrices are introduced in “MIMO Channel Correlation Matrices” on page 1-112.

High Speed Train Condition

The high speed train condition defines a non-fading propagation channel with single multipath component, the position of which is fixed in time. This single multipath represents the Doppler shift,

which is caused due to a high speed train moving past a base station, as shown in the following figure.



The expression $D_s/2$ is the initial distance of the train from eNodeB, and $D_{\min}$ is the minimum distance between eNodeB and the railway track. Both variables are measured in meters. The variable v is the velocity of the train in meters per second. The Doppler shift due to a moving train is given in the following equation.

$$f_s(t) = f_d \cos\theta(t)$$

The variable $f_s(t)$ is the Doppler shift and f_d is the maximum Doppler frequency. The cosine of angle $\theta(t)$ is given by the following equation.

$$\cos\theta(t) = \frac{D_s/2 - vt}{\sqrt{D_{\min}{}^2 + (D_s/2 - vt)^2}}, 0 \le t \le D_s/v$$

$$\cos\theta(t) = \frac{-1.5D_s + vt}{\sqrt{D_{\min}{}^2 + (-1.5D_s + vt)^2}}, D_s/v < t \le 2D_s/v$$

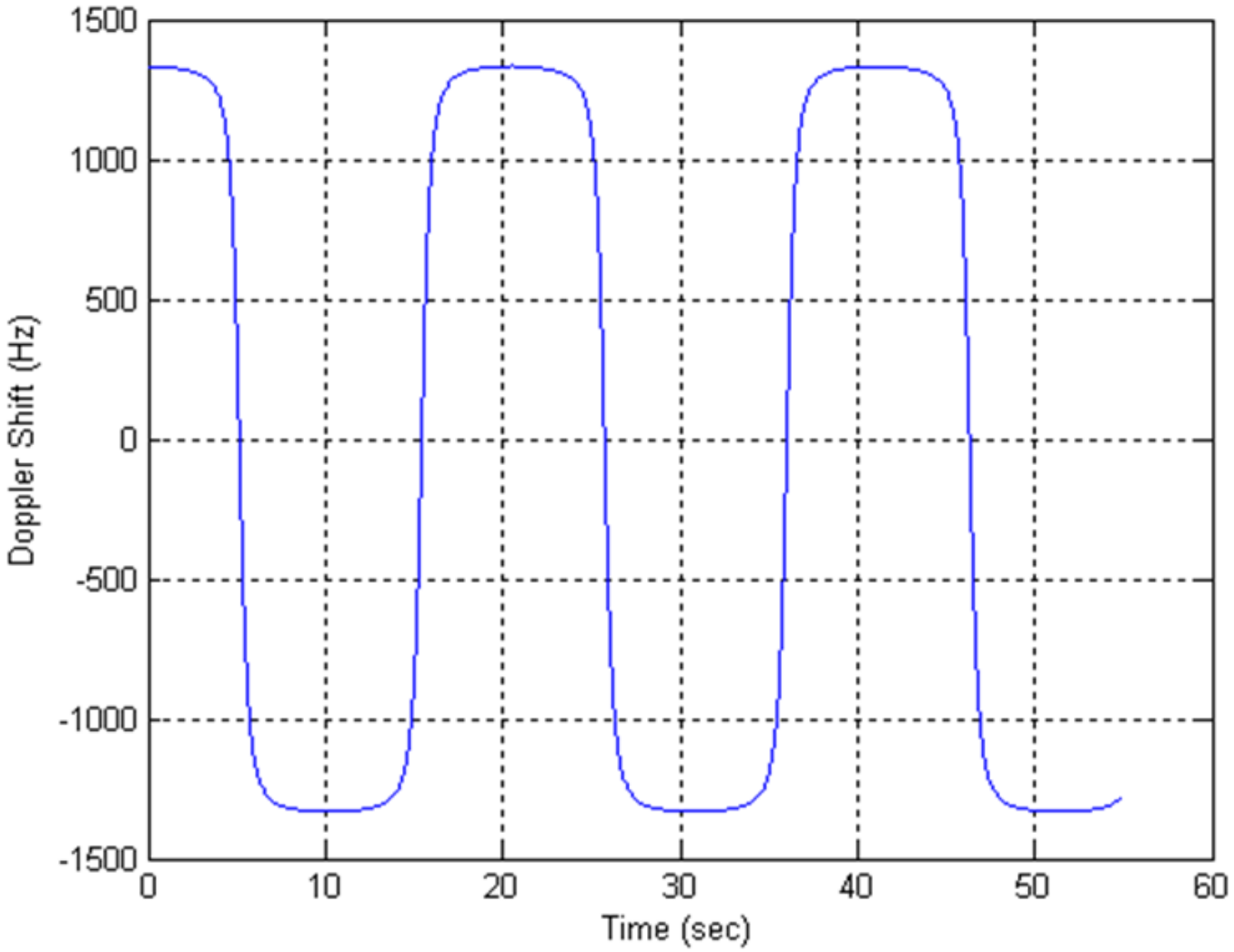
$$\cos\theta(t) = \cos\theta(t \mod (2D_s/v)), t > 2D_s/v$$

For eNodeB testing, two high speed train scenarios are defined that use the parameters listed in the following table. The Doppler shift, $f_s(t)$, is calculated using the preceding equations and the parameters listed in the following table.

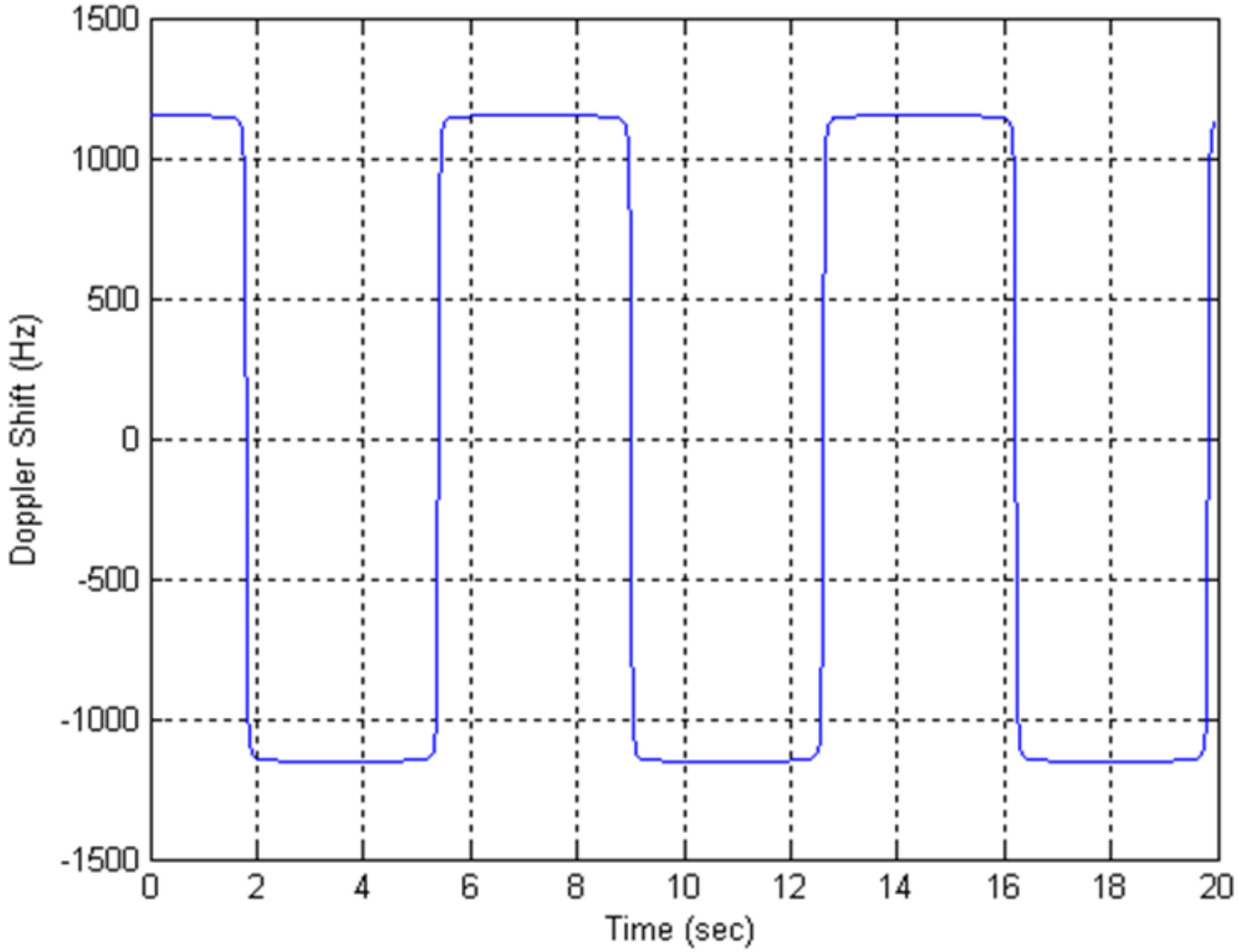
Parameter	Value	
	Scenario 1	Scenario 3
D_s	1,000 m	300 m

Parameter	Value	
$D_{\min}$	50 m	2 m
ν	350 km/hr	300 km/kr
f_d	1,340 Hz	1,150 Hz

Both of these scenarios result in Doppler shifts that apply to all frequency bands. The Doppler shift trajectory for scenario 1 is shown in the following figure.



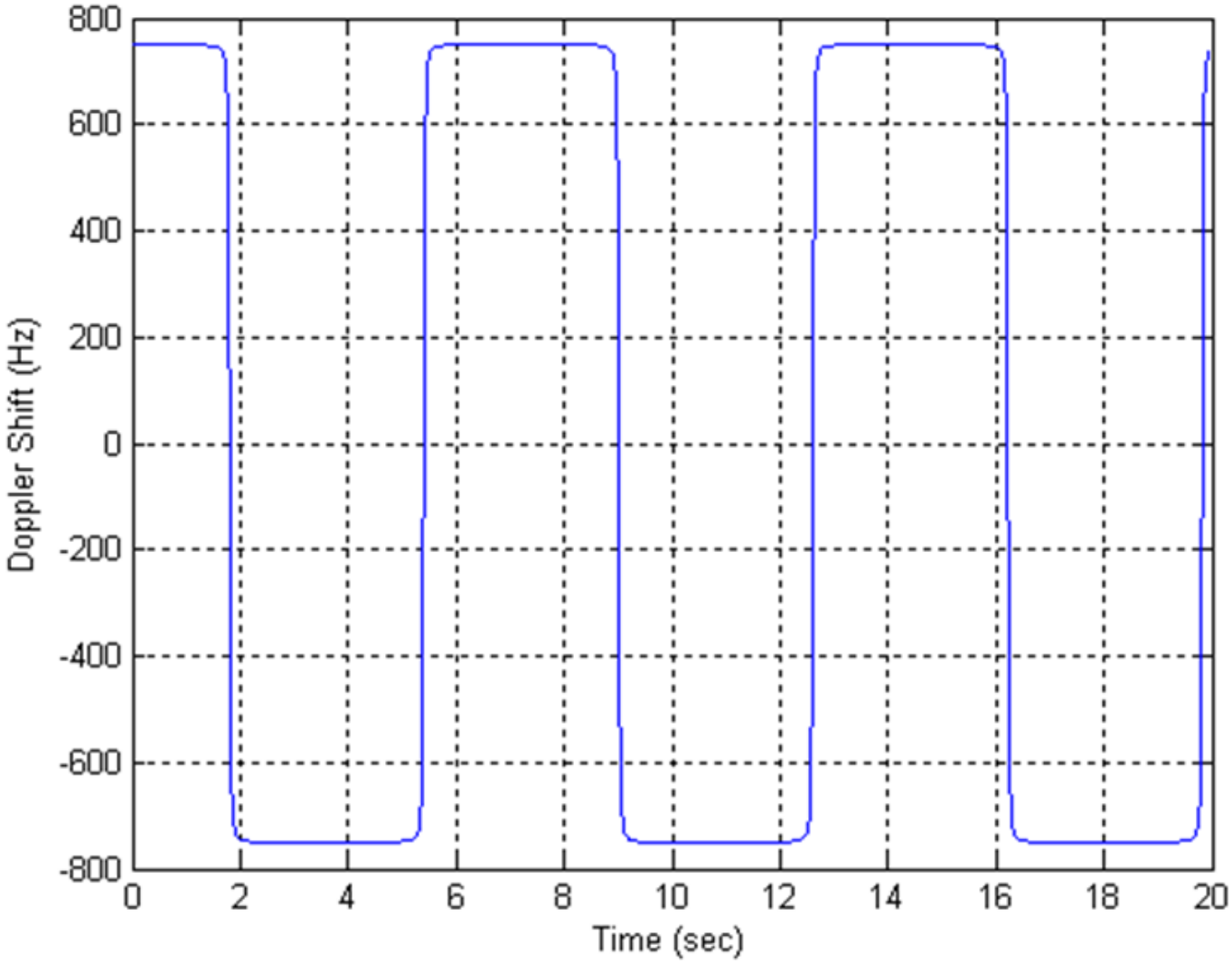
The Doppler shift trajectory for scenario 3 is shown in the following figure.



For UE testing, the Doppler shift, $f_s(t)$, is calculated using the preceding equations and the parameters listed in the following table.

Parameter	Value
D_s	300 m
$D_{\min}$	2 m
ν	300 km/hr
f_d	750 Hz

These parameters result in the Doppler shift, applied to all frequency bands, shown in the following figure.



Moving Propagation Condition

The moving propagation channel in LTE defines a channel condition where the location of multipath components changes. The time difference between the reference time and the first tap, $\Delta\tau$, is given by the following equation.

$$\Delta\tau = \frac{A}{2} \cdot \sin(\Delta\omega \cdot t)$$

The variable A represents the starting time in seconds and $\Delta\omega$ represents angular rotation in radians per second.

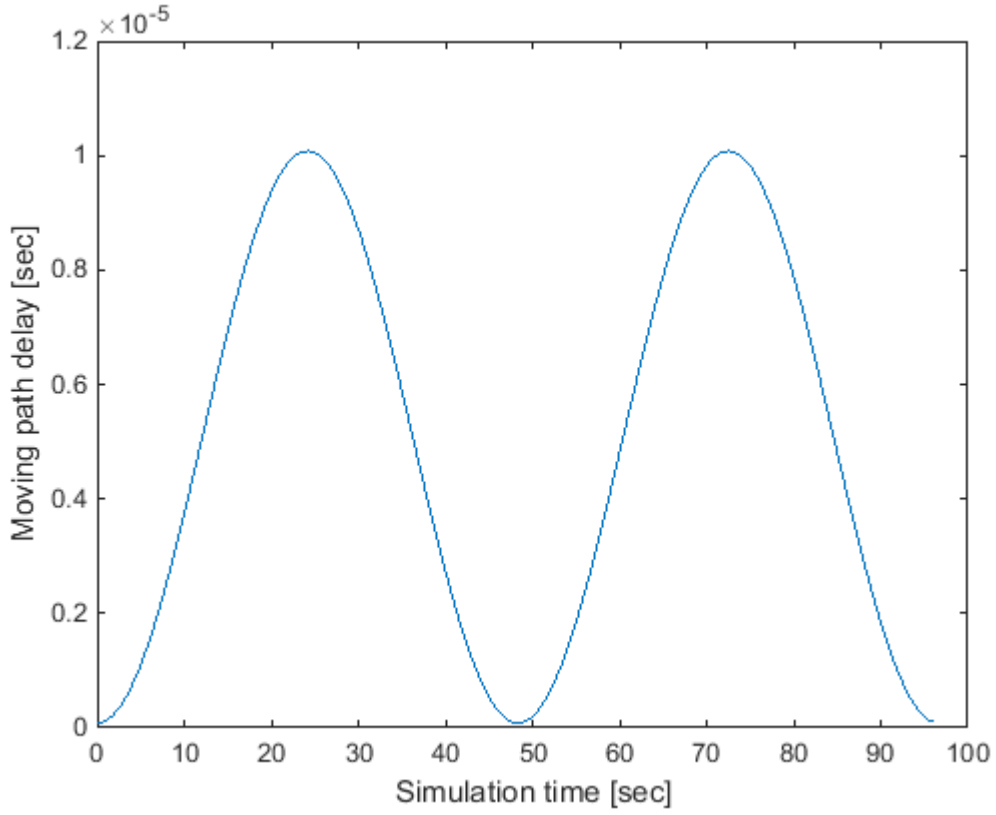
Note Relative time between multipath components stays fixed.

The parameters for the moving propagation conditions are shown in the following table.

Parameter	Scenario 1	Scenario 2
Channel model	ETU200	AWGN
UE speed	120 km/hr	350 km/hr
CP length	Normal	Normal
A	10 µs	10 µs
$\Delta\omega$	0.04 s^{-1}	0.13 s^{-1}

Doppler shift only applies for generating fading samples for scenario 1. In scenario 2, a single non-fading multipath component with additive white gaussian noise (AWGN) is modeled. The location of this multipath component changes with time, according to the preceding equation.

An example of a moving channel with a single non-fading tap is shown in the following figure.



MIMO Channel Correlation Matrices

In MIMO systems, there is correlation between transmit and receive antennas. This depends on a number of factors such as the separation between antenna and the carrier frequency. For maximum capacity, it is desirable to minimize the correlation between transmit and receive antennas.

There are different ways to model antenna correlation. One technique makes use of correlation matrices to describe the correlation between multiple antennas both at the transmitter and the receiver. These matrices are computed independently at both the transmitter-receiver and then combined by means of a Kronecker product in order to generate a channel spatial correlation matrix.

Three different correlation levels are defined in [1].

1. low or no correlation
2. medium correlation
3. high correlation

The parameters α and β are defined for each level of correlation as shown in the following table of correlation values.

Low correlation		Medium correlation		High correlation	
α	β	α	β	α	β
0	0	0.3	0.9	0.9	0.9

The independent correlation matrices at eNodeB and UE, R_{eNB} and R_{UE}, respectively, are shown for different set of antennas (1, 2 and 4) in the following table.

Correlation	One antenna	Two antennas	Four antennas
eNodeB	$R_{eNB} = 1$	$R_{eNB} = \begin{pmatrix} 1 & \alpha \\ \alpha^* & 1 \end{pmatrix}$	$R_{eNB} = \begin{pmatrix} 1 & \alpha^{1/9} & \alpha^{4/9} & \alpha \\ \alpha^{1/9*} & 1 & \alpha^{1/9} & \alpha^{4/9} \\ \alpha^{4/9*} & \alpha^{1/9*} & 1 & \alpha^{1/9} \\ \alpha^* & \alpha^{4/9*} & \alpha^{1/9*} & 1 \end{pmatrix}$
UE	$R_{UE} = 1$	$R_{UE} = \begin{pmatrix} 1 & \beta \\ \beta^* & 1 \end{pmatrix}$	$R_{UE} = \begin{pmatrix} 1 & \beta^{1/9} & \beta^{4/9} & \beta \\ \beta^{1/9*} & 1 & \beta^{1/9} & \beta^{4/9} \\ \beta^{4/9*} & \beta^{1/9*} & 1 & \beta^{1/9} \\ \beta^* & \beta^{4/9*} & \beta^{1/9*} & 1 \end{pmatrix}$

The channel spatial correlation matrix, R_{spat}, is given by the following equation.

$$R_{spat} = R_{eNB} \otimes R_{UE}$$

The symbol $\otimes$ represents the Kronecker product. The values of the channel spatial correlation matrix, R_{spat}, for different matrix sizes are defined in the following table.

Matrix size	R_{spat} values
1×2 case	$R_{spat} = R_{UE} = \begin{pmatrix} 1 & \beta \\ \beta^* & 1 \end{pmatrix}$
2×2 case	$R_{spat} = R_{eNB} \otimes R_{UE} = \begin{pmatrix} 1 & \alpha \\ \alpha^* & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & \beta \\ \beta^* & 1 \end{pmatrix} = \begin{pmatrix} 1 & \beta & \alpha & \alpha\beta \\ \beta^* & 1 & \alpha\beta^* & \alpha \\ \alpha^* & \alpha^*\beta & 1 & \beta \\ \alpha^*\beta^* & \alpha^* & \beta^* & 1 \end{pmatrix}$

Matrix size	R_{spat} values
4×2 case	$R_{spat} = R_{eNB} \otimes R_{UE} = \begin{pmatrix} 1 & \alpha^{1/9} & \alpha^{4/9} & \alpha \\ \alpha^{1/9*} & 1 & \alpha^{1/9} & \alpha^{4/9} \\ \alpha^{4/9*} & \alpha^{1/9*} & 1 & \alpha^{1/9} \\ \alpha^{*} & \alpha^{4/9*} & \alpha^{1/9*} & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & \beta \\ \beta^{*} & 1 \end{pmatrix}$
4×4 case	$R_{spat} = R_{eNB} \otimes R_{UE} = \begin{pmatrix} 1 & \alpha^{1/9} & \alpha^{4/9} & \alpha \\ \alpha^{1/9*} & 1 & \alpha^{1/9} & \alpha^{4/9} \\ \alpha^{4/9*} & \alpha^{1/9*} & 1 & \alpha^{1/9} \\ \alpha^{*} & \alpha^{4/9*} & \alpha^{1/9*} & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & \beta^{1/9} & \beta^{4/9} & \beta \\ \beta^{1/9*} & 1 & \beta^{1/9} & \beta^{4/9} \\ \beta^{4/9*} & \beta^{1/9*} & 1 & \beta^{1/9} \\ \beta^{*} & \beta^{4/9*} & \beta^{1/9*} & 1 \end{pmatrix}$

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` | `lteHSTChannel` | `lteMovingChannel`

Related Examples

- “Simulate Propagation Channels” on page 3-22

Channel Estimation

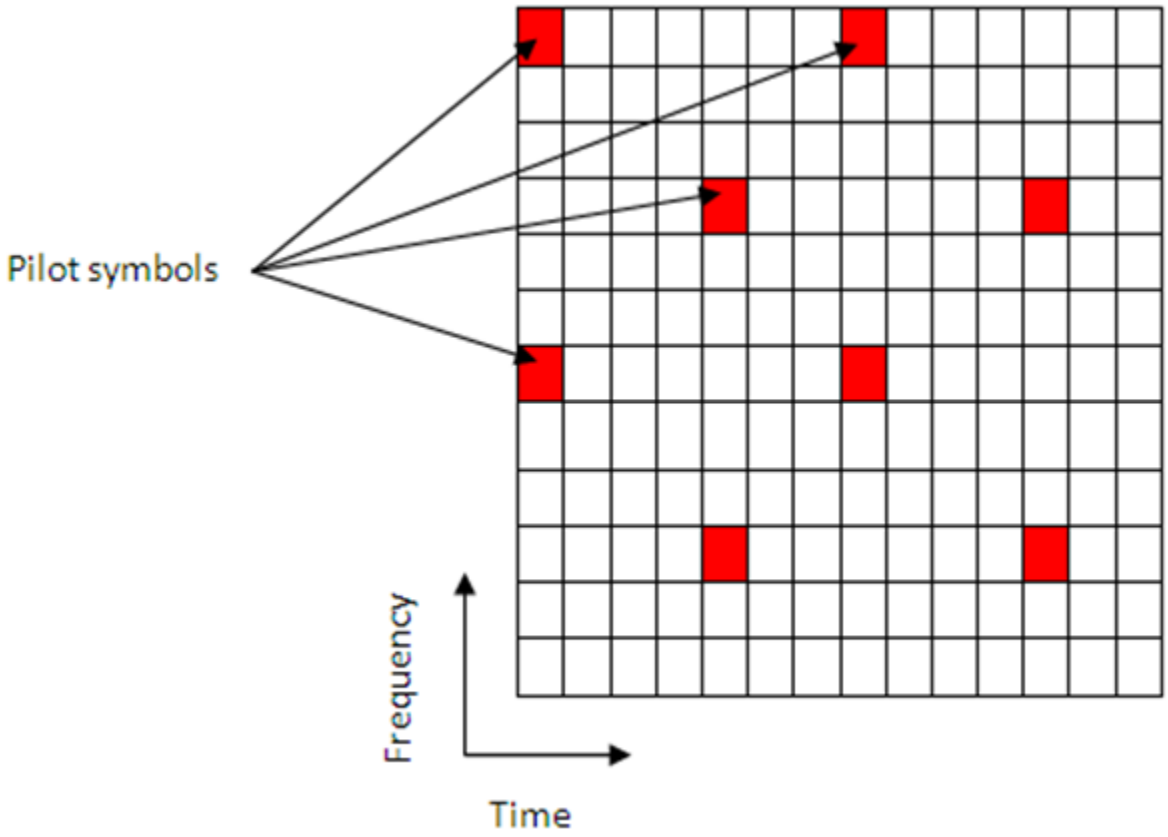
In this section...

The LTE Toolbox product uses orthogonal frequency division multiplexing (OFDM) as its digital multicarrier modulation scheme. Channel estimation plays an important part in an OFDM system. It is used for increasing the capacity of orthogonal frequency division multiple access (OFDMA) systems by improving the system performance in terms of bit error rate.

To facilitate the estimation of the channel characteristics, LTE uses cell-specific reference signals (pilot symbols) inserted in both time and frequency. These pilot symbols provide an estimate of the channel at given locations within a subframe. Through interpolation, it is possible to estimate the channel across an arbitrary number of subframes.

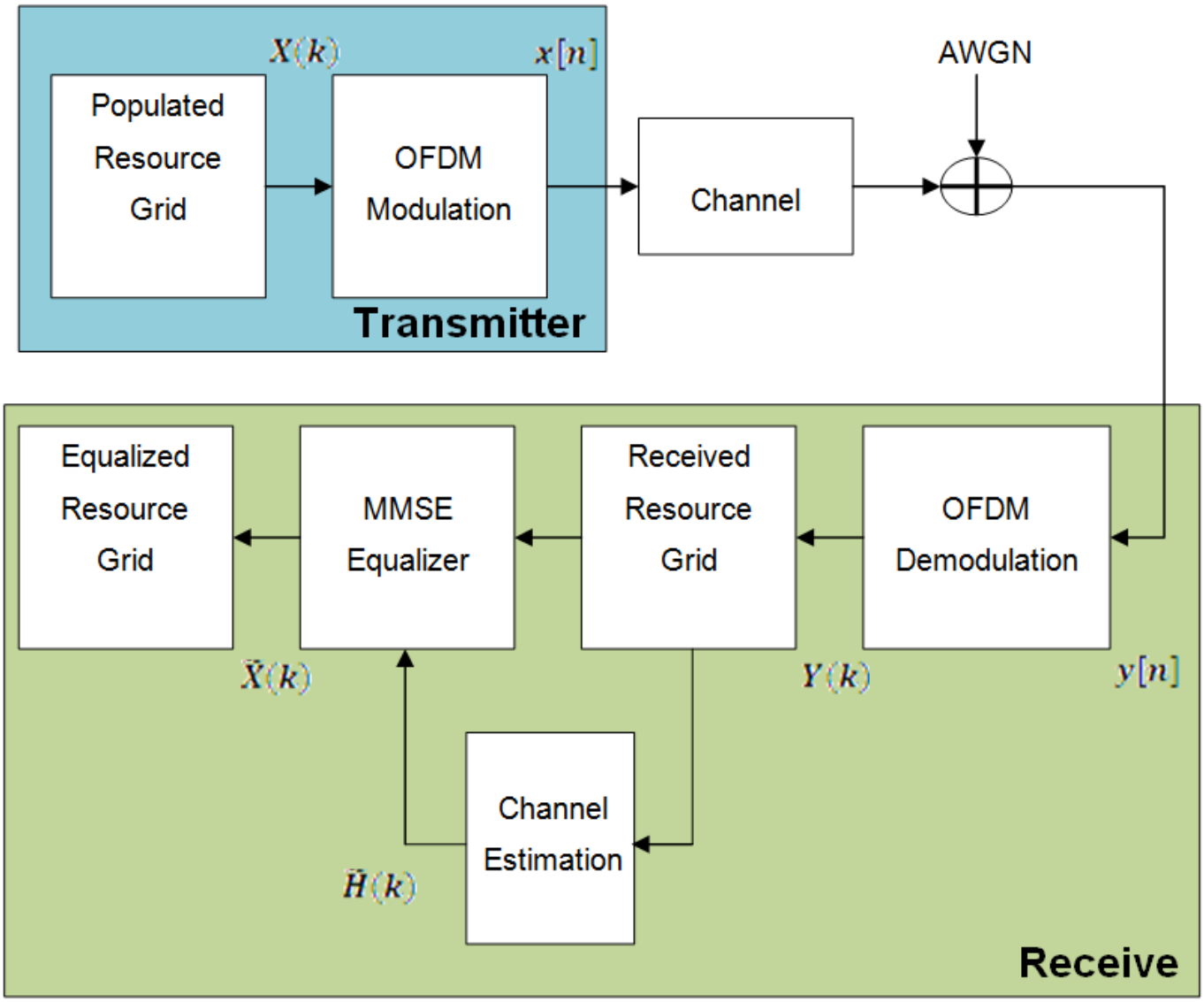
Channel Estimation Overview

The pilot symbols in LTE are assigned positions within a subframe depending on the eNodeB cell identification number and which transmit antenna is being used, as shown in the following figure.



The unique positioning of the pilots ensures that they do not interfere with one another and can be used to provide a reliable estimate of the complex gains imparted onto each resource element within the transmitted grid by the propagation channel.

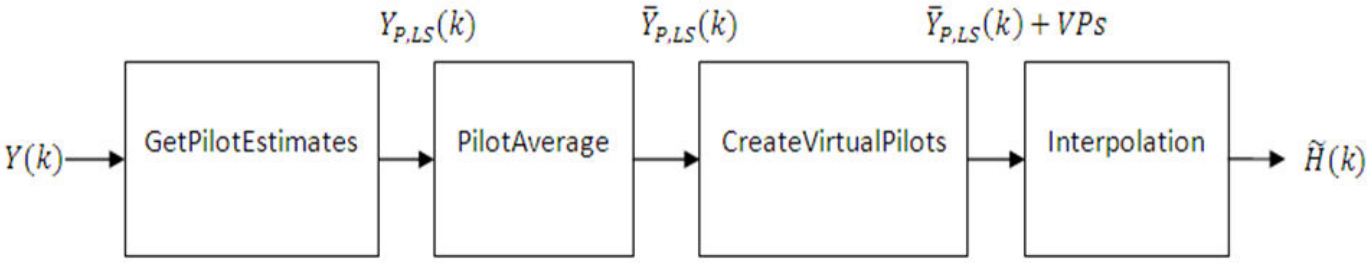
Both transmit and receive chains and the propagation channel model are shown in the following block diagram.



The populated resource grid represents several subframes containing data. This grid is then OFDM modulated and passed through the model of the propagation channel. Channel noise in the form of additive white Gaussian noise (AWGN) is added before the signal enters the receiver. Once inside the receiver the signal is OFDM demodulated and a received resource grid can be constructed. The received resource grid contains the transmitted resource elements which have been affected by the complex channel gains and the channel noise. Using the known pilot symbols to estimate the channel, it is possible to equalize the effects of the channel and reduce the noise on the received resource grid.

LTE assigns each antenna port a unique set of locations within a subframe to which to map reference signals. Because no other antenna transmits data at these locations in time and frequency, channel estimation for multi-antenna configurations can be performed. The channel estimation algorithm extracts the reference signals for a transmit/receive antenna pair from the received grid. The least squares estimates of the channel frequency response at the pilot symbols are calculated as described in *On Channel Estimation in OFDM Systems* [2]. The least squares estimates are then averaged to

reduce any unwanted noise from the pilot symbols. Because it is possible that no pilots are located near the subframe edge, virtual pilot symbols are created to aid the interpolation process near the edge of the subframe. Using the averaged pilot symbol estimates and the calculated virtual pilot symbols, interpolation is then carried out to estimate the entire subframe. This process is demonstrated in the following block diagram.



Get Pilot Estimates Subsystem

The first step in determining the least squares estimate is to extract the pilot symbols from their known location within the received subframe. Because the value of these pilot symbols is known, the channel response at these locations can be determined using the least squares estimate. The least squares estimate is obtained by dividing the received pilot symbols by their expected value.

$$Y(k) = H(k)X(k) + noise$$

Where:

- $Y(k)$ is a received complex symbol value.
- $X(k)$ is a transmitted complex symbol value.
- $H(k)$ is a complex channel gain experienced by a symbol.

Known pilot symbols can be sent to estimate the channel for a subset of REs within a subframe. In particular, if pilot symbol $X_P(k)$ is sent in an RE, an instantaneous channel estimate $\tilde{H}_P(k)$ for that RE can be computed using:

$$\tilde{H}_P(k) = \frac{Y_P(k)}{X_P(k)} = H_P(k) + noise$$

Where:

- $Y_P(k)$ represents the received pilot symbol values.
- $X_P(k)$ represents the known transmitted pilot symbol values.
- $\tilde{H}_P(k)$ is the true channel response for the RE occupied by the pilot symbol.

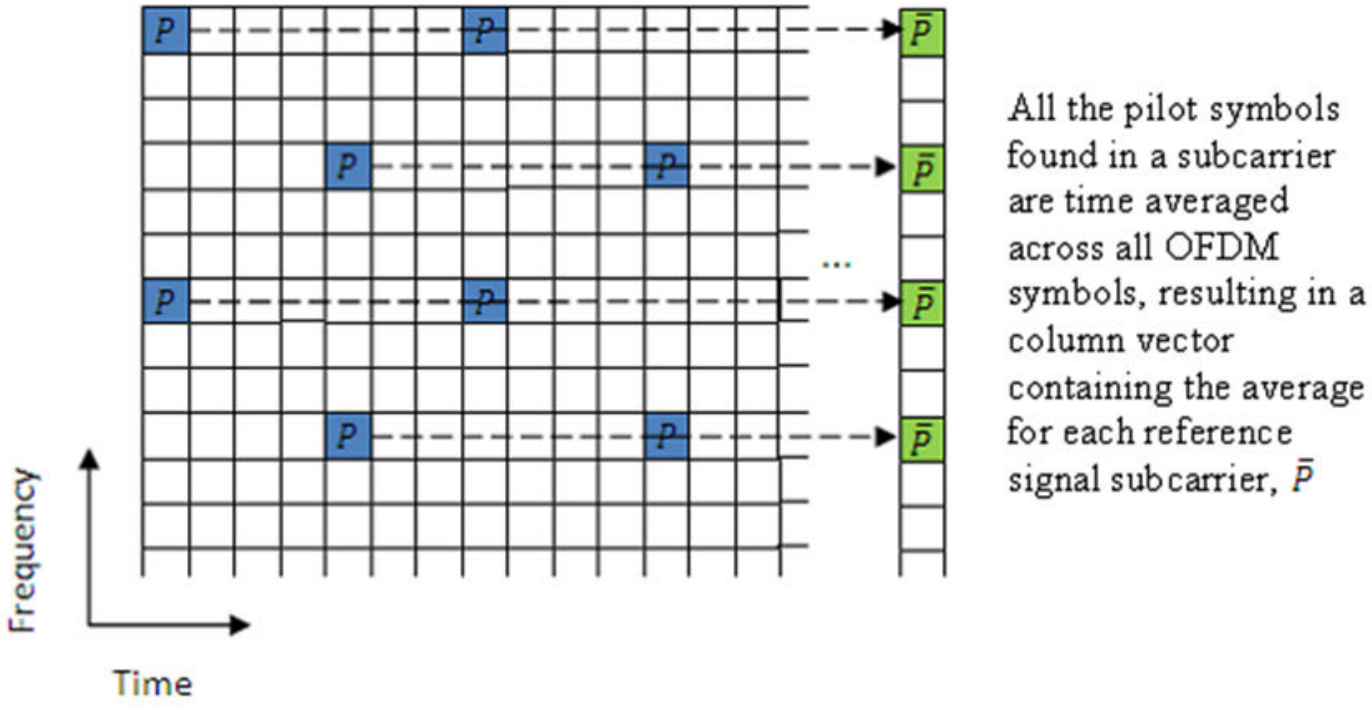
Pilot Average Subsystem

To minimize the effects of noise on the channel estimates, the least square estimates are averaged using an averaging window. This simple method produces a substantial reduction in the level of noise found on the pilot REs. The following two pilot symbol averaging methods are available.

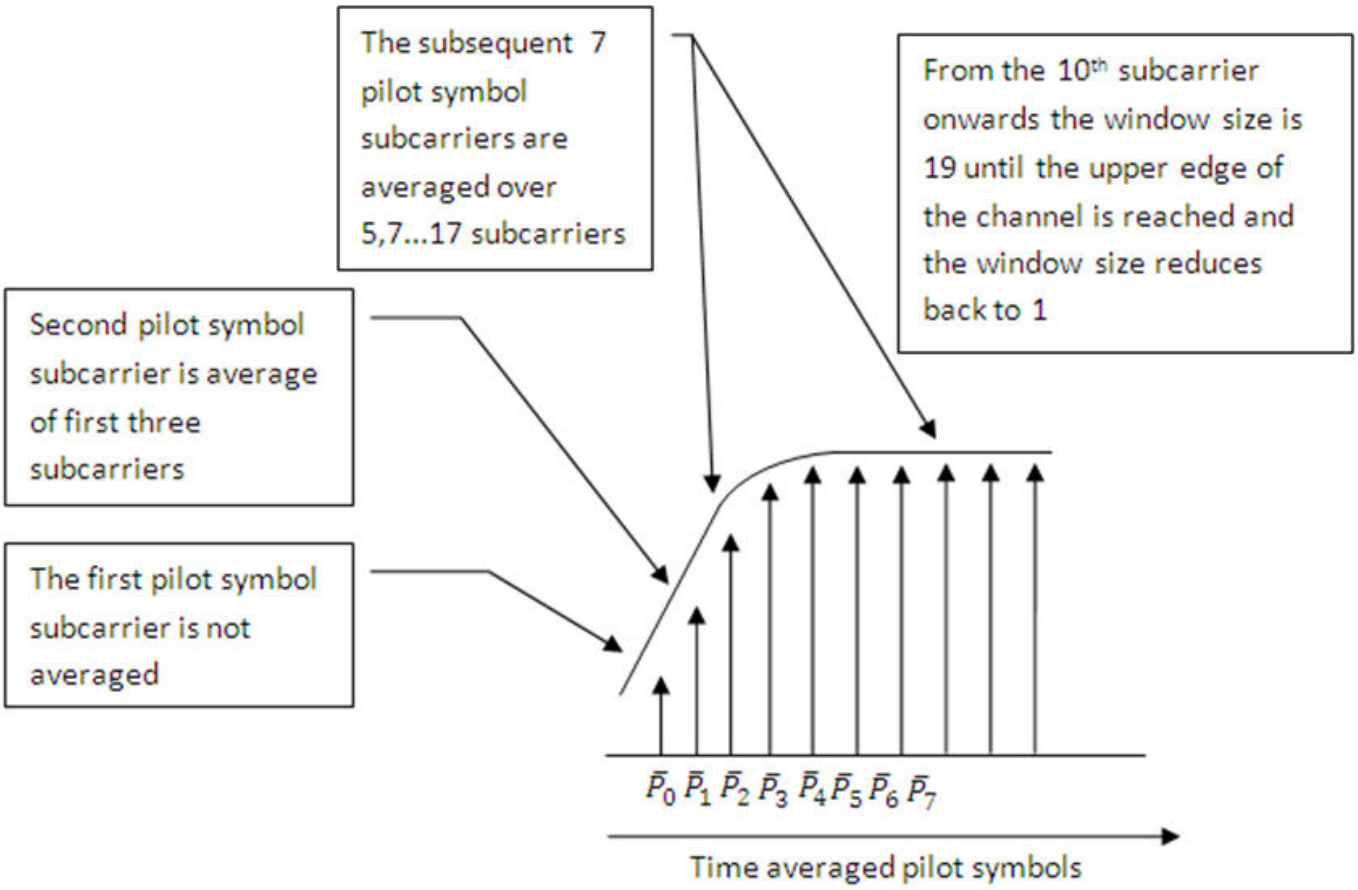
- `'TestEVM'` — follows the method described in TS 36.141 [1], Annex F.3.4.
- `'UserDefined'` — allows you to define the window size and direction of averaging used on the pilot symbols plus other settings used for the interpolation.

'TestEVM'

The first method, `'TestEVM'`, uses the approach described in TS 36.141 [1], Annex F.3.4. Time averaging is performed across each subcarrier that contains a pilot symbol, resulting in a column vector containing an average amplitude and phase for each subcarrier that is carrying a reference signal.



The averages of the pilot symbol subcarriers are then frequency averaged using a moving window of maximum size 19.



Note When using `'TestEVM'` pilot symbol averaging, there are no user-defined parameters and control of channel estimation parameters is not possible. The estimation is performed using the method described in TS 36.141 [1]. Except that averaging across 10 subframes is not strictly required. The `lteDLChannelEstimate` function averages across the number of subframes included in the input `rxgrid`. The greater the number of subframes in `rxgrid`, the more effective the noise averaging in the time direction.

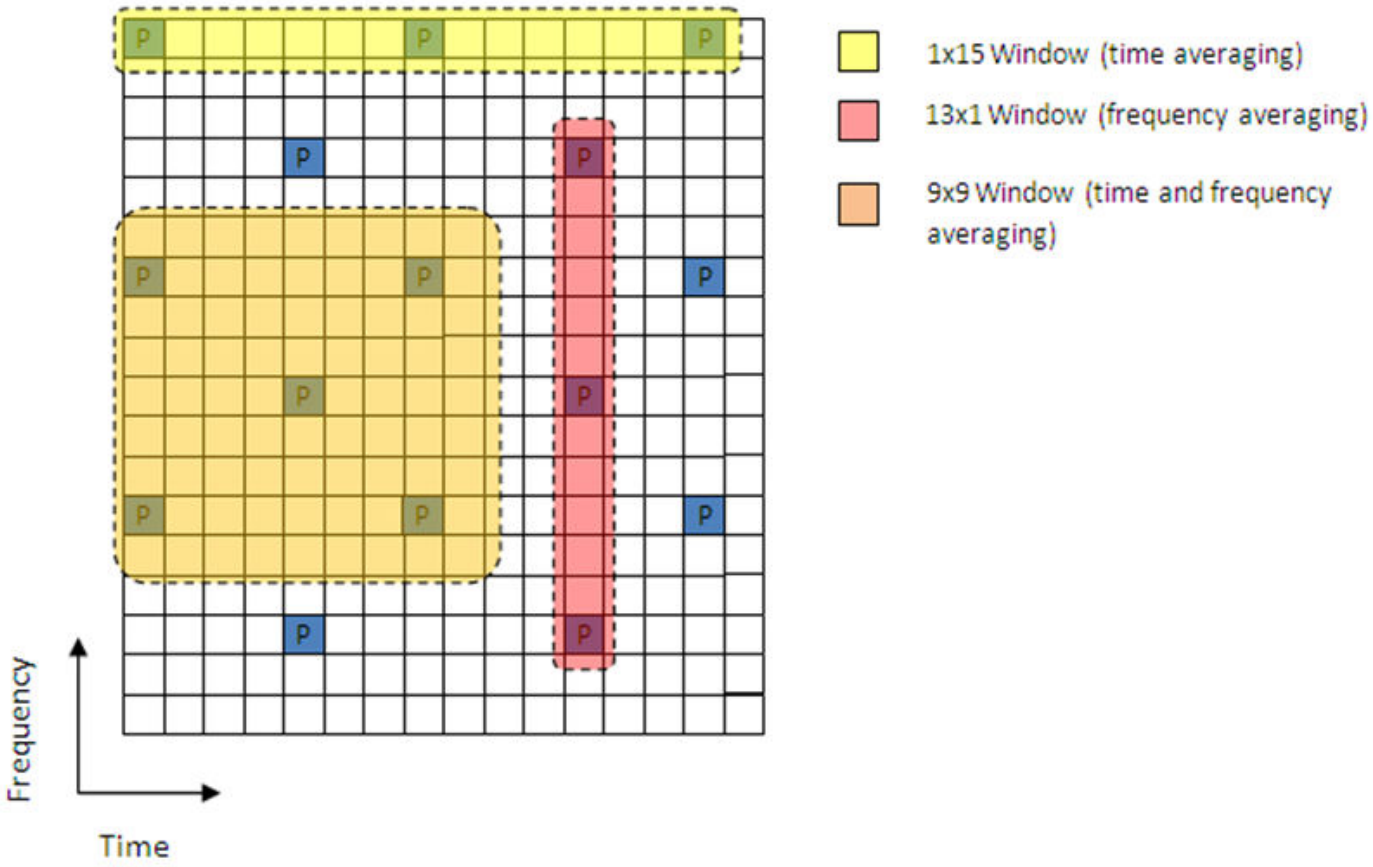
'UserDefined'

The second pilot symbol averaging method, `'UserDefined'`, allows the user to define the size of the averaging window, which direction averaging will be done in (time, frequency or both) and certain aspects of the interpolation that can be adjusted to suit the available data. For more information, see “Interpolation Subsystem” on page 1-123.

The averaging window size is defined in terms of resource elements. Any pilot symbols located within the window are used to average the value of the pilot symbol found at the centre of the window. The window size must be an odd number ensuring that there is a pilot at the center.

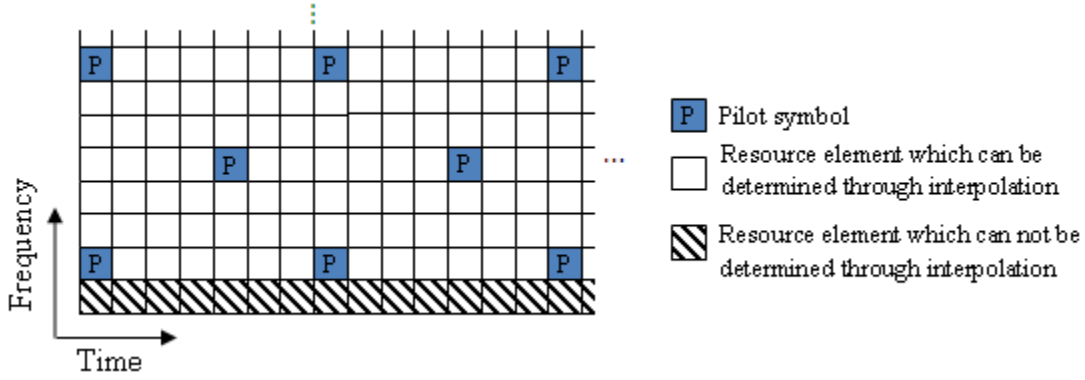
Note Averaging the channel estimates at pilot symbol locations is a simple yet powerful tool, but the window size must be carefully chosen. Using a large window size on a fast fading channel could result in averaging out not only noise, but also channel characteristics. Performing too much averaging on a

system with a small amount of noise can have an adverse effect on the quality of the channel estimates. Therefore, using a large averaging window for a fast changing channel could cause the estimate of the channel to appear flat, resulting in a poor estimate of the channel and affecting the quality of the equalization.



Create Virtual Pilots Subsystem

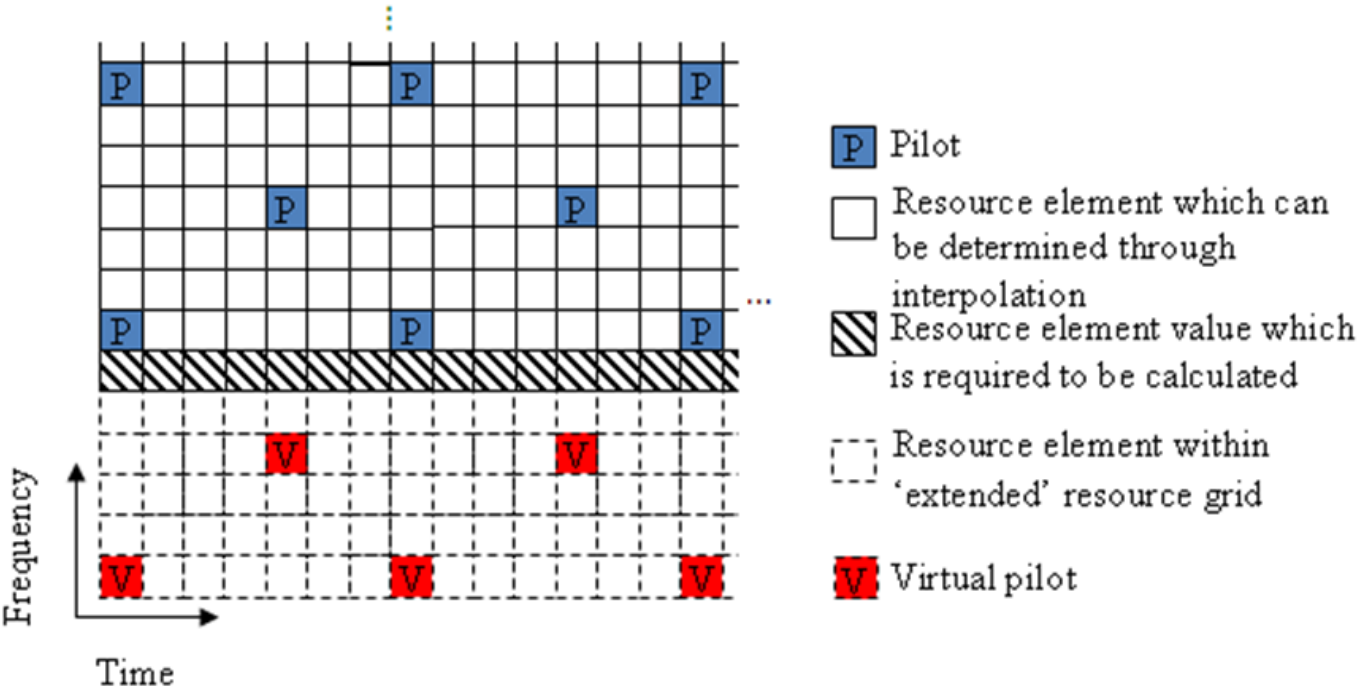
In many instances, edges of the resource grid do not contain any pilot symbols. This effect is shown in the resource grid in the following figure.



In this case, channel estimates at the edges cannot be interpolated from the pilot symbols. To overcome this problem, virtual pilot symbols are created. The function `lteDLChannelEstimate` creates virtual pilot symbols on all the edges of the received grid to allow cubic interpolation.

Virtual Pilot Placement

Virtual pilot symbols are created as shown in the following figure.



In this system, the resource grid is extended, with virtual pilot symbols created in locations which follow the original reference signal pattern. The presence of virtual pilot symbols allows the channel estimate at the resource elements, which previously could not be calculated by interpolation, to be calculated by interpolation using the original and virtual pilot symbols.

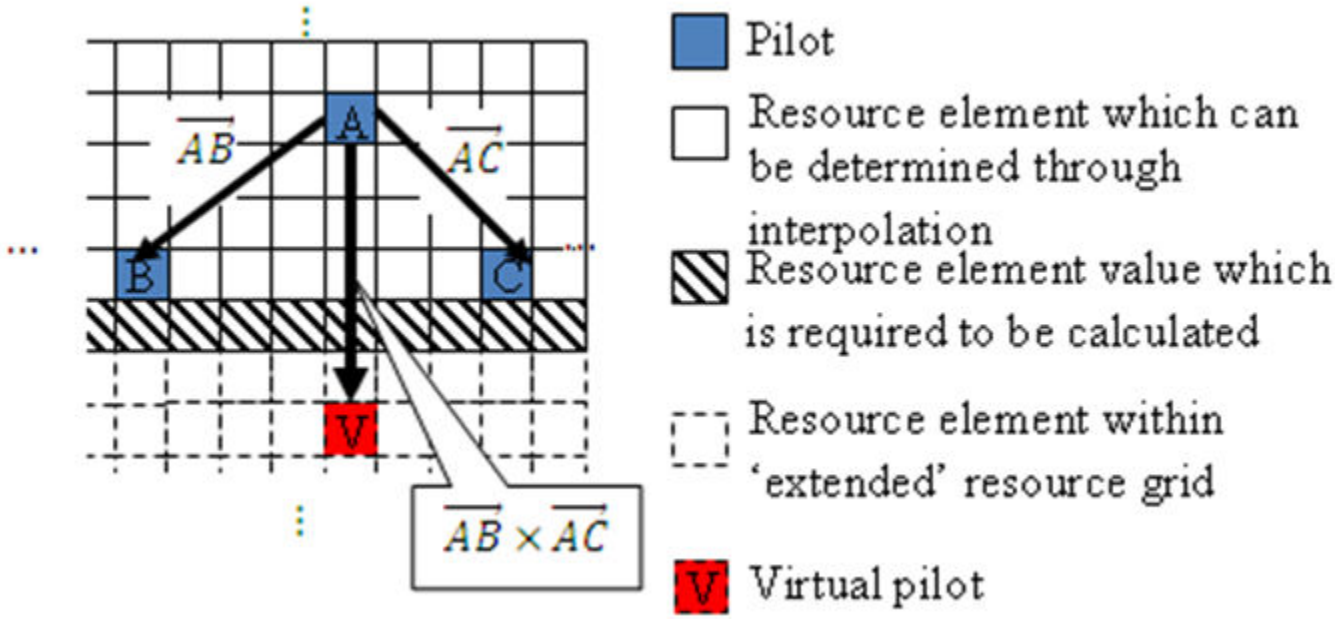
Calculating Virtual Pilot Symbol Values

The virtual pilot symbols are calculated using the original pilot symbols. For each virtual pilot symbol, the value is calculated following these steps:

1. The closest 10 ordinary pilots in terms of Euclidian distance in time and frequency are selected. The search is optimized to consider 10 these pilots, rather than checking all possible pilots. Based on the possible configurations of the cell RS, using 10 pilots provides sufficient time and frequency diversity in the pilots for the virtual pilot calculation.
2. Using this set of the 10 pilots, the closest three pilot symbols are selected. These three symbols must occupy at least two unique subcarriers and two unique OFDM symbols.
3. Using this set of three pilots, two vectors are created. One vector between the closest and furthest pilot symbols, and one vector between the second closest and furthest pilot symbols.
4. The cross-product of these two vectors is calculated to create a plane on which the three points reside.

5 The plane is extended to the position of the virtual pilot to calculate the value based on one of the actual pilot values.

This diagram shows the virtual pilot calculation.



Note Virtual pilots are only created for the MATLAB® `'linear'` and `'cubic'` interpolation methods.

Interpolation Subsystem

Once the noise has been reduced or removed from the least squares pilot symbol averages and sufficient virtual pilots have been determined, it is possible to use interpolation to estimate the missing values from the channel estimation grid. The `lteDLChannelEstimate` function has two pilot symbol averaging methods, `'TestEVM'` and `'UserDefined'`. The pilot symbol averaging methods also define the interpolation method performed to obtain the channel estimate.

The `'TestEVM'` pilot averaging method described in TS 36.141 [1], Annex F.3.4, requires the use of simple linear interpolation on the time-averaged and frequency-averaged column vector. The interpolation is one-dimensional, since it only estimates the values between the averaged pilot symbol subcarriers in the column vector. The resulting vector is then replicated and used as the channel estimate for the entire resource grid.

The `'UserDefined'` pilot-averaging method performs two-dimensional interpolation to estimate the channel response between the available pilot symbols. An interpolation window is used to specify which data is used to perform the interpolation. The `InterpWindow` field defines the causal nature of the available data. Valid settings for `cec.InterpWindow` are `'Causal'`, `'Non-causal'`, or `'Centered'`.

Use the `InterpWindow` setting:

- `'Causal'` when using past data.
- `'Non-causal'` when using future data. It is the opposite of `'Causal'`. Relying only on future data is commonly referred to as an anti-causal method of interpolation.
- `'Centered'` or `'Centred'` when using a combination of past, present, and future data.

The size of this interpolation window can also be adjusted to suit the available data. To specify this window size, set the `InterpWinSize` field.

Noise Estimation

The performance of some receivers can be improved through knowledge of the noise power present on the received signal. The function `lteDLChannelEstimate` provides an estimate of the noise power spectral density (PSD) using the estimated channel response at known reference signal locations. The noise power can be determined by analyzing the noisy least squares estimates and the noise averaged estimates.

The noisy least-squares estimates from the “Get Pilot Estimates Subsystem” on page 1-118 and the noise averaged pilot symbol estimates from the “Pilot Average Subsystem” on page 1-118 provide an indication of the channel noise. The least-squares estimates and the averaged estimates contain the same data, apart from additive noise. Simply taking the difference between the two estimates results in a noise level value for the least squares channel estimates at pilot symbol locations. Considering again,

$$\widetilde{H}_P(k) = \frac{Y_P(k)}{X_P(k)} = H_P(k) + noise$$

Averaging the instantaneous channel estimates over the smoothing window, we have

$$\widetilde{H}_P^{AVG}(k) = \frac{1}{|S|}\sum_{m \in S} \widetilde{H}_P(m) \approx H_P(k)$$

where S is the set of pilots in the smoothing window and $|S|$ is the number of pilots in S. Thus, an estimate of the noise at a particular pilot RE can be formed using:

$$\widetilde{noise} = \widetilde{H}_P(m) - \widetilde{H}_P^{AVG}(k)$$

In practice, it is not possible to remove all the noise using averaging. Because it is only possible to reduce the noise, only an estimate of the noise power can be made.

Note In the case of a noise free system or system with a high SNR, averaging could have a detrimental effect on the quality of the least squares estimates.

Using the value of the noise power found in the channel response at pilot symbol locations, the noise power per resource element (RE) can be calculated by taking the variance of the resulting noise vector. The noise power per RE for each transmit and receive antenna pair is calculated and stored. The mean of this matrix is returned as the estimate of the noise power per RE.

For a demonstration on how to set up a full transmit and receive chain for channel estimation, see “PDSCH Transmit Diversity Throughput Simulation” on page 2-32. In this example, multiple antennas are used and transmission is simulated through a propagation channel model.

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] Van de Beek, J.-J., O. Edfors, M. Sandell, S. K. Wilson, and P. O. Borjesson. “On Channel Estimation in OFDM Systems." Vehicular Technology Conference, IEEE 45th, Volume 2, IEEE, 1995.

See Also

`lteDLChannelEstimate` | `lteEqualizeMMSE` | `lteEqualizeZF` | `lteOFDMDemodulate`

Related Examples

- “LTE Downlink Channel Estimation and Equalization” on page 2-97

Transmission Modes and Transmission Schemes

The UE procedure for receiving the physical downlink shared channel (PDSCH) is outlined in TS 36.213 [1], Section 7.1. Transmission mode combinations the UE shall use to decode PDCCH and any corresponding PDSCH, as defined in TS 36.213 [1], Table 7.1-5, are indicated here:

Transmission modes (TM)	DCI format	Transmission schemes
TM1	1A	Single antenna port, port0
	1	
TM2	1A	Transmit diversity
	1	
TM3	1A	Transmit diversity
	2A	Open-loop codebook based precoding (large delay CDD) or transmit diversity
TM4	1A	Transmit diversity
	2	Closed-loop codebook based precoding
TM5	1A	Transmit diversity
	1D	Multiuser MIMO version of TM4
TM6	1A	Transmit diversity
	1B	Closed loop codebook based precoding for single layer
TM7	1A	Single antenna port, port 0 if the number of PBCH antenna ports is one, otherwise transmit diversity
	1	Non-codebook based precoding for single layer (single antenna port, port 5)
TM8	1A	Single antenna port, port 0 if the number of PBCH antenna ports is one, transmit diversity otherwise
	2B	Non-codebook based precoding for up to two layers (dual layer port 7 and 8 or single antenna port, port 7 or 8)

Transmission modes (TM)	DCI format	Transmission schemes
TM9	1A	• For non-MBSFN subframe: Single antenna port, port 0 if the number of PBCH antenna ports is one, transmit diversity otherwise • MBSFN subframe: single antenna port, port 7
	2C	Non-codebook based precoding for up to eight layers (up to eight layer transmission ports 7–14 or single antenna port, port 7 or 8)
TM10	1A	• For non-MBSFN subframe: Single antenna port, port 0 if the number of PBCH antenna ports is one, transmit diversity otherwise • MBSFN subframe: single antenna port, port 7
	2D	Extension of TM9 for CoMP (up to eight layer transmission ports 7–14 or single antenna port, port 7 or 8)

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.

Examples and Demos

Create an Empty Resource Grid

This example shows how to create an empty resource grid of the right dimensions for the cell-wide settings specified in structure `enb`.

Specify the cell-wide settings in a parameter structure.

```
enb.CyclicPrefix = 'Normal';
enb.NDLRB = 9;
enb.CellRefP = 2;
```

Create an empty resource grid by calling the `lteDLResourceGrid` function.

```
resourceGrid1 = lteDLResourceGrid(enb);
size(resourceGrid1)
```

```
ans = 1×3

   108    14     2
```

The resulting matrix `resourceGrid` is 3-dimensional.

Alternatively, you can use the MATLAB® `zeros` function. Specify the parameter structure. For normal cyclic prefix, specify seven symbols per slot.

```
enb.CyclicPrefix = 'Normal';
enb.NDLRB = 9;
enb.CellRefP = 2;
noSymbolsSlot = 7;
```

Create an empty resource grid. This time, call the MATLAB `zeros` function.

```
resourceGrid2 = zeros(enb.NDLRB*12, noSymbolsSlot*2, enb.CellRefP);
size(resourceGrid1)
```

```
ans = 1×3

   108    14     2
```

Compare the two resource grids to show they are equal in size and content.

```
isequal(resourceGrid1,resourceGrid2)
```

```
ans = logical
   1
```

See Also

`lteDLResourceGrid` | `zeros`

Map Reference Signal to Resource Grid

This example shows how to map the cell specific reference signals to the resource grid for a subframe in the two antenna case.

Specify the parameter structure. In this scenario, there are 6 resource blocks in the downlink.

```
enb.CyclicPrefix = 'Normal';
enb.NDLRB = 6;
enb.CellRefP = 2;
enb.DuplexMode = 'FDD';
```

Create an empty resource grid for a subframe.

```
resourceGrid = lteDLResourceGrid(enb);
```

Specify `NCellID` and `NSubframe` in the input parameter structure. These fields are required to generate the cell-specific reference signals. In this example, select subframe number 0.

```
enb.NCellID = 10;
enb.NSubframe = 0;
```

Generate the cell-specific reference signals for the two antenna ports by calling the `lteCellRS` function.

```
rsAnt0 = lteCellRS(enb,0);
rsAnt1 = lteCellRS(enb,1);
```

Generate mapping indices for the two antenna ports. You need these mapping indices to map the generated complex symbols to the resource grid.

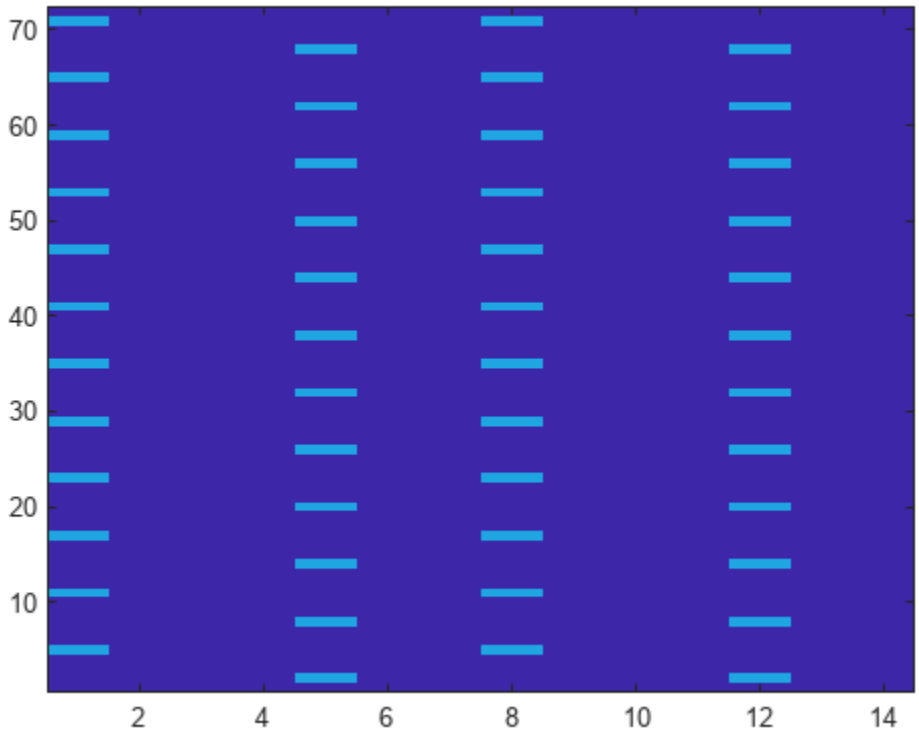
```
indAnt0 = lteCellRSIndices(enb,0);
indAnt1 = lteCellRSIndices(enb,1);
```

Map the reference signal complex symbols to the resource grid. Use the generated indices in linear form. Plot the resource grid.

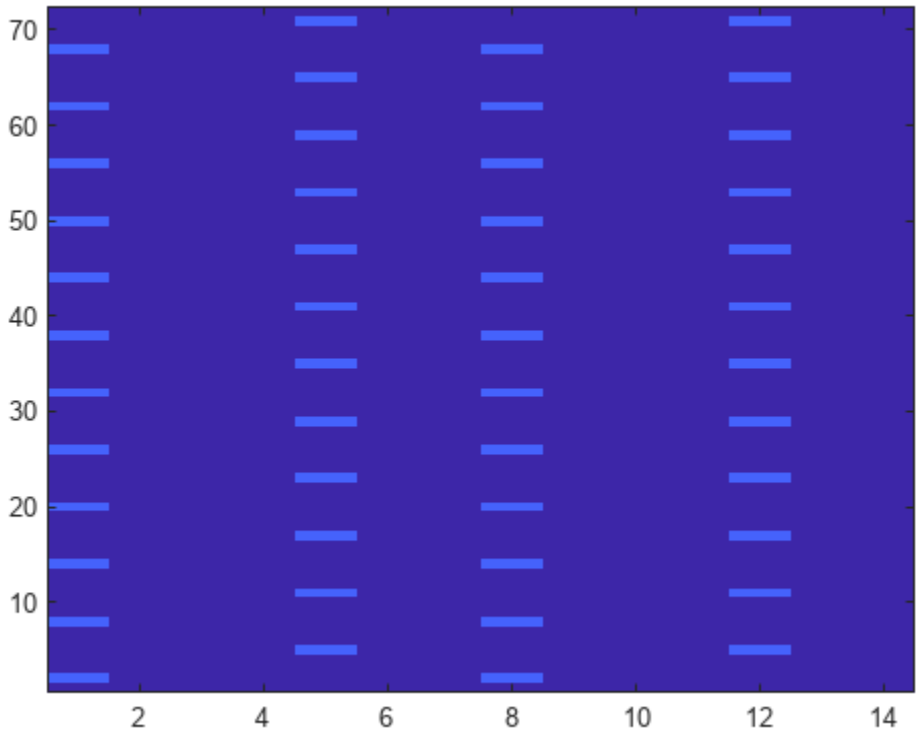
```
resourceGrid(indAnt0) = rsAnt0;
resourceGrid(indAnt1) = rsAnt1;
size(resourceGrid)
```

```
ans = 1×3

    72    14     2
```

```
image(100*abs(resourceGrid(:,:,1)))
axis xy
```




```
figure
image(50*abs(resourceGrid(:,:,2)))
axis xy
```



The resulting matrix has the complex symbols in `rsAnt0` and `rsAnt1` in the appropriate locations, specified by `indAnt0` and `indAnt1`.

See Also

`lteDLResourceGrid` | `lteCellRS` | `lteCellRSIndices`

Generate a Test Model

In this section...

The LTE specifications define conformance test models for transmitter tests. These tests include transmit signal quality, output power dynamics, EVM for various modulation schemes, BS output power, and RS absolute accuracy. These different test models waveforms can be generated using functions in the LTE Toolbox product.

E-UTRA Test Models

All E-UTRA test models (E-TMs), as defined in clause 6 of [1], use the following general parameters.

- Single antenna port, 1 code word, 1 layer without any precoding
- Duration is 10 subframes (10 ms)
- Normal cyclic prefix
- Virtual resource blocks of localized type
- UE-specific reference signals are not used

The following physical channels and signals are generated.

- Reference signals (CellRS)
- Primary Synchronization signal (PSS)
- Secondary Synchronization signal (SSS)
- Physical broadcast channel (PBCH)
- Physical control format indicator channel (PCFICH)
- Physical hybrid ARQ indicator channel (PHICH)
- Physical downlink control channel (PDCCH)
- Physical downlink shared channel (PDSCH)

Test models are selected according to the required test case. In this example, the test model under consideration, E-TM1.1, is used to test the following criteria.

- BS output power
- Unwanted emissions
  - Occupied bandwidth
  - ACLR
  - Operating band unwanted emissions
  - Transmitter spurious emissions
- Transmitter intermodulation
- Reference signals absolute accuracy

Generate Test Model Waveform

This example shows how to generate a test model waveform, a time-domain (post-OFDM modulation) signal for a single antenna port and 10 subframes.

Specify, as a string, the test model number. In this example, generate test model 1.1. Test model 1.1 and other test models are defined in TS 36.141 [1], Section 6.

```
tm = '1.1';
```

Valid test model values are `'1.1'`, `'1.2'`, `'2'`, `'3.1'`, `'3.2'`, and `'3.3'`.

Specify, as a string, the channel bandwidth. Select a non-standard bandwidth of 9 RB.

```
bw = '9RB';
```

Valid bandwidth values are `'1.4MHz'`, `'3MHz'`, `'5MHz'`, `'10MHz'`, `'15MHz'`, `'20MHz'`, `'9RB'`, `'11RB'`, `'27RB'`, `'45RB'`, `'64RB'`, and `'91RB'`.

Generate the test model waveform.

```
[timeDomainSig,grid] = lteTestModelTool(tm,bw);
```

The channel model number and the bandwidth determine the physical channel and signal parameters, as specified in TS 36.141 [1]. The generated waveform, `timeDomainSig`, is a time-domain signal that has undergone OFDM modulation, cyclic prefix insertion, and windowing. The second output, `grid`, is a 2-dimensional array representing the resource grid spanning 10 subframes.

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.

See Also

`lteTestModelTool`

Analyze Throughput for PDSCH Demodulation Performance Test

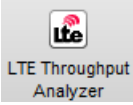
In this section...

LTE Throughput Analyzer Overview

You can use the LTE Throughput Analyzer app to execute a physical downlink shared channel (PDSCH) demodulation performance test.

Open LTE Throughput Analyzer App

To open the LTE Throughput Analyzer app, select the **APPS** tab on the MATLAB desktop and click the following icon.

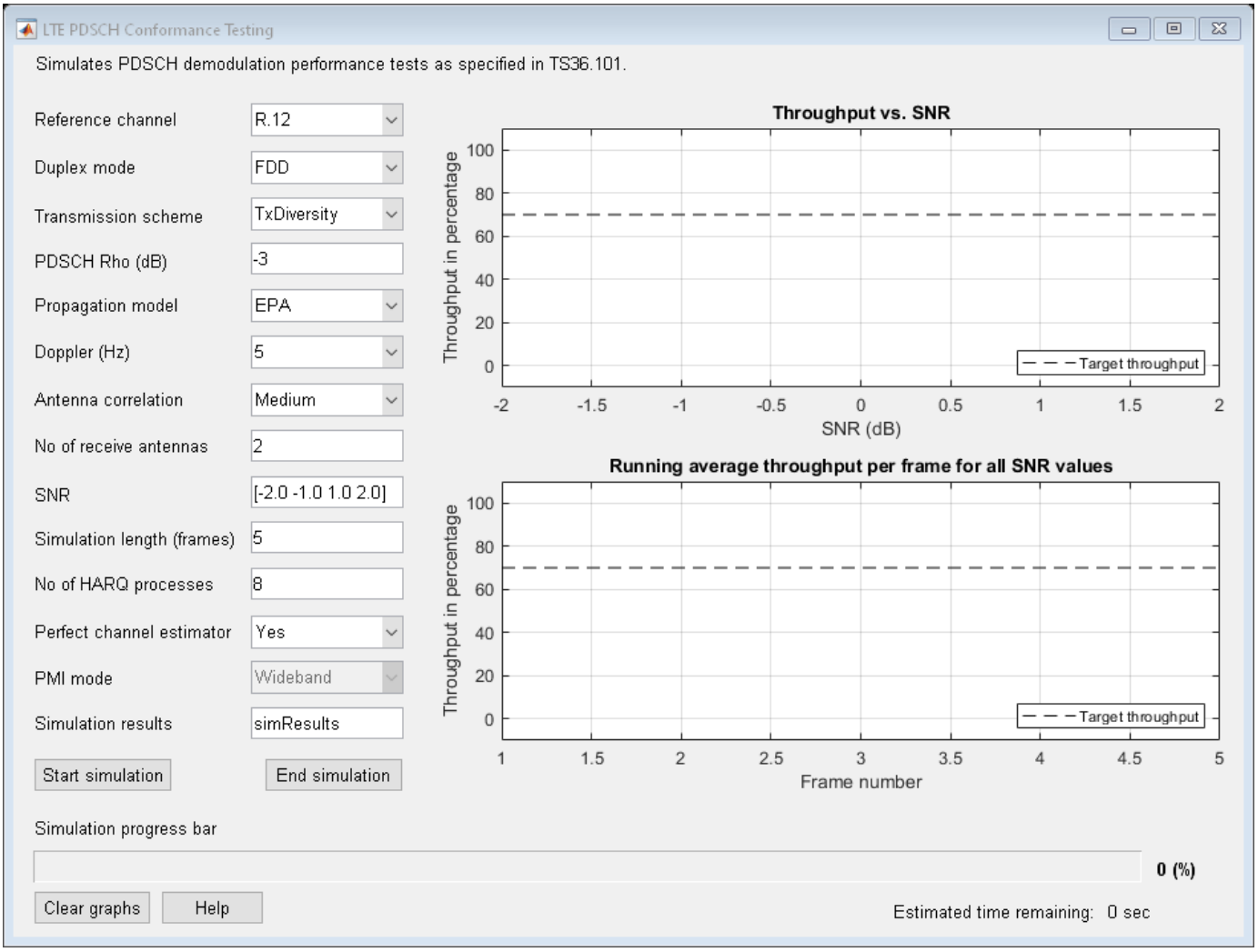


Alternatively, the LTE Throughput Analyzer app can be launched from the MATLAB command window.

Open LTE Throughput Analyzer App from Command Line

The LTE Throughput Analyzer dialog box appears when you execute the `lteDLConformanceTestTool` function with no input arguments.

```
lteDLConformanceTestTool
```



Dialog Box Inputs and Outputs

Parameters

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. • 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. • `'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. 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. 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: • `'FDD'` for Frequency Division Duplex • `'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” on page 1-106.</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>

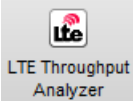
Parameter (Equivalent Field)	Values	Description
PMI mode (`PMIMode`)	`'Wideband'` (default), `'Subband'`	PMI reporting mode. `PMIMode='Wideband'` corresponds to PUSCH reporting Mode 1-2 or PUCCH reporting Mode 1-1 (PUCCH Report Type 2) and `PMIMode='Subband'` corresponds to PUSCH reporting Mode 3-1.
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.

Examples

Perform 4-by-2 Transmit Diversity Conformance Test

This example shows how 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. Select the **APPS** tab on the MATLAB desktop and click the following icon.



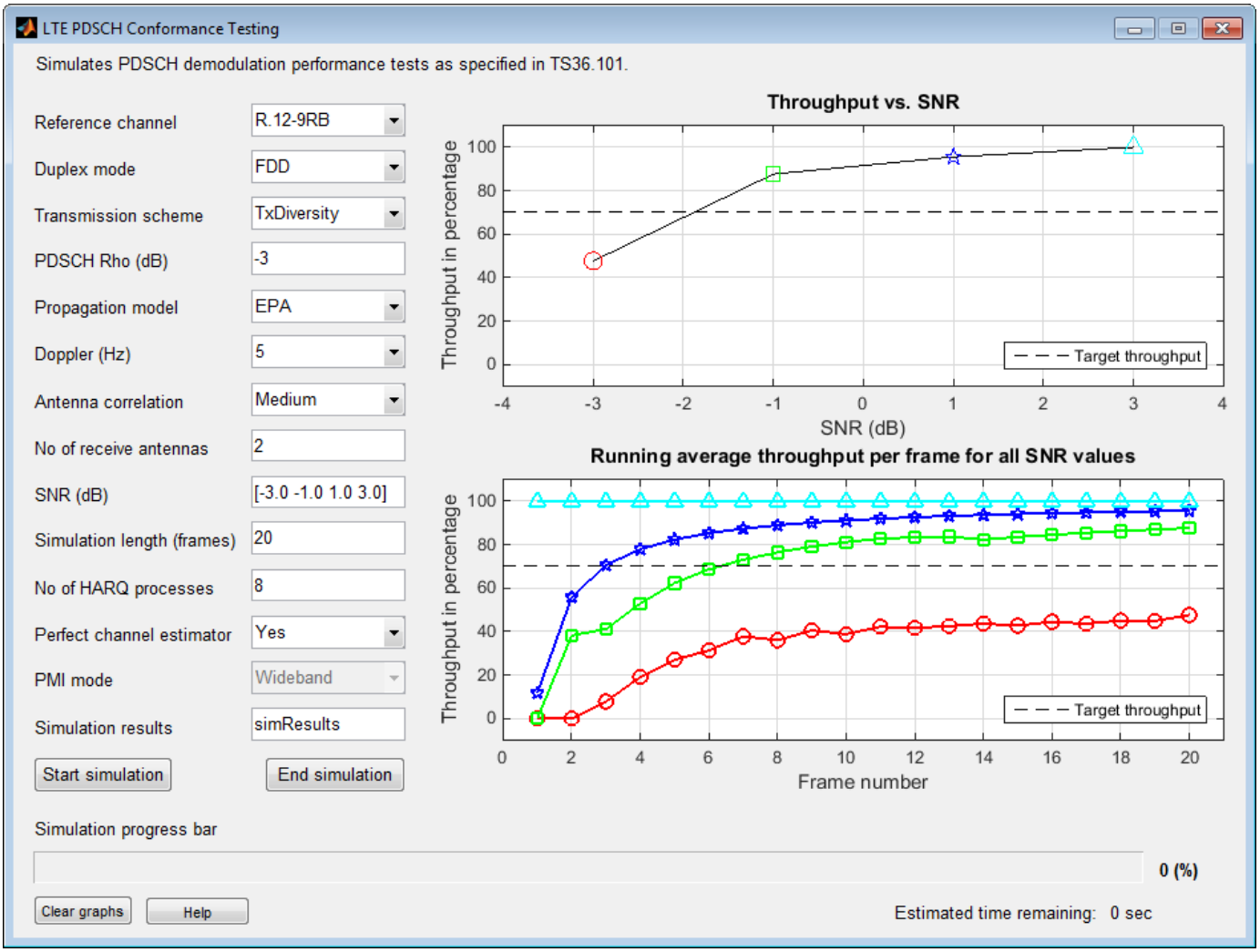
The LTE PDSCH Conformance Testing dialog box opens.

From the **Reference channel** drop-down list, choose `R.12-9RB`.

For **SNR**, enter `[-3.0 -1.0 1.0 3.0]`.

For **Simulation length (frames)**, enter `20`.

Click the **Start simulation** button. Wait a few minutes for the simulation to run. In the bottom-right corner of the window, next to **Estimated time remaining**, the tool displays an approximation of how long the simulation still needs to run. When the simulation finishes, the dialog box appears as shown in the following figure.



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. Enter `simResults` to see its contents.

```
simResults
```

```
simResults =
```

```
1x4 struct array with fields:

    throughput
    tpPerFrame
    rawBER
```

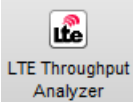
Perform Customized Conformance Test with User-Defined Configuration

This example shows how to run a conformance test for a user-defined configuration structure. You can carry out performance analysis and testing under user-defined settings. To do so, select **'User defined'** from the “Reference channel” popup menu, which will then prompt for the configuration structure variable name. The test bench will expect this variable to be present (already defined by the user) in the 'base' workspace.

Perform the single physical resource block (PRB) RMC R.0 conformance test, except with the allocated resource block moved to the upper band edge rather than lower band edge. First, create the basic configuration structure with the function `lteRMCDL`. Choose the most closely-matched RMC. Then, modify it with this the `PRBSet` requirement.

```
rmc = lteRMCDL('R.0');
rmc.PDSCH.PRBSet = rmc.NDLRB-1;
```

Open the LTE Throughput Analyzer app. Select the **APPS** tab on the MATLAB desktop and click the following icon.

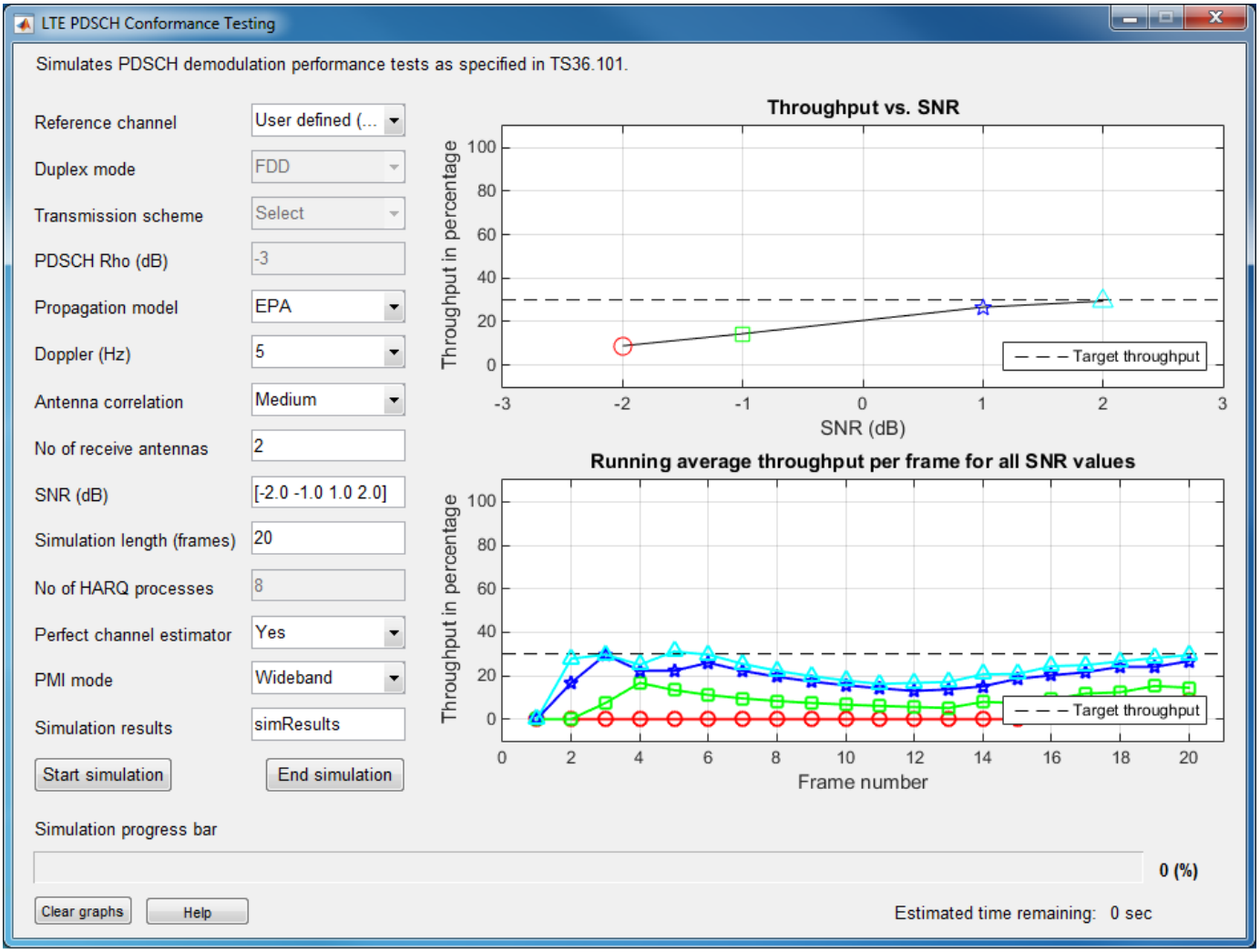


The LTE PDSCH Conformance Testing dialog box opens.

From the **Reference channel** drop-down list, choose `User defined`. The User Defined Configuration dialog box opens.

For **Configuration structure variable name**, enter `rmc`. Click **OK**.

Click the **Start simulation** button. Wait a few minutes for the simulation to run. In the bottom-right corner of the window, next to **Estimated time remaining**, the tool displays an approximation of how long the simulation still needs to run. When the simulation finishes, the dialog box appears as shown in the following figure.



The simulation result for a 20-frame run is displayed in the MATLAB Command Window.

```
Result for -2 dB SNR
Throughput: 7.22%

Result for -1 dB SNR
Throughput: 15.56%

Result for 1 dB SNR
Throughput: 28.33%

Result for 2 dB SNR
Throughput: 33.89%
```

In addition, the `simResults` variable now appears in the MATLAB workspace. Enter `simResults` to see its contents.

```
simResults
```

```
simResults =
```

```
1x4 struct array with fields:

    throughput
    tpPerFrame
    rawBER
```

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` | `lteTestModelTool`

LTE Parameterization for Waveform Generation and Simulation

This example shows how to parameterize end-to-end simulations and generate static waveforms by using LTE Toolbox™ software. This example focuses on downlink transmissions, but the concepts discussed also apply to uplink transmissions.

Introduction

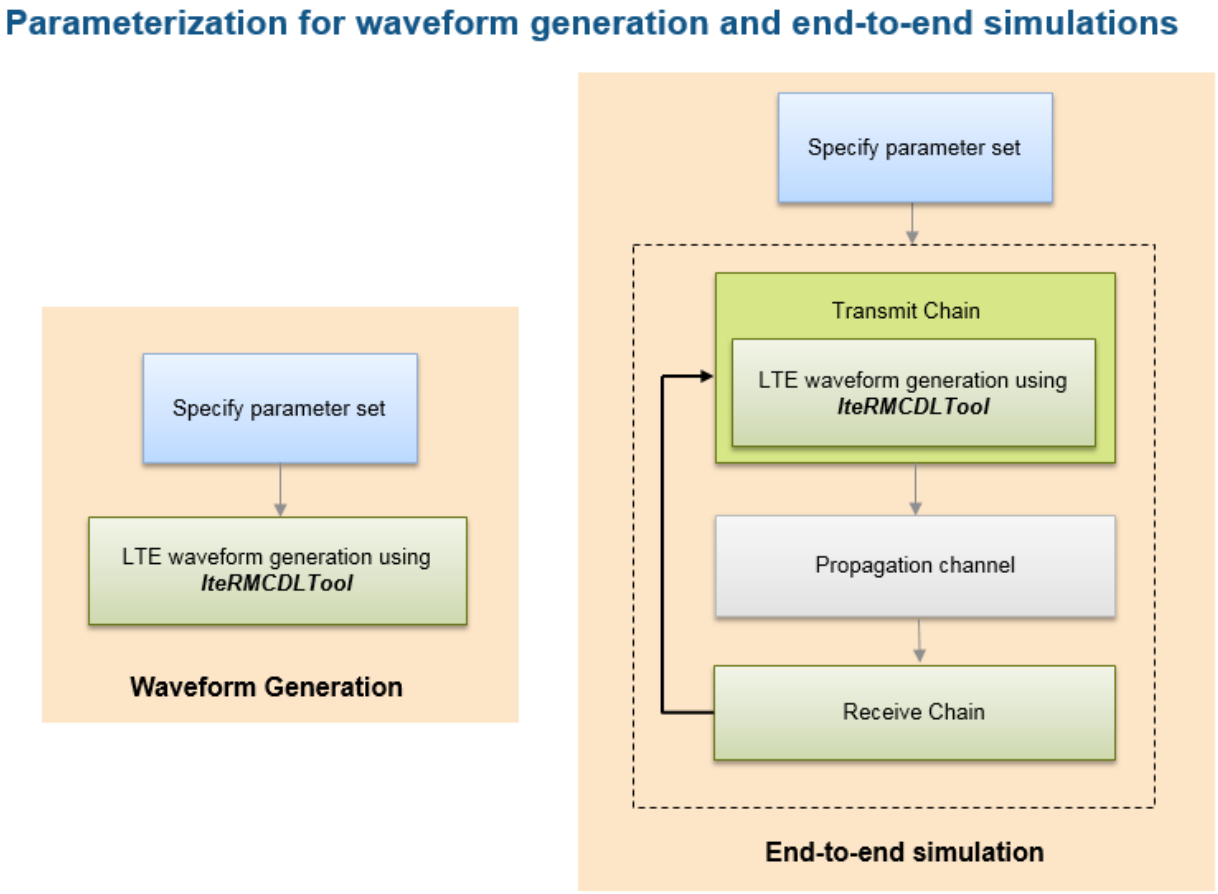
The LTE Toolbox can be used to generate standard compliant LTE/LTE-Advanced uplink, downlink and sidelink complex baseband waveforms which could be used for a number of end user applications including end-to-end simulations, static waveform generation, regression testing and performance analysis. The toolbox provides functions for flexible and easy generation of the full link, tailored to user requirements. Due to the multiple channels and signals in each link, the toolbox also provides a means to generate pre-defined parameter sets corresponding to standard defined measurement channels which can be used as such or can be further modified to parameterize waveform generation and end-to-end simulations. For the downlink, the toolbox includes these pre-defined parameter sets in the form of the Reference Measurement Channels (RMC) defined in TS 36.101 [1]. This example demonstrates how the `lteRMCDL` and `lteRMCDLTool` functions combine to support LTE downlink waveform generation for different user requirements. The corresponding uplink functions are `lteRMCUL` and `lteRMCULTool`.

LTE Downlink Parameterization and Waveform Generation Functions

In this example we discuss the two top-level functions provided by the toolbox: `lteRMCDL`, which creates a full parameter set, and `lteRMCDLTool`, which generates the downlink waveform. By combining these two functions, standards-compliant LTE waveforms can be generated easily.

The downlink waveform generator function requires a single hierarchical MATLAB structure which specifies the set of all parameters for the transport channels, physical channels and physical signals present in the output waveform. The generator function returns the time domain waveform, the populated resource grid and the parameter set used in the creation of the waveform.

The toolbox includes the `lteRMCDL` function, which can provide a fully populated parameter structure for the pre-configured Reference Measurement Channels (RMC) as well as custom configurations. This parameter structure can be directly used by the `lteRMCDLTool` function to generate the waveforms or it can be used as a template for creating waveforms with user specified values for any of the constituent channels or signals. For example, changing the transmission scheme/mode, modulation scheme, code rate or changing the power level of the physical channels. It is important to note that all the user provided values are defined prior to calling the `lteRMCDL` function. This is because the `lteRMCDL` function does not overwrite any parameter values already defined at the input (except for read-only parameters). The following diagram shows parameterization for typical simulation setups.



LTE Downlink Parameterization Options

The LTE Toolbox supports different ways of specifying the parameter set defining the constituent physical channels and signals. These are explained further in subsequent sections:

- *Create the parameter set from important cell-wide and PDSCH parameters*: The `lteRMCDL` function provides parameter expansion and transport block size processing from cell-wide and PDSCH parameters. All downlink and special (if TDD mode) subframes are assumed to be scheduled. This allows a subset of the parameters to be specified and the function then calculates compatible missing parameters to create the full set. This approach can be used in general to create configurations where subframe 5 is active.
- *Using one of the pre-defined parameter sets*: The `lteRMCDL` function supports a number of standard defined parameter sets in the form of RMCs. If there is a configuration that exactly matches the requirements or if we want to generate a waveform corresponding to an RMC, we can use that RMC number directly for RMC table lookup and parameter set creation. The RMCs supported and its top-level parameters are shown below:

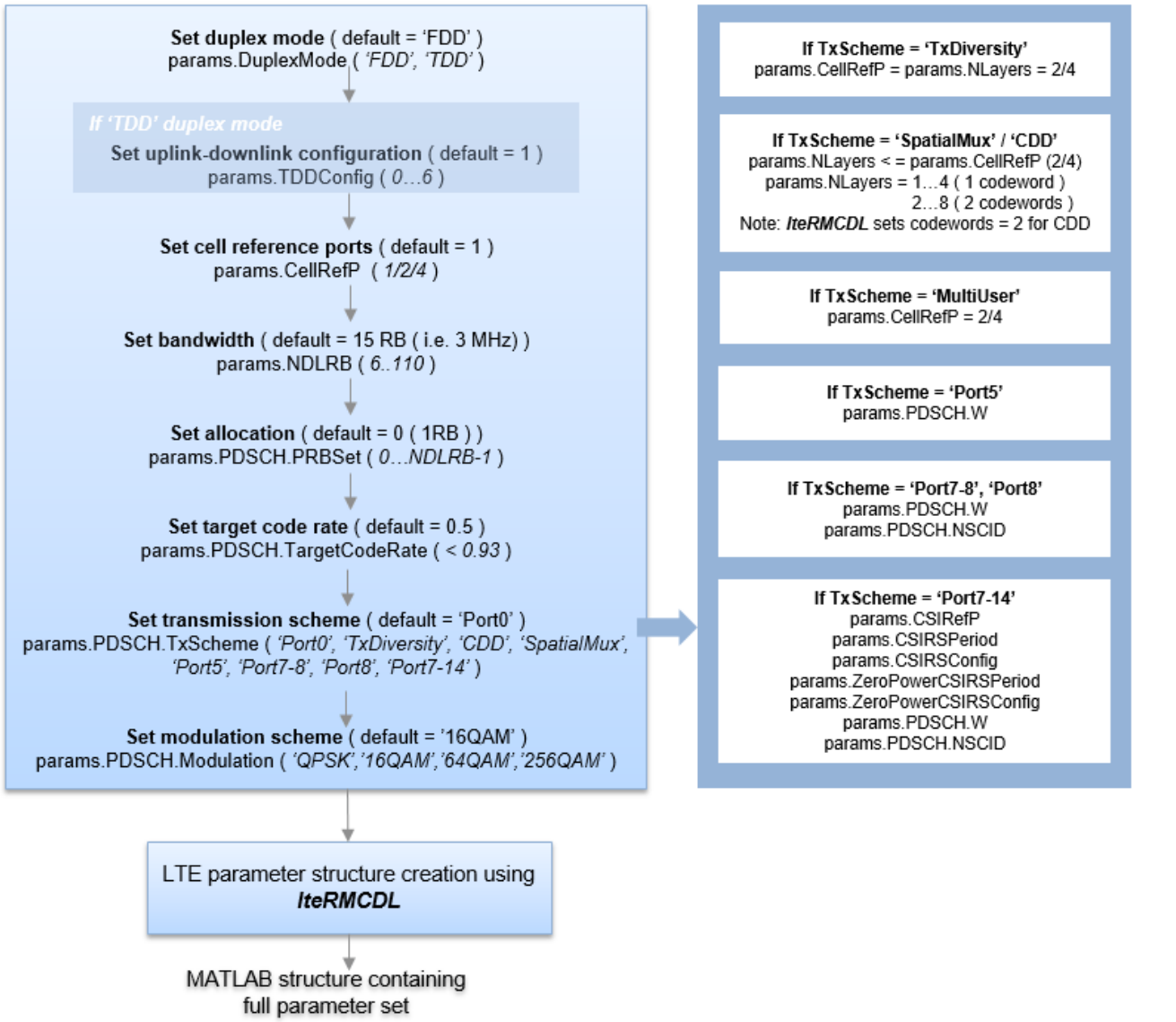
Reference channels	Reference channels (continued)
R.0 (Port0, 1 RB, 16QAM, CellRefP=1, R=1/2)	R.31-3A FDD (CDD, 50 RB, 64QAM, CellRefP=2, R=0.85-0.90)
R.1 (Port0, 1 RB, 16QAM, CellRefP=1, R=1/2)	R.31-3A TDD (CDD, 68 RB, 64QAM, CellRefP=2, R=0.87-0.90)
R.2 (Port0, 50 RB, QPSK, CellRefP=1, R=1/3)	R.31-4 (CDD, 100 RB, 64QAM, CellRefP=2, R=0.87-0.90)
R.3 (Port0, 50 RB, 16QAM, CellRefP=1, R=1/2)	R.43 FDD (Port7-14, 50 RB, QPSK, CellRefP=2, R=1/3)
R.4 (Port0, 6 RB, QPSK, CellRefP=1, R=1/3)	R.43 TDD (SpatialMux, 100 RB, 16QAM, CellRefP=4, R=1/2)
R.5 (Port0, 15 RB, 64QAM, CellRefP=1, R=3/4)	R.44 FDD (Port7-14, 50 RB, QPSK, CellRefP=2, R=1/3)
R.6 (Port0, 25 RB, 64QAM, CellRefP=1, R=3/4)	R.44 TDD (Port7-14, 50 RB, 64QAM, CellRefP=2, R=1/2)
R.7 (Port0, 50 RB, 64QAM, CellRefP=1, R=3/4)	R.45 (Port7-14, 50 RB, 16QAM, CellRefP=2, R=1/2)
R.8 (Port0, 75 RB, 64QAM, CellRefP=1, R=3/4)	R.45-1 (Port7-14, 39 RB, 16QAM, CellRefP=2, R=1/2)
R.9 (Port0, 100 RB, 64QAM, CellRefP=1, R=3/4)	R.48 (Port7-14, 50 RB, QPSK, CellRefP=2, R=1/2)
R.10 (TxDiversity|SpatialMux, 50 RB, QPSK, CellRefP=2, R=1/3)	R.50 FDD (Port7-14, 50 RB, 64QAM, CellRefP=2, R=1/2)
R.11 (TxDiversity|SpatialMux|CDD, 50 RB, 16QAM, CellRefP=2, R=1/2)	R.50 TDD (Port7-14, 50 RB, QPSK, CellRefP=2, R=1/3)
R.12 (TxDiversity, 6 RB, QPSK, CellRefP=4, R=1/3)	R.51 (Port7-14, 50 RB, 16QAM, CellRefP=2, R=1/2)
R.13 (SpatialMux, 50 RB, QPSK, CellRefP=4, R=1/3)	R.6-27RB (Port0, 27 RB, 64QAM, CellRefP=1, R=3/4)
R.14 (SpatialMux|CDD, 50 RB, 16QAM, CellRefP=4, R=1/2)	R.12-9RB (TxDiversity, 9 RB, QPSK, CellRefP=4, R=1/3)
R.25 (Port5, 50 RB, QPSK, CellRefP=1, R=1/3)	R.11-45RB (CDD, 45 RB, 16QAM, CellRefP=2, R=1/2)
R.26 (Port5, 50 RB, 16QAM, CellRefP=1, R=1/2)	
R.27 (Port5, 50 RB, 64QAM, CellRefP=1, R=3/4)	
R.28 (Port5, 1 RB, 16QAM, CellRefP=1, R=1/2)	

- *Customizing one of the pre-defined parameter sets*: There are many scenarios where we want a slightly different waveform configuration than given by the pre-defined set. In this case, we can start with one of the pre-defined RMCs and modify parameter(s) which require different values to create the full customized parameter set. This is illustrated by an example in the Parameterization Using Code Rate and Reference PDSCH in Subframe 5 section. Note that the subframes with user data would be as per the RMC. If the `TDD` duplex mode is used and `TDDConfig` is changed to a different value from the RMC, then the behavior of subframe 0, 5 and special subframes will remain unchanged and all other downlink subframes will inherit the properties (i.e. active/ nonactive, allocation, target code rate) of subframe 9.

Parameterization Using Important Cell-wide and PDSCH Parameters

The flowchart below explains how to set up a parameter set using some of the key cell-wide and PDSCH parameters. From a subset of these parameters, the `lteRMCDL` function can create the full parameter set through parameter expansion.

Parameterization using code rate/ modulation / allocation / Tx Scheme




```
%  The following example shows how to create a 20MHz, QPSK, 3/4 rate
%  waveform corresponding to transmission mode 8 ('Port7-8' transmission
%  scheme) with full allocation and 2 transmit antennas
dataStream = [1 0 0 1];     % Define the input user data stream
params = struct();          % Initialize the parameter structure
params.NDLRB = 100;         % 20 MHz bandwidth
params.CellRefP = 2;        % Cell reference signals on the first two ports
params.PDSCH.PRBSet = (0:params.NDLRB-1)'; % Full allocation
params.PDSCH.TargetCodeRate = 3/4; % The target code rate
params.PDSCH.TxScheme = 'Port7-8'; % Transmission mode 8
params.PDSCH.NLayers = 2;          % 2 layer transmission
params.PDSCH.Modulation = 'QPSK';  % Modulation scheme
params.PDSCH.NSCID = 0;            % Scrambling identity
params.PDSCH.NTxAnts = 2;          % 2 transmit antennas
```

```
params.PDSCH.W = lteCSICodebook(params.PDSCH.NLayers,...
                        params.PDSCH.NTxAnts,0).'; % Precoding matrix

% Now use lteRMCDL to populate other parameter fields.
fullParams = lteRMCDL(params);
% Generate the waveform using the full parameter set.
[dlWaveform, dlGrid, dlParams] = lteRMCDLTool(fullParams,dataStream);
% dlWaveform is the time domain waveform, dlGrid is the resource grid and
% dlParams is the full set of parameters used in the waveform generation.
```

Parameterization Using Pre-defined Parameter Set

If there is a predefined parameter set that exactly matches the requirements or if we want to generate a waveform corresponding to an RMC, use that RMC number to create the full parameter set.

To create a waveform corresponding to the R.0 RMC specified in TS 36.101, Annex A.3 [1]

```
params = lteRMCDL('R.0'); % Define the parameter set
[dlWaveform, dlGrid, dlParams] = lteRMCDLTool(params,dataStream);

% If the end application is waveform generation, we can also use the RMC
% number directly with the generator to create the waveforms
[dlWaveform, dlGrid, dlParams] = lteRMCDLTool('R.0',dataStream);
```

Parameterization Using Code Rate and Reference PDSCH in Subframe 5

Suppose we want to define a two codeword full-band 10MHz PDSCH using 2 layer open loop spatial multiplexing, 16QAM modulation and 1/2 rate with reference PDSCH transmission in subframe 5. Looking at TS 36.101 Table A.3.1.1-1: Overview of Downlink reference measurement channels, R.31-3A matches these criteria but with 64QAM modulation and variable code rate.

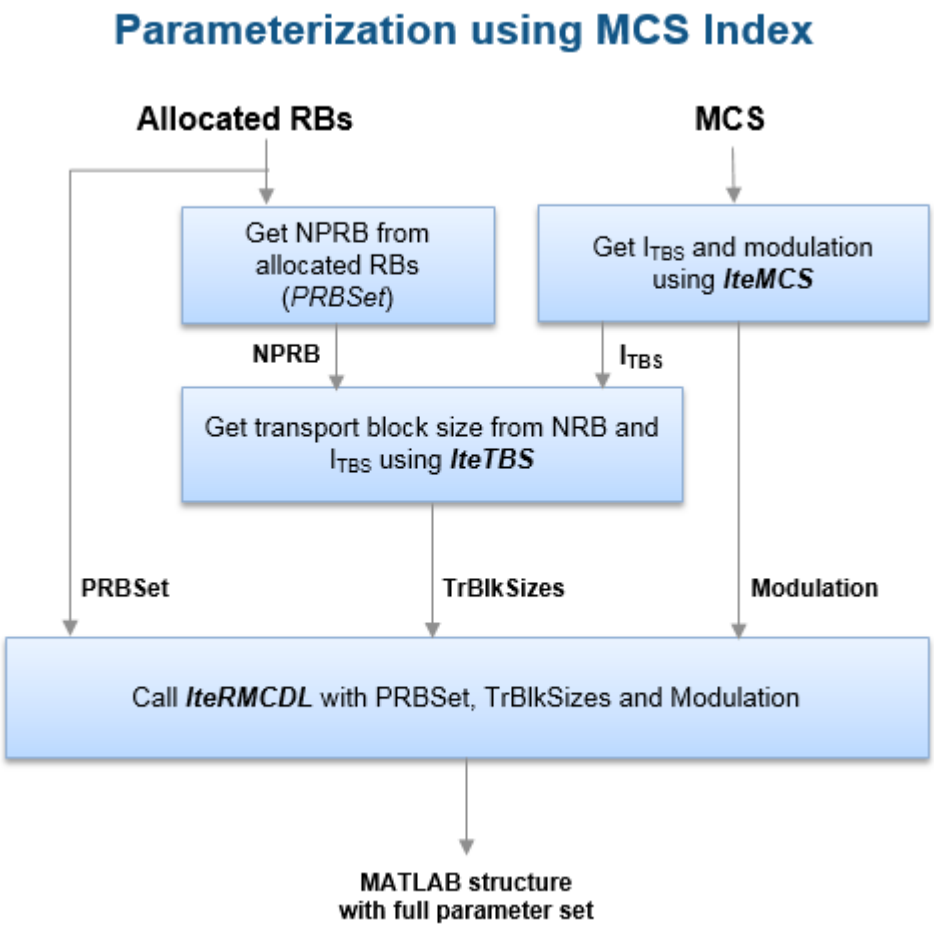
To create the required parameter set, we start off with the R.31-3A RMC to enable PDSCH transmission in subframe 5. We then override the modulation and code rate. The `lteRMCDL` function performs the transport block size calculation according to the code rate.

```
params = struct(); % Initialize the parameter structure
params.RC = 'R.31-3A';
params.PDSCH.TargetCodeRate = 1/2;
params.PDSCH.Modulation = '16QAM';
% Now use lteRMCDL to populate other parameter fields.
fullParams = lteRMCDL(params);
% Generate the waveform using the full parameter set.
[dlWaveform,dlGrid,dlParams] = lteRMCDLTool(fullParams,{dataStream, dataStream}); %#ok<*ASGLU>
```

Note that we used 'R.31-3A' as a starting point as our required parameter set closely matched this RMC (including reference PDSCH in subframe 5). We can also generate the parameter set by not specifying the RC above (or setting RC to be empty ([])). In this case the parameter set would correspond to reference PDSCH in all downlink and special (if TDD mode) subframes.

Parameterization Using MCS/transport Block Sizes

There are cases where we know the MCS or the transport block size and want to create a corresponding waveform. The following figure shows the steps involved in parameterization using MCS.



For example, to create a parameter set for MCS index 10, given the resource allocation is 50 RB:

```
mcsIndex = 10;
% Get the ITBS and modulation from MCS value
[itbs,modulation] = lteMCS(mcsIndex);
params = struct(); % Initialize the parameter structure
% Bandwidth (NDLRB) must be greater than or equal to allocations
params.NDLRB = 50;                          % Set the bandwidth
params.PDSCH.PRBSet = (0:params.NDLRB-1)'; % Full allocation
params.PDSCH.Modulation = modulation;      % Set the modulation scheme
nrb = size(params.PDSCH.PRBSet,1);         % Get number of RBs allocated
tbs = double(lteTBS(nrb,itbs));            % Get the transport block size
% Now create the 'TrBlkSizes' vector with no transmission in subframe 5
params.PDSCH.TrBlkSizes = [ones(1,5)*tbs 0 ones(1,4)*tbs];
% Now use lteRMCDL to populate other parameter fields.
fullParams = lteRMCDL(params);
% Now generate the waveform using the full parameter set.
[dlWaveform, dlGrid, dlParams] = lteRMCDLTool(fullParams,dataStream);
```

This approach can also be used to create a parameter set with a transport block size that is not standard defined, or when the required transport block size corresponds to a code rate greater than 0.93 (the standard restricts the code rate to be a maximum of 0.93). For these cases, we can specify

the transport block size as shown in the example above and other parameters will be updated accordingly by the `lteRMCDL` function. Note that RMCs don't typically define a reference PSDCH transmission in subframe 5 because of the possible presence of the SIB1 PDSCH. If a reference PDSCH is required then there are two ways to enable it:

1. The RMC is specified via the 'RC' field and is either 'R.31-3A' or 'R.31-4'.
2. The 'RC' field is not present or is specified as empty (e.g. params.RC = []).

Parameterization using variable code rate and resource allocation

The `lteRMCDL` and `lteRMCDLTool` functions can be used to generate waveforms where parameters vary over the subframes in a frame (e.g. CFI, PRBSet, TargetCodeRate). The CFI and target code rate can be specified as a vector and PRBSet as a cell array, when the values are changing per subframe.

In this example, we create a waveform corresponding to R.31-3 FDD RMC where the code rate and allocation varies per subframe. This is a two codeword RMC with code rate of 0.61 in subframe 0, 0.62 in subframe 5 and 0.59 in all other subframes. The number of allocated resource blocks are (4...99) in subframe 5 and full bandwidth (0...99) in all other subframes

```
params = struct();            % Initialize the parameter structure
params.NDLRB = 100;           % Set the bandwidth (20 MHz)
params.CellRefP = 2;          % Set the cell-specific reference signal ports
params.CFI = 1;               % 1 symbol allocated to PDCCH
params.PDSCH.PRBSet = {(0:99)' (0:99)' (0:99)' (0:99)' (0:99)'  ...
                       (4:99)' (0:99)' (0:99)' (0:99)' (0:99)'};
params.PDSCH.TargetCodeRate = [0.61 0.59 0.59 0.59 0.59 0.62 0.59 0.59 0.59 0.59];
params.PDSCH.TxScheme = 'CDD';% 2 codeword closed loop spatial mux
params.PDSCH.NLayers = 2;     % 1 layer per codeword
params.PDSCH.Modulation = {'64QAM', '64QAM'};% Set the modulation for 2 codewords

% Use lteRMCDL to populate other parameter fields. The resulting
% 'fullParams' can be manually checked against those given by the R.31-3
% FDD RMC in TS 36.101 Table A.3.9.1-1.
fullParams = lteRMCDL(params);
% Generate the waveform using the full parameter set.
[dlWaveform, dlGrid, dlParams] = lteRMCDLTool(fullParams,{dataStream, dataStream});
```

References

1. 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception"

LTE Waveform Modeling Using Downlink Transport and Physical Channels

This example shows how to generate a time domain waveform containing a Physical Downlink Shared Channel (PDSCH), corresponding Physical Downlink Control Channel (PDCCH) transmission and the Physical Control Format Indicator Channel (PCFICH), for one subframe.

Introduction

This example demonstrates how to generate a complete Downlink Shared Channel (DL-SCH) transmission for 6 resource blocks, 4 antenna transmit diversity using the functions from the LTE Toolbox™. The following physical channels are modeled:

- Physical Downlink Shared Channel (PDSCH)
- Physical Downlink Control Channel (PDCCH)
- Physical Downlink Control Format Indicator Channel (PCFICH)

This example generates a time domain (post OFDM modulation) for all 4 antenna ports. A single subframe (number 0) is considered in this example.

Note: The recommended way to generate RMC waveforms is using `lteRMCDLTool`, this example shows how a waveform can be built by creating and combining individual physical channels, as happens in an LTE system.

Cell-wide Settings

eNodeB settings are configured with a structure.

```
enb.NDLRB = 6;                % No of Downlink Resource Blocks(DL-RB)
enb.CyclicPrefix = 'Normal';  % CP length
enb.PHICHDuration = 'Normal'; % Normal PHICH duration
enb.DuplexMode = 'FDD';       % FDD duplex mode
enb.CFI = 3;                  % 4 PDCCH symbols
enb.Ng = 'Sixth';             % HICH groups
enb.CellRefP = 4;             % 4-antenna ports
enb.NCellID = 10;             % Cell id
enb.NSubframe = 0;            % Subframe number 0
```

Subframe Resource Grid Generation

A resource grid can easily be created using the `lteDLResourceGrid` function. This creates an empty resource grid for one subframe. The subframe is a 3 dimensional matrix. The number of rows represents the number of subcarriers available, this is equal to `12*enb.NDLRB` since there are 12 subcarriers per resource block. The number of columns equals the number of OFDM symbols in a subframe, i.e. 7*2, since we have 7 OFDM symbols per slot for normal cyclic prefix and there are 2 slots in a subframe. The number of planes (3rd dimension) in subframe is 4 corresponding to the 4 antenna ports as specified in `enb.CellRefP`.

```
subframe = lteDLResourceGrid(enb);
```

DL-SCH and PDSCH Settings

The DL-SCH and PDSCH are configured using a structure `pdsch`. The settings here configure 4 antenna transmit diversity with QPSK modulation.

```
pdsch.NLayers = 4;                          % No of layers
pdsch.TxScheme = 'TxDiversity';             % Transmission scheme
pdsch.Modulation = 'QPSK';                  % Modulation scheme
pdsch.RNTI = 1;                             % 16-bit UE-specific mask
pdsch.RV = 0;                               % Redundancy Version
```

PDSCH Mapping Indices Generation

The indices to map the PDSCH complex symbols to the subframe resource grid are generated using `ltePDSCHIndices`. The parameters required by this function include some of the cell-wide settings in `enb`, channel transmission configuration `pdsch` and the physical resource blocks (PRBs). The latter indicates the resource allocation for transmission of the PDSCH. In this example we have assumed all resource blocks are allocated to the PDSCH. This is specified using a column vector as shown below.

These indices are made '1based' for direct mapping on the resource grid as MATLAB® uses 1 based indexing. In this case we have assumed that both slots in the subframe share the same resource allocation. It is possible to have different allocations for each slot by specifying a two column matrix as allocation, where each column will refer to each slot in the subframe.

The resulting matrix `pdschIndices` has 4 columns, each column contains a set of indices in linear style pointing to the resource elements to be used for the PDSCH in each antenna port. Note that this function returns indices avoiding resource elements allocated to reference signals, the control region, broadcast channels and synchronization signals.

The generated indices are represented in 1-base format as used by MATLAB but can be made standard specific 0-based using the option `'0based'` instead of `'1based'`. If this option is not specified the default is 1-based index generation.

The coded block size for DL-SCH transmission can be calculated by the `ltePDSCHIndices` function. The `ltePDSCHIndices` function returns an information structure as its second output, which contains the parameter `G` which specifies the number of coded and rate-matched DL-SCH data bits to satisfy the physical PDSCH capacity. This value will be used subsequently to parameterize the DL-SCH channel coding.

```
pdsch.PRBSet = (0:enb.NDLRB-1).';   % Subframe resource allocation
[pdschIndices,pdschInfo] = ...
    ltePDSCHIndices(enb, pdsch, pdsch.PRBSet, {'1based'});
```

DL-SCH Channel Coding

We now generate the DL-SCH bits and apply channel coding. This includes CRC calculation, code block segmentation and CRC insertion, turbo coding, rate matching and code block concatenation. It can be performed using `lteDLSCH`.

The DL-SCH transport block size is chosen according to rules in TS36.101, Annex A.2.1.2 [1] "Determination of payload size" with target code rate $R = 1/3$ and number of bits per subframe given by `codedTrBlkSize`.

```
codedTrBlkSize = pdschInfo.G;   % Available PDSCH bits

transportBlkSize = 152;                % Transport block size
dlschTransportBlk = randi([0 1], transportBlkSize, 1);

% Perform Channel Coding
codedTrBlock = lteDLSCH(enb, pdsch, codedTrBlkSize, ...
               dlschTransportBlk);
```

PDSCH Complex Symbols Generation

The following operations are applied to the coded transport block to generate the Physical Downlink Shared Channel complex symbols: scrambling, modulation, layer mapping and precoding. This can be achieved using `ltePDSCH`. As well as some of the cell-wide settings specified in `enb` this function also requires other parameters related to the modulation and channel transmission configuration, `pdsch`. The resulting matrix `pdschSymbols` has 4 columns. Each column contains the complex symbols to map to each antenna port.

```
pdschSymbols = ltePDSCH(enb, pdsch, codedTrBlock);
```

PDSCH Mapping

The complex PDSCH symbols are then easily mapped to each of the resource grids for each antenna port using a simple assignment operation. The locations of the PDSCH symbols in the resource grids are given by `pdschIndices`.

```
% Map PDSCH symbols on resource grid
subframe(pdschIndices) = pdschSymbols;
```

DCI Message Configuration

Downlink Control Information (DCI), conveys information about the DL-SCH resource allocation, transport format, and information related to the DL-SCH hybrid ARQ. `lteDCI` can be used to generate a DCI message to be mapped to the Physical Downlink Control Channel (PDCCH). These parameters include the number of downlink Resource Blocks (RBs), the DCI format and the Resource Indication Value (RIV). The RIV of 26 correspond to full bandwidth assignment. The `lteDCI` function returns a structure and a vector containing the DCI message bits. Both contain the same information. The structure is more readable, while the serialized DCI message is a more suitable format to send to the channel coding stages.

```
dci.DCIFormat = 'Format1A';  % DCI message format
dci.Allocation.RIV = 26;     % Resource indication value

[dciMessage, dciMessageBits] = lteDCI(enb, dci); % DCI message
```

DCI Channel Coding

The DCI message bits are channel coded. This includes the following operations: CRC insertion, tail-biting convolutional coding and rate matching. The field `PDCCHFormat` indicates that one Control Channel Element (CCE) is used for the transmission of PDCCH, where a CCE is composed of 36 useful resource elements.

```
pdcch.NDLRB = enb.NDLRB;  % Number of DL-RB in total BW
pdcch.RNTI = pdsch.RNTI;  % 16-bit value number
pdcch.PDCCHFormat = 0;    % 1-CCE of aggregation level 1

% Performing DCI message bits coding to form coded DCI bits
codedDciBits = lteDCIEncode(pdcch, dciMessageBits);
```

PDCCH Bits Generation

The capacity of the control region depends on the bandwidth, the Control Format Indicator (CFI), the number of antenna ports and the PHICH groups. The total number of resources available for PDCCH can be calculated using `ltePDCCHInfo`. This returns a structure `pdcchInfo` where the different fields express the resources available to the PDCCH in different units: bits, CCEs, Resource Elements

(REs) and Resource Elements Groups (REGs). The total number of bits available in the PDCCH region can be found in the field `pdcchInfo.MTot`. This allows us to build a vector with the appropriate number of elements. Not all the available bits in the PDCCH region are necessarily used. Therefore the convention adopted is to set unused bits to -1, while bit locations with values 0 or 1 are used.

Note that we have initialized all elements in `pdcchBits` to -1, indicating that initially all the bits are unused. The elements of `codedDciBits` are mapped to the appropriate locations in `pdcchBits`.

Only a subset of all the bits in `pdcchBits` may be used, these are called the candidate bits. Indices to these can be calculated using `ltePDCCHSpace`. This returns a two column matrix. Each row indicates the available candidate locations for the provided cell-wide settings `enb` and PDCCH configuration structure `pdcch`. The first and second columns contain the indices of the first and last locations respectively of each group of candidates. In this case these indices are 1-based and refer to bits, hence they can be used to access locations in `pdcchBits`. The vector `pdcchBits` has 664 elements. The 72 bits of `codedDciBits` are mapped to the chosen candidate in `pdcchBits`. Therefore out of 664 elements, 72 will take 0 and 1 values, while the rest remain set to -1. `ltePDCCH` will interpret these locations as unused and will only consider those with 1s and 0s.

```
pdcchInfo = ltePDCCHInfo(enb);    % Get the total resources for PDCCH
pdcchBits = -1*ones(pdcchInfo.MTot, 1); % Initialized with -1

% Performing search space for UE-specific control channel candidates
candidates = ltePDCCHSpace(enb, pdcch, {'bits','1based'});

% Mapping PDCCH payload on available UE-specific candidate. In this example
% the first available candidate is used to map the coded DCI bits.
pdcchBits( candidates(1, 1) : candidates(1, 2) ) = codedDciBits;
```

PDCCH Complex Symbol Generation

From the set of bits used in `pdcchBits` (values not set to -1) PDCCH complex symbols are generated. The following operations are required: scrambling, QPSK modulation, layer mapping and precoding.

The `ltePDCCH` function takes a set of PDCCH bits and generates complex-valued PDCCH symbols performing the operations mentioned above. In this case pdcchSymbols is a 4 column matrix, each corresponding to each antenna port.

```
pdcchSymbols = ltePDCCH(enb, pdcchBits);
```

PDCCH Mapping Indices Generation and Resource Grid Mapping

PDCCH indices are generated for symbol mapping on resource grid. `pdcchIndices` is a matrix with 4 columns, one column per antenna port. The rows contain the indices in linear form for mapping the PDCCH symbols to the subframe resource grid.

```
pdcchIndices = ltePDCCHIndices(enb, {'1based'});

% The complex PDCCH symbols are easily mapped to each of the resource grids
% for each antenna port
subframe(pdcchIndices) = pdcchSymbols;
```

CFI Channel Coding

The number of OFDM symbols in a subframe is linked to the Control Format Indicator (CFI) value. Cell-wide settings structure `enb` specifies a CFI value of 3, which means that 4 OFDM symbols are used for the control region in the case of 6 downlink resource blocks. The CFI is channel coded using `lteCFI`. The resulting set of coded bits is a 32 element vector.

```
cfiBits = lteCFI(enb);
```

PCFICH Complex Symbol Generation

The CFI coded bits are then scrambled, QPSK modulated, mapped to layers and precoded to form the PCFICH complex symbols. The `pcfichSymbols` is a matrix having 4 columns where each column contains the PCFICH complex symbols that map to each of the antenna ports.

```
pcfichSymbols = ltePCFICH(enb, cfiBits);
```

PCFICH Indices Generation and Resource Grid Mapping

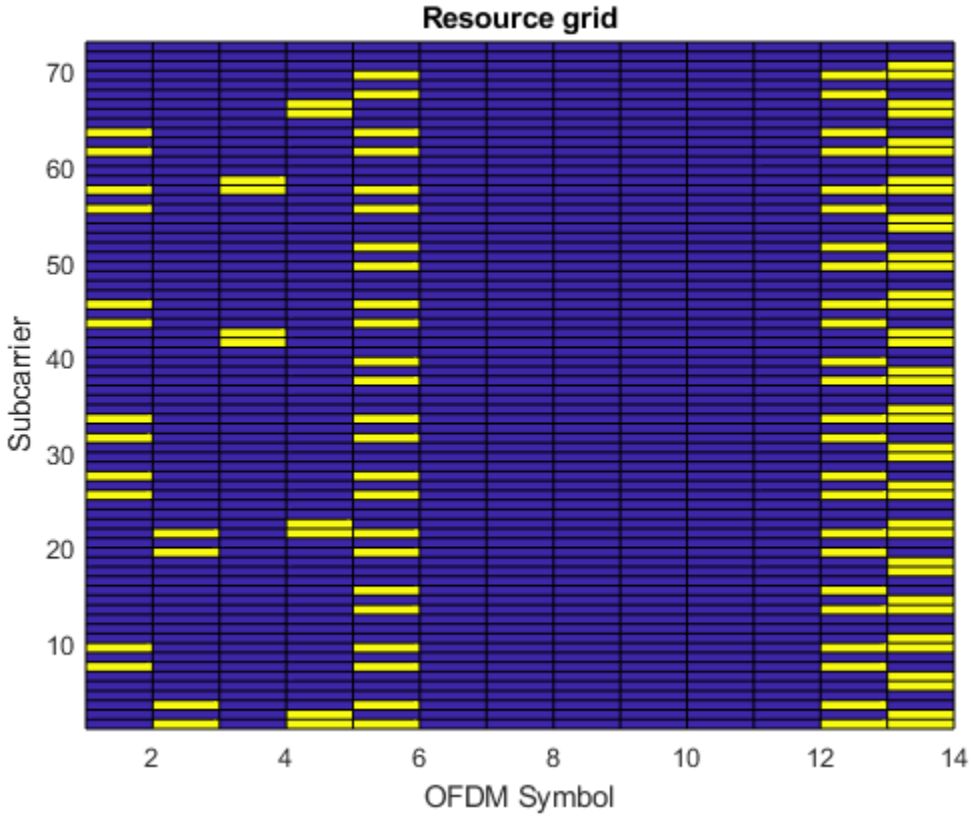
The PCFICH complex symbols are mapped to the subframe resource grid using the appropriate mapping indices. These are generated using `ltePCFICHIndices` and will be used to map the PCFICH symbol quadruplets to resource element groups in the first OFDM symbol in a subframe. All antenna ports are considered, and resource elements used by Reference Signals (RSs) are avoided. Note that the resulting matrix has 4 columns; each column contains the indices in linear form for each antenna port. These indices are 1-based, however they can also be generated using 0-based. The linear indexing style used makes the resource grid mapping process straight forward. The resulting matrix contains the complex symbols in `pcfichSymbols` in the locations specified by `pcfichIndices`.

```
pcfichIndices = ltePCFICHIndices(enb);

% Map PCFICH symbols to resource grid
subframe(pcfichIndices) = pcfichSymbols;
```

Plot Grid

Plot the resource grid for the first antenna. This includes (in yellow) the physical channels added in the example: PDSCH, PDCCH and PCFICH. Use `surf()` to avoid aliasing. This plots the values (REs) of the grid and joins them with lines (for 12 subcarriers, `surf()` plots 12 points and 11 lines; thus, the last subcarrier is not visible). Repeat the last row of the resource grid so that the last subcarrier is visible.

```
surf(abs([subframe(:,:,1);subframe(end,:,1)]));
view(2);
h = rotate3d; setAllowAxesRotate(h,gca,false);
axis tight;
xlabel('OFDM Symbol');
ylabel('Subcarrier');
title('Resource grid');
```



OFDM Modulation

Time domain mapping by performing OFDM modulation for downlink symbols. The resulting matrix has 4 columns; each column contains the samples for each antenna port.

```
[timeDomainMapped, timeDomainInfo] = lteOFDMModulate(enb, subframe);
```

Selected Bibliography

1 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception"

PDSCH Transmit Diversity Throughput Simulation

This example demonstrates how to measure the Physical Downlink Shared Channel (PDSCH) throughput of a transmit/receive chain using the LTE Toolbox™.

Introduction

The example uses the LTE Toolbox functions to generate a multi antenna downlink Reference Measurement Channel (RMC) R.12. Transmission is simulated using the Extended Pedestrian A (EPA) propagation channel model. Channel noise is added to the received waveform which is then OFDM demodulated, resulting in a received resource grid for each receive antenna. Channel estimation is performed to determine the channel between each transmit/receive antenna pair. The PDSCH data is then extracted and decoded from the received resource grid. Using the result of the block CRC the throughput performance of the transmit/receive chain is determined.

eNodeB Setup

```
% eNodeB Configuration
enb = struct;                       % eNodeB config structure
enb.RC = 'R.12';                    % RMC number
enb.NCellID = 10;                   % Cell ID
% Total number of subframes lteRMCDLTool  will generate per call. This
% example generates 1 frame (10 subframes) worth of data each call within
% the processing loop. lteRMCDLTool will be called 10 (NFrames) times.
enb.TotSubframes = 10;
enb.PDSCH.TxScheme = 'TxDiversity'; % Transmission scheme
enb.PDSCH.RNTI = 1;                 % 16-bit UE specific mask
enb.PDSCH.Rho = -3;                 % Downlink power allocation
enb.PDSCH.CSI = 'On';               % CSI scaling of soft bits
enb.PDSCH.RVSeq = 0;                % Disable HARQ

NFrames = 10;                       % Number of frames to simulate
SNRIn = [1 7];                      % SNR points to simulate
```

RMC Configuration

Generate the configuration for the specified RMC and obtain associated information: number of bits in a transport block for each subframe; number of OFDM symbols per subframe and number of transmit antennas.

```
% Generate RMC configuration for test model specified in enb structure
rmc = lteRMCDL(enb);

% Transport block sizes of each subframe within a frame
trblksize = rmc.PDSCH.TrBlkSizes;
ncw = size(trblksize,1);             % no of codewords associated to RMC

% dims is a three element vector [K; L; P]: where K = no of subcarriers,
% L = no of OFDM symbols and P = no of transmit antennas
dims = lteDLResourceGridSize(rmc);  % Determine resource grid dimensions
L = dims(2);                        % Number of OFDM symbols per subframe
P = dims(3);                        % Number of transmit antennas
```

Propagation Channel Model Configuration

```
cfg = struct;                    % Channel config structure
cfg.Seed = 13;                    % Channel seed
```

```
cfg.NRxAnts = 2;                % 2 receive antennas
cfg.DelayProfile ='EPA';        % Delay profile
cfg.DopplerFreq = 5;            % Doppler frequency in Hz
cfg.MIMOCorrelation = 'Medium'; % Multi-antenna correlation
cfg.NTerms = 16;                % Oscillators used in fading model
cfg.ModelType = 'GMEDS';        % Rayleigh fading model type
cfg.InitPhase = 'Random';       % Random initial phases
cfg.NormalizePathGains = 'On';  % Normalize delay profile power
cfg.NormalizeTxAnts = 'On';     % Normalize for transmit antennas
```

Channel Estimation Configuration

To reduce the impact of noise on pilot estimates, an averaging window of 15-by-13 Resource Elements (REs) is used. An EPA delay profile causes the channel response to change slowly over frequency. Therefore a large frequency averaging window of 15 REs is used. The Doppler frequency is 5 Hz, causing the channel to fade very slowly over time. Therefore a time averaging window of 13 REs is used.

```
cec = struct;                       % Channel estimation config structure
cec.PilotAverage = 'UserDefined';   % Type of pilot symbol averaging
cec.FreqWindow = 15;                % Frequency window size in REs
cec.TimeWindow = 13;                % Time window size in REs
```

Interpolation is performed by the channel estimator between pilot estimates to create a channel estimate for all REs. To improve the estimate multiple subframes can be used when interpolating. An interpolation window of 3 subframes with a centered interpolation window uses pilot estimates from 3 consecutive subframes to estimate the center subframe.

```
cec.InterpType = 'Cubic';         % Cubic interpolation
cec.InterpWinSize = 3;            % Interpolate up to 3 subframes
                                  % simultaneously
cec.InterpWindow = 'Centred';     % Interpolation windowing method
```

Processing Loop

For each SNR point the following operations are performed for each frame:

- *RMC Generation and OFDM Modulation*: Generate the OFDM modulated waveform for RMC using random data.
- *Propagation Channel Model*: The OFDM modulated waveform is transmitted through the propagation channel. The channel model is initialized appropriately to guarantee continuity of the fading waveforms between frames. The output of the channel `rxWaveform` has two columns, one per receive antenna.
- *Add Channel Noise*: The channel noise is modeled by AWGN.
- *Receiver Synchronization and OFDM Demodulation*: Determine the delay suffered during propagation. This is calculated by calling `lteDLFrameOffset`, which uses the primary and secondary synchronization signals. If it has not been possible to detect the primary and secondary synchronization signals due to extreme channel conditions or noise, then the previously calculated frame offset value is used. OFDM demodulation is performed after synchronization.
- *Channel Estimation*: Provides an estimate of the channel response at each element of the resource grid for a transmit/receive antenna pair. This estimate is then used to remove the effect of the channel on the transmitted signal. The channel estimation also aims to reduce the channel noise experienced during transmission by averaging the reference signals (pilot symbols).

- *Throughput Measurement*: PDSCH data is analyzed on a subframe by subframe basis. As specified by RMC R.12 no PDSCH data is transmitted on subframe 5. Therefore it is not decoded and does not count towards the throughput calculation. A specific subframe worth of data for all receive antennas is extracted from the array of received grids, the same defined subframe worth of information is extracted from the estimate of channel response for all transmit and receive antenna pairs. These, along with the noise estimate, are input into the function `ltePDSCHDecode`, which deprecodes the PDSCH data using an orthogonal space frequency block code (OSFBC) decoder. This returns a cell array of soft bit vectors (codewords) which are input to the `lteDLSCHDecode` function; this decodes the codeword and returns the block CRC error which is used to determine the throughput of the system.

```
% Set the random number generator to default value
rng('default');

% Initialize offset vector
offsets = 0;

% Array to store the maximum throughput for all SNR points
maxThroughput = zeros(length(SNRIn),1);
% Array to store the simulation throughput for all SNR points
simThroughput = zeros(length(SNRIn),1);

for snrIdx = 1:numel(SNRIn)
    SNRdB = SNRIn(snrIdx);

    fprintf('\nSimulating at %gdB SNR for a total %d Frame(s)\n',...
    SNRdB,NFrames);

    for FrameNo = 1:NFrames
        % Generate random bits for the frame. The number of bits to
        % generate for each subframe is specified by each element of
        % trblksize. For a full frame generate sum(trblksize) bits
        trdata = randi([0 1], sum(trblksize), 1);

        % Generate populated LTE resource grid using RMC generator and OFDM
        % modulate. info is a structure containing the sampling rate used
        % for OFDM modulation
        [txWaveform,txGrid,info] = lteRMCDLTool(rmc,trdata);

        % Set sampling rate of channel to that of OFDM modulation
        cfg.SamplingRate = info.SamplingRate;

        % Set channel offset to current frame (1 frame = 10ms)
        cfg.InitTime = (FrameNo-1)*(rmc.TotSubframes)/1000;

        % Pass data through the fading channel model.
        % An additional 25 samples are added to the end of the waveform.
        % These are to cover the range of delays expected from the channel
        % modeling (a combination of implementation delay and channel
        % delay spread).
        rxWaveform = lteFadingChannel(cfg,[txWaveform ; zeros(25,P)]);

        % Noise setup
        SNR = 10^(SNRdB/20);    % Linear SNR

        % Normalize noise power to take account of sampling rate, which is
        % a function of the IFFT size used in OFDM modulation, and the
```

```
% number of antennas
N0 = 1/(sqrt(2.0*rmc.CellRefP*double(info.Nfft))*SNR);

% Create additive white Gaussian noise
noise = N0*complex(randn(size(rxWaveform)), ...
                   randn(size(rxWaveform)));

% Add AWGN to the received time domain waveform
rxWaveform = rxWaveform + noise;

% Perform receiver synchronization
offset = lteDLFrameOffset(rmc,rxWaveform);

% Determine if frame synchronization was successful
if (offset > 25)
    offset = offsets(end);
else
    offsets = [offsets offset]; %#ok
end
if (offset>0)
    rxWaveform = rxWaveform(1+offset:end,:);
end

% Perform OFDM demodulation on the received data to recreate the
% resource grid
rxGrid = lteOFDMDemodulate(rmc,rxWaveform);

% Perform channel estimation
[estChannelGrid, noiseest] = lteDLChannelEstimate(rmc,cec,rxGrid);

% Process subframes 0 to 9 within received frame
for sf = 0:rmc.TotSubframes-1

    % Increment NSubframe for correct decoding of data
    rmc.NSubframe = mod(sf,10);

    % Extract one subframe for analyzing at a time from the
    % received grid
    rxSubframe = rxGrid(:,L*sf+1:L*(sf+1),:);

    % Extract the estimated channel response for the subframe
    % being analyzed
    chSubframe = estChannelGrid(:,L*sf+1:L*(sf+1),:,:);

    % Perform deprecoding, layer demapping, demodulation and
    % descrambling on the received data using the estimate of
    % the channel
    rxEncodedBits = ltePDSCHDecode(rmc,rmc.PDSCH,...
          rxSubframe*(10^(-rmc.PDSCH.Rho/20)),chSubframe,noiseest);

    % Transport block sizes
    outLen = ones(1,ncw)*trblksize(rmc.NSubframe+1);

    % Decode DownLink Shared Channel (DL-SCH)
    [decbits,blkcrc] = lteDLSCHDecode(rmc,rmc.PDSCH,outLen,...
                       rxEncodedBits);

    % Skip empty transport blocks, this will be the case of
```

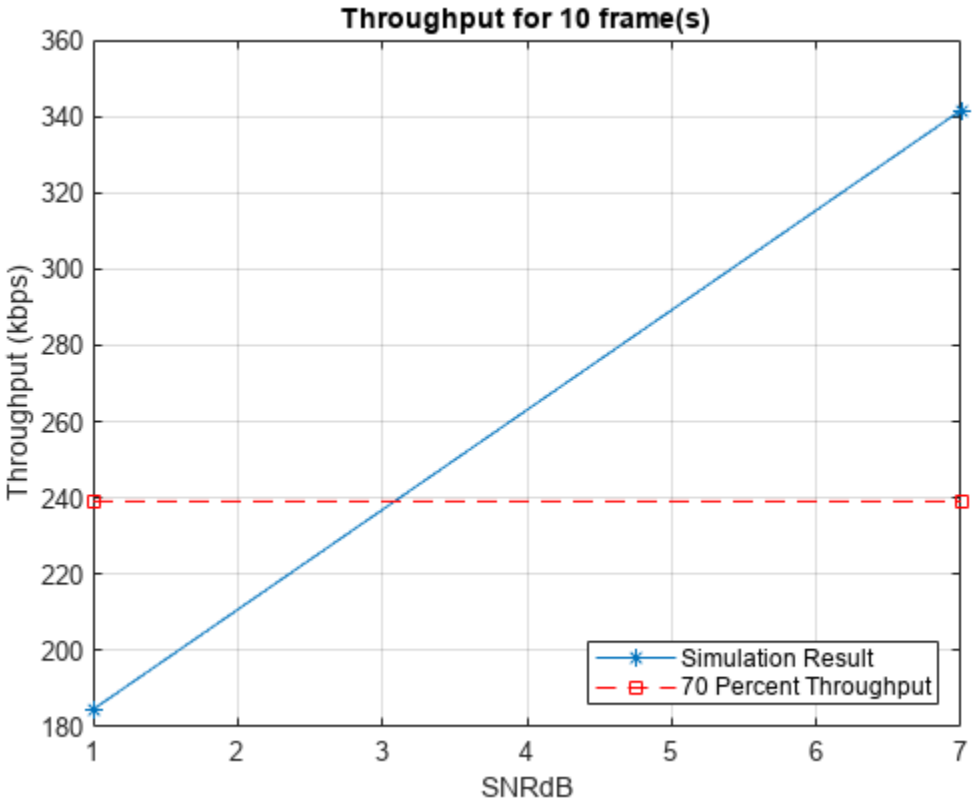
```
            % subframe 5, where no data is transmitted in RMC R.12
            if(outLen ~= 0)
                simThroughput(snrIdx) = simThroughput(snrIdx) + sum(~blkcrc .* outLen);
                maxThroughput(snrIdx) = maxThroughput(snrIdx) + sum(outLen);
            end
        end

        rmc.NSubframe = 0;
    end

    offsets = 0;
end
```

```
Simulating at 1dB SNR for a total 10 Frame(s)

Simulating at 7dB SNR for a total 10 Frame(s)
```

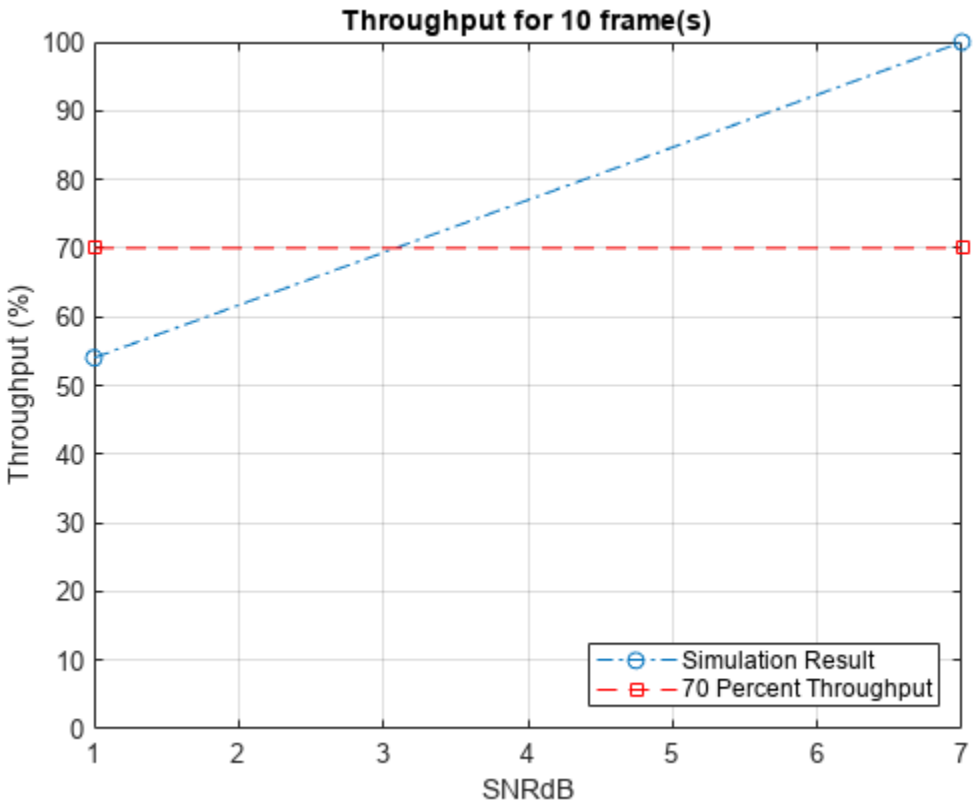
Plot Results

First graph shows the throughput as total bits per second against the range of SNRs, second plots total throughput as a percentage of block CRC errors against SNR range.

```
figure
plot(SNRIn,1e-3*simThroughput/(NFrames*10e-3),'-*',...
     SNRIn,1e-3*0.7*maxThroughput/(NFrames*10e-3),'--rs');
title(['Throughput for ', num2str(NFrames) ' frame(s)'] )
xlabel('SNRdB'); ylabel('Throughput (kbps)');
grid on;
legend('Simulation Result','70 Percent Throughput','Location',...
    'SouthEast')
```




```
figure
plot(SNRIn,simThroughput*100./maxThroughput,'o-.',...
    SNRIn,70*ones(1,numel(SNRIn)),'--rs');
title(['Throughput for ', num2str(NFrames) ' frame(s)'] )
xlabel('SNRdB'); ylabel('Throughput (%)');
grid on;
legend('Simulation Result','70 Percent Throughput','Location',...
    'SouthEast')
ylim([0 100])
```



Selected Bibliography

1 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception"

PDSCH Port 5 UE-Specific Beamforming

This example demonstrates release 8 port 5 UE-specific beamforming with the LTE Toolbox™.

Introduction

There are seven transmission modes in Release 8:

1. Single antenna port, port 0
2. Transmit diversity
3. Open-loop spatial multiplexing,large-delay Cyclic Delay Diversity (CDD)
4. Closed-loop spatial multiplexing
5. Multi-user MIMO
6. Codebook based beamforming (closed-loop spatial multiplexing using a single transmission layer)
7. UE-specific beamforming (single antenna port, port 5)

In transmission mode 7, UE-specific beamforming, arbitrary beamforming is applied and the User Equipment (UE) is not notified of the precoding matrix used, therefore the UE needs to estimate the channel including the effect of beamforming. As the UE requires only the UE specific reference signal for demodulation of the Physical Downlink Shared Channel (PDSCH), the data transmission for the UE appears to have been received from only one transmit antenna, therefore, this transmission mode is described as "single-antenna port, port 5".

Transmissions in this scheme are made on a single layer with a single reference signal and can be beamformed onto any number of transmission antennas using any appropriately dimensioned beamforming vector; the choice of the number of transmission antennas and beamforming vector values are not specified in the standard.

This example shows how the "single antenna port, port 5" transmission scheme can be implemented using the LTE Toolbox to transmit and receive a PDSCH. It also demonstrates that the appropriate choice of beamforming vector leads to better performance.

RMC Generator Setup

In this example a Reference Measurement Channel (RMC) configuration is created using `lteRMCDL`, and reconfigured to describe a UE-specific beamforming configuration. The generation is configured for R.6 and the PDSCH is configured for `TxScheme='Port5'`, the transmission scheme associated with Release 8 UE-specific beamforming in the LTE Toolbox. The number of PDSCH transmission antennas is then set to 4 (The number of columns of the precoding matrix `W` indicates `NTxAnts`), indicating that the UE-specific beamforming will beamform onto 4 transmission antennas. Note that `rmc.CellRefP=1`, meaning that there is only one cell-specific reference signal; this reference signal and associated transmissions that will be mapped onto the first of the 4 transmission antennas.

```
rmc = lteRMCDL('R.6');          % RMC configuration
rmc.TotSubframes = 1;           % Number of subframes to generate
rmc.PDSCH.TxScheme = 'Port5';   % Set UE-specific beamforming scheme
rmc.PDSCH.CSI = 'On';           % CSI scaling of soft bits
```

Channel Estimation Configuration

The channel estimation settings are defined using a structure `cec`. A conservative 9-by-9 pilot averaging window is used to reduce the impact of noise on the channel estimate. Channel estimation is performed using UE-specific reference signals as the `Port5` transmission scheme is used.

```
cec.PilotAverage = 'UserDefined';   % Type of pilot symbol averaging
cec.FreqWindow = 9;                 % Frequency window size
cec.TimeWindow = 9;                 % Time window size
cec.InterpType = 'Cubic';           % 2D interpolation type
cec.InterpWindow = 'Centered';      % Interpolation window type
cec.InterpWinSize = 1;              % Interpolation window size
cec.Reference = 'DMRS';             % Reference for channel estimation
```

System Processing

The following steps are used to create and receive a UE-specific beamformed PDSCH:

- *Create a Populated Transmit Resource Grid:* A transmit grid is created with cell-specific channels but no PDSCH. To do this, `lteRMCDLTool` is used with the 2nd (data) input set to an empty vector. This creates the resource array containing the cell-specific channels of the RMC: Primary Synchronization Signal (PSS), Secondary Synchronization Signal (SSS), Reference Signal (RS), Physical Broadcast Channel (PBCH) and Physical Control Format Indicator Channel (PCFICH). This resource array is mapped to the first transmission antenna within txGrid as `CellRefP=1`.
- *Set the Beamforming Vector:* The beamforming vector `rmc.PDSCH.W` is a field of the PDSCH configuration structure. `rmc.PDSCH.W` is a 1-by-NTxAnts (row) vector indicating the complex gains to be applied to the single layer PDSCH transmission and its associated reference signal.
- *Create and Map UE-Specific Reference Signals:* `lteDMRSIndices` creates the indices for mapping the UE-specific reference signal onto the transmit resource array. `lteDMRS` creates the UE-specific reference signal as a column vector (size M-by-1, where M is the number of UE-specific Reference Signals (RS) Resource Elements (REs) in a subframe) which is the same size as the lteDMRSIndices output. Note that as with other precoding operations in the LTE Toolbox (e.g. using `lteDLPrecode`), the overall beamforming vector `W` is the transpose of what would be expected from the LTE specification i.e. the symbols for layers and antennas lie in columns rather than rows. This is because the LTE Toolbox uses the 2nd (column) rather than 1st (row) dimension to represent transmit antennas (this is consistent with the representation of multichannel signals in MATLAB®).
- *Create and Map the PDSCH:* `ltePDSCHIndices` creates the indices for mapping the PDSCH on the one transmission layer and extends these single-layer indices onto all the transmit planes, resulting in an `rmc.PDSCH.NTxAnts`-column matrix of indices. `ltePDSCH` scrambles and modulates the random input data provided, resulting in a column vector of modulation symbols, performs beamforming of the column vector by multiplication with `rmc.PDSCH.W` to give an `rmc.PDSCH.NTxAnts`-column matrix, which will be the same size as the ltePDSCHIndices output.
- *Create Transmit Waveform:* OFDM modulate the transmit resource grid.
- *Noisy Propagation Channel Modeling:* Channel modeling is performed by multiplying the transmitted waveform `txWaveform` with a fixed channel matrix `H` of size 1-by-NTxAnts which models reception of the 4-antenna transmission on a single antenna. Note that the transpose operations are required when applying the channel matrix `H` as `H` is defined with the typical NRxAnts-by-NTxAnts shape, whereas `txWaveform` uses the 2nd dimension to represent transmit antennas. Additive noise at 28.0 dB SNR is then applied to the received signal.
- *Synchronization, Demodulation, and Channel Estimation:* From the perspective of the receiver, the transmission made using UE-specific beamforming is effectively from a single antenna. Therefore

the channel estimation and equalization attempts to estimate and equalize back to the original single transmission layer; the beamforming vector `W` is part of the overall channel response `HW.'` that will be estimated and equalized. Therefore channel estimation is performed using the UE-specific reference signals; the 2nd argument provides the PDSCH configuration when `TxScheme='Port5'`.

- *PDSCH Reception:* `ltePDSCHIndices` provides matrix `ind` which contains `rmc.PDSCH.NTxAnts` columns. Only the first column of indices is required as there is only one receive antenna. `ltePDSCHDecode` is called to return soft bit estimates `rxBits` along with the receive symbol constellation `rxSymbols` which is plotted for the case of each of the beamforming vectors. Note that within `ltePDSCHDecode`, for UE-specific beamforming the receiver will carry out MMSE equalization across the receive antennas, to perform diversity combining (in this example there is only one receive antenna).

This example runs twice in a loop, the first iteration shows the PDSCH receive constellation when transmitting on all four antennas with the same weighting and the second when transmitting on four antennas with a beamforming vector `W` which is matched to the channel response. In each case a plot of the PDSCH receive constellation is shown. The second constellation exhibits a lower level of noise than the first, indicating better performance.

It is important to note that all the elements of both beamforming vectors have the same magnitude and consequently the transmit antenna powers are the same across all 4 antennas and the total transmit power for either choice of beamforming vector is the same - this means that neither beamforming vector is given an unfair advantage i.e. more transmit power. For the 2nd simulation loop, `W` was chosen such that the overall channel response is 1: `HW.'=1` therefore `W=conj(H)`. For the first simulation loop (i.e. for `W=[17 17 17 17]/34`), if we compute `HW.'` the result is `(16-4*j)/34`, which has a magnitude of sqrt(4/17) which is approximately 0.485. Therefore the beamformer that is matched to the channel has achieved a channel response with a considerably better gain (1 versus approximately 0.485).

```
% Initialize storage variables for comparison
rxSymbolsStore = cell(1, 2);
WStore = zeros(2, 4);

% Loop for transmitting with and without optimal beamforming
for optimalbeamforming = 0:1

    % Configure random number generators
    rng('default');

    % Set PDSCH beamforming vector
    if (optimalbeamforming)
        % Use beamforming vector matched to channel response
        rmc.PDSCH.W = [17 8-15*1i -8+15*1i 15+8*1i]/34;

    else
        % Use equal transmission gains for each antenna
        rmc.PDSCH.W = [17 17 17 17]/34;
    end

    % Create a resource grid without the PDSCH. PDSCH can be turned off by
    % specifying the transport stream input to be empty
    [~, txGrid, info] = lteRMCDLTool(rmc,[]);

    % Create and map UE-specific reference signals
    rmc.PDSCH.NTxAnts = size(rmc.PDSCH.W,2);
```

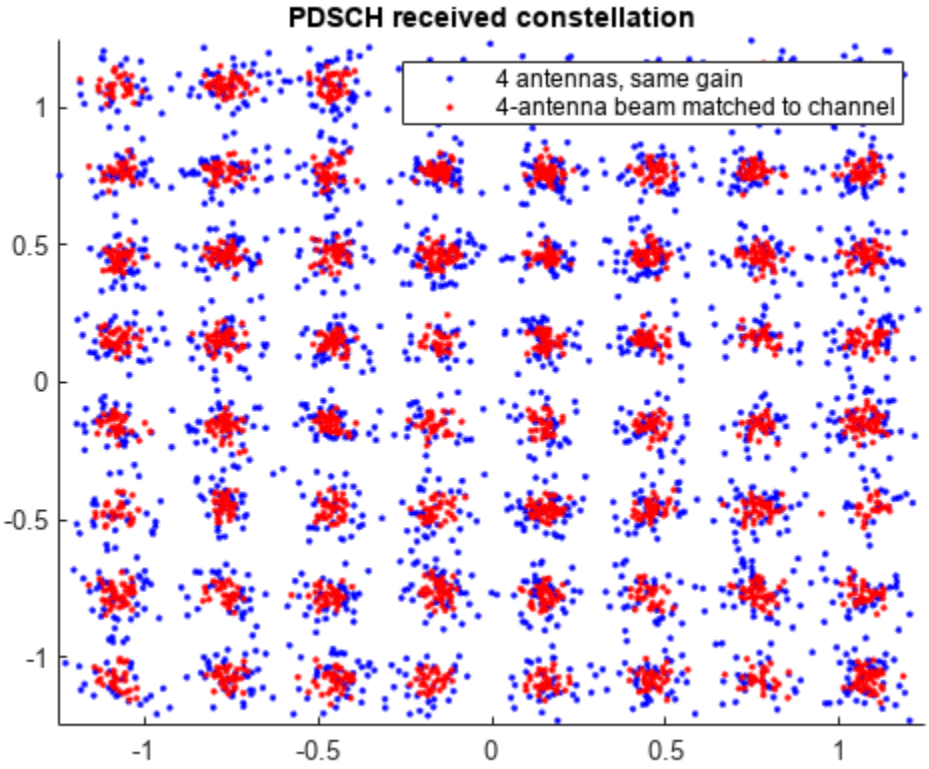
```
    dmRsIndices = lteDMRSIndices(rmc,rmc.PDSCH);
    dmRsSymbols = lteDMRS(rmc,rmc.PDSCH);
    txGrid(dmRsIndices) = dmRsSymbols;

    % Create and map the PDSCH reference signals
    [pdschIndices, pdschIndicesDims] = ltePDSCHIndices(rmc, rmc.PDSCH, ...
        rmc.PDSCH.PRBSet);
    pdschSymbols = ltePDSCH(rmc, rmc.PDSCH, ...
        randi([0 1], pdschIndicesDims.G, 1));
    txGrid(pdschIndices) = pdschSymbols;

    % OFDM modulate to create a transmit waveform
    txWaveform = lteOFDMModulate(rmc, txGrid);

    % Pass waveform through channel
    H = [17 8+15*1i -8-15*1i 15-8*1i]/34;  % Channel response
    rxWaveform = (H*txWaveform.').';

    % Add AWGN noise
    SNRdB = 28;
    SNR = 10^(SNRdB/20);
    N = 1/(sqrt(2.0*double(info.Nfft))*SNR); % Scale for IFFT gain
    noise = N*complex(randn(size(rxWaveform)), randn(size(rxWaveform)));
    rxWaveform = rxWaveform + noise;

    % Synchronization
    offset = lteDLFrameOffset(rmc,rxWaveform);
    rxWaveform = rxWaveform(1+offset:end,:);

    % OFDM demodulation to recover resource grid
    rxGrid = lteOFDMDemodulate(rmc, rxWaveform);

    % Channel and noise estimation
    [hest, nest] = lteDLChannelEstimate(rmc, rmc.PDSCH, cec, rxGrid);

    % Perform Minimum Mean Squared Error (MMSE) equalization and decode the
    % PDSCH
    ind = ltePDSCHIndices(rmc, rmc.PDSCH, rmc.PDSCH.PRBSet);
    ind = ind(:, 1); % Only use one receive antenna
    [rxBits, rxSymbols] = ltePDSCHDecode(rmc, rmc.PDSCH, rxGrid(ind), ...
        hest(ind), nest);

    % Store received symbols and beamforming vector for comparison
    rxSymbolsStore{optimalbeamforming+1} = rxSymbols;
    WStore(optimalbeamforming+1, :) = rmc.PDSCH.W;

end
```

Analysis

The performance in the two simulation loops is compared by plotting the received PDSCH constellations and also displaying the combined channel response `HW.'`. As can be seen from the figure, the system performs better when the beamforming vector has been matched to the channel response. Note that within the LTE specification, no assistance is provided in determining best beamforming vector. Possible approaches to determining the beamforming vector for example would be to exploit channel reciprocity in Time Division Duplex (TDD), or use angle of arrival estimation of the uplink signal in Frequency Division Duplex (FDD).

```
hUESpecificBeamformingResults(rxSymbolsStore, H, WStore);
```

```
4 antennas, same gain, combined channel response HW.': 0.47059-0.11765i
4-antenna beam matched to channel, combined channel response HW.': 1
```



Appendix

This example uses this helper function.

- hUESpecificBeamformingResults.m

Release 10 PDSCH Enhanced UE-Specific Beamforming

This example demonstrates the Release 10 UE-specific beamforming capability of the LTE Toolbox™ and shows how an appropriate choice of beamforming matrix leads to better performance.

Introduction

Release 10 allows for transmission of up to 8 layers on antenna ports 7-14 (TS36.213, Section 7.1.5B [1]). Transmissions in this scheme are made on one or more layers with a reference signal (port) for each layer, and can be beamformed onto any number of transmission antennas using any appropriately-dimensioned beamforming matrix; the choice of the number of transmission antennas and beamforming matrix values are not specified in the standard.

This example shows the Error Vector Magnitude (EVM) improvement achieved when using Precoding Matrix Indicator (PMI) feedback based on channel state information reference signal. A waveform with Physical Downlink Shared Channel (PDSCH) information is created and passed through a noisy fading channel. The received waveform is demodulated, resulting in a received resource grid for each receive antenna. An estimate of the channel is then used to decode the PDSCH, calculate the SNR and singular values of the channel and select an appropriate precoding matrix. The EVM of the received signal is calculated and used to estimate the effective channel SNR. This process is carried out with and without Channel State Information (CSI) and Reference Signal (RS) based PMI feedback to demonstrate the impact on performance.

Reference Measurement Channel Configuration

Generate Reference Measurement Channel (RMC) configuration structure for RMC R.5; amended to create only one subframe and 5 Resource Blocks (RBs), and use 8 antennas.

Release 10 UE-specific beamforming capability is parameterized within the LTE Toolbox as `TxScheme = 'Port7-14'` in conjunction with the appropriate choice of the number of layers `NLayers`. There are a number of standard defined RMCs using Port7-14 transmission scheme supported by LTE Toolbox. Examples are R.43, R.44, R.45, R.45-1, R.48, R.50 and R.51. See `lteRMCDL` for details. The beamforming matrix is represented by the PDSCH configuration field `W`; the number of transmission antennas is given by the number of columns of `W`.

```
rmc = struct;                       % RMC config structure
rmc.RC ='R.5';                      % Base configuration on RMC R.5
rmc.DuplexMode = 'TDD';             % User Time Division Duplex (TDD)
rmc.TotSubframes = 1;               % Configure a single subframe

% Generate the base configuration from RMC R.5 and amend to set the
% parameters required for the Port7-14 transmission scheme. Note that if
% the standard defined RMCs using Port7-14 transmission scheme supported by
% lteRMCDL is used, these parameters will be pre-configured.
rmc = lteRMCDL(rmc);
rmc.NDLRB = 25;                     % 25 Resource Blocks
rmc.NCellID = 10;                   % Cell identity 10
rmc.PDSCH.TxScheme = 'Port7-14';    % Up to 8 layer transmission, ports 7-14
rmc.PDSCH.NLayers = 2;              % 2 transmission layers for beamforming
rmc.PDSCH.NSCID = 0;                % Scrambling identity 0
rmc.CSIRefP = 8;                    % 8 CSI-RS ports
rmc.CSIRSConfig = 0;                % CSI-RS configuration 0
rmc.CSIRSPeriod = 'On';             % Configure CSI-RS always 'on'
rmc.ZeroPowerCSIRSPeriod = 'Off';   % Configure Zero Power CSI-RS 'off'
```

```
rmc.PDSCH.PRBSet = (4:8).';         % 5 allocated RBs
rmc.PDSCH.PMIMode = 'Wideband';     % Wideband PMI mode
rmc.PDSCH.CSI = 'On';               % CSI scaling of soft bits
% Codebook subset definition allowing all codebook entries
rmc.PDSCH.CodebookSubset = '0x1FFFFFFFFFFFFFFFFFFFFFFFFFF';
```

Channel Configuration

The fading channel is configured with an Extended Vehicular A (EVA) profile for 3 receive antennas.

```
channel = struct;                    % Channel config structure
channel.Seed = 8;                    % Channel seed
channel.NRxAnts = 3;                 % 3 receive antennas
channel.DelayProfile = 'EVA';        % Delay profile
channel.DopplerFreq = 5.0;           % Doppler frequency in Hz
channel.MIMOCorrelation = 'Medium'; % Multi-antenna correlation
channel.NTerms = 16;                 % Oscillators used in fading model
channel.ModelType = 'GMEDS';         % Rayleigh fading model type
channel.InitTime = 0.0;              % Initial time
channel.InitPhase = 'Random';        % Random initial phases
channel.NormalizePathGains = 'On';  % Normalize delay profile power
channel.NormalizeTxAnts = 'On';     % Normalize for transmit antennas
```

Channel Estimator Configuration

A special mode of the channel estimator must be used when estimating the channel using UE-specific RS or CSI-RS for the `Port7-14` transmission scheme. This mode provides the appropriate "despreading" operation for the case of reference symbols occupying the same time-frequency locations. `lteDLChannelEstimate` is configured for this mode by configuring a `UserDefined` pilot averaging window, of size 1-by-2 (in frequency and time).

```
cec = struct;                       % Channel estimation config structure
cec.PilotAverage = 'UserDefined';   % Type of pilot symbol averaging
cec.FreqWindow = 1;                 % Frequency window size (special mode)
cec.TimeWindow = 2;                 % Time window size (special mode)
cec.InterpType = 'Cubic';           % 2D interpolation type
cec.InterpWindow ='Centered';       % Interpolation window type
cec.InterpWinSize = 1;              % Interpolation window size
```

Simulation Loop

The simulation is run twice to demonstrate the performance gain when using CSI-RS-based PMI feedback. Two plots are produced:

- The PDSCH receive constellation when transmitting on 2 layers, each on 1 out of 8 transmit antennas
- The PDSCH receive constellation when transmitting on 8 antennas with a beamforming matrix `W` which is matched to the channel response. This is chosen using CSI-RS-based PMI feedback.

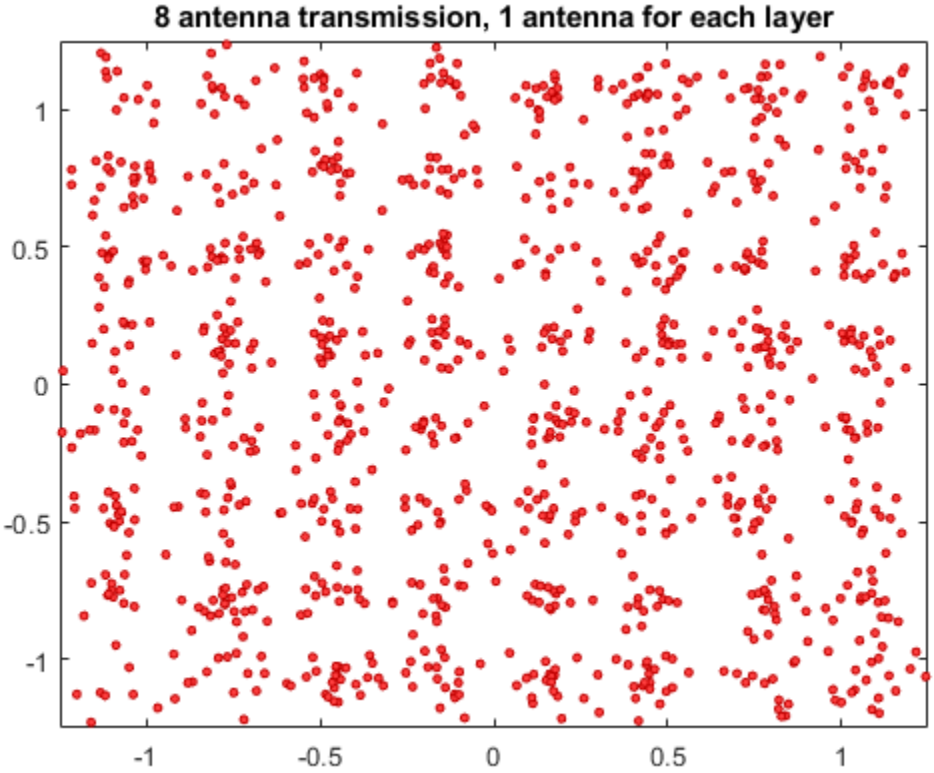
The singular values of each of the two transmissions are also shown, with the channel response being averaged across all allocated PDSCH Resource Elements (REs). These singular values are combined to give the effective SNR of the channel. Finally the SNR of the received symbols is estimated using a measure of the EVM between the transmitted and received PDSCH symbols. These numerical results show an improvement in SNR of approximately 2dB with the beamforming matrix `W` chosen by CSI-RS-based PMI feedback.

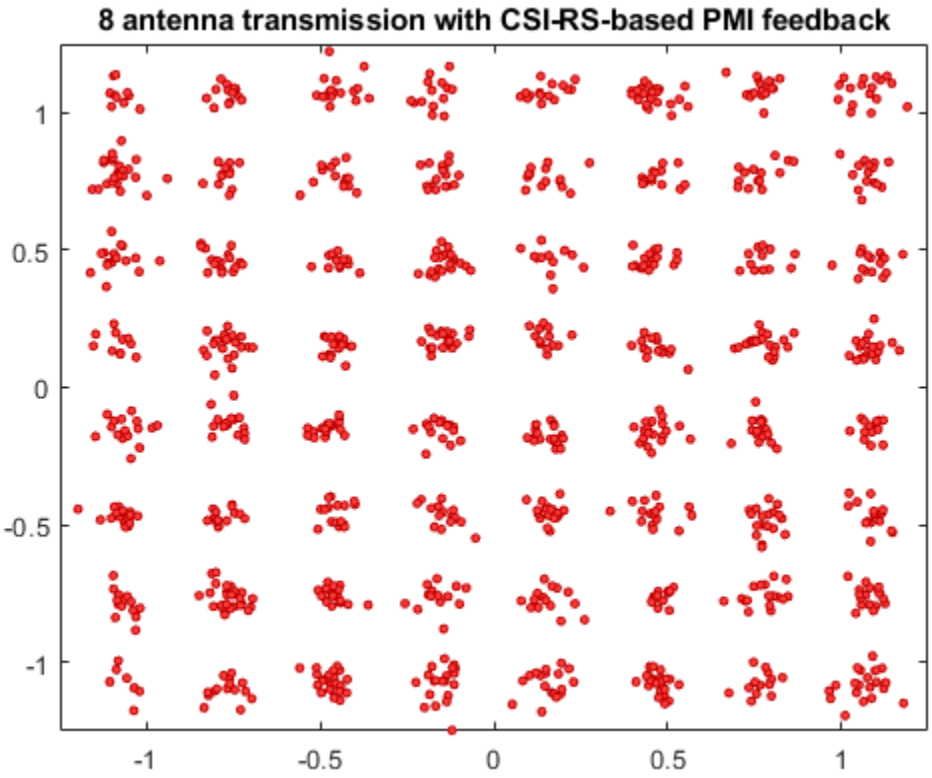
```
% Transmit without then with CSI-RS-based PMI feedback
for csirsFeedback = 0:1
```

```
% Configure random number generators
rng('default');

% Configure PDSCH substructure with transmission beamforming matrix W.
% In the first iteration of the loop transmit each layer on one of the
% 8 antennas. In the second iteration, transmit the layers on 2 beams
% matched to the channel response using CSI-RS-based PMI feedback. The
% PMI fed back to the second iteration is calculated at the end of the
% first
if ~csirsFeedback
    rmc.PDSCH.W = [1 0 0 0 0 0 0 0; ...
                   0 0 0 0 1 0 0 0]/sqrt(2);
else
    rmc.PDSCH.W = lteCSICodebook(rmc.PDSCH.NLayers, ...
                                 rmc.CSIRefP, [PMI(1) PMI(2)]).';
end

% Generate transmission with PDSCH with the beamforming matrix W, onto
% 1st of 8 antenna planes (note that CellRefP = 1 for this RMC). The
% transmitted grid contains UE-specific reference signal (UE-RS / DMRS)
% for channel estimation and CSI-RS reference signal for PMI selection
[~, txGrid, rmcinfo] = lteRMCDLTool(rmc, [1;0;0;1]);
channel.SamplingRate = rmcinfo.SamplingRate;

% OFDM modulation. The additional 25 samples added to the end of the
% waveform are to cover the range of delays expected from the channel
% modeling (a combination of implementation delay and channel delay
% spread)
[txWaveform, ofdmDims] = lteOFDMModulate(rmc, txGrid, 0);
txWaveform = [txWaveform; zeros(25, size(txWaveform,2))]; %#ok

% Fading channel
rxWaveform = lteFadingChannel(channel, txWaveform);

% Create and apply additive white Gaussian noise
if ~csirsFeedback
    SNRdB = 27;
    SNR = 10^(SNRdB/20);
    N = 1/(sqrt(2.0*rmc.CSIRefP*double(ofdmDims.Nfft))*SNR);
    v = N*complex(randn(size(rxWaveform)), randn(size(rxWaveform)));
end
rxWaveform = rxWaveform + v;

% Perform synchronization
offset = lteDLFrameOffset(rmc, rxWaveform);
rxWaveform = rxWaveform(1+offset:end, :);

% Perform OFDM demodulation on the received data to recreate the
% resource grid
rxGrid = lteOFDMDemodulate(rmc, rxWaveform);

% Channel estimation using the UE-specific DMRS for PDSCH reception
cec.Reference = 'DMRS';
[hest, nest] = lteDLChannelEstimate(rmc, rmc.PDSCH, cec, rxGrid);

% Equalize (back to layers) and demodulate the PDSCH.
% Extract REs corresponding to the 2 layers of the PDSCH from the given
```

```
    % subframe across all receive antennas and channel estimates.
    ind = ltePDSCHIndices(rmc, rmc.PDSCH, rmc.PDSCH.PRBSet);
    [pdschRx, pdschHest] = lteExtractResources(ind, rxGrid, hest);
    [rxBits, rxSymbols] = ltePDSCHDecode(rmc, rmc.PDSCH, ...
        pdschRx, pdschHest, nest);

    % Compute singular values of the channel and calculate SNR
    H = squeeze(mean(pdschHest));
    d = svd(H);

    % Print singular values and effective channel SNR
    if csirsFeedback
        label = '8 antenna transmission with CSI-RS-based PMI feedback';
    else
        label = '8 antenna transmission, 1 antenna for each layer';
    end
    fprintf('%s:\n\n', label);
    svdb = sprintf('     %0.2fdB', 20*log10(d));
    fprintf('       Channel singular values:%s\n', svdb);
    fprintf('         Effective channel SNR:     %0.2fdB\n', ...
        SNRdB+10*log10(rmc.PDSCH.NLayers)+10*log10(sum(d.^2)));

    % Regenerate PDSCH from hard bit decisions and demodulate to estimate
    % transmitted symbols
    remod = ltePDSCH(rmc, rmc.PDSCH, rxBits{1}>0);
    [rxBitsRef, rxSymbolsRef] = ltePDSCHDecode(rmc, rmc.PDSCH, remod);

    % Use EVM measurement to estimate SNR
    EVM = comm.EVM;
    evmRMS = EVM(rxSymbolsRef{1},rxSymbols{1});
    SNRest = 20*log10(1/(evmRMS/100));
    fprintf('SNR estimate from receiver EVM:     %0.2fdB\n\n',SNRest);

    % Now compute PMI (via CSI-RS) for use in second iteration. Channel
    % realization remains the same
    if ~csirsFeedback
        % Channel estimation via CSI-RS for PMI selection
        cec.Reference = 'CSIRS';
        [hestPMI, nestPMI] = lteDLChannelEstimate(rmc, rmc.PDSCH, ...
            cec, rxGrid);
        % PMI selection
        PMI = ltePMISelect(rmc, rmc.PDSCH, hestPMI, nestPMI);
    end

    % Plot received constellation
    figure(csirsFeedback+1);
    plot(rxSymbols{1}, 'o', 'MarkerEdgeColor', [0.75 0 0], ...
        'MarkerFaceColor', [1 0.25 0.25], 'MarkerSize',3);
    axis([-1.25 1.25 -1.25 1.25]);
    title(label);

end

8 antenna transmission, 1 antenna for each layer:

       Channel singular values:     -10.19dB     -15.32dB
         Effective channel SNR:     20.98dB
SNR estimate from receiver EVM:     20.73dB
```

```
8 antenna transmission with CSI-RS-based PMI feedback:

       Channel singular values:     -5.38dB     -11.00dB
         Effective channel SNR:     25.68dB
SNR estimate from receiver EVM:     25.06dB
```





Selected Bibliography

1 3GPP TS 36.213 "Physical layer procedures"

LTE DL-SCH and PDSCH Processing Chain

In LTE the Downlink Shared Channel (DL-SCH) is a transport channel used for the transmission of user data, dedicated control and user-specific higher layer information and downlink system information. The Physical Downlink Shared Channel (PDSCH) is the physical channel that carries the DL-SCH coded data. This example shows the different stages involved in the Downlink Shared Channel (DL-SCH) and Physical Downlink Shared Channel (PDSCH) processing and provides access to the data from these intermediate stages.

Introduction

The LTE Toolbox™ provides functions for physical layer modeling with varying levels of granularity ranging from system level functions that can generate the full uplink and downlink waveforms to PHY channel level functions that perform the transport/physical channel processing and individual channel processing stage functions performing CRC coding, turbo coding, etc. These functions, with the simple interface and ease of parameterization, help in rapid prototyping of standard compliant models and therefore are useful in a wide variety of applications. The advantages of a test and verification workflow using individual channel processing stages illustrated in this example are:

- Use as golden reference for alternate implementations
- Ease of creating static or dynamic test vectors for receiver or hardware unit testing
- Understand the DL-SCH/PDSCH processing

The varying levels of granularity allows the users to create models with as much access to intermediate data as required and generate a large number of waveforms or test vectors for automated testing. For the DL-SCH and PDSCH processing and decoding, the toolbox provides `lteDLSCH`, `ltePDSCH`, `ltePDSCHDecode` and `lteDLSCHDecode`. These are channel level functions capable of processing all stages of the relevant transport or physical channel as described in TS 36.212 Section 5.3.2 [1] and TS 36.211 Section 6.4 [2]. This example shows how to use the functions performing individual channel processing steps for DL-SCH and PDSCH encoding and decoding for the use cases where access to the intermediate values/processing stages are required. The various stages of the processing chain and the functions the LTE Toolbox provides for the DL-SCH and PDSCH are shown by the diagrams below.

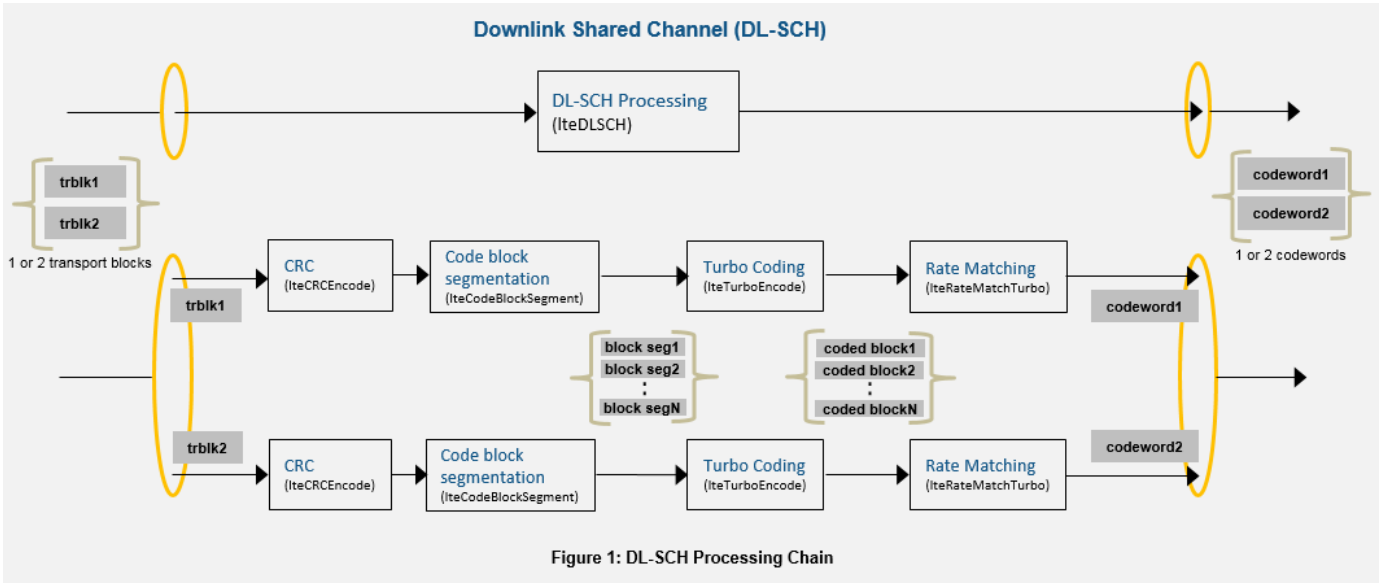


Figure 1: DL-SCH Processing Chain

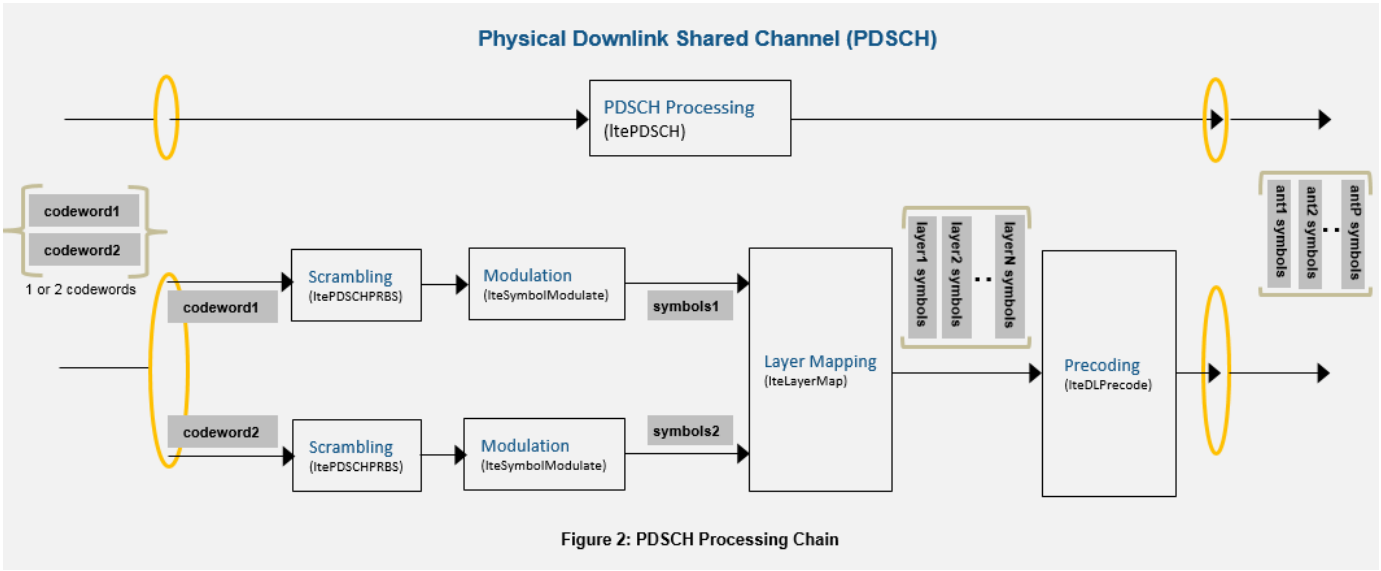


Figure 2: PDSCH Processing Chain

Setup

The functions used in the example require a combination of cell-wide parameters and channel specific parameters. These are input to the functions as fields of structures or as individual parameters.

```
% Cell-wide Settings
% The cell-wide parameters are grouped into a single structure enb. A
% number of the functions used in this example require a subset of the
% parameters specified below. In this example we use the configuration
% according to the RMC R.14 FDD specified in TS 36.101 Annex A.3.4 which
% uses 50 RB, 4 port, 'SpatialMux' transmission scheme, '16QAM' symbol
% modulation, 2 codewords and a code rate of 1/2.
enb.NDLRB = 50;                 % Number of resource blocks
enb.CellRefP = 4;               % Cell-specific reference signal ports
enb.NCellID = 0;                % Cell ID
enb.CyclicPrefix = 'Normal';    % Normal cyclic prefix
```

```
enb.CFI = 2;                    % Length of control region
enb.DuplexMode = 'FDD';         % FDD duplex mode
enb.TDDConfig = 1;              % Uplink/Downlink configuration (TDD only)
enb.SSC = 4;                    % Special subframe configuration (TDD only)
enb.NSubframe = 0;              % Subframe number

% Transport/Physical channel settings for ease of use the DL-SCH and PDSCH
% channel specific settings are specified in a parameter structure pdsch.
% For the R.14 FDD RMC, there are two codewords, so the modulation scheme
% is specified as a cell array containing the modulation schemes of both
% codewords. If configuring for one codeword, the modulation scheme can be
% a character vector or a cell array with character vectors.
% It is also important to configure the TrBlkSizes parameter to have the
% correct number of elements as the intended number of codewords. The
% number of soft bits for the rate matching stage is decided by the UE
% category as specified in TS 36.306 Table 4.1-1. In this example, the
% transport block size is looked up from tables in TS 36.101 Annex A.3.4.
% This can also be done by using the lteRMCDL function for R.14 RMC.

% DL-SCH Settings
TrBlkSizes = [11448; 11448];    % 2 elements for 2 codeword transmission
pdsch.RV = [0 0];               % RV for the 2 codewords
pdsch.NSoftbits = 1237248;      % No of soft channel bits for UE category 2
% PDSCH Settings
pdsch.TxScheme = 'SpatialMux';  % Transmission scheme used
pdsch.Modulation = {'16QAM','16QAM'}; % Symbol modulation for 2 codewords
pdsch.NLayers = 2;              % Two spatial transmission layers
pdsch.NTxAnts = 2;              % Number of transmit antennas
pdsch.RNTI = 1;                 % The RNTI value
pdsch.PRBSet = (0:enb.NDLRB-1)';% The PRBs for full allocation
pdsch.PMISet = 0;               % Precoding matrix index
pdsch.W = 1;                    % No UE-specific beamforming
% Only required for 'Port5', 'Port7-8', 'Port8' and 'Port7-14' schemes
if any(strcmpi(pdsch.TxScheme,{'Port5','Port7-8','Port8', 'Port7-14'}))
    pdsch.W = transpose(lteCSICodebook(pdsch.NLayers,pdsch.NTxAnts,[0 0]));
end
```

Downlink Shared Channel (DL-SCH) Processing

This section explains the DL-SCH transport channel coding. One transport block enters the processing chain every scheduled subframe (for spatial multiplexing schemes, there can be two transport blocks). The transport blocks get coded and rate matched to the PDSCH channel bit capacity. The PDSCH capacity depends on the PRB allocations, modulation scheme, and transmission scheme and this value is provided as an output from the `ltePDSCHIndices` function. The transport channel encoding process includes the following stages as shown in figure 1 above.

- *Transport Block CRC attachment*: Error detection for the transport blocks are provided by a 24-bit CRC according to TS 36.212 Section 5.3.2.1 [1].
- *Code block segmentation and code block CRC attachment*: As shown in the figure 1 above, code block segmentation splits the input data bit vector into a cell array of code block segments (with filler bits and type-24B CRC appended as appropriate) according to the rules of TS 36.212 Section 5.3.2.2 [1]. The function `lteDLSCHInfo` provides code block segmentation information for the given block size.
- *Channel Coding*: The code blocks are individually turbo coded according to TS 36.212 Section 5.3.2.3 [1]. The turbo coder (`lteTurboEncode`) can process a cell array containing all code

block segments in parallel and returns a cell array containing the individual turbo coded block segments.

- *Rate Matching and code block concatenation*: The turbo coded blocks are then individually rate matched according to TS 36.212 Section 5.3.2.4 [1] and the resulting rate matched blocks are concatenated as per TS 36.212 Section 5.3.2.5 [1] to create a single codeword for transmission on the PDSCH.

```
% Random number initialization for creating random transport block(s)
rng('default');

% Convert the modulation scheme char array or cell array to string array
% for uniform processing
 pdsch.Modulation = string(pdsch.Modulation);

% Get the number of codewords from the number of transport blocks
nCodewords = numel(TrBlkSizes);

% Generate the transport block(s)
trBlk = cell(1,nCodewords); % Initialize the codeword(s)
for n=1:nCodewords
    trBlk{n} = randi([0 1],TrBlkSizes(n),1);
end
% Get the physical channel bit capacity required for rate matching from
% ltePDSCHIndices info output
[~,pdschInfo] = ltePDSCHIndices(enb,pdsch,pdsch.PRBSet);

% Define a structure array with parameters for lteRateMatchTurbo
chs = pdsch;
chs(nCodewords) = pdsch; % For 2 codewords, the array has two elements
% Initialize the codeword(s)
cw = cell(1,nCodewords);
for n=1:nCodewords
    % CRC addition for the transport block
    crccoded = lteCRCEncode(trBlk{n},'24A');
    % Code block segmentation returns a cell array of code block segments
    % with filler bits and type-24B CRC appended as required
    blksegmented = lteCodeBlockSegment(crccoded);
    % Channel coding returns the turbo coded segments in a cell array
    chencoded = lteTurboEncode(blksegmented);

    % Bundle the parameters in structure chs for rate matching as the
    % function requires both cell-wide and channel specific parameters
    chs(n).Modulation = pdsch.Modulation{n};
    chs(n).DuplexMode = enb.DuplexMode;
    chs(n).TDDConfig = enb.TDDConfig;
    % Calculate number of layers for the codeword
    if n==1
        chs(n).NLayers = floor(pdsch.NLayers/nCodewords);
    else
        chs(n).NLayers = ceil(pdsch.NLayers/nCodewords);
    end
    % Rate matching returns a codeword after sub-block interleaving, bit
    % collection and bit selection and pruning defined for turbo encoded
    % data and merging the cell array of code block segments
    cw{n} = lteRateMatchTurbo(chencoded,pdschInfo.G(n),pdsch.RV(n),chs(n));
end
```

Physical Downlink Shared Channel (PDSCH) Processing

One or two transport coded blocks (codewords) can be transmitted simultaneously on the PDSCH depending on the transmission scheme used (see TS 36.211 section 6.4 [2]). As shown in figure 2 above, the codewords undergo scrambling, modulation, layer mapping, precoding, optional UE-specific beamforming and resource element mapping. The size of the matrix `precoded` is N-by-P with N being the number of modulation symbols for one antenna port, and P being the number of transmission antennas.

- *Scrambling:* Up to two codewords can be transmitted in a subframe and for each codeword, the bits are scrambled with a different scrambling sequence according to TS 36.211 Section 6.3.1 [2]. The scrambling sequence is initialized at the start of each subframe and depends on `RNTI`, `NCellID`, `NSubframe` and the codeword index.
- *Modulation:* The scrambled codeword(s) is then symbol modulated using one of the modulation schemes ('QPSK', '16QAM', '64QAM' or '256QAM')
- *Layer Mapping:* The complex modulated symbols are then mapped on to one or several layers according to the transmission scheme used (TS 36.211 Section 6.3.3 [1]). For single port (port 0, 5, 7 or 8), a single layer is used. For transmit diversity only one codeword is allowed and the number of layers (2 or 4) must be equal to the number of antenna ports used for the transmission of the physical channel. For spatial multiplexing 1 or 2 codewords can be transmitted on up to 8 layers. The number of layers is less than or equal to the number of antenna ports used for transmission of the physical channel.
- *Precoding:* The precoding stage takes in the M-by-Layers matrix from the layer mapping stage and returns the matrix of size M-by-P for transmission on P antennas as defined in TS 36.211 Section 6.3.4 [2]. For single port (port 0, 5, 7 or 8), this stage is transparent and for transmit diversity, precoding is applied for 2 or 4 antenna ports. Precoding for spatial multiplexing depends on whether antenna ports with cell-specific reference signals ('SpatialMux', 'CDD' and 'MultiUser' transmission schemes) or antenna ports with UE-specific reference signals ('Port5', 'Port7-8', 'Port8' and 'Port7-14' transmission schemes) are used.
- *Mapping to Resource Elements:* The complex modulated symbols are then mapped on to the resource elements as defined in TS 36.211 Section 6.3.5 [2] to create the grid for transmission. This stage is not shown in this example, but can be easily done by creating an empty resource grid using `lteDLResourceGrid` and mapping the symbols to resource elements returned by the `ltePDSCHIndices` function.

```
% Initialize the modulated symbols
modulated = cell(1,nCodewords);
for n=1:nCodewords
   % Generate the scrambling sequence
   scramseq = ltePDSCHPRBS(enb,pdsch.RNTI,n-1,length(cw{n}));
   % Scramble the codewords
   scrambled = xor(scramseq,cw{n});
   % Symbol modulate the scrambled codewords
   modulated{n} = lteSymbolModulate(scrambled,pdsch.Modulation{n});
end
% Layer mapping results in a (symbols per layer)-by-NLayers matrix
layermapped = lteLayerMap(pdsch,modulated);
% Precoding results in a (symbols per antenna)-by-NTxAnts matrix
precoded = lteDLPrecode(enb, pdsch, layermapped);
% Apply beamforming optionally (W should be 1 or identity if no beamforming)
pdschsymbols = precoded*pdsch.W;
```

PDSCH Decoding

The decoding is the inverse of Physical Downlink Shared Channel (PDSCH) processing on the matrix of complex modulated PDSCH symbols, depending on cell-wide settings structure `enb` and channel-specific configuration structure `pdsch`. The channel inverse processing includes the deprecoding, layer demapping and codeword separation, soft demodulation and descrambling. The deprecoding is performed using matrix pseudo inversion of the precoding matrices. For applications involving propagation channels and/or noise, channel estimation and equalization is done on the received symbols before decoding. See `ltePDSCHDecode` for further information.

```
% Deprecoding (pseudo-inverse based) returns (Number of symbols)-by-NLayers matrix
if (any(strcmpi(pdsch.TxScheme,{'Port5' 'Port7-8' 'Port8' 'Port7-14'})))
    rxdeprecoded=pdschsymbols*pinv(pdsch.W);
else
    rxdeprecoded = lteDLDeprecode(enb,pdsch,pdschsymbols);
end
% Layer demapping returns a cell array containing one or two codewords. The
% number of codewords is deduced from the number of modulation scheme
% character vectors
layerdemapped = lteLayerDemap(pdsch,rxdeprecoded);

% Initialize the recovered codewords
cws = cell(1,nCodewords);
for n=1:nCodewords
    % Soft demodulation of received symbols
    demodulated = lteSymbolDemodulate(layerdemapped{n},pdsch.Modulation{n},'Soft');
    % Scrambling sequence generation for descrambling
    scramseq = ltePDSCHPRBS(enb,pdsch.RNTI,n-1,length(demodulated),'signed');
    % Descrambling of received bits
    cws{n} = demodulated.*scramseq;
end
```

DL-SCH Decoding

The Downlink Shared Channel (DL-SCH) decoding includes rate recovery, turbo decoding, block concatenation and CRC calculations. Alternatively the function `lteDLSCHDecode` also provides the same functionality. This function also returns the type-24A transport block CRC decoding result, type-24B code block set CRC decoding result, the HARQ process decoding state and provides parameterization for specifying the initial HARQ process state.

```
% Initialize the received transport block and CRC
rxTrBlk = cell(1,nCodewords);
crcError = zeros(1,nCodewords);
for n=1:nCodewords
    % Rate recovery stage also allows combining with soft information for
    % the HARQ process, using the input cbsbuffers. For the first
    % transmission of the transport block, the soft buffers are initialized
    % as empty. For retransmissions, the parameter cbsbuffers should be the
    % soft information from the previous transmission
    cbsbuffers = [];  % Initial transmission of the HARQ process
    % Rate recovery returns a cell array of turbo encoded code blocks
    raterecovered = lteRateRecoverTurbo(cws{n},TrBlkSizes,pdsch.RV(n),chs(n),cbsbuffers);
    NTurboDecIts = 5; % Number of turbo decoding iteration cycles
    % Turbo decoding returns a cell array of decoded code blocks
    turbodecoded = lteTurboDecode(raterecovered,NTurboDecIts);
    % Code block desegmentation concatenates the input code block segments
    % into a single output data block, after removing any filler and
```

```
    % type-24B CRC bits that may be present
    [blkdesegmented,segErr] = lteCodeBlockDesegment(turbodecoded,(TrBlkSizes+24));
    % CRC decoding returns the transport block after checking for CRC error
    [rxTrBlk{n},crcError(n)] = lteCRCDecode(blkdesegmented,'24A');
end
```

Conclusion

This example explained the Downlink Shared Channel (DL-SCH) and Physical Downlink Shared Channel (PDSCH) processing and provided an insight into the different functions available within LTE Toolbox to support these channels. The example also illustrated how the low level functions can be used to model the channels and this approach can be used in applications including golden reference test vector generation from these intermediate processing stages to independently validate the different processing stages of alternate implementations. This example also shows how the LTE Toolbox and MATLAB platform enables the creation of a powerful environment for large scale verification and test.

Further Exploration

You can modify the parameters provided in this example to experiment with different configurations. For e.g. when simulating for different transmission modes, some of the parameters of interest are transmission scheme (`TxScheme`), modulation scheme (`Modulation`), number of codewords (number of elements of `TrBlkSizes`).

Selected Bibliography

1. 3GPP TS 36.212 "Multiplexing and channel coding"
2. 3GPP TS 36.211 "Physical channels and modulation"
3. 3GPP TS 36.213 "Physical layer procedures"
4. 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception"
5. 3GPP TS 36.306 "User Equipment (UE) radio access capabilities"

Modeling and Testing an LTE RF Transmitter

This example shows how to characterize the impact of radio frequency (RF) impairments, such as in-phase and quadrature (IQ) imbalance, phase noise, and power amplifier (PA) nonlinearities, on the performance of an LTE transmitter. The example generates the baseband LTE waveform, which consists of an E-UTRA test model (E-TM), by using LTE Toolbox™ software and models the RF transmitter by using RF Blockset™ software.

Introduction

This example characterizes the impact of RF impairments such as in-phase and quadrature imbalance, phase noise, and power amplifier nonlinearities on the performance of an LTE RF transmitter. To evaluate the performance, the example performs these measurements:

- Error vector magnitude (EVM): vector difference at a given time between the ideal (transmitted) signal and the measured (received) signal. The example performs EVM measurements according to the specifications in TS 36.104, Annex E.
- Occupied bandwidth: bandwidth that contains 99% of the total integrated power of the signal, centered on the assigned channel frequency
- Channel power: filtered mean power centered on the assigned channel frequency
- Complementary cumulative distribution function (CCDF): probability that the signal's instantaneous power is at a specified level above its average power
- Peak-to-average power ratio (PAPR): relation between the peak power of the signal and its average power

The example works on a subframe-by-subframe basis and uses a Simulink® model to perform these steps:

1. Generate the baseband E-TM waveform using LTE Toolbox functions.
2. Import the baseband waveform into the RF Transmitter Subsystem block implemented by using RF Blockset blocks. The model uses an RF intermediate frequency to carry the baseband information in RF Blockset.
3. Model the effects of upconverting the waveform to the carrier frequency by using the RF Transmitter Subsystem block. This block models the impairments introduced by an RF transmitter using RF Blockset blocks.
4. Calculate the occupied bandwidth and the channel power by using the Spectrum Analyzer block.
5. Compute the CCDF and PAPR.
6. Extract the data symbols and measure the EVM by demodulating the baseband waveform.

The Simulink model uses LTE Toolbox and DSP System Toolbox™ features to process the baseband signal (steps 1, and 4-6) and uses RF Blockset blocks to model the RF transmitter (steps 2 and 3). This model supports Normal and Accelerator simulation modes.

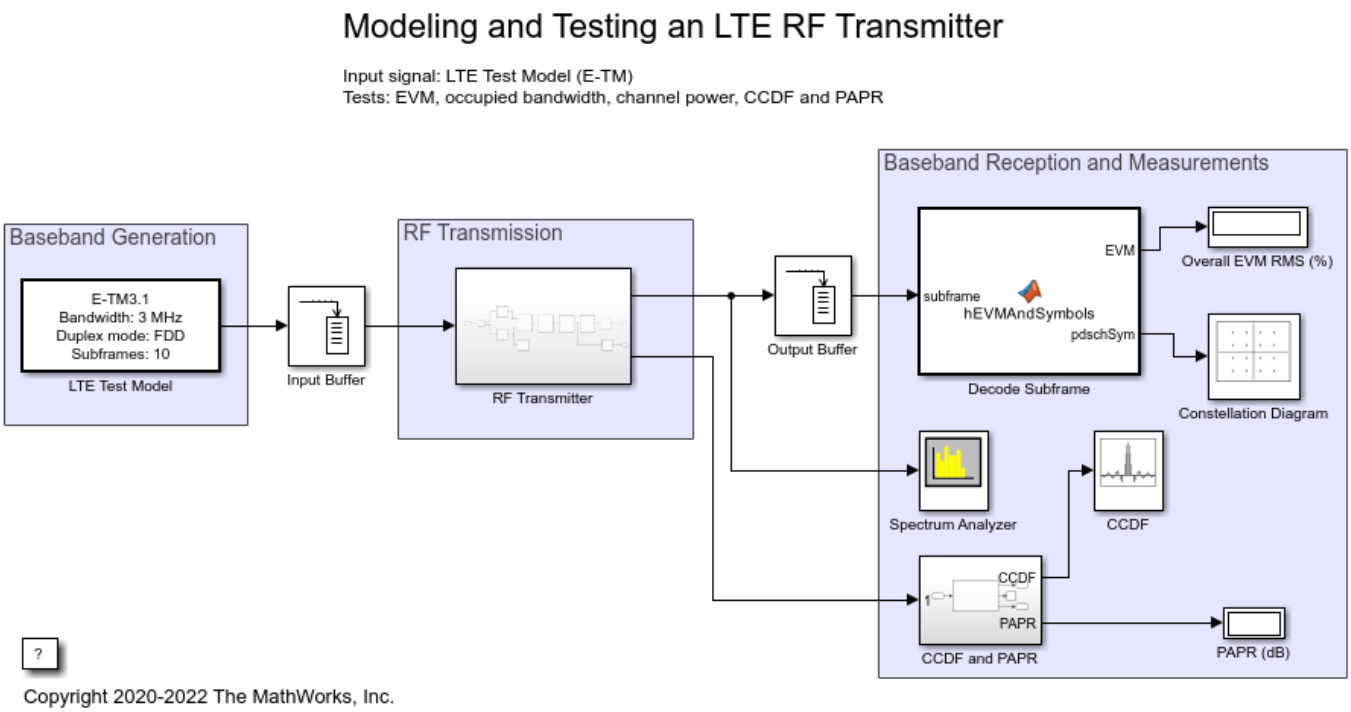
Simulink Model Structure

The model contains three main parts:

- Baseband Waveform Generation: generates the baseband E-TM waveforms
- RF Transmission: models the effects of upconverting the waveform to the carrier frequency

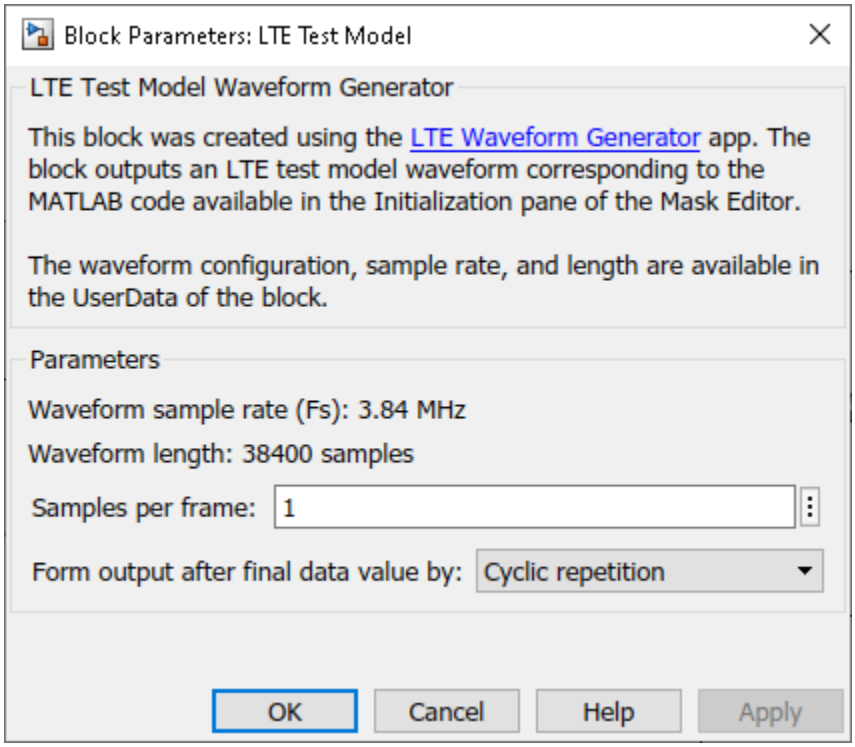
- Baseband Reception and Measurements: performs the RF measurements and calculates EVM by demodulating the baseband waveform

```
modelName = 'RFLTETransmitterModel';
open_system(modelName);
```



Baseband Generation

The LTE Test Model block transmits a standard-compliant LTE test model 3.1 (E-TM3.1) waveform, as defined in TS 36.141. This block is generated using the **LTE Waveform Generator** app. You can access the waveform configuration parameters in the user data of the block. This example uses the **InitFcn** in the **Model callbacks** to store the structure available in the user data in a Base Workspace variable, `LTEInfo`. For more information about this block, see Waveform From Wireless Waveform Generator App.



To use a different E-TM waveform, open the **LTE Waveform Generator** app, select the E-TM configuration, and export a new block. For more information on how to generate and use this block, see “Generate Wireless Waveform in Simulink Using App-Generated Block” on page 2-85.

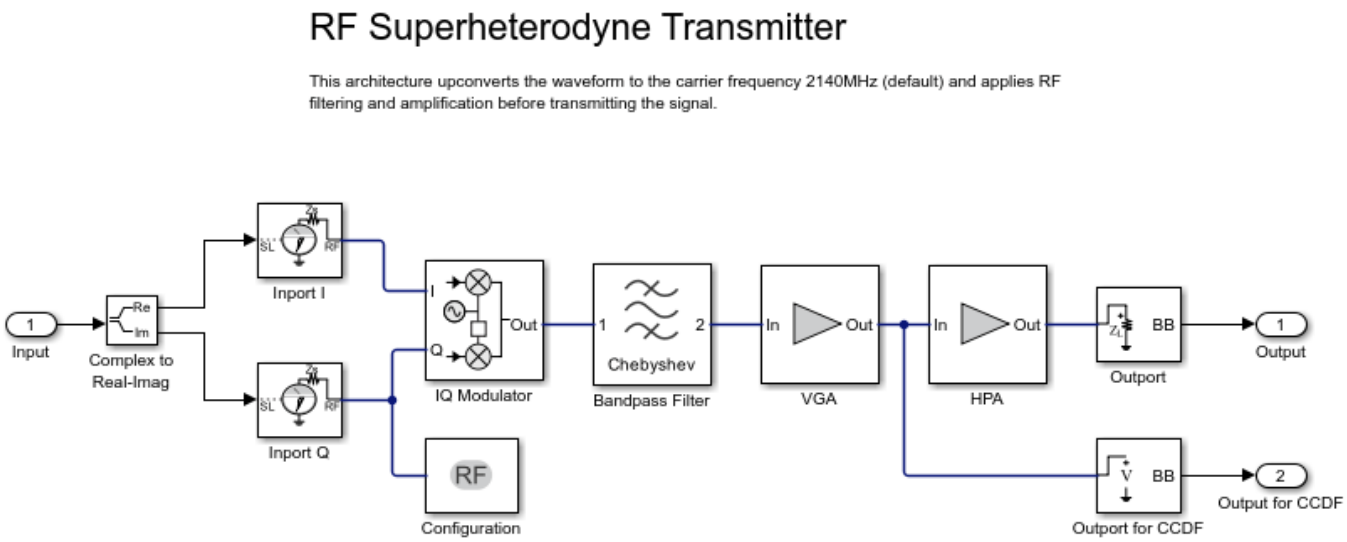
RF Transmission

The RF Transmitter Subsystem block is based on a superheterodyne transmitter architecture. This architecture models the effects of upconverting the waveform to the carrier frequency by characterizing these RF components:

- IQ modulator consisting of mixers, a phase shifter, and a local oscillator
- Bandpass filter
- Power amplifier

In addition to these components, this RF Transmitter Subsystem block also includes a variable gain amplifier (VGA) to control the input back-off (IBO) level of the high power amplifier (HPA).

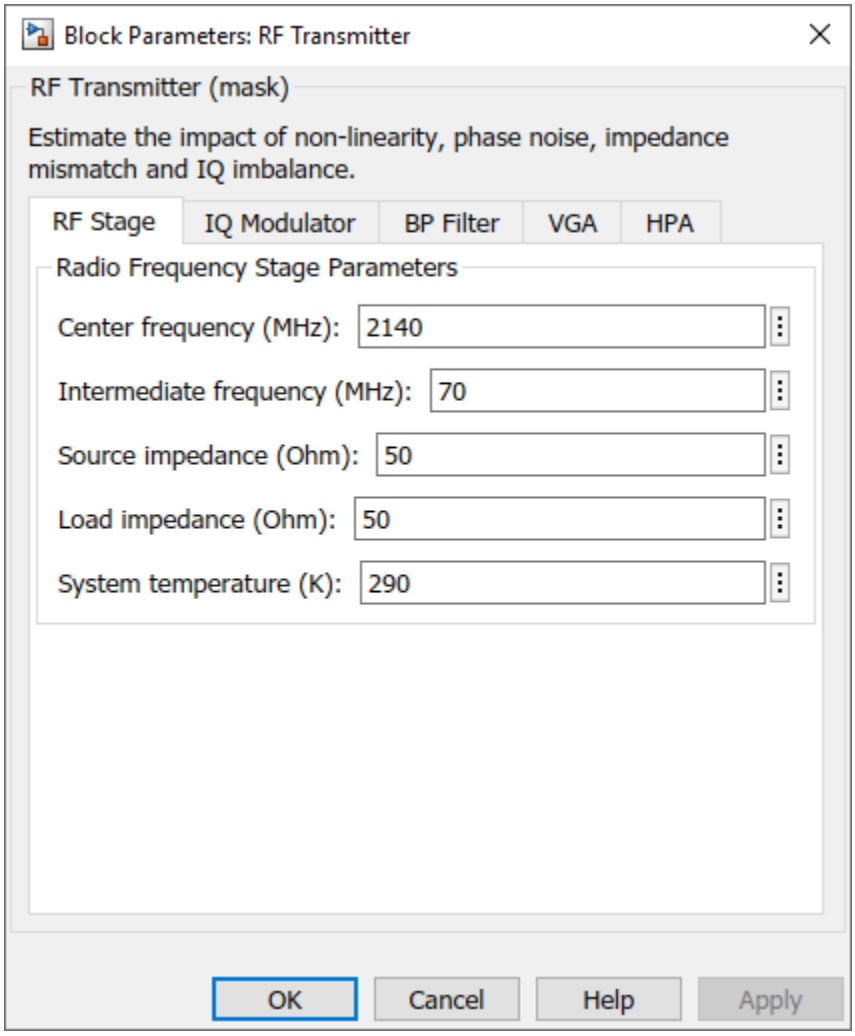
```
set_param(modelName,'Open','off');
set_param([modelName '/RF Transmitter'],'Open','on');
```



Use an Input Buffer block to send one sample at a time to the RF Transmitter Subsystem block.

The Inport block inside the RF Transmitter Subsystem block converts the Simulink complex baseband waveform into the RF Blockset Circuit Envelope simulation environment. The **Carrier frequencies** parameter of the Inport block specifies the center frequency of the carrier in the RF Blockset domain. The Outport block converts the RF Blockset signal back into Simulink complex baseband.

You can configure the RF Transmitter components using the RF Transmitter Subsystem block mask.



The RF Transmitter Subsystem block models typical impairments, including:

- I/Q imbalance as a result of gain or phase mismatches between the parallel sections of the transmitter chain dealing with the IQ signal paths.
- Phase noise as a secondary effect directly related to the thermal noise within the active devices of the oscillator.
- HPA nonlinearities due to DC power limitation when the amplifier works in saturation region.

Before sending the samples onto the Decode Subframe block, the Output Buffer (after the RF Transmitter) collects all samples within a subframe.

The use of buffers in the model generates time delays. As the duration of the delay is equivalent to the transmission of a subframe, the Decode Subframe block does not demodulate the first subframe.

Baseband Reception and Measurements

The Decode Subframe block performs OFDM demodulation of the received subframe, channel estimation, and equalization to recover and plot the PDSCH symbols in the Constellation Diagram. This block also performs EVM measurements according to the specifications in TS 36.104, Annex E.

These specifications suggest measuring the EVM at two locations in time (low and high), where the low and high locations correspond to the alignment of the FFT window within the start and end of the cyclic prefix, respectively. For more information regarding how to measure EVM, see “PDSCH Error Vector Magnitude (EVM) Measurement” on page 2-583.

This example carries out EVM measurements averaged over the allocated PDSCH symbols within a subcarrier, resource block, OFDM symbol, subframe, frame, and overall grid. The EVM per subframe results show the EVM for the high and low locations in time while the other results only depict the highest EVM of both locations.

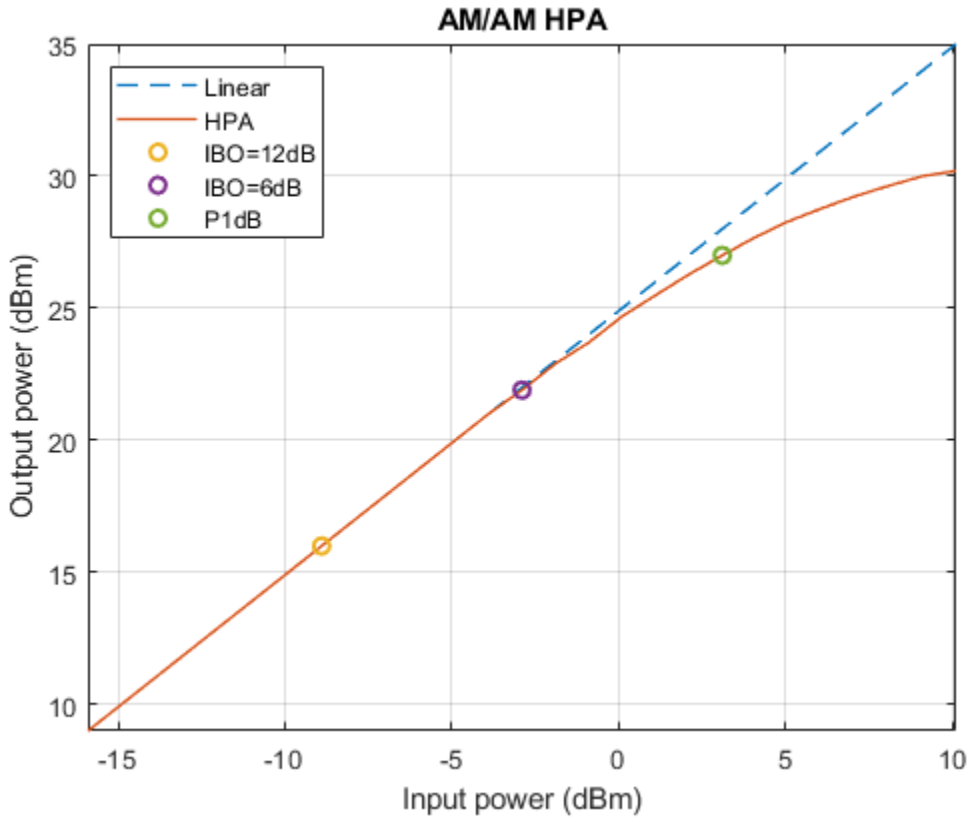
The Spectrum Analyzer block provides frequency-domain measurements such as occupied bandwidth and channel power. A Power Meter block, called CCDF and PAPR, connected at the input of the HPA block depicts the CCDF and PAPR measurements.

To receive one frame, you must simulate a total of 10 ms for FDD and 20 ms for TDD. If the simulation time is longer than a frame, the LTE Test Model block cyclically transmits the same LTE frame.

Effect of Power Amplifier

To characterize the impact of the HPA in the EVM evaluation, you can measure the amplitude-to-amplitude modulation (AM/AM) of the HPA. The AM/AM refers to the output power levels in terms of the input power levels. The helper function `hPlotHPACurveLTE` displays the AM/AM characteristic of the HPA selected for this model.

```
hPlotHPACurveLTE();
figHPA = gcf;
```

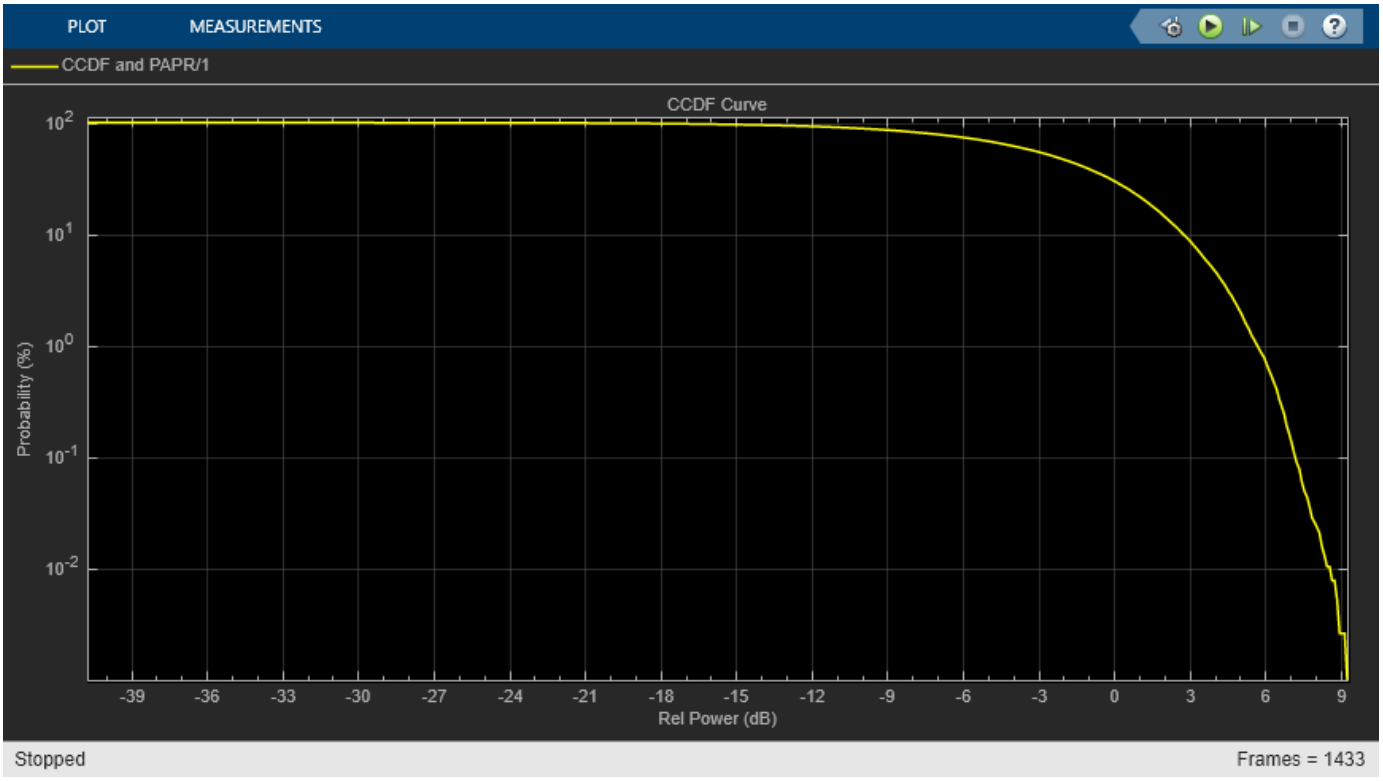


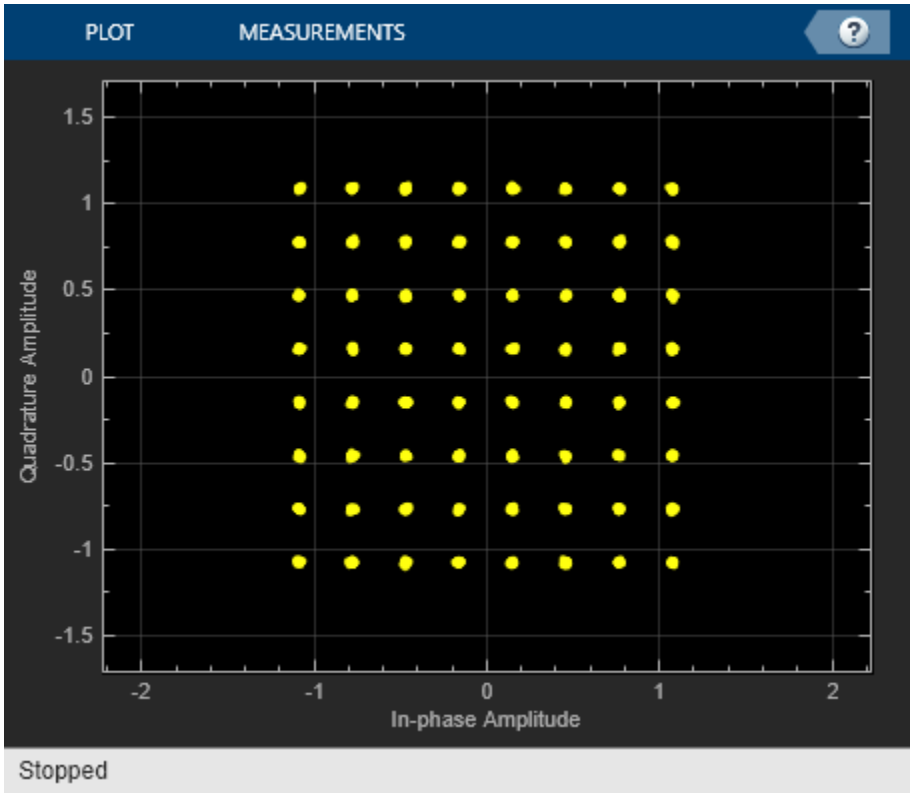
P1dB is the power at 1 dB compression point and is normally used as a reference when selecting the IBO level of the HPA. You can see the HPA impact on the RF transmitter by analyzing the EVM results for different operating points of the HPA. For example, compare the case when IBO = 12 dB, corresponding to HPA operating in the linear region, with the case when IBO = 6 dB, corresponding to HPA starting to operate in the nonlinear region. The gain of the VGA controls the IBO level. To keep a VGA linear behavior, select gain values lower than 20 dB.

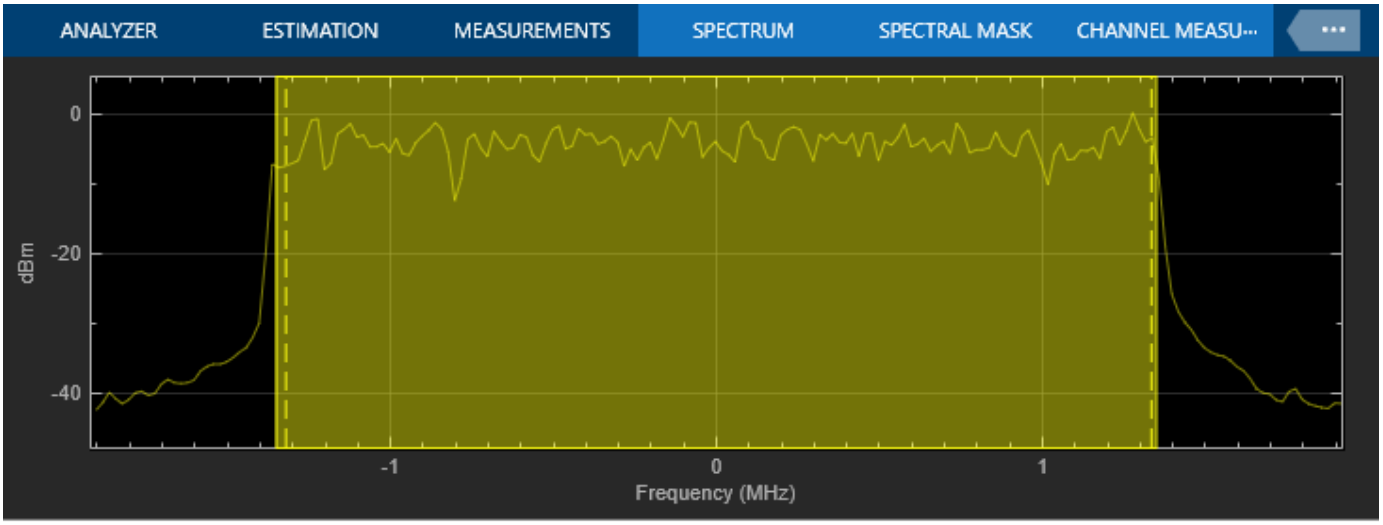
- *Linear HPA (IBO = 12 dB)*. To operate at an IBO level of 12 dB, set the **Available power gain** parameter of the VGA block to 0 dB. To simulate a whole frame, run the simulation long enough to capture 10 subframes (Stop Time equal to 10 ms for FDD). During simulation, the model displays the spectrum and the constellation diagram.

```
set_param([modelName '/RF Transmitter'],'vgaGain','0');
sim(modelName);
```

```
Low edge EVM, subframe 0: 1.653%
High edge EVM, subframe 0: 1.688%
Low edge EVM, subframe 1: 1.451%
High edge EVM, subframe 1: 1.486%
Low edge EVM, subframe 2: 1.591%
High edge EVM, subframe 2: 1.617%
Low edge EVM, subframe 3: 1.574%
High edge EVM, subframe 3: 1.605%
Low edge EVM, subframe 4: 1.349%
High edge EVM, subframe 4: 1.365%
Low edge EVM, subframe 5: 1.144%
High edge EVM, subframe 5: 1.184%
Low edge EVM, subframe 6: 1.348%
High edge EVM, subframe 6: 1.384%
Low edge EVM, subframe 7: 2.081%
High edge EVM, subframe 7: 2.086%
Low edge EVM, subframe 8: 2.179%
High edge EVM, subframe 8: 2.187%
Low edge EVM, subframe 9: 1.890%
High edge EVM, subframe 9: 1.901%
Averaged low edge EVM, frame 0: 1.659%
Averaged high edge EVM, frame 0: 1.681%
Averaged EVM frame 0: 1.681%
Averaged overall EVM: 1.681%
```


PLOT
MEASUREMENTS
CCDF and PAPR/1
CCDF Curve
Probability (%)
Rel Power (dB)
Stopped
Frames = 1433

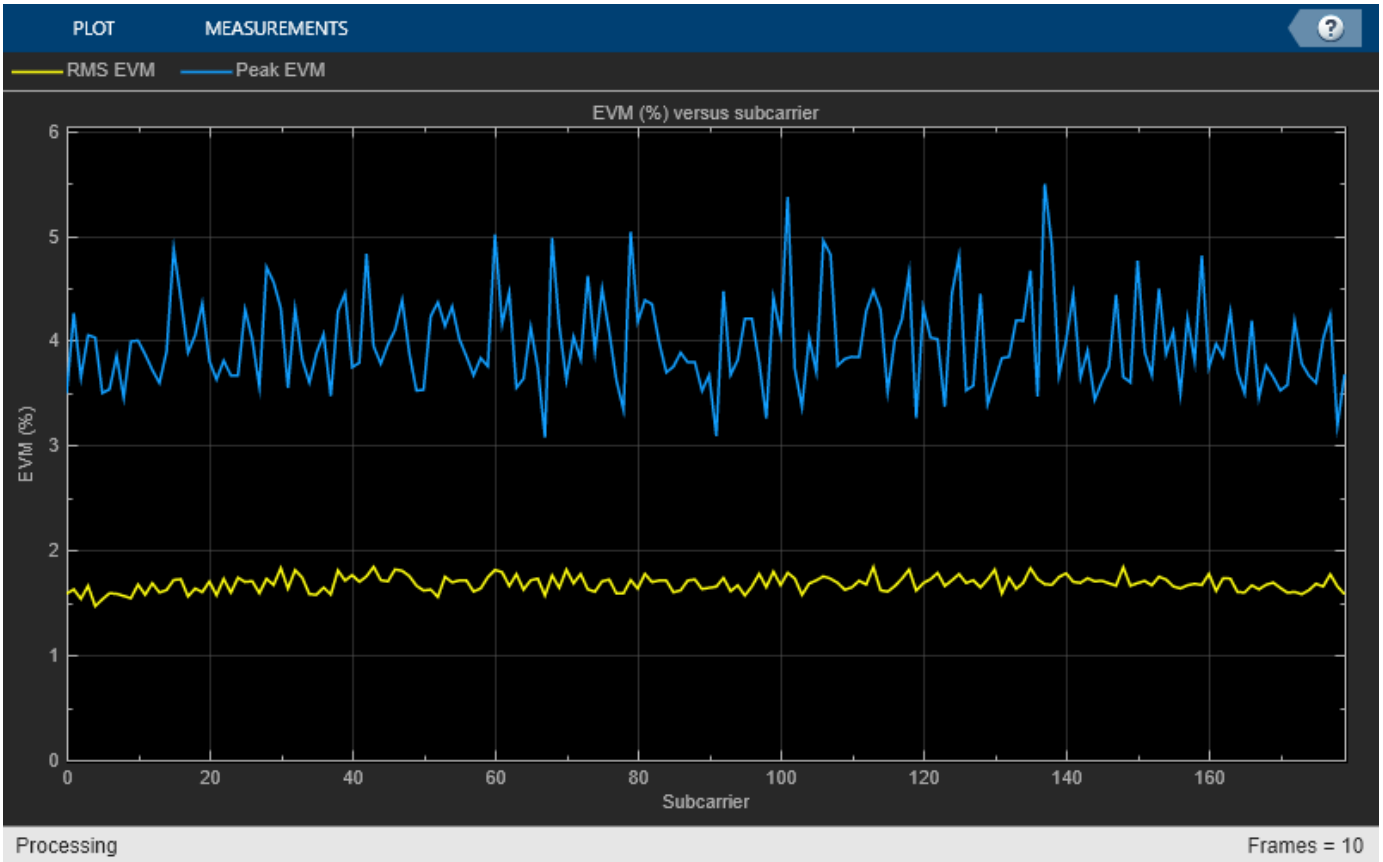

PLOT
MEASUREMENTS
Quadrature Amplitude
In-phase Amplitude
Stopped



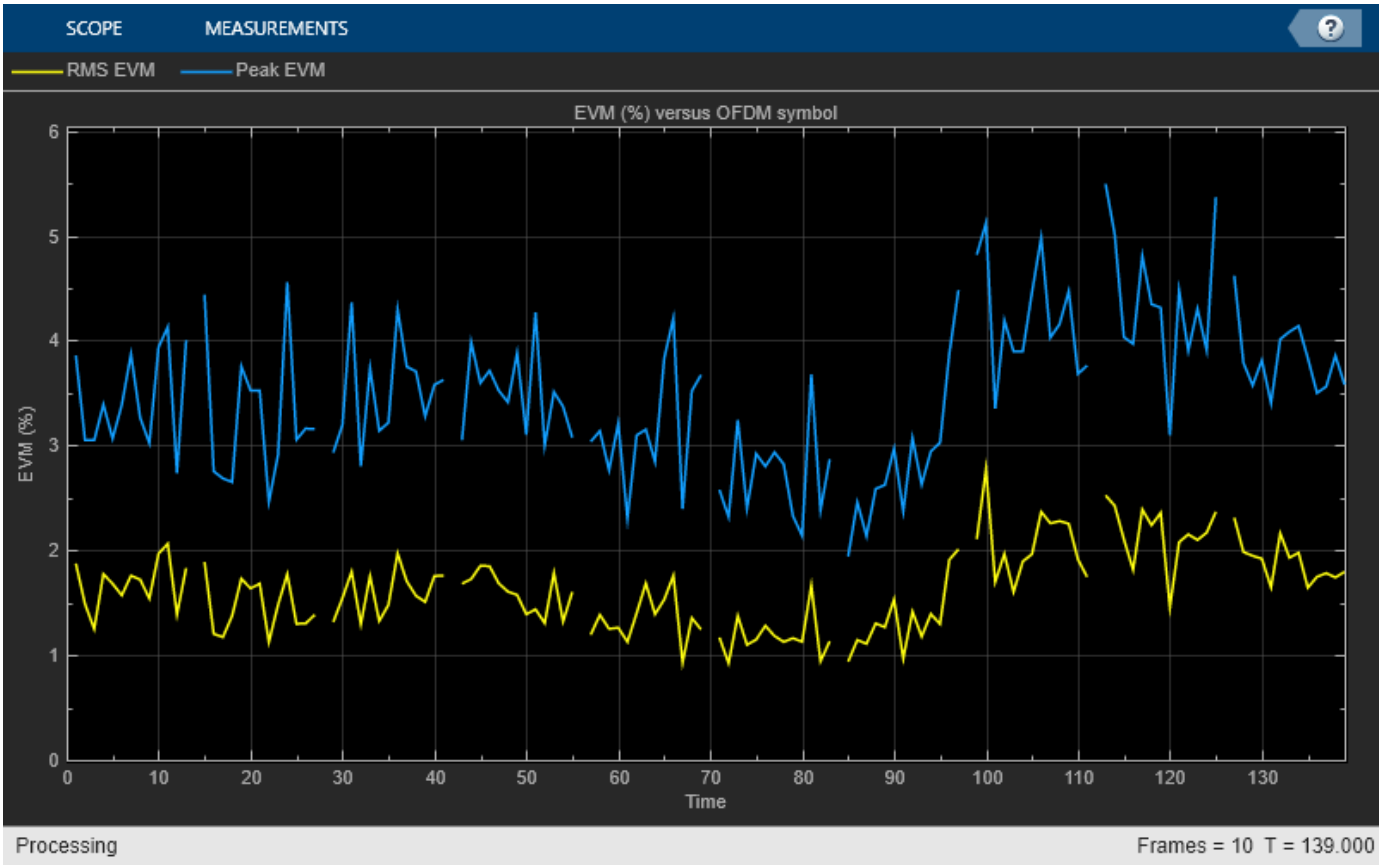
Occupied Bandwidth

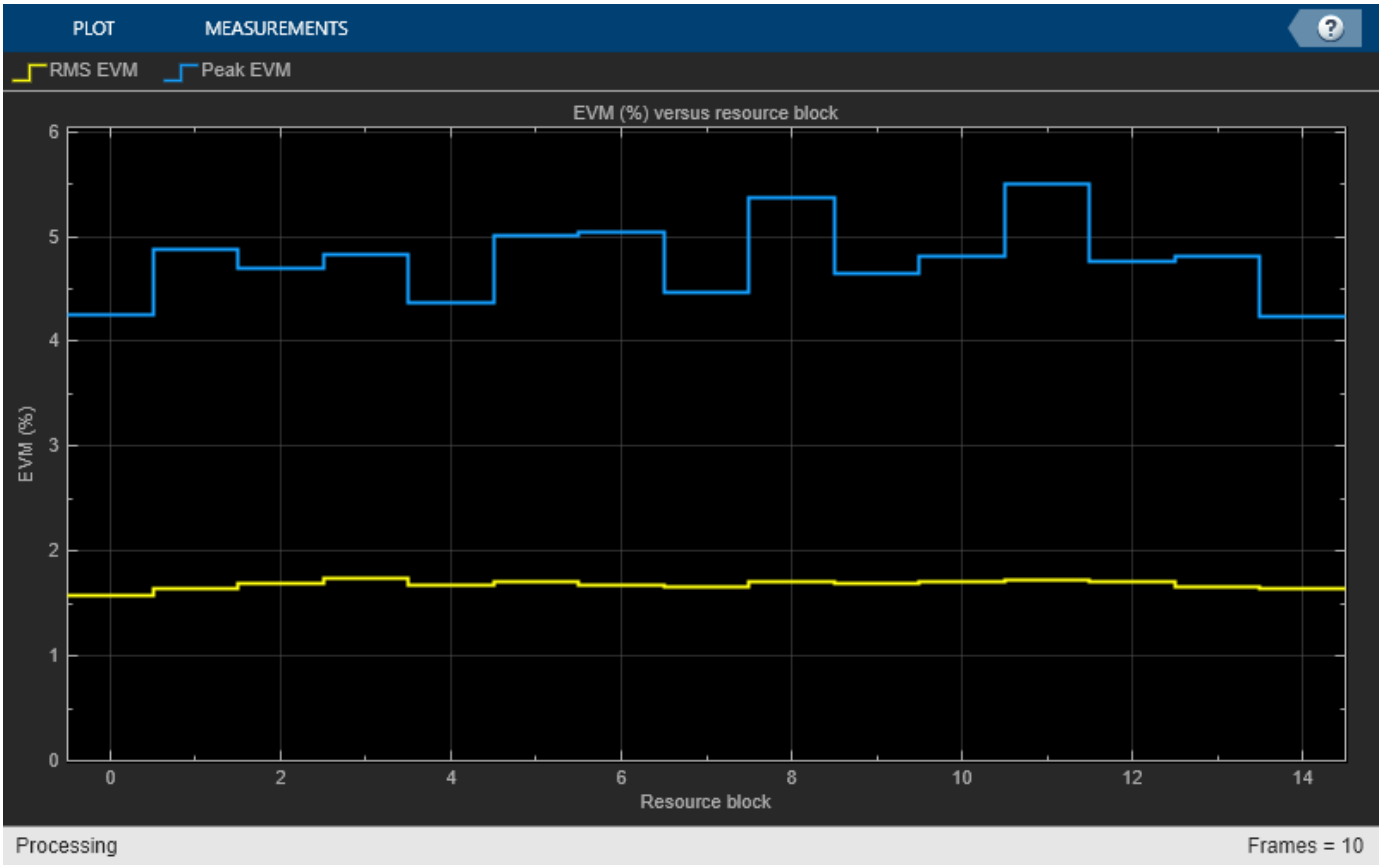
Channel 1	Channel Power (dBm)	Occupied Bandwidth (MHz)	Frequency Error (kHz)
	15.6983	2.6554	9.9457

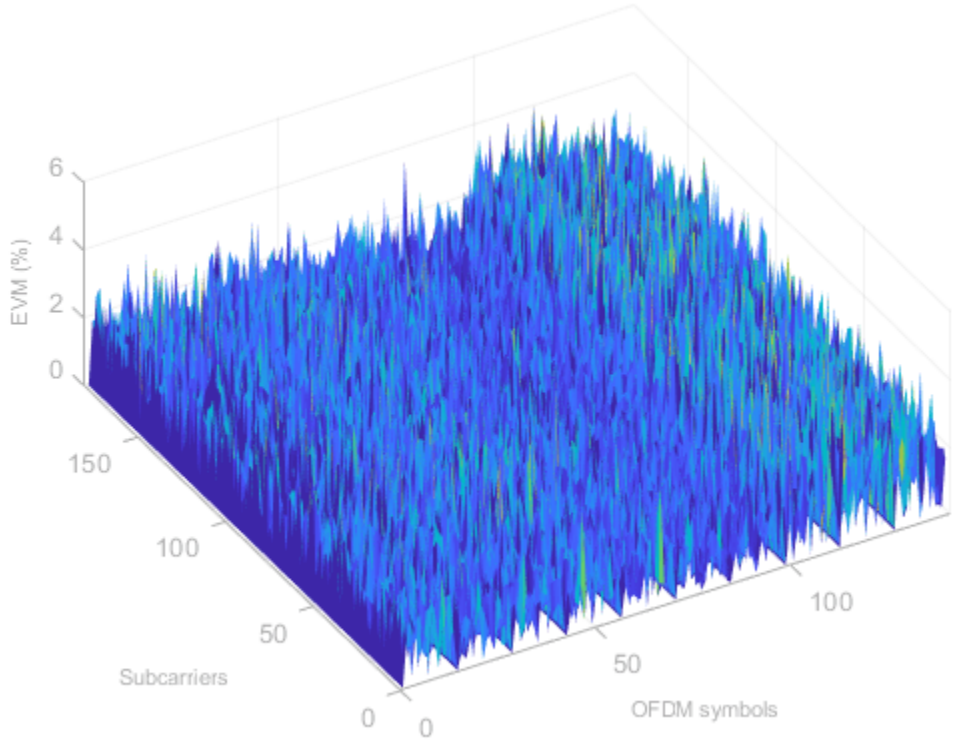
Stopped VBW = 3.00000 kHz RBW = 30.0000 kHz Sample Rate = 3.84000 MHz Frames = 4311 T = 0.00999271



Processing Frames = 10


SCOPE
MEASUREMENTS
RMS EVM
Peak EVM
EVM (%) versus OFDM symbol
EVM (%)
Time
0
1
2
3
4
5
6
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Processing
Frames = 10 T = 139.000


PLOT
MEASUREMENTS
RMS EVM
Peak EVM
EVM (%) versus resource block
EVM (%)
Resource block
0
1
2
3
4
5
6
0
2
4
6
8
10
12
14
Processing
Frames = 10



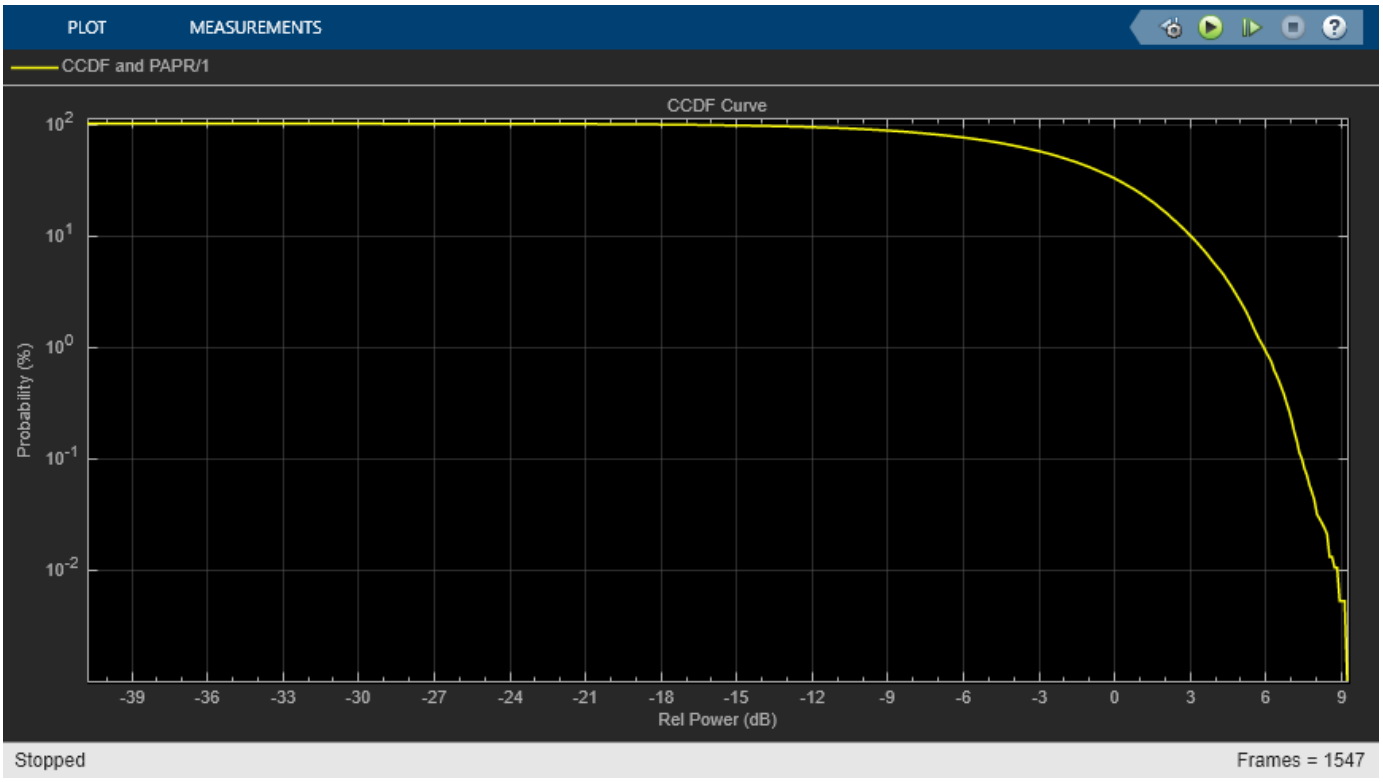
According to TS 36.104, the maximum EVM when the constellation is 64-QAM is 8%. As the overall EVM is around 1.7%, this architecture falls within the requirements of TS 36.104.

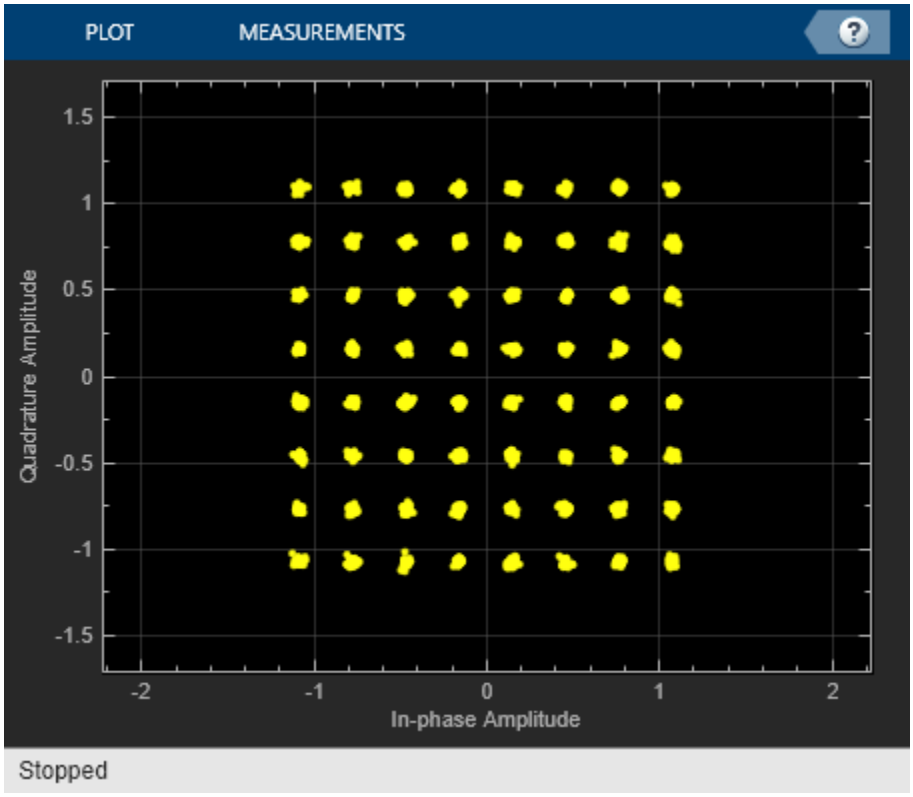
- *Nonlinear HPA (IBO = 6 dB)*. To operate at an IBO level of 6 dB, set the **Available power gain** parameter of the VGA block to 6 dB.

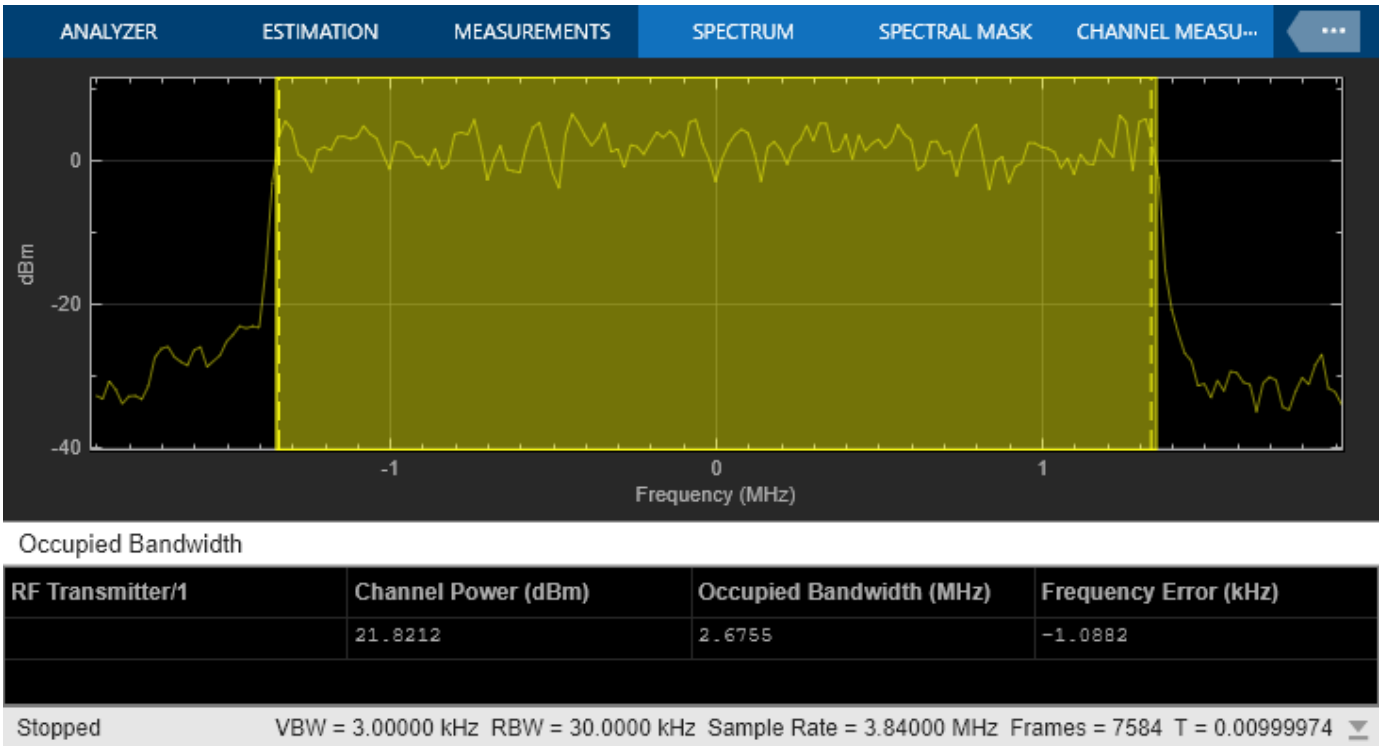
```
set_param([modelName '/RF Transmitter'],'vgaGain','6');
sim(modelName);
slmsgviewer.DeleteInstance();

% Restore to default parameters
set_param([modelName '/RF Transmitter'],'vgaGain','0');
```

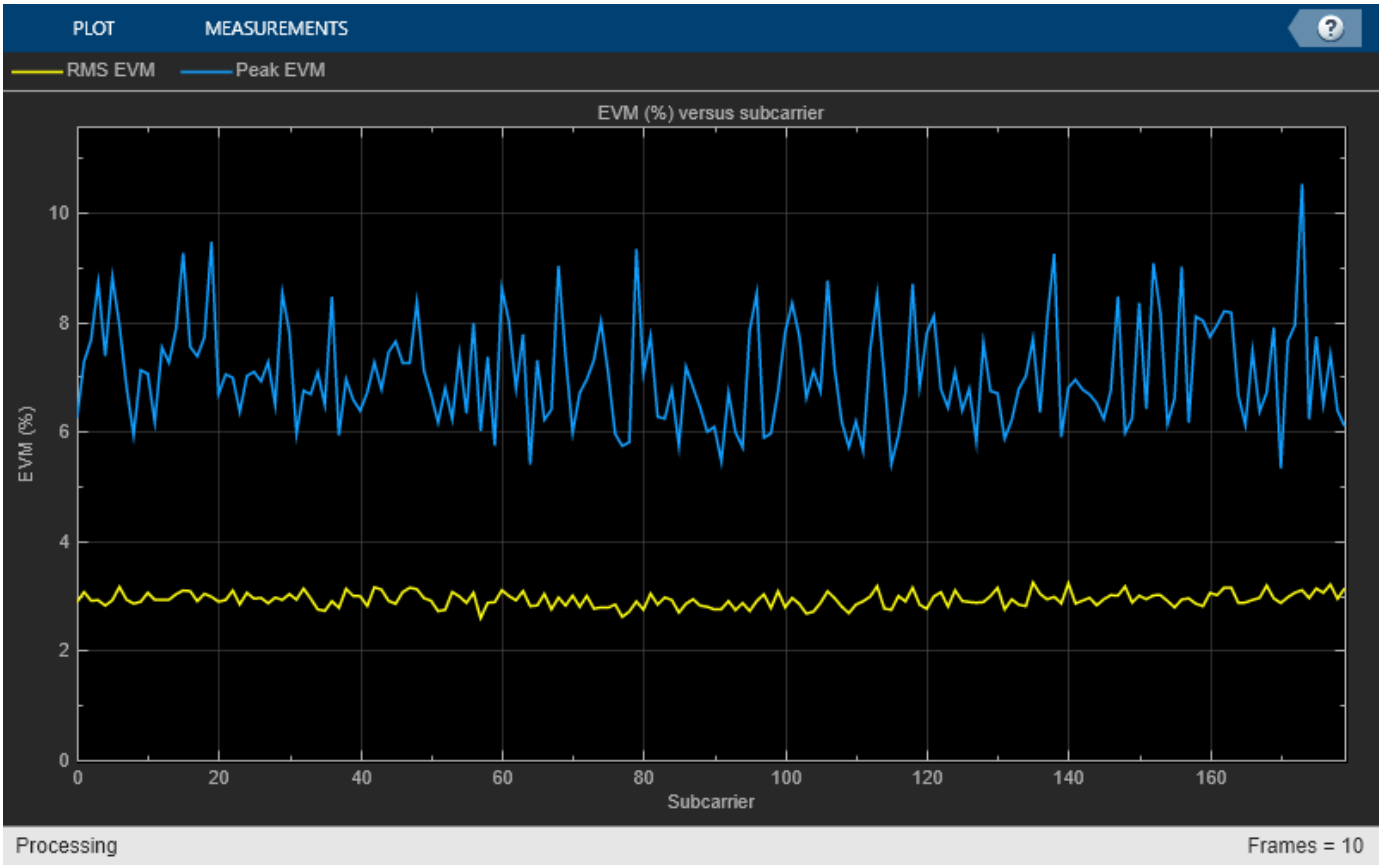
```
Low edge EVM, subframe 0: 2.926%
High edge EVM, subframe 0: 2.950%
Low edge EVM, subframe 1: 2.662%
High edge EVM, subframe 1: 2.678%
Low edge EVM, subframe 2: 2.675%
High edge EVM, subframe 2: 2.695%
Low edge EVM, subframe 3: 2.664%
High edge EVM, subframe 3: 2.683%
Low edge EVM, subframe 4: 2.593%
High edge EVM, subframe 4: 2.598%
Low edge EVM, subframe 5: 2.344%
High edge EVM, subframe 5: 2.362%
Low edge EVM, subframe 6: 2.556%
High edge EVM, subframe 6: 2.576%
Low edge EVM, subframe 7: 3.650%
```

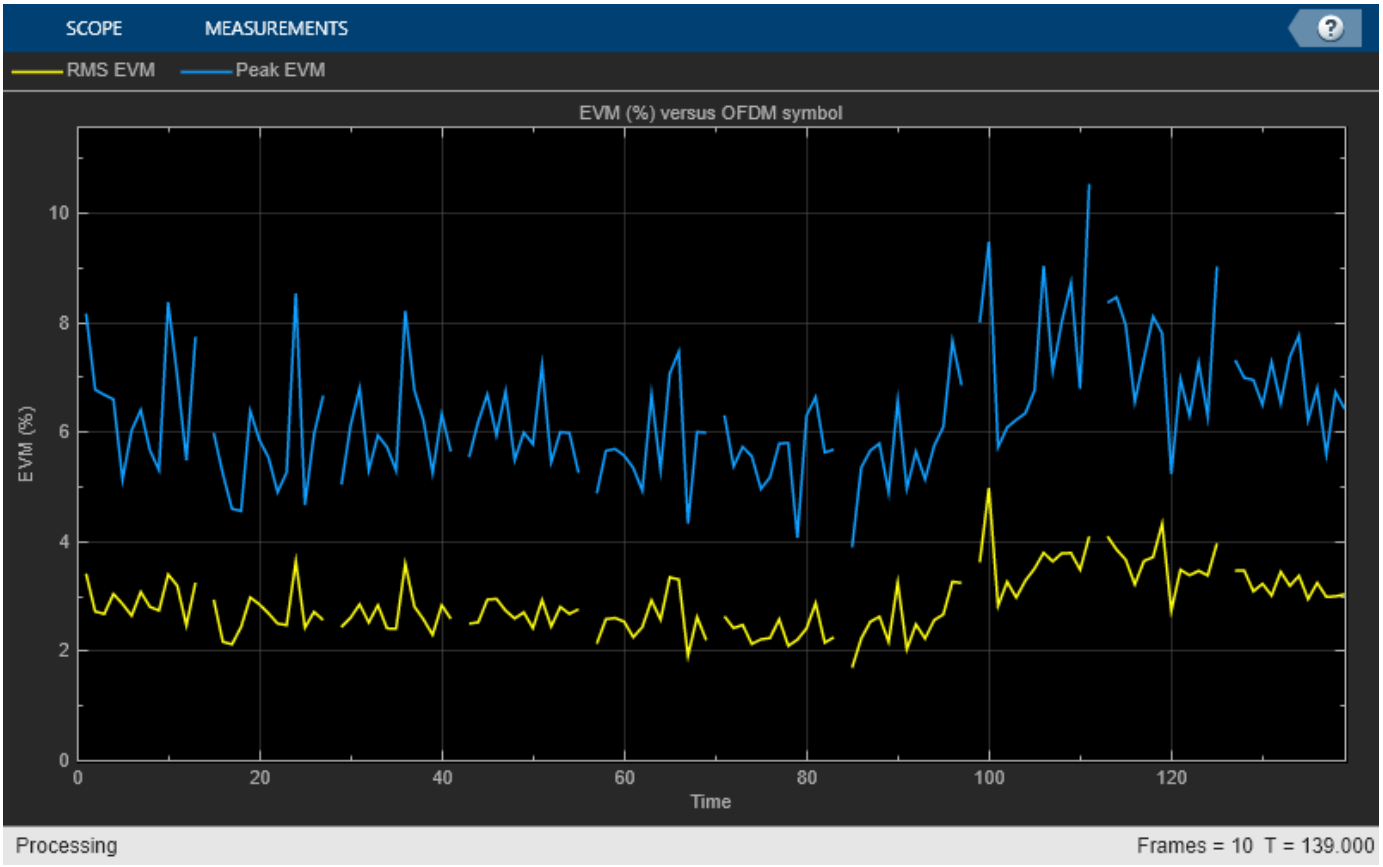
```
High edge EVM, subframe 7: 3.650%
Low edge EVM, subframe 8: 3.620%
High edge EVM, subframe 8: 3.618%
Low edge EVM, subframe 9: 3.188%
High edge EVM, subframe 9: 3.190%
Averaged low edge EVM, frame 0: 2.923%
Averaged high edge EVM, frame 0: 2.934%
Averaged EVM frame 0: 2.934%
Averaged overall EVM: 2.934%
```

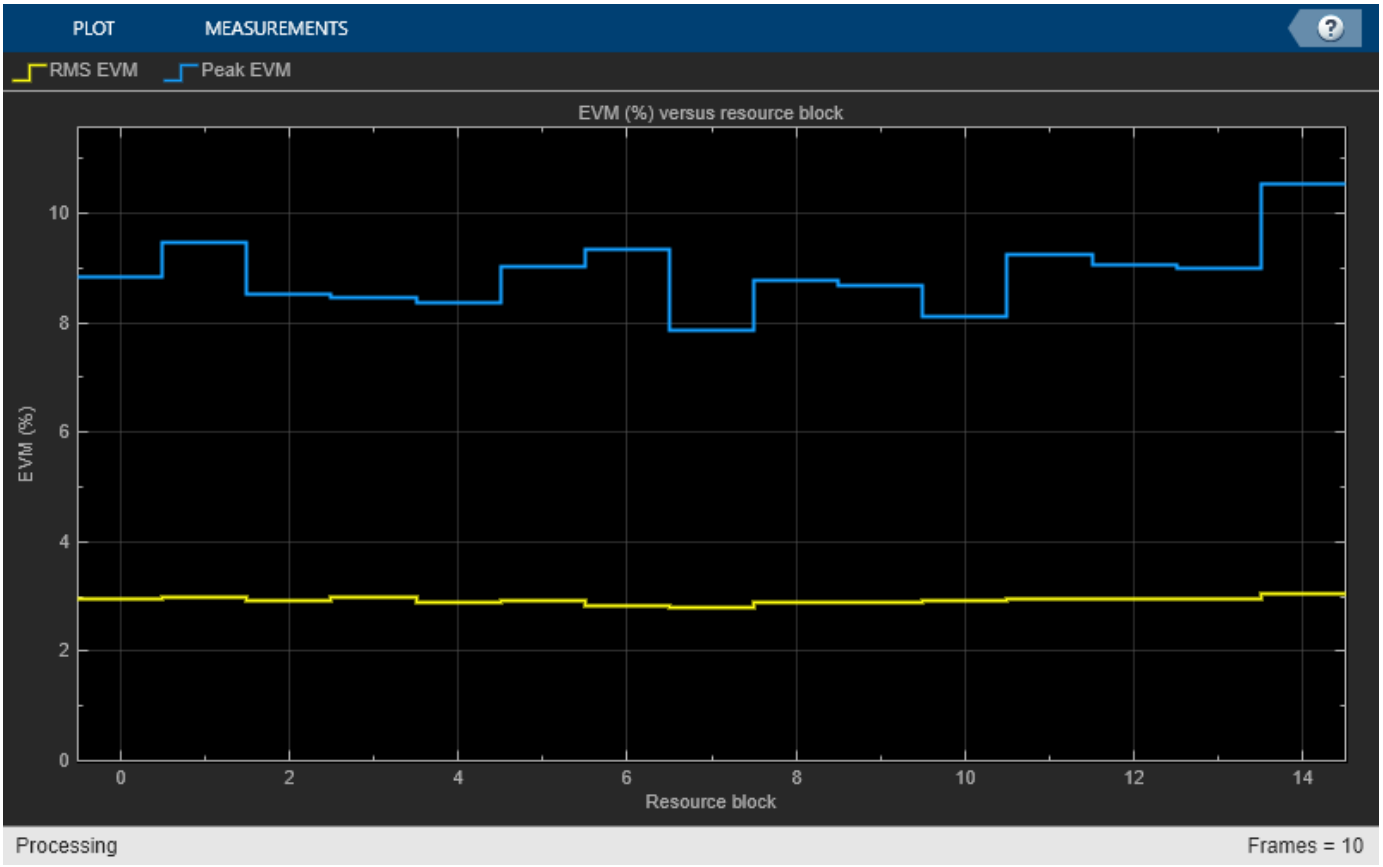


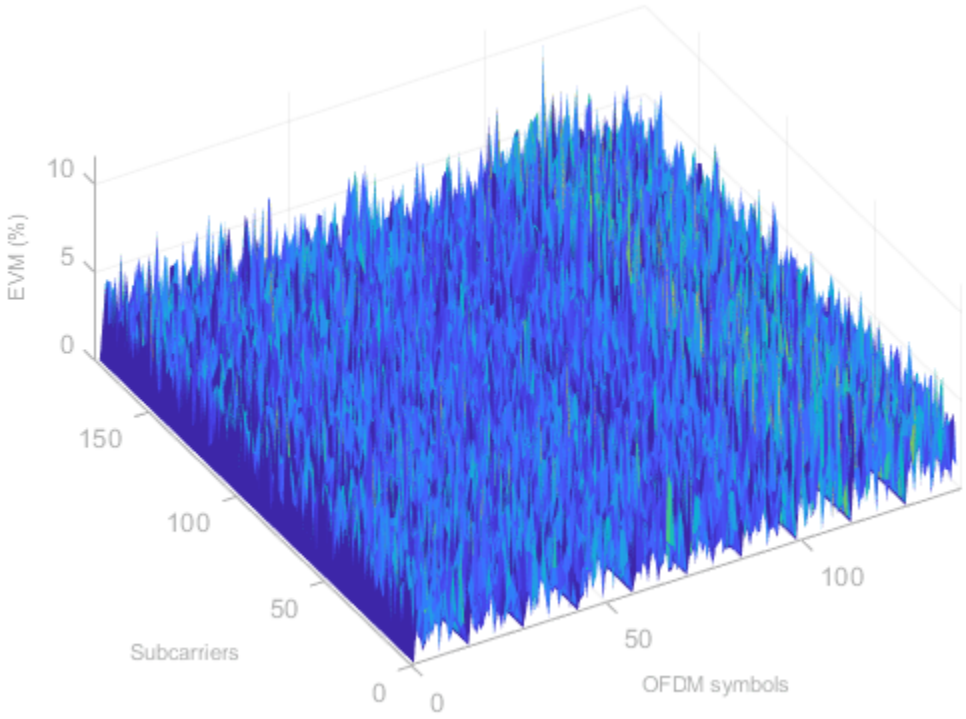

PLOT
MEASUREMENTS
1.5
1
0.5
0
-0.5
-1
-1.5
Quadrature Amplitude
-2
-1
0
1
2
In-phase Amplitude
Stopped


ANALYZER
ESTIMATION
MEASUREMENTS
SPECTRUM
SPECTRAL MASK
CHANNEL MEASU...
0
-20
-40
dBm
-1
0
1
Frequency (MHz)
Occupied Bandwidth
RF Transmitter/1
Channel Power (dBm)
Occupied Bandwidth (MHz)
Frequency Error (kHz)
21.8212
2.6755
-1.0882
Stopped
VBW = 3.00000 kHz RBW = 30.0000 kHz Sample Rate = 3.84000 MHz Frames = 7584 T = 0.00999974


PLOT
MEASUREMENTS
RMS EVM
Peak EVM
EVM (%) versus subcarrier
EVM (%)
Subcarrier
0
2
4
6
8
10
0
20
40
60
80
100
120
140
160
Processing
Frames = 10


SCOPE
MEASUREMENTS
RMS EVM
Peak EVM
EVM (%) versus OFDM symbol
EVM (%)
Time
0
2
4
6
8
10
0
20
40
60
80
100
120
Processing
Frames = 10 T = 139.000


PLOT
MEASUREMENTS
RMS EVM
Peak EVM
EVM (%) versus resource block
EVM (%)
Resource block
0
2
4
6
8
10
12
14
Processing
Frames = 10



Compared to the previous case, the constellation diagram is more distorted. In terms of measurements, the overall EVM, around 3%, still falls within the requirements of TS 36.104.

If you want to push the HPA to operate further in the nonlinear region, you will need to oversample the signal (around 5 times its baseband bandwidth) so that the simulation bandwidth is large enough to capture in-band spectral regrowth.

Summary and Further Exploration

This example demonstrates how to model and test an LTE RF transmitter in Simulink. The RF transmitter consists of an IQ modulator, a bandpass filter and amplifiers. To evaluate the performance, the Simulink model considers EVM measurements. The example highlights the effect of HPA nonlinearities on the performance of the RF transmitter. You can explore the impact of altering other impairments as well. For example:

- Increase I/Q imbalance by using the **I/Q gain mismatch (dB)** and **I/Q phase mismatch (Deg)** parameters on the **IQ Modulator** tab of the RF Transmitter Subsystem block.
- Increase the phase noise by using **Phase noise offset (Hz)** and **Phase noise level (dBc/Hz)** parameters on the **IQ Modulator** tab of the RF Transmitter Subsystem block.

The RF Transmitter Subsystem block is configured to work with the current E-TM waveform parameters selected in the LTE Test Model block and with the LTE carrier centered at 2140 MHz. This carrier is within the E-UTRA operating band 1. If you modify the **Center frequency (MHz)** parameter of the RF Transmitter Subsystem block or the waveform configuration of the LTE Test Model block, check if you need to update the parameters of the RF Transmitter components as these

parameters are set to work with the current example configuration. For instance, a change in the carrier frequency requires revising the **Passband frequencies** and **Stopband frequencies** parameters of the Bandpass Filter block inside the RF transmitter. If you select a bandwidth wider than 3 MHz, check if you need to update the **Impulse response duration** and **Phase noise frequency offset (Hz)** parameters of the IQ Modulator (RF Blockset) block. The phase noise offset determines the lower limit of the impulse response duration. If the phase noise frequency offset resolution is high for a given impulse response duration, a warning message appears, specifying the minimum duration suitable for the required resolution.

You can use this example as the basis for testing E-TM waveforms for different RF configurations. You can try replacing the RF Transmitter Subsystem block by another RF subsystem and then configure the model accordingly.

References

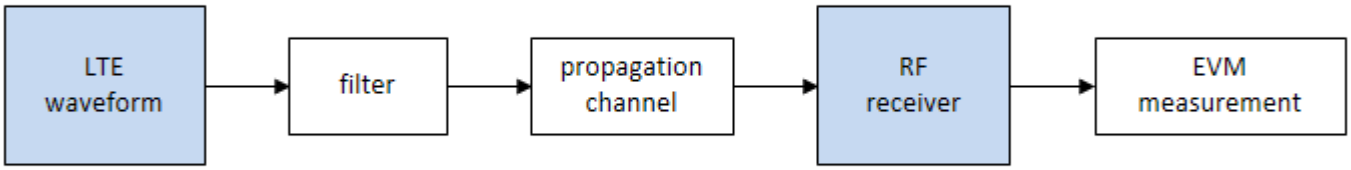
1. 3GPP TS 36.141 "E-UTRA; Base Station (BS) conformance testing" *3rd Generation Partnership Project; Technical Specification Group Radio Access Network.*
2. 3GPP TS 36.104 "E-UTRA; Base Station (BS) radio transmission and reception" *3rd Generation Partnership Project; Technical Specification Group Radio Access Network.*
3. 3GPP TS 36.101. "E-UTRA; User Equipment (UE) radio transmission and reception." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network.*

Modeling and Testing an LTE RF Receiver

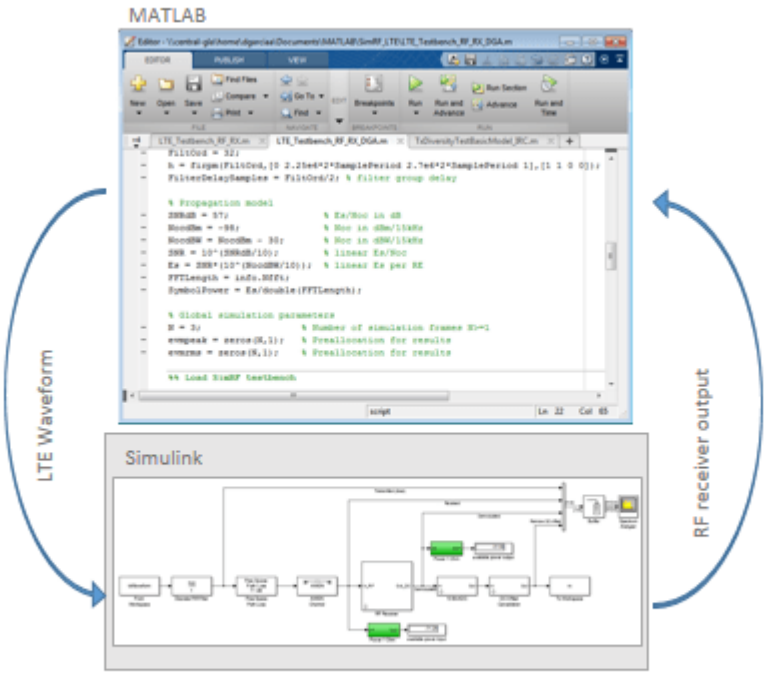
This example demonstrates how to model and test an LTE RF receiver using LTE Toolbox™ and RF Blockset™.

Model Description

The figure below shows the main parts of this example. An LTE waveform is generated using the LTE Toolbox. This waveform is filtered and transmitted through a propagation channel before feeding it to the RF receiver model implemented with RF Blockset. This model is based on commercially available parts. EVM figures are then provided for the output of the RF receiver.



This example is implemented using MATLAB® and Simulink®, which interact at runtime. The functional partition is shown in the figure below



The MATLAB script implements the simulation test-bench, and the Simulink model is the device under test (DUT). LTE frames are streamed between the test-bench and the DUT.

Generate LTE Waveform

In this section we generate the LTE waveform using the LTE Toolbox. We use the reference measurement channel (RMC) R.6 as defined in TS 36.101 [1]. This RMC specifies a 25 resource elements (REs) bandwidth, equivalent to 5 MHz. A 64 QAM modulation is used. All REs are allocated. Additionally, OCNG noise is enabled in unused REs.

Only one frame is generated. This frame will then be repeated a number of times to perform the EVM measurements.

```
% Configuration TS 36.101 25 REs (5 MHz), 64-QAM, full allocation
rmc = lteRMCDL('R.6');
rmc.OCNGPDSCHEnable = 'On';

% Create eNodeB transmission with fixed PDSCH data
rng(2);                 % Fixed random seed (arbitrary)
data = randi([0 1], sum(rmc.PDSCH.TrBlkSizes),1);

% Generate 1 frame, to be repeated to simulate a total of N frames
[tx, ~, info] = lteRMCDLTool(rmc, data); % 1 frame

% Calculate the sampling period and the length of the frame.
```

```
SamplePeriod = 1/info.SamplingRate;
FrameLength = length(tx);
```

Initialize Simulation Components

This section initializes some of the simulation components:

- Band limiting filter: design the filter coefficients, which will be used by the Simulink model. The filter has order 32, with passband frequency equal to 2.25 MHz, and stopband frequency equal to 2.7 MHz.
- SNR and signal energy
- Number of frames: this is the number of times the generated frame is repeated
- Preallocate result vectors

```
% Band limiting interpolation filter
FiltOrd = 32;
h = firpm(FiltOrd,[0 2.25e6*2*SamplePeriod 2.7e6*2*SamplePeriod 1],[1 1 0 0]);
FilterDelaySamples = FiltOrd/2; % filter group delay

% Propagation model
SNRdB = 57;                 % Es/Noc in dB
NocdBm = -98;               % Noc in dBm/15kHz
NocdBW = NocdBm - 30;       % Noc in dBW/15kHz57
SNR = 10^(SNRdB/10);        % linear Es/Noc
Es = SNR*(10^(NocdBW/10));  % linear Es per RE
FFTLength = info.Nfft;
SymbolPower = Es/double(FFTLength);

% Number of simulation frames N>=1
N = 3;

% Preallocate vectors for results for N-1 frames
% EVM is not measured in the first frame to avoid transient effects
evmpeak = zeros(N-1,1);   % Preallocation for results
evmrms = zeros(N-1,1);    % Preallocation for results
```

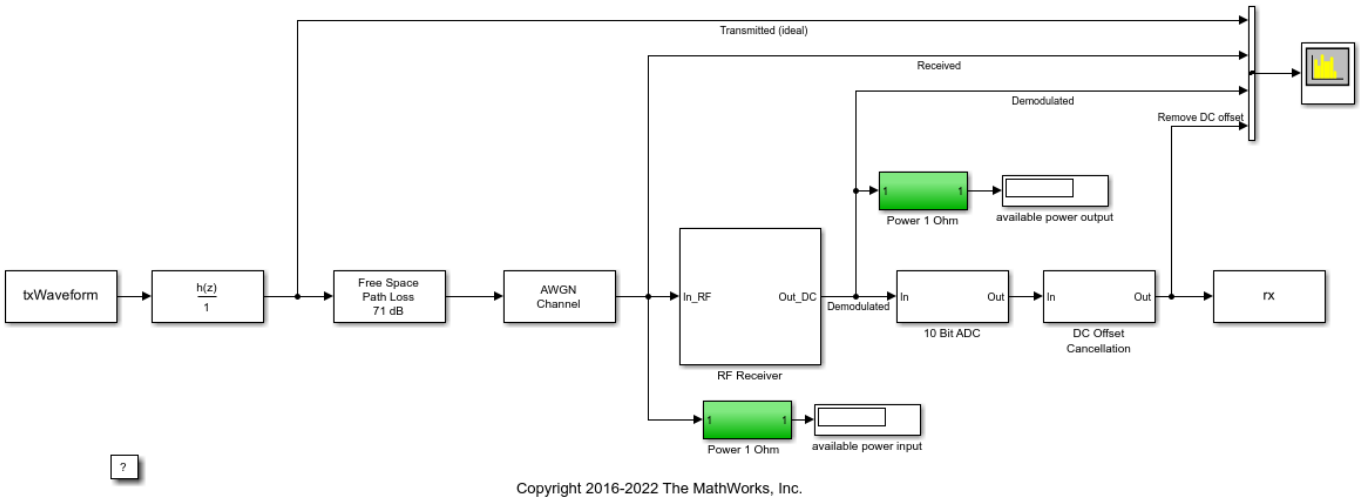
Load RF Blockset Testbench

This section loads the Simulink model shown below. This model includes the following components:

- Reading the LTE waveform and the sampling period from the workspace
- Bandlimiting filtering
- Channel model: this includes free space path loss and AWGN
- RF receiver, including direct conversion demodulator
- ADC and DC offset cancellation
- Save results to workspace

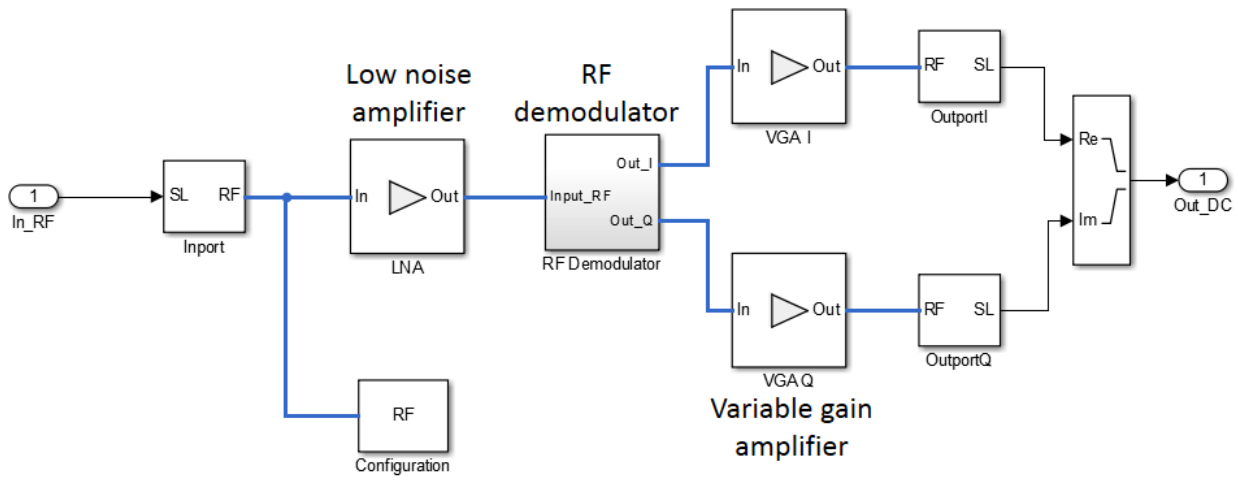
```
% Specify and open Simulink model
model = 'RFLTEReceiverModel';
disp('Starting Simulink');
open_system(model);
```

```
Starting Simulink
```



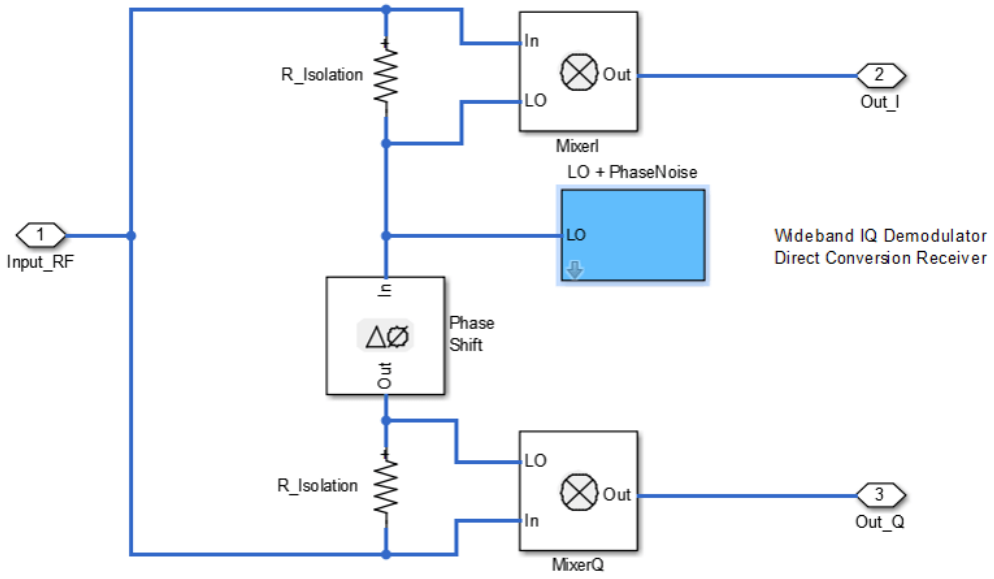
RF Receiver Model

The RF receiver model includes the elements shown below



The RF demodulator includes the following components and is shown below:

- Local oscillator (LO) and phase noise model
- Phase shift for I and Q components generator
- Mixers



Simulate Frames

This section simulates the specified number of frames. This is done in two stages:

- Simulate the first frame
- Simulate the rest of the frames in a loop

The reason for splitting the processing in these two stages is to simplify the code. During the processing of the first frame we need to take into account the delay of the band limiting filter. This is not the case for subsequent frames, since the filter state is maintained between frames. Therefore, the length of the first frame has to be increased slightly to take into account the delay introduced by the filter.

Simulate First LTE Frame

As mentioned for the first simulated frame we need to increase the length of the signal fed to the Simulink model to compensate for the delay introduced by the filter. Next we launch the simulation of the Simulink model without loading any initial state. After processing the first frame with the Simulink model, its state (`xFinal`) is stored and assigned to `xInitial` for the next time the model is run.

The output of the Simulink model is stored in the variable `rx`, which is available in the workspace. Any delays introduced to this signal are removed after performing synchronization. The EVM is measured on the resulting waveform.

```
% Generate test data for RF receiver
time = (0:FrameLength+FilterDelaySamples)*SamplePeriod;
% Append to the end of the frame enough samples to compensate for the delay
% of the filter
txWaveform = timeseries([tx; tx(1:FilterDelaySamples+1)],time);
```

```
% Simulate RF Blockset model of RF RX
set_param(model, 'LoadInitialState', 'off');
disp('Simulating LTE frame 1 ...');
sim(model, time(end));
% Save the final state of the model in xInitial for next frame processing
xInitial = xFinal;

% Synchronize to received waveform
Offset = lteDLFrameOffset(rmc,squeeze(rx),'TestEVM');
% In this case Offset = FilterDelaySamples therefore the following
% frames do not require synchronization
```

```
Simulating LTE frame 1 ...
```

Simulate Successive LTE Frames

Now the rest of the frames can be simulated. First, the model state is set using the value stored in `xInitial` at the output of the previous iteration.

```
% Load state after execution of previous frame. Since we are repeating the
% same frame the model state will be the same after every frame execution.
set_param(model, 'LoadInitialState', 'on', 'InitialState','xInitial');
% Modify input vector to take into account the delay of the bandlimiting
% filter
RepeatFrame = [tx(FilterDelaySamples+1:end); tx(1:FilterDelaySamples+1)];
EVMalg.EnablePlotting = 'Off';
cec.PilotAverage = 'TestEVM';

for n = 2:N % for all remaining frames
    % Generate data
    time = ( (n-1)*FrameLength+(0:FrameLength) + FilterDelaySamples)*SamplePeriod;
    txWaveform = timeseries(RepeatFrame,time);

    % Execute Simulink RF Blockset testbench
    disp(['Simulating LTE frame ',num2str(n),' ...']);
    sim(model, time(end));
    xInitial = xFinal; % Save model state

    % Compute and display EVM measurements
    evmmeas = hPDSCHEVM(rmc,cec,squeeze(rx),EVMalg);
    evmpeak(n-1) = evmmeas.Peak;
    evmrms(n-1) = evmmeas.RMS;
end
```

```
Simulating LTE frame 2 ...
Low edge EVM, subframe 0: 2.911%
High edge EVM, subframe 0: 2.923%
Low edge EVM, subframe 1: 2.685%
High edge EVM, subframe 1: 2.699%
Low edge EVM, subframe 2: 2.926%
High edge EVM, subframe 2: 2.934%
Low edge EVM, subframe 3: 3.435%
High edge EVM, subframe 3: 3.426%
Low edge EVM, subframe 4: 2.981%
High edge EVM, subframe 4: 2.995%
Low edge EVM, subframe 6: 3.125%
High edge EVM, subframe 6: 3.121%
Low edge EVM, subframe 7: 3.145%
```

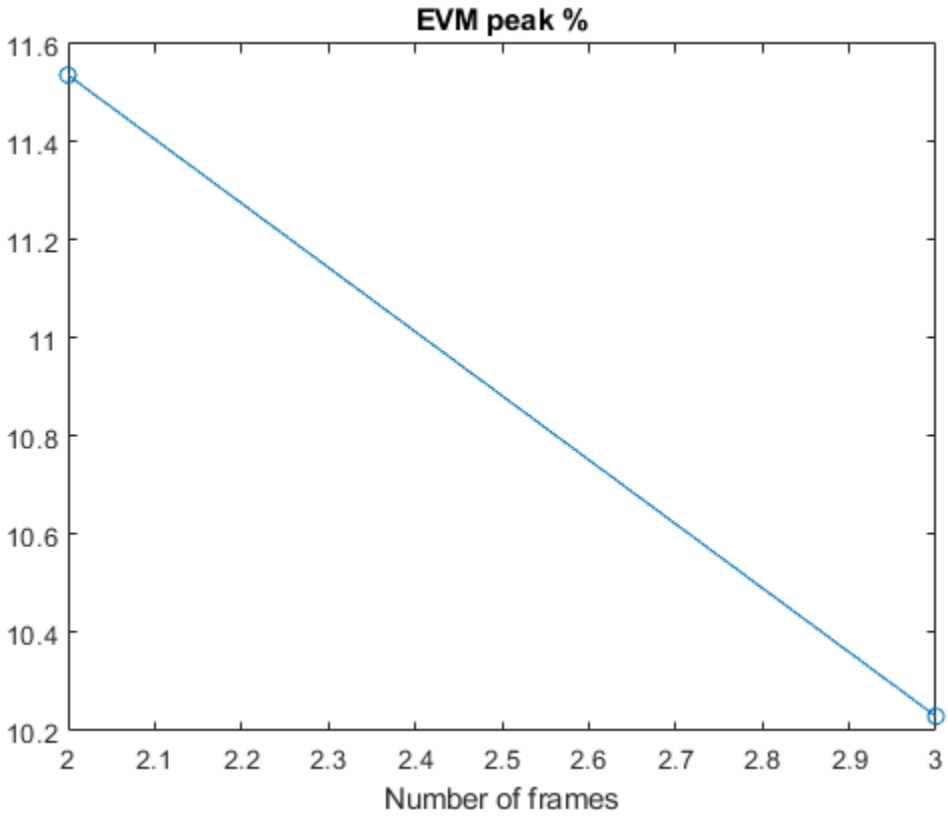
```
High edge EVM, subframe 7: 3.147%
Low edge EVM, subframe 8: 3.224%
High edge EVM, subframe 8: 3.224%
Low edge EVM, subframe 9: 3.231%
High edge EVM, subframe 9: 3.229%
Averaged low edge EVM, frame 0: 3.083%
Averaged high edge EVM, frame 0: 3.087%
Averaged EVM frame 0: 3.087%
Averaged overall EVM: 3.087%
Simulating LTE frame 3 ...
Low edge EVM, subframe 0: 2.899%
High edge EVM, subframe 0: 2.906%
Low edge EVM, subframe 1: 2.698%
High edge EVM, subframe 1: 2.706%
Low edge EVM, subframe 2: 3.005%
High edge EVM, subframe 2: 3.009%
Low edge EVM, subframe 3: 3.462%
High edge EVM, subframe 3: 3.433%
Low edge EVM, subframe 4: 2.913%
High edge EVM, subframe 4: 2.920%
Low edge EVM, subframe 6: 3.109%
High edge EVM, subframe 6: 3.108%
Low edge EVM, subframe 7: 3.171%
High edge EVM, subframe 7: 3.169%
Low edge EVM, subframe 8: 3.142%
High edge EVM, subframe 8: 3.135%
Low edge EVM, subframe 9: 3.135%
High edge EVM, subframe 9: 3.130%
Averaged low edge EVM, frame 0: 3.068%
Averaged high edge EVM, frame 0: 3.066%
Averaged EVM frame 0: 3.068%
Averaged overall EVM: 3.068%
```

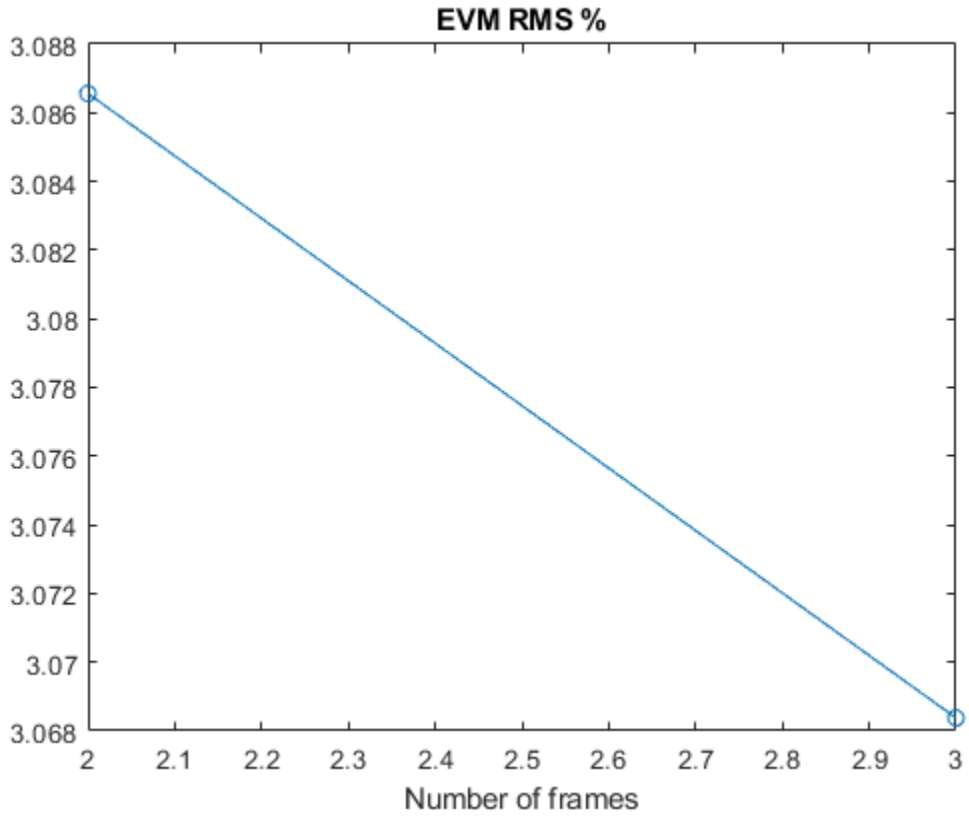
According to TS 36.104 [2], the maximum EVM when the constellation is 64-QAM is 8%. As the overall EVM, around 3%, is lower than 8%, this measurement falls within the requirements of TS 36.104 [2].

Visualize Measured EVM

This section plots the measured peak and RMS EVM for each simulated frame.

```
figure;
plot((2:N), 100*evmpeak,'o-')
title('EVM peak %');
xlabel('Number of frames');
figure;
plot((2:N), 100*evmrms,'o-');
title('EVM RMS %');
xlabel('Number of frames');
```


EVM peak %
11.6
11.4
11.2
11
10.8
10.6
10.4
10.2
2
2.1
2.2
2.3
2.4
2.5
2.6
2.7
2.8
2.9
3
Number of frames



Cleaning Up

Close the Simulink model and remove the generated files.

```
bdclose(model);
clear([model,'_acc']);
```

Appendix

This example uses the following helper functions:

- hPDSCHEVM.m

Selected Bibliography

1 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception"

2 3GPP TS 36.104 "E-UTRA; Base Station (BS) radio transmission and reception" *3rd Generation Partnership Project; Technical Specification Group Radio Access Network.*

Generate Wireless Waveform in Simulink Using App-Generated Block

This example shows how to configure and use the block that is generated using the **Export to Simulink** capability that is available in the **Wireless Waveform Generator** app.

Introduction

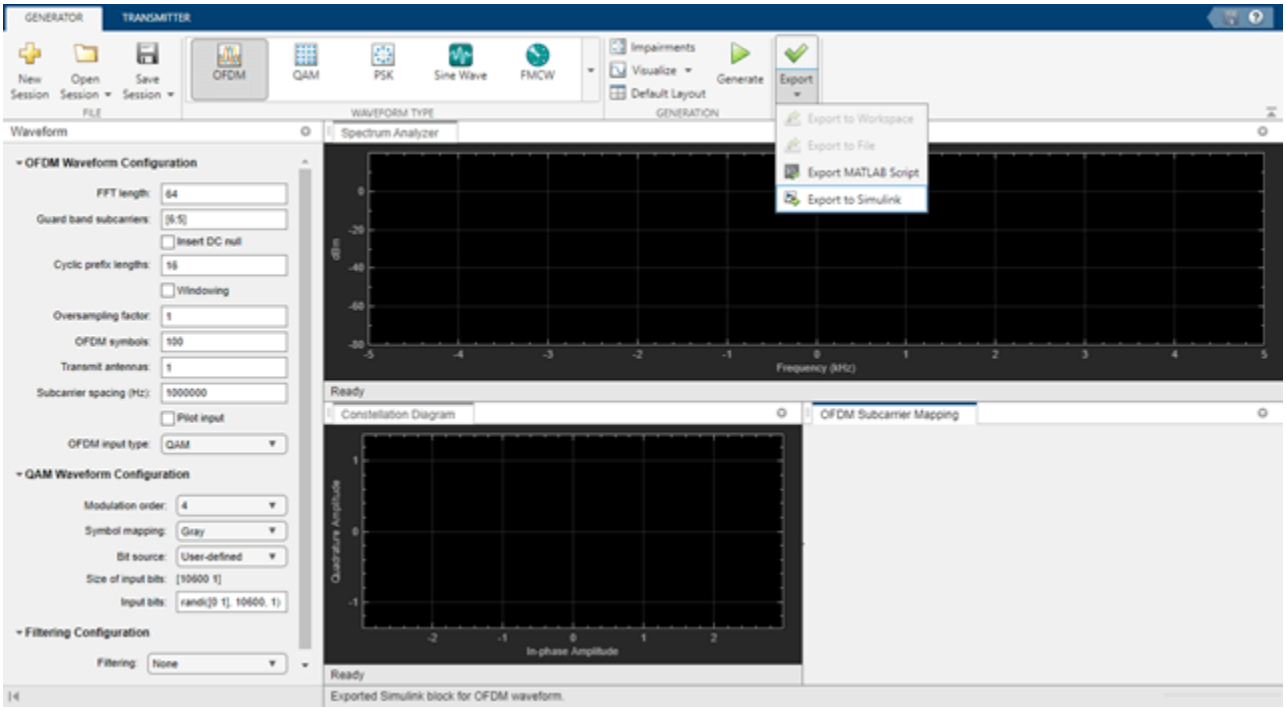
The **Wireless Waveform Generator** app is an interactive tool for creating, impairing, visualizing, and exporting waveforms. You can export the waveform to your workspace or to a `.mat` or `.bb` file. You can also export the waveform generation parameters to a runnable MATLAB® script or a Simulink® block. You can use the exported Simulink block to reproduce your waveform in Simulink. This example shows how to use the **Export to Simulink** capability of the app and how to configure the exported block to generate waveforms in Simulink.

Although this example focuses on exporting an OFDM waveform, the same process applies for all of the supported waveform types.

Export Wireless Waveform Configuration to Simulink

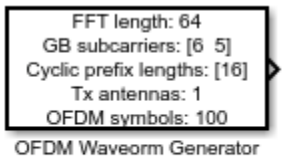
Open the **Wireless Waveform Generator** app by clicking the app icon on the **Apps** tab, under **Signal Processing and Communications**. Alternatively, enter `wirelessWaveformGenerator` at the MATLAB command prompt.

In the **Waveform Type** section, select an OFDM waveform by clicking **OFDM**. In the left-most pane of the app, adjust any configuration parameters for the selected waveform. Then export the configuration by clicking **Export** in the app toolstrip and selecting **Export to Simulink**.



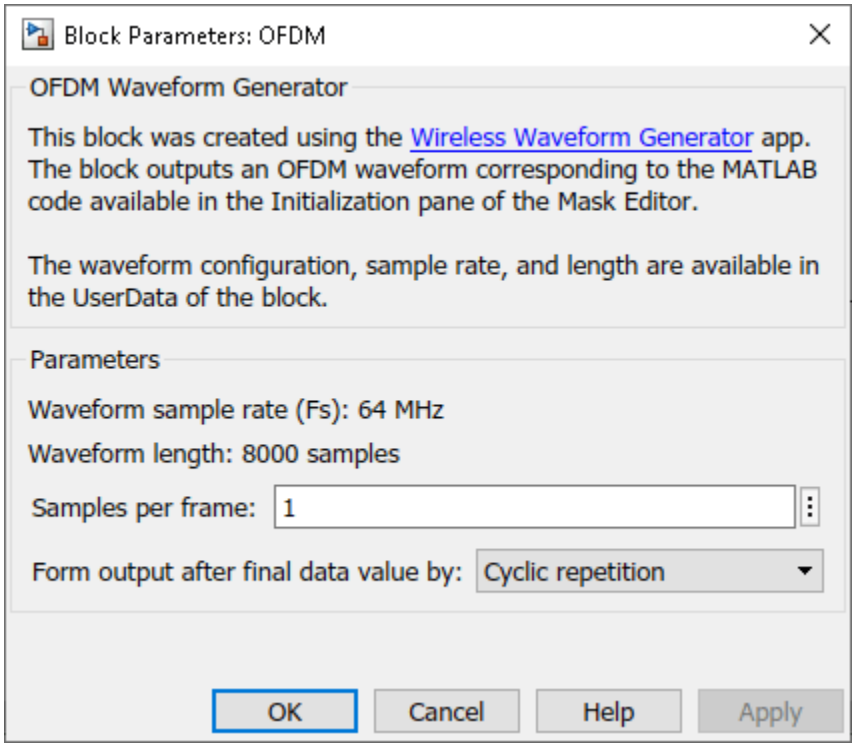
The **Export to Simulink** option creates a Simulink block, which outputs the selected waveform when you run the Simulink model. The block is exported to a new model if no open models exist.

```
modelName = 'WWGExport2SimulinkBlock';
open_system(modelName);
```




```
% Copyright 2021-2023 The MathWorks, Inc.
```

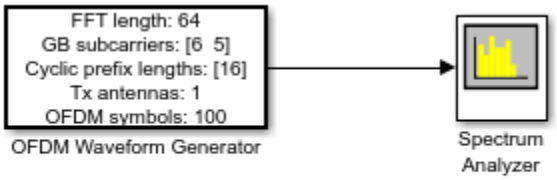
The **Form output after final data value by** block parameter specifies the output after all of the specified signal samples are generated. The value options for this parameter are `Cyclic repetition` and `Setting to zero`. The `Cyclic repetition` option repeats the signal from the beginning after it reaches the last sample in the signal. The `Setting to zero` option generates zero-valued outputs for the duration of the simulation after generating the last frame of the signal. The **Waveform sample rate (Fs)** and **Waveform length** block parameters are derived from the waveform configuration that is available in the **Code** tab of the Mask Editor dialog box. For further information about the block parameters, see Waveform From Wireless Waveform Generator App. This figure shows the parameters of the exported block.




```
close_system(modelName);
```

Connect a Spectrum Analyzer block to the exported block.

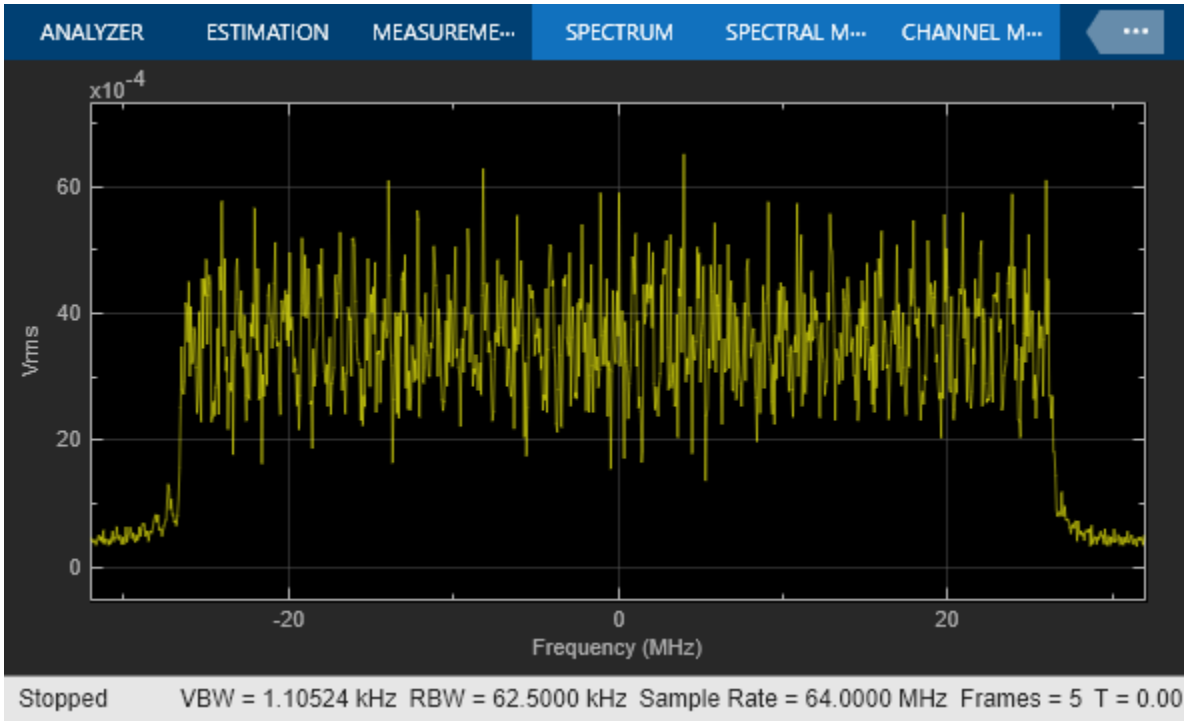
```
modelName = 'WWGExport2SimulinkModel';
open_system(modelName);
```




```
% Copyright 2021-2023 The MathWorks, Inc.
```

Simulate the model to visualize the waveform using the current configuration.

```
sim(modelName);
```



The Spectrum Analyzer block inherits the **Waveform sample rate (Fs)** parameter, which is 64 MHz.

```
close_system(modelName);
```

Modify Wireless Waveform Configuration

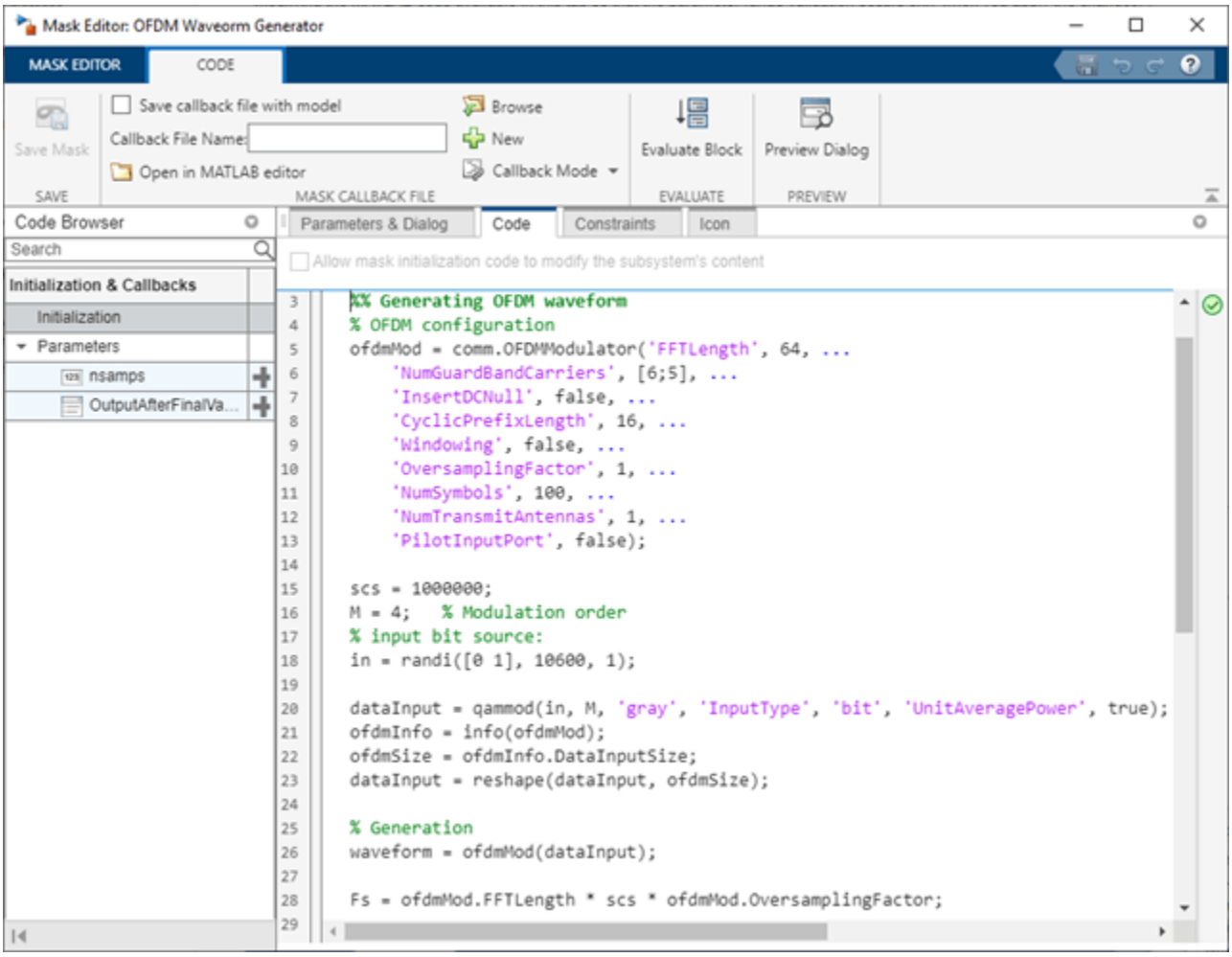
When you run the Simulink model, the exported block outputs the waveform generated in the **Code** tab of the Mask Editor dialog box for the block. The MATLAB code that initializes the waveform in this tab corresponds to the configuration that you selected in the **Wireless Waveform Generator** app before exporting the block. To modify the configuration of the waveform, choose one of these options:

- Open the **Wireless Waveform Generator** app, select the configuration of your choice, and export a new block. This option provides interaction with an app interface instead of MATLAB code,

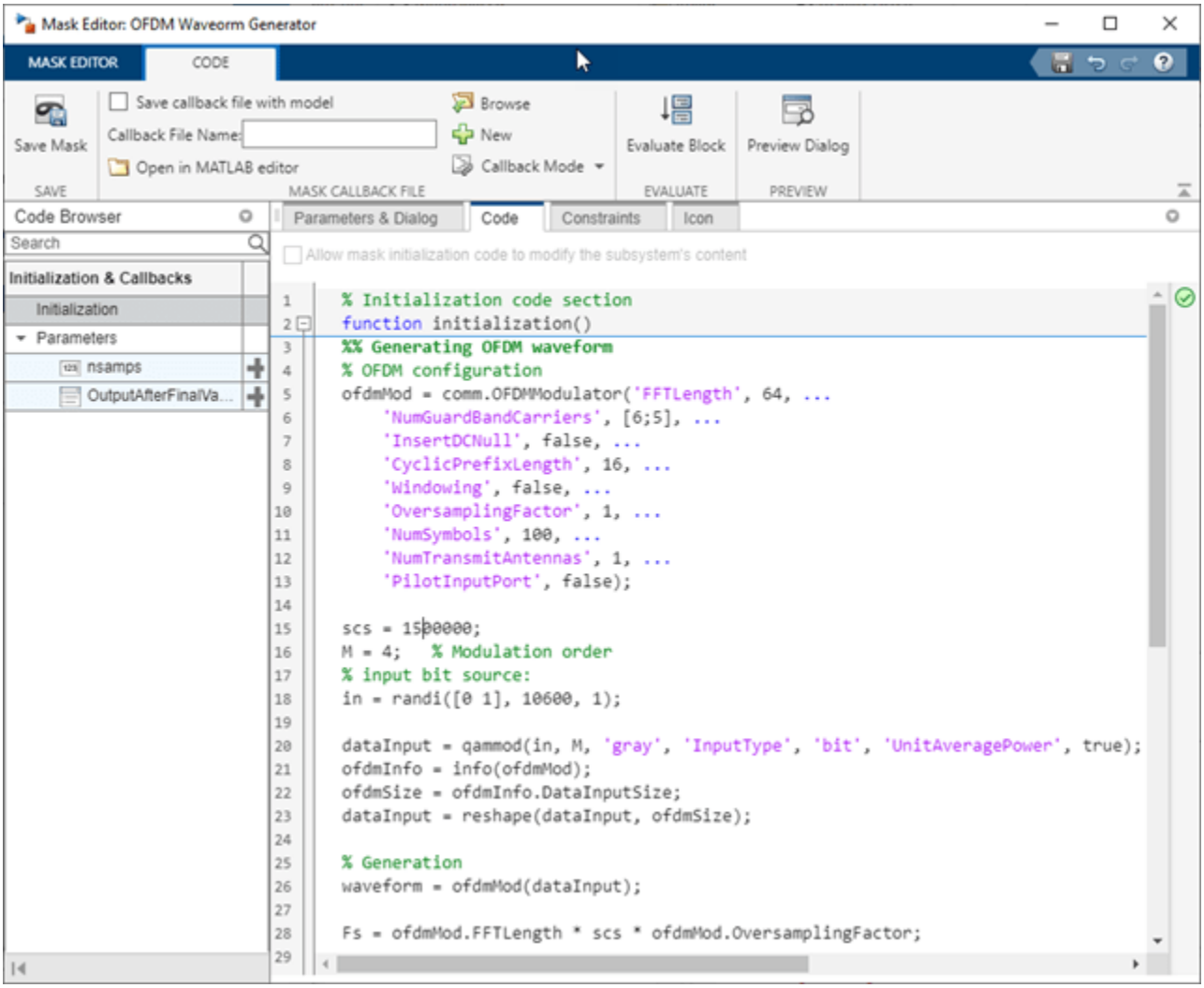
parameter range validation during the parameterization process, and visualization of the waveform before running the Simulink model.

- Update the configuration parameters that are available in the **Code** tab of the Mask Editor dialog box of the exported block. This option requires modifying the MATLAB code available in this tab so that the parameter range validation occurs only when you apply the changes. This option does not provide visualization of the waveform before running the Simulink model. Modifying the waveform parameters using this option is not recommended if you are not familiar with the MATLAB code that generates the selected waveform.

You can update the configuration in the **Code** tab of the Mask Editor. To open the Mask Editor, click the exported block and press **Ctrl+M**.

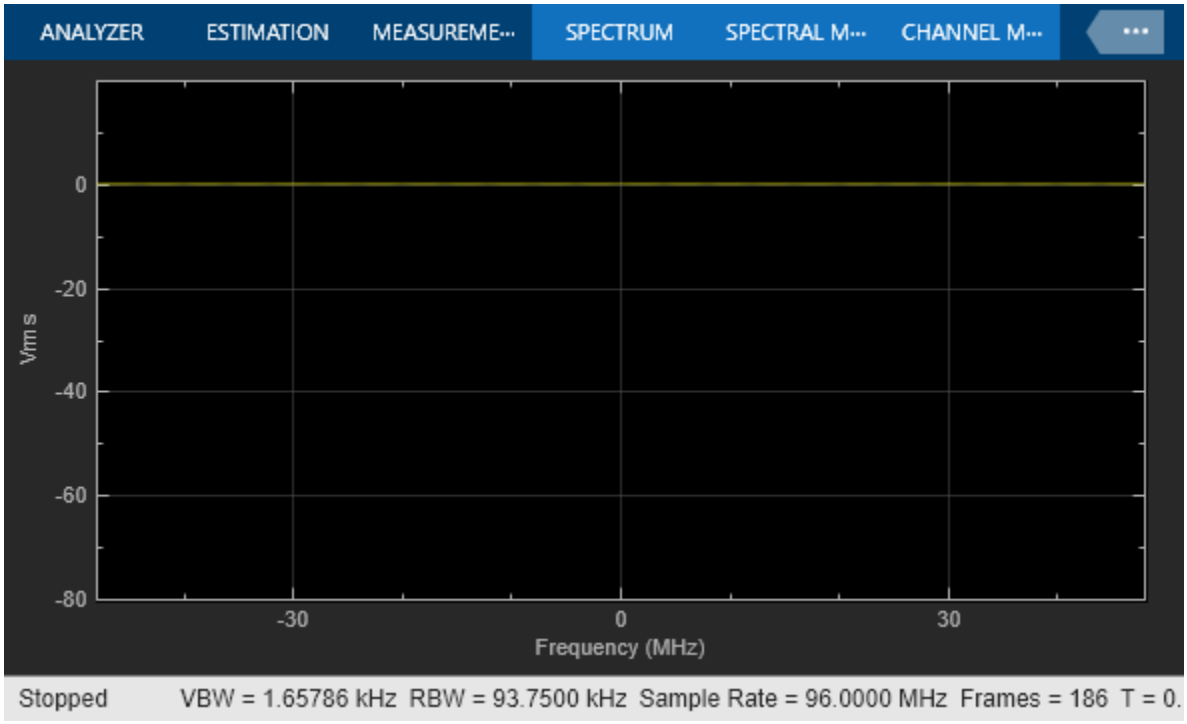


Use the MATLAB code that is available in the **Code** tab to update the parameters of your choice. For example, set the subcarrier spacing, `scs`, to 1,500,000 Hz.



Click **OK** to apply the changes and close the Mask Editor dialog box. Simulate the model to visualize the updated waveform.

```
modelName = 'WWGExport2SimulinkModelSCSModified';
sim(modelName);
```



The Spectrum Analyzer block now shows a sample rate of 96 MHz, which is 1.5 times the previous sample rate, as expected.

Share Wireless Waveform Configuration with Other Blocks in the Model

To access read-only block parameters and waveform configuration parameters, use the `UserData` common block property, which is a structure with these fields.

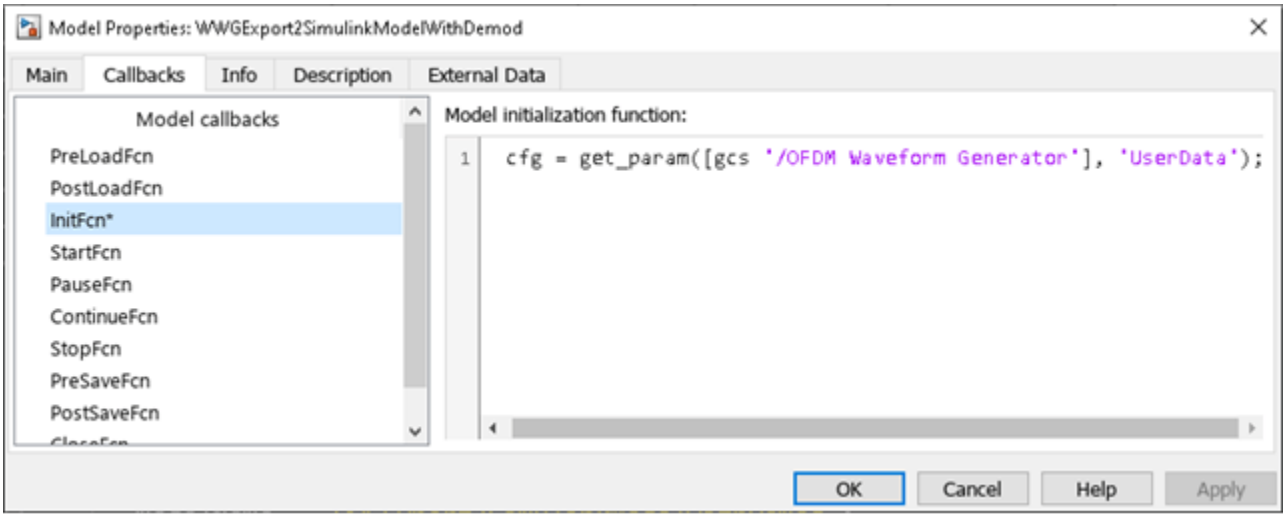
- `WaveformConfig`: Waveform configuration
- `WaveformLength`: Waveform length
- `Fs`: Waveform sample rate

You can access the user data of the exported block by using the `get_param` function.

```
get_param([gcs '/OFDM Waveform Generator'],'UserData')
```

```
ans =

  struct with fields:

    WaveformConfig: [1x1 comm.OFDMModulator]
    WaveformLength: 8000
                Fs: 96000000
```

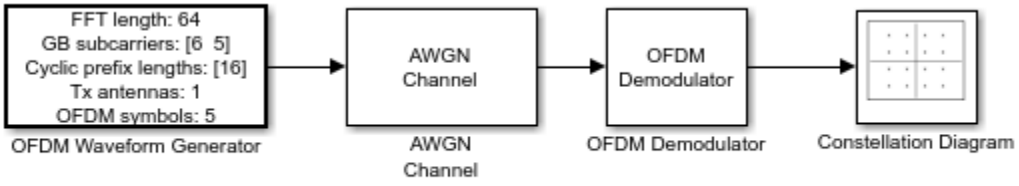
Store the structure available in the user data in a base workspace variable by using the `InitFcn` in the callback. The `InitFcn` callback is executed during a model update and simulation. To use this callback, click the **MODELING** tab, then click the **Model Settings** dropdown, and click the **Model Properties** option. In the **Callbacks** pane, select the `InitFcn` callback. Assign the user data to a new base workspace variable (for example, `cfg`).



The parameters that are available in the user data of the exported block are updated every time you apply configuration changes in the **Code** tab.

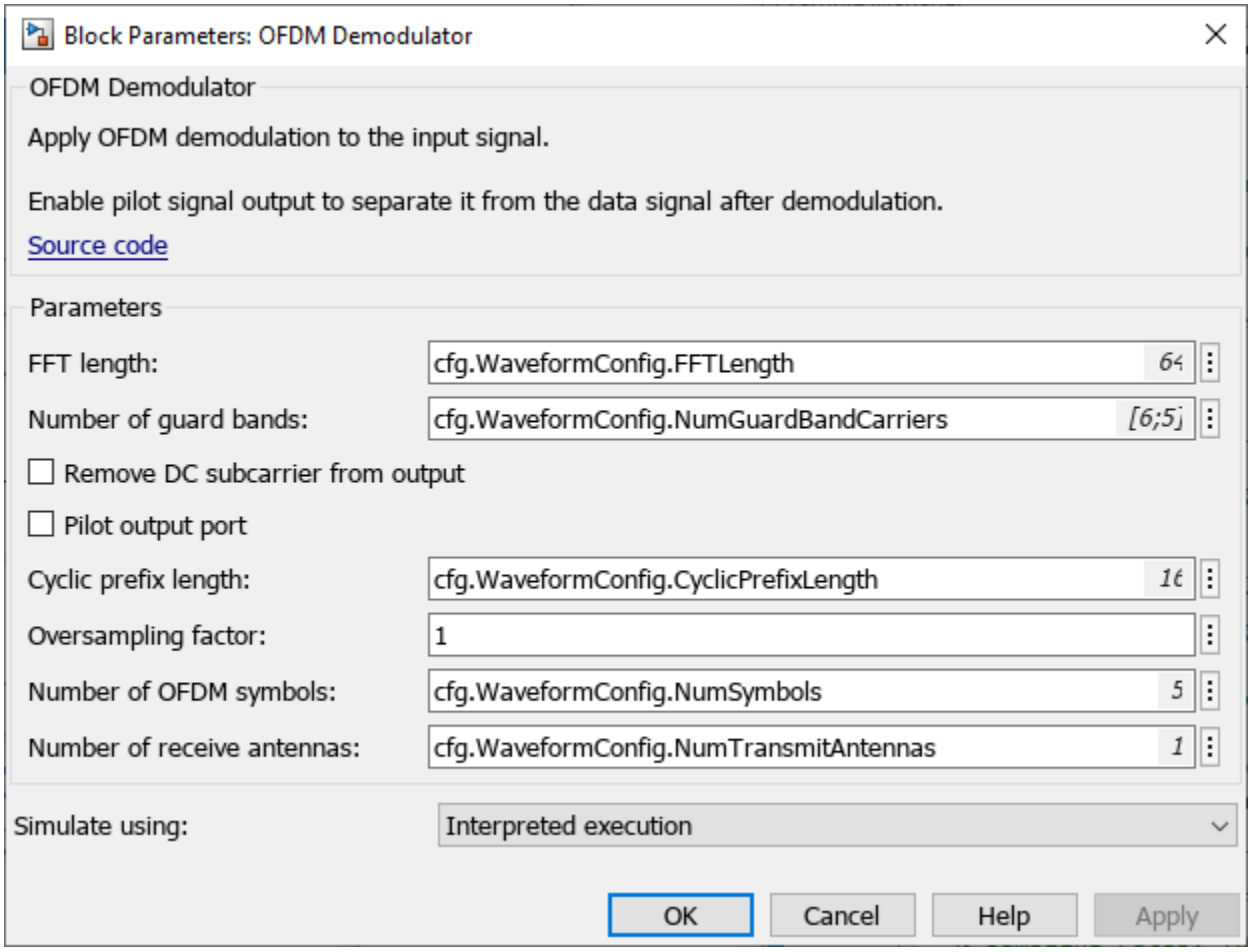
To demodulate the OFDM waveform, add an OFDM Demodulator block to the model. Connect an AWGN Channel block between the OFDM Waveform Generator and OFDM Demodulator blocks to add white Gaussian noise to the input signal. Also add a Constellation Diagram block to plot the demodulated symbols.

```
modelName = 'WWGExport2SimulinkModelWithDemod';
open_system(modelName);
```



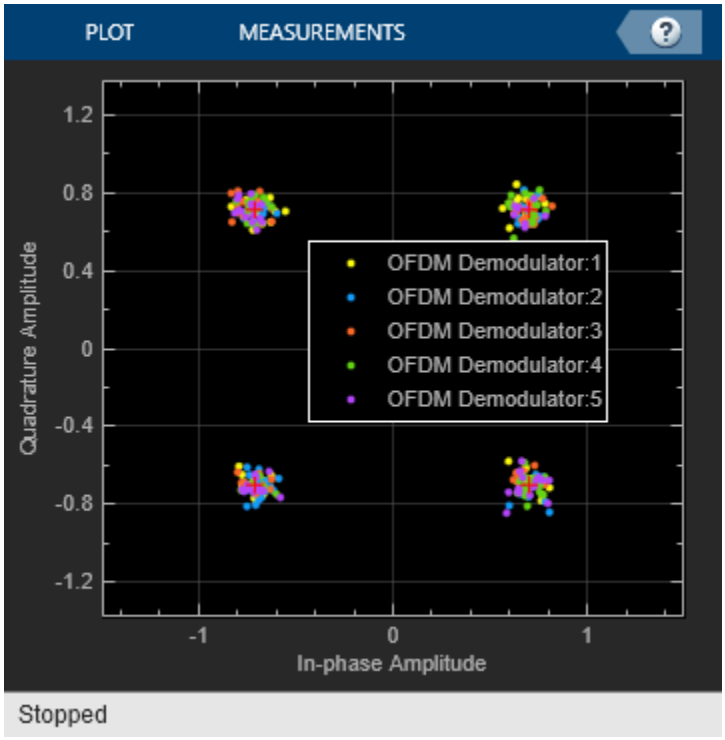

```
% Copyright 2021-2023 The MathWorks, Inc.
```

The parameters that are required to configure the OFDM Demodulator block must match the parameters that are used to configure the exported block, (otherwise, demodulation fails). To access the configuration parameters of the exported block, use the variable `cfg`. This figure shows the parameters of the OFDM Demodulator block.



Because the OFDM Demodulator block requires the entire OFDM waveform for demodulation, set the **Samples per frame** parameter in the exported block to `cfg.WaveformLength`. Simulate the model.

```
sim(modelName);
```

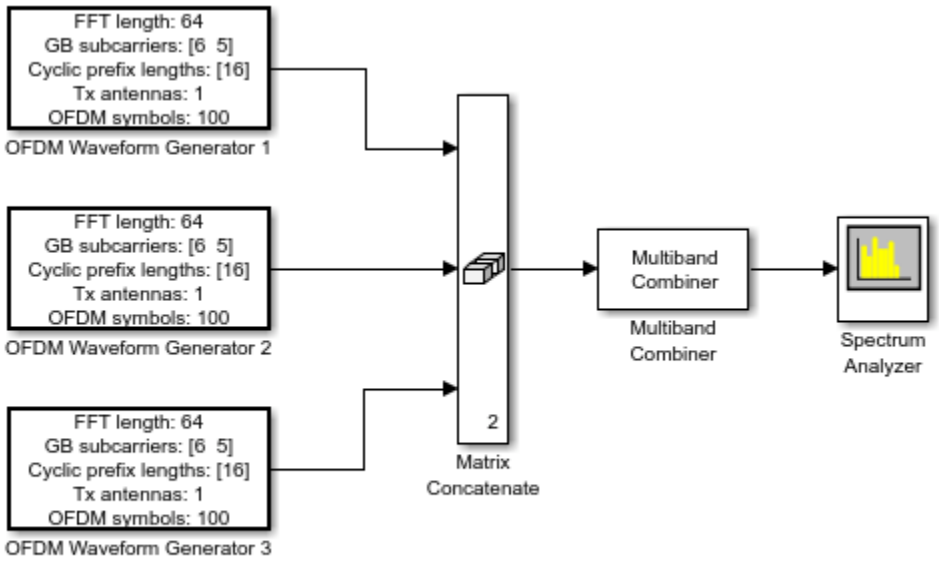


After demodulating the OFDM waveform by using the OFDM Demodulator block, the Constellation Diagram block displays the resulting QAM symbols.

Generate Multicarrier Waveforms

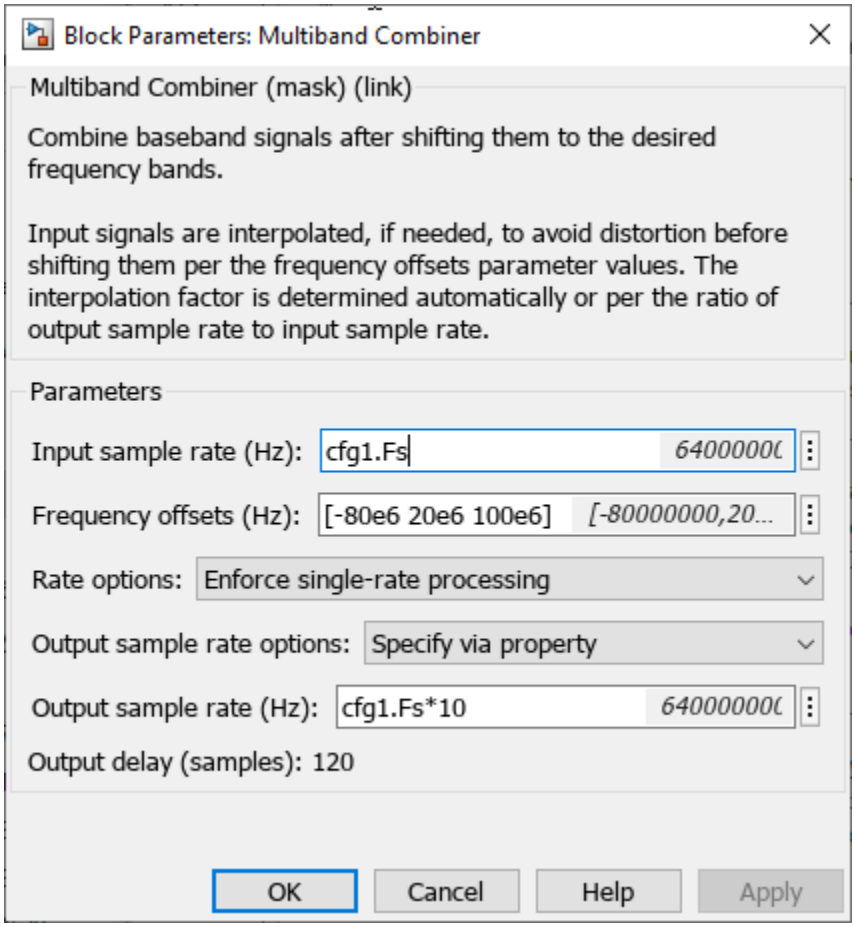
For multicarrier generation, the sampling rates for all of the waveforms must be the same. To shift the waveforms to a carrier offset and aggregate them, you can use the Multiband Combiner block.

```
modelName = 'WWGExport2SimulinkMulticarrier';
open_system(modelName);
```



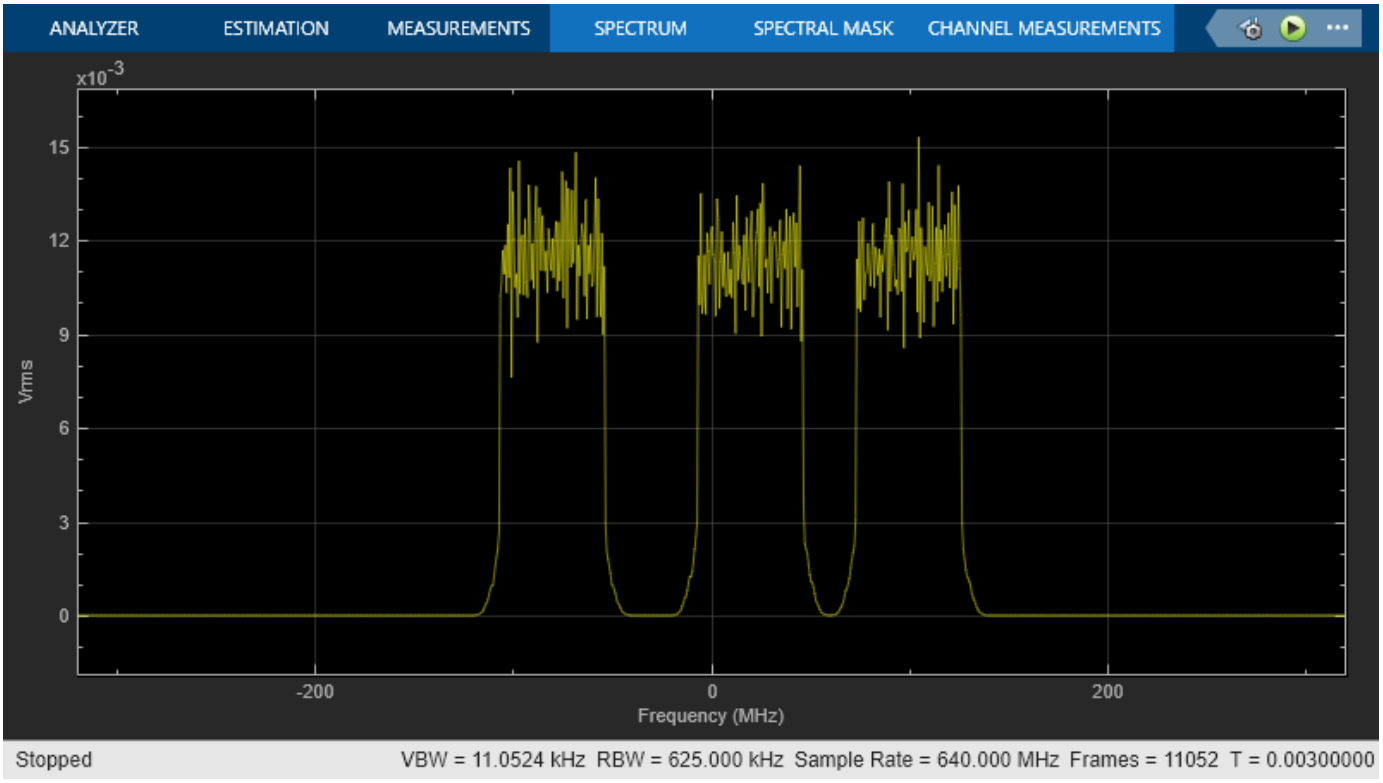

```
% Copyright 2021-2023 The MathWorks, Inc.
```

To shift the waveforms in frequency, you might have to increase the sampling rates. The Multiband Combiner block provides the option to oversample the input waveforms before shifting and combining them. This figure shows the parameters of the Multiband Combiner block.



Simulate the model to visualize the waveforms that are centered at -80, 20, and 100 MHz.

```
sim(modelName);
```



See Also

Blocks
Waveform From Wireless Waveform Generator App

Related Examples

- “Modeling and Testing an LTE RF Transmitter” on page 2-57
- “Modeling and Testing an LTE RF Receiver” on page 2-76

LTE Downlink Channel Estimation and Equalization

This example shows how to use the LTE Toolbox™ to create a frame worth of data, pass it through a fading channel and perform channel estimation and equalization. Two figures are created illustrating the received and equalized frame.

Introduction

This example shows how a simple transmitter-channel-receiver simulation may be created using functions from the LTE Toolbox. The example generates a frame worth of data on one antenna port. As no transport channel is created in this example the data is random bits, QPSK modulated and mapped to every symbol in a subframe. A cell specific reference signal and primary and secondary synchronization signals are created and mapped to the subframe. 10 subframes are individually generated to create a frame. The frame is OFDM modulated, passed through an Extended Vehicular A Model (EVA5) fading channel, additive white Gaussian noise added and demodulated. MMSE equalization using channel and noise estimation is applied and finally the received and equalized resource grids are plotted.

Cell-Wide Settings

The cell-wide settings are specified in a structure `enb`. A number of the functions used in this example require a subset of the settings specified below. In this example only one transmit antenna is used.

```
enb.NDLRB = 15;                 % Number of resource blocks
enb.CellRefP = 1;               % One transmit antenna port
enb.NCellID = 10;               % Cell ID
enb.CyclicPrefix = 'Normal';    % Normal cyclic prefix
enb.DuplexMode = 'FDD';         % FDD
```

SNR Configuration

The operating SNR is configured in decibels by the value `SNRdB` which is also converted into a linear SNR.

```
SNRdB = 22;             % Desired SNR in dB
SNR = 10^(SNRdB/20);    % Linear SNR
rng('default');         % Configure random number generators
```

Channel Model Configuration

The channel model is configured using a structure. In this example a fading channel with an Extended Vehicular A (EVA) delay profile and 120Hz Doppler frequency is used. These parameters along with MIMO correlation and other channel model specific parameters are set.

```
cfg.Seed = 1;                   % Channel seed
cfg.NRxAnts = 1;                % 1 receive antenna
cfg.DelayProfile = 'EVA';       % EVA delay spread
cfg.DopplerFreq = 120;          % 120Hz Doppler frequency
cfg.MIMOCorrelation = 'Low';    % Low (no) MIMO correlation
cfg.InitTime = 0;               % Initialize at time zero
cfg.NTerms = 16;                % Oscillators used in fading model
cfg.ModelType = 'GMEDS';        % Rayleigh fading model type
cfg.InitPhase = 'Random';       % Random initial phases
```

```
cfg.NormalizePathGains = 'On'; % Normalize delay profile power
cfg.NormalizeTxAnts = 'On';    % Normalize for transmit antennas
```

Channel Estimator Configuration

A user defined window is used to average pilot symbols to reduce the effect of noise. The averaging window size is configured in terms of resource elements (REs), in time and frequency. A conservative 9-by-9 window is used in this example as an EVA delay profile and 120Hz Doppler frequency cause the channel changes quickly over time and frequency. A 9-by-9 window includes the 4 pilots immediately surrounding the pilot of interest when averaging. Selecting an averaging window is discussed in “Channel Estimation” on page 1-115.

```
cec.PilotAverage = 'UserDefined'; % Pilot averaging method
cec.FreqWindow = 9;               % Frequency averaging window in REs
cec.TimeWindow = 9;               % Time averaging window in REs
```

Interpolation is performed by the channel estimator between pilot estimates to create a channel estimate for all REs. To improve the estimate multiple subframes can be used when interpolating. An interpolation window of 3 subframes with a centered interpolation window uses pilot estimates from 3 consecutive subframes to estimate the center subframe.

```
cec.InterpType = 'Cubic';         % Cubic interpolation
cec.InterpWinSize = 3;            % Interpolate up to 3 subframes
                                  % simultaneously
cec.InterpWindow = 'Centred';     % Interpolation windowing method
```

Subframe Resource Grid Size

In this example it is useful to have access to the subframe resource grid dimensions. These are determined using `lteDLResourceGridSize`. This function returns an array containing the number of subcarriers, number of OFDM symbols and number of transmit antenna ports in that order.

```
gridsize = lteDLResourceGridSize(enb);
K = gridsize(1);    % Number of subcarriers
L = gridsize(2);    % Number of OFDM symbols in one subframe
P = gridsize(3);    % Number of transmit antenna ports
```

Transmit Resource Grid

An empty resource grid `txGrid` is created which will be populated with subframes.

```
txGrid = [];
```

Payload Data Generation

As no transport channel is used in this example the data sent over the channel will be random QPSK modulated symbols. A subframe worth of symbols is created so a symbol can be mapped to every resource element. Other signals required for transmission and reception will overwrite these symbols in the resource grid.

```
% Number of bits needed is size of resource grid (K*L*P) * number of bits
% per symbol (2 for QPSK)
numberOfBits = K*L*P*2;

% Create random bit stream
inputBits = randi([0 1], numberOfBits, 1);
```

```
% Modulate input bits
inputSym = lteSymbolModulate(inputBits,'QPSK');
```

Frame Generation

The frame will be created by generating individual subframes within a loop and appending each created subframe to the previous subframes. The collection of appended subframes are contained within `txGrid`. This appending is repeated ten times to create a frame. When the OFDM modulated time domain waveform is passed through a channel the waveform will experience a delay. To avoid any samples being missed due to this delay an extra subframe is generated, therefore 11 subframes are generated in total. For each subframe the Cell-Specific Reference Signal (Cell RS) is added. The Primary Synchronization Signal (PSS) and Secondary Synchronization Signal (SSS) are also added. Note that these synchronization signals only occur in subframes 0 and 5, but the LTE Toolbox takes care of generating empty signals and indices in the other subframes so that the calling syntax here can be completely uniform across the subframes.

```
% For all subframes within the frame
for sf = 0:10

    % Set subframe number
    enb.NSubframe = mod(sf,10);

    % Generate empty subframe
    subframe = lteDLResourceGrid(enb);

    % Map input symbols to grid
    subframe(:) = inputSym;

    % Generate synchronizing signals
    pssSym = ltePSS(enb);
    sssSym = lteSSS(enb);
    pssInd = ltePSSIndices(enb);
    sssInd = lteSSSIndices(enb);

    % Map synchronizing signals to the grid
    subframe(pssInd) = pssSym;
    subframe(sssInd) = sssSym;

    % Generate cell specific reference signal symbols and indices
    cellRsSym = lteCellRS(enb);
    cellRsInd = lteCellRSIndices(enb);

    % Map cell specific reference signal to grid
    subframe(cellRsInd) = cellRsSym;

    % Append subframe to grid to be transmitted
    txGrid = [txGrid subframe]; %#ok

end
```

OFDM Modulation

In order to transform the frequency domain OFDM symbols into the time domain, OFDM modulation is required. This is achieved using `lteOFDMModulate`. The function returns two values; a matrix `txWaveform` and a structure `info` containing the sampling rate. `txWaveform` is the resulting time domain waveform. Each column contains the time domain signal for each antenna port. In this example, as only one antenna port is used, only one column is returned. `info.SamplingRate` is the

sampling rate at which the time domain waveform was created. This value is required by the channel model.

```
[txWaveform,info] = lteOFDMModulate(enb,txGrid);
txGrid = txGrid(:,1:140);
```

Fading Channel

The time domain waveform is passed through the channel model (`lteFadingChannel`) configured by the structure `cfg`. The channel model requires the sampling rate of the time domain waveform so the parameter `cfg.SamplingRate` is set to the value returned by `lteOFDMModulate`. The waveform generated by the channel model function contains one column per receive antenna. In this example one receive antenna is used, therefore the returned waveform has one column.

```
cfg.SamplingRate = info.SamplingRate;

% Pass data through the fading channel model
rxWaveform = lteFadingChannel(cfg,txWaveform);
```

Additive Noise

The SNR is given by SNR = E_s/N_0 where E_s is the energy of the signal of interest and N_0 is the noise power. The noise added before OFDM demodulation will be amplified by the FFT. Therefore to normalize the SNR at the receiver (after OFDM demodulation) the noise must be scaled. The amplification is the square root of the size of the FFT. The size of the FFT can be determined from the sampling rate of the time domain waveform (`info.SamplingRate`) and the subcarrier spacing (15 kHz). The power of the noise to be added can be scaled so that E_s and N_0 are normalized after the OFDM demodulation to achieve the desired SNR (`SNRdB`).

```
% Calculate noise gain
N0 = 1/(sqrt(2.0*enb.CellRefP*double(info.Nfft))*SNR);

% Create additive white Gaussian noise
noise = N0*complex(randn(size(rxWaveform)),randn(size(rxWaveform)));

% Add noise to the received time domain waveform
rxWaveform = rxWaveform + noise;
```

Synchronization

The offset caused by the channel in the received time domain signal is obtained using `lteDLFrameOffset`. This function returns a value `offset` which indicates how many samples the waveform has been delayed. The offset is considered identical for waveforms received on all antennas. The received time domain waveform can then be manipulated to remove the delay using `offset`.

```
offset = lteDLFrameOffset(enb,rxWaveform);
rxWaveform = rxWaveform(1+offset:end,:);
```

OFDM Demodulation

The time domain waveform undergoes OFDM demodulation to transform it to the frequency domain and recreate a resource grid. This is accomplished using `lteOFDMDemodulate`. The resulting grid is a 3-dimensional matrix. The number of rows represents the number of subcarriers. The number of columns equals the number of OFDM symbols in a subframe. The number of subcarriers and symbols is the same for the returned grid from OFDM demodulation as the grid passed into

`lteOFDMModulate`. The number of planes (3rd dimension) in the grid corresponds to the number of receive antennas.

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

Channel Estimation

To create an estimation of the channel over the duration of the transmitted resource grid `lteDLChannelEstimate` is used. The channel estimation function is configured by the structure `cec`. `lteDLChannelEstimate` assumes the first subframe within the resource grid is subframe number `enb.NSubframe` and therefore the subframe number must be set prior to calling the function. In this example the whole received frame will be estimated in one call and the first subframe within the frame is subframe number 0. The function returns a 4-D array of complex weights which the channel applies to each resource element in the transmitted grid for each possible transmit and receive antenna combination. The possible combinations are based upon the eNodeB configuration `enb` and the number of receive antennas (determined by the size of the received resource grid). The 1st dimension is the subcarrier, the 2nd dimension is the OFDM symbol, the 3rd dimension is the receive antenna and the 4th dimension is the transmit antenna. In this example one transmit and one receive antenna is used therefore the size of `estChannel` is 180-by-140-by-1-by-1.

```
enb.NSubframe = 0;
[estChannel, noiseEst] = lteDLChannelEstimate(enb,cec,rxGrid);
```

MMSE Equalization

The effects of the channel on the received resource grid are equalized using `lteEqualizeMMSE`. This function uses the estimate of the channel `estChannel` and noise `noiseEst` to equalize the received resource grid `rxGrid`. The function returns `eqGrid` which is the equalized grid. The dimensions of the equalized grid are the same as the original transmitted grid (`txGrid`) before OFDM modulation.

```
eqGrid = lteEqualizeMMSE(rxGrid, estChannel, noiseEst);
```

Analysis

The received resource grid is compared with the equalized resource grid. The error between the transmitted and equalized grid and transmitted and received grids are calculated. This creates two matrices (the same size as the resource arrays) which contain the error for each symbol. To allow easy inspection the received and equalized grids are plotted on a logarithmic scale using `surf` within `hDownlinkEstimationEqualizationResults.m`. These diagrams show that performing channel equalization drastically reduces the error in the received resource grid.

```
% Calculate error between transmitted and equalized grid
eqError = txGrid - eqGrid;
rxError = txGrid - rxGrid;

% Compute EVM across all input values
% EVM of pre-equalized receive signal
EVM = comm.EVM;
EVM.AveragingDimensions = [1 2];
preEqualisedEVM = EVM(txGrid,rxGrid);
fprintf('Percentage RMS EVM of Pre-Equalized signal: %0.3f%%\n', ...
        preEqualisedEVM);
```

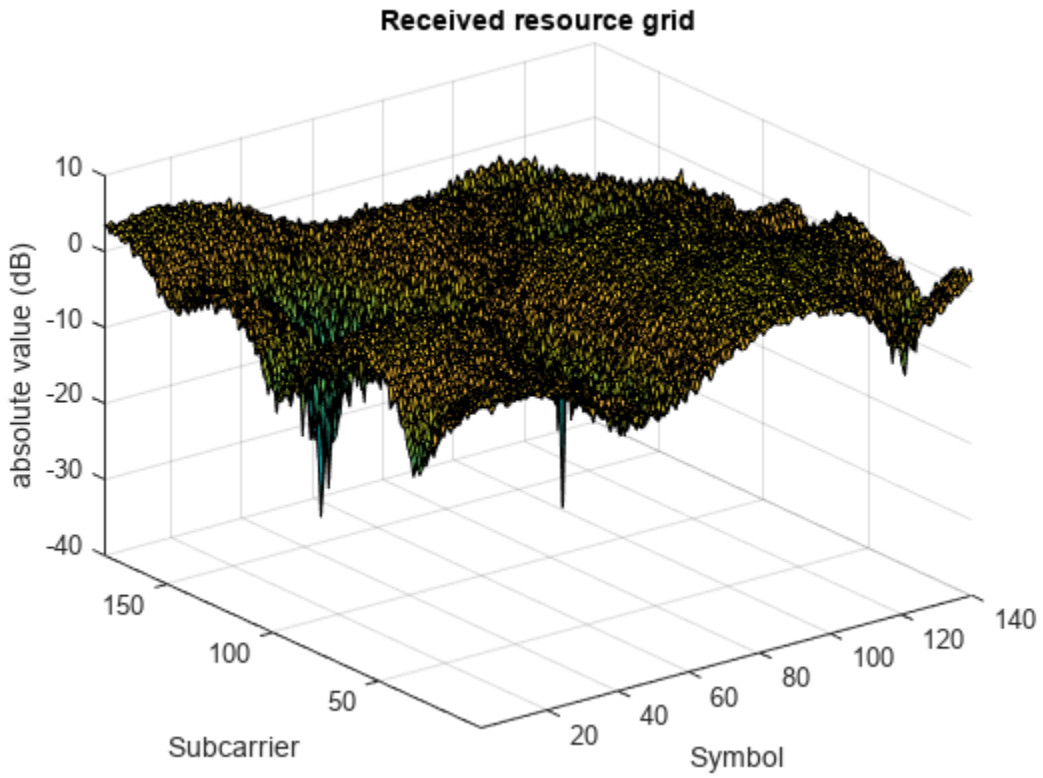
```
Percentage RMS EVM of Pre-Equalized signal: 124.133%
```

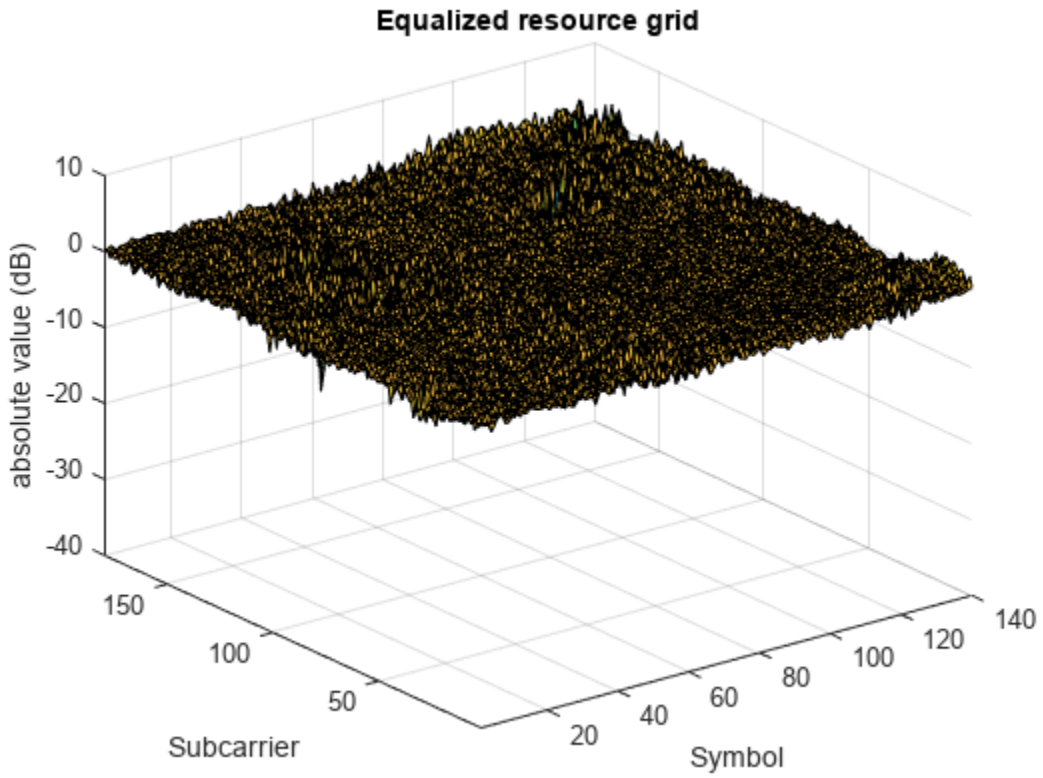
```
% EVM of post-equalized receive signal
postEqualisedEVM = EVM(txGrid,eqGrid);
```

```
fprintf('Percentage RMS EVM of Post-Equalized signal: %0.3f%%\n', ...
        postEqualisedEVM);
```

```
Percentage RMS EVM of Post-Equalized signal: 15.598%
```

```
% Plot the received and equalized resource grids
hDownlinkEstimationEqualizationResults(rxGrid, eqGrid);
```





Appendix

This example uses the helper function:

- hDownlinkEstimationEqualizationResults.m

NB-IoT Downlink Waveform Generation

This example shows how to generate LTE-Advanced Pro Release 13 narrowband IoT (NB-IoT) waveforms for test and measurement applications using LTE Toolbox™ software.

Introduction

3GPP introduced a new air interface, Narrowband IoT (NB-IoT) optimized for low data rate machine type communications in LTE-Advanced Pro Release 13. NB-IoT provides cost and power efficiency improvements as it avoids the need for complex signaling overhead required for LTE based systems.

The LTE Toolbox can be used to generate standard compliant NB-IoT downlink complex baseband waveforms representing the 180 kHz narrowband carrier suitable for test and measurement applications. The waveform consists of the individual physical layer channels and signals and the MATLAB value class NBIoTDownlinkWaveformGenerator can be used for full resource element (RE) grid and time-domain waveform generation. The LTE Toolbox supports all the NB-IoT modes of operation - standalone, guardband and in-band.

- Standalone: NB-IoT carrier deployed outside the LTE spectrum, e.g. the spectrum used for GSM or satellite communications
- Guardband: NB-IoT carrier deployed in the guardband between two LTE carriers
- In-band: NB-IoT carrier deployed in resource blocks of a LTE carrier

In-band mode can be further grouped depending on the physical cell identity (PCI) used, Inband-SamePCI and Inband-DifferentPCI. If Inband-SamePCI, the physical layer cell identity and PCI are the same and the UE can make assumptions about the ports and channel from LTE signals. The operation mode is indicated in the NB-IoT MIB (MIB-NB), which provides essential information for the user equipment (UE). The network can use radio resource control information to allocate a non-anchor carrier to an UE operating in an anchor carrier, see TS 36.331 section 6.7.3.2 [5] and section 7.3.2.4 in [9].

This example creates the NB-IoT downlink, where the relevant physical layer channels and signals are:

- Narrowband primary synchronization signal (NPSS)
- Narrowband secondary synchronization signal (NSSS)
- Narrowband reference signal (NRS)
- Narrowband physical broadcast channel (NPBCH)
- Narrowband physical downlink shared channel (NPDSCH)
- Narrowband physical downlink control channel (NPDCCH)

NB-IoT supports two carrier configurations:

- Anchor: The carrier used by the UE for initial NB-IoT cell selection, obtaining MIBs and system information blocks (SIBs), and random access in the idle mode. The NPSS, NSSS, NPBCH, and system information are transmitted on the carrier.
- Non-anchor: A carrier only used for actual exchange of data in the connected mode. NPSS, NSSS, NPBCH, and system information are not transmitted on the carrier.

In this example, we demonstrate the NB-IoT downlink RE grid and waveform generation. The sections below explain the physical signals and channels that form the grid along with key concepts including

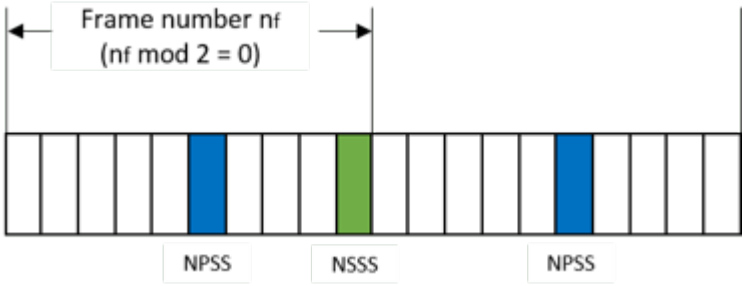
subframe repetition, logical and transport channel mappings, and the corresponding grids for the different configurations. Moreover, the parameters involved in the waveform generation are shown to create waveforms as per user requirements.

The example outputs the complex baseband waveform along with the populated grid containing all the above mentioned physical channels and signals and also information about the type of data contained in each subframe. The waveform can be used for a range of applications from RF testing to simulation of receiver implementations.

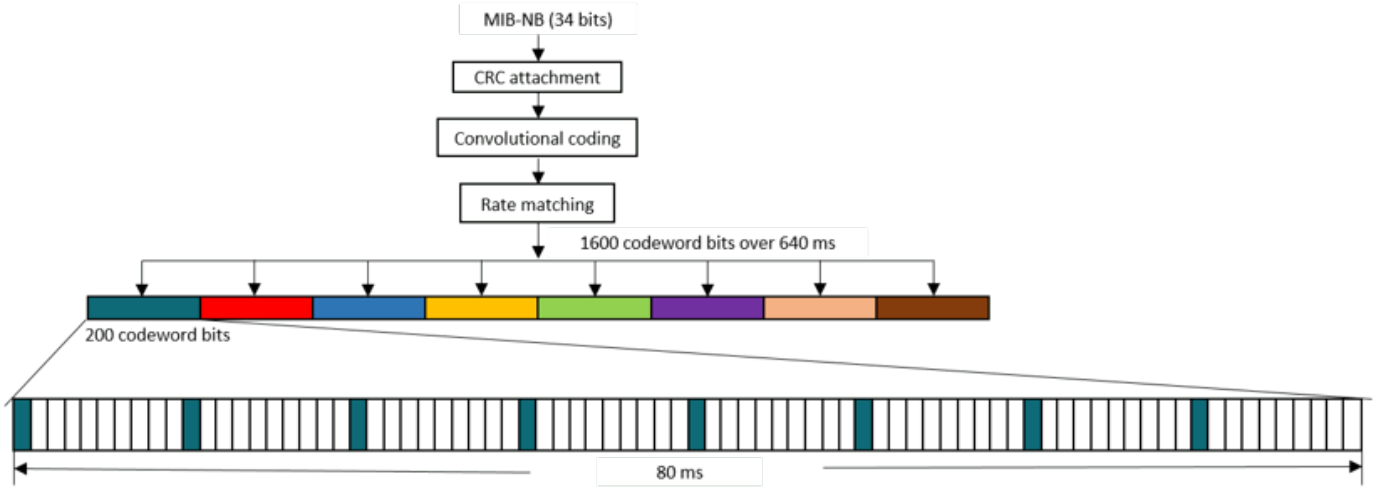
NB-IoT Downlink Subframe Allocation

This section explains how the physical layer channels and signals mentioned above are mapped into the downlink subframes.

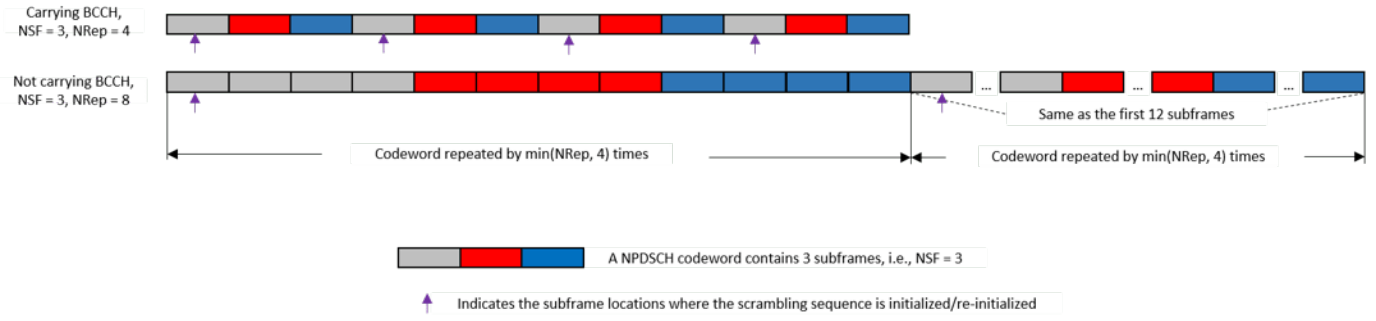
- *Downlink bit map*: A logical vector to configure the NB-IoT downlink subframes. NB-IoT downlink subframes are defined as the subframes used for NPDCCH and NPDSCH not carrying SIB1-NB, it does not include subframes carrying NPSS, NSSS, NPBCH and NB-IoT system information block type 1 (SIB1-NB). The NB-IoT downlink subframes can be configured using parameter `Config.DownlinkBitmap` in class NBIoTDownlinkWaveformGenerator.
- *NPSS & NSSS*: As illustrated in the figure below, the NPSS is transmitted in subframe 5 in every frame, and the NSSS is transmitted in subframe 9 in frames with frame number `nf` fulfilling `nf` mod 2 = 0. The NPSS and NSSS allow the UE to synchronize to the NB-IoT cell.



- *NPBCH*: The NPBCH is used to carry the 34 bit MIB-NB (TS 36.212 section 6.4.1 [2]). The MIB-NB is coded to form a 1600 bits codeword (TS 36.211 section 10.2.4.1 [1]). The codeword is evenly segmented into 8 sub-blocks, each one has 200 bits which are transmitted on subframe 0 and repeated in the 7 following consecutive frames (TS 36.211 section 10.2.4.4 [1]). The mapping of the codeword bits is illustrated in the figure below.



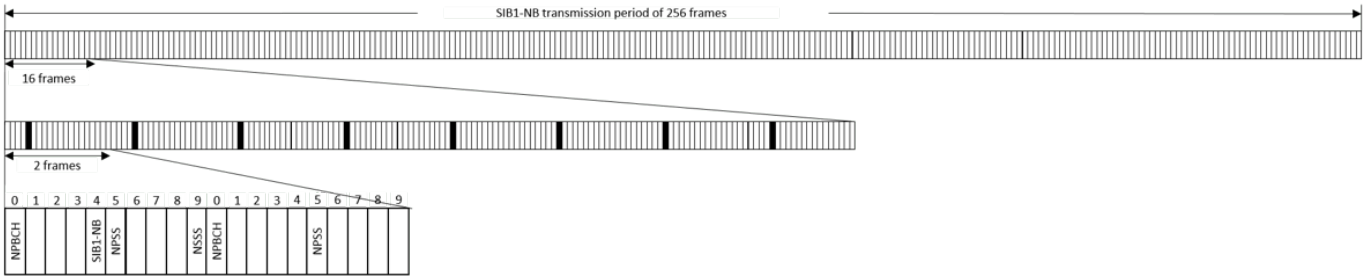
- *NPDSCH*: The key feature of NPDSCH is subframe repetition. NB-IoT defines two repetition schemes for the case when NPDSCH is carrying broadcast control channel (BCCH) or not. BCCH is the logical channel to carry SIB1-NB, system information messages, etc. Not carrying BCCH implies that NPDSCH may carry Paging Control Channel (PCCH), Common Control Channel (CCCH), Dedicated Control Channel (DCCH), Dedicated Traffic Channel (DTCH), etc (TS 36.300 sections 6.1.3.2 and 5.3.1a [6]). The two repetition schemes are illustrated in the figure below. The NPDSCH repetition parameters includes the number of subframes in a codeword `NSF` and the number of repetitions `NRep`. The example in the figure shows the repetition pattern with `NRep` = 4 for the case when NPDSCH is carrying BCCH and `NRep` = 8 for the case when NPDSCH is not carrying BCCH and `NSF` = 3 for both cases. For the case when carrying BCCH, all the subframes of a codeword are transmitted before repeating the codeword. For the case when not carrying BCCH, a subframe in the codeword is repeated by `min(NRep,4)` before performing the same repetition for the other subframes. After the codeword is repeated by `min(NRep,4)` times, the same procedure is performed until the full repetition is complete, i.e., repeated by `NRep` times. The detailed specification of the repetition scheme can be found in TS 36.211 section 10.2.3 [1].



The repeated subframes shown in the above figure are mapped to the available subframes allocated for NPDSCH transmission. As a typical example, the following figure illustrates how to perform mapping for the subframes of NPDSCH carrying SIB1-NB, which contains the most important system information for an UE (TS 36.331 section 6.7.2 [5]).

A SIB1-NB is carried in 8 subframes (`NSF` = 8), and mapped to subframe 4 in every other frame in 16 continuous frames which are repeated `NRep` times (`NRep` = 4, 8 or 16, see TS 36.213 section 16.4.1.3 Table 16.4.1.3-3 [3]). The repetitions are equally spaced within a period of 256 frames (TS 36.331 section 5.2.1.2a [5]). The starting frame number for the first NPDSCH transmission in the period

depends on the narrowband physical cell identity `NNCellID` as well as the number of repetitions `NRep` (TS 36.213 Table 16.4.1.3-4 [3]).



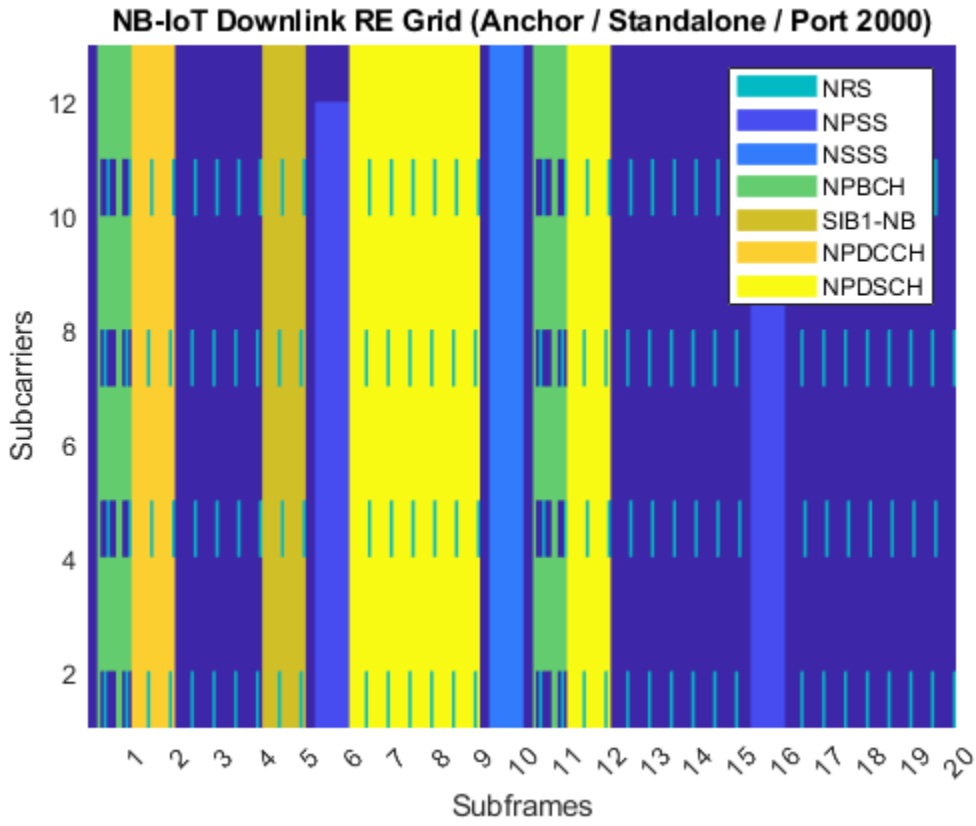
NB-IoT Downlink Grid

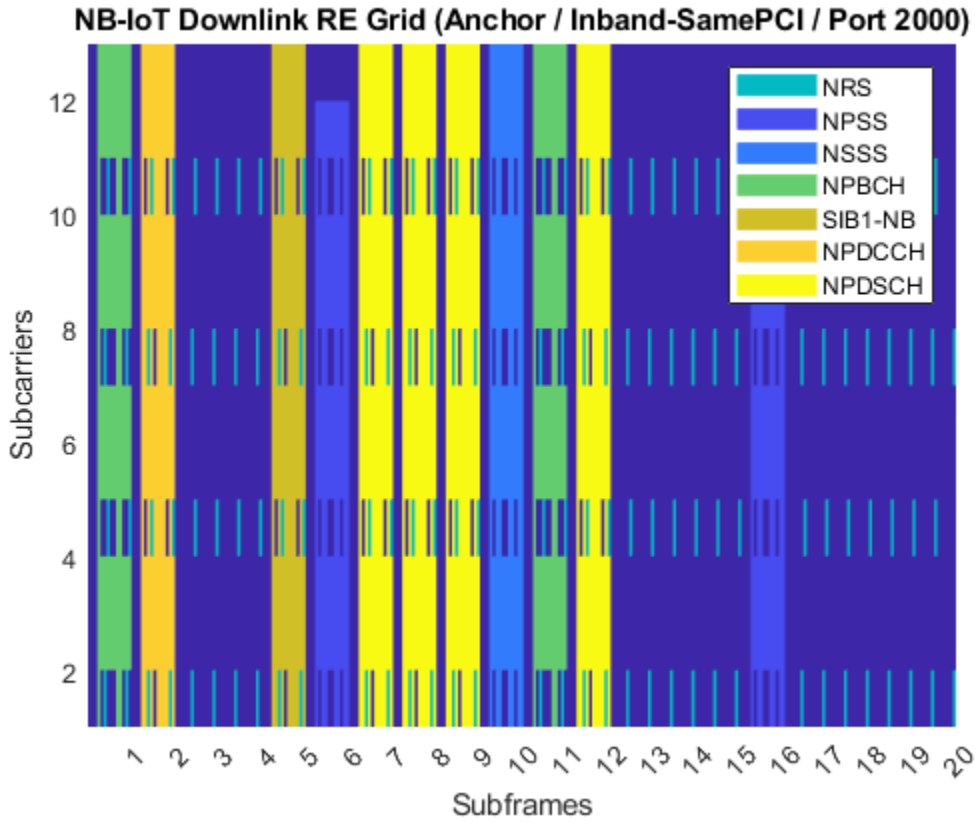
In addition to the subframe allocation described above, the generated grids below further explain the RE allocation in a subframe. The grid is for two frames of an anchor carrier containing NPSS, NSSS, NRS, NPBCH, SIB1-NB and NB-IoT downlink subframes carrying NPDSCH and NPDCCH. The grids are compared in `Standalone` and `Inband-SamePCI` operation modes. The grid can be generated using the grid display method in the class NBIoTDownlinkWaveformGenerator, i.e., creating an object `ngen` of type NBIoTDownlinkWaveformGenerator and calling `ngen.displayResourceGrid`.

- *NRS*: The RE positions can be configured by the number of NRS ports and the narrowband physical cell identity, i.e., the parameter fields `NBRefP` and `NNCellID` in structure `ngen.Config`, respectively.
- *NPSS & NSSS*: The first 11 subcarriers are used for NPSS, all 12 subcarriers in a physical resource block are used for NSSS. The first 3 OFDM symbols in a subframe are not used for NPSS/NSSS. NRS is not transmitted in any subframe containing NPSS/NSSS. The REs of NPSS/NSSS are punctured by the LTE cell-specific reference signal (CRS) only in in-band modes. The number of CRS ports that affects the puncturing can be configured by parameter field `CellRefP` in structure `ngen.Config` (TS 36.211 sections 10.2.6 and 10.2.7 [1]).
- *NPBCH*: The REs are punctured by the NRS and the CRS using the maximum number of NRS and CRS antenna ports (2 and 4, respectively), for both operation modes (TS 36.211 section 10.2.4 [1]). This is because the UE has no knowledge about the number of used antenna ports and the operation mode.
- *NPDSCH*: For operation modes `Standalone` and `Guardband`, the REs are punctured by NRS only; for in-band operation modes, the REs are punctured by both NRS and CRS. When the operation mode is in-band, the first 3 OFDM symbols in the subframe are not used for NPDSCH carrying SIB1-NB, the first `ControlRegionSize` OFDM symbols in the subframe are not used when NPDSCH is carried by a NB-IoT downlink subframe. `ControlRegionSize` is a parameter field in structure `ngen.Config` to configure the LTE control region size for NPDSCH RE allocations (TS 36.211 section 10.2.3.4 [1], TS 36.213 section 16.4.1.4 [3] and TS 36.331 section 6.7.2 [5]). The LTE control region size configures the starting OFDM symbol position in a NB-IoT downlink subframe carrying NPDSCH and NPDCCH in in-band operation modes.
- *NPDCCH*: The NRS and CRS puncturing are the same as that in NPDSCH described above. When the operation mode is in-band, the first `ControlRegionSize` OFDM symbols in the subframe are not used for NPDCCH. Same as that for NPDSCH, `ControlRegionSize` is used to configure the NPDCCH RE allocations (TS 36.211 section 10.2.3.4 [1], TS 36.213 section 16.4.1.4 [3] and TS 36.331 section 6.7.2 [5]).

```
ngen = NBIoTDownlinkWaveformGenerator;
```

```
figure;
% Display the resource grid with default 'Standalone' operation mode
ngen.displayResourceGrid;
figure;
% Change the operation mode to 'Inband-SamePCI'
ngen.Config.OperationMode = 'Inband-SamePCI';
ngen.displayResourceGrid;
```





The object `ngen` contains the following:

- Resource allocation and transport block size (TBS) tables for NPDSCH (see TS 36.213 sections 16.4.1.3 and 16.4.1.5 [3])
- Structure `ngen.Config` with the NB-IoT eNodeB configuration where parameter fields `NPBCH`, `SIB1NPDSCH`, `NPDCCH` and `NPDSCH` configure NPBCH, NPDSCH carrying SIB1-NB, NPDCCH and NPDSCH not carrying SIB1-NB (i.e., NPDSCH carried by NB-IoT downlink subframes), respectively.

```
ngen                                    % Show all available properties
nsfTable = ngen.NSFTable                % Display number of subframe table in TS 36.213 sectio
enbConfig = ngen.Config                 % Display NB-IoT eNodeB configuration
npbchConfig = ngen.Config.NPBCH         % Display NPBCH configuration
sib1npdschConfig = ngen.Config.SIB1NPDSCH % Display the configuration of NPDSCH when carrying SI
```

```
ngen =

  NBIoTDownlinkWaveformGenerator with properties:

          Config: [1x1 struct]
        NSFTable: [8x2 table]
       NRepTable: [16x2 table]
        TBSTable: [112x3 table]
    NRepTableSIB1: [12x2 table]
     TBSTableSIB1: [12x2 table]
```

```
nsfTable =

  8x2 table

    ISF    NSF
    ___    ___

     0      1
     1      2
     2      3
     3      4
     4      5
     5      6
     6      8
     7     10


enbConfig =

  struct with fields:

          TotSubframes: 20
              NNCellID: 0
                NBRefP: 1
               CellRefP: 4
     ControlRegionSize: 3
                NFrame: 0
         OperationMode: 'Inband-SamePCI'
        DownlinkBitmap: [1 1 1 1 1 1 1 1 1 1]
           CarrierType: 'Anchor'
        DLGapThreshold: 32
      DLGapPeriodicity: 64
    DLGapDurationCoeff: 0.1250
             NPSSPower: 0
             NSSSPower: 0
                 NPBCH: [1x1 struct]
            SIB1NPDSCH: [1x1 struct]
                NPDCCH: [1x1 struct]
                NPDSCH: [1x1 struct]


npbchConfig =

  struct with fields:

           Power: 0
    EnableCoding: 'On'
     DataBlkSize: 34
      DataSource: 'PN9'


sib1npdschConfig =

  struct with fields:

          Enable: 'On'
           Power: 0
```

```
          NRep: 4
 EnableCoding: 'On'
  DataBlkSize: 208
   DataSource: 'PN9'
```

NPDCCH/NPDSCH Configuration

The generated RE grid below explains how to configure the subframes used for NB-IoT downlink subframes using downlink bit map, and how to configure the parameters of NPDCCH/NPDSCH carried by the NB-IoT downlink subframes. The NB-IoT downlink subframes can be configured by parameter field `DownlinkBitmap` in the structure `ngen.Config`. In the example, a non-anchor carrier is used to disable NPSS, NSSS, NPBCH and SIB1-NB in the grid.

The waveform generator supports multiple NPDCCH and NPDSCH. Each NPDSCH and NPDCCH contains a single transport block and downlink control information (DCI) message, respectively. The parameters of a NPDCCH/NPDSCH is specified by a structure, and multiple NPDCCH/NPDSCH are expressed as a structure vector.

A NPDCCH is configured by the following parameters in the structure `ngen.Config.NPDCCH`:

- `Enable`: Enabling or disabling the NPDCCH ('On', 'Off').
- `Power`: Relative power for NPDCCH symbols in the frequency domain with assumption that the NRS power is 1.
- `NCCE`: The selected narrowband control channel elements (NCCEs) to carry the NDPCCH. The value can be either a scalar or a vector of two entries. A scalar or vector indicates NPDCCH format 0 or 1, respectively. Entry value 0 (or 1) indicates NCCE 0 (or NCCE 1). NCCE 0 occupies subcarriers 0 to 5 and NCCE 1 occupies subcarriers 6 to 11, respectively. (TS 36.211 section 10.2.5.1 [1]).
- `NRep`: Number of repetitions for a NPDCCH candidate (TS 36.213 section 16.6 [3]). Allowed values are 2^n, where n = 1...10.
- `Rmax`: Maximum number of repetitions for NPDCCH (TS 36.213 section 16.6 [3]). It affects the NPDCCH transmission gap. Allowed values are 2^n, where n = 1...10.
- `RNTI`: Radio network temporary identifier used for scrambling.
- `StartSubframe`: 0-based starting subframe index of the NPDCCH in the generated RE grid.
- `DataBlkSize`: Length of the DCI information bits.
- `DataSource`: DCI information bits. The parameter can be defined directly as a bit vector or using a random data type e.g., 'PN9-ITU', 'PN9', 'PN11', 'PN15', 'PN23'. For the latter case, a random bit vector is generated for the selected type.

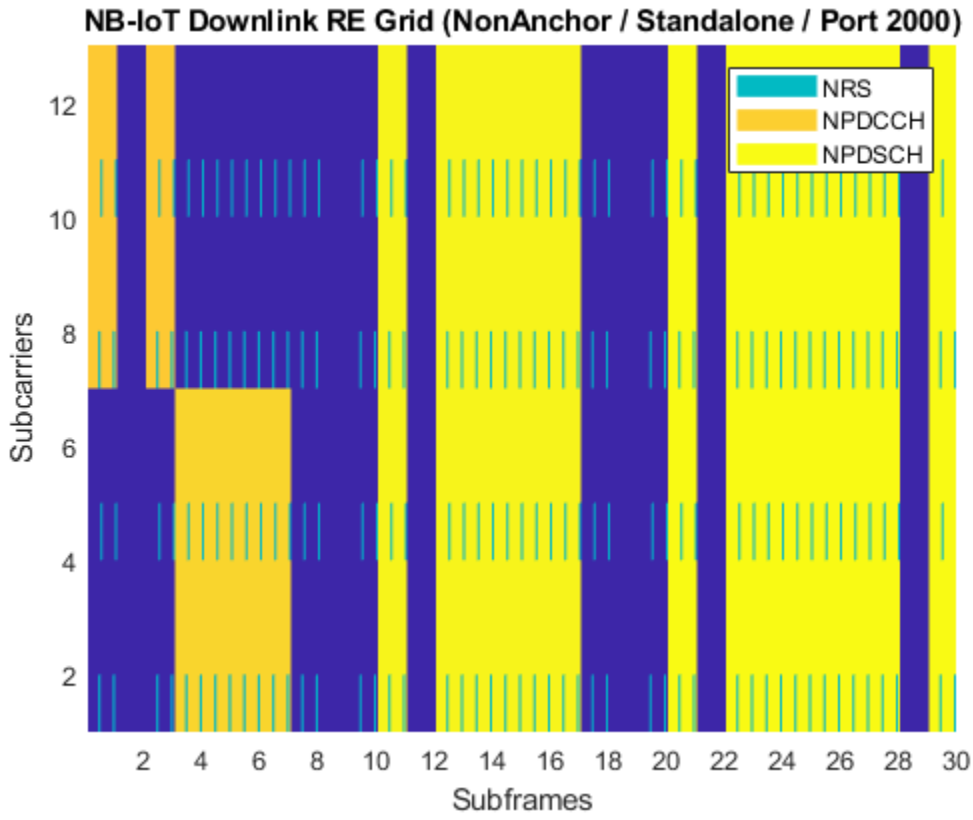
A NPDSCH is configured by the following parameters in the structure `ngen.Config.NPDSCH`:

- `Enable`: Enabling or disabling the NPDSCH ('On', 'Off').
- `Power`: Relative power for NPDSCH symbols in the frequency domain with assumption that the NRS power is 1.
- `NPDSCHDataType`: Type of data the NPDSCH carries, the allowed values are either 'BCCHNotSIB1NB' or 'NotBCCH'. This value affects the repetition scheme as described above.
- `NRep`: The number of repetitions (TS 36.211 section 16.4.1.3 [1]).
- `NSF`: The number of subframes in a NPDSCH codeword (TS 36.211 section 16.4.1.3 [1]).

- `Rmax`: Maximum number of repetitions for the NPDCCH associated to the NPDSCH, its value should be equal to the value of the same field in the corresponding NPDCCH.
- `StartSubframe`: 0-based starting subframe index of the NPDSCH in the generated RE grid.
- `DataBlkSize`: Transport block size.
- `DataSource`: Information bits of the transport block. The usage is the same as the same field in structure `ngen.Config.NPDCCH`.

```
ngen = NBIoTDownlinkWaveformGenerator;
ngen.Config.CarrierType = 'NonAnchor'; % Anchor or NonAnchor
ngen.Config.DownlinkBitmap = [1 0 1 1 1 1 1 1 0 1];
ngen.Config.TotSubframes = 30;

% Configure the parameters of the first NPDCCH
npdcch1.Enable = 'On';
npdcch1.Power = 0;
npdcch1.NCCE = 1;  % NPDCCH format 0 with NCCE 1
npdcch1.NRep = 2;
npdcch1.Rmax = 16;
npdcch1.RNTI = 0;
npdcch1.StartSubframe = 0;
npdcch1.DataBlkSize = 23;
npdcch1.DataSource = randi([0 1],npdcch1.DataBlkSize,1); % Users can define their own informatio

% Configure the parameters of the second NPDCCH
npdcch2.Enable = 'On';
npdcch2.Power = 0;
npdcch2.NCCE = 0;  % NPDCCH format 0 with NCCE 0
npdcch2.NRep = 4;
npdcch2.Rmax = 16;
npdcch2.RNTI = 1;
npdcch2.StartSubframe = 3;
npdcch2.DataBlkSize = 23;
npdcch2.DataSource = 'PN9';

% Configure the parameters of the first NPDSCH
npdsch1.Enable = 'On';
npdsch1.Power = 0;
npdsch1.NPDSCHDataType = 'BCCHNotSIB1NB';
npdsch1.NSF = 3;
npdsch1.NRep = 2;
npdsch1.Rmax = npdcch1.Rmax;
npdsch1.RNTI = 0;
npdsch1.StartSubframe = 10;
npdsch1.DataBlkSize = 616;
npdsch1.DataSource = 'PN15';

% Configure the parameters of the second NPDSCH
npdsch2.Enable = 'On';
npdsch2.Power = 0;
npdsch2.NPDSCHDataType = 'NotBCCH';
npdsch2.NSF = 2;
npdsch2.NRep = 4;
npdsch2.Rmax = npdcch2.Rmax;
npdsch2.RNTI = 1;
npdsch2.StartSubframe = 20;
npdsch2.DataBlkSize = 616;
```

```
npdsch2.DataSource = 'PN23';

% Prepare NPDSCH and NPDCCH structure vectors
ngen.Config.NPDCCH = [npdcch1 npdcch2];
ngen.Config.NPDSCH = [npdsch1 npdsch2];

figure;
ngen.displayResourceGrid;
```



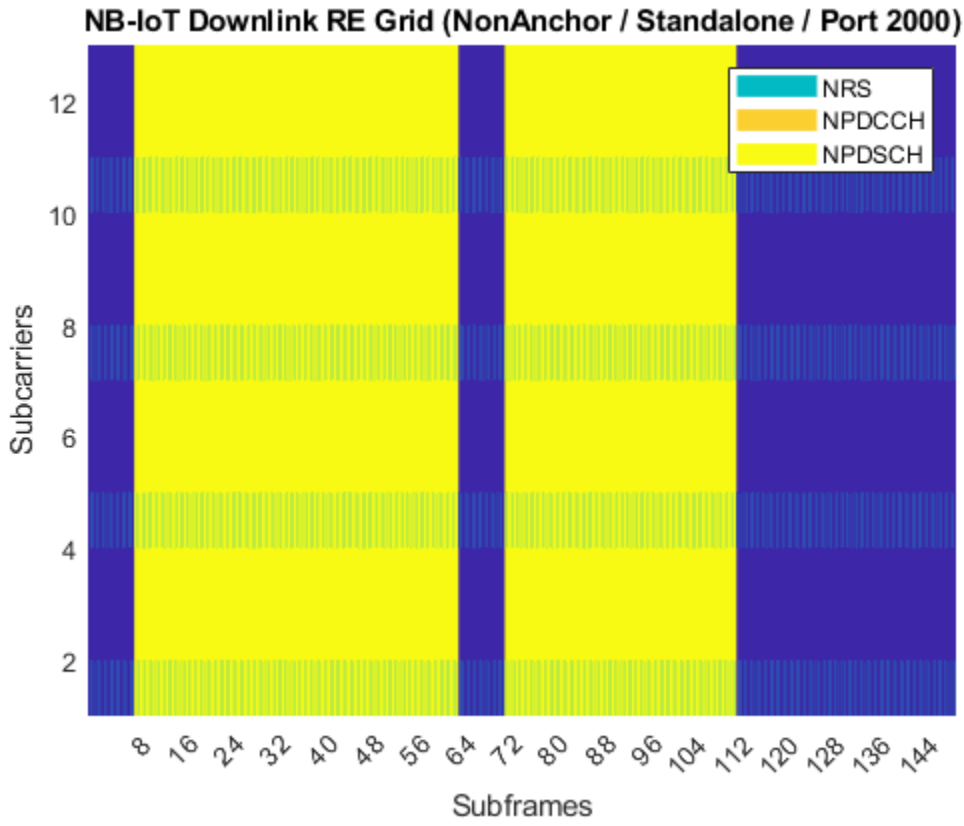
The generated RE grid below explains how to configure the transmission gap for NPDSCH/NPDCCH according to TS 36.211 sections 10.2.3.4 and 10.2.5.5 [1]. The gap is defined by parameter field `Rmax` mentioned above, as well as the following parameters in structure `ngen.Config`:

- `DLGapThreshold`: Threshold to trigger the transmission gap, i.e., there are no gaps in the NPDSCH transmission if `Rmax` < `DLGapThreshold`. Allowed values are 32, 64, 128, 256 (TS 36.331 section 6.7.1 [5]).
- `DLGapPeriodicity`: The gap periodicity in number of subframes. It also defines the starting subframe of the gap, i.e., the starting subframe number `sf` satisfying condition (sf mod `DLGapPeriodicity`) = 0. Allowed values are 64, 128, 256, 512 (TS 36.331 section 6.7.1 [5]).
- `DLGapDurationCoeff`: Used to calculate the gap duration in number of subframes, together with `DLGapPeriodicity`. The gap duration is given by `DLGapPeriodicity` × `DLGapDurationCoeff`. Allowed values are 1/8, 1/4, 3/8, 1/2 (TS 36.331 section 6.7.1 [5]).

The following example uses defaulting parameters `DLGapThreshold` = 32, `DLGapPeriodicity` = 64 and `DLGapDurationCoeff` = 1/8 to explain the NPDSCH transmission gap. The figure below

illustrates that the 96 (NRep × NSF) NPDSCH subframes is disrupted by two gaps with a duration of 8 subframes, and the two gaps start at subframe 0 and subframe 64, respectively.

```
ngen = NBIoTDownlinkWaveformGenerator;
ngen.Config.CarrierType = 'NonAnchor'; % Anchor or NonAnchor
ngen.Config.TotSubframes = 150;
ngen.Config.NPDCCH.Enable = 'Off';     % Disable the NPDCCH
ngen.Config.NPDSCH.StartSubframe = 0;
ngen.Config.NPDSCH.Rmax = ngen.Config.DLGapThreshold; % Minimum value to trigger the transmissio
ngen.Config.NPDSCH.NRep = 32;
ngen.Config.NPDSCH.NSF = 3;

figure;
ngen.displayResourceGrid;
```



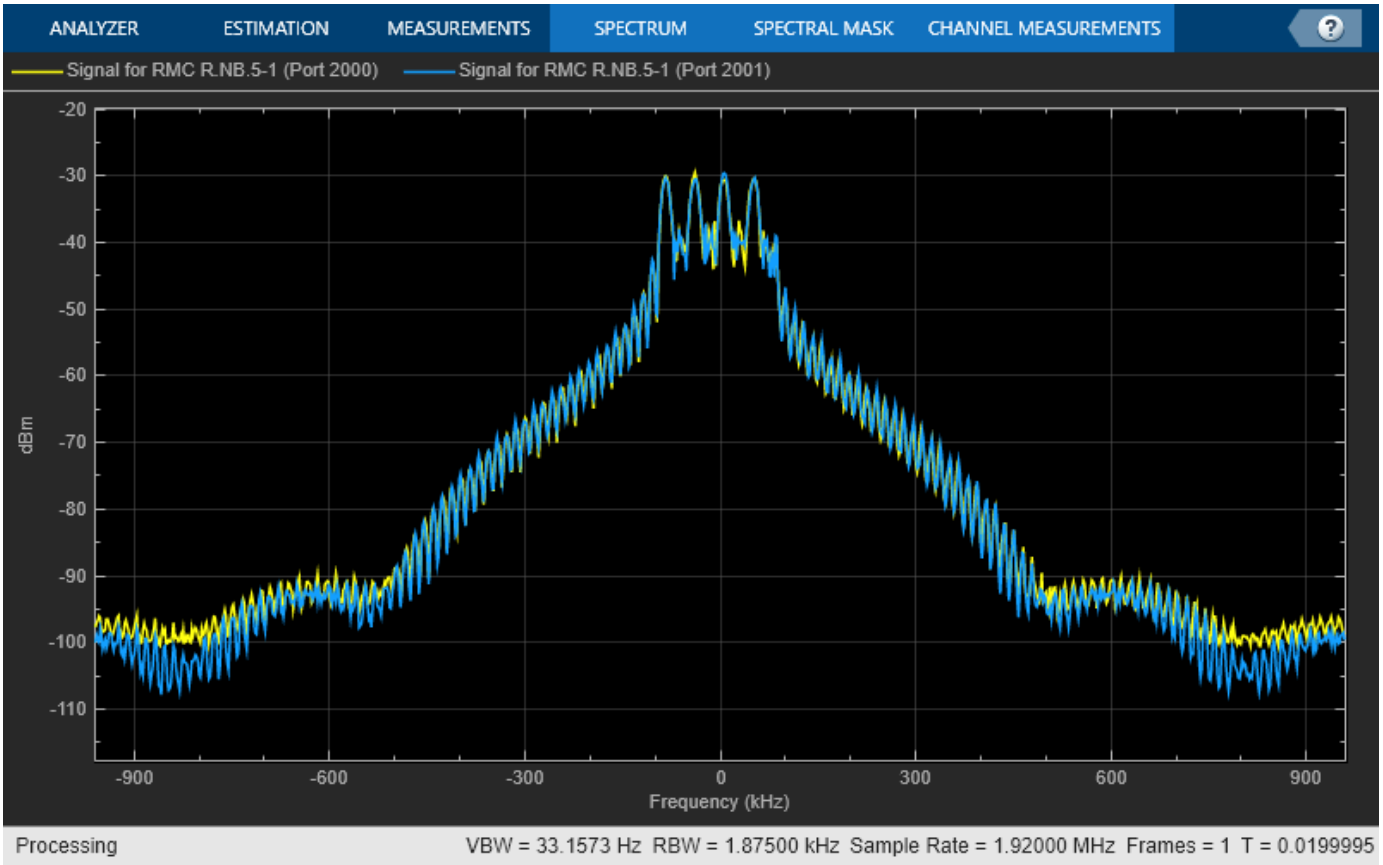
NB-IoT Downlink Waveform Generation

Generate the time-domain waveform of a reference measurement channel (RMC) for NPDSCH performance requirements, according to TS 36.101 Appendix A.3.12 [7], or the NB Test Model (N-TM), defined in TS 36.141 sections 6.1.3-6.1.6 [8]. The waveform can be generated using the waveform generation method in the class NBIoTDownlinkWaveformGenerator, to call the method, an object ngen is created and the method can be called by using ngen.generateWaveform.

```
rc = 'R.NB.5-1'; % Allowed values are 'R.NB.5','R.NB.5-1','R.NB.6','R.NB.6-1','R.NB.7','N-TM'
ngen = NBIoTDownlinkWaveformGenerator(rc);
[waveform,grid,ofdmInfo] = ngen.generateWaveform;
ofdmInfo          % Display OFDM configuration
```

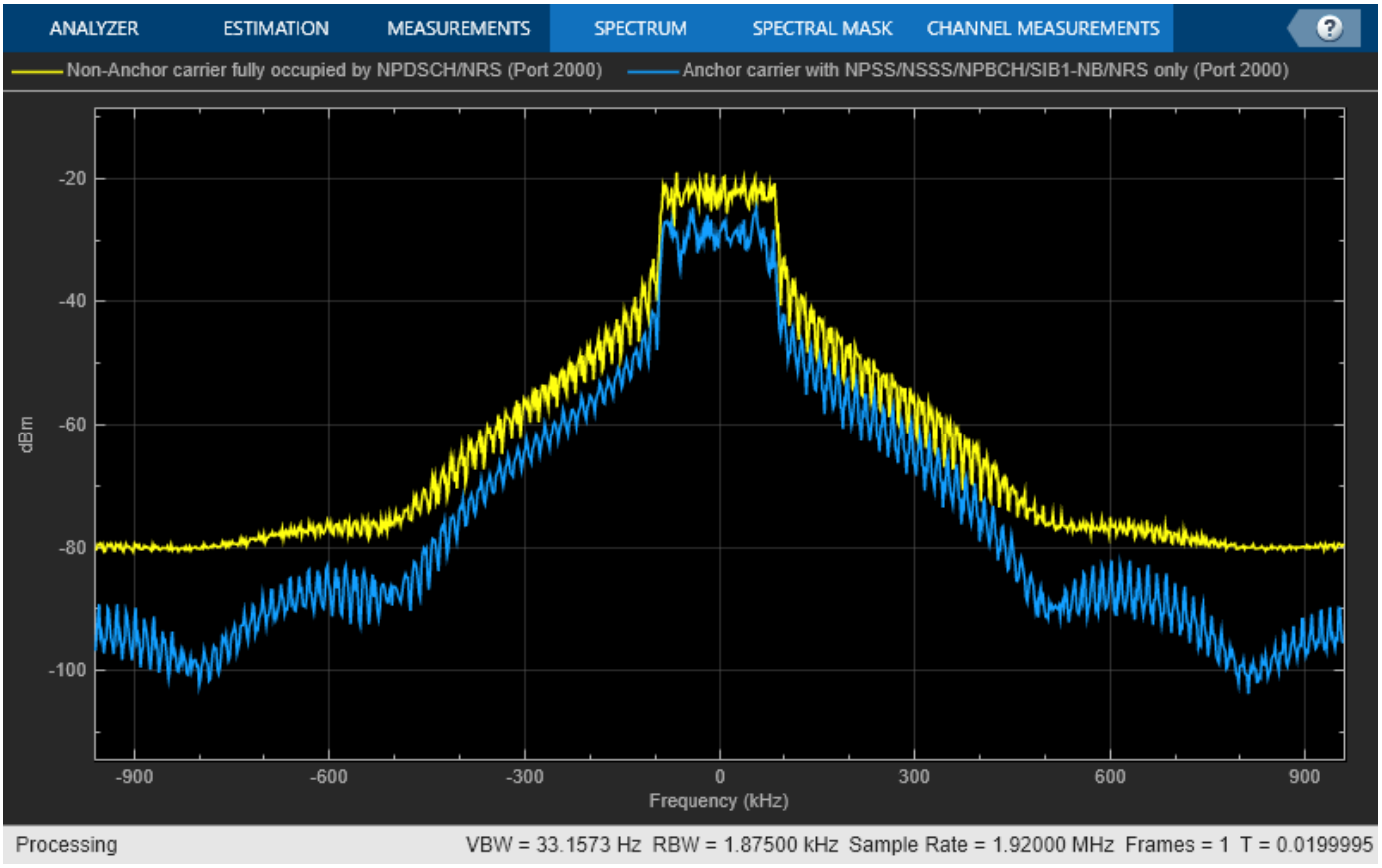
```
spectrumScope = spectrumAnalyzer;
spectrumScope.SampleRate = ofdmInfo.SamplingRate;
spectrumScope.ShowLegend = true;
if ngen.Config.NBRefP == 1
    spectrumScope.ChannelNames = {['Signal for RMC ' rc ' (Port 2000)']};
    spectrumScope(waveform);
else % NBRefP == 2
    spectrumScope.ChannelNames = {['Signal for RMC ' rc ' (Port 2000)'], ...
        ['Signal for RMC ' rc ' (Port 2001)']};
    spectrumScope(waveform(:,1),waveform(:,2));
end
```

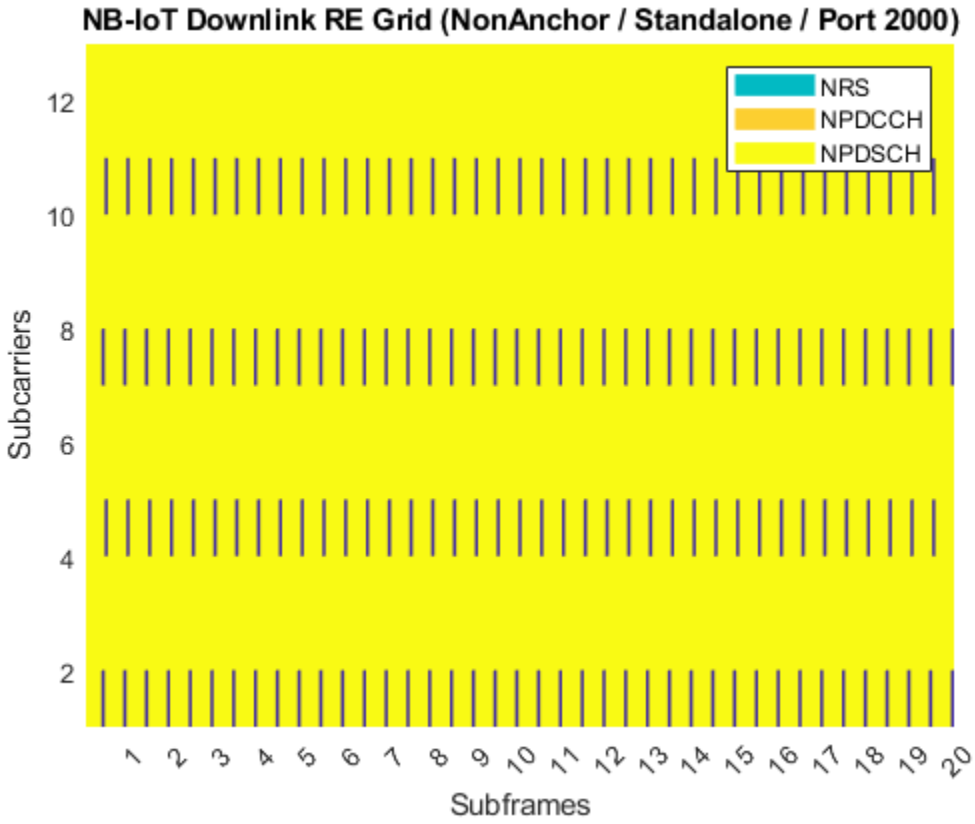
```
ofdmInfo =

  struct with fields:

           SamplingRate: 1920000
                   Nfft: 128
              Windowing: 6
     CyclicPrefixLengths: [10 9 9 9 9 9 9 10 9 9 9 9 9 9]
    SubframeChannelTypes: ["NPDCCH"    "Unused"    "Unused"    ...    ]
```

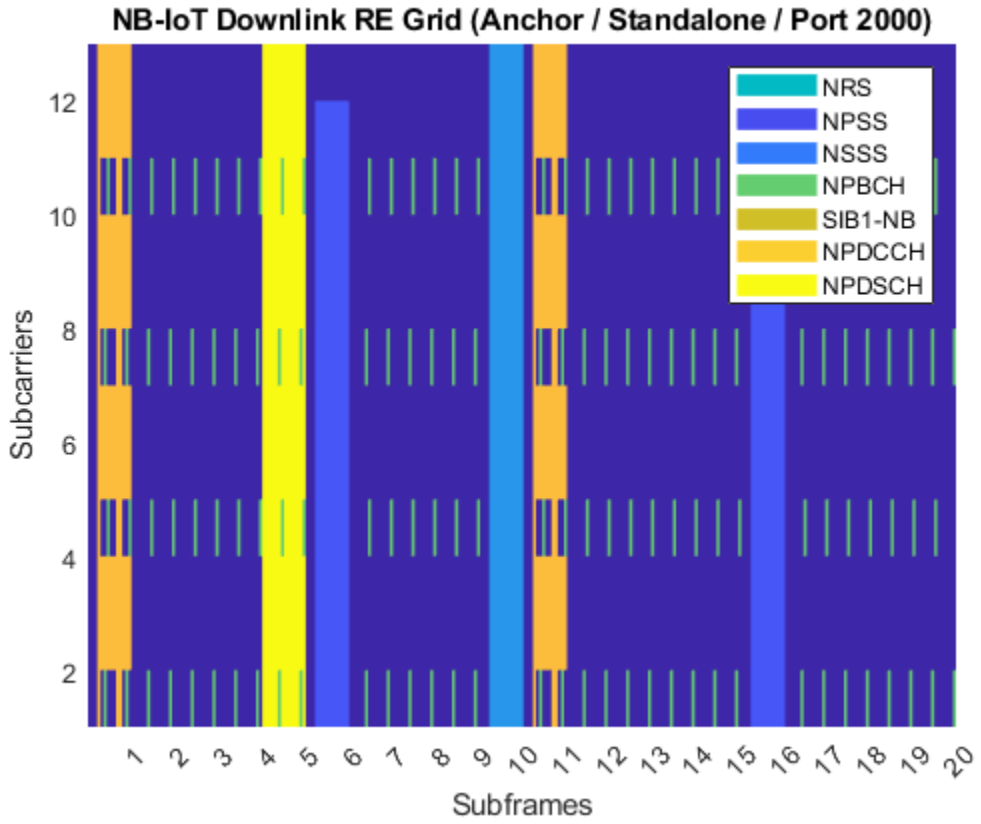


The following figures compare the signal spectrums for the generated time-domain waveform. As shown in the spectrum analyzer, the signal with yellow color has stronger power than the one with blue color. This is because the RE usage percentage of yellow one is higher.

```
% Set up a standalone, non-anchor carrier, disable the NPDCCH and make the
% REs grid fully occupied by NPDSCH/NRS.
ngen = NBIoTDownlinkWaveformGenerator;
ngen.Config.CarrierType = 'NonAnchor';
ngen.Config.NPDCCH.Enable = 'Off'; % Disable the NPDCCH
ngen.Config.NPDSCH.StartSubframe = 0;
ngen.Config.NPDSCH.NRep = 8;
ngen.Config.NPDSCH.NSF = 5;

% Display the grid and generate the time-domain signal
figure;
ngen.displayResourceGrid;
[waveform1,~,ofdmInfo1] = ngen.generateWaveform;

% Set up a standalone, anchor carrier, disable the NPDSCH and NPDCCH so
% that it contains NPSS/NSSS/NPBCH/SIB1-NB/NRS only.
ngen = NBIoTDownlinkWaveformGenerator;
ngen.Config.NPDCCH.Enable = 'Off'; % Disable the NPDCCH
ngen.Config.NPDSCH.Enable = 'Off'; % Disable the NPDSCH

% Display the grid and generate the time-domain signal
figure;
ngen.displayResourceGrid;
waveform2 = ngen.generateWaveform;

% Plot the signal spectrums of the generated two waveforms.
release(spectrumScope);
spectrumScope.ShowLegend = true;
spectrumScope.ChannelNames = {'Non-Anchor carrier fully occupied by NPDSCH/NRS (Port 2000)', ...
    'Anchor carrier with NPSS/NSSS/NPBCH/SIB1-NB/NRS only (Port 2000)'};
spectrumScope.SampleRate = ofdmInfo1.SamplingRate;
spectrumScope(waveform1,waveform2);
```


ANALYZER
ESTIMATION
MEASUREMENTS
SPECTRUM
SPECTRAL MASK
CHANNEL MEASUREMENTS
Non-Anchor carrier fully occupied by NPDSCH/NRS (Port 2000)
Anchor carrier with NPSS/NSSS/NPBCH/SIB1-NB/NRS only (Port 2000)
-20
-40
-60
-80
-100
dBm
-900
-600
-300
0
300
600
900
Frequency (kHz)
Processing
VBW = 33.1573 Hz RBW = 1.87500 kHz Sample Rate = 1.92000 MHz Frames = 1 T = 0.0199995


NB-IoT Downlink RE Grid (NonAnchor / Standalone / Port 2000)
NRS
NPDCCH
NPDSCH
Subcarriers
12
10
8
6
4
2
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Subframes



Selected Bibliography

1 3GPP TS 36.211 "Physical channels and modulation"

2 3GPP TS 36.212 "Multiplexing and channel coding"

3 3GPP TS 36.213 "Physical layer procedures"

4 3GPP TS 36.321 "Medium Access Control (MAC); Protocol specification"

5 3GPP TS 36.331 "Radio Resource Control (RRC); Protocol specification"

6 3GPP TS 36.300 "Overall description; Stage 2"

7 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception"

8 3GPP TS 36.141 "Base Station (BS) conformance testing"

9 O. Liberg, M. Sundberg, Y.-P. Wang, J. Bergman and J. Sachs, Cellular Internet of Things: Technologies, Standards and Performance, Elsevier, 2018.

LTE-M Downlink Waveform Generation

This example shows how to create a downlink LTE-M transmission consisting of MTC Physical Downlink Control Channel (MPDCCH) as well as its associated Physical Downlink Shared Channel (PDSCH) and Physical Broadcast Channel (PBCH), including repetitions and frequency hopping. When compared to pre-Release 13 devices, Cat-M devices offer lower cost and complexity, enhanced coverage by the introduction of repetitions and an extended DRX for further power saving.

Introduction

The introduction of LTE-M to the LTE standard added support specifically aimed at Machine-Type Communications (MTC). Starting with Cat-0 devices in Release 12, this was further extended in later releases to define a separate Cat-M class of devices. Two categories have now been added to LTE, Cat-M1 in Release 13 (through the *eMTC* work items) followed by Cat-M2 in Release 14 (*feMTC*).

There are two coverage enhancement (CE) modes of operation for Cat-M devices (also referred to as BL/CE UEs in 3GPP standard documents), CE mode A and CE mode B. The CE mode A targets modest coverage enhancement with up to 32 repetitions whereas CE mode B targets extensive coverage enhancement with up to 2048 repetitions for the data channel. CE mode A is signaled via DCI format 6-0A/6-1A messages and CE mode B via 6-0B/6-1B messages. The MPDCCH processing chain, DM-RS creation and mapping is almost identical to that of the Enhanced Physical Downlink Control Channel (EPDCCH), plus the addition of repetitions and frequency hopping behavior. This example makes specific use of LTE Toolbox™'s EPDCCH functionality to implement the MPDCCH model. The LTE-M data channel is the LTE PDSCH with addition of repetitions and frequency hopping. The LTE-M broadcast channel consists of two parts; the 'core' part corresponding to the LTE PBCH and the LTE-M specific 'repetition' part where the symbols and cell RS signals in the 'core' part are repeated.

LTE-M can be deployed in standard LTE cells, therefore not all subframes are necessarily used for BL/CE transmission. The BL/CE subframes are indicated by a bitmap sent in the MIB-M message (similar to the mechanism in NB-IoT). For simplicity, in this example, we assume that all subframes are defined to be BL/CE. The MPDCCH/PDSCH starting OFDM symbol for LTE-M transmissions are cell-specific and is broadcasted in the System Information (SI).

The main output of this MATLAB® example is a multi-subframe Cat-M1/Cat-M2 resource grid containing the MPDCCH Resource Elements (RE), associated DM-RS, the LTE-M PDSCH REs, PBCH REs (both 'core' and 'repetition' parts along with corresponding cell reference signals) and other reference signals as configured for the full transmission sequence. This grid is also OFDM modulated to generate the associated time-domain baseband waveform. Plots are created to provide visualization of the RE assignment in the grid and of the magnitude of the baseband signal.

LTE-M Physical Layer Concepts

- *Repetitions* - To enable significantly extended coverage for LTE-M devices, repetitions for MPDCCH, PDSCH and PBCH were introduced for Cat-M devices in Release 13. This would provide at least 15dB performance improvement over pre-release 13 devices. MPDCCH can be repeated up to a maximum of 256 times, PDSCH up to 2048 and PBCH up to 5 including the 'core' part.
- *Narrowbands* - LTE-M uses the concept of narrowbands to allocate subcarriers in the wideband LTE carrier for data and control channels. Each narrowband consists of 6 consecutive PRBs but not all PRBs are necessarily part of the narrowbands (dependent on the overall carrier BW).
- *Widebands* - Release 14 LTE-M uses the concept of widebands to allocate higher bandwidth for the data channel. There can be one or more widebands, each consisting of 4 narrowbands or a

single wideband consisting of 1,2 or 3 narrowbands. For a Cat-M2 device configured in CE mode A (DCI format 6-1A), the data channel is transmitted on a single wideband if `ce-pdsch-maxBandwidth-config` is set to 5MHz or `resource block assignment flag` is set to 1 or across the full LTE carrier bandwidth via an RBG bitmap in the case of `resource block assignment flag` is set to 0. The `resource block assignment flag` is only present if the `ce-pdsch-maxBandwidth-config` is set to 20MHz. For a Cat-M2 device configured in CE mode B (DCI format 6-1B), the data channel is transmitted on a single wideband if `ce-pdsch-maxBandwidth-config` is set to 5MHz and up to 4 widebands if `ce-pdsch-maxBandwidth-config` is set to 20MHz.

- *Frequency Hopping* - Although a Release 13 eMTC device has an instantaneous bandwidth of 1.4MHz (the simplest LTE-M devices have a maximum bandwidth of 1.4MHz), it can access a wider LTE carrier through different narrowbands between the subframes. If the transmission involves repetition over multiple subframes, optional inter-subframe frequency hopping can be applied. The frequency hopping happens between different narrowbands and in blocks of 1 to 16 subframes depending on CE mode. The hopping block length and hopping offset are cell-specific parameters. In the downlink, for both control and data, the frequency hopping can be over 2 narrowbands or 4 equally spaced narrowbands.

PDSCH Physical Layer Processing and Procedures

LTE-M data transmission uses the same physical channel (PDSCH) as in the case of LTE. The difference however is that LTE-M enables optional repetitions and frequency hopping.

- *Transmission modes* - The valid transmission modes for PDSCH for LTE-M are 1 (single antenna), 2 (transmit diversity), 6 (closed loop codebook based precoding) and 9 (non-codebook based precoding, single layer). Transmission mode 6 is only supported in CE mode A, whereas 1, 2 and 9 are supported in both CE mode A and CE mode B. Unlike LTE PDSCH, multiple input multiple output (MIMO) operation is not supported in LTE-M because most LTE-M devices are expected to be low cost devices with a single receive antenna.

- *Repetitions and scrambling blocks* - The PDSCH is transmitted in `NRepPDSCH` (>=1) consecutive BL/CE downlink subframes. These `NRepPDSCH` subframes are transmitted in blocks of `Nacc` subframes (see TS 36.211 section 6.8B.2 [1]) and the same scrambling sequence is applied to all subframes in the block. `Nacc` can be 1 or 4 in FDD duplex mode and 1 or 10 in TDD duplex mode. There can be up to 32 repetitions in CE mode A and up to 2048 repetitions in CE mode B. The number of PDSCH repetitions is defined by a combination of semi-static configuration and dynamic selection for a particular transmission.

- *Narrowbands and Widebands* - The PRBs used for LTE-M PDSCH transmissions must belong to narrowband(s) (and wideband(s) if Cat-M2). Cat-M1 devices have an instantaneous transmission bandwidth of 1.4MHz in a narrowband. Cat-M2 devices have a bandwidth of 5MHz or 20MHz as indicated by the higher layer parameter `ce-pdsch-maxBandwidth-config`, in this case the PDSCH transmission can be in wideband(s) where each wideband consists of up to 4 non-overlapping narrowbands.

- *Frequency Hopping* - If the PDSCH is repeated over multiple subframes, inter-subframe frequency hopping can be optionally applied where the hopping occurs for blocks of subframes of length 1,2,4,8 for devices in CE mode A and 2,4,8,16 in CE mode B. The hopping block length and hopping offset are cell-specific parameters. The frequency hopping can be over 2 narrowbands or 4 equally spaced narrowbands. The relative positions of the RBs carrying the PDSCH within each hopping narrowbands remains the same.

MPDCCH Physical Layer Processing and Procedures

MPDCCH carries the LTE-M downlink control information (DCI). The MPDCCH and corresponding Demodulation Reference Signals (DM-RS) are transmitted on one or more ports from the set (107,108,109,110). The physical layer processing stages and operations of the MPDCCH and EPDCCH are very similar, with the differences being the repetitions, optional frequency hopping and how the scrambling is applied.

- *Repetitions and scrambling blocks* - The MPDCCH is transmitted using an aggregation of one or more consecutive Enhanced Control Channel Elements (ECCEs) in `NRepMPDCCH` (>=1) consecutive BL/CE downlink subframes. These `NRepMPDCCH` subframes are transmitted in blocks of `Nacc` subframes (see TS 36.211 section 6.8B.2 [1]) and the same scrambling sequence is applied to all subframes in the block. As an example, for MPDCCH not associated with P-RNTI or SC-RNTI and configured with CE mode A, `Nacc` = 1 as described in TS 36.211 section 6.8B.2 [1] so the scrambling gets initialized every BL/CE subframe, as in the case of EPDCCH. `Nacc` can be 1 or 4 in FDD duplex mode and 1 or 10 in TDD duplex mode. The number of MPDCCH repetitions is defined by a combination of semi-static configuration and dynamic selection for a particular transmission. The network semi-statically configures the maximum repetitions `Rmax` from the set (1,2,4,8,16,32,64,128,256) then dynamically selects the actual repetitions for a specific transmission from the set (Rmax, Rmax/2, Rmax/4, Rmax/8).

- *Narrowbands* - The MPDCCH is transmitted in 2, 4 or 6 RBs in a single narrowband within each subframe. MPDCCH PRB set consists of ECCEs numbered from 0...N'ECCEpk where N'ECCEpk is the number of ECCEs in MPDCCH-PRB-set p of subframe k. The number of ECCEs used for one MPDCCH is given by TS 36.211 Table 6.8A.1-2 [1] for 2 or 4 PRBs if repetition is not configured and Table 6.8B.1-2 for all other cases.

- *Frequency Hopping* - If the MPDCCH is repeated over multiple subframes, inter-subframe frequency hopping can be optionally applied where the hopping occurs for blocks of subframes of length 1,2,4,8 for devices in CE mode A and 2,4,8,16 in CE mode B. The hopping block length and hopping offset are cell-specific parameters. The frequency hopping can be over 2 narrowbands or 4 equally spaced narrowbands. The relative positions of the RBs carrying the MPDCCH within each hopping narrowbands remains the same.

PBCH Physical Layer Processing and Procedures

Similar to the PDSCH, LTE-M uses the same physical broadcast channel (PBCH) as in the case of LTE but with optional repetitions.

- *Core PBCH and Non-core PBCH* - LTE-M PBCH is divided into two parts; the 'core' PBCH which is the normal LTE PBCH and is transmitted in the first subframe of every frame with 40ms periodicity. The non-core part is the LTE-M specific repetitions where the 'core' PBCH symbols are repeated up to 5 times depending on the duplex mode and cyclic prefix length.

- *Repetitions* - PBCH repetitions occur in subframe 9 of the preceding frame and subframe 0 of the current radio frame for FDD as given by TS 36.211 Table 6.6.4-1 [1]. For TDD, the repetitions occur in subframes 0 and 5 of the same radio frame as given by Table 6.6.4-2. The subcarriers remain the same for the core part and the repetitions. Note that unlike the core part, the PBCH repetitions should not map to REs used by CSI reference signals. The repetitions are defined according to the duplex mode and cyclic prefix length. For FDD, all PBCH core symbols are repeated 4 times if normal cyclic prefix and 3 times if extended cyclic prefix. For TDD, if extended cyclic prefix, all symbols are repeated 3 times. For TDD and normal cyclic prefix, PBCH core symbols 0 and 1 are repeated 5 times and symbols 2 and 3 are repeated 3 times.

Simulation Settings

This example is divided into the following steps:

- Configure the simulation parameters (cell-wide and MPDCCH/PDSCH specific)
- Create and encode a DCI message payload for transmission on MPDCCH
- Create and encode a DL-SCH payload for transmission on PDSCH
- Generate the symbols for the MPDCCH subframe sequence and DM-RS, and map them into the resource elements of the multi-subframe resource grid
- Generate the symbols for the LTE-M PDSCH subframe sequence, map them into the resource elements of the multi-subframe resource grid
- Generate the symbols for the LTE-M PBCH subframe sequence, map them into the resource elements of the multi-subframe resource grid
- Apply CP-OFDM modulation to final grid to create the associated baseband waveform
- Display plots of the resource grid and time-series waveform

In this first section, we specify the simulation parameters to be used in later sections for the DCI, MPDCCH and DM-RS creation. This example uses an LTE FDD carrier with bandwidth of 5MHz and normal cyclic prefix. Cell-wide settings are contained in the structure `enb`, MPDCCH specific parameters are contained in structure `mpdcch` and PDSCH specific parameters are contained in structure `pdsch`.

```
enb = struct();
enb.NDLRB = 25;                       % LTE carrier BW
enb.NCellID = 1;                      % The cell ID
enb.CellRefP = 1;                     % Number of cell-specific ports
enb.CFI = 3;                          % CFI indicator
enb.DuplexMode = 'FDD';               % Duplex mode
enb.TDDConfig = 1;                    % Uplink/Downlink configuration (TDD)
enb.CyclicPrefix = 'Normal';          % Cyclic prefix duration
enb.NSubframe = 0;                    % Subframe number
enb.NFrame = 0;                       % Frame number
enb.CSIRSPeriod = 'Off';              % CSI-RS period control
enb.CSIRSConfig = 0;                  % CSI-RS configuration
enb.CSIRefP = enb.CellRefP;           % Number of CSI-RS antenna ports
enb.ZeroPowerCSIRSPeriod = 'Off';     % Zero power CSI-RS period control
enb.ZeroPowerCSIRSConfig = 0;         % Zero power CSI-RS configuration
enb.Ng = 'Sixth';                     % HICH group multiplier
enb.PHICHDuration = 'Extended';       % PHICH duration

% Set up hopping specific parameters
enb.HoppingOffset = 1;% Hopping offset 1...maxAvailableNarrowbands (TS 36.331)
enb.NChDLNBhop = 2;   % Number of narrowbands over which MPDCCH/PDSCH hops (2 or 4)
enb.NChDLNB = 2;      % Number of subframes in one hop/hopping block length

% MPDCCH Configuration
mpdcch = struct();
mpdcch.Hopping = true;          % Enable/Disable frequency hopping (only for 1.4MHz)
mpdcch.NRepMPDCCH = 8;          % The total number of MPDCCH repetitions
mpdcch.EPDCCHECCE = [0 7];      % ECCE range used for this MPDCCH (8 ECCE)
mpdcch.EPDCCHType = 'Localized';% Transmission type ('Localized', 'Distributed')
mpdcch.EPDCCHNID = 1;           % Cell ID/MPDCCH ID depending on search space type
mpdcch.EPDCCHStart = 4;         % MPDCCH start OFDM symbol in each subframe
mpdcch.RNTI = 1;                % RNTI for use with 'Localized' transmission
```

MPDCCH Allocation - The (initial) set of PRBs used for the MPDCCH transmission is defined by a vector of 0-based PRB indices specifying the absolute positions of the PRB within the wideband carrier. These PRBs should fall within a valid narrowband. If frequency hopping is disabled for MPDCCH, an MPDCCH candidate will always be transmitted in the PRBs specified by the 'InitPRBSet' parameter. If frequency hopping is enabled for MPDCCH, an MPDCCH candidate will be transmitted in the same relative positions of PRBs, but in different narrowbands as defined in TS 36.211 section 6.8B.5. For the 5MHz LTE carrier used in this example, there are four narrowbands as given by TS 36.211 section 6.2.7 [1]. The first narrowband has PRBs 0...5, second has PRBs 6...11, third has PRBs 13...18 and the fourth has PRBs 19...24. All other PRBs within the 5MHz band fall outside the narrowbands and cannot be used for LTE-M transmission. In this example, we use MPDCCH format 2, where 8 ECCEs are used for the MPDCCH transmission. Each PRB slot pair consists of 4 ECCEs therefore we need to define a minimum allocation of 2 PRB. In this example, we select the 3rd and 4th PRBs in the first narrowband

```
mpdcch.InitPRBSet = (2:3)';
mpdcch.InitNSubframe = 0; % Absolute subframe number of first MPDCCH subframe

% PDSCH Configuration
pdsch = struct();
pdsch.Hopping = true;       % Enable/Disable frequency hopping (only for 1.4MHz)
pdsch.NRepPDSCH = 16;       % The total number of PDSCH repetitions
pdsch.CEMode = 'A';         % A for CE mode A and B for CE mode B
pdsch.TxScheme = 'Port0';  % Port 0 or TxDiversity or Spatial Mux or Port 7
pdsch.Modulation  = 'QPSK';% Modulation scheme (QPSK/16QAM)
pdsch.NLayers = 1;          % Number of layers
pdsch.RV = 0;               % Redundancy version for DL-SCH processing
pdsch.RNTI = 1;             % RNTI used for the UE
pdsch.NSCID = 0;            % Scrambling identity
pdsch.TrBlkSizes = 100;     % Transport block size
```

PDSCH Allocation - The PRB allocation is flexible in the case of CE mode A (signaled via DCI format 6-1A) and fixed in the case of CE mode B (signaled via DCI format 6-1B). In CE mode A, the allocation can be on 1 to 6 PRBs within one or more narrowbands as defined by resource allocation type 2 or for the entire LTE carrier bandwidth via resource allocation type 0. In CE mode B, the allocation spans either the entire 6 PRBs or the first 4 PRBs of the narrowband(s) configured. For Cat-M1 devices, the transmission is in a single narrowband. For Cat-M2 devices, if the higher layer parameter `ce-pdsch-maxBandwidth-config` is set to 20MHz in CE mode B, the transmission can span multiple narrowbands and the overall allocation can be up to a maximum of 96 PRBs (grouped into widebands, where each wideband consists of 4 narrowbands). The widebands used are signaled via the wideband combination index field in the DCI as described in TS 36.213 Table 7.1.6-2 [3]. If `ce-pdsch-maxBandwidth-config` is set to 5MHz, the transmission occurs in a single wideband consisting of 1,2,3 or 4 narrowbands depending on the LTE carrier bandwidth (NDLRB).

In this example, we specify the allocation using the `InitPRBSet` parameter and `InitNarrowbandIndex`. If frequency hopping is disabled, LTE-M PDSCH will always be transmitted in the PRBs specified by the `InitPRBSet` and `InitNarrowbandIndex` parameters. If frequency hopping is enabled, PDSCH will be transmitted in the same relative positions of PRBs, but in different narrowbands as defined in TS 36.211 section 6.4.1 [1]. If the allocation spans more than one narrowband or one or more widebands, the `InitNarrowbandIndex` should be a vector specifying the constituent narrowband indices. As an example, if `ce-pdsch-maxBandwidth-config` is 20MHz and the LTE-M transmission is within a 20MHz LTE carrier, there are 4 widebands (0,1,2,3) available for LTE-M PDSCH transmission with corresponding narrowband indices {(0 1 2 3) (4 5 6 7) (8 9 10 11) (12 13 14 15)}. If the desired allocation is widebands 2 & 3, then the `InitNarrowbandIndex` must be then set to [8 9 10 11 12 13 14 15].

```
pdsch.InitPRBSet = (1:2)';     % 0-based PRB index
pdsch.InitNarrowbandIndex = 0; % 0-based wideband index
% In this example, we disable frequency hopping when transmission is over 2
% or more narrowbands
if numel(pdsch.InitNarrowbandIndex) > 1
    pdsch.Hopping = false;
    mpdcch.Hopping = false;
end
```

Enable or disable PBCH repetitions

```
enb.RepPBCHEnable = true;

% Specify the power scaling in dB for MPDCCH, MPDCCH DM-RS, PDSCH and
% reference signals (Cell RS or DM-RS)
mpdcch.MPDCCHPower = 30;
mpdcch.MPDCCHDMRSPower = 32;
pdsch.Rho = 25;
pdsch.RSPower = 80;
% Power levels for the PBCH core and reps parts
enb.PBCHPower = 33;
enb.PBCHRepsPower = 36;
% Power level for the PBCH Cell RS reps
enb.PBCHCellRSRepsPower = 28;
```

Total subframes to simulate (all downlink subframes are BL/CE subframes) and the MPDCCH and PDSCH are transmitted without any subframe gaps

```
totmtcSubframes = mpdcch.InitNSubframe+mpdcch.NRepMPDCCH+2+pdsch.NRepPDSCH;
% Identify all downlink subframes in a frame
info = arrayfun(@(x)lteDuplexingInfo(setfield(enb,'NSubframe',x)),0:9);
dlsfs = arrayfun(@(x)strcmpi(x.SubframeType,'Downlink'),info);

% Total absolute subframes to simulate
sfsnumlastFrame = getlastabsSF(dlsfs,totmtcSubframes);
totSubframes = floor(totmtcSubframes/sum(dlsfs)) *10 + sfsnumlastFrame;

% Absolute subframe number of first PDSCH subframe, this would be two
% subframes after the MPDCCH subframes (assuming no non BL/CE subframes)
lastSfForMPDCCHPlus2 = getlastabsSF(dlsfs,mpdcch.NRepMPDCCH + 2); % Last subframe number in last
pdsch.InitNSubframe = mpdcch.InitNSubframe + floor((mpdcch.NRepMPDCCH + 2)/sum(dlsfs)) *10 + las

% Find the last absolute subframe for MPDCCH transmission
lastSfForMPDCCH = getlastabsSF(dlsfs,mpdcch.NRepMPDCCH); % Last subframe number in last frame fo
mtclastabsSfForMPDCCH = mpdcch.InitNSubframe + floor((mpdcch.NRepMPDCCH)/sum(dlsfs)) *10 + lastS
```

DCI Message Encoding

The bit vector `dciBits` is passed to the function `lteDCIEncode` which performs CRC masking and insertion, tail-biting convolutional coding and rate matching, following TS 36.212 sections 5.3.3.2 to 5.3.3.4. Note that the third argument to `lteDCIEncode` specifies the rate matching output size equal to the MPDCCH data bit capacity in a subframe. Here we use a dummy 26 bit DCI message of all ones, corresponding to a DCI Format 6-1A message as specified in TS 36.212 section 5.3.3.1.12.

```
% Find the MPDCCH data bit capacity
[~,info] = lteEPDCCHIndices(enb,setfield(mpdcch,'EPDCCHPRBSet',mpdcch.InitPRBSet)); %#ok<SFLD>
% Define DCI message bits
dciBits = ones(26,1);
```

```
% Create the coded DCI bits
codedDciBits = lteDCIEncode(mpdcch,dciBits,info.EPDCCHG);
```

DL-SCH Encoding

```
% Find the LTE-M PDSCH data bit capacity, this should not include the PBCH,
% PSS, SSS and CSI-RS REs. So we select the subframe 9 which is a DL
% subframe in both FDD and TDD duplex modes (Note: subframe 9 is not DL for
% TDDConfig 0, TDDConfig 0 does not have a 'normal' DL subframe)
fullPDSCHsf = 9;

[~,fullInfo] = ltePDSCHIndices(setfield(enb,'NSubframe',fullPDSCHsf),pdsch,getPDSCHAllocation(enb
% Define DL-SCH message bits
trData = ones(pdsch.TrBlkSizes(1),1);
% Create the coded DL-SCH bits
codedTrBlock = lteDLSCH(enb,pdsch,fullInfo.G,trData);
```

MPDCCH Generation

In this example, we use the localized type with MPDCCH format 2 and 2 PRBs. MPDCCH with 2 or 4 PRBs and no repetitions use the same format table as the EPDCCH (TS 36.211 Table 6.8A.1-2). For all other cases, the formats are given by TS 36.211 Table 6.8B.1-2. Also with CE mode A, no repetitions and all LTE-M subframes, the scrambling gets initialized every subframe as for the EPDCCH and the `cinit` is the same (because `Nacc` = 1 for `CEmodeA` and `NabsMPDCCH` = 1). All other processing stages i.e. symbol modulation, layer mapping, precoding and mapping to resource elements are the same for EPDCCH and MPDCCH. Hence, we use the EPDCCH functions to generate the MPDCCH symbols and indices.

```
% Number of subframes in a scrambling block
Nacc = 1;
if strcmpi(enb.DuplexMode,'FDD') && mpdcch.NRepMPDCCH >= 4
    Nacc = 4;
elseif strcmpi(enb.DuplexMode,'TDD') && mpdcch.NRepMPDCCH >= 10
    Nacc = 10;
end

% Create a resource grid for the entire transmission. The MPDCCH, PDSCH and
% DM-RS symbols will be mapped in this array. Note that we are creating the
% grid for 4 antenna planes as the MPDCCH transmission can be sent on ports
% selected from the set (107, 108, 109 and 110)
subframeSize = lteDLResourceGridSize(enb,4);
sfgrid = zeros([subframeSize(1) subframeSize(2)*totSubframes subframeSize(3:end)]);

mpdcchSym = []; % Initialize MPDCCH symbols
mpdschSym = []; % Initialize PDSCH symbols
mpbchCoreSymFull = [];  % Initialize PBCH symbols
startSubframe = enb.NFrame*10+enb.NSubframe; % Initial absolute subframe number
for sf = startSubframe + (0:totSubframes -1)

    % Set current absolute subframe and frame numbers
    enb.NSubframe = mod(sf,10);
    enb.NFrame = floor((sf)/10);

    % Skip processing if this is not a downlink subframe
    duplexInfo = lteDuplexingInfo(enb);
    if ~strcmpi(duplexInfo.SubframeType,'Downlink')
        continue
    end
```

```
% Transmitting the MPDCCH
if (sf >= mpdcch.InitNSubframe) && (sf < mtclastabsSfForMPDCCH)
    % Calculate the PRBSet used in the current subframe
    prbset = getHoppingAllocation(enb,mpdcch);

    % Calculate the MPDCCH indices for the current subframe
    mpdcch.EPDCCHPRBSet = prbset;
    [mpdcchIndices,info] = lteEPDCCHIndices(enb,mpdcch);

    % Create an empty subframe grid
    subframe = lteDLResourceGrid(enb,4);

    % Encode MPDCCH symbols from DCI codeword
    % In the case of repetition, the same symbols are repeated in each of
    % a block of NRepMPDCCH subframes. Frequency hopping is applied as required
    if ~mod(sf,Nacc) || isempty(mpdcchSym)
        mpdcchSym = lteEPDCCH(enb,mpdcch,codedDciBits)*db2mag(mpdcch.MPDCCHPower);
    end
    % Map MPDCCH symbols to the subframe grid
    subframe(mpdcchIndices) = mpdcchSym;

    % Create MPDCCH DM-RS
    % The MPDCCH and its reference symbols are transmitted on the same
    % port(s) and is transmitted only on the PRBs in which the
    % corresponding MPDCCH is mapped. The DM-RS sequence is the same as for
    % the EPDCCH as given by the equations in TS 36.211 section 6.10.3A
    mpdcchDMRS = lteEPDCCHDMRS(enb,mpdcch)*db2mag(mpdcch.MPDCCHDMRSPower); % MPDCCH DM-RS sy
    mpdcchDMRSIndices = lteEPDCCHDMRSIndices(enb,mpdcch); % MPDCCH DM-RS indices
    subframe(mpdcchDMRSIndices) = mpdcchDMRS;   % Map DM-RS signals to the grid

    % Now assign the current subframe into the overall grid
    sfgrid(:,(1:subframeSize(2))+sf*subframeSize(2),:) = subframe;
end
```

BL/CE PDSCH Generation

For the LTE-M PDSCH, the PBCH, PSS, SSS and CSI-RS RE locations are counted in the mapping, but not used in the transmission. This means that the rate matched capacity should include these RE locations, but the symbols for these locations are not transmitted. So we create the codeword accordingly and puncture the symbols corresponding to the conflicting locations

```
if (sf >= pdsch.InitNSubframe)

    % Calculate the PRBSet used in the current subframe
    prbset = getPDSCHAllocation(enb,pdsch);

    % Calculate the PDSCH indices for the current subframe
    pdsch.PRBSet = prbset;
    mpdschIndices = ltePDSCHIndices(enb,pdsch,pdsch.PRBSet);
    % If the subframe contains PBCH, PSS, SSS, CSI-RS or Zero Power
    % CSI-RS REs, then we need to puncture the corresponding symbols
    % from the mapping but the rate matching should be to the full
    % PDSCH capacity ignoring the possible presence of these symbols.
    % This is done by setting the subframe and TDDConfig (if TDD mode)
    % to a subframe containing no PBCH, PSS and SSS and turning off the
    % CSI-RS and ZP CSI-RS. The full set of possible PDSCH indices are
    % recalculated every subframe as these can change when frequency
```

```
        % hopping
        enbTemp = enb;
        enbTemp.TDDConfig = 1;              % TDDConfig 0 has no full PDSCH subframe
        enbTemp.NSubframe = fullPDSCHsf;    % Set the subframe to a full PDSCH subframe
        enbTemp.CSIRSPeriod = 'Off';        % CSI-RS period control
        enbTemp.ZeroPowerCSIRSPeriod = 'Off'; % Zero power CSI-RS period control
        mpdschIndicesFull = ltePDSCHIndices(enbTemp,pdsch,pdsch.PRBSet);
        [~, txmpdschIndicesPositions] = intersect(mpdschIndicesFull,mpdschIndices);

        % Create an empty subframe grid
        subframe = lteDLResourceGrid(enb,4);

        % Encode PDSCH symbols from the codeword
        % In the case of repetition, the same symbols are repeated in each of
        % a block of NRepPDSCH subframes. Frequency hopping is applied as required
        if ~mod(sf,Nacc) || isempty(mpdschSym)
            mpdschSym = ltePDSCH(enb,pdsch,codedTrBlock)*db2mag(pdsch.Rho);
        end
        % Map punctured PDSCH symbols to the subframe grid
        subframe(mpdschIndices) = mpdschSym(txmpdschIndicesPositions);

        % Transmit UE-specific reference signal (DM-RS) if applicable
        if any(strcmpi(pdsch.TxScheme,{'Port5' 'Port7-8' 'Port8' 'Port7-14'}))
            ueRSIndices = lteDMRSIndices(enb,pdsch);
            ueRSSymbols = lteDMRS(enb,pdsch);
            subframe(ueRSIndices) = ueRSSymbols*db2mag(pdsch.RSPower);  % Map symbols to the gric
        end

        % Now assign the current subframe into the overall grid
        sfgrid(:,(1:subframeSize(2))+sf*subframeSize(2),:) = subframe;
    end
```

BL/CE PBCH Generation

PBCH symbol generation and mapping on REs of a resource grid. LTE-M PBCH consists of the normal LTE 'core' part and the LTE-M specific repetitions. Core PBCH symbols are only present in first subframe with a periodicity of 4 frames.

```
    subframe = sfgrid(:,(1:subframeSize(2))+sf*subframeSize(2),:);
    if(mod(enb.NSubframe,10)==0)
        % Generate symbols if its the first simulated frame or
        % when mod(NFrame,4) is 0;
        if ~mod(enb.NFrame,4) || isempty(mpbchCoreSymFull)
            mpbchCoreSymFull = getMPBCHCore(enb);
        end
        mpbchCoreIndices = ltePBCHIndices(enb,{'1based'});
        % Now extract out the core part for the Frame
        mpbchCoreSym = mpbchCoreSymFull(:,mod(enb.NFrame,4)+1);
        % Map the PBCH core symbols to the subframe
        subframe(mpbchCoreIndices) = mpbchCoreSym*db2mag(enb.PBCHPower);

        % Now assign the current subframe into the overall grid
        sfgrid(:,(1:subframeSize(2))+sf*subframeSize(2),:) = subframe;

        % Get the cell RS symbols and indices to be repeated if PBCH
        % repetitions are enabled
        [mpbchCoreCellRSSymbols,mpbchCoreCellRSIndices] = getPBCHCoreCellRS(enb);
```

```
    elseif enb.RepPBCHEnable && strcmpi(enb.DuplexMode,'FDD') && (mod(enb.NSubframe,10)==9)
        % If this is the 9th subframe in FDD mode, then create the core
        % symbols and indices for the next frame to be used for PBCH
        % repetitions in this subframe
        enbNext = enb;
        enbNext.NSubframe = 0;
        enbNext.NFrame = enbNext.NFrame+1; % Advance to the next frame
        % if the current frame contained the last PBCH block, then we need
        % the new set of PBCH symbols
        if mod(enb.NFrame,4)==3
            mpbchCoreSymFull = getMPBCHCore(enbNext);
        end
        % Now extract out the core part for the Frame
        mpbchCoreSym = mpbchCoreSymFull(:,mod(enbNext.NFrame,4)+1);
        mpbchCoreIndices = ltePBCHIndices(enbNext,{'1based'});
        [mpbchCoreCellRSSymbols,mpbchCoreCellRSIndices] = getPBCHCoreCellRS(enbNext);
    end
    % PBCH repetition part if enabled
    if (enb.RepPBCHEnable)
        % Get the PBCH repetition part consisting of repeating PBCH symbols
        % and repeating Cell RS signals and corresponding indices
        [pbchrepSymbols, pbchrepIndices, pbchCellRSrepSymbols, pbchCellRSrepIndices] = getPBCHRep
        % Map the PBCH repetitions to the grid
        subframe(pbchrepIndices) = pbchrepSymbols*db2mag(enb.PBCHRepsPower);
        % Map the Cell RS repetitions to the grid
        subframe(pbchCellRSrepIndices) = pbchCellRSrepSymbols*db2mag(enb.PBCHCellRSRepsPower);
    end

    % Now assign the current subframe into the overall grid
    sfgrid(:,(1:subframeSize(2))+sf*subframeSize(2),:) = subframe;

end
```

Create Time Domain Baseband Waveform

Create the time domain baseband waveform by OFDM modulating the resource grid. The resulting matrix has four columns; one of which will contain the complex baseband time-domain waveform samples for the MPDCCH

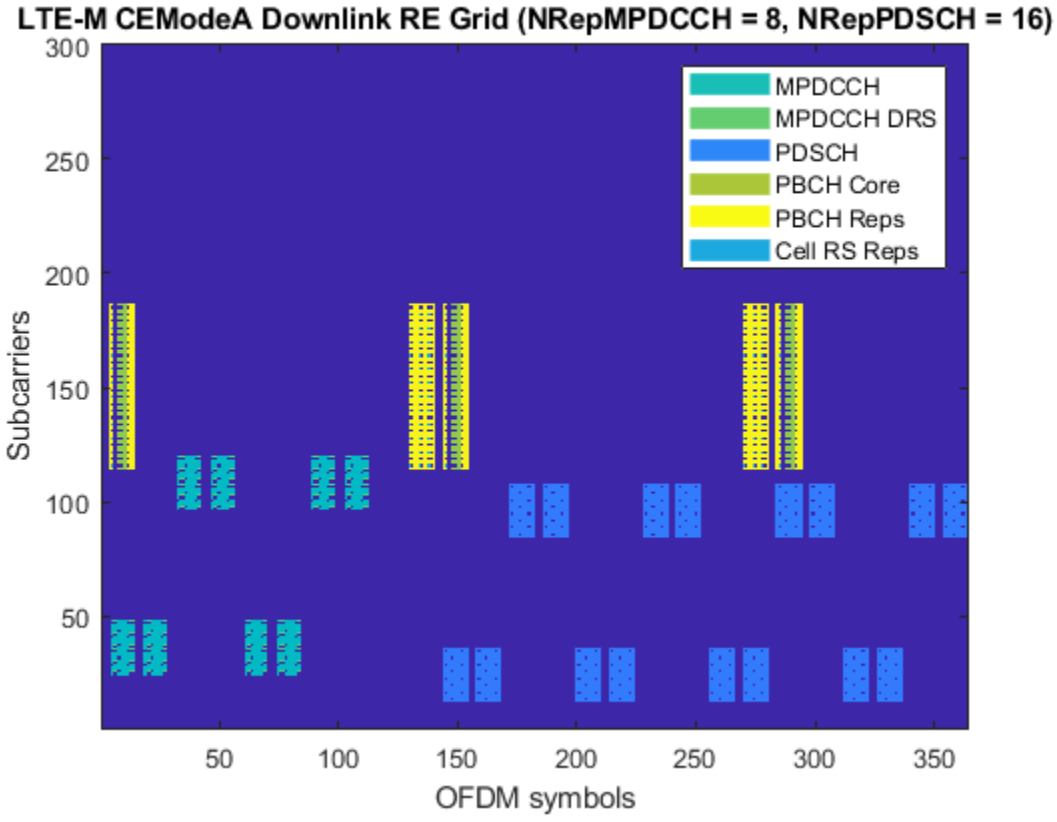
```
waveform = lteOFDMModulate(enb,sfgrid);
```

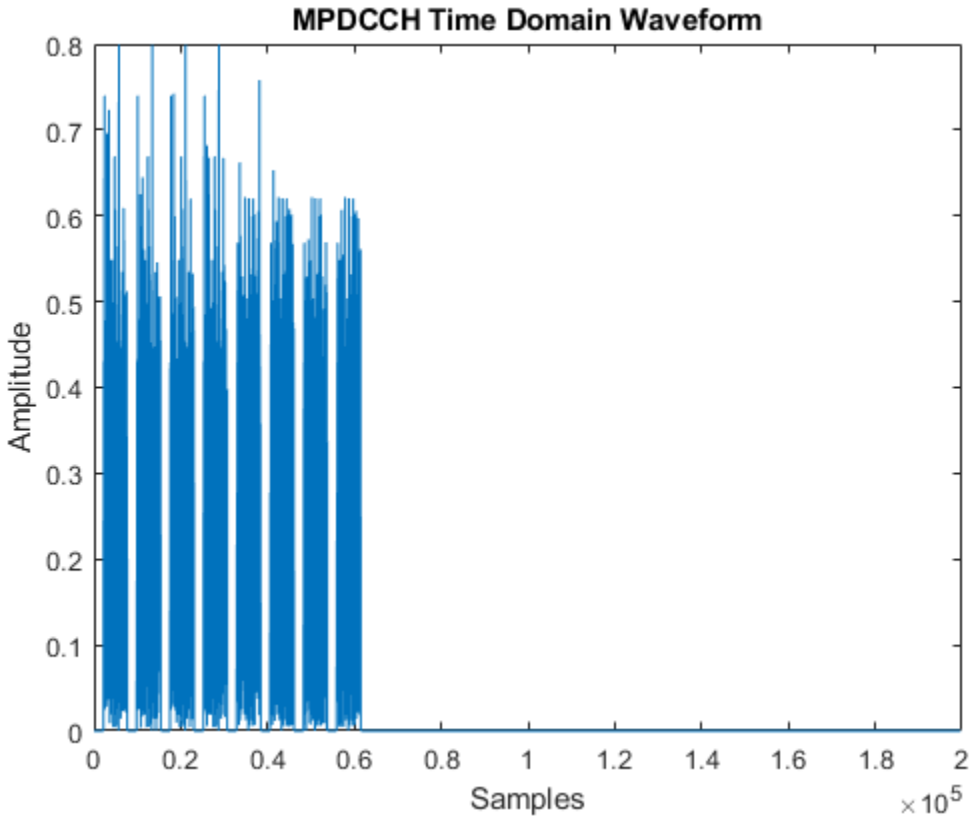
Plot Transmitted Grid and Baseband Waveform

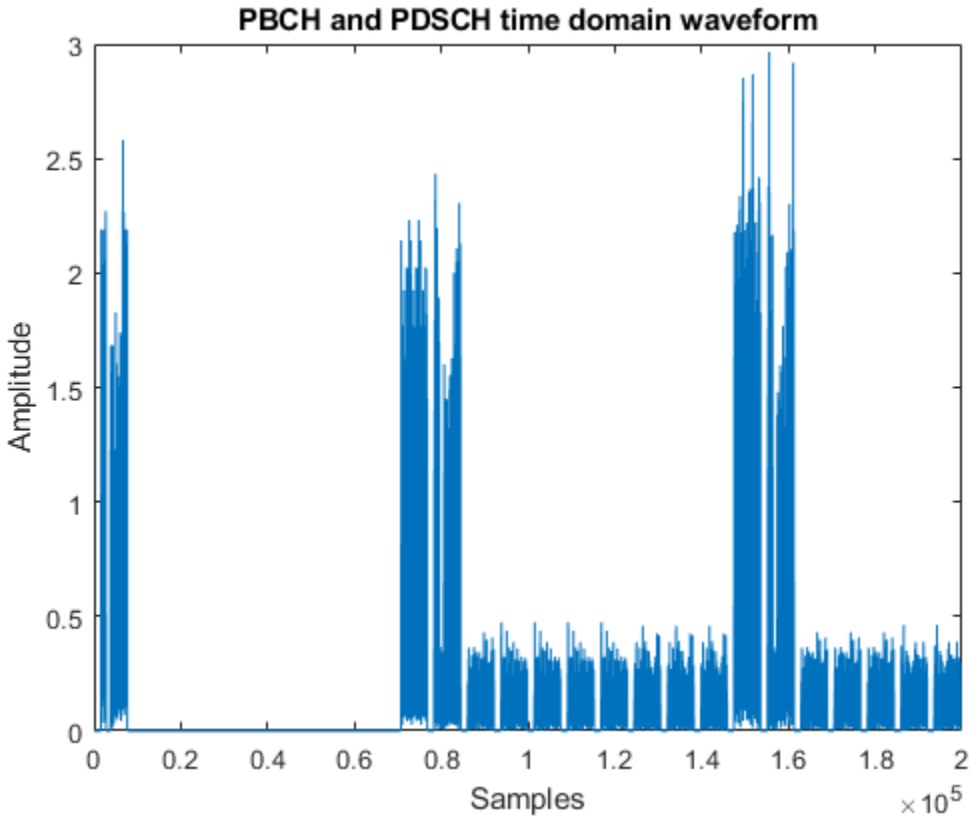
Plot the grid and time domain baseband waveform. If the transmission uses more than one port, only the first port is shown. Note that the resource grid plot uses the power levels of the individual channels and signals to assign colors to the resource elements.

```
% Create an image of overall resource grid
% Plot the port used by MPDCCH along with the first port for all other channels
figure
im = image(abs(sfgrid(:,:,info.EPDCCHPorts(1))+ sfgrid(:,:,1)));
cmap = parula(64);
colormap(im.Parent,cmap);
axis xy;
title(sprintf('LTE-M CEMode%s Downlink RE Grid (NRepMPDCCH = %d, NRepPDSCH = %d)',pdsch.CEMode,m
xlabel('OFDM symbols')
ylabel('Subcarriers')
% Create the legend box to indicate the channel/signal types associated with the REs
```

```
reNames = {'MPDCCH';'MPDCCH DRS';'PDSCH';'PBCH Core'; 'PBCH Reps'; 'Cell RS Reps'};
clevels = round(db2mag([mpdcch.MPDCCHPower mpdcch.MPDCCHDMRSPower  pdsch.Rho enb.PBCHPower enb.PB
% If using DM-RS, include in legend
if any(strcmpi(pdsch.TxScheme,{'Port5' 'Port7-8' 'Port8' 'Port7-14'}))
    reNames{end+1} = 'DMRS';
    clevels(end+1) = pdsch.RSPower;
end
N = numel(reNames);
L = line(ones(N),ones(N), 'LineWidth',8); % Generate lines
% Set the colors according to cmap
set(L,{'color'},mat2cell(cmap( min(1+clevels,length(cmap) ),:),ones(1,N),3));   % Set the colors
legend(reNames{:});

% Create separate plots of the control/data waveforms
figure
plot(abs(waveform(:,info.EPDCCHPorts(1))))
title('MPDCCH Time Domain Waveform')
xlabel('Samples')
ylabel('Amplitude')
figure
plot(abs(waveform(:,1)))
title('PBCH and PDSCH time domain waveform')
xlabel('Samples')
ylabel('Amplitude')
```




MPDCCH Time Domain Waveform
Amplitude
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
Samples
×10^5



Local Functions

The following local functions are used in this example:

- `calcNarrowbandPRBSets` - Calculate narrowbands and associated PRBs
- `getHoppingAllocation` - Calculate the subframe allocation
- `getPDSCHAllocation` - Calculate the PDSCH allocation
- `getPBCHCore` - Calculate the core PBCH symbols and indices
- `getPBCHCoreCellRS` - Calculate the core PBCH symbols and indices
- `getPBCHRep` - Calculate the PBCH repetition part

Selected Bibliography

1 3GPP TS 36.211 "Physical channels and modulation"

2 3GPP TS 36.212 "Multiplexing and channel coding"

3 3GPP TS 36.213 "Physical layer procedures"

4 3GPP TS 36.331 "Radio Resource Control (RRC) Protocol specification"

5 O. Liberg, M. Sundberg, Y.-P. Wang, J. Bergman and J. Sachs, Cellular Internet of Things: Technologies, Standards and Performance, Elsevier, 2018.

6 E. Dahlman, S. Parkvall and J Skold 4G LTE-Advanced Pro and The Road to 5G

Local Functions

```
% Calculate the widebands, narrowbands and PRBSets for the LTE carrier bandwidth
function [prbsets,nNB,nWB] = calcNarrowbandPRBSets(NDLRB)
    % Narrowbands & Widebands (See 36.211 section 6.2.7)
    NDLNB = floor(NDLRB/6);
    nNB = 0:(NDLNB-1); % Narrowbands
    if NDLNB >= 4
        NDLWB = floor(NDLNB/4);
    else
        NDLWB = 1;
    end
    nWB = 0:(NDLWB-1); % Widebands

    % PRBs in a narrowband
    ii = 0:5;
    ii0 = floor(NDLRB/2) - 6*(NDLNB/2);
    prbsets = zeros(6,numel(nNB));
    for nb = 1:numel(nNB)
        if mod(NDLRB,2) && nNB(nb)>= (NDLNB/2)
            prbsets(:,nb) = 6*(nNB(nb))+ii0+ii + 1;
        else
            prbsets(:,nb) = 6*(nNB(nb))+ii0+ii;
        end
    end
end

% Calculate the resource blocks allocated in the hopping narrowband
function prbset = getHoppingAllocation(enb,chs)

    % If frequency hopping is disabled, the allocation is same as initial
    if ~chs.Hopping
        prbset = chs.InitPRBSet;
        return
    end

    % Hopping narrowband calculation according to TS 36.211 section 6.8B.5
    nNBi0ss = 0;
    % Get the possible narrowbands and associated PRBSets
    if strcmpi(enb.DuplexMode,'FDD')
        idelta = 0;
    else
        idelta = enb.NChDLNB-2;
    end
    j0 = floor((chs.InitNSubframe+idelta)/enb.NChDLNB);
    % Get the narrowbands and corresponding resources
    [prbsets,nNB] = calcNarrowbandPRBSets(enb.NDLRB);
    % Calculate the narrowband for this subframe
    enb.NSubframe = enb.NFrame*10+enb.NSubframe; % Get the absolute subframe number
    nnBi = mod( (nNBi0ss + (mod(floor((enb.NSubframe+idelta)/enb.NChDLNB - j0),enb.NChDLNBhop))*e
    % Calculate the PRBSet for this subframe, they are on the same RBs
    % within the narrowband
    [rbstartIndex,nbstartIndex] = find(prbsets == chs.InitPRBSet(1));
    [rbendIndex,nbendIndex] = find(prbsets == chs.InitPRBSet(end));
    if (isempty(rbstartIndex) || isempty(rbendIndex)) || (nbstartIndex ~= nbendIndex)
       error('Invalid PRBSet specified, must be resources within single narrowband');
    end
    prbset = prbsets(rbstartIndex:rbstartIndex+numel(chs.InitPRBSet)-1,nnBi+1);
```

```
end

% Calculate the PDSCH allocation
function prbset = getPDSCHAllocation(enb,PDSCH)
    if PDSCH.Hopping
        % Cat-M1 mode with hopping
        prbset = getHoppingAllocation(enb,PDSCH);
    else
        % Calculate the allocations in narrowband(s)
        [prbsets,nNB] = calcNarrowbandPRBSets(enb.NDLRB);
        % Calculate the PRBSet for this subframe, they are on the same RBs
        % within all narrowbands
        rbstartIndex = mod(find(prbsets == PDSCH.InitPRBSet(1))-1,6)+1;
        rbendIndex = mod(find(prbsets == PDSCH.InitPRBSet(end))-1,6)+1;
        if isempty(rbstartIndex) || isempty(rbendIndex)
           error('Invalid PRBSet specified, must be resources within narrowbands');
        end
        if any(PDSCH.InitNarrowbandIndex > max(nNB))
           error('Invalid InitNarrowbandIndex specified, must be from the set 0...%d',max(nNB));
        end
        prbset = prbsets(rbstartIndex:rbendIndex,PDSCH.InitNarrowbandIndex+1);
        prbset = prbset(:);

    end

end

% Calculate the Cell RS REs and symbols corresponding to the 'core' PBCH
% part
function [pbchCoreCellRSSymbols,pbchCoreCellRSIndices] = getPBCHCoreCellRS(enb)

    % We need to repeat the cell reference signals within the (k,l) region
    NscRB = 12;
    k = (enb.NDLRB*NscRB)/2 -36 + (0:71) + 1; % 1-based full possible PBCH subcarrier locations
    if strcmpi(enb.CyclicPrefix,'Normal')
        NsymbDL = 7;
    else
        NsymbDL = 6;
    end
    l = NsymbDL+ (0:3)+ 1; % 1-based OFDM symbol numbers in the subframe corresponding to PBCH co

    % NOTE: The cell RS symbols and indices can be looked up from the
    % grid if provided or can be created here as shown below
    cellRSIndices = lteCellRSIndices(enb);
    cellRSSymbols = lteCellRS(enb);
    rsgrid = lteDLResourceGrid(enb);
    rsgrid(cellRSIndices) = cellRSSymbols;
    % Now remove all RS symbols outside of the core PBCH band
    excludeSubs = setdiff(1:enb.NDLRB*NscRB,k);
    excludeofdmSymbols = setdiff(1:NsymbDL*2,l);
    % Now remove all the unwanted RE locations
    rsgrid(:,excludeofdmSymbols,:) = 0;
    rsgrid(excludeSubs,:,:) = 0;

    % What is in the grid is the RS symbols to be repeated
    pbchCoreCellRSIndices = find(rsgrid);
    pbchCoreCellRSSymbols = rsgrid(pbchCoreCellRSIndices);
```

```
end

% Calculate the PRBCH and Cell RS REs and symbols corresponding to the
% repetition part
function [pbchrepSymbols,pbchrepIndices, pbchCellRSrepSymbols, pbchCellRSrepIndices] = getPBCHRep
    pbchrepIndices = [];
    pbchrepSymbols = [];
    symMappings = {};
    pbchCellRSrepIndices = [];
    pbchCellRSrepSymbols = [];

    % For both FDD and TDD modes, there is no repetition if NDLRB = 6
    if enb.NDLRB==6
        return
    end

    % Get the subcarriers for the PBCH core part
    [pbchCoreSubcarriers,ofdmSymbols,~] = ind2sub(lteDLResourceGridSize(enb),pbchCoreIndices);
    % Get the subcarriers for the Cell RS core part
    [pbchCellRSCoreSubcarriers,ofdmSymbolsCellRS,~] = ind2sub(lteDLResourceGridSize(enb),pbchCore

    % FDD reps only in subframes 9 (frame n-1) and 0 (frame n) and TDD only
    % reps in subframes 0 and 5 in the same frame
    % Get the subframe symbol numbers in which the reps go onto
    if strcmpi(enb.DuplexMode,'FDD')
        if strcmpi(enb.CyclicPrefix,'Normal')
            NsymbDL = 7;
            if (mod(enb.NSubframe,10) == 0)
                symMappings = { 5 ;
                                12 ;
                                13 ;
                                [4 14]};
            elseif (mod(enb.NSubframe,10)==9)
                symMappings = { [4 8 12] ;
                                 [5 9 13] ;
                                 [6 10 14] ;
                                 [7 11]};
            end
        else
            NsymbDL = 6;
            if (mod(enb.NSubframe,10) == 0)
                symMappings = {[] ;
                               4  ;
                               11 ;
                               12};
            elseif (mod(enb.NSubframe,10)==9)
                symMappings = { [4 7] ;
                                 [5 8] ;
                                 [6 9] ;
                                 [10 11]};
            end
        end

    else
        if strcmpi(enb.CyclicPrefix,'Normal')
            NsymbDL = 7;
            if (mod(enb.NSubframe,10) == 0)
```

```
                symMappings = { [4 12]  ;
                                [5 13] ;
                                6 ;
                                7};
            elseif (mod(enb.NSubframe,10)==5) && enb.NDLRB>15
                symMappings = { [4 8 12] ;
                                [5 9 13] ;
                                [6 10] ;
                                [7 11]};
            end
        else
            NsymbDL = 6;
            if (mod(enb.NSubframe,10) == 0)
                symMappings = {4 ;
                               5 ;
                               6 ;
                               11};
            elseif (mod(enb.NSubframe,10)==5) && enb.NDLRB>15
                symMappings = { [4 7] ;
                                [5 8] ;
                                [6 9] ;
                                [10 11]};
            end
        end

    end

    % If this is a repetition subframe, find the indices
    if ~isempty(symMappings)

        % Create an empty subframe grid
        sfgrid = lteDLResourceGrid(enb);
        sfgridRS = lteDLResourceGrid(enb);
        for osymb = 1:4 % For all 4 PBCH symbols
            % Extract out each core symbol to map to one or more OFDM
            % symbols
            coreOFDMsymb = pbchCoreSymbols(ofdmSymbols==(osymb+NsymbDL));
            coreOFDMsymbRS = pbchCoreCellRSSymbols(ofdmSymbolsCellRS==(osymb+NsymbDL));

            % Map to all new repetition OFDM symbols in the current
            % subframe
            for m = 1:numel(symMappings{osymb,1})
                % create the indices for the current symbol (subcarriers
                % are the same as core symbols)
                symtoMap = symMappings{osymb,1}(m);
                sfgrid(pbchCoreSubcarriers(ofdmSymbols==(osymb+NsymbDL)),symtoMap,:) = coreOFDMsy
                sfgridRS(pbchCellRSCoreSubcarriers(ofdmSymbolsCellRS==(osymb+NsymbDL)),symtoMap,

            end

        end

        % As per TS 36.211 section 6.6.4, the PBCH repetitions and cell RS
        % repetitions are punctured by CSI (& ZP CSI) RS signals. So clear
        % these positions
        csirsIndices = lteCSIRSIndices(enb);
        sfgrid(csirsIndices) = 0;
        sfgridRS(csirsIndices) = 0;
```

```
            % Now get the PBCH repeating symbols and indices
            pbchrepIndices = find(sfgrid);
            pbchrepSymbols = sfgrid(pbchrepIndices);
            % Now get the CellRS repeating symbols and indices
            pbchCellRSrepIndices = find(sfgridRS);
            pbchCellRSrepSymbols = sfgridRS(pbchCellRSrepIndices);

        end
    end

    % Get the MPBCH Core part symbols
    function mpbchCoreSym = getMPBCHCore(enb)

        bchBits = lteMIB(enb);
        pbchBits = lteBCH(enb,bchBits);
        mpbchCoreSym = ltePBCH(enb,pbchBits);
        % Reshape to the four parts to go on to 4 frames
        mpbchCoreSym = reshape(mpbchCoreSym,numel(mpbchCoreSym)/4,4);
    end

    % Get the absolute subframe number in a frame which is used for the last
    % transmission of a channel
    function sfsnumlastFrame = getlastabsSF(dlsfs,totmtcSubframes)
        sfslastFrame = mod(totmtcSubframes,sum(dlsfs)); % subframes to tx in the last frame
        if sfslastFrame
            % Find the subframe number corresponding to the last subframe to transmit
            sfsnumlastFrame = find(dlsfs,sfslastFrame);
            sfsnumlastFrame = sfsnumlastFrame(end);
        else
            % No partial frames required
            sfsnumlastFrame = 0;
        end

    end
```

NB-IoT NTN NPDSCH Throughput

This example shows how to perform a narrowband Internet of Things (NB-IoT) narrowband physical downlink shared channel (NPDSCH) throughput simulation in a non-terrestrial network (NTN) channel. This example supports these two narrowband NTN channels.

- European Telecommunication Standards Institute (ETSI) Rician fading channel
- International Telecommunication Union Radiocommunication Sector (ITU-R) P.681 land mobile-satellite (LMS) channel

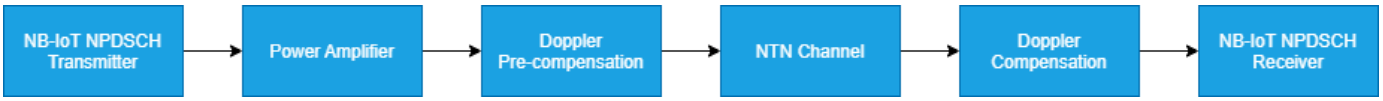
Introduction

This example measures the NPDSCH throughput of an NB-IoT link, as defined by the 3GPP NB-IoT standards [3], [4], [5], [6], and [7].

The example models these features.

- Transport channel coding
- NPDSCH, narrowband reference signal (NRS), and synchronization signals (narrowband primary synchronization signal NPSS, narrowband secondary synchronization signal NSSS)
- ETSI Rician channel and ITU-R P.681 LMS channel
- Single-input-single-output (SISO) link
- Doppler pre-compensation at the transmitter, and Doppler compensation at the receiver
- Optional power amplifier modeling

The figure shows the implemented processing chain. For clarity, the figure does not show the NRS and synchronization signals.



For more information about the NB-IoT NPDSCH transmitter and receiver processing units, see the “NB-IoT NPDSCH Block Error Rate Simulation” on page 2-304 example.

To reduce the total simulation time, you can use Parallel Computing Toolbox™ to execute the range of transmit power values of the transmit power loop in parallel.

Configure Simulation Length, Transmitter, and Receiver

Set the length of the simulation in terms of the number of transport blocks. By default, the example uses 10 transport blocks, but you must use a large number of transport blocks to produce meaningful throughput results. Set the range of transmit power values to simulate. The transmitter power is defined as the power of the time-domain waveform before passing to the power amplifier. The receiver includes its noise figure and the antenna temperature. The noise figure models the receiver internal noise, and the antenna temperature models the input noise. This receiver specifies the noise per antenna element. In this example, you perform the simulation for different repetition values and compare the performance improvement with repetitions. The `iReps` variable is applicable only when NPDSCH does not carry the scheduling information block SIB1-NB.

```
numTrBlks = 10;             % Number of simulated transport blocks
iReps = [0 5];              % Range of repetitions simulated
```

```
txPower = 30:5:45;          % Transmit power (dBm)
rxNoiseFigure = 6;          % Noise figure (dB)
rxAntennaTemperature = 290; % Antenna temperature (K)
```

Power Amplifier Configuration

To configure a memoryless power amplifier nonlinearity, use `enablePA`. You can choose one of these power amplifier models, as defined in Annex A of TR 38.803.

- 2.1 GHz Gallium Arsenide (GaAs)
- 2.1 GHz Gallium Nitride (GaN)

Alternatively, you can set `paModel` to `Custom` and use `paCharacteristics` to define the power amplifier characteristics in a matrix with three columns. The first column defines the input power in dBm. The second column defines the output power in dBm. The third column defines the output phase in degrees. When the `paCharacteristics` variable is set to empty and the `paModel` is set to `Custom`, this example uses a 2.1 GHz laterally-diffused metal-oxide semiconductor (LDMOS) Doherty-based amplifier.

The memoryless nonlinearity applied to the waveform follows this equation for power amplifiers (excluding a custom configuration).

$$y_P(n) = \sum_{k \in K_p} a_k \, x(n) \, |x(n)|^{2k}$$

In this equation,

- $y_P(n)$ is the output signal.
- $x(n)$ is the input signal.
- K_p is the set of polynomial degree(s).
- a_k is the polynomial coefficient.

By default, the example sets `enablePA` to `false`.

```
enablePA = ;                % true or false
paModel = 2.1GHz GaAs; % "2.1GHz GaAs", "2.1GHz GaN", or "Custom"
paCharacteristics = [];     % Lookup table as empty or a matrix with columns: Pin (dBm) | Pou

% If enablePA is set to true, visualize the power amplifier gain and phase
% characteristics
paModelImpl = paModel;
if enablePA == 1
    % Set the power amplifier as applicable for the further processing
    if lower(paModel) == "custom"
        if isempty(paCharacteristics)
            tableLookup = getDefaultCustomPA;
        else
            tableLookup = paCharacteristics;
        end
        % Use table look-up option of comm.MemorylessNonlinearity and provide
        % the power amplifier characteristics
        mnl = comm.MemorylessNonlinearity(Method="Lookup table", ...
            Table=tableLookup);
```

```
        plot(mnl)
        paModelImpl = mnl;
    else
        paMemorylessNonlinearity(paModel)
    end
end
```

Doppler Compensation Configuration

The example supports two Doppler compensation configurations: one at the transmitter end and the other at the receiver end. For compensation at the transmitter end, enable `txDopplerCompensator`. Setting the `txDopplerCompensator` variable to `true` pre-compensates the transmitted waveform for Doppler caused by the satellite movement. For compensation at the receiver end, enable `rxDopplerCompensator`. Setting the `rxDopplerCompensator` variable to `true` performs Doppler compensation at the receiver using NPSS. Note that, in the case of Doppler compensation at the receiver, if the operating SNR is below -10 dB, the Doppler estimates can be inaccurate.

```
txDopplerCompensator = ; % true or false
rxDopplerCompensator = ; % true or false
```

Set Up Higher Layer Parameters

To configure the eNodeB and NPDSCH parameters, set these higher layer parameters.

```
npdschDataType = NotBCCH; % "SIB1NB", "BCCHNotSIB1NB", or "NotBCCH"
iSF = 0;                  % Resource assignment field in DCI (DCI format N1 or N2)
schedulingInfoSIB1 = 0;   % Scheduling information field in MasterInformationBlock-N
iMCS = 4;                 % Modulation and coding scheme field in DCI (DCI format N1
```

eNodeB and NPDSCH Configuration

Set these eNodeB parameters.

- NB-IoT physical layer cell identity
- Operation mode

```
enb = struct;
enb.NNCellID = 0;                    % NB-IoT physical layer cell identity
enb.OperationMode = Standalone; % "Standalone", "Guardband", "Inband-SamePCI", or "In
```

Set the radio network temporary identifier in the NPDSCH structure.

```
npdsch = struct;
npdsch.RNTI = 1; % Radio network temporary identifier
```

Propagation Channel Model Configuration

Create a channel model object for the simulation. The example supports the ETSI Rician channel and ITU-R P.681 LMS channel. To obtain the NTN channel from these channels, apply an additional Doppler shift. Use this equation to calculate the Doppler shift due to satellite movement, as specified in TR 38.811.

$$f_{d,\mathrm{sat}} = \left(\frac{v_{\mathrm{sat}}}{c}\right) * \left(\frac{R}{R+h}\cos(\alpha_{\mathrm{model}})\right) * f_c$$

In this equation,

- v_{sat} is the satellite speed.
- c is the speed of light.
- R is the Earth radius.
- h is the satellite altitude.
- α_{model} is the satellite elevation angle.
- f_c is the carrier frequency.

By default, this example considers a satellite in low Earth orbit at an altitude of 600 km, with a carrier frequency of 2 GHz. The NB-IoT user equipment (UE) is moving at a speed of 3 km/h. Also, this example assumes that satellites move in circular orbits. Verify that the satellite speed and satellite altitude are in agreement with each other.

```
channel = struct;
channel.NTNChannelType = 'ETSI Rician'; % "ETSI Rician" or "ITU-R P.681"
channel.CarrierFrequency = 2e9;                % Carrier frequency (in Hz)
channel.ElevationAngle = 50;                   % Elevation angle (in degrees)
channel.MobileSpeed = 3*1000/3600;             % UE speed (in m/s)
channel.SatelliteSpeed = 7562.2;               % Satellite speed (in m/s)
channel.SatelliteAltitude = 600e3;             % Satellite altitude (in m)
channel.Seed = 73;                             % Random seed

channel.IncludeFreeSpacePathLoss = 1;          % Include or exclude free space path loss

% Set these fields based on the type of channel selected
if lower(channel.NTNChannelType) == "etsi rician"
    % For ETSI Rician channel, set KFactor
    channel.KFactor = 10;           % In dB
else
    % For ITU-R P.681, set Environment and AzimuthOrientation
    channel.Environment = "Urban";  % "Urban", "Suburban", "RuralWooded", or "Residential"
    channel.AzimuthOrientation = 0; % In degrees
end
```

Channel Estimator Configuration

Configure a practical channel estimator by using the `cec` structure. By default, this example configures the channel with these specifications.

- Carrier frequency — 2 GHz
- Speed of NB-IoT UE — 3 km/h

This configuration results in a Doppler spread of 5.5 Hz. Therefore, perform frequency averaging over pilot estimates with these settings.

- Time window — 1 resource element (RE)
- Frequency window — 25 REs, to ensure averaging over all subcarriers for the resource block

```
% Configure channel estimator
cec.PilotAverage = "UserDefined";   % Type of pilot symbol averaging
cec.TimeWindow = 1;                 % Time window size in REs
cec.FreqWindow = 25;                % Frequency window size in REs
```

```
cec.InterpType = "Cubic";           % 2-D interpolation type
cec.InterpWindow = "Centered";      % Interpolation window type
cec.InterpWinSize = 3;              % Interpolation window size
cec.Reference = "NRS";              % Channel estimator reference signal
```

Processing Loop

To determine the throughput at each repetition index and SNR index, follow these steps.

1. **Generate the transport block** — Get the transport block size depending on the configured higher layer parameters.
2. **Generate the resource grid** — Map the modulated bits, along with the NPSS, NSSS, and NRS, signals to the resource grid. The `lteNDLSCH` function performs transport channel coding on the input transport block. The `lteNPDSCH` function then modulates the encoded data bits.
3. **Generate the waveform** — Generate the NB-IoT time-domain OFDM waveform with half subcarrier shift using the `lteSCFDMAModulate` function.
4. **Apply power amplifier nonlinearities** — Apply the memoryless nonlinearities to the baseband OFDM signal.
5. **Apply Doppler pre-compensation** — Apply the Doppler shift due to satellite movement to the generated waveform to pre-compensate the channel-induced satellite Doppler shift.
6. **Model and apply a noisy channel** — Pass the generated waveform through an ETSI Rician or ITU-R P.681 LMS fading channel to get the faded waveform. Apply path loss and add thermal noise to the faded waveform.
7. **Apply Doppler compensation** — Estimate the Doppler shift in the received waveform, and compensate the Doppler shift.
8. **Perform synchronization and OFDM demodulation** — Perform timing synchronization by correlating the received waveform with the NPSS. The `lteSCFDMADemodulate` function then demodulates the synchronized signal.
9. **Perform channel estimation** — Estimate the channel using NRS.
10. **Decode the NPDSCH** — Decode the NPDSCH, with the estimated channel and noise variance, by using the `lteNPDSCHDecode` function.
11. **Decode the transport block** — Decode the soft bits using the `lteNDLSCHDecode` function. The function decodes the codeword data and returns the block cyclic redundancy check (CRC) error.

```
% Use the higher layer parameters, and check if the provided configuration
% is valid
numRep = numel(iReps);
npdschInfo = hNPDSCHInfo;
npdschInfo.NPDSCHDataType = npdschDataType;
npdschInfo.ISF = iSF;
npdschDataTypeLower = lower(npdschDataType);
if npdschDataTypeLower == "sib1nb"  % NPDSCH carrying SIB1-NB
    npdschInfo.SchedulingInfoSIB1 = schedulingInfoSIB1;
    % Store a copy of the information structure for all the repetitions
    npdschInfo = repmat(npdschInfo,numRep,1);
else % NPDSCH not carrying SIB1-NB
    npdschInfo.IMCS = iMCS;              % Modulation and coding scheme field in DCI (DCI format
    % Store a copy of the information structure for all the repetitions
    npdschInfo = repmat(npdschInfo,numRep,1);
    for repIdx = 1:numRep
        npdschInfo(repIdx).IRep = iReps(repIdx); % Repetition number field in DCI (DCI format N1
    end
```

```
end

% Initialize some parameters of enb
enb.NFrame = 0;
enb.NSubframe = 0;
enb.NBRefP = 1;
opMode = lower(enb.OperationMode);
inbandSamePCI = (opMode == "inband-samepci");
inbandDifferentPCI = (opMode == "inband-differentpci");
if inbandSamePCI
    enb.CellRefP = enb.NBRefP;     % Number of cell RS antenna ports (Fixed to 1 in this example)
    enb.NCellID = enb.NNCellID;
elseif inbandDifferentPCI
    enb.CellRefP = enb.NBRefP;     % Number of cell RS antenna ports (Fixed to 1 in this example)
    enb.NCellID = 1;
end
if ((npdschDataTypeLower == "bccnnotsib1nb") || (npdschDataType == "notbcch")) && ...
        (inbandSamePCI || inbandDifferentPCI)
    enb.ControlRegionSize = 3;     % The allowed values are 0...13
end

% Apply default window size according to TS 36.104 Table E.5.1-1a
if(~isfield(enb,"Windowing"))
    enb.Windowing = 6;
end

% Store enb structure with a name used for OFDM modulation and
% demodulation. The NB-IoT downlink waveform is a 1/2 subcarrier shift
% waveform. The lteSCFDMAModulate and lteSCFDMADemodulate functions use the
% NBULSubcarrierSpacing field to modulate and demodulate the NB-IoT
% downlink waveform, respectively.
enbOFDM = enb;
enbOFDM.NBULSubcarrierSpacing = "15kHz";

% Get the waveform information and set up the NTN channel
waveformInfo = lteSCFDMAInfo(enbOFDM);
ntnChannel = setupNTNChannel(channel,waveformInfo.SamplingRate);

% Compute the noise amplitude per receive antenna
kBoltz = physconst('Boltzmann');
NF = 10^(rxNoiseFigure/10);
T0 = 290;                                              % Noise temperature at the input (K)
Teq = rxAntennaTemperature + T0*(NF-1);                % K
N0_ampl = sqrt(kBoltz*waveformInfo.SamplingRate*Teq/2.0);

% Compute path loss based on the elevation angle and satellite altitude
re = physconst("earthradius");
c = physconst("lightspeed");
h = channel.SatelliteAltitude;
elevAngle = channel.ElevationAngle;
d = -re*sind(elevAngle) + sqrt((re*sind(elevAngle)).^2 + h*h + 2*re*h);
lambda = c/channel.CarrierFrequency;
pathLoss = fspl(d,lambda)*double(channel.IncludeFreeSpacePathLoss); % in dB

% Initialize throughput result
numTxPow = numel(txPower);
throughputPercent = zeros(numTxPow,numRep);
```

```
% Absolute subframe number at the starting point of the simulation
NSubframe = enb.NFrame*10+enb.NSubframe;

% Loop over repetitions
repVal = zeros(numRep,1);
for repIdx = 1:numRep
    % Add these fields to the npdsch structure
    npdsch.NSF = npdschInfo(repIdx).NSF;
    npdsch.NRep = npdschInfo(repIdx).NRep;
    npdsch.NPDSCHDataType = npdschDataType;
    repVal(repIdx) = npdsch.NRep;

    % Get the bit capacity and transport block length
    [~,info] = lteNPDSCHIndices(enb,npdsch);
    rmoutlen = info.G;                 % Bit length after rate matching (codeword length)
    trblklen = npdschInfo(repIdx).TBS; % Transport block size

    % The temporary variables 'enb_init', 'enbOFDM_init', and
    % 'channel_init' create the temporary variables 'enb', 'enbOFDM', and
    % 'ntnChannel' within the SNR loop to create independent simulation
    % loops for the 'parfor' loop
    enb_init = enb;
    enbOFDM_init = enbOFDM;
    channel_init = ntnChannel;

    for txPowIdx = 1:numTxPow
    % parfor txPowIdx = 1:numTxPow
    % To enable the use of parallel computing for increased the speed,
    % comment out the 'for' statement and uncomment the 'parfor' statement.
    % This functionality requires the Parallel Computing Toolbox. If you do
    % not have Parallel Computing Toolbox, 'parfor' defaults to the normal
    % 'for' statement.

        % Reset the random number generator so that each transmit power
        % point experiences the same noise realization
        rng(0,"threefry");

        enb = enb_init;                          % Initialize eNodeB configuration
        enbOFDM = enbOFDM_init;                  % Initialize eNodeB configuration related to OFDM
        ntnChannel = channel_init;               % Initialize fading channel configuration
        txcw = [];                               % Initialize the transmitted codeword
        numBlkErrors = 0;                        % Number of transport blocks with errors
        estate = [];                             % Initialize NPDSCH encoder state
        dstate = [];                             % Initialize NPDSCH decoder state
        lastOffset = 0;                          % Initialize overall frame timing offset
        offset = 0;                              % Initialize frame timing offset
        subframeGrid = lteNBResourceGrid(enb); % Initialize the subframe grid
        foffsetRS = 0;                           % Initialize frequency offset using reference sig

        N0 = N0_ampl;
        pl_dB = pathLoss;
        subframeIdx = NSubframe;
        numRxTrBlks = 0;
        reset(ntnChannel.BaseChannel);
        reset(ntnChannel.ChannelFilter);
        while (numRxTrBlks < numTrBlks)

            % Set current subframe and frame numbers
```

```
enb.NSubframe = mod(subframeIdx,10);
enb.NFrame = floor((subframeIdx)/10);

% Generate the NPSS symbols and indices
npssSymbols = lteNPSS(enb);
npssIndices = lteNPSSIndices(enb);
% Map the symbols to the subframe grid
subframeGrid(npssIndices) = npssSymbols;

% Generate the NSSS symbols and indices
nsssSymbols = lteNSSS(enb);
nsssIndices = lteNSSSIndices(enb);
% Map the symbols to the subframe grid
subframeGrid(nsssIndices) = nsssSymbols;

% Establish if either NPSS or NSSS is transmitted, and if so,
% do not transmit NPDSCH in this subframe
isDataSubframe = isempty(npssSymbols) && isempty(nsssSymbols);

% Create a new transport block, and encode it when the
% transmitted codeword is empty. The receiver sets the codeword
% to empty to signal that all subframes in a bundle have been
% received (it is also empty before the first transmission)
if isempty(txcw)
    txTrBlk = randi([0 1],trblklen,1);
    txcw = lteNDLSCH(rmoutlen,txTrBlk);
end

if (isDataSubframe)
    % Generate NPDSCH symbols and indices for a subframe
    [txNpdschSymbols,estate] = lteNPDSCH(enb,npdsch,txcw,estate);
    npdschIndices = lteNPDSCHIndices(enb,npdsch);
    % Map the symbols to the subframe grid
    subframeGrid(npdschIndices) = txNpdschSymbols;
    % Generate the NRS symbols and indices
    nrsSymbols = lteNRS(enb);
    nrsIndices = lteNRSIndices(enb);
    % Map the symbols to the subframe grid
    subframeGrid(nrsIndices) = nrsSymbols;
end

% Perform OFDM modulation to generate the time domain waveform.
% Use NB-IoT SC-FDMA to get the 1/2 subcarrier shift on the
% OFDM modulation.
txWaveform = lteSCFDMAModulate(enbOFDM,subframeGrid);

% Scale the waveform power based on the input transmit power
wavePower = 10*log10(sum(var(txWaveform)));
desiredPower = (txPower(txPowIdx)-30)-wavePower;      % In dB
txWaveform0 = db2mag(desiredPower)*txWaveform;

% Apply power amplifier nonlinearities
txWaveform1 = paMemorylessNonlinearity(paModelImpl,txWaveform0,enablePA);

% Pad waveform with 25 samples. This covers the range of
% delays expected from channel modeling (a combination of
% implementation delay and channel delay spread)
txWaveform1 = [txWaveform1; zeros(25,enb.NBRefP)]; %#ok<AGROW>
```

```
% Apply Doppler pre-compensation
txWaveform2 = compensateDopplerShift(enbOFDM,txWaveform1, ...
    ntnChannel.SatelliteDopplerShift,txDopplerCompensator);

% Pass data through channel model
rxWaveform = generateNTNChannel(ntnChannel,txWaveform2);

% Apply path loss to the signal
rxWaveform = rxWaveform*db2mag(-pl_dB);

% Add thermal noise to the received time-domain waveform. Multiply
% the noise variance with 2 as wgn function performs the scaling
% within.
noise = wgn(size(rxWaveform,1),size(rxWaveform,2),2*(N0^2),1,"linear","complex");
rxWaveform = rxWaveform + noise;

% Perform receiver Doppler compensation using reference signals
if (enb.NSubframe == 5)
    % Use NPSS signal for estimating Doppler
    refInd = npssIndices;
    refSym = npssSymbols;
    foffsetRS = estimateDopplerShiftUsingRS(enbOFDM,rxWaveform,refInd, ...
        refSym,rxDopplerCompensator);
end
rxWaveform1 = compensateDopplerShift(enbOFDM,rxWaveform,foffsetRS, ...
    rxDopplerCompensator);

% In this example, the subframe offset calculation relies
% on NPSS present in subframe 5, so we need to pad the
% subframes before it so that the frame offset returned by
% lteNBDLFrameOffset is the offset for subframe 5
sfTsamples = waveformInfo.SamplingRate*1e-3;
if (enb.NSubframe==5)
    padding = zeros([sfTsamples*5,size(rxWaveform1,2)]);
    offset = lteNBDLFrameOffset(enb,[padding; rxWaveform1]);
    if (offset > 25) || (offset < 0)
        offset = lastOffset;
    end
    lastOffset = offset;
end

% Synchronize the received waveform
rxWaveform1 = rxWaveform1(1+offset:end,:);

% Perform OFDM demodulation on the received data to recreate
% the resource grid. Use NB-IoT SC-FDMA to get the 1/2
% subcarrier shift on the OFDM demodulation.
rxSubframe = lteSCFDMADemodulate(enbOFDM,rxWaveform1,0.55);

% Channel estimation
[estChannelGrid,noiseEst] = lteDLChannelEstimate( ...
    enb,cec,rxSubframe);

% Data decoding
if (isDataSubframe)
    % Get NPDSCH indices
    npdschIndices = lteNPDSCHIndices(enb,npdsch);
```

```
                % Get PDSCH resource elements from the received subframe.
                % Scale the received subframe by the PDSCH power factor
                % Rho. The PDSCH is scaled by this amount, while the
                % reference symbols used for channel estimation (used in
                % the PDSCH decoding stage) are not.
                [rxNpdschSymbols,npdschHest] = lteExtractResources(npdschIndices, ...
                    rxSubframe,estChannelGrid);

                % Decode NPDSCH
                [rxcw,dstate,symbols] = lteNPDSCHDecode( ...
                                      enb,npdsch,rxNpdschSymbols,npdschHest,noiseEst,dstate);

                % Decode the transport block when all the subframes in a bundle
                % have been received
                if dstate.EndOfTx
                   [trblkout,blkerr] = lteNDLSCHDecode(trblklen,rxcw);
                   numBlkErrors = numBlkErrors + blkerr;
                   numRxTrBlks = numRxTrBlks + 1;
                   % Re-initialize to enable the transmission of a new transport block
                   txcw = [];
                end
            end

            subframeIdx = subframeIdx + 1;

        end

        % Calculate the throughput percentage
        throughputPercent(txPowIdx,repIdx) = 100*(1-(numBlkErrors/numTrBlks));
        fprintf("Throughput(%%) for %d transport block(s) at transmit power %d dBm with %d repeti
            numTrBlks,txPower(txPowIdx),npdsch.NRep,throughputPercent(txPowIdx,repIdx))

    end

end
```

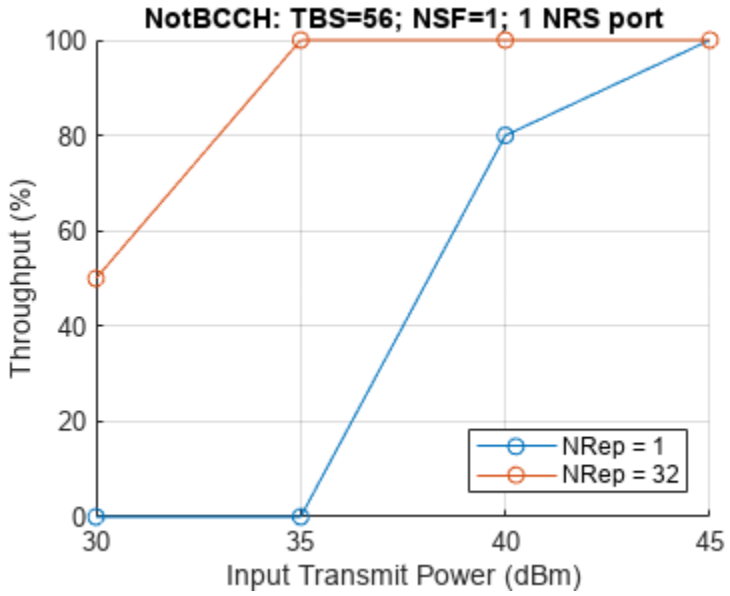
```
Throughput(%) for 10 transport block(s) at transmit power 30 dBm with 1 repetition(s): 0.0000
Throughput(%) for 10 transport block(s) at transmit power 35 dBm with 1 repetition(s): 0.0000
Throughput(%) for 10 transport block(s) at transmit power 40 dBm with 1 repetition(s): 80.0000
Throughput(%) for 10 transport block(s) at transmit power 45 dBm with 1 repetition(s): 100.0000
Throughput(%) for 10 transport block(s) at transmit power 30 dBm with 32 repetition(s): 50.0000
Throughput(%) for 10 transport block(s) at transmit power 35 dBm with 32 repetition(s): 100.0000
Throughput(%) for 10 transport block(s) at transmit power 40 dBm with 32 repetition(s): 100.0000
Throughput(%) for 10 transport block(s) at transmit power 45 dBm with 32 repetition(s): 100.0000
```

Results

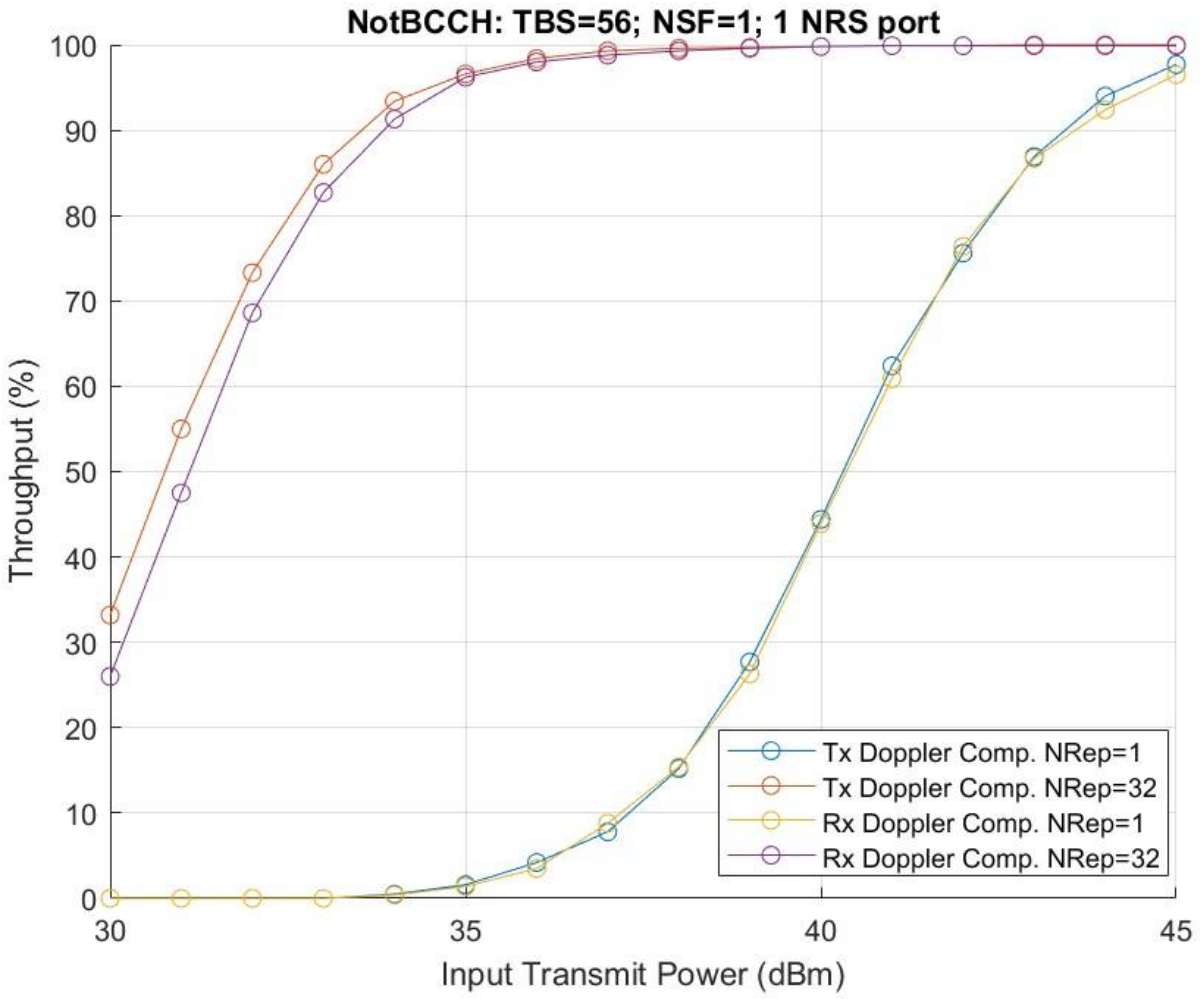
Display the measured throughput, which is the percentage of the maximum possible throughput of the link given the available resources for data transmission.

```
% Set figure title
if strcmpi(npdschDataType,"SIB1NB")
    npdsch.NSF = 8;
end
figure; grid on; hold on;
legendstr = repmat("",numRep,1);
for repIdx = 1:numRep
```

```
    plot(txPower,throughputPercent(:,repIdx),"-o")
    legendstr(repIdx) = "NRep = " + repVal(repIdx);
end
hold off;xlabel('Input Transmit Power (dBm)'); ylabel('Throughput (%)');
title(npdsch.NPDSCHDataType + ": TBS=" + trblklen + ...
    "; NSF=" + npdsch.NSF + "; " + enb_init.NBRefP + " NRS port")
legend(legendstr,Location="southeast")
```



This example figure shows the throughput results obtained by simulating 1000 transport blocks (`numTrBlks = 1000`, `txPower = 30:45`) for repetition indices 0 and 5. The simulation setup includes the default higher layer, eNodeB, and NPDSCH configuration with an ETSI Rician channel. To obtain the lines corresponding to the `Tx Doppler Comp.`, you can set `txDopplerCompensator` to `true` and `rxDopplerCompensator` to `false`. For the lines corresponding to the `Rx Doppler Comp.`, you can set `txDopplerCompensator` to `false` and `rxDopplerCompensator` to `true`.



Further Exploration

This example shows the throughput simulation for NB-IoT NPDSCH in an NTN channel. In addition to the default behavior, you can run the example for these cases.

- Analyze the throughput at each transmit power value and repetition index for a different satellite orbit by varying the satellite altitude and satellite speed.
- Observe the link performance without any Doppler compensation techniques by setting the `txDopplerCompensator` and `rxDopplerCompensator` variables to `false`.
- Observe the link performance with only Doppler compensation at only the receiver by setting `txDopplerCompensator` to `false` and `rxDopplerCompensator` to `true`.
- Check the throughput performance for different repetitions by changing the `iReps` value.
- Compare the throughput performance in an NTN and terrestrial network by using the `lteFadingChannel` channel as shown in “NB-IoT NPDSCH Block Error Rate Simulation” on page 2-304 example.

Appendix

This example uses this helper function.

- hNPDSCHInfo — NB-IoT NPDSCH information

References

1. ETSI TS 101 376-5-5 V1.3.1 (2005-02). GEO-Mobile Radio Interface Specifications (Release 1); Part 5: Radio interface physical layer specifications; Sub-part 5: Radio Transmission and Reception; GMR-1 05.005.
2. ITU-R Recommendation P.681-11 (08/2019). “Propagation data required for the design systems in the land mobile-satellite service.” P Series; Radio-wave propagation.
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)*.
4. 3GPP TS 36.213, "Physical layer procedures", *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)*.
5. 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)*.
6. 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)*.
7. 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)*.
8. 3GPP TR 36.763, "Study on Narrow-Band Internet of Things (NB-IoT) / enhanced Machine Type Communication (eMTC) support for Non-Terrestrial Networks (NTN) (Release 17), *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
9. 3GPP TR 38.803, "Study on new radio access technology: Radio Frequency (RF) and co-existence aspects (Release 14)", *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
10. 3GPP TR 38.811, "Study on New Radio (NR) to support non-terrestrial networks (Release 15)", *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
11. O. Hammi, S. Carichner, B. Vassilakis, and F.M. Ghannouchi. “Power Amplifiers’ Model Assessment and Memory Effects Intensity Quantification Using Memoryless Post-Compensation Technique.” IEEE Transactions on Microwave Theory and Techniques 56, no. 12 (December 2008): 3170–79.

Local Functions

This example uses these local functions.

```
function chanOut = setupNTNChannel(channel,sampleRate)
% Setup NTN channel

    % Assign temporary variables for carrier frequency and maximum Doppler
    % shift due to mobile movement
    fc = double(channel.CarrierFrequency);
```

```
        c = physconst("LightSpeed");
        maxDoppler = (double(channel.MobileSpeed)*fc)/c;
        elevAngle = double(channel.ElevationAngle);
        h = double(channel.SatelliteAltitude);
        v = double(channel.SatelliteSpeed);
        % Calculate the Doppler shift due to satellite movement
        maxDopplerSat = satcom.internal.dopplerShift(fc,v,elevAngle,h);
        % Check the maximum Doppler shift and sample rate
        if ((maxDoppler+maxDopplerSat) >= (sampleRate/10))
            error("satcom:setupNTNChannel:MaxDoppler", ...
                "The maximum Doppler shift (%d Hz) due to mobile and satellite " + ...
                "movement, must be less than %d Hz which is one-tenth of SampleRate.", ...
                (maxDoppler + maxDopplerSat),sampleRate/10)
        end

        chanOut = struct;
        chanTypeLower = lower(channel.NTNChannelType);
        if chanTypeLower == "etsi rician"
            channelName = "ETSI Rician";
            baseChannel = etsiRicianChannel;
            baseChannel.SampleRate = sampleRate;
            baseChannel.KFactor = channel.KFactor;
            baseChannel.MaximumDopplerShift = maxDoppler;
        elseif chanTypeLower == "itu-r p.681"
            channelName = "ITU-R P.681";
            baseChannel = p681LMSChannel;
            baseChannel.SampleRate = sampleRate;
            baseChannel.Environment = channel.Environment;
            baseChannel.CarrierFrequency = channel.CarrierFrequency;
            baseChannel.MobileSpeed = channel.MobileSpeed;
            baseChannel.ElevationAngle = channel.ElevationAngle;
            baseChannel.AzimuthOrientation = channel.AzimuthOrientation;
            baseChannel.FadingTechnique = "Sum of sinusoids";
        end
        baseChannel.RandomStream = "mt19937ar with seed";
        baseChannel.Seed = channel.Seed;

        % Set the channel filter
        chanFilt = comm.ChannelFilter( ...
                    SampleRate=sampleRate,PathDelays=0, ...
                    NormalizeChannelOutputs=false);

        % Set the output structure
        chanOut.ChannelName = channelName;
        chanOut.CarrierFrequency = fc;
        chanOut.SatelliteSpeed = v;
        chanOut.SatelliteAltitude = h;
        chanOut.ElevationAngle = elevAngle;
        chanOut.BaseChannel = baseChannel;
        chanOut.SatelliteDopplerShift = maxDopplerSat;
        chanOut.ChannelFilter = chanFilt;

    end

    function [out,sampleTimes] = generateNTNChannel(channel,in)
    % Generate NTN channel

        % Get the channel information before channel processing
```

```
        prevInfo = info(channel.BaseChannel);
        numSamplesStart = prevInfo.NumSamplesProcessed;

        % Get the path gains of base channel
        [~,pathGainsBase] = channel.BaseChannel(in);

        % Get the channel information after channel processing
        postInfo = info(channel.BaseChannel);
        numSamplesEnd = postInfo.NumSamplesProcessed;

        % Get the channel sample times
        sampleTimes = (numSamplesStart:(numSamplesEnd-1)).'/channel.BaseChannel.SampleRate;

        % Apply satellite Doppler shift to the base channel path gains
        pathGains = pathGainsBase.*exp(1i*2*pi*channel.SatelliteDopplerShift*sampleTimes);

        % Perform channel filtering
        out = channel.ChannelFilter(in,pathGains);

    end

    function out = compensateDopplerShift(enb,inWave,foffset,flag)
    % Perform Doppler shift correction

        if flag
            % Correct frequency offset
            out = lteFrequencyCorrect(enb,inWave,foffset);
        else
            out = inWave;
        end

    end

    function out = estimateDopplerShiftUsingRS(enb,rxWave,refInd, ...
        refSym,flag)
    % Estimate the Doppler shift using NPSS

        if flag
            % Set the Windowing field to 0, as this information is not known at
            % the receiver
            enb.Windowing = 0;
            ofdmInfo = lteSCFDMAInfo(enb);
            K = 12;                             % Number of subcarriers
            L = 14;                             % Number of OFDM symbols in slot

            % Initialize temporary variables
            rxWave1 = [rxWave; zeros((mod(size(rxWave,1),2)),1)]; % Append zero, if required
            rxLen = size(rxWave1,1);

            % Generate reference waveform
            refGrid = complex(zeros([K L]));
            refGrid(refInd) = refSym;
            refWave = lteSCFDMAModulate(enb,refGrid);
            refWave = [refWave; zeros((rxLen-size(refWave,1)),1)];

            % Compute the correlation of received waveform with reference
            % waveform
            x_wave = rxWave1.*conj(refWave);
```

```
        % Compute FFT of the resultant waveform
        x_fft = fftshift(fft(x_wave));

        % FFT bin values
        fftBinValues = (-rxLen/2:(rxLen/2-1))*(ofdmInfo.SamplingRate/rxLen);

        % Use the FFT bin index corresponding to the maximum FFT value.
        % The FFT bin value corresponding to this bin index is the integer
        % frequency offset.
        [~,binIndex] = max(x_fft);
        out = fftBinValues(binIndex);
    else
        out = 0;
    end

end

function varargout = paMemorylessNonlinearity(paModel,varargin)
% Apply power amplifier nonlinearity (TR 38.803)
% out = paMemorylessNonlinearity(paModel,in,enable) returns the
% impaired output.
% paMemorylessNonlinearity(paModel) returns the plot with the gain and
% phase characteristics of the power amplifier

    if nargin == 1
        in_NoScale = randn(1e6,1)+1j*randn(1e6,1);
        scaleFactor = 1/sqrt(2);
        enable = 1;
    else
        in_NoScale = varargin{1};
        scaleFactor = 1;
        enable = varargin{2};
    end

    if enable
        in = scaleFactor*in_NoScale;
        if isa(paModel,"comm.MemorylessNonlinearity")
            % paModel is a comm.MemorylessNonlinearity System object
            out = paModel(in);
            paModelName = "";
        else
            absIn = abs(in);
            paModelName = paModel;
            switch lower(paModel)
                case "2.1ghz gaas"
                    % 2.1GHz GaAs
                    out = (-0.618347-0.785905i) * in + (2.0831-1.69506i) * in .* absIn.^(2) + ..
                        (-14.7229+16.8335i) * in .* absIn.^(2*2) + (61.6423-76.9171i) * in .* abs
                        (-145.139+184.765i) * in .* absIn.^(2*4) + (190.61-239.371i)* in .* absIn
                        (-130.184+158.957i) * in .* absIn.^(2*6) + (36.0047-42.5192i) * in .* abs
                otherwise
                    % 2.1GHz GaN
                    out = (0.999952-0.00981788i) * in + (-0.0618171+0.118845i) * in .* absIn.^(2
                        (-1.69917-0.464933i) * in .* absIn.^(2*2) + (3.27962+0.829737i) * in .* a
                        (-1.80821-0.454331i) * in .* absIn.^(2*4);
            end
        end
```

```
    else
        out = in_NoScale;
    end

    if nargout > 0
        varargout{1} = out;
    end

    if nargin == 1 || (nargout == 0)
        % Gain Plot
        inpPower = 20*log10(absIn);
        gain = 20*log10(abs(out))-inpPower;
        figure
        subplot(211)
        plot(inpPower,gain,".")
        grid on
        ylim([-Inf 1])
        xlim([-30 0])
        xlabel("Normalized input power (dB)")
        ylabel("Gain (dB)")
        title("Gain Characteristics of PA Model " + paModelName)

        % Phase Plot
        phase = angle(out.*conj(in))*180/pi;
        subplot(212)
        plot(inpPower,phase,".")
        grid on
        xlim([-30 0])
        xlabel("Normalized input power (dB)")
        ylabel("Phase (deg)")
        title("Phase Characteristics of PA Model " + paModelName)
    end

end

function paChar = getDefaultCustomPA()
% The operating specifications for the LDMOS-based Doherty amplifier are:
% * A frequency of 2110 MHz
% * A peak power of 300 W
% * A small signal gain of 61 dB
% Each row in HAV08_Table specifies Pin (dBm), gain (dB), and phase shift
% (degrees) as derived from figure 4 of Hammi, Oualid, et al. "Power
% amplifiers' model assessment and memory effects intensity quantification
% using memoryless post-compensation technique." IEEE Transactions on
% Microwave Theory and Techniques 56.12 (2008): 3170-3179.

    HAV08_Table = ...
        [-35,60.53,0.01;
        -34,60.53,0.01;
        -33,60.53,0.08;
        -32,60.54,0.08;
        -31,60.55,0.1;
        -30,60.56,0.08;
        -29,60.57,0.14;
        -28,60.59,0.19;
        -27,60.6,0.23;
        -26,60.64,0.21;
        -25,60.69,0.28;
```

```
            -24,60.76,0.21;
            -23,60.85,0.12;
            -22,60.97,0.08;
            -21,61.12,-0.13;
            -20,61.31,-0.44;
            -19,61.52,-0.94;
            -18,61.76,-1.59;
            -17,62.01,-2.73;
            -16,62.25,-4.31;
            -15,62.47,-6.85;
            -14,62.56,-9.82;
            -13,62.47,-12.29;
            -12,62.31,-13.82;
            -11,62.2,-15.03;
            -10,62.15,-16.27;
            -9,62,-18.05;
            -8,61.53,-20.21;
            -7,60.93,-23.38;
            -6,60.2,-26.64;
            -5,59.38,-28.75];
        % Convert the second column of the HAV08_Table from gain to Pout for
        % use by the memoryless nonlinearity System object.
        paChar = HAV08_Table;
        paChar(:,2) = paChar(:,1) + paChar(:,2);

    end
```

Time Difference of Arrival Positioning Using PRS

This example shows how to use the Time Difference Of Arrival (TDOA) positioning approach in conjunction with the Release 9 Positioning Reference Signal (PRS) to calculate the position of a User Equipment (UE) within a network of eNodeBs using the LTE Toolbox™.

Introduction

In this example, a number of eNodeB transmissions are created and combined with different delays and received powers to model the reception of all the eNodeB waveforms by one UE. The UE performs correlation with the Positioning Reference Signal (PRS) to establish the delay from each eNodeB and subsequently the delay difference between all pairs of eNodeBs. These delay differences are used to compute hyperbolas of constant delay difference, which are plotted relative to known eNodeB positions and intersect at the position of the UE.

Transmitter Configuration

A set (cell array) of eNodeB configurations `enb` is created, with the number of eNodeBs specified by `NeNodeB`. The configurations are derived from Reference Measurement Channel (RMC) R.5 using `lteRMCDL`. R.5 describes a 3 MHz bandwidth Downlink Shared Channel (PDSCH) transmission using 64-QAM modulation. For each eNodeB the configuration is updated to make the cell identity `NCellID` unique and the PRS parameters `NPRSRB`, `IPRS` and `PRSPeriod` are set.

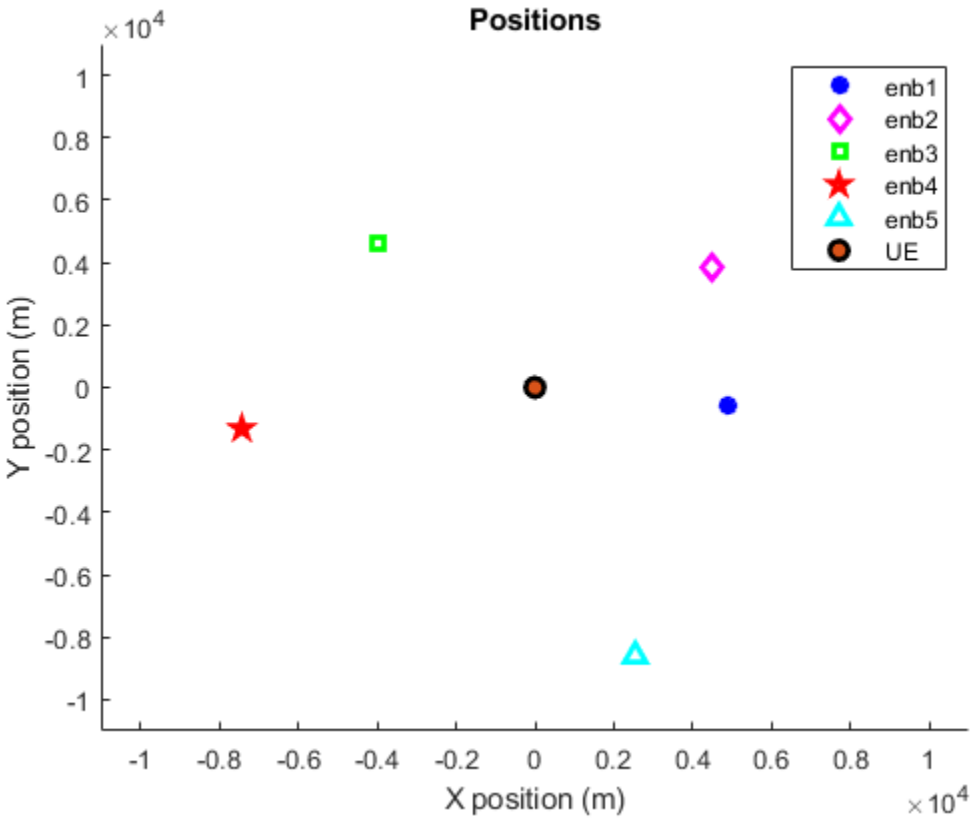
A random position given by an X and Y coordinate for each eNodeB is generated by the `hPositioningPosition` function.

```
rng('default'); % Initialize the random number generator stream
NeNodeB = 5;    % Number of eNodeB

% Create eNodeB configurations
enb = cell(1,NeNodeB);
for i=1:NeNodeB
   enb{i}=lteRMCDL('R.5');            % Get configuration based on RMC
   enb{i}.NCellID = mod((i-1)*2,504); % Set arbitrary cell identity
   enb{i}.TotSubframes = 1;           % Number of subframes to generate
   enb{i}.NPRSRB = 2;                 % Bandwidth of PRS in resource blocks
   enb{i}.IPRS = 0;                   % PRS configuration index
   enb{i}.PRSPeriod = 'On';           % PRS present in all subframes
   enb{i}.Position = hPositioningPosition(i-1, NeNodeB); % eNodeB position
end

% Use the first eNodeB configuration for general settings
info = lteOFDMInfo(enb{1});
```

Plot Location of eNodeBs and UE

The positions of the eNodeBs and the UE are plotted for reference. The UE lies at (0,0) and the eNodeBs are distributed around the UE.

```
hPositioningPlotPositions(enb);
```



Generate Transmissions

For each eNodeB, a transmission is made consisting of the PRS, Primary Synchronization Signal (PSS), Secondary Synchronization Signal (SSS) and Cell-specific Reference Signal (Cell RS). An empty resource grid is created and a PRS is generated and mapped onto the grid using `ltePRS` and `ltePRSIndices`. The PSS, SSS and Cell RS are added in a similar fashion. The resultant grid is OFDM modulated to produce a transmit waveform.

```
tx = cell(1,NeNodeB);
for i = 1:NeNodeB
    grid = [];
    for nsf = 0:19
        enb{i}.NSubframe = mod(nsf,10);
        sfgrid = lteDLResourceGrid(enb{i});       % Empty subframe
        sfgrid(ltePRSIndices(enb{i})) = ltePRS(enb{i});       % PRS REs
        sfgrid(ltePSSIndices(enb{i})) = ltePSS(enb{i});       % PSS REs
        sfgrid(lteSSSIndices(enb{i})) = lteSSS(enb{i});       % SSS REs
        sfgrid(lteCellRSIndices(enb{i})) = lteCellRS(enb{i}); % Cell RS REs
        grid = [grid sfgrid]; %#ok<AGROW>
    end
    enb{i}.NSubframe = 0;
    tx{i} = lteOFDMModulate(enb{i}, grid);        % OFDM modulate
end
```

Compute Delays from eNodeBs to UEs

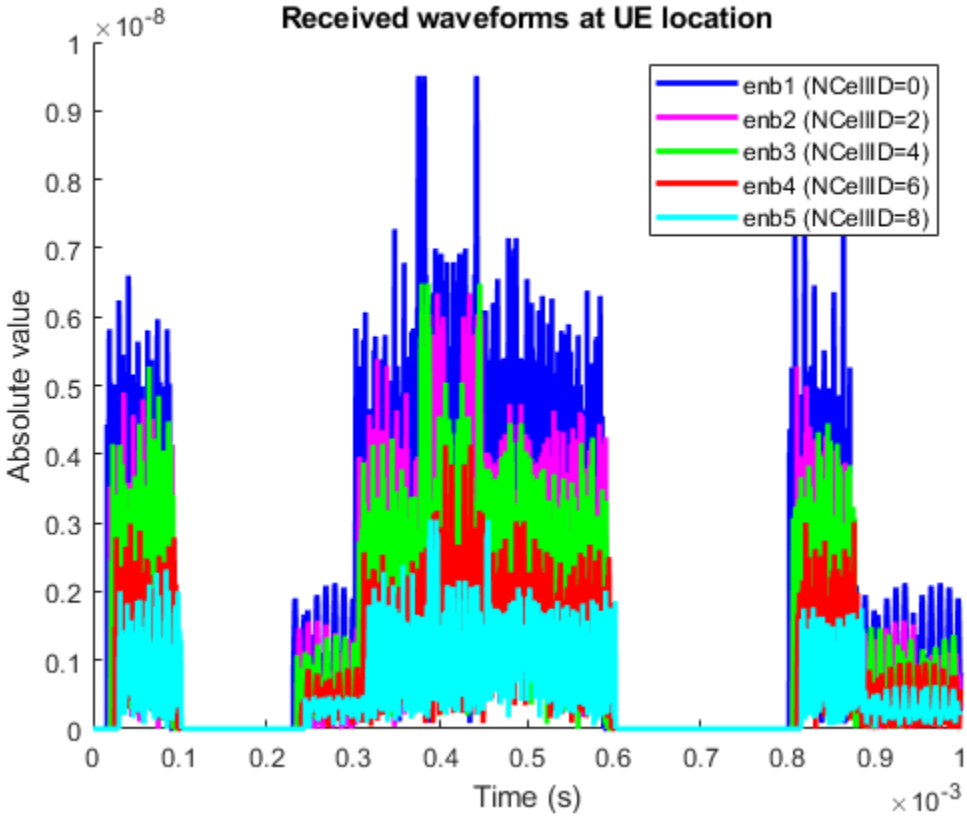
Using the known eNodeB positions, the time delay from each eNodeB to the UE is calculated using the distance between the UE and eNodeB, `radius`, and the speed of propagation (speed of light). Using knowledge of the sampling rate, `info.SamplingRate`, the sample delay is calculated and stored in `sampleDelay`. These variables will be used to model the environment between the eNodeBs and the UE but the information will NOT be provided to the UE.

```
speedOfLight = 299792458.0; % Speed of light in m/s

sampleDelay = zeros(1, NeNodeB);
radius = cell(1, NeNodeB);
for i = 1:NeNodeB
   [~, radius{i}] = cart2pol(enb{i}.Position(1), enb{i}.Position(2));
   delay = radius{i}/speedOfLight;                  % Delay in seconds
   sampleDelay(i) = round(delay*info.SamplingRate); % Delay in samples
end
```

Create Sum of Received Waveforms and Plot Received Waveforms

The received signal at the UE is modeled by delaying each eNodeB transmission according to the values in `sampleDelay`, and attenuating the received signal from each eNodeB using the values in `radius` in conjunction with an implementation of the TR 36.814 [1] Urban Macro Line Of Sight (LOS) path loss model. The received waveform from each eNodeB is padded with zeros to ensure all waveforms are the same length.

```
sumrx = zeros(length(tx{1})+max(sampleDelay), 1);
rx = cell(1, NeNodeB);
for i = 1:NeNodeB
    % Urban Macro LOS path loss per TR36.814
    PLdB = hPositioningPathLoss(radius{i}, 2.1e9);
    PL = 10^(PLdB/10);

    % Add delay, pad and attenuate
    rx{i} = [zeros(sampleDelay(i), 1); tx{i}; ...
                zeros(max(sampleDelay)-sampleDelay(i), 1)]/ sqrt(PL);

    % Sum waveforms from all eNodeBs
    sumrx = sumrx + rx{i};
end

% Plot received waveforms
hPositioningPlotRx(enb, rx, info.SamplingRate);
```



Perform Cell Search to Establish Cell Identities

Multi-cell search is performed in order to identify the cell identity of each eNodeB. An array of configurations `rxcfg` for the eNodeBs is then created based on the detected cell identities and assuming that the PRS configuration has been given by higher-layer signaling and is therefore known to the UE. A number of other physical layer parameters such as the cyclic prefix length and duplex mode are assumed to be known and are assumed to be equal for each eNodeB. See the “Cell Search, MIB and SIB1 Recovery” on page 2-351 example for more information on detecting these parameters.

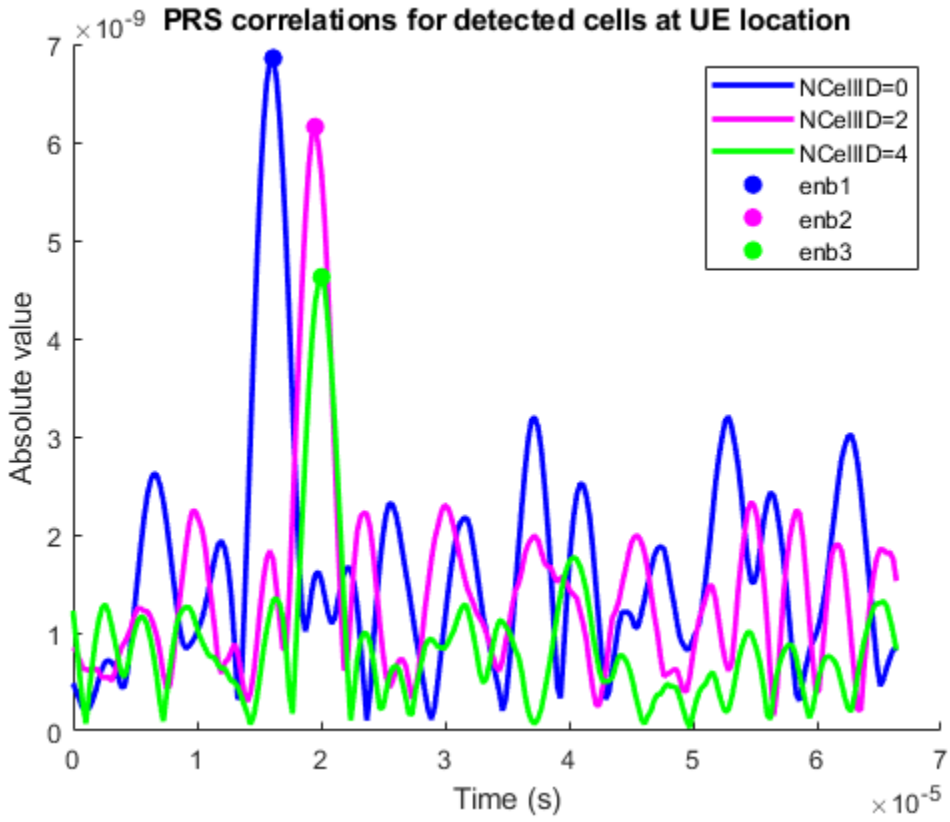
```
% Assumed parameters for cell search
searchcfg.CyclicPrefix = enb{1}.CyclicPrefix;
searchcfg.DuplexMode = enb{1}.DuplexMode;
searchcfg.NDLRB = enb{1}.NDLRB;

% Perform multi-cell search
searchalg.MaxCellCount = NeNodeB;
searchalg.SSSDetection = 'PostFFT';
[cellIDs,offsets] = lteCellSearch(searchcfg,sumrx,searchalg);

% Set up configurations for each detected cell; cells are considered as
% detected here if they meet a minimum RSRQ threshold Qqualmin
Qqualmin = -20;
RSRQdB = zeros(1,searchalg.MaxCellCount);
rxcfg = cell(1,searchalg.MaxCellCount);
for i = 1:searchalg.MaxCellCount
    % Assumed parameters
    rxcfg{i} = enb{1};
```

```
    % Use cell identity that was detected
    rxcfg{i}.NCellID = cellIDs(i);
    % Measure RSRQ
    rxgrid = lteOFDMDemodulate(rxcfg{i},sumrx(1+offsets(i):end,:));
    meas = hRSMeasurements(rxcfg{i},rxgrid);
    RSRQdB(i) = meas.RSRQdB;
end
rxcfg(RSRQdB<Qqualmin) = [];
Ndetected = numel(rxcfg);
```

Estimate Arrival Times

The arrival times of the signals from each eNodeB are established at the UE by correlating the incoming signal with a local PRS generated with the cell identity of each eNodeB. Note that the absolute arrival times cannot be used at the UE to calculate its position as it has no knowledge of how far away the eNodeBs are, only the difference in distances given by the difference in arrival times. Therefore, the peak correlation for each eNodeB is used as a delay estimate to allow comparison.

```
ref = cell(1, Ndetected);
corr = cell(1, Ndetected);
delayEst = zeros(1, Ndetected);
for i = 1:Ndetected
    % Generate reference PRS
    sfgrid = lteDLResourceGrid(rxcfg{i});
    sfgrid(ltePRSIndices(rxcfg{i})) = ltePRS(rxcfg{i});
    ref{i} = lteOFDMModulate(rxcfg{i},sfgrid);

    % Correlate received signal with each reference PRS
    c = abs(xcorr(sumrx,ref{i}));

    % Reduced length of correlation vector for positioning and plotting
    c(1:length(sumrx)) = [];    % Remove meaningless result at beginning
    corr{i} = c(1:info.Nfft);   % Extract an OFDM symbol's worth of data

    % Delay estimate is at point of maximum correlation
    delayEst(i) = find(corr{i}==max(corr{i}));
end

% Plot correlation
if (Ndetected>0)
    hPositioningPlotCorr(rxcfg, corr, info.SamplingRate);
end
```

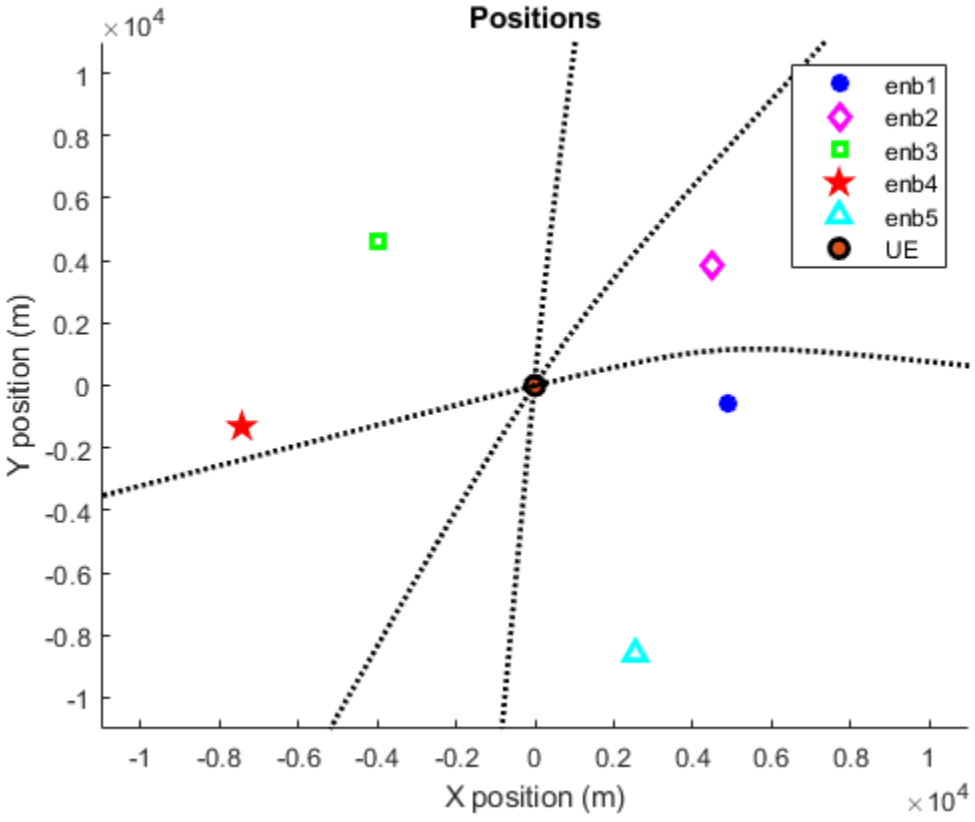


Compute TDOA and Plot Constant TDOA Hyperbolas

Using the arrival times, the time differences of arrival between each pair of eNodeBs is calculated using the `hPositioningTDOA` function. The particular time difference of arrival between a pair of eNodeBs can result from the UE being located at any position where two circles, each centered on an eNodeB, intersect. The two circles have radii which differ by the distance covered at the speed of light in the given time difference. The complete set of possible UE positions across all possible radii for one circle (with the other circle maintaining a radius appropriate to the time difference as already described) forms a hyperbola. The "hyperbolas of constant delay difference" for all the different pairs of eNodeBs are plotted relative to the known eNodeB positions and intersect at the position of the UE.

```
% Estimate time difference of arrival from each eNodeB
tdoa = hPositioningTDOA(delayEst,info.SamplingRate);

% Plot hyperbolas
figure(1);
legendstr = feval(@(x)x.String,legend);
enbs = [enb{:}];
txCellIDs = [enbs.NCellID];
for j = 1:Ndetected
    for i = (j+1):Ndetected
        dd = tdoa(i,j)*speedOfLight; % Delay distance
        % establish the eNodeBs for which the delay distance
        % is applicable by examining the detected cell identities
        txi = find(txCellIDs==rxcfg{i}.NCellID);
        txj = find(txCellIDs==rxcfg{j}.NCellID);
```

```
        if (~isempty(txi) && ~isempty(txj))
            % plot TDOA curve
            [x, y] = hPositioningTDOACurve(enb{txi}.Position, ...
                enb{txj}.Position, dd);
            plot(x, y, 'k:', 'LineWidth', 2);
        end
    end
end
legend(legendstr);
```



Appendix

This example uses these helper functions.

- hPositioningPosition.m
- hPositioningPlotPositions.m
- hPositioningPathLoss.m
- hPositioningPlotRx.m
- hPositioningPlotCorr.m
- hPositioningTDOA.m
- hPositioningTDOACurve.m

References

1 3GPP TR 36.814 "Further advancements for E-UTRA physical layer aspects"

DL-SCH HARQ Modeling

This example demonstrates Hybrid Automatic Repeat reQuest (Hybrid-ARQ) Incremental Redundancy (IR) in the Downlink Shared Channel (DL-SCH) transmission using the LTE Toolbox™.

Introduction

The Downlink Shared Channel (DL-SCH) is described in TS36.212, Section 5.3.2 [2]. This example demonstrates how a transmitter retransmits a single codeword over a single layer, using a different Redundancy Version (RV) each time until the CRC of the received codeword indicates a successful transmission. The DL-SCH transmission is 16QAM modulated with a 1/2 target coding rate transmitted over single antenna port. The setup used in this example is based on Fixed Reference Channel R.3 defined in Table A.3.3.1-2 of TS36.101 [1].

Hybrid Automatic Repeat reQuest (Hybrid-ARQ) is a combination of Forward Error Correction (FEC) and Automatic Repeat reQuest in an optimal manner. Hybrid-ARQ schemes are commonly used to facilitate reliable communication over noisy wireless channels. HARQ is able to compensate for link adaptation errors and provides a finer granularity of coding rate, resulting in better throughput performance than other FEC schemes.

There are three types of Hybrid ARQ: Type I, Type-II and Type-III.

Hybrid ARQ Type I

The simplest method: Hybrid ARQ Type I uses the CRC to detect whether or not an error in transmission has occurred. If a packet is found to be in error a retransmission request will be sent to the transmitter and the erroneous packet will be discarded. The transmitter will then retransmit the same packet until the packet is successfully decoded by the receiver or the maximum retransmission limit is reached.

Hybrid ARQ Type I can be extended to include packet combining, this is known as Hybrid ARQ Type I with Packet Combing or Chase Combining. After each failed retransmission the erroneous packets are stored in a buffer. The receiver then uses maximum ratio combining to combine each received channel bit with any previous transmissions of the same bit and the combined signal is fed to the decoder. Chase combining does not give any additional coding gain, it only increases the accumulated received signal to noise ratio $\mathrm{Eb/N0}$ for each retransmission.

Hybrid ARQ Type II

In Hybrid ARQ Type II, also known as full Incremental Redundancy (IR), each retransmission is not necessarily identical to the original transmission. Instead, multiple sets of coded bits are generated and whenever a retransmission is required the retransmitted data represents a different set of coded bits than the previous transmission. The receiver combines the retransmission with previous transmission attempts of the same packet. As the retransmission contains additional parity bits, not included in the previous transmission attempts, the resulting code rate is generally lowered by subsequent retransmissions. Each transmission contains a different set of parity bits resulting in a higher coding gain when compared to chase combining.

Hybrid ARQ Type III

The final method Hybrid ARQ Type III, also known as partial IR, decreases the coding rate by sending additional redundancy bits in each retransmission. It does however ensure that retransmissions are

able to self-decode. This means the retransmitted packet can be chase combined with previous packets to increase the diversity gain.

HARQ Process in LTE

LTE utilizes IR HARQ with a 1/3 turbo encoder used for FEC. The Transport Block (TB) CRC is used to detect errors. The receiver only receives different punctured versions of the same turbo-encoded data; each of these retransmissions are self decodable. Thus, it falls into category of a type III Hybrid ARQ.

In LTE retransmissions are sent with an initial coding rate of 1/2 or 3/4. The maximum number of simultaneous DL-HARQ processes (number of PDSCH transmissions catered for) is limited to 8 as specified in TS36.213, Section 7 [3].

In LTE, the N-channel stop-and-wait protocol is used as the Hybrid ARQ protocol as it offers low buffering requirements and low Acknowledgment (ACK)/Negative Acknowledgment (NACK) feedback overhead.

DL-HARQ Process in the LTE Toolbox

In this example a transport block is generated and undergoes DL-SCH coding to create a codeword. The codeword undergoes physical downlink shared channel coding to form complex modulated symbols.

Additive White Gaussian Noise is added to the symbols. The noisy symbols then undergo receiver processing to obtain the transmitted codeword. The codeword is then turbo rate recovered, code block desegmented and CRC block decoded to check if transmission was successful. If a CRC error is detected in the transport block then a retransmission is made using a different RV. This process continues until the transmission is successful or the retransmission limit is reached.

Cell-wide Settings

Cell wide settings are specified in a structure `enb`.

```
enb.NDLRB = 50;                % No of Downlink RBs in total BW
enb.CyclicPrefix = 'Normal';   % CP length
enb.PHICHDuration = 'Normal';  % PHICH duration
enb.NCellID = 10;              % Cell ID
enb.CellRefP = 1;              % Single antenna ports
enb.DuplexMode = 'FDD';        % FDD Duplex mode
enb.CFI = 2;                   % 2 PDCCH symbols
enb.Ng = 'sixth';              % HICH groups
enb.NSubframe = 0;             % Subframe number 0
```

PDSCH Transmission Mode Configuration

The Physical Downlink Shared Channel (PDSCH) is configured using a structure `pdsch` for a single antenna transmission scheme.

```
pdsch.NLayers = 1;                % No of layers to map the transport block
pdsch.TxScheme = 'Port0';         % Transmission scheme
pdsch.Modulation = {'16QAM'};     % Modulation
pdsch.RV = 0;                     % Initialize Redundancy Version
pdsch.RNTI = 500;                 % Radio Network Temporary Identifier
pdsch.NTurboDecIts = 5;           % Number of turbo decoder iterations
pdsch.PRBSet = (0:enb.NDLRB-1).'; % Define the PRBSet
pdsch.CSI = 'On';                 % CSI scaling of soft bits
```

Downlink Coding Configuration

Define the parameters required for DL-SCH encoding. The transport block size used here is as defined for R.3 RMC in Table A.3.3.1-2 of TS36.101 [1]. The DL-SCH coded block size can be calculated by the `ltePDSCHIndices` function using `enb` and `pdsch`. The `ltePDSCHIndices` function returns an information structure as its second output, containing the parameter `G` which specifies the number of coded and rate-matched DL-SCH data bits to satisfy the physical PDSCH capacity.

```
rvIndex = 0;                                    % Redundancy Version index
transportBlkSize = 12960;                       % Transport block size
[~,pdschIndicesInfo] = ltePDSCHIndices(enb,pdsch,pdsch.PRBSet);
codedTrBlkSize = pdschIndicesInfo.G;            % Available PDSCH bits
dlschTransportBlk = randi([0 1], transportBlkSize, 1); % DL-SCH data bits

% Possible redundancy versions (number of retransmissions)
redundancyVersions = 0:3;
```

Retransmission Loop

This example models a single HARQ process. After every transmission the value of `blkCRCerr` is used to check for a successful transmission of a transport block. If a CRC error is detected i.e. `blkCRCerr >=1` then a retransmission is performed using a different RV value.

The first transmission is done using a RV of 0, this indicates the initialization stage. If a CRC error is detected by the User Equipment (UE) it sends a NACK to the Base Station (BS) so that a retransmission is initiated using a different RV value. A value of 1 or greater returned for the block CRC error.

The eNodeB will keep transmitting the same transport block using different RV values until the UE receives an error free transport block or the total retransmission limit occurs. In LTE the total number of HARQ processes which can be initiated at any given time is 8.

To transmit and receive a transport block the following steps occur:

- *DL-SCH Channel Coding.* The DL-SCH bits are generated and undergo channel coding. Included in this process is transport block 24A-type CRC insertion, code block segmentation and code block CRC insertion, turbo coding, rate matching and code block concatenation. The number of code block segmentation and CRC insertions into each segment depends on the given transport block size. Each segmented block is separately turbo encoded and rate matched after a code block 24B-type CRC insertion. The concatenation process is applied on the rate matched turbo coded blocks to form a codeword. If the transmission results in error then the UE signals a NACK. The retransmission of an erroneous packet is done using different RVs. Each RV corresponds to a different set of parity bits from the same encoded block; the RV controls this variation. All these operations can be performed using the toolbox function `lteDLSCH`.
- *PDSCH Complex Symbols Generation.* Scrambling, modulation, layer mapping and precoding are applied to the coded transport block to generate the PDSCH complex symbols. This is achieved using `ltePDSCHPRBS`.
- *Noise Addition.* Generated noise is then added to PDSCH complex symbols. By varying the value of the variance `nVariance` the number of retransmissions will also vary, this is because the number of errors detected will fluctuate with the amount of noise present on the symbols.
- *PDSCH Receiver Processing.* In the PDSCH receiver deprecoding, layer demapping, soft demodulation and descrambling are applied to the noisy PDSCH complex symbols.

- *DL-SCH Channel Decoding.* The channel decoding is performed using `lteDLSCHDecode` which performs rate recovery, soft combining, code block desegmentation, CRC removal and block CRC decoding. This function takes a soft buffer as an input parameter which is then used in soft combining with received codeword soft bits prior the decoding of bits.

```
% Define soft buffer
decState = [];

% Noise power can be varied to see the different RV
SNR = 4; % dB

% Initial value
blkCRCerr = 1;

while blkCRCerr >= 1

    % Increment redundancy version for every retransmission
    rvIndex = rvIndex + 1;
    if rvIndex > length(redundancyVersions)
        error('Failed transmission');
    end
    pdsch.RV = redundancyVersions(rvIndex);

    % PDSCH payload
    codedTrBlock = lteDLSCH(enb, pdsch, codedTrBlkSize, ...
                   dlschTransportBlk);

    % PDSCH symbol generation
    pdschSymbols = ltePDSCH(enb, pdsch, {codedTrBlock});

    % Add noise to pdschSymbols to create noisy complex modulated symbols
    pdschSymbolsNoisy = awgn(pdschSymbols,SNR);

    % PDSCH receiver processing
    rxCW = ltePDSCHDecode(enb, pdsch, pdschSymbolsNoisy);

    % DL-SCH channel decoding
    [rxBits, blkCRCerr, decState] = lteDLSCHDecode(enb, ...
        pdsch, transportBlkSize, rxCW, decState);

end
```

`blkCRCerr` is the block CRC error for the received transport block. The UE sends a NACK if it detects CRC error on received transport block. Also, the new soft buffer, `decState`, contents are available at the output of this function to be used next time around.

Using this example it is possible to observe the effect that the noise has on the number of retransmissions required for successful reception. In this example, for the given level of noise added to the transmitted symbols, a total of 1 retransmission is required to successfully receive the data.

```
fprintf(['\n\nTransmission successful, total number of Redundancy ' ...
    'Versions used is ' num2str(redundancyVersions(rvIndex) + 1) ' \n\n']);
```

```
Transmission successful, total number of Redundancy Versions used is 2
```

Selected Bibliography

1. 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception"
2. 3GPP TS 36.212 "Multiplexing and channel coding"
3. 3GPP TS 36.213 "Physical layer procedures"

PDCCH Blind Search and DCI Decoding

The example aids understanding of the control region used in an LTE downlink subframe and its channel structure by showing how a downlink control information (DCI) message is generated and transmitted over a physical downlink control channel (PDCCH) and recovered by performing blind decoding using the LTE Toolbox™.

Introduction

To support the transmission of downlink and uplink transport channels Downlink Shared Channel(DL-SCH) and Uplink Shared Channel (UL-SCH) control signaling is required. This control signaling enables a UE to successfully receive, demodulate, and decode the DL-SCH. The Downlink Control Information (DCI) is transmitted through the Physical Downlink Control Channel (PDCCH) and includes information about the DL-SCH resource allocation (the set of resource blocks containing the DL-SCH), transport format and information related to the DL-SCH Hybrid Automatic Repeat reQuest (ARQ).

To form the PDCCH payload the DCI undergoes channel coding: addition of a CRC attachment followed by convolutional coding and rate matching according to PDCCH format capacity. The coded DCI bits i.e. PDCCH payload, are then mapped to Control Channel Elements (CCEs) according to the PDCCH format. These coded bits are then converted to complex modulated symbols after performing operations including scrambling, QPSK modulation, layer mapping and precoding. Finally, the modulated symbols are interleaved and mapped to physical Resource Elements (REs).

After performing deinterleaving, deprecoding, symbol combining, symbol demodulation and descrambling at the receiver, the UE is required to perform blind decoding of the PDCCH payload as it is not aware of the detailed control channel structure, including the number of control channels and the number of CCEs to which each control channel is mapped. Multiple PDCCHs can be transmitted in a single subframe which may and may not be all relevant to a particular UE. The UE finds the PDCCH specific to it by monitoring a set of PDCCH candidates (a set of consecutive CCEs on which a PDCCH could be mapped) in every subframe. The UE uses its Radio Network Temporary Identifier (RNTI) to try and decode candidates. The RNTI is used to demask a PDCCH candidate's CRC. If no CRC error is detected the UE determines that PDCCH carries its own control information.

Control Region

Downlink control signaling is located at the start of each downlink subframe (up to the first three OFDM symbols). One of the advantages of transmitting control channel at the start of every subframe is if the UE is not scheduled it may turn off its receiver circuitry for larger part of the subframe which results in reduced power consumption. Downlink control signaling is carried by three physical channels. The Physical Control Format Indicator Channel (PCFICH) to indicate the number of OFDM symbols used for control signaling in this subframe, Physical Hybrid-ARQ Indicator Channel (PHICH) which carries downlink Acknowledgment (ACK)/Negative Acknowledgment (NACK) for uplink data transmission and Physical Downlink Common Control Channel (PDCCH) which carries the downlink scheduling assignment and uplink scheduling grants.

The PDCCH carries scheduling assignments and other control information in the form of DCI messages. A PDCCH is transmitted on one CCE or an aggregation of several consecutive CCEs, where a CCE corresponds to 9 Resource Element Groups (REGs). In PDCCH transmission, only those REGs are used which are not assigned to PCFICH or PHICH. Each REG contains 4 Resource Elements (REs). Thus, REGs are used for defining the mapping of control channels to resource elements.

The control region of a downlink subframe comprises the multiplexing of all PDCCHs bits into a single block of data which is subsequently processed to form complex modulated symbols. These symbols are then divided to form blocks of complex-valued symbols quadruplets. These quadruplets are then interleaved and cyclically shifted prior to PDCCH resource mapping.

PDCCH and DCI Formats

The number of CCEs in a PDCCH transmission depends upon the PDCCH format, which can be 0, 1, 2, and 3 depending upon number of bits to be transmitted. The PDCCH bits are created from a DCI message after performing CRC attachment, channel coding and rate matching. Multiple PDCCHs can be transmitted in a subframe thus the UE must monitor all PDCCH in given subframe control region.

The DCI message transmits uplink or downlink scheduling information or an uplink Transmit Power Control (TPC) command. Depending on the purpose of control message, different DCI formats are defined. The information provided contains everything necessary for the UE to be able to identify the resources required to receive the Physical Downlink Data Channel (PDSCH) in that subframe and to decode it. The DCI formats are:

- Format 0 for transmission of Uplink Shared Channel (UL-SCH) allocation
- Format 1 for transmission of DL-SCH allocation for Single Input Multiple Output (SIMO) operation
- Format 1A for compact transmission of DL-SCH allocation for SIMO operation or allocating a dedicated preamble signature to a UE for random access
- Format 1B for transmission control information of Multiple Input Multiple Output (MIMO) rank 1 based compact resource assignment
- Format 1C for very compact transmission of PDSCH assignment
- Format 1D same as format1B with additional information of power offset
- Format 2 and Format 2A for transmission of DL-SCH allocation for closed and open loop MIMO operation, respectively
- Format 2B for the scheduling of dual layer transmission (antenna ports 7 & 8)
- Format 2C for the scheduling of up to 8 layer transmission (antenna ports 7 to 14) using TM9
- Format 2D for the scheduling of up to 8 layer transmission (antenna ports 7 to 14) using TM10
- Format 3 and Format 3A for transmission of TPC command for an uplink channel
- Format 4 for the scheduling of PUSCH with multi-antenna port transmission mode

Search Space and PDCCH Candidates

Once the DCI message is generated and channel coded according to the required DCI and PDCCH format, PDCCH multiplexing, scrambling, modulation, precoding, interleaving and layer mapping are performed to form complex symbols. These complex symbols are now ready to be mapped onto REs. The REs are defined in terms of REGs/CCEs allocated to a transmission. The number of CCEs allocated is given by the PDCCH format. The control region of a subframe is a collection of CCEs and can contain PDCCHs for multiple UEs thus the UE has to monitor a large area to extract its own control information. As the UE is not explicitly informed of the detailed control channel structure it has to blindly attempt to decode the control region. Unfortunately, this may impose a substantial burden on the UE as at large bandwidths the control region may be very large. This may exceed practical hardware limitations and lead to increased cost and/or reduced performance of the UE.

To simplify the decoding task at the UE, the whole control region is sub-divided into common and UE-specific search spaces which the UE should monitor (attempt to decode each of the PDCCHs). Each

space comprises 2, 4 or 6 PDCCH candidates whose data length depends on the PDCCH format; each PDCCH must be transmitted on 1, 2, 4 or 8 CCE(s) (1 CCE = 9 REGs = 9*4 REs = 72 bits).

The PDCCH candidates consist of consecutive CCEs. The candidates within the PDCCH candidate set do not have to be unique, especially for smaller bandwidths. The common and UE-specific search spaces can overlap with each other. The size of search space is determined by the number of PDCCH candidates and the size of CCE aggregation level. That is, the size of the search space is integer times the size of CCE aggregation level or the number of PDCCH candidates.

Common Search Space

The common search space carries the common control information and is monitored by all UEs in a cell. The number of CCE aggregation levels supported by the common search space is limited to two i.e. 4 and 8 as compared to the UE-specific search space where four CCE aggregation levels are possible. This reduces the burden on UE for decoding common control information compared to decoding UE-specific control information. The common control space is used to carry important initial information including paging information, system information and random access procedures.

When searching the common control space the decoder always starts decoding from the first CCE. This restriction further simplifies the common search. The decoding is done on every possible PDCCH candidate set for given PDCCH format until it successfully decodes the PDCCH present in common search space.

UE-Specific Search Space

The UE-specific search space carries control information specific to a particular UE and is monitored by at least one UE in a cell. Unlike the common space search, the starting location of the UE-specific search space may be varied for each subframe or UE. The starting location of the UE-specific search space is determined in every subframe using a hash function, as specified in TS36.213, Clause 9 [1].

In the UE-specific search space the UE finds its PDCCH by monitoring a set of PDCCH candidates (a set of consecutive CCEs on which PDCCH could be mapped) in every subframe. If no CRC error is detected when the UE uses its RNTI to demask the CRC (16-bit value also refers as C-RNTI) on a PDCCH, the UE determines that PDCCH carries its own control information. The PDCCH candidate sets correspond to different PDCCH formats. There are 4 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 other PDCCH formats. This process is repeated for all possible PDCCH formats until all directed PDCCHs are successfully decoded in UE-specific search space.

Example DCI Generation, Transmission and Recovery

In this example, a control channel containing a downlink control information (DCI) message is generated and transmitted over a physical downlink control channel (PDCCH). Once the PDCCH payload is generated, this example demonstrates how blind decoding is performed to decode the PDCCH in a given subframe.

Cell-Wide Settings

A structure `enbConfig` is used to configure the eNodeB.

```
enbConfig.NDLRB = 6;                % No of Downlink RBs in total BW
enbConfig.CyclicPrefix = 'Normal';  % CP length
enbConfig.CFI = 3;                  % 4 PDCCH symbols as NDLRB <= 10
enbConfig.Ng = 'Sixth';             % HICH groups
enbConfig.CellRefP = 1;             % 1-antenna ports
```

```
enbConfig.NCellID = 10;             % Physical layer cell identity
enbConfig.NSubframe = 0;            % Subframe number 0
enbConfig.DuplexMode = 'FDD';       % Frame structure
```

DCI Message Generation

Generate a DCI message to be mapped to the PDCCH.

```
dciConfig.DCIFormat = 'Format1A';   % DCI message format
dciConfig.Allocation.RIV = 26;      % Resource indication value

% Create DCI message for given configuration
[dciMessage, dciMessageBits] = lteDCI(enbConfig, dciConfig);
```

DCI Channel Coding

DCI message channel coding includes the following operations: CRC insertion, tail-biting convolutional coding and rate matching. The field `PDCCHFormat` indicates that one control channel element (CCE) is used for the transmission of PDCCH, where a CCE is composed of 36 useful resource elements.

```
C_RNTI = 100;                       % 16-bit UE-specific mask
pdcchConfig.RNTI = C_RNTI;          % Radio network temporary identifier
pdcchConfig.PDCCHFormat = 0;        % PDCCH format

% DCI message bits coding to form coded DCI bits
codedDciBits = lteDCIEncode(pdcchConfig, dciMessageBits);
```

PDCCH Bits Generation

The capacity of the control region depends on the bandwidth, the Control Format Indicator (CFI), the number of antenna ports and the HICH groups. The total number of resources available for PDCCH can be calculated using `ltePDCCHInfo`.

Not all the available bits in the PDCCH region are necessarily used. Therefore the convention adopted is to set unused bits to -1, while bit locations with values 0 or 1 are used. Initially, all elements are initialized with -1 to indicate that all the bits are unused.

```
pdcchDims = ltePDCCHInfo(enbConfig);

% Initialize elements with -1 to indicate that all the bits are unused
pdcchBits = -1*ones(pdcchDims.MTot, 1);

% Perform search space for UE-specific control channel candidates.
candidates = ltePDCCHSpace(enbConfig, pdcchConfig, {'bits', '1based'});

% Map PDCCH payload on available UE-specific candidate. In this example the
% first available candidate is used to map the coded DCI bits.
pdcchBits ( candidates(1, 1) : candidates(1, 2) ) = codedDciBits;
```

PDCCH Complex-Valued Modulated Symbol Generation

From the set of bits used in `pdcchBits` (values not set to -1) the PDCCH complex symbols are generated. The following operations are required: scrambling, QPSK modulation, layer mapping, precoding and interleaving.

```
pdcchSymbols = ltePDCCH(enbConfig, pdcchBits);
```

Noise Addition

The PDCCH complex symbols are then passed through an AWGN channel. The channel is generated using `randn`; its variance can be configured by `nVariance`. The variation in number of retransmissions can be simulated using wide range of `nVariance` parameter.

```
nVariance = 0.01; % Noise power
noise = complex(randn(size(pdcchSymbols))*sqrt(nVariance/2), ...
        randn(size(pdcchSymbols))*sqrt(nVariance/2));  % Generate noise
pdcchSymbolsNoisy = pdcchSymbols + noise;    % Add noise to PDSCH symbols
```

PDCCH Decoding

Perform PDCCH receiver processing including deinterleaving, cyclic shifting, deprecoding, layer demapping, QPSK soft demodulation and descrambling.

```
recPdcchBits = ltePDCCHDecode(enbConfig, pdcchSymbolsNoisy);
```

Blind Decoding Using DCI Search

The UE is only informed of the number of OFDM symbols within the control region of a subframe and is not provided with the location of its corresponding PDCCH. The UE finds its PDCCH by monitoring a set of PDCCH candidates in every subframe. This is referred to as blind decoding. The UE demasks each control candidate's CRC using its Radio Network Temporary Identifier (RNTI). If no CRC error is detected, the UE considers it as a successful decoding attempt and reads the control information within the successful candidate.

The eNodeB determines a PDCCH format to be transmitted to the UE, creates an appropriate DCI and attaches a CRC. The CRC is then masked with an RNTI according to the owner or usage of PDCCH. If the PDCCH is for a specific UE, the CRC will be masked with a UE unique identifier, for example a Cell-RNTI (C-RNTI). If the PDCCH contains paging information, the CRC will be masked with a paging indication identifier i.e. Paging-RNTI (P-RNTI). If the PDCCH contains system information, a system information identifier i.e. a system information-RNTI (SI-RNTI) will be used to mask the CRC.

With the possibilities of different RNTIs, PDCCH candidates, DCI and PDCCH formats, a significant number of attempts may be required to successfully decode the PDCCH. To overcome this complexity the UE first tries to blindly decode the first CCE in the control channel candidate set of a subframe. If the blind decoding fails, the UE tries to blindly decode the first 2, 4 then 8 CCEs sequentially, where the starting location is fixed for common search case and is given by hash function, as defined in TS36.213, Clause 9 [1], for UE-specific case.

`ltePDCCHSearch` first tries to decode PDCCHs in the common search space before trying in UE-specific search space. When searching the common search space it iterates for only two aggregation levels i.e. 4 and 8 and tries decode all PDDCH candidates for all possible common space DCI formats. The UE-specific search is carried out on four aggregation levels i.e. 1, 2, 4 and 8. The PDCCH candidates are generated using `ltePDCCHSpace`. If no CRC error is detected during a decoding attempt, the UE considers it a successful decoding and reads decoded DCI message.

`decDCI` is a cell array of structures containing the fields associated with one or more decoded DCI message(s). As multiple PDCCHs can be transmitted in a subframe thus UE has to monitor all possible PDCCHs directed at it.

`decDCIBits` is a cell array containing one or more vectors of bit values corresponding to successfully decoded DCI messages.

```
ueConfig.RNTI = C_RNTI;
ueConfig.ControlChannelType = 'PDCCH';
ueConfig.EnableCarrierIndication = 'Off';
ueConfig.SearchSpace = 'UESpecific';
ueConfig.EnableMultipleCSIRequest = 'Off';
ueConfig.EnableSRSRequest = 'Off';
ueConfig.NTxAnts = 1;
[rxDCI, rxDCIBits] = ltePDCCHSearch(enbConfig, ueConfig, recPdcchBits);
decDCI = rxDCI{1};          % Decoded DCI information
decDCIBits = rxDCIBits{1};  % Decoded DCI bits
```

Display the Recovered RIV

```
fprintf(['\n\n Blind Decoding successful, the recovered resource '...
       'allocation is ' num2str(decDCI.Allocation.RIV) ' \n\n']);
```

```
 Blind Decoding successful, the recovered resource allocation is 26
```

Selected Bibliography

1 3GPP TS 36.213 "Physical layer procedures"

Enhanced Physical Downlink Control Channel (EPDCCH) Generation

This example shows how to generate an Enhanced Physical Downlink Control Channel (EPDCCH) transmission using the LTE Toolbox™.

Introduction

This example shows how to generate a downlink transmission including EPDCCH, EPDCCH Demodulation Reference Signal (DMRS), Cell-Specific Reference Signal (CRS) and Channel State Information Reference Signal (CSI-RS). Beamforming of the maEPDCCH is included for both Localized and Distributed transmission. The output of the example is a resource grid populated with the transmitted channels and an OFDM modulated time-domain waveform which transmits that resource grid. A plot is also produced which details the Resource Element (RE) usage for each of the channels.

eNodeB Configuration

Cell-wide settings are configured with a structure `enb`. This structure contains parameters that belong to the eNodeB.

```
% Number of downlink resource blocks corresponding to 5MHz bandwidth
enb.NDLRB = 25;

% Duplexing mode: 'FDD' or 'TDD'
enb.DuplexMode = 'FDD';

% Number of Cell-specific Reference Signal (CRS) antenna ports
enb.CellRefP = 1;

% Subframe number
enb.NSubframe = 0;

% Cyclic prefix length: 'Normal' or 'Extended'
enb.CyclicPrefix = 'Normal';

% Frame number
enb.NFrame = 0;

% Cell identity
enb.NCellID = 0;

% Channel State Information Reference Signal (CSI-RS) subframe schedule
enb.CSIRSPeriod = 'On';

% Configuration index of the CSI-RS
enb.CSIRSConfig = 1;

% Number of CSI-RS antenna ports in use with this configuration
enb.CSIRefP = 2;

% Zero-Power CSI-RS subframe schedule
enb.ZeroPowerCSIRSPeriod = 'Off';
```

EPDCCH Configuration

A structure `chs` is created, containing parameters relevant to the transmission of a DCI message which will be coded and modulated on the EPDCCH channel. Unlike the structure `enb` above which configures cell-wide settings, the structure here configures channel-specific settings for the transmission of a particular channel, in this case the EPDCCH. (The structure name `chs` is an abbreviation of "channel-specific".)

```
% DCI format to send on the EPDCCH
chs.DCIFormat = 'Format1A';

% Radio Network Temporary Identifier (RNTI)
chs.RNTI = 1;

% Transmission type: 'Localized' or 'Distributed'
chs.EPDCCHType = 'Localized';

% Zero-based indices of PRB pair set associated with EPDCCH search space
chs.EPDCCHPRBSet = 4:5;

% Initial transmission symbol for EPDCCH transmission
chs.EPDCCHStart = 2;

% Scrambling identity for the EPDCCH
chs.EPDCCHNID = 0;

% EPDCCH format
chs.EPDCCHFormat = 1;
```

Subframe Resource Grid Creation

An empty resource grid `subframe` is created for one subframe. In this example, the 3rd dimension (planes) of this resource grid are intended to represent physical antennas. The mapping between antenna ports and physical antennas for the various channels and signals used in this example will be described when the channels and signals are mapped to this resource grid. The empty subframe resource grid is created using the `lteDLResourceGrid` function. The optional second input argument allows the 3rd dimension size (the number of planes) to be specified explicitly (by default the value of `enb.CellRefP`, the number of CRS ports, is used to determine the number of planes). Note that for distributed EPDCCH transmission, two antenna ports are used and therefore a minimum of two physical antennas are required; if the number of CRS ports or CSI-RS ports is greater than 2 then the largest of these values is used, this allows the resource grid to contain all configured CRS or CSI-RS ports.

```
maxEpdcchPorts = 2;
nTxAnts = max([enb.CSIRefP enb.CellRefP maxEpdcchPorts]);
subframe = lteDLResourceGrid(enb,nTxAnts);
```

DCI Message Creation

A DCI message of the format indicated by `chs.DCIFormat` is created using the function `lteDCI`. The output structure `dci` represents the DCI message as described in TS36.212 Section 5.3.3.1 [3]. The bit vector `dciBits` contains the actual message bits to be encoded.

```
% Create a DCI message
[dci,dciBits] = lteDCI(enb,chs);
```

EPDCCH Candidate Selection

The EPDCCH is transmitted in an "EPDCCH candidate", a set of Enhanced Control Channel Elements (ECCEs). Each ECCE maps onto a set of Enhanced Resource Element Groups (EREGs), which in turn map onto particular resource elements in the subframe resource grid. In order to create the resource element indices for a particular EPDCCH transmission, an EPDCCH candidate must be chosen. The function `lteEPDCCHSpace` creates a matrix of valid EPDCCH candidates for the given configuration, with each row of the matrix giving the inclusive [begin,end] indices of a single EPDCCH candidate. Any row of this matrix can therefore be extracted and assigned to the `chs.EPDCCHECCE` parameter field which will be used by the function `ltePDCCHIndices` to determine the resource element indices for the EPDCCH transmission.

```
candidates = lteEPDCCHSpace(enb,chs);

chs.EPDCCHECCE = candidates(3,:);
```

EPDCCH Data Bit Capacity

In order to determine the EPDCCH data bit capacity, the function `lteEPDCCHIndices` is used, which creates the resource element indices for the EPDCCH, following TS36.211 Section 6.8A.5 [3]; these indices will be used later for mapping the EPDCCH transmission to the subframe resource grid. This function also returns a structure containing useful EPDCCH "dimensionality information" values including `epdcchInfo.EPDCCHG`, the EPDCCH data bit capacity, which will be used to configure the rate matching when encoding the DCI message.

```
% Calculate EPDCCH resource element indices and associated dimensionality
% information including the EPDCCH data bit capacity epdcchInfo.EPDCCHG
[epdcchIndices,epdcchInfo] = lteEPDCCHIndices(enb,chs);
```

DCI Message Encoding

Next, the bit vector `dciBits` is passed to the function `lteDCIEncode` which performs CRC insertion, tail-biting convolutional coding and rate matching, following TS36.212 Sections 5.3.3.2 to 5.3.3.4 [3]. Note that the third argument to `lteDCIEncode` specifies the rate matching capacity (for non-enhanced PDCCH transmissions, this third argument can be omitted and the rate matching capacity is derived internally from the PDCCH format).

```
% Perform DCI message encoding with a rate matching output size equal to
% the EPDCCH data bit capacity
codedDciBits = lteDCIEncode(chs,dciBits,epdcchInfo.EPDCCHG);
```

EPDCCH Modulation

The EPDCCH modulation is performed using the function `lteEPDCCH`, following TS36.211 Sections 6.8A-2 to 6.8A-4 [2]. The resulting symbols `epdcchSymbols` will be mapped to the subframe resource grid after appropriate beamforming. The EPDCCH is transmitted on a subset of the antenna ports 107...110:

- For `chs.EPDCCHType='Localized'`, the EPDCCH is transmitted on a single antenna port chosen from 107...110 as a function of a number of parameters including the RNTI.
- For `chs.EPDCCHType='Distributed'`, the EPDCCH is transmitted on two antenna ports, either {107,109} for normal cyclic prefix or {107,108} for extended cyclic prefix.

The indices produced by the `lteEPDCCHIndices` function map antenna ports 107...110 (0-based) to planes 1...4 (1-based) of a subframe resource grid. A plane is the third dimension of a subframe

resource grid as described in the “Represent Resource Grids” documentation. Typically LTE Toolbox functions separate the symbols and indices for different antenna ports into separate columns. However for the EPDCCH the symbols and their indices are presented in a single column and the indices therein correspond to the appropriate antenna port for each resource element. This approach is taken for two reasons:

- The number of EPDCCH symbols mapped to each antenna port may be different.
- The mapping between symbols and antenna ports is dependent on many parameters and to represent this in `lteEPDCCH` would result in a very significant increase in the number of parameters required by this function.

Beamforming of the EPDCCH for transmission upon physical antennas will be described later.

```
epdcchSymbols = lteEPDCCH(enb,chs,codedDciBits);
```

EPDCCH DMRS Modulation

The DMRS associated with the EPDCCH, `epdcchDmrsSymbols`, is created using the function `lteEPDCCHDMRS`, following TS36.211 Section 6.10.3A.1 [2]. The associated resource element indices, `epdcchDmrsIndices`, are also created using the function `lteEPDCCHDMRSIndices`, following TS36.211 Section 6.10.2.A.2 [2]. The arrangement of DMRS symbols and their indices for different antenna ports mirrors that described for the EPDCCH symbols and indices above. Beamforming of the EPDCCH DMRS for transmission upon physical antennas will be described later.

```
epdcchDmrsSymbols = lteEPDCCHDMRS(enb,chs);
epdcchDmrsIndices = lteEPDCCHDMRSIndices(enb,chs);
```

Generate CRS and CSI-RS

The CRS and CSI-RS signals and their corresponding resource element indices are created, and the signals are mapped into the subframe resource grid:

- The indices produced by `lteCellRSIndices` map antennas ports 0...3 (0-based) to planes 1...4 (1-based) of the subframe resource grid. In this example, a single CRS port (`enb.CellRefP=1`) is configured and therefore the CRS will be mapped to the first plane of the subframe resource grid.
- The indices produced by `lteCSIRSIndices` map antenna ports 15...22 (0-based) to planes 1...8 (1-based) of the subframe resource grid. In this example, two CSI-RS ports (`enb.CSIRefP=2`) are configured and therefore the CSI-RS will be mapped to the first two planes of the subframe resource grid.

This mapping matches the generic beamforming model described in TS36.101 Annex B.4.3 [1].

```
% Create CSI-RS and map to the subframe resource grid
csirsIndices = lteCSIRSIndices(enb);
csirsSymbols = lteCSIRS(enb);
subframe(csirsIndices) = csirsSymbols;

% Create CRS and map to the subframe resource grid
crsIndices = lteCellRSIndices(enb);
crsSymbols = lteCellRS(enb);
subframe(crsIndices) = crsSymbols;
```

Beamforming of EPDCCH Transmission

The EPDCCH and its DMRS must now be beamformed and mapped to physical antennas for transmission. The beamforming vectors here are chosen in accordance with TS36.101 Annex B.4.4 [1] for distributed transmission and TS36.101 Annex B.4.5 [1] for localized transmission.

In preparation for beamforming, the EPDCCH symbols are concatenated with the EPDCCH DMRS symbols and the corresponding indices are also concatenated. The EPDCCH and its DMRS must undergo the same beamforming, therefore they can be processed together when applying the beamforming.

```
% Concatenate EPDCCH symbols/indices with EPDCCH DMRS symbols/indices to
% facilitate beamforming
allSymbols = [epdcchSymbols; epdcchDmrsSymbols];
allIndices = [epdcchIndices; epdcchDmrsIndices];
```

The complete set of indices `allIndices` is then converted to subscripts and the subscripts are processed to obtain a list of active EPDCCH antenna ports `ports` and active EPDCCH resource blocks `rbs`. `ports` and `rbs` will be used to control two loops which carry out the beamforming on a per port and per resource block basis.

```
% Determine the set of EPDCCH antenna ports 'ports' and resource blocks
% 'rbs' used by the EPDCCH and its DMRS; the value of 4 below is the
% number of EPDCCH antenna ports (107...110) on which an EPDCCH might be
% transmitted (i.e. 'allIndices' may generally contain indices for any of
% the 4 EPDCCH antenna ports).
[K,L,~] = size(subframe);
[resubs,~,portsubs] = ind2sub([K L 4],allIndices);
rbsubs = floor((resubs-1)/12)+1;
rbs = unique(rbsubs);
ports = unique(portsubs.');
```

For localized EPDCCH transmission, the beamforming described in TS36.101 Annex B.4.5 [1] uses a single random beamforming vector across all resource blocks.

```
% For localized transmission, a single beamforming vector 'W' is used for
% the single EPDCCH port and across all resource blocks
if (strcmpi(chs.EPDCCHType,'Localized'))
    codebookIdx = randi([0 3],1);
    W = lteDLPrecode(1,nTxAnts,'SpatialMux',codebookIdx);
end
```

For distributed EPDCCH transmission, the beamforming described in TS36.101 Annex B.4.4 [1] uses a different beamforming vector `W` for each resource block and each of the two antenna ports used. Therefore the beamforming vector selection is carried out in loops across the set of resource blocks and ports. Note that the loops below are applicable for both distributed and localized transmission: for localized transmission the beamforming vector selected above is applied to each resource block (for the single port used), whereas for distributed transmission the beamforming vector is both selected and applied for each resource block and antenna port. The code is structured as follows:

For each antenna port and for each Resource Block (RB):

- the vector of logical values `thisport` is true for EPDCCH/DMRS symbols which apply to the current antenna port.
- the vector of logical values `thisrb` is true for EPDCCH/DMRS symbols which apply to the current RB.

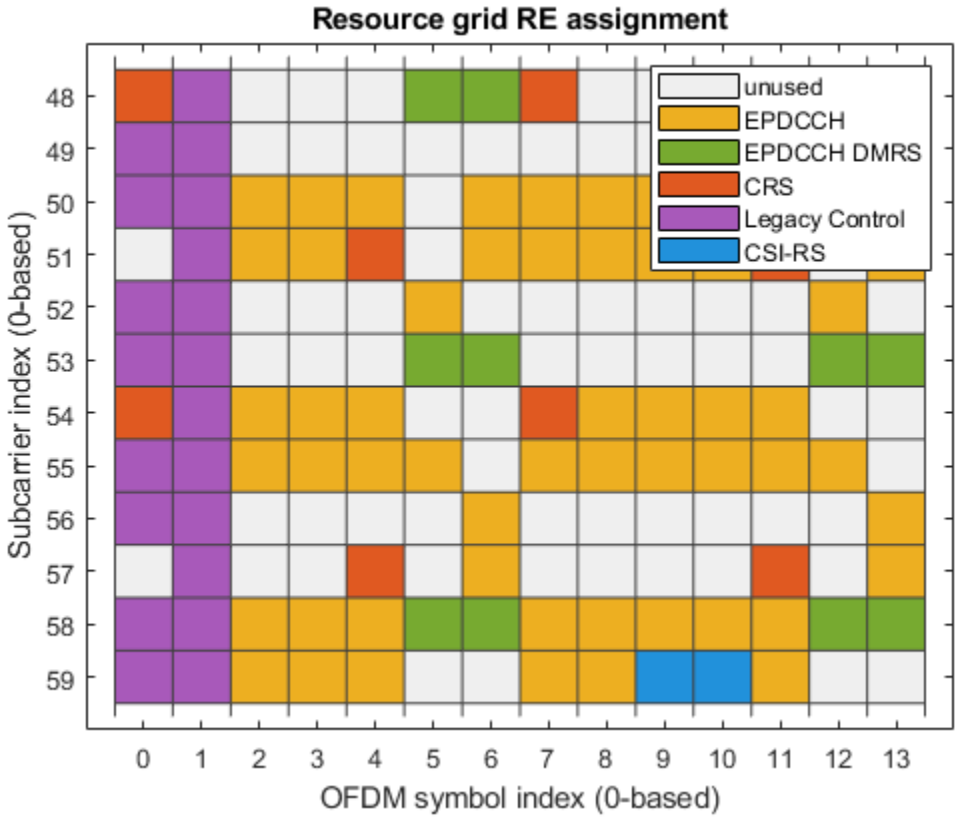
- For distributed transmission, select a beamforming vector `W`
- Apply beamforming to the EPDCCH/DMRS symbols and EPDCCH/DMRS indices for the current antenna port and current RB. To assist with the beamforming of indices, the function `lteExtractResources` is used. The call to this function takes the resource elements indices for a given antenna port as the input. The first output of the function, which is skipped here, is the set of REs for all physical antennas in the subframe resource grid `subframe` in the same time/frequency locations as the input indices. The second output, which is used as our beamformed indices, contains the corresponding indices for the REs in the first output, essentially a projection of the input indices onto all physical antennas. The symbols and indices for the current antenna port and current RB are obtained by using the logical vectors `thisport` and `thisrb` to extract the appropriate elements of the EPDCCH/DMRS symbols `allSymbols` and EPDCCH/DMRS indices `allIndices`.
- Map the beamformed EPDCCH/DMRS to the subframe resource grid.

```
% Matrix to store the codebook index used for each resource block and
% antenna port
codebookIdxs = zeros(length(rbs),length(ports));

% For each EPDCCH antenna port in use:
for p=1:length(ports)
    thisport = (portsubs==ports(p));

    % For each RB in use:
    for r = 1:length(rbs)
        thisrb = (rbsubs==rbs(r));

        % For distributed transmission, a beamforming vector 'W' is
        % selected for each EPDCCH port and each resource block; the
        % beamforming vector chosen for a particular resource block must be
        % different on each of the antenna ports
        if (strcmpi(chs.EPDCCHType,'Distributed'))
            unusedIdxs = setxor(0:3,codebookIdxs(r,1:p-1));
            codebookIdx = unusedIdxs(randi(length(unusedIdxs),1));
            W = lteDLPrecode(1,nTxAnts,'SpatialMux',codebookIdx);
        end

        % Record the codebook index used for this resource block and
        % antenna port
        codebookIdxs(r,p) = codebookIdx;

        % Apply beamforming to the symbols and indices
        bfSymbols = allSymbols(thisport & thisrb) * W;
        [~,bfIndices] = ...
            lteExtractResources(allIndices(thisport & thisrb),subframe);

        % Map the beamformed symbols to the physical antennas. The symbols
        % must be added to the existing subframe resource grid because
        % different ports may share the same time/frequency locations and
        % therefore the beamformed symbols must be superposed.
        subframe(bfIndices) = subframe(bfIndices) + bfSymbols;

    end

end
```

Plot Resource Element Assignments

The subframe resource grid is plotted to indicate the locations of the EPDCCH, EPDCCH DMRS, CRS and CSI-RS. If `chs.EPDCCHStart>0`, the location of the legacy control region is also shown.

```
hEPDCCHGenerationPlot(enb,chs);
```



OFDM Modulation

Finally the subframe resource grid is OFDM modulated; the resulting matrix has two columns; each column contains the complex baseband time-domain waveform samples for each physical antenna.

```
waveform = lteOFDMModulate(enb,subframe);
```

Appendix

This example uses the helper function:

- hEPDCCHGenerationPlot.m

Selected Bibliography

1 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception"

2 3GPP TS 36.211 "Physical channels and modulation"

3 3GPP TS 36.212 "Multiplexing and channel coding"

Uplink Waveform Modeling Using SRS and PUCCH

This example demonstrates how to configure User Equipment (UE) and cell-specific Sounding Reference Signals (SRS) transmission. Physical Uplink Control Channel (PUCCH) is also configured for transmission.

Introduction

The SRS configuration is split into 2 parts - UE-specific and cell-specific. The UE-specific part describes the schedule and content of actual SRS transmissions for this UE. The cell-specific part describes the time schedule when any UE in the cell can transmit - the UE-specific schedule must be a subset of this schedule.

In this example the cell-specific SRS configuration has 5ms periodicity with an offset of 0 (signaled by `srs.SubframeConfig = 3` as indicated in TS36.211, Table 5.5.3.3-1 [1]). The UE-specific SRS configuration has 10ms periodicity with an offset of 0 (signaled by `srs.ConfigIdx = 7` as indicated in TS36.213, Table 8.2-1 [2]). The cell-specific configuration means that for this cell, two opportunities for SRS transmission exist within each frame, subframes 0 and 5. All UEs in the cell must shorten their Physical Uplink Control Channel (PUCCH) transmissions during these subframes to allow for SRS reception without interference, even if they are not transmitting SRS themselves. The UE-specific configuration means that this UE is configured to generate SRS only in subframe 0.

The output at the MATLAB® command window when running this example shows PUCCH transmission in all 10 subframes, with shortening in subframes 0 and 5, and an SRS transmission in subframe 0.

UE Configuration

```
ue = struct;
ue.NULRB = 15;                  % Number of resource blocks
ue.NCellID = 10;                % Physical layer cell identity
ue.Hopping = 'Off';             % Disable frequency hopping
ue.CyclicPrefixUL = 'Normal';   % Normal cyclic prefix
ue.DuplexMode = 'FDD';          % Frequency Division Duplex (FDD)
ue.NTxAnts = 1;                 % Number of transmit antennas
ue.NFrame = 0;                  % Frame number
```

PUCCH Configuration

```
pucch = struct;
% Vector of PUCCH resource indices, one per transmission antenna
pucch.ResourceIdx = 0:ue.NTxAnts-1;
pucch.DeltaShift = 1;               % PUCCH delta shift parameter
pucch.CyclicShifts = 0;             % PUCCH delta offset parameter
pucch.ResourceSize = 0;             % Size of resources allocated to PUCCH
```

SRS Configuration

```
srs = struct;
srs.NTxAnts = 1;        % Number of transmit antennas
srs.SubframeConfig = 3; % Cell-specific SRS period = 5ms, offset = 0
srs.BWConfig = 6;       % Cell-specific SRS bandwidth configuration
srs.BW = 0;             % UE-specific SRS bandwidth configuration
srs.HoppingBW = 0;      % SRS frequency hopping configuration
srs.TxComb = 0;         % Even indices for comb transmission
```

```
srs.FreqPosition = 0;   % Frequency domain position
srs.ConfigIdx = 7;      % UE-specific SRS period = 10ms, offset = 0
srs.CyclicShift = 0;    % UE-cyclic shift
```

Subframe Loop

The processing loop generates a subframe at a time. These are all concatenated to create the resource grid for a frame (10 subframes). The loop performs the following operations:

- *SRS Information*: By calling `lteSRSInfo` we can get information related to SRS for a given subframe. The `IsSRSSubframe` field of the structure `srsInfo` returned from the `lteSRSInfo` call indicates if the current subframe (given by `ue.NSubframe`) is a cell-specific SRS subframe (`IsSRSSubframe = 1`) or not (`IsSRSSubframe = 0`). The value of this field can be copied into the `ue.Shortened` field. This ensures that the subsequent PUCCH generation will correctly respect the cell-specific SRS configuration for all subframes, omitting the last symbol of the PUCCH in the cell-specific SRS subframes.
- *PUCCH 1 Demodulation Reference Signal (DRS) Generation and Mapping*: The DRS signal is located in the 3rd, 4th and 5th symbols of each slot and therefore never has the potential to collide with the SRS.
- *PUCCH 1 Generation and Mapping*: Unlike the DRS, the PUCCH 1 transmission can occupy the last symbol of the subframe unless `ue.Shortened = 1`. In this case the last symbol of the subframe will be left empty.
- *SRS Generation and Mapping*: Here we generate and map the SRS according to the UE-specific SRS configuration. Both the `lteSRSIndices` and `lteSRS` functions use the fields `ue.NSubframe` and `srs.ConfigIdx` to determine if the current subframe is configured for SRS transmission; if not, the output of both functions is empty.

```
txGrid = [];    % Create empty resource grid

for i = 1:10    % Process 10 subframes

        % Configure subframe number (0-based)
        ue.NSubframe = i-1;
        fprintf('Subframe %d:\n',ue.NSubframe);

        % Establish if this subframe is a cell-specific SRS subframe,
        % and if so configure the PUCCH for shortened transmission
        srsInfo = lteSRSInfo(ue, srs);
        ue.Shortened = srsInfo.IsSRSSubframe; % Copy SRS info to ue struct

        % Create empty uplink subframe
        txSubframe = lteULResourceGrid(ue);

        % Generate and map PUCCH1 DRS to resource grid
        drsIndices = ltePUCCH1DRSIndices(ue, pucch);% DRS indices
        drsSymbols = ltePUCCH1DRS(ue, pucch);       % DRS sequence
        txSubframe(drsIndices) = drsSymbols;        % Map to resource grid

        % Generate and map PUCCH1 to resource grid
        pucchIndices = ltePUCCH1Indices(ue, pucch); % PUCCH1 indices
        ACK = [0; 1];                               % HARQ indicator values
        pucchSymbols = ltePUCCH1(ue, pucch, ACK);   % PUCCH1 sequence
        txSubframe(pucchIndices) = pucchSymbols;    % Map to resource grid
        if (ue.Shortened)
```

```
            disp('Transmitting shortened PUCCH');
        else
            disp('Transmitting full-length PUCCH');
        end

        % Configure the SRS sequence group number (u) according to TS
        % 36.211 Section 5.5.1.3 with group hopping disabled
        srs.SeqGroup = mod(ue.NCellID,30);

        % Configure the SRS base sequence number (v) according to TS 36.211
        % Section 5.5.1.4 with sequence hopping disabled
        srs.SeqIdx = 0;

        % Generate and map SRS to resource grid
        % (if active under UE-specific SRS configuration)
        [srsIndices, srsIndicesInfo] = lteSRSIndices(ue, srs);% SRS indices
        srsSymbols = lteSRS(ue, srs);                         % SRS seq.
        if (srs.NTxAnts == 1 && ue.NTxAnts > 1) % Map to resource grid
            % Select antenna for multiple antenna selection diversity
            txSubframe( ...
                hSRSOffsetIndices(ue, srsIndices, srsIndicesInfo.Port)) = ...
                srsSymbols;
        else
            txSubframe(srsIndices) = srsSymbols;
        end
        % Message to console indicating when SRS is mapped to the resource
        % grid.
        if(~isempty(srsIndices))
            disp('Transmitting SRS');
        end

        % Concatenate subframes to form frame
        txGrid = [txGrid txSubframe]; %#ok
end
```

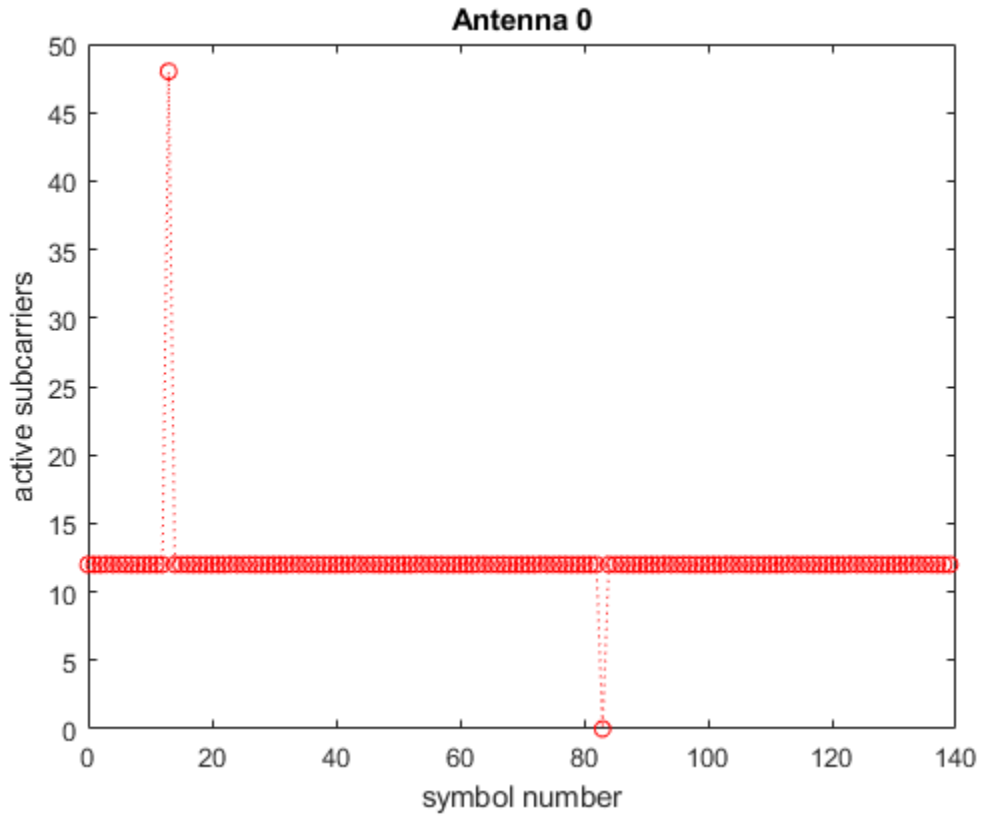
```
Subframe 0:
Transmitting shortened PUCCH
Transmitting SRS
Subframe 1:
Transmitting full-length PUCCH
Subframe 2:
Transmitting full-length PUCCH
Subframe 3:
Transmitting full-length PUCCH
Subframe 4:
Transmitting full-length PUCCH
Subframe 5:
Transmitting shortened PUCCH
Subframe 6:
Transmitting full-length PUCCH
Subframe 7:
Transmitting full-length PUCCH
Subframe 8:
Transmitting full-length PUCCH
Subframe 9:
Transmitting full-length PUCCH
```

Results

The figure produced shows the number of active subcarriers in each SC-FDMA symbol across the 140 symbols in `txGrid`. All SC-FDMA symbols contain 12 active subcarriers corresponding to the single resource block bandwidth of the PUCCH except:

- symbol 13, the last symbol of subframe 0 which has 48 active subcarriers corresponding to an 8 resource block SRS transmission
- symbol 83, the last symbol of subframe 5 which has 0 active subcarriers corresponding to the shortened PUCCH (last symbol empty) to allow for potential SRS transmission by another UE in this cell.

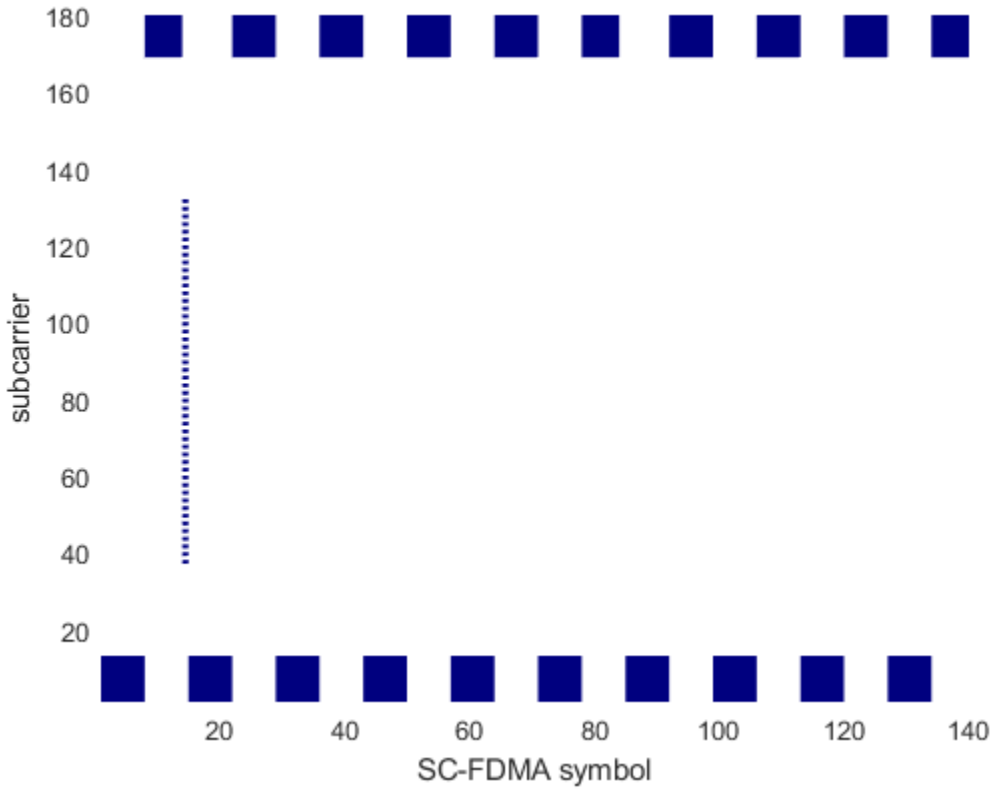
```
figure;
for i = 1:ue.NTxAnts
    subplot(ue.NTxAnts,1,i);
    plot(0:size(txGrid,2)-1,sum(abs(txGrid(:,:,i)) ~= 0),'r:o')
    xlabel('symbol number');
    ylabel('active subcarriers');
    title(sprintf('Antenna %d',i-1));
end
```



Plot the resource grid with the PUCCH at the band edges and the SRS comb transmission in subframe 0.

```
figure;
pcolor(abs(txGrid));
colormap([1 1 1; 0 0 0.5])
```

```
shading flat;
xlabel('SC-FDMA symbol'); ylabel('subcarrier')
```



Further Exploration

SRS transmit antenna selection can be demonstrated by setting `ue.NTxAnts = 2` and examining the subplots produced for each antenna; the SRS is transmitted on antenna 0 while the PUCCH is shortened on both (all) antennas. A pattern of antenna selection across this one-frame run can be shown by further configuring `srs.SubframeConfig = 0` and `srs.ConfigIdx = 0`. This configures a cell-specific SRS configuration of 2ms periodicity with an offset of 0 (signaled by `srs.SubframeConfig = 0`) and also a UE-specific SRS configuration of 2ms periodicity with an offset of 0 (signaled by `srs.ConfigIdx = 0`). In this case an SRS is transmitted by this UE on even subframes, and the transmit antenna alternates with each transmission.

SRS transmission on multiple antennas using resource diversity can be shown by setting `ue.NTxAnts = 2` and `srs.NTxAnts = 2`. In this case the SRS is always transmitted on both (all) antennas with orthogonal resources on each antenna.

Appendix

This example uses this helper function.

- hSRSOffsetIndices.m

Selected Bibliography

1. 3GPP TS 36.211 "Physical channels and modulation"
2. 3GPP TS 36.213 "Physical layer procedures"

Mixed PUCCH Format Transmission and Reception

This example shows the transmission and reception of Physical Uplink Control Channel (PUCCH) Formats 1 and 2, including the case where the same physical resource is shared between transmissions of Format 1 and Format 2 simultaneously from two different User Equipments (UEs) using the LTE Toolbox™.

Introduction

This example configures two User Equipments (UEs) to transmit a Physical Uplink Control Channel (PUCCH) Format 1 signal from the first UE and a PUCCH Format 2 signal from the second UE. Appropriate Demodulation Reference Signals (DRS) are also generated. The transmitted signals are passed through two different fading channels and added, together with Additive White Gaussian Noise (AWGN), simulating the reception of the signals from the two UEs at an eNodeB. Each signal (i.e. that belonging to each UE) is then synchronized, SC-FDMA demodulated, equalized, PUCCH demodulated and then finally decoded. A plot is produced showing that the channels can be estimated independently for the two different signals, even though they share the same physical Resource Elements (REs).

UE 1 Configuration

The first UE is configured using a structure `ue1`.

```
ue1.NULRB = 6;                  % Number of resource blocks
ue1.NSubframe = 0;              % Subframe number
ue1.NCellID = 10;               % Physical layer cell identity
ue1.RNTI = 61;                  % Radio network temporary identifier
ue1.CyclicPrefixUL = 'Normal';  % Cyclic prefix
ue1.Hopping = 'Off';            % Frequency hopping
ue1.Shortened = 0;              % Reserve last symbols for SRS transmission
ue1.NTxAnts = 1;                % Number of transmit antennas
```

UE 2 Configuration

Similarly a configuration structure is used to configure the second UE, `ue2`. This structure is identical to the configuration of `ue1` with two exceptions:

- No `Shortened` field as this does not apply to PUCCH Format 2.
- A different Radio Network Temporary Identifier (RNTI) value (not used here as it is only relevant for Physical Uplink Shared Channel (PUSCH) transmission, but different UEs would have different RNTI).

```
ue2.NULRB = 6;                  % Number of resource blocks
ue2.NSubframe = 0;              % Subframe number
ue2.NCellID = 10;               % Physical layer cell identity
ue2.RNTI = 77;                  % Radio network temporary identifier
ue2.CyclicPrefixUL = 'Normal';  % Cyclic prefix
ue2.Hopping = 'Off';            % Frequency hopping
ue2.NTxAnts = 1;                % Number of transmit antennas
```

PUCCH 1 Configuration

For the first UE, a PUCCH of Format 1 is used, so an appropriate configuration structure `pucch1` is created. The parameter `CyclicShifts` specifies the number of cyclic shifts used by PUCCH Format

1 in resource blocks where a mixture of PUCCH Format 1 and PUCCH Format 2 are to be transmitted. The parameter `ResourceSize` specifies the size of the resources used by PUCCH Format 2, effectively determining the starting position of PUCCH Format 1 transmissions; here we specify `ResourceIdx=0` which will use the first PUCCH Format 1 resource.

```
pucch1.ResourceIdx = 0;   % PUCCH resource index
pucch1.DeltaShift = 1;    % Delta shift
pucch1.CyclicShifts = 1;  % Number of cyclic shifts
pucch1.ResourceSize = 0;  % Size of resources allocated to PUCCH Format 2
```

PUCCH 2 Configuration

For the second UE, a PUCCH of Format 2 is used, so an appropriate configuration structure `pucch2` is created. The values of parameters `CyclicShifts` and `ResourceSize` are the same as in the PUCCH Format 1 configuration. The value of `ResourceIdx` is set to the first PUCCH Format 2 resource, meaning that the physical resource blocks now configured for PUCCH Format 1 and PUCCH Format 2 will be the same.

```
pucch2.ResourceIdx = 0;   % PUCCH resource index
pucch2.CyclicShifts = 1;  % Number of cyclic shifts
pucch2.ResourceSize = 0;  % Size of resources allocated to PUCCH Format 2
```

Channel Propagation Model Configuration

The propagation channel that the two UEs will transmit through is configured using a structure `channel`. The sampling rate of the channel is configured to match the sampling rate at the output of the first UE; note that the same sampling rate is used at the output of the second UE because `ue1.NULRB` and `ue2.NULRB` are the same. When we use this channel configuration for each UE, the `Seed` parameter of the structure will be set differently for each UE so that different propagation conditions result.

```
channel.NRxAnts = 4;                % Number of receive antennas
channel.DelayProfile = 'ETU';       % Delay profile
channel.DopplerFreq = 300.0;        % Doppler frequency
channel.MIMOCorrelation = 'Low';    % MIMO correlation
channel.InitTime = 0.0;             % Initialization time
channel.NTerms = 16;                % Oscillators used in fading model
channel.ModelType = 'GMEDS';        % Rayleigh fading model type
channel.InitPhase = 'Random';       % Random initial phases
channel.NormalizePathGains = 'On';  % Normalize delay profile power
channel.NormalizeTxAnts = 'On';     % Normalize for transmit antennas

% Set sampling rate
info = lteSCFDMAInfo(ue1);
channel.SamplingRate = info.SamplingRate;
```

Noise Configuration

The SNR is given by SNR = E_s/N_0 where E_s is the energy of the signal of interest and N_0 is the noise power. The power of the noise to be added can be determined so that E_s and N_0 are normalized after the SC-FDMA demodulation to achieve the desired SNR `SNRdB`. The noise added before SC-FDMA demodulation will be amplified by the IFFT. The amplification is the square root of the size of the IFFT. In this simulation this is taken into consideration by dividing the desired noise power by this value. In addition, because real and imaginary parts of the noise are created separately before being combined into complex additive white Gaussian noise, the noise amplitude must be scaled by $1/\sqrt{2}$ so the generated noise power is 1.

```
SNRdB = 21.0;

% Normalize noise power
SNR = 10^(SNRdB/20);
N = 1/(SNR*sqrt(double(info.Nfft)))/sqrt(2.0);

% Configure random number generators
rng('default');
```

Channel Estimation Configuration

The channel estimator is configured using a structure `cec`. Here cubic interpolation will be used with an averaging window of 12-by-1 REs. This configures the channel estimator to use a special mode which ensures the ability to despread and orthogonalize the different overlapping PUCCH transmissions.

```
cec = struct;                     % Channel estimation config structure
cec.PilotAverage = 'UserDefined'; % Type of pilot averaging
cec.FreqWindow = 12;              % Frequency averaging window in REs (special mode)
cec.TimeWindow = 1;               % Time averaging window in REs (Special mode)
cec.InterpType = 'cubic';         % Cubic interpolation
```

PUCCH Format 1 Generation

Now all the necessary configuration is complete, the PUCCH Format 1 and its DRS are generated. The PUCCH Format 1 carries the HARQ indicators `hi1` and in this case there are 2 indicators, meaning that the transmission will be of Format 1b. The PUCCH Format 1 DRS carries no data.

```
% PUCCH 1 modulation/coding
hi1 = [0; 1];  % Create HARQ indicators
disp('hi1:');
```

```
hi1:
```

```
disp(hi1.');
```

```
     0     1
```

```
pucch1Sym = ltePUCCH1(ue1, pucch1, hi1);

% PUCCH 1 DRS creation
pucch1DRSSym = ltePUCCH1DRS(ue1, pucch1);
```

PUCCH Format 2 Generation

The PUCCH Format 2 DRS carries the HARQ indicators `hi2` and in this case there are 2 indicators, meaning that the transmission will be of Format 2b. The PUCCH Format 2 itself carries coded Channel Quality Information (CQI). The information `cqi` here is coded and then modulated.

```
% PUCCH 2 DRS modulation
hi2 = [1; 1];  % Create HARQ indicators
disp('hi2:');
```

```
hi2:
```

```
disp(hi2.');
```

```
     1     1
```

```
pucch2DRSSym = ltePUCCH2DRS(ue2, pucch2, hi2);

% PUCCH 2 coding
cqi = [0; 1; 1; 0; 0; 1];  % Create channel quality information
disp('cqi:');

cqi:

disp(cqi.');

     0     1     1     0     0     1

codedcqi = lteUCIEncode(cqi);

% PUCCH 2 modulation
pucch2Sym = ltePUCCH2(ue2, pucch2, codedcqi);
```

PUCCH Index Generation

The indices for the PUCCH and PUCCH DRS transmissions are created

```
pucch1Indices = ltePUCCH1Indices(ue1, pucch1);
pucch2Indices = ltePUCCH2Indices(ue2, pucch2);

pucch1DRSIndices = ltePUCCH1DRSIndices(ue1, pucch1);
pucch2DRSIndices = ltePUCCH2DRSIndices(ue2, pucch2);
```

Transmission for UE 1

The overall signal for the first UE is now transmitted. The steps are to map the PUCCH Format 1 and corresponding DRS signal into an empty resource grid, perform SC-FDMA modulation and then transmit through a fading propagation channel.

```
% Create resource grid
grid1 = lteULResourceGrid(ue1);
grid1(pucch1Indices) = pucch1Sym;
grid1(pucch1DRSIndices) = pucch1DRSSym;

% SC-FDMA modulation
txwave1 = lteSCFDMAModulate(ue1, grid1);

% Channel modeling. An additional 25 samples added to the end of the
% waveform to cover the range of delays expected from the channel modeling
% (a combination of implementation delay and channel delay spread)
channel.Seed = 13;
rxwave1 = lteFadingChannel(channel,[txwave1; zeros(25,1)]);
```

Transmission for UE 2

The overall signal for the second UE is now transmitted. Note that a different random seed `channel.Seed` is used compared to that used for the first UE. This ensures that different propagations are used for the two transmissions.

```
% Create resource grid
grid2 = lteULResourceGrid(ue2);
grid2(pucch2Indices) = pucch2Sym;
grid2(pucch2DRSIndices) = pucch2DRSSym;

% SC-FDMA modulation
```

```
txwave2 = lteSCFDMAModulate(ue2, grid2);

% Channel modeling. An additional 25 samples added to the end of the
% waveform to cover the range of delays expected from the channel modeling
% (a combination of implementation delay and channel delay spread)
channel.Seed = 15;
rxwave2 = lteFadingChannel(channel, [txwave2; zeros(25, 1)]);
```

Reception at the Base Station

The input to the base station receiver is modeled by adding the two faded signals together with Gaussian noise with power as described above.

```
rxwave = rxwave1 + rxwave2;

% Add noise
noise = N*complex(randn(size(rxwave)), randn(size(rxwave)));
rxwave = rxwave + noise;
```

Synchronization and SC-FDMA Demodulation for UE 1

The uplink frame timing estimate for UE1 is calculated using the PUCCH 1 DRS signals and then used to demodulate the SC-FDMA signal. The resulting grid `rxgrid1` is a 3 dimensional matrix. The number of rows represents the number of subcarriers. The number of columns equals the number of SC-FDMA symbols in a subframe. The number of subcarriers and symbols is the same for the returned grid from `lteSLSCFDMADemodulate` as the grid passed into `lteSLSCFDMAInfo`. The number of planes (3rd dimension) in the grid corresponds to the number of receive antennas.

```
% Synchronization
offset1 = lteULFrameOffsetPUCCH1(ue1, pucch1, rxwave);

% SC-FDMA demodulation
rxgrid1 = lteSCFDMADemodulate(ue1, rxwave(1+offset1:end, :));
```

Channel Estimation and Equalization for UE 1

An estimate of the channel between each transmitter and the base station receiver is obtained and used to equalize its effects. To create an estimation of the channel `lteULChannelEstimatePUCCH1` is used. The channel estimation function is configured by the structure `cec`. The function returns a 3-D matrix of complex weights which are applied to each resource element by the channel in the transmitted grid. The 1st dimension is the subcarrier, the 2nd dimension is the SC-FDMA symbol and the 3rd dimension is the receive antenna. The effect of the channel on the received resource grid is equalized using `lteEqualizeMMSE`. This function uses the estimate of the channel (`H1`) to equalize the received resource grid (`rxGrid1`).

```
% Channel estimation
[H1, n0] = lteULChannelEstimatePUCCH1(ue1, pucch1, cec, rxgrid1);

% Extract REs corresponding to the PUCCH from the given subframe across all
% receive antennas and channel estimates
[pucchrx1, pucchH1] = lteExtractResources(pucch1Indices, rxgrid1, H1);

% Equalization
eqgrid1 = lteULResourceGrid(ue1);
eqgrid1(pucch1Indices) = lteEqualizeMMSE(pucchrx1, pucchH1, n0);
```

PUCCH 1 Decoding

Finally the PUCCH Format 1 channel is decoded and the useful HARQ indicator bits are extracted.

```
rxhi1 = ltePUCCH1Decode(ue1, pucch1, length(hi1), ...
    eqgrid1(pucch1Indices));
disp('rxhi1:');
```

```
rxhi1:
```

```
disp(rxhi1.');
```

```
   0   1
```

Receiver for UE 2

The uplink frame timing estimate for UE2 is calculated using the PUCCH 2 DRS signals and then used to demodulate the SC-FDMA signal. In this case, the Hybrid ARQ indicators as conveyed on the PUCCH Format 2 DRS are also found. The resulting grid `rxgrid2` is a 3 dimensional matrix. To create an estimation of the channel `lteULChannelEstimatePUCCH2` is used. The effect of the channel on the received resource grid is equalized using lteEqualizeMMSE. Finally the PUCCH Format 2 channel is decoded and the useful CQI information bits are extracted.

```
% Synchronization (and PUCCH 2 DRS demodulation/decoding)
[offset2,rxhi2] = lteULFrameOffsetPUCCH2(ue2,pucch2,rxwave,length(hi2));
disp('rxhi2:');
```

```
rxhi2:
```

```
disp(rxhi2.');
```

```
   1   1
```

```
% SC-FDMA demodulation
rxgrid2 = lteSCFDMADemodulate(ue2, rxwave(1+offset2:end, :));

% Channel estimation
[H2, n0] = lteULChannelEstimatePUCCH2(ue2, pucch2, cec, rxgrid2, rxhi2);

% Extract REs corresponding to the PUCCH from the given subframe across all
% receive antennas and channel estimates
[pucchrx2, pucchH2] = lteExtractResources(pucch2Indices, rxgrid2, H2);

% Equalization
eqgrid2 = lteULResourceGrid(ue2);
eqgrid2(pucch2Indices) = lteEqualizeMMSE(pucchrx2, pucchH2, n0);

% PUCCH 2 demodulation
rxcodedcqi = ltePUCCH2Decode(ue2, pucch2, eqgrid2(pucch2Indices));

% PUCCH 2 decoding
rxcqi = lteUCIDecode(rxcodedcqi, length(cqi));
disp('rxcqi:');
```

```
rxcqi:
```

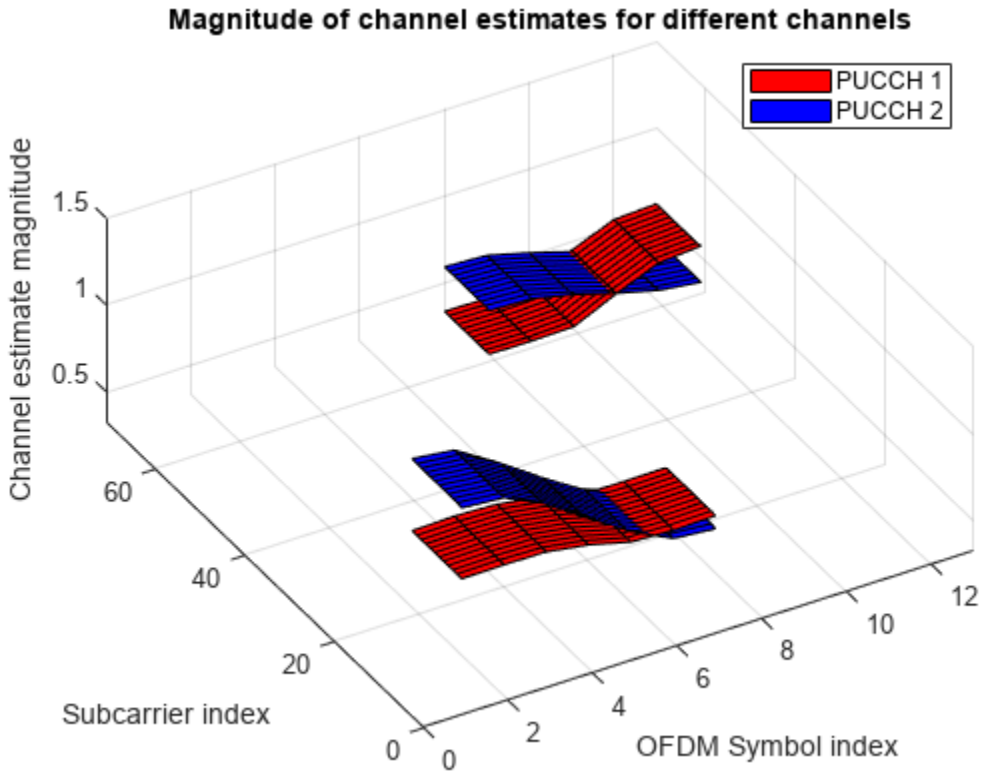
```
disp(rxcqi.');
```

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

Display Estimated Channels

A plot is produced showing that the channels can be estimated independently for the two different signals, even although they share the same physical REs. The PUCCH Format 1 channel estimate is shown in red and the PUCCH Format 2 channel estimate is shown in blue.

```
hPUCCHMixedFormatDisplay(H1, eqgrid1, H2, eqgrid2);
```



Appendix

This example uses this helper function.

- hPUCCHMixedFormatDisplay.m

LTE-M Uplink Waveform Generation

This example shows how to create an uplink LTE-M transmission consisting of the Physical Uplink Shared Channel (PUSCH) and associated demodulation reference signals (DM-RS) including repetitions and frequency hopping. When compared to pre-release 13 devices, Cat-M devices offer lower cost and complexity, enhanced coverage by the introduction of repetitions and an extended DRX for further power saving.

Introduction

The cell-specific subframe bitmap broadcasted on the System Information (SI) indicates which subframes are capable of LTE-M transmissions. LTE-M devices can optionally enable repetitions for the PUSCH and PUCCH to enhance coverage. The network configures a set of possible repetitions containing 4 values for CE mode A (`pusch-maxNumRepetitionCEmodeA` as given in TS 36.213 Table 8-2b [3]) and 8 values for CE mode B (`pusch-maxNumRepetitionCEmodeB` as given in TS 36.213 Table 8-2c [3]). From this set, the network dynamically selects the actual repetitions and signals this to the UE as part of the scheduling grant. Uplink scheduling grants for LTE-M devices are carried on the MPDCCH using DCI format 6-0A for devices operating in CE mode A and 6-0B for devices operating in CE mode B. A scheduling grant ending in downlink subframe n is valid for an uplink PUSCH transmission starting in uplink subframe n+4. In the case of transmissions with frequency hopping, the grant indicates the narrowband for the first transmission. Subsequent allocations can hop between narrowbands as defined in TS 36.211 Section 5.3.4 [1].

PUSCH

For Cat-M1 (Release 13) devices, PUSCH is always transmitted in a single narrowband. In Cat-M2 (Release 14), devices in CE mode A can optionally transmit over up to 24 PRBs if the higher layer parameter `ce-pusch-maxBandwidth-config` is set to 5MHz as indicated in TS 36.211 Section 5.3.4 [1]/TS 36.306 Section 4.3.4.64 [4]. LTE-M PUSCH can have up to 32 repetitions in CE mode A (see TS 36.213 Table 8-2b [3])and up to 2048 repetitions in CE mode B (see TS 36.213 Table 8-2c [3]). For BL/CE UE in CE mode A, PUSCH frequency hopping is enabled when the higher-layer parameter `pusch-HoppingConfig` is set and the frequency hopping flag in DCI format 6-0A indicates frequency hopping. For BL/CE UE in CE mode B, PUSCH frequency hopping is enabled when the higher-layer parameter `pusch-HoppingConfig` is set. If frequency hopping is not enabled for PUSCH, all PUSCH repetitions are located at the same PRB resources. If frequency hopping is enabled for PUSCH, PUSCH is transmitted in a subframe within the `NabsPUSCH` consecutive uplink subframes using the same number of consecutive PRBs as in the previous subframe starting from the same starting PRB resource within narrowband. If a resource assignment or frequency hopping would result in a PUSCH resource allocation outside the allocatable PRBs then the PUSCH transmission in that subframe is dropped.

```
ue = struct();                % Initialize the structure
ue.NULRB = 50;                % Bandwidth
ue.DuplexMode = 'FDD';        % Duplex mode
ue.TDDConfig = 1;             % UL/DL configuration if TDD duplex mode
ue.CyclicPrefixUL = 'Normal'; % The cyclic prefix length
ue.NCellID = 1;               % Cell identity
ue.RNTI = 1;                  % RNTI value
ue.NFrame = 0;                % Frame number
ue.NSubframe = 0;             % Subframe number
ue.NTxAnts = 1;               % Number of transmit antennas
ue.Shortened = 1;             % Last symbol availability (allocation for SRS)
```

```
% Set up hopping specific parameters
ue.HoppingOffset = 1;% Narrowband offset between one narrowband and the next narrowband
                     % a PUSCH hops to, expressed as a number of uplink narrowbands
ue.NChULNB = 2;      % Number of consecutive absolute subframes over which
                     % PUCCH or PUSCH stays at the same

pusch  = struct();
pusch.CEMode = 'A';         % CE mode A or CE mode B
pusch.Hopping = true;       % Enable/Disable frequency hopping
pusch.NRepPUSCH = 8;        % The total number of PUSCH repetitions
pusch.Modulation = 'QPSK';  % Symbol modulation
pusch.RV = 0;               % Redundancy version for UL-SCH processing
pusch.NLayers = 1;          % Number of layers
pusch.TrBlkSizes = 100;     % Transport block size
```

PUSCH Allocation - The PUSCH bandwidth is usually a single 1.4MHz narrowband. There are 6 RBs in each narrowband, all can be allocated in CE mode A and 1 or 2 RBs in CE mode B. An extended BW of 5MHz possible in Cat-M2 CE mode A configuration (See TS 36.306 Section 4.3.4.64 [4]). We use the `InitPRBSet` and `InitNarrowbandIndex` to specify the PRBs in a narrowband and the narrowband used in transmission. If frequency hopping is disabled, LTE-M PDSCH will be transmitted in the PRBs specified by the `InitPRBSet` and `InitNarrowbandIndex` parameters. If hopping is enabled, the hopping rules determine the narrowband used per subframe. The 5MHz bandwidth is inferred via the use of more than 6 PRBs in the `InitPRBSet` parameter. In this case the hopping will be disabled and the `InitNarrowbandIndex` ignored.

```
% Specify 1-based relative indices of RBs within a narrowband for all cases
% except 5MHz Cat-M2 CE mode A. If 5MHz Cat-M2 CE mode A, these are the
% absolute PRBs used for transmission
pusch.InitPRBSet = (2:3)';
% Narrowband used for transmission (non-hopping, non-5MHz)
pusch.InitNarrowbandIndex = 1;

% Specify the power scaling in dB for PUSCH, PUSCH DM-RS
pusch.PUSCHPower = 30;
pusch.PUSCHDMRSPower = 100;

% Turn off hopping if allocation spans multiple narrowbands
if numel(pusch.InitPRBSet) > 6
    pusch.Hopping = false;
end
```

UL-SCH Encoding

For BL/CE UEs in CE mode B, resource elements in the last SC-FDMA symbol in a subframe configured with cell specific SRS shall be counted in the PUSCH mapping but not used for transmission of the PUSCH. Hence if CE mode B, turn off shortening when creating the coded transport block.

```
% Identify all uplink subframes in a frame
info = arrayfun(@(x)lteDuplexingInfo(setfield(ue,'NSubframe',x)),0:9);
ulsfs = arrayfun(@(x)strcmpi(x.SubframeType,'Uplink'),info);
% In this example, we assume that the first absolute subframe in which
% PUSCH is transmitted is the first available uplink subframe
pusch.InitNSubframe = find(ulsfs,1)-1;

% Calculate the allocation
pusch.PRBSet = getPUSCHAllocation(ue,pusch);
```

```
ueTemp = ue;
% Create coded transport block for all symbols
if strcmpi(pusch.CEMode,'B') && ue.Shortened
    ueTemp.Shortened = 0;
end
[~,info] = ltePUSCHIndices(ueTemp,pusch);
% Define UL-SCH message bits
trData = ones(pusch.TrBlkSizes(1),1);
% Create the coded UL-SCH bits
pusch.BetaCQI = 2.0;
pusch.BetaRI = 2.0;
pusch.BetaACK = 2.0;
codedTrBlock = lteULSCH(ueTemp,pusch,info.G,trData);
```

LTE-M PUSCH Generation

In this example, we generate the LTE-M PUSCH and the corresponding DM-RS signals with repetitions and optional frequency hopping. `pusch.NRepPUSCH` controls the number of PUSCH repetitions. The UE-specific parameter `pusch.Hopping` enables hopping and the cell-wide parameters `ue.HoppingOffset` and `ue.NChULNB` defines the hopping pattern. In this example, if the allocation spans more than one narrowband, frequency hopping will be disabled. For LTE-M, the same scrambling sequence is applied per subframe to PUSCH for a block of `Nacc` subframes, all other processing stages i.e. symbol modulation, layer mapping, precoding and mapping to resource elements are the same for the LTE PUSCH.

```
% Number of subframes in a scrambling block
Nacc = 1;
if strcmpi(ue.DuplexMode,'FDD') && strcmpi(pusch.CEMode,'B')
    Nacc = 4;
elseif strcmpi(ue.DuplexMode,'TDD') && strcmpi(pusch.CEMode,'B')
    Nacc = 5;
end

% Total BL/CE subframes to simulate (all uplink subframes are BL/CE
% subframes) and the PUSCH is transmitted without any subframe gaps
totmtcSubframes = pusch.NRepPUSCH;

% Total absolute subframes to simulate
startSubframe = ue.NFrame*10+ue.NSubframe; % Initial absolute subframe number
lastabssf = getlastabsSF(ulsfs,pusch.InitNSubframe,totmtcSubframes);
totSubframes = lastabssf-startSubframe+1;

% Create a resource grid for the entire transmission. The PUSCH and
% DM-RS symbols will be mapped in this array
subframeSize = lteULResourceGridSize(ue);
sfgrid = zeros([subframeSize(1) subframeSize(2)*totSubframes subframeSize(3:end)]);

mpuschSym = []; % Initialize PUSCH symbols

for sf = startSubframe + (0:totSubframes -1)

    % Set current absolute subframe and frame numbers
    ue.NSubframe = mod(sf,10);
    ue.NFrame = floor((sf)/10);

    % Skip processing if this is not an uplink subframe
    duplexInfo = lteDuplexingInfo(ue);
```

```
    if ~strcmpi(duplexInfo.SubframeType,'Uplink')
        continue
    end

   % Calculate the PRBSet used in the current subframe
    prbset = getPUSCHAllocation(ue,pusch);

    % Calculate the PDSCH indices for the current subframe. For BL/CE UEs
    % in CE mode B, resource elements in the last SC-FDMA symbol in a
    % subframe configured with cell specific SRS shall be counted in the
    % PUSCH mapping but not used for transmission of the PUSCH
    pusch.PRBSet = prbset;
    mpuschIndices = ltePUSCHIndices(ue,pusch);

    % Create an empty subframe grid
    subframe = lteULResourceGrid(ue);

    % Encode PUSCH symbols from the codeword
    % In the case of repetition, the same symbols are repeated in each of
    % a block of NRepPUSCH subframes. Frequency hopping is applied as required
    if ~mod(sf,Nacc) || isempty(mpuschSym)
        ueTemp = ue;
        if strcmpi(pusch.CEMode,'B') && ue.Shortened
            ueTemp.Shortened = 0;  % Create symbols for full subframe
        end
        mpuschSym = ltePUSCH(ueTemp,pusch,codedTrBlock)*db2mag(pusch.PUSCHPower);
    end
    % Map SRS punctured PUSCH symbols to the subframe grid
    subframe(mpuschIndices) = mpuschSym(1:numel(mpuschIndices));


    % Create and map the DMRS symbols.
    ue.Hopping = 'Off';    % DRS hopping
    ue.SeqGroup = 0;       % PUSCH sequence group
    ue.CyclicShift = 0;    % Used for n1DMRS
    % For LTE-M UEs, a cyclic shift field of '000' shall be assumed when
    % determining n2DMRS from Table 5.5.2.1.1-1 of TS 36.211
    pusch.DynCyclicShift = 0; % Cyclic shift of '000' for n2DMRS
    pusch.OrthCover = 'Off';  % No orthogonal cover sequence
    mpuschDrs = ltePUSCHDRS(ue,pusch)*db2mag(pusch.PUSCHDMRSPower);
    mpuschDrsIndices = ltePUSCHDRSIndices(ue,pusch);
    subframe(mpuschDrsIndices) = mpuschDrs;

    % Now assign the current subframe into the overall grid
    sfgrid(:,(1:subframeSize(2))+sf*subframeSize(2),:) = subframe;

end
```

Create Time Domain Baseband Waveform

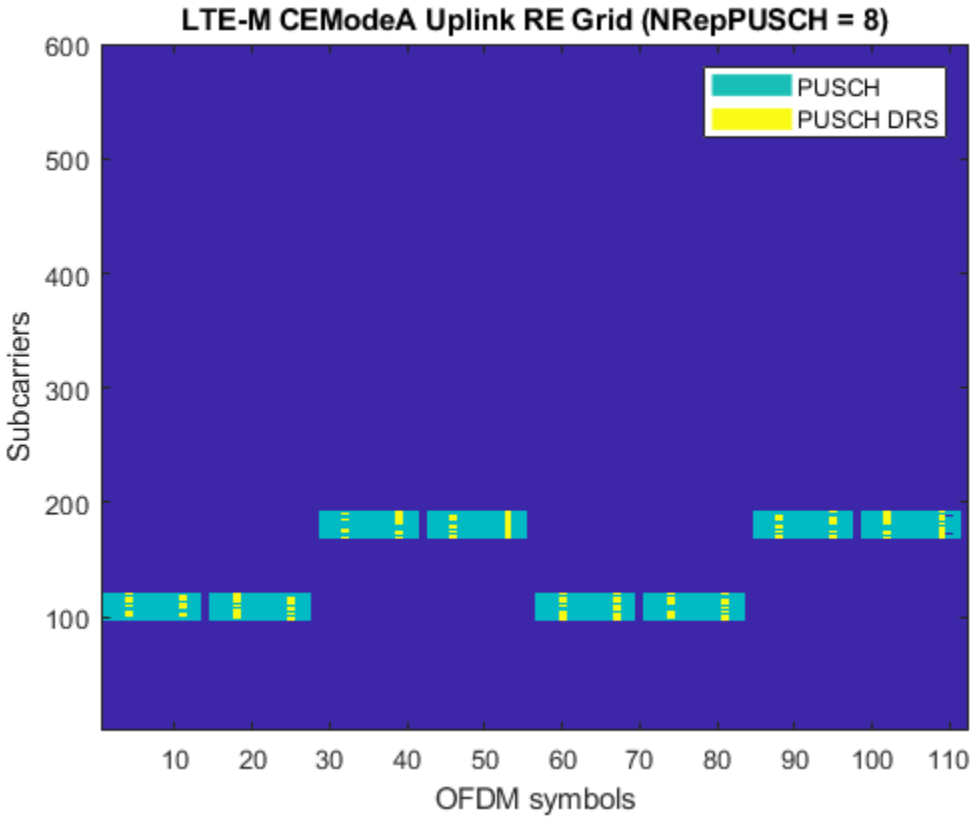
Create the time domain baseband waveform by OFDM modulating the resource grid. The resulting matrix has four columns; one of which will contain the complex baseband time-domain waveform samples for the MPDCCH

```
waveform = lteSCFDMAModulate(ue,sfgrid);
```

Plot Transmitted Grid and Baseband Waveform

Plot the grid and time domain baseband waveform. If the transmission uses more than one port, only the first port is shown. Note that the resource grid plot uses the power levels of the PUSCH and the DM-RS to assign colors to the resource elements.

```
% Create an image of overall resource grid. Since the PUSCH undergo
% transform precoding, we need to assign a single power level to all
% symbols to visualize in the plot
plotgrid = abs(sfgrid(:,:,1));
% Get the DM-RS positions
drspos = (plotgrid==db2mag(pusch.PUSCHDMRSPower));
plotgrid(drspos) = 0;
% Now set all PUSCH symbols to one power level to plot
plotgrid(plotgrid~=0) = db2mag(pusch.PUSCHPower);
% Now write back the DRS and plot
plotgrid(drspos) = db2mag(pusch.PUSCHDMRSPower);
figure
im = image(plotgrid);
cmap = parula(64);
colormap(im.Parent,cmap);
axis xy;
title(sprintf('LTE-M CEMode%s Uplink RE Grid (NRepPUSCH = %d)',pusch.CEMode,pusch.NRepPUSCH))
xlabel('OFDM symbols')
ylabel('Subcarriers')
% Create the legend box to indicate the channel/signal types associated with the REs
reNames = {'PUSCH';'PUSCH DRS'};
clevels = round(db2mag([pusch.PUSCHPower pusch.PUSCHDMRSPower]));
N = numel(reNames);
L = line(ones(N),ones(N), 'LineWidth',8); % Generate lines
% Set the colors according to cmap
set(L,{'color'},mat2cell(cmap( min(1+clevels,length(cmap) ),:),ones(1,N),3));   % Set the colors
legend(reNames{:});
```



Local Functions

The following local functions are used in this example:

- `calcNarrowbandPRBSets` - Calculate narrowbands and associated PRBs
- `getPDSCHAllocation` - Calculate the PUSCH subframe allocation
- `getlastabsSF` - Calculate the last subframe number for PUSCH

Selected Bibliography

1 3GPP TS 36.211 "Physical channels and modulation"

2 3GPP TS 36.212 "Multiplexing and channel coding"

3 3GPP TS 36.213 "Physical layer procedures"

4 3GPP TS 36.306 "User Equipment (UE) radio access capabilities"

5 3GPP TS 36.331 "Radio Resource Control (RRC) Protocol specification"

6 O. Liberg, M. Sundberg, Y.-P. Wang, J. Bergman and J. Sachs, Cellular Internet of Things: Technologies, Standards and Performance, Elsevier, 2018.

7 E. Dahlman, S. Parkvall and J Skold 4G LTE-Advanced Pro and The Road to 5G

Local Functions

```
% Calculate the widebands, narrowbands and PRBSets for the LTE carrier bandwidth
function [prbsets,nNB,nWB] = calcNarrowbandPRBSets(NULRB)
    % Narrowbands & Widebands (See 36.211 Section 5.2.4)
```

```
    NULNB = floor(NULRB/6);
    nNB = 0:(NULNB-1); % Narrowbands
    if NULNB >= 4
        NULWB = floor(NULNB/4);
    else
        NULWB = 1;
    end
    nWB = 0:(NULWB-1); % Widebands

    % PRBs in a narrowband
    ii = 0:5;
    ii0 = floor(NULRB/2) - 6*(NULNB/2);
    prbsets = zeros(6,numel(nNB));
    for nb = 1:numel(nNB)
        if mod(NULRB,2) && nNB(nb)>= (NULNB/2)
            prbsets(:,nb) = 6*(nNB(nb))+ii0+ii + 1;
        else
            prbsets(:,nb) = 6*(nNB(nb))+ii0+ii;
        end
    end
end

% Calculate the resource blocks allocated for PUSCH in the subframe
function prbset = getPUSCHAllocation(ue,chs)

    % If 5MHz mode (up to 24 PRBs can be used), the allocation is the same
    % as InitPRBSet
    if numel(chs.InitPRBSet) > 6
        prbset = chs.InitPRBSet;
        return;
    end

    % Get the narrowbands and corresponding resources
    [prbsets,nNB] = calcNarrowbandPRBSets(ue.NULRB);
    if max(chs.InitNarrowbandIndex) > max(nNB)
        error('Invalid narrowband(s) specified. There are only %d narrowbands in the bandwidth f
    end
    % If frequency hopping is disabled, the allocation is the same for
    % every subframe
    if ~chs.Hopping
        prbset = prbsets(chs.InitPRBSet,chs.InitNarrowbandIndex+1);
        return
    end

    % Hopping narrowband calculation according to TS 36.211 Section 5.3.4
    j0 = floor((chs.InitNSubframe)/ue.NChULNB);

    % Calculate the narrowband for this subframe
    ue.NSubframe = ue.NFrame*10+ue.NSubframe; % Get the absolute subframe number
    if mod(floor(ue.NSubframe/ue.NChULNB-j0),2) == 0
        nnBi = chs.InitNarrowbandIndex;
    else
        nnBi = mod(chs.InitNarrowbandIndex+ue.HoppingOffset,numel(nNB));
    end
    % Calculate the PRBSet for this subframe, they are on the same RBs
    % within the narrowband
    [rbstartIndex,nbstartIndex] = find(prbsets == chs.InitPRBSet(1));
    [rbendIndex,nbendIndex] = find(prbsets == chs.InitPRBSet(end));
```

```
        if (isempty(rbstartIndex) || isempty(rbendIndex)) || (nbstartIndex ~= nbendIndex)
           error('Invalid PRBSet specified, must be resources within single narrowband');
        end
        prbset = prbsets(rbstartIndex:rbstartIndex+numel(chs.InitPRBSet)-1,nnBi+1);

    end

    % Get the absolute subframe number which is used for the last transmission
    % of a channel
    function lastabssf = getlastabsSF(ulsfs,InitNSubframe,totmtcSubframes)

        numulsfsinFrame = sum(ulsfs); % Number of active sfs in a frame
        ulsfsinFrame = find(ulsfs);   % UL subframes in the frame (1-based)

        % Find the first absolute subframe and frame
        initabssf = mod(InitNSubframe,10);
        initabsf = floor(InitNSubframe/10);

        startIdxwithinFrame = initabssf+1; % 1-based index to the UL sf
        if ~ulsfs(startIdxwithinFrame)
            error(['Invalid absolute subframe number of the first uplink subframe', ...
                  ' intended for PUSCH (%d) specified. This is not an uplink subframe'],InitNSubframe
        end

        sfslastFrame = mod((find(ulsfsinFrame==startIdxwithinFrame)-1)+totmtcSubframes,numulsfsinFra
        if sfslastFrame
            % Find the subframe number corresponding to the last subframe to transmit
            sfsnumlastFrame = find(ulsfs,sfslastFrame)-1;
            sfsnumlastFrame = sfsnumlastFrame(end);
        else
            % No partial frames required
            sfsnumlastFrame = 0;
        end
        lastabssf = (initabsf + floor(((find(ulsfsinFrame==startIdxwithinFrame)-1)+totmtcSubframes)/

    end
```

Release 10 PUSCH Multiple Codeword Transmit and Receive Modeling

This example demonstrates the multicodeword transmission and reception in the uplink.

Introduction

This example shows how to implement multicodeword transmission and reception using LTE Toolbox™. This is done using Fixed Reference Channel (FRC) A3-2 as specified in TS36.104, Annex A3 [1]. The configuration is then amended to transmit two identically configured codewords.

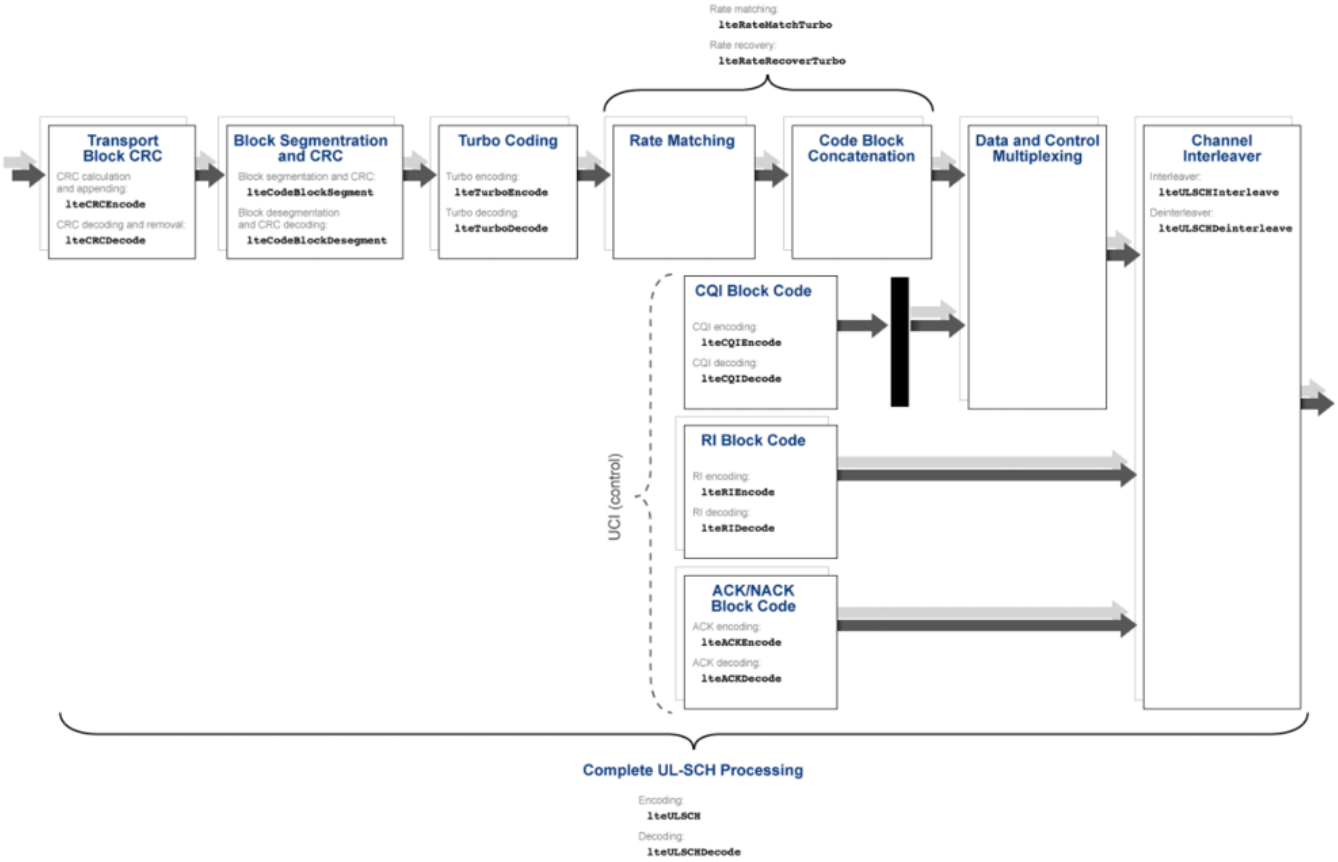
Setup

This section sets the User Equipment (UE) configuration structure associated to FRC A3-2 and modifies it to use 2 codewords. The configuration for the two codewords is identical.

```
% Generate configuration for FRC A3-2
frc = lteRMCUL('A3-2');

% UE configuration
frc.TotSubframes = 1;   % Total number of subframes
frc.NTxAnts = 2;        % Number of transmit antennas

% Update Physical Uplink Shared Channel (PUSCH) configuration for 2
% identically configured codewords
frc.PUSCH.NLayers = 2;
frc.PUSCH.Modulation = repmat({frc.PUSCH.Modulation},1,2);
frc.PUSCH.RV = repmat(frc.PUSCH.RV,1,2);
frc.PUSCH.TrBlkSizes = repmat(frc.PUSCH.TrBlkSizes,2,1);
```

Encoding

This section sets up the transport blocks and the Uplink Control Information (UCI). This is then coded to generate the Uplink Shared Channel (UL-SCH). The diagram below shows the operations performed internally by `lteULSCH`.



PUSCH modulation is applied to the generated codewords.

```
% Set up the transport block sizes and data for both codewords
TBSs = frc.PUSCH.TrBlkSizes(:,frc.NSubframe+1); % transport block sizes
trBlks = {(randi([0 1], TBSs(1), 1)) (randi([0 1], TBSs(2), 1))}; % data

% Set up UCI contents
CQI = [1 0 1 0 0 0 1 1 1 0 0 0 1 1].';
RI  = [0 1 1 0].';
ACK = [1 0].';

% UL-SCH coding including UCI coding
cws = lteULSCH(frc,frc.PUSCH,trBlks,CQI,RI,ACK);

% PUSCH modulation
puschSymbols = ltePUSCH(frc,frc.PUSCH,cws);
```

Decoding

This section demodulates the PUSCH and applies channel decoding. The resulting UCI is then decoded to produce the received Channel Quality Indicator (CQI), Rank Indication (RI) and Acknowledgment (ACK).

```
% PUSCH demodulation
ulschInfo = lteULSCHInfo(frc,frc.PUSCH,TBSs,length(CQI),length(RI),...
            length(ACK),'chsconcat');    % Get UL-SCH information
llrs = ltePUSCHDecode(frc,ulschInfo,puschSymbols); % Decode PUSCH
```

```
% UL-SCH decoding
softBuffer = [];
[rxtrblks,crc,softBuffer] = lteULSCHDecode(frc,ulschInfo,TBSs,llrs,...
                            softBuffer);

% UCI decoding
[llrsData,llrsCQI,llrsRI,llrsACK] = lteULSCHDeinterleave(frc,ulschInfo,...
                                    llrs);
rxCQI = lteCQIDecode(ulschInfo,llrsCQI);    % Decode CQI
rxRI = lteRIDecode(ulschInfo,llrsRI);       % Decode RI
rxACK = lteACKDecode(ulschInfo,llrsACK);    % Decode ACK
```

Results

The decoded CRC for both codewords is displayed. The transmitted and received CQI, RI and ACK bits are also shown.

```
hULMulticodewordTxRxDisplayResults(crc,CQI,RI,ACK,rxCQI,rxRI,rxACK);
```

```
CRCs:
Codeword 1: 0
Codeword 2: 0

CQI:
transmitted: 1  0  1  0  0  0  1  1  1  0  0  0  1  1
received   : 1  0  1  0  0  0  1  1  1  0  0  0  1  1

RI:
transmitted: 0  1  1  0
received   : 0  1  1  0

ACK:
transmitted: 1  0
received   : 1  0
```

Appendix

This example uses this helper function.

- hULMulticodewordTxRxDisplayResults.m

Selected Bibliography

1 3GPP TS 36.104 "Base Station (BS) radio transmission and reception"

Release 10 PUSCH Multiple Codeword Throughput Conformance Test

This example measures the throughput performance of the Physical Uplink Shared Channel (PUSCH) with closed-loop spatial multiplexing using the LTE Toolbox™ under a 2-codeword Release 10 UL-MIMO scenario, based on conformance test conditions as defined in Table 8.2.1.1-7 of TS 36.104.

Introduction

TS 36.104, Table 8.2.1.1-7 [1] defines a minimum fraction of 70% throughput for a physical uplink shared channel (PUSCH) transmission for a given signal-to-noise ratio (SNR) assuming hybrid automatic repeat request (HARQ) retransmissions. A normal cyclic prefix, Extended Pedestrian A (EPA5) propagation channel and Fixed Reference Channel (FRC) A3-2 are used, but amended to transmit two identically-configured codewords in parallel. In order to assist Precoding Matrix Indicator (PMI) selection for closed-loop spatial multiplexing, the Sounding Reference Signal (SRS) is transmitted which allows full-rank channel estimation even when the PUSCH is being transmitted with fewer transmit layers than transmit antennas.

If SRS is used the PUSCH capacity is shorted as the last symbol of the subframe where SRS is transmitted is not used for PUSCH in any User Equipment (UE) in the cell.

Simulation Configuration

The example is executed for a simulation length of 10 frames at an SNR of -0.1 dB as per TS 36.104, Table 8.2.1.1-7 [1]. A large number of `NFrames` should be used to produce meaningful throughput results. `SNRIn` can be an array of values or a scalar.

```
NFrames = 10; % Number of frames
SNRIn = -0.1; % SNR range in dB
```

UE Configuration

User Equipment (UE) settings are specified in a structure.

```
frc.RC = 'A3-2';                % FRC number
frc.DuplexMode = 'FDD';         % Duplex mode
frc.TotSubframes = 1;           % Total number of subframes to generate
frc.NCellID = 10;               % Cell identity
frc.CyclicPrefixUL = 'Normal';  % Uplink cyclic prefix length
frc.CyclicPrefix = 'Normal';    % Downlink cyclic prefix length
frc.NTxAnts = 4;                % Number of transmit antennas
```

Configure SRS within UE configuration to allow for channel estimation.

```
frc.SRS.SubframeConfig = 1;    % Cell-specific schedule: 2 ms periodicity
frc.SRS.NTxAnts = frc.NTxAnts; % Configure SORTD same as no of UE antennas
frc.SRS.BWConfig = 7;          % Cell-specific SRS bandwidth
frc.SRS.BW = 0;                % UE-specific SRS bandwidth
frc.SRS.ConfigIdx = 0;         % UE-specific schedule:  2ms periodicity
frc.SRS.CyclicShift = 0;       % Cyclic shift 0
frc.SRS.SeqGroup = 0;          % Sequence group 0
frc.SRS.SeqIdx = 0;            % Base sequence 0
frc.SRS.TxComb = 0;            % Transmission comb 0
frc.SRS.FreqPosition = 0;      % Frequency-domain position 0
frc.SRS.HoppingBW = 0;         % Disable hopping (as HoppingBW >=0 BW)
```

Set the PMI delay for the closed-loop spatial multiplexing. This is the delay of a PMI being passed from UE to eNodeB.

```
pmiDelay = 8;
```

Propagation Channel Model Configuration

Propagation channel model characteristics are set using a structure containing the fields specified below. These are set according to TS 36.104, Table 8.2.1.1-7 [1].

```
chcfg.NRxAnts = 4;                % Number of receive antennas
chcfg.DelayProfile = 'EPA';       % Delay profile
chcfg.DopplerFreq = 5.0;          % Doppler frequency
chcfg.MIMOCorrelation = 'Low';    % MIMO correlation
chcfg.Seed = 9;                   % Channel seed
chcfg.NTerms = 16;                % Oscillators used in fading model
chcfg.ModelType = 'GMEDS';        % Rayleigh fading model type
chcfg.InitPhase = 'Random';       % Random initial phases
chcfg.NormalizePathGains = 'On'; % Normalize delay profile power
chcfg.NormalizeTxAnts = 'On';     % Normalize for transmit antennas
```

Channel Estimator Configuration

The variable `perfectChanEstimator` controls channel estimator behavior. Valid values are `true` or `false`. When set to `true` a perfect channel estimator is used otherwise an imperfect estimate of the channel is used, based on the values of received pilot signals.

```
% Controls channel estimator behavior
perfectChanEstimator = false;      % Valid values are true or false
reference = 'Antennas';            % Channel reference
```

Imperfect channel estimation is configured using a structure `cec`. Here cubic interpolation will be used with an averaging window of 12-by-1 Resource Elements (REs). This configures the channel estimator to use a special mode which ensures the ability to despread and orthogonalize the different overlapping PUSCH transmissions.

```
% Channel estimator configuration
cec.PilotAverage = 'UserDefined'; % Type of pilot symbol averaging
cec.FreqWindow = 12;              % Frequency averaging window in REs (special mode)
cec.TimeWindow = 1;               % Time averaging window in REs (special mode)
cec.InterpType = 'Cubic';         % 2D interpolation type
```

RMC-UL Configuration

To generate the uplink Reference Model Channel (RMC) the LTE Toolbox functions `lteRMCUL` and `lteRMCULTool` are used. `lteRMCULTool` creates a configuration structure for given UE settings; specific to a given Fixed Reference Channel (FRC). This configuration structure is constructed as per TS 36.104, Annex A [1] and is used by `lteRMCULTool` to generate an SC-FDMA modulated waveform. The sub-structure `PUSCH` defines the parameters associated with PUSCH and contains the vector defining the transport data capacity per subframe. These lengths are used when decoding Uplink Shared Channel (UL-SCH).

In this example two codewords are used for the PUSCH. The size of four parameters depend on the number of codewords used:

- *`frc.PUSCH.Modulation`* - Each cell contains the modulation scheme for a codeword.

- *frc.PUSCH.TrBlkSizes* - Each row contains the transport block sizes for a codeword.
- *frc.PUSCH.CodedTrBlkSizes* - Each row contains the coded transport block sizes for a codeword.
- *frc.PUSCH.RVSeq* - Each row contains the Redundancy Versions (RVs) for a codeword.

```
% Generate parameter structure for single codeword A3-2 FRC
frc = lteRMCUL(frc);

% Then update Physical Uplink Shared Channel (PUSCH) parameters to define
% two identically configured single layer codewords
frc.PUSCH.NLayers = 2;  % the layers are shared across the 2 codewords
frc.PUSCH.Modulation = repmat({frc.PUSCH.Modulation}, 1, 2);
frc.PUSCH.TrBlkSizes = repmat(frc.PUSCH.TrBlkSizes, 2, 1);
frc.PUSCH.RVSeq = repmat(frc.PUSCH.RVSeq, 2, 1);

% Record transport block sizes for each subframe in a frame
trBlkSizes = frc.PUSCH.TrBlkSizes;

% The number of codewords is the number of transport block sizes
ncw = size(trBlkSizes,1);

% Record RV sequence
rvSequence = frc.PUSCH.RVSeq;
```

Set Propagation Channel Model Sampling Rate

The sampling rate for the channel model is set using the value returned from `lteSCFDMAInfo`.

```
dims = lteSCFDMAInfo(frc);
chcfg.SamplingRate = dims.SamplingRate;
```

System Processing

The conformance test may be carried out over a number of SNR points. To determine the throughput at each SNR point, the PUSCH data is analyzed on a subframe by subframe basis using the following steps:

- *Update Current HARQ Process.* The HARQ process either carries new transport data or a retransmission of previously sent transport data depending upon the Acknowledgment (ACK) or Negative Acknowledgment (NACK) based on CRC results. All this is handled by the HARQ scheduler, hHARQScheduling.m. The PUSCH data is updated based on the HARQ state.
- *Set PMI.* A PMI is taken sequentially from a set of PMIs, `txPMIs`, each subframe and used by the eNodeB to select a precoding matrix. When a PMI is used by the eNodeB for a transmission it is replaced with a PMI selected by the UE. This PMI is then used to select a precoding matrix after `pmiDelay` subframes. Initially a set of `pmiDelay` random PMIs is used.
- *Create Transmit Waveform.* The data generated by the HARQ process is passed to `lteRMCULTool`, which produces an OFDM modulated waveform, containing the physical channels and signals.
- *Noisy Channel Modeling.* The waveform is passed through a fading channel and Additive White Gaussian Noise (AWGN) noise added. The noise power is normalized to take account of the sampling rate.

- *Synchronization and SC-FDMA Demodulation.* The received symbols are offset to account for a combination of implementation delay and channel delay spread. The symbols are then SC-FDMA demodulated.
- *Channel Estimation.* The channel response and noise are estimated using either a perfect or imperfect channel estimator. These estimates are used to aid the soft decoding of the PUSCH. If imperfect channel estimation is used, the SRS is utilized to enhance estimation. This information is passed in the variable `refGrid` which contains known transmitted SRS symbols in their correct locations. All other locations are represented by a `NaN`.
- *Equalization, Combining and PUSCH Decoding.* The equalization is performed using `lteEqualizeMMSE` and then `ltePUSCHDecode` completes the reception processing under the assumption that the input is already equalized.
- *UL-SCH Decoding.* The vector of decoded soft bits is passed to `lteULSCHDecode`; this decodes the codewords and returns the block CRC error used to determine the throughput of the system. The contents of the new soft buffer, `harqProc(harqID).decState`, is available at the output of this function to be used for the next subframe.
- *Update PMI.* A PMI is selected and fed back to the eNodeB for use in future subframes. For perfect channel estimation the PMI is updated every subframe. For imperfect channel estimation a PMI update only occurs when an SRS transmission occurs and therefore a full-rank channel estimate is available, from which PMI can be estimated and selected.

```
% Store results for each SNR point and each subframe containing data for
% the whole simulation
nDataTBS = sum(trBlkSizes(:)~=0)*NFrames;
crc = zeros(numel(SNRIn), nDataTBS);  % Total block CRC error vector
tput = zeros(numel(SNRIn), nDataTBS); % Total throughput vector
ResultIndex = 1; % SNR point result index

for SNRdB = SNRIn

    % Configure random number generators
    rng('default');

    % Noise configuration
    fprintf('\nSimulating at %gdB SNR for a total %d Frame(s)', ...
            SNRdB, NFrames);
    SNR = 10^(SNRdB/20);
    N = 1/(SNR*sqrt(double(dims.Nfft)))/sqrt(2.0*frc.NTxAnts);

    % Initialize state of all HARQ processes
    harqProcesses = hNewHARQProcess(frc);
    % Initialize HARQ process IDs to 1 as the first non-zero transport
    % block will always be transmitted using the first HARQ process. This
    % will be updated with the full sequence output by lteRMCULTool after
    % the first call to the function
    harqProcessSequence = 1;

    % Use random PMIs for the first 'pmiDelay' subframes until feedback is
    % available from the UE if multiple transmit antennas are used
    if (frc.NTxAnts>1)
        pmidims = lteULPMIInfo(frc, frc.PUSCH);
        txPMIs = randi([0 pmidims.MaxPMI], pmidims.NSubbands, pmiDelay);
    end
```

```
% Initialize result store for SNR point tested
crcSNR = zeros(nDataTBS/ncw, ncw);  % Intermediate block CRC
tputSNR = zeros(nDataTBS/ncw, ncw); % Intermediate throughput
dataSubframeIndex = 1;

% Loop for all subframes at this SNR
offsetused = 0;
for subframeNo = 0:(NFrames*10-1)

    % Update subframe number
    frc.NSubframe = mod(subframeNo, 10);

    % If this is an uplink subframe
    duplexDims = lteDuplexingInfo(frc);
    if(strcmp(duplexDims.SubframeType,'Uplink')==1)

        % Get HARQ process ID for the subframe from HARQ process sequence
        harqID = harqProcessSequence(mod(subframeNo, length(harqProcessSequence))+1);

        % If there is a transport block scheduled in the current subframe
        % (indicated by non-zero 'harqID'), perform transmission and
        % reception. Otherwise continue to the next subframe
        if harqID == 0
            continue;
        end

        % Update current HARQ process
        harqProcesses(harqID) = hHARQScheduling( ...
            harqProcesses(harqID), subframeNo, rvSequence);

        % Update the PUSCH transmission config with HARQ process state
        frc.PUSCH = harqProcesses(harqID).txConfig;
        data = harqProcesses(harqID).data;

        % Set the PMI to the appropriate value in the delay queue (if
        % multiple antennas are being used)
        if (frc.NTxAnts>1)
            pmiIdx = mod(subframeNo, pmiDelay);
            frc.PUSCH.PMI = txPMIs(:, pmiIdx+1);
        end

        % Create transmit waveform and get the HARQ scheduling ID
        % sequence from 'frcOut' structure output which also contains
        % the waveform configuration and OFDM modulation parameters
        [txWaveform,~,frcOut] = lteRMCULTool(frc, data);

        % Add 25 sample padding. This is to cover the range of delays
        % expected from channel modeling (a combination of
        % implementation delay and channel delay spread)
        txWaveform =  [txWaveform; zeros(25, frc.NTxAnts)]; %#ok<AGROW>

        % Get the HARQ ID sequence from 'frcOut' for HARQ processing
        harqProcessSequence = frcOut.PUSCH.HARQProcessSequence;

        % Pass data through the fading channel model. The
        % initialization time for channel modeling is set each subframe
        % to simulate a continuously varying channel.
        chcfg.InitTime = subframeNo/1000;
```

```
rxWaveform = lteFadingChannel(chcfg, txWaveform);

% Add noise at the receiver
noise = N*complex(randn(size(rxWaveform)), ...
    randn(size(rxWaveform)));
rxWaveform = rxWaveform + noise;

% Calculate synchronization offset
offset = lteULFrameOffset(frc, frc.PUSCH, rxWaveform);
if (offset < 25)
    offsetused = offset;
end

% SC-FDMA demodulation
rxSubframe = lteSCFDMADemodulate(frc, rxWaveform...
            (1+offsetused:end, :));

% Mark this subframe as not being involved in PMI update
updatePMI = false;

% Channel and noise estimation
if (perfectChanEstimator)
    % Create reference grid - not required for perfect channel
    % estimator, but needed as an input to lteULPMISelect to
    % match call with imperfect channel estimator.
    refGrid = NaN(lteULResourceGridSize(frc)); %#ok

    % Perfect channel estimation, perfect knowledge of all REs
    % and perfect knowledge of channel noise
    estChannelGrid = lteULPerfectChannelEstimate( ...
        frc, chcfg, offsetused);
    n = lteSCFDMADemodulate(frc, noise(1+offsetused:end, :));
    noiseEst = var(reshape(n, numel(n), 1));
    updatePMI = true;
else
    % Use imperfect channel estimator with support from the SRS
    % if available
    if (strcmpi(reference, 'Antennas')==1 || ...
            (frc.NTxAnts == frc.PUSCH.NLayers))
        refGrid = NaN(lteULResourceGridSize(frc));
        if (isfield(frc, 'SRS'))
            [srsIndices, srsIndicesDims] = lteSRSIndices( ...
                    frc, frc.SRS);
            if (~isempty(srsIndices))
                srsSymbols = lteSRS(frc, frc.SRS);
                refGrid(srsIndices) = srsSymbols;
                updatePMI = true;
            end
        end
    end
    cec.Reference = reference;
    [estChannelGrid, noiseEst] = lteULChannelEstimate( ...
        frc, frc.PUSCH, cec, rxSubframe, refGrid);
end

% Shorten PUSCH capacity shortening as appropriate
if (isfield(frc, 'SRS'))
    srsDims = lteSRSInfo(frc, frc.SRS);
```

```
            frc.Shortened = srsDims.IsSRSSubframe;
        else
            frc.Shortened = 0;
        end

        % Set up variable indicating the current transport block sizes
        TBSs = trBlkSizes(:, mod(subframeNo, 10)+1).';

        % Extract REs corresponding to the PUSCH from the given
        % subframe across all receive antennas and channel estimates.
        puschIndices = ltePUSCHIndices(frc, frc.PUSCH);
        [puschRx, puschEstCh] = lteExtractResources( ...
            puschIndices, rxSubframe, estChannelGrid);

        % Equalization and combining, layer demapping, transform
        % deprecoding, demodulation and descrambling of the received
        % data
        ulschDims = lteULSCHInfo(frc, frc.PUSCH, TBSs, 'chsconcat');
        rxSymbols = lteEqualizeMMSE(puschRx, puschEstCh, noiseEst);
        rxEncodedBits = ltePUSCHDecode(frc, ulschDims, rxSymbols);

        % UL-SCH transport decoding
        [rxdata, harqProcesses(harqID).blkerr, ...
            harqProcesses(harqID).decState] = lteULSCHDecode( ...
            frc, ulschDims, TBSs, rxEncodedBits, ...
            harqProcesses(harqID).decState);

        % Store block CRC and throughput results for subframes
        % containing transport data
        crcSNR(dataSubframeIndex, :) = harqProcesses(harqID).blkerr;
        tputSNR(dataSubframeIndex, :) = TBSs.* ...
            (1-harqProcesses(harqID).blkerr);
        dataSubframeIndex = dataSubframeIndex + 1;

        % Provide PMI feedback (if multiple antennas are being used)
        if (frc.NTxAnts>1)
            if (~any(TBSs) || ~updatePMI)
                % Use previous PMI value if in a subframe not used for
                % uplink in TDD, or if PMI update is not configured for
                % this subframe.
                PMI = txPMIs(:, mod(pmiIdx-1, pmiDelay)+1);
            else
                % Update PMI estimate. If not using the perfect channel
                % estimator, redo channel estimation using only the SRS
                % for PMI selection purposes.
                if (~perfectChanEstimator)
                    cec.Reference = 'None';
                    [estChannelGrid, noiseEst] = ...
                        lteULChannelEstimate(frc, frc.PUSCH, cec, ...
                        rxSubframe, refGrid);
                end
                PMI = lteULPMISelect(frc, frc.PUSCH, ...
                    estChannelGrid, noiseEst, refGrid, cec);
            end
            txPMIs(:, pmiIdx+1) = PMI;
        end
    end
end
```

```
    % Record the block CRC error and bit throughput for the total number of
    % frames simulated at an SNR point
    crc(ResultIndex, :) = crcSNR(:);
    tput(ResultIndex, :) = tputSNR(:);
    ResultIndex = ResultIndex + 1;
    disp(' ');

end
```

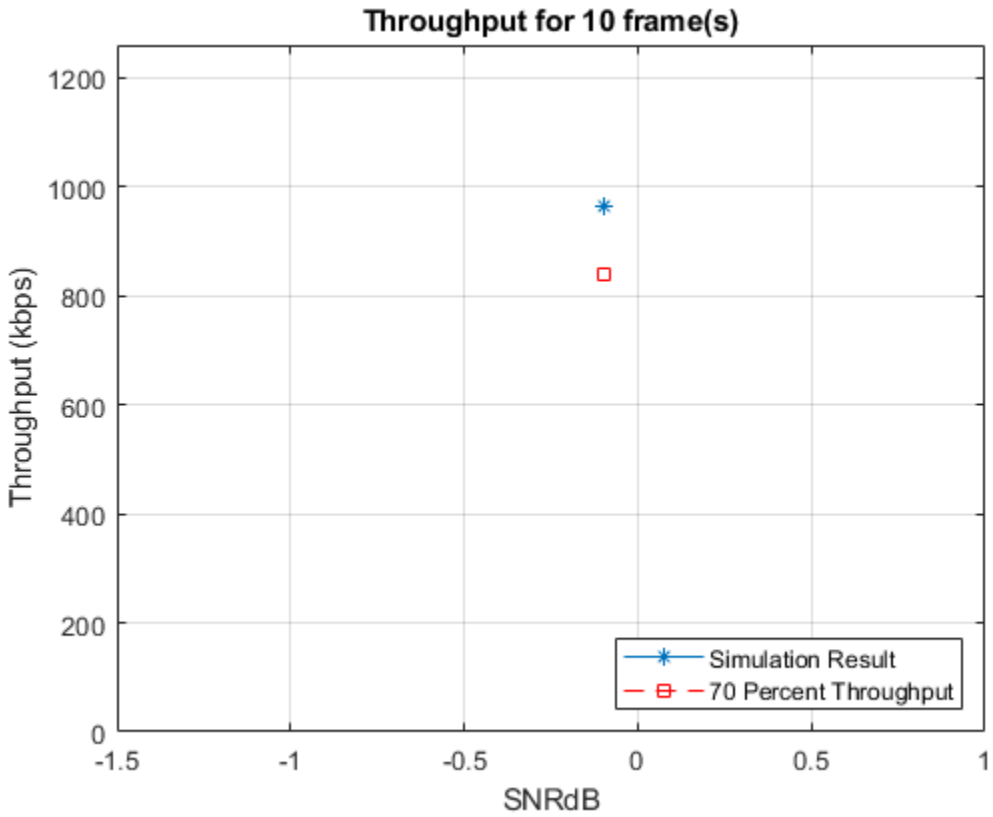
```
Simulating at -0.1dB SNR for a total 10 Frame(s)
```

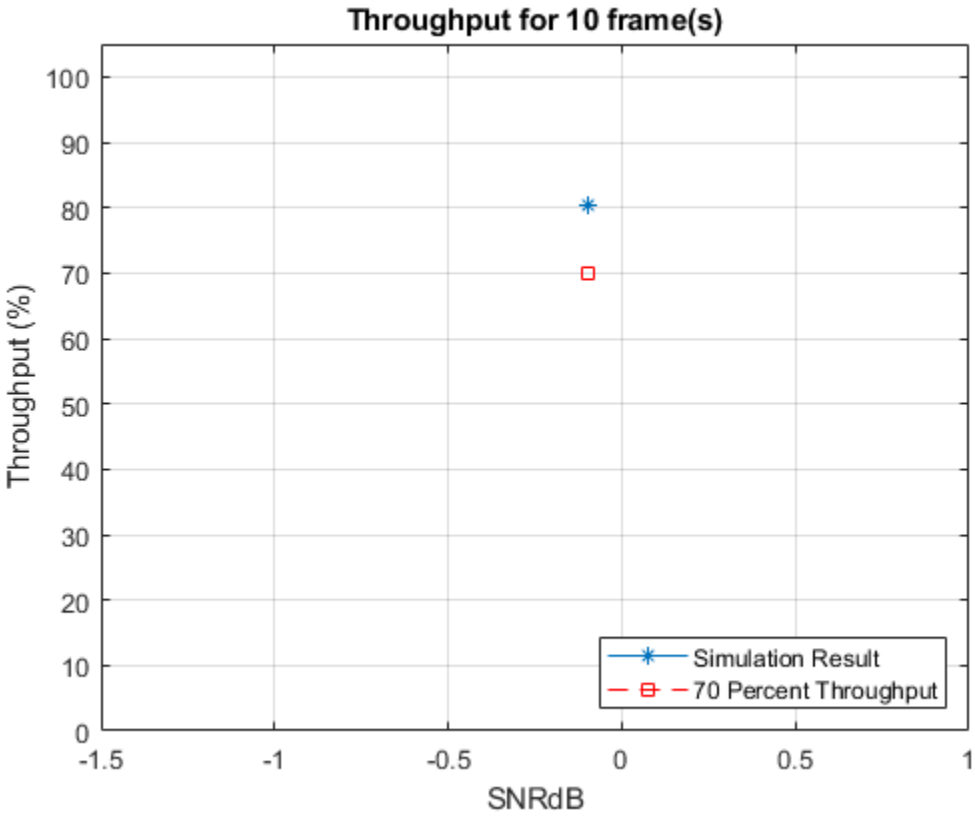
Results

The throughput results for the simulation are contained in `crc` and `tput`. `crc` is a matrix where each row contains the results of decoding the CRC for a particular SNR. Each column contains the CRC result for a transport block containing PUSCH data at an SNR. `tput` is a matrix where each row contains the bit throughput per subframe for a particular SNR. Each column contains the throughput result for a transport block containing PUSCH data at an SNR.

The throughput results are plotted as a percentage of total capacity and actual bit throughput for the range of SNR values input using hMultiCodewordPUSCHResults.m.

```
hMultiCodewordPUSCHResults(SNRIn, NFrames, trBlkSizes, crc, tput);
```





Appendix

This example uses these helper functions.

- hNewHARQProcess.m
- hHARQScheduling.m
- hMultiCodewordPUSCHResults.m

Selected Bibliography

1 3GPP TS 36.104 "Base Station (BS) radio transmission and reception"

LTE Sidelink Resource Pools and PSCCH Period

This example shows how sidelink direct communication resource pools and PSCCH periods are defined and parameterized. It shows the relationship between the semi-static RRC pool parameters and the PSCCH period structure. It also shows how the dynamic scheduling parameters (DCI and SCI) for transmission mode 1 and mode 2 affect the final transmission resource selection.

Introduction to ProSe Direct Communications and the Sidelink

Release 12 of the 3GPP LTE standard introduced a new device-to-device (D2D) interface aimed primarily at allowing LTE to support public safety communication systems. In terms of the overall LTE RAN, this interface enables two kinds of proximity based services (known in 3GPP as ProSe):

- **ProSe Direct Communication**, where direct UE to multiple UE communication (group communication) is possible without data transmission on the uplink or downlink. This is allowed for public safety applications only and supports one or more UEs being out of coverage (network and/or frequency).
- **ProSe Direct Discovery**, where business services can be enabled for UEs that are close to each other. This feature can be used for commercial applications (for example, service advertising) in the case when both UEs are in network coverage.

In terms of the lower stack, the LTE D2D interface is called the **sidelink** and, in terms of the system architecture, it is known as **PC5** (in contrast to the UE/eNodeB interface, Uu). It comprises a collection of new physical signals, physical channels, transport channels and messages. Since the sidelink is transmitted by a UE, it is closely related to the uplink, but it also incorporates some aspects of downlink synchronization and control signaling. One significant result of this design is that a UE will now have to receive and generate uplink-style (sidelink).

This example focuses solely on direct communications, where the relevant sidelink physical layer channels and signals are,

- Physical Sidelink Shared Channel (PSSCH)
- Physical Sidelink Control Channel (PSCCH) (carrying SCI)
- Physical Sidelink Broadcast Channel (PSBCH)
- Sidelink Shared Channel (SL-SCH)
- Sidelink Broadcast Channel (SL-BCH) (carrying MIB-SL)
- Sidelink Synchronization Signals

In addition to the above channels, new physical layer procedures have been introduced as described in TS 36.213 Section 14. A key concept in these processes is that of a **resource pool** which defines the subset of available subframes and resource blocks for either sidelink transmission or reception. Sidelink communication is a half-duplex scheme and a UE can be configured with multiple transmit resource pools and multiple receive resource pools. The resource pools are configured semi-statically by layer 3 messaging. When data is to be sent using a resource pool, the actual transmission resources are selected dynamically from within the pool using one of two different modes:

- **Transmission mode 1** - the serving eNodeB specifies the resources via a DCI format 5 message sent to the transmitting UE. This mode requires the UE to be fully connected to the network (RRC_CONNECTED state).

- **Transmission mode 2** - the transmitting UE self-selects the resources according to rules aimed at minimizing the collision risk. This mode can be used when the UE is connected, idle (RRC_IDLE) or out-of-network coverage.

Introduction to Sidelink Resource Pools and PSCCH Period

A sidelink direct communication resource pool is configured on a semi-static basis using the layer 3 `SL-CommResourcePool` RRC message (TS 36.331 Section 6.3.8). The layer 1 physical resources (subframes and resource blocks) associated with the pool are partitioned into a sequence of repeating 'hyperframes' known as **PSCCH periods**. This is the standardized term used in TS 36.213 but it is also sometimes referred to as the SA (scheduling assignment) period or SC (sidelink control) period. Within a PSCCH period there are separate *subframe pools* and *resource block pools* for control (PSCCH) and data (PSSCH). The PSCCH subframes always precede those for PSSCH transmission. This is analogous to the symbol layout of the PDCCH and PDSCH OFDM symbols within a single downlink subframe, where the control region precedes the data part. The PSCCH carries *sidelink control information* (SCI) messages, which describe the dynamic transmission properties of the PSSCH that follow it. The receiving UE searches all configured PSCCH resource pools for SCI transmissions of interest to it. A UE can be a member of more than one sidelink communications group.

These subframe and resource block pools are defined by different parameters in the `SL-CommResourcePool-r12` message. The ASN.1 definition of the message type (see TS 36.331 Section 6.1 for general terms) is given by,

```
SL-CommResourcePool-r12 ::= SEQUENCE {

    sc-CP-Len-r12                       SL-CP-Len-r12,

    sc-Period-r12                       SL-PeriodComm-r12,

    sc-TF-ResourceConfig-r12            SL-TF-ResourceConfig-r12,

    data-CP-Len-r12                     SL-CP-Len-r12,

    dataHoppingConfig-r12               SL-HoppingConfigComm-r12,

    ue-SelectedResourceConfig-r12       SEQUENCE {

        data-TF-ResourceConfig-r12          SL-TF-ResourceConfig-r12,

        trpt-Subset-r12                     SL-TRPT-Subset-r12          OPTIONAL    -- Need OP

    }                                                                   OPTIONAL,   -- Need OR

    rxParametersNCell-r12               SEQUENCE {

        tdd-Config-r12                      TDD-Config                  OPTIONAL,   -- Need OP

        syncConfigIndex-r12                 INTEGER (0..15)

    }                                                                   OPTIONAL,   -- Need OR

    txParameters-r12                    SEQUENCE {

        sc-TxParameters-r12                 SL-TxParameters-r12,

        dataTxParameters-r12                SL-TxParameters-r12

    }                                                                   OPTIONAL,   -- Cond Tx
```

```
    ...

}
```

This example uses a MATLAB structure to contain all the simulation parameters including those representing a subset of the `SL-CommResourcePool-r12` message.

```
% This example bundles all parameters into a structure based on
% SL-CommResourcePool-r12. Compare this parameter structure with the RRC
% message definition from TS 36.331 Section 6.3.8
commpoolparameters = PSCCHPeriod.defaultConfig(1,'5MHz')
```

```
commpoolparameters =

  struct with fields:

                         NSLRB: 25
                    DuplexMode: 'FDD'
                     TDDConfig: 0
                    UESelected: 'On'
                    SyncEnable: 'On'
                  NPSCCHPeriod: 0
                sc_CP_Len_r12: 'Normal'
                sc_Period_r12: 40
      sc_TF_ResourceConfig_r12: [1x1 struct]
               data_CP_Len_r12: 'Normal'
         dataHoppingConfig_r12: [1x1 struct]
  ue_SelectedResourceConfig_r12: [1x1 struct]
                    syncConfig: [1x1 struct]
```

Note that some of the parameters or information elements (IE) in the message are optional, for example, depending on whether the pool configuration is for transmit or receive. If `ue-SelectedResourceConfig-r12` is included in the message then the UE is in transmission mode 2 (UE selected) otherwise it is in transmission mode 1 (eNodeB scheduled). For more information, see the following 3GPP technical standard documents: * TS 36.331 Section 6.3.8 for the definition of all the sidelink related messages and information elements, * TS 36.331 Section 5.10 for layer 3 sidelink procedures, * TS 36.213 Section 14 for layer 1 sidelink procedures.

Modeling Sidelink Communication Pools and PSCCH Period with LTE Toolbox

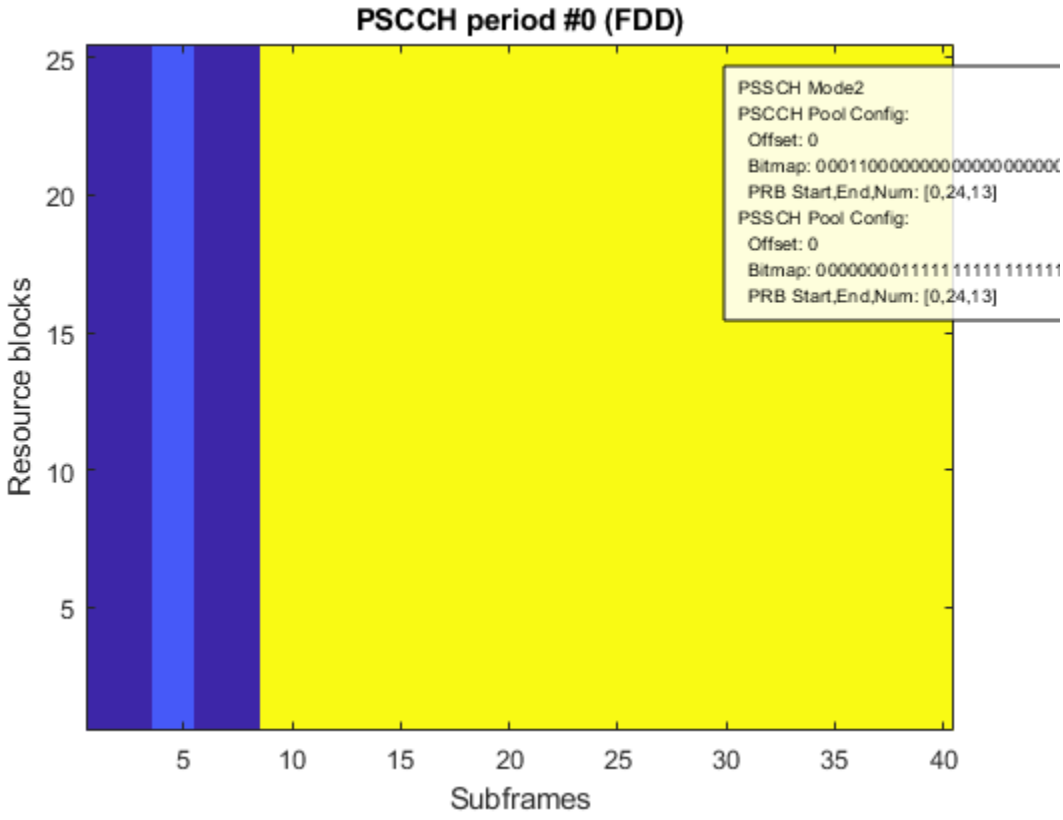
This example uses a MATLAB handle class called `PSCCHPeriod` to represent the structure of a PSCCH period for a single sidelink direct communications resource pool. Objects of type `PSCCHPeriod` can be constructed using a parameter structure which combines general transmission parameters, like transmission bandwidth and duplexing mode, with semi-static layer 3 RRC parameters, primarily from the `SL-CommResourcePool` message (TS 36.331 Section 6.3.8). An object can then be used to,

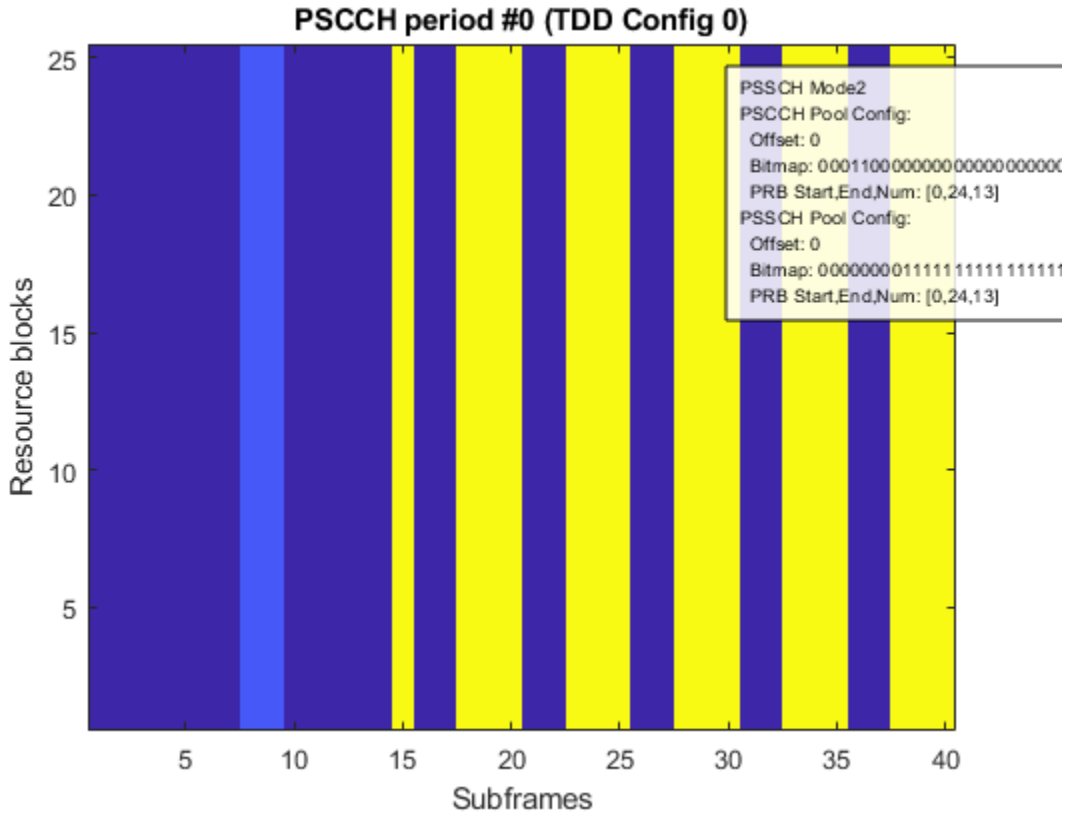
- Get properties which provide key information about the procedural entities in the PSCCH period, such as the subframe pools and resource block pools
- Display an image representing the resources used in the PSCCH period, both for resource pools and actual transmission resources
- Generate a baseband waveform for the PSCCH period containing PSCCH, PSSCH and synchronization transmissions

The following code shows how a `PSCCHPeriod` object can be created, the relationship between its configuration parameters and the `SL-CommResourcePool` message, and how it can display the locations of the physical resource pools within the PSCCH period.

```
% Construct a default PSCCH period object to illustrate PSCCH/PSSCH
% resource pool layout within the period. This default example is
% configured for 5MHz bandwidth and 40ms length, so the overall period
% contains 40 subframes. The displayed properties of the PSCCHPeriod object
% include the 0-based indices for both PSCCH and PSSCH subframe and
% resource block pools. The subframe pool indices are relative to the
% beginning of the period
period = PSCCHPeriod

% Display an image representing the structure of this particular PSCCH
% period. The lighter blue part represents the PSCCH resource pool for SCI
% control information, and the yellow region is the PSSCH resource pool for
% PSSCH shared data
displayPeriod(period);
snapnow;

% Re-configure for TDD then display the updated properties and pool
% locations
period.Config.DuplexMode = 'TDD';
displayPeriod(period);
snapnow;

% The PSCCHPeriod class includes other default parameter structures that
% can be used to configure an object. These are based on a subset of the
% reference pool configurations of TS 36.101 Section A.7.2
configuration = PSCCHPeriod.defaultConfig(1,'5MHz')
```

```
period =

  PSCCHPeriod with properties:

           NSubframeBegin: 0
             PeriodLength: 40
                   TxMode: 'Mode2'
         PSCCHSubframePool: [3 4]
    PSCCHResourceBlockPool: [25x1 double]
          NumPSCCHResource: 24
         PSSCHSubframePool: [8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 ... ]
    PSSCHResourceBlockPool: [25x1 double]
              AllowedITRP: [70x1 double]
            SyncSubframes: []
                   Config: [1x1 struct]
```


PSCCH period #0 (FDD)
Resource blocks
Subframes
PSSCH Mode2
PSCCH Pool Config:
Offset: 0
PRB Start,End,Num: [0,24,13]
PSSCH Pool Config:
Offset: 0
PRB Start,End,Num: [0,24,13]




```
configuration =

  struct with fields:

                        NSLRB: 25
                   DuplexMode: 'FDD'
                    TDDConfig: 0
                   UESelected: 'On'
                   SyncEnable: 'On'
                 NPSCCHPeriod: 0
                sc_CP_Len_r12: 'Normal'
                sc_Period_r12: 40
      sc_TF_ResourceConfig_r12: [1x1 struct]
               data_CP_Len_r12: 'Normal'
         dataHoppingConfig_r12: [1x1 struct]
  ue_SelectedResourceConfig_r12: [1x1 struct]
                   syncConfig: [1x1 struct]
```

PSCCH Subframe and Resource Block Pools

The **PSCCH resource pool** is parameterized by the `sc-TF-ResourceConfig-r12` information element which is of type `SL-TF-ResourceConfig-r12`. This type is also used to define the PSSCH resource pool in the case of PSSCH transmission mode 2. The parameters in this IE determine both the PSCCH subframe and resource block pools. It contains the following parameters:

```
SL-TF-ResourceConfig-r12 ::= SEQUENCE {
    prb-Num-r12                         INTEGER (1..100),
    prb-Start-r12                       INTEGER (0..99),
    prb-End-r12                         INTEGER (0..99),
    offsetIndicator-r12                 SL-OffsetIndicator-r12,
    subframeBitmap-r12                  SubframeBitmapSL-r12
}
```

In the case of the PSCCH, the `offsetIndicator-r12` parameter defines the offset of the PSCCH period sequence relative to SFN/DFN #0. The first subframe of the i-th PSCCH period is given by *jbegin* = `offsetIndicator-r12` + *i* * `sc-Period-r12`. The `subframeBitmap-r12` parameter is used to select subframes from the start of a period for the PSCCH subframe pool. The three parameters, `prb-Num-r12`, `prb-Start-r12` and `prb-End-r12` are used to select the PRB for the PSCCH resource block pool. Depending on the parameter values this pool can be formed from either one or two contiguous sets of resource blocks. This is described in more detail below.

```
% Display the parameter structure used to configure the PSCCH resource pools
period = PSCCHPeriod;
pscchpoolparams = period.Config.sc_TF_ResourceConfig_r12
```

```
pscchpoolparams =

  struct with fields:

            prb_Num_r12: 13
          prb_Start_r12: 0
            prb_End_r12: 24
    offsetIndicator_r12: 0
     subframeBitmap_r12: '00011000000000000000000000000000000000000'
```

The **PSCCH subframe pool** is defined by the `subframeBitmap-r12` parameter which is part of the `sc-TF-ResourceConfig-r12` information element. For the subframe pool the first N' uplink subframes in the PSCCH period are selected where N' is the length of the bitmap. These subframes are denoted by their indices $(l_0, l_1, \ldots, l_{N'-1})$. The PSCCH subframe pool then comprises the uplink subframes associated with the 1's in the bitmap $(a_0, a_1, \ldots, a_{N'-1})$ and the resulting pool is denoted by $(l_0^{PSCCH}, l_1^{PSCCH}, \ldots, l_{L_{PSCCH}-1}^{PSCCH})$ where L_{PSCCH} is the number of subframes in the pool.

```
% Display the PSCCH subframe pool bitmap parameter, subframeBitmap_r12
pscchsubframebitmap = period.Config.sc_TF_ResourceConfig_r12.subframeBitmap_r12
% Display the subframe pool indices (0-based, relative to the start of the
% PSCCH period) selected by the 1's in the bitmap
pscchsubframepool = period.PSCCHSubframePool

% Change the duplexing mode to TDD and observe the difference in subframe
% pool indices to account for the new uplink subframe positions for the
% current TDD configuration
period.Config.DuplexMode = 'TDD';
tddconfig = period.Config.TDDConfig
pscchsubframepool = period.PSCCHSubframePool
```

```
pscchsubframebitmap =

    '00011000000000000000000000000000000000000'


pscchsubframepool =

     3     4


tddconfig =

     0


pscchsubframepool =

     7     8
```

The **PSCCH resource block pool** is defined by the three parameters `prb-Start-r12`, `prb-End-r12` and `prb-Num-r12`, which are also part of the `sc-TF-ResourceConfig-r12` information element.

The parameters define two bands `prb-Start-r12` $\leq q <$ (`prb-Start-r12` + `prb-Num-r12`) and (`prb-End-r12` - `prb-Num-12`) $< q \leq$ `prb-End-r12`. The resulting indices are denoted by $(m_0^{PSCCH}, m_1^{PSCCH}, \ldots, m_{M_{RB}^{PSCCH\_RP}-1}^{PSCCH})$ where $M_{RB}^{PSCCH\_RP}$ is the number of blocks in the resource pool. The bands can overlap to form a single contiguous block.

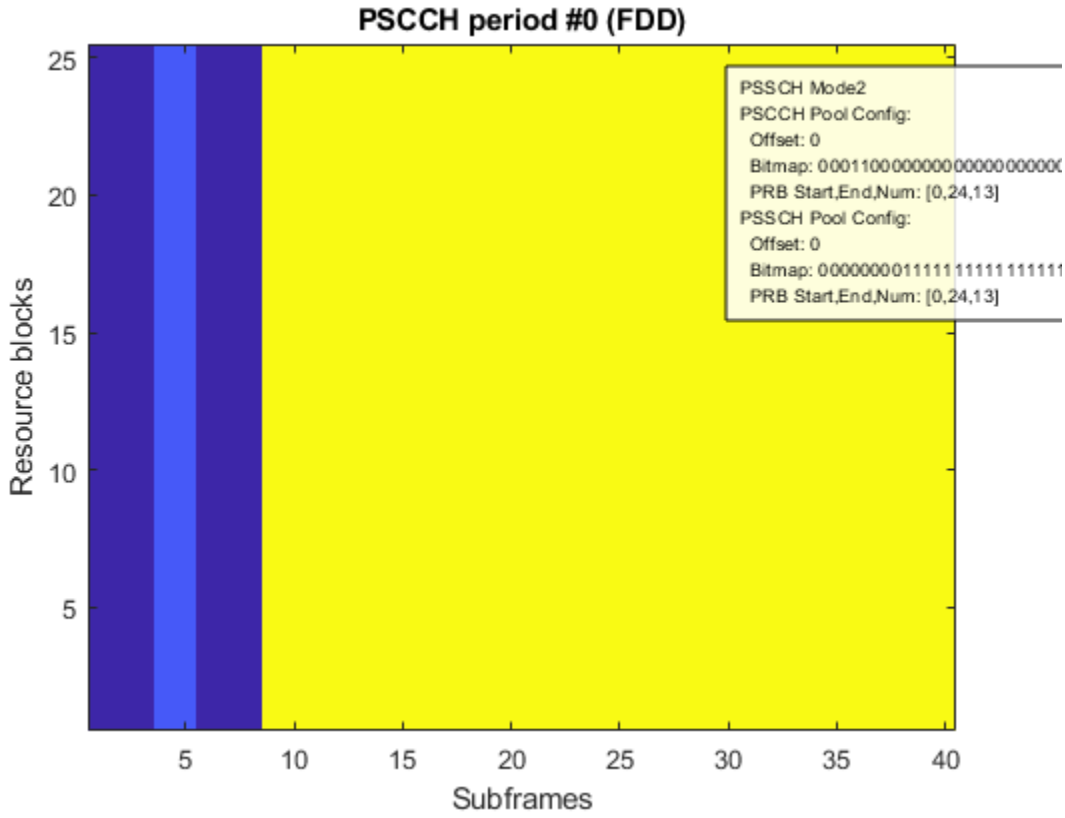
```
% Display the PSCCH resource block pool parameters, prb_Start-r12, prb-End-r12, prb-Num-r12
prbstart = period.Config.sc_TF_ResourceConfig_r12.prb_Start_r12
prbend = period.Config.sc_TF_ResourceConfig_r12.prb_End_r12
prbnum = period.Config.sc_TF_ResourceConfig_r12.prb_Num_r12

% Display the resource block pool indices (0-based) resulting from these
% parameters
pscchprbpool = period.PSCCHResourceBlockPool

% Change the prb-Num-r12 parameter and observe the difference in the
% resource block pool indices to account for the reduction in the size of
% the pool bands
prbnum = 2
period.Config.sc_TF_ResourceConfig_r12.prb_Num_r12 = prbnum;
pscchprbpool = period.PSCCHResourceBlockPool


prbstart =

     0


prbend =

    24
```

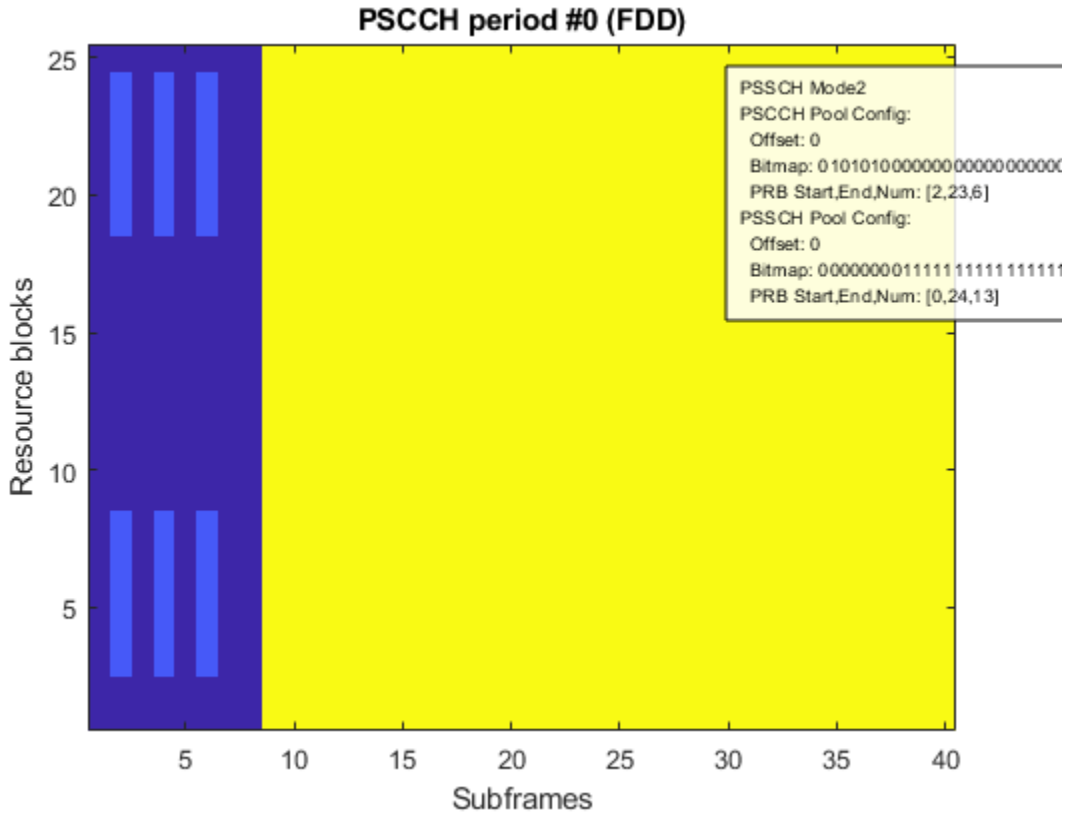
```
prbnum =

    13


pscchprbpool =

     0
     1
     2
     3
     4
     5
     6
     7
     8
     9
    10
    11
    12
    13
    14
    15
    16
    17
    18
    19
    20
    21
    22
    23
    24


prbnum =

     2


pscchprbpool =

     0
     1
    23
    24
```

The following demonstrates some of the effects of these parameters visually.

```
% Display the PSCCH pool resource locations for the default configuration.
% For this parameterization the PSCCH resource block pool spans the entire
% bandwidth and the PSCCH subframe pool is a pair of contiguous subframes
% (the lighter blue area towards the start of the period)
period = PSCCHPeriod;
displayPeriod(period);
snapnow;
```

```
% Modify the subframe bitmap and resource block pool parameters to create
% non-contiguous pools
newconfig.prb_Num_r12 = 6;
newconfig.prb_Start_r12 = 2;
newconfig.prb_End_r12 = 23;
newconfig.offsetIndicator_r12 = 0;
newconfig.subframeBitmap_r12 = '01010100000000000000000000000000000000000';
period.Config.sc_TF_ResourceConfig_r12 = newconfig;
displayPeriod(period);
snapnow;
```





PSSCH Subframe and Resource Block Pools

The parameterization and structure of the PSSCH subframe and resource block pool depends on the transmission mode.

For **transmission mode 1**, the **PSSCH subframe pool** comprises all the remaining uplink subframes that start immediately after the last subframe of the PSCCH subframe pool, $l_{L_{PSCCH}-1}^{PSCCH}$. The **PSSCH resource block pool** comprises the full transmission bandwidth, $(0,\ldots,N_{RB}^{SL})$.

For **transmission mode 2**, the RRC message uses a similar parameterization approach to that of the PSCCH. If the communication pool message contains a `ue-SelectedResourceConfig-r12` element, then the UE is in transmission mode 2 and it will self-select its final transmission resources from the PSSCH resource pools. These pools are defined using another instance of the same `SL-TF-ResourceConfig-r12` parameter set that is used to structure the PSCCH pools.

The `ue-SelectedResourceConfig-r12` information element is given by,

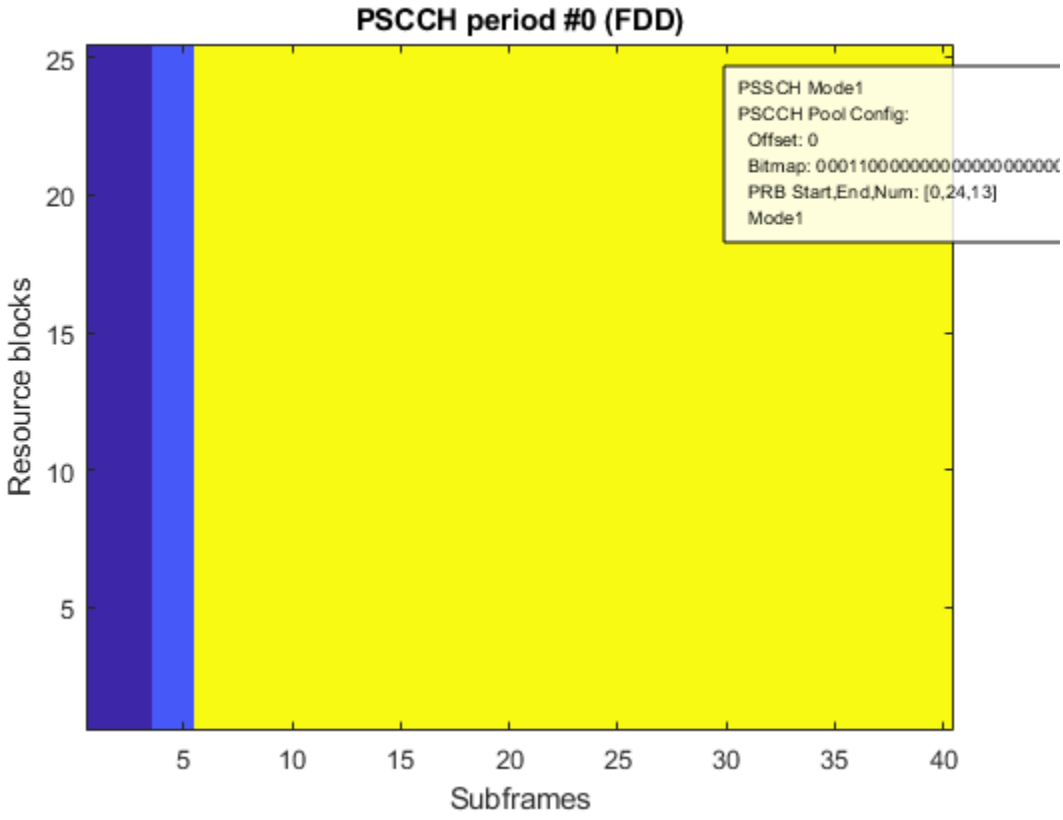
```
ue-SelectedResourceConfig-r12 ::= SEQUENCE {
    data-TF-ResourceConfig-r12              SL-TF-ResourceConfig-r12,
    trpt-Subset-r12                         SL-TRPT-Subset-r12  OPTIONAL  -- Need OP
}
```

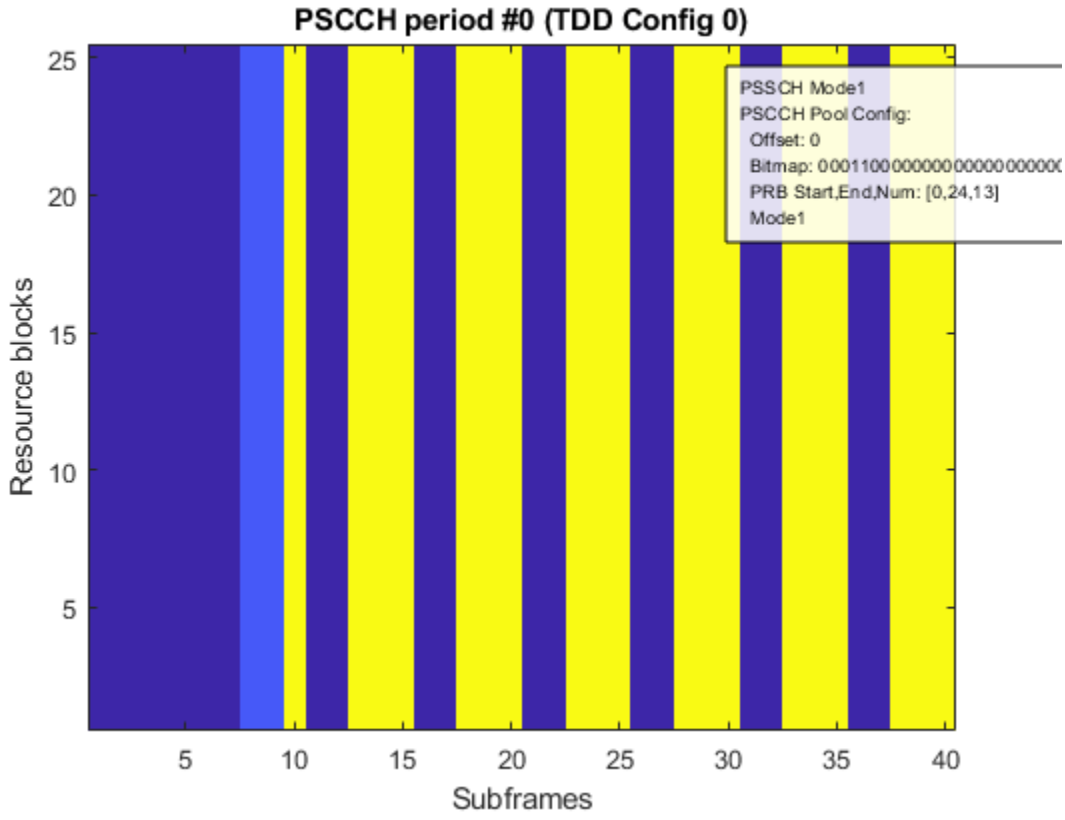
The additional `trpt-Subset-r12` parameter is a small bitmap (3 to 5 bits) which is used to restrict the set of I_{TRP} (time resource pattern index) values selectable by the UE. This affects the total

number of transmission subframes that the UE can select from the PSSCH subframe pool and therefore the maximum number of transport blocks that can be sent in a scheduled period.

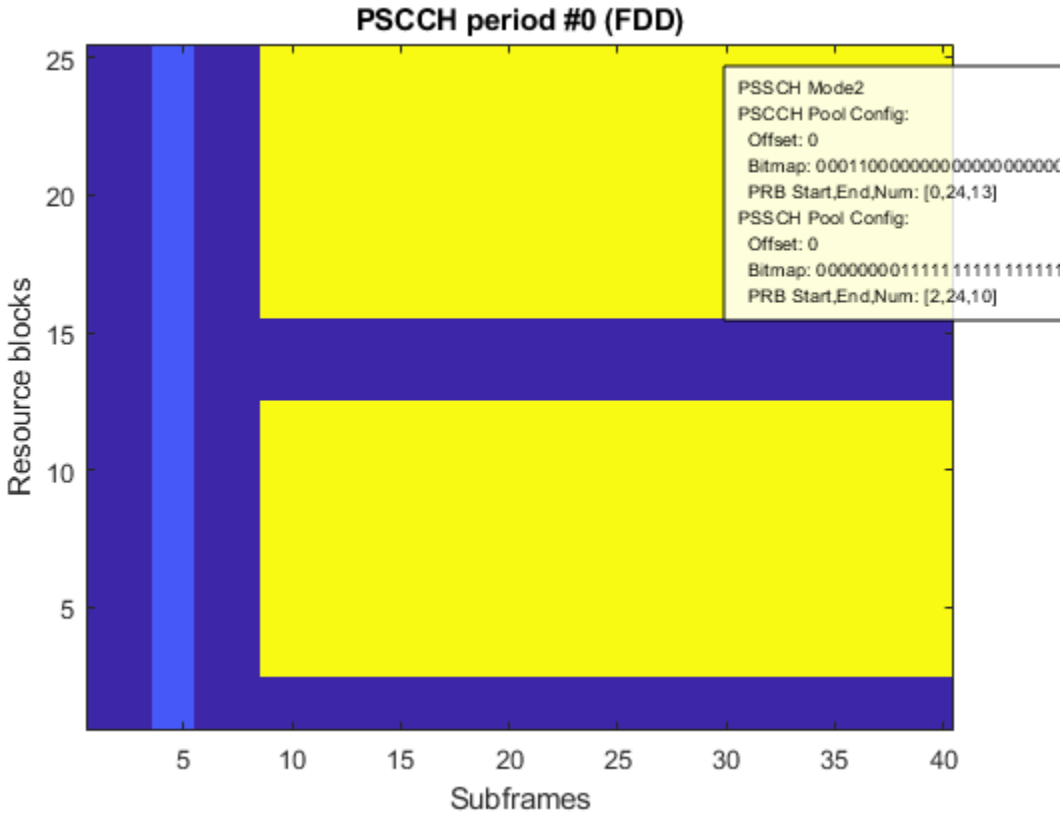
For the **PSSCH subframe pool**, the `subframeBitmap-r12` bitmap selects the pool subframes from the set of all uplink subframes that start at the subframe number given by `offsetIndicator-r12` (relative to the start of the period) and continue to the end of the period. The `subframeBitmap-r12` bitmap is repeated so that it is at least as long as the uplink subframe set and is used to select the final PSSCH subframe pool. The **PSSCH resource block pool** is defined in the same way as with the PSCCH using the three parameters, `prb-Num-r12`, `prb-Start-r12` and `prb-End-r12`.

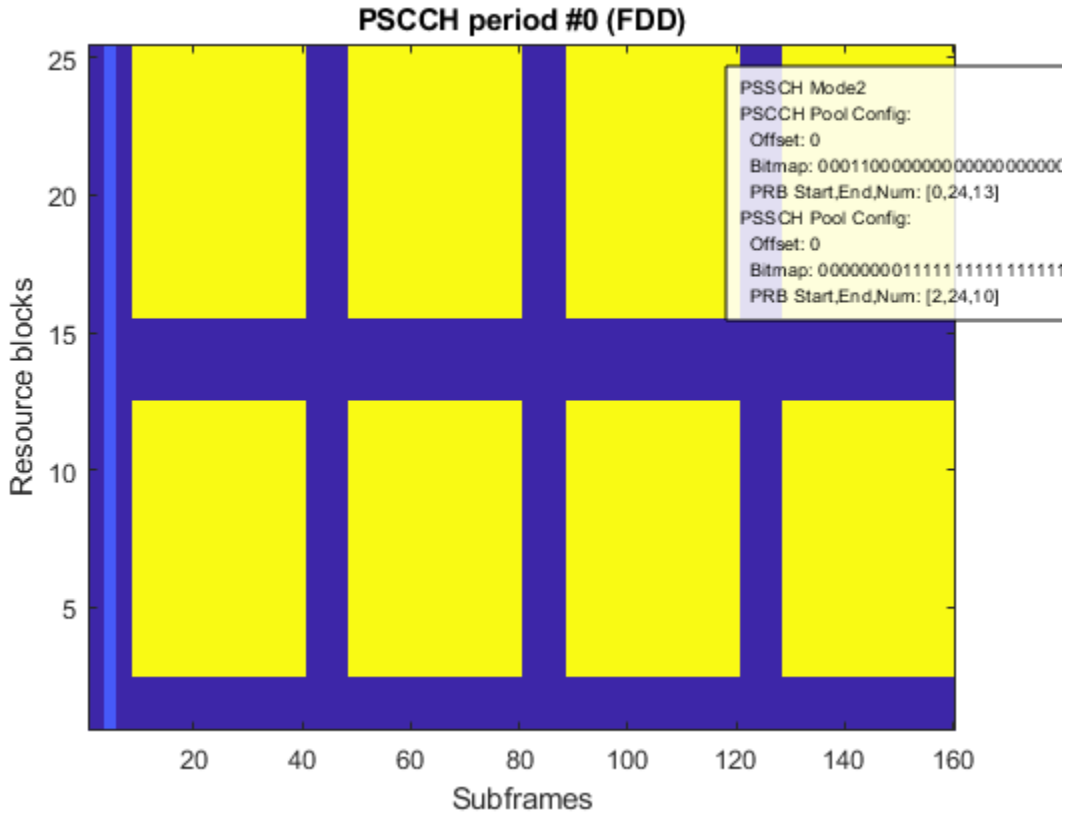
```
% Configure for transmission mode 1 (not UE selected).
% Note that the PSSCH resource pool (yellow) is always full band and
% includes all uplink subframes in the period starting immediately after
% the PSCCH pool (lighter blue)
period = PSCCHPeriod;
period.Config.UESelected = 'off';
displayPeriod(period);
snapnow;

% Change to TDD and note the subframe gaps in the period due to the
% downlink subframes
period.Config.DuplexMode = 'TDD';
displayPeriod(period);
snapnow;

% Change back to transmission mode 2 (UE selected) and FDD.
% Modify the PSSCH resource block pool parameters to create two distinct
% PRB bands
period.Config.UESelected = 'on';
period.Config.DuplexMode = 'FDD';
period.Config.ue_SelectedResourceConfig_r12.data_TF_ResourceConfig_r12.prb_Num_r12 = 10;
period.Config.ue_SelectedResourceConfig_r12.data_TF_ResourceConfig_r12.prb_Start_r12 = 2;
% Although the offset indicator is 0 relative to the start of the period,
% the leading 0's in the subframe bitmap create the gap between the period
% start and that of the PSSCH subframe pool
display(period.Config.ue_SelectedResourceConfig_r12.data_TF_ResourceConfig_r12);
displayPeriod(period);
snapnow;

% Increase the length of the period and note the gaps created in PSSCH
% subframe pool due to the repetition of the pattern of 0's in the
% configured subframe bitmap (40 bits) to cover the increased number of
% uplink subframes in the period.
period.Config.sc_Period_r12 = 160;  % 40,60,70,80,120,140,160,240,280,320 subframes, depending o
displayPeriod(period);
snapnow;
```


PSCCH period #0 (FDD)
PSSCH Mode1
PSCCH Pool Config:
Offset: 0
Bitmap: 0001100000000000000000
PRB Start,End,Num: [0,24,13]
Mode1
Resource blocks
Subframes
25
20
15
10
5
5
10
15
20
25
30
35
40




```
struct with fields:

          prb_Num_r12: 10
        prb_Start_r12: 2
          prb_End_r12: 24
  offsetIndicator_r12: 0
   subframeBitmap_r12: '00000000111111111111111111111111111111111'
```


PSCCH period #0 (FDD)
Resource blocks
Subframes
25
20
15
10
5
5
10
15
20
25
30
35
40
PSSCH Mode2
PSCCH Pool Config:
Offset: 0
Bitmap: 0001100000000000000000
PRB Start,End,Num: [0,24,13]
PSSCH Pool Config:
Offset: 0
Bitmap: 0000000011111111111111
PRB Start,End,Num: [2,24,10]



Sidelink Transmission and Dynamic Resource Scheduling

As described above, when data is to be sent using a resource pool, the actual transmission resources are selected dynamically from within the pool using one of two different modes,

- **Transmission mode 1** - the serving eNodeB directs the resources via a DCI format 5 message sent to the transmitting UE
- **Transmission mode 2** - the transmitting UE self-selects the resources according to rules aimed at minimizing the collision risk

In both cases, the same physical layer parameters are used to manage the actual resource selection. The difference being that, for mode 1, these parameters are provided by the network whereas, for mode 2, they are chosen randomly by the UE (TS 36.321 Section 5.14.1.1 specifies - "*randomly select the time and frequency resources for SL-SCH and SCI of a sidelink grant from the resource pool configured by upper layers. The random function shall be such that each of the allowed selections can be chosen with equal probability*".)

The physical layer parameters are:

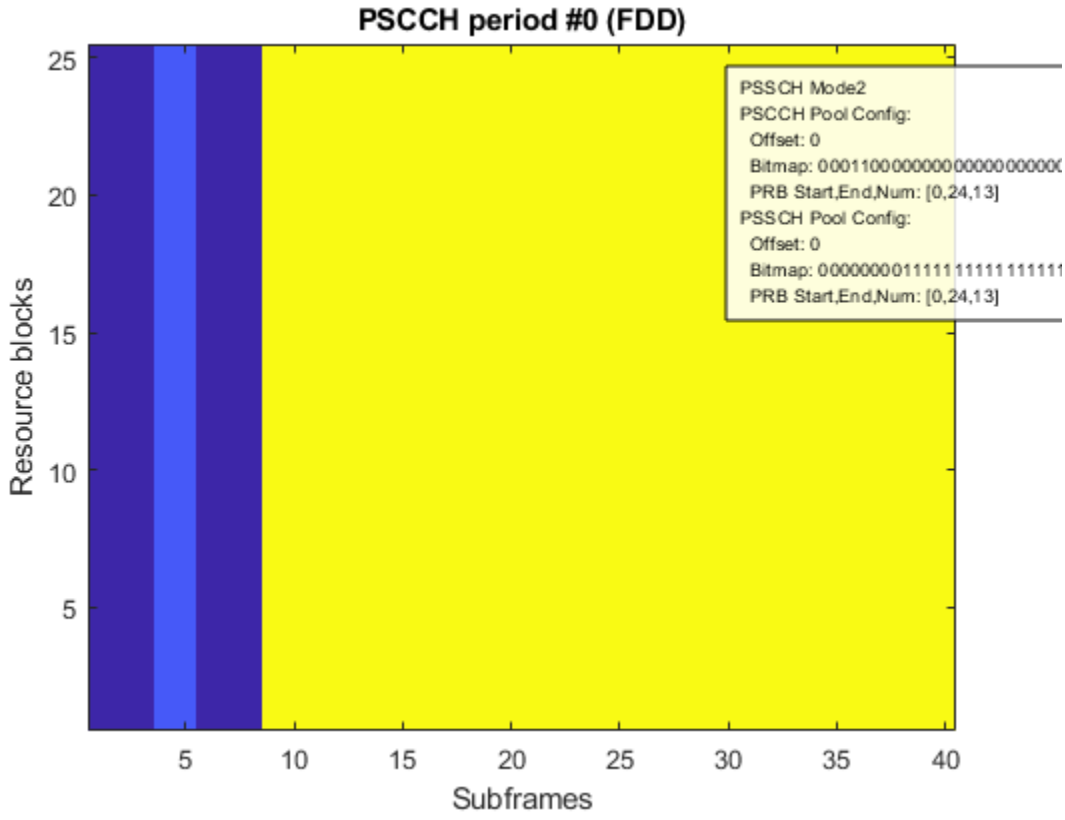
- Resource for PSCCH value (n_{PSCCH}) - PSCCH subframes and resource blocks
- Time Resource Pattern Index (I_{TRP}) - PSSCH subframes
- Resource Allocation parameters (*RIV, hopping bits*) - PSSCH resource blocks

Resource Selection for PSCCH Transmission

The PSCCH control information associated with any PSSCH data transmission is sent twice on two separate PSCCH instances. Each PSCCH uses a different single PRB selected from the PSCCH resource block pool. The pair of subframes are selected from the PSCCH subframe pool. These PSCCH resources are signaled by a single scalar value n_{PSCCH} ("Resource for PSCCH"). The two subframe and PRB index pairs are derived according to TS 36.213 Sections 14.2.1.1 and 14.2.1.2. The range of allowed values is $0 \le n_{PSCCH} < \lfloor M_{RB}^{PSCCH\_RP}/2 \rfloor \cdot L_{PSCCH}$. The number of allowed values is given by the `NumPSCCHResource` property.

```
% Create an example PSCCH period and observe the location of the PSCCH
% subframe and resource block pools. Note that the PSCCH subframe pool
% contains only 2 entries in this case so all of the pool subframes will
% be used
period = PSCCHPeriod
displayPeriod(period);
snapnow;

% Select a valid nPSCCH value (use the last value in the allowed range)
% and return the associated PSCCH subframe and PRB indices
dci.PSCCHResource = period.NumPSCCHResource-1;
[subframes1,prb1,selected1] = period.getPSCCHResources(dci)

% Let the function select an nPSCCH value at random, as required by
% the collision avoidance mechanism used in transmission mode 2
sci.PSCCHResource = [];
[subframes2,prb2,selected2] = period.getPSCCHResources(sci)
```

```
period =

  PSCCHPeriod with properties:

           NSubframeBegin: 0
             PeriodLength: 40
                   TxMode: 'Mode2'
         PSCCHSubframePool: [3 4]
    PSCCHResourceBlockPool: [25x1 double]
          NumPSCCHResource: 24
         PSSCHSubframePool: [8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 ... ]
    PSSCHResourceBlockPool: [25x1 double]
              AllowedITRP: [70x1 double]
             SyncSubframes: []
                   Config: [1x1 struct]
```




```
subframes1 =

     4     3

prb1 =

    11    23

selected1 =

    23

subframes2 =

     4     3

prb2 =

     9    21

selected2 =
```

```
    19
```

Similar to the use of the PDCCH and DCI in the downlink, the pair of PSCCH instances carry an SCI format 0 message which contains information used by the receiving UEs to decode the associated PSSCH sequence. There is no RNTI CRC masking in the SCI encoding. Instead the receiving UEs use the group destination ID contained within the SCI message payload to help filter out the PSSCH communications of interest (additional destination filtering is also done by higher layers).

SCI Format 0 message

Release 12 of the LTE standard specifies a single SCI format. For more information see `lteSCI`. The SCI format 0 is defined in TS 36.212 Section 5.4.3.1.1 by the following information fields:

- `Frequency hopping flag` - 1 bit as defined in TS 36.213 section 14.1.1

- `Resource block assignment and hopping resource allocation` - $\lceil log_2(N_{RB}^{SL}(N_{RB}^{SL}+1)/2)\rceil$ bits

-- For PSSCH hopping:

- `Hopping bits` - N_{SL_hop} MSB bits are used to obtain the value of $\tilde{n}_{PRB}(i)$ as indicated in TS 36.213 section 8.4
- `RIV` - $(\lceil log_2(N_{RB}^{SL}(N_{RB}^{SL}+1)/2)\rceil)$ bits provide the resource allocation in the subframe

-- For non-hopping PSSCH:

- `RIV` - $(\lceil log_2(N_{RB}^{SL}(N_{RB}^{SL}+1)/2)\rceil - N_{SL\_hop})$ bits provide the resource allocation in the subframe as defined in TS 36.213 section 8.1.1

- `Time resource pattern` - 7 bits as defined in TS 36.213 section 14.1.1 (I_{TRP})

- `Modulation and coding scheme` - 5 bits as defined in TS 36.213 section 14.1.1 (I_{MCS})

- `Timing advance indication` - 11 bits as defined in TS 36.213 section 14.2.1

- `Group destination ID` - 8 bits as defined by higher layers (N_{ID}^{SA})

```
% Display the SCI format 0 message field sizes for this example (5 MHz BW)
sci0 = lteSCI(period.Config,struct('SCIFormat','Format0','FreqHopping',1),'fieldsizes')
allocfields = sci0.Allocation

% Change BW to 10 MHz and notice the difference in resource field sizes
period.Config.NSLRB = 50;
sci0 = lteSCI(period.Config,struct('SCIFormat','Format0','FreqHopping',1),'fieldsizes')
allocfields = sci0.Allocation
```

```
sci0 =

  struct with fields:
```

```
              SCIFormat: 'Format0'
             FreqHopping: 1
              Allocation: [1x1 struct]
    TimeResourcePattern: 7
              ModCoding: 5
             TimeAdvance: 11
                   NSAID: 8
                 Padding: 0


allocfields =

  struct with fields:

    HoppingBits: 1
            RIV: 8


sci0 =

  struct with fields:

              SCIFormat: 'Format0'
             FreqHopping: 1
              Allocation: [1x1 struct]
    TimeResourcePattern: 7
              ModCoding: 5
             TimeAdvance: 11
                   NSAID: 8
                 Padding: 0


allocfields =

  struct with fields:

    HoppingBits: 2
            RIV: 9
```

Resource Selection for PSSCH Transmission

In the case of PSSCH, different parameters are used to specify the time and frequency resources. This differs from PSCCH, which signals the subframes and PRB to be used by a single value.

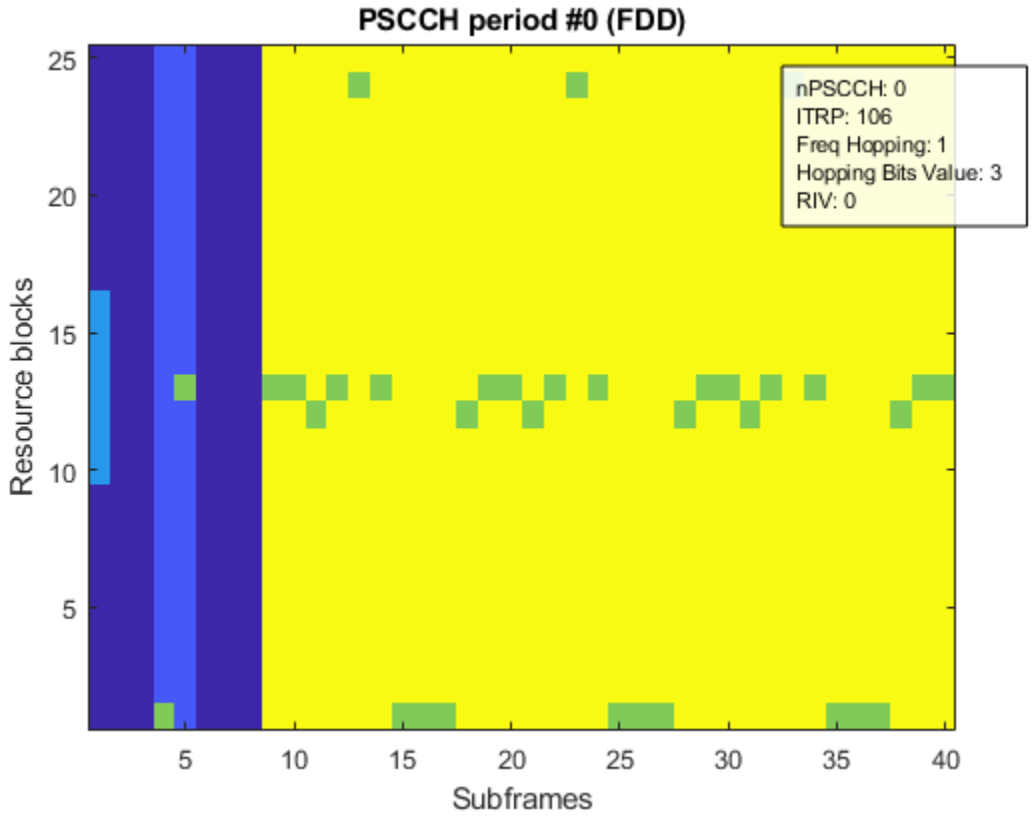
The subframes associated with the PSSCH transmission are indicated by the time resource pattern index, I_{TRP}. This index is used to look up a bitmap from a set of tables, with the choice of table being dependent on the duplexing configuration. The selected bitmap is denoted by $(b'_0, b'_1, \ldots, b'_{N_{TRP}-1})$ where N_{TRP} is 6,7 or 8 depending on the table. This bitmap is repeated to form an extended bitmap $(b_0, b_1, \ldots, b_{L_{PSSCH}-1})$ which covers the entire PSSCH subframe pool. The subframes used for PSSCH transmission are selected by the *1* values in this extended bitmap to give the final subframe set denoted by $(n_0^{PSSCH}, n_1^{PSSCH}, \ldots, n_{N_{PSSCH}-1}^{PSSCH})$ where N_{PSSCH} is the number of subframes that can be used for PSSCH transmission in the PSCCH period and which will also be a multiple of 4. This aligns with the fact that each transport block transmitted within the period will be sent four times using the

fixed HARQ RV sequence = 0,2,3,1. During a PSCCH period, as many of the scheduled quadruples are used as transport blocks that are available to send at that time.

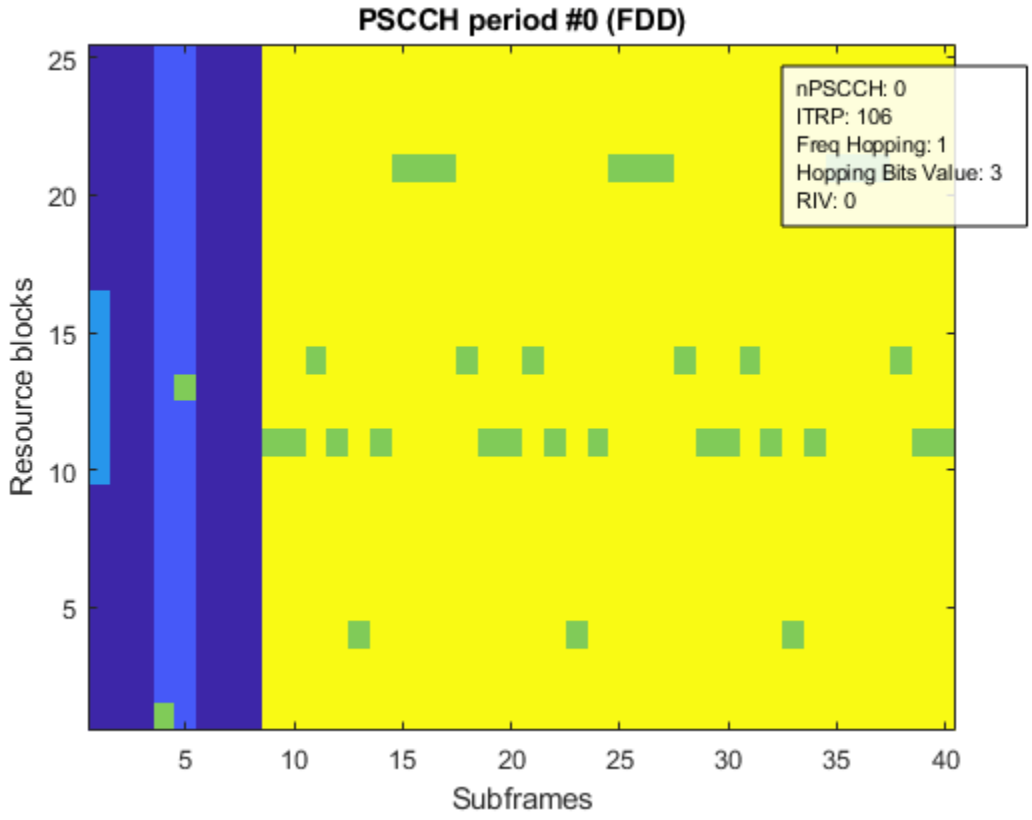
If frequency hopping is enabled, the resource blocks used in each of the transmission subframes depends on the *RIV* field and *hopping bits*. This is in addition to the semi-static `dataHoppingConfig-r12` parameters and dependent resource block pool. The PRB will then depend on the position of the active subframe *within the subframe pool*.

```
% Display the transmission resources used within the PSCCH/PSSCH resource
% pools. Turn PSSS/SSSS/PSBCH on for this example
period = PSCCHPeriod;
period.Config.SyncEnable = 'on';
period.Config.syncConfig.syncOffsetIndicator_r12 = 0;

% Define all the allocation control parameters, including an explicit PSCCH
% resource. Although in mode 2, this is effectively full DCI format 5
% parameterization, indicating SCI format 0 and PSCCH resource control
dci.PSCCHResource = 0;             % Select a specific PSCCH resource value
dci.TimeResourcePattern = 106;     % Select an unrestricted bitmap (all 1's)
dci.FreqHopping = 1;               % Configure frequency hopping with hopping type 2 (predefined
dci.Allocation.HoppingBits = 3;    % Setting the value=3 will enable hopping type 2 for all BW (1

% Get the set of RIV that is associated with contiguous allocations within
% the current PSSCH resource pool. Set the first RIV which will be a single
% PRB allocation
[riv,range] = getAllowedRIV(period,dci);
dci.Allocation.RIV = riv(1);

% Display subframe indices and prb indices associated with the dynamic allocation
[subframes,prb,poolindices] = period.getPSSCHResources(dci)
% Display the transmission resources in addition to the pool positions
displayPeriod(period,dci);
snapnow;

% Display the RRC parameters that affect the PSSCH resource allocation and
% modify the RB offset to move the PRB allocation away from the PRB pool edges
period.Config.dataHoppingConfig_r12
period.Config.dataHoppingConfig_r12.numSubbands_r12 = 2;
period.Config.dataHoppingConfig_r12.rb_Offset_r12 = 4;
displayPeriod(period,dci);
snapnow;

% Display the UE selected (mode 2) PSCCH resource pool configuration then
% modify the PRB resource block pool parameters to created two distinct
% resource groups in the pool away from the band edges
datareSconfig = period.Config.ue_SelectedResourceConfig_r12.data_TF_ResourceConfig_r12
datareSconfig.prb_Start_r12 = 0;
datareSconfig.prb_End_r12 = 22;
datareSconfig.prb_Num_r12 = 8;
period.Config.ue_SelectedResourceConfig_r12.data_TF_ResourceConfig_r12 = datareSconfig;
% Display the updated resource pool and its effect on the transmission resources
displayPeriod(period,dci);
snapnow;

% Finally generate and plot the associated baseband waveform
figure
waveform = generateWaveform(period,dci);
```

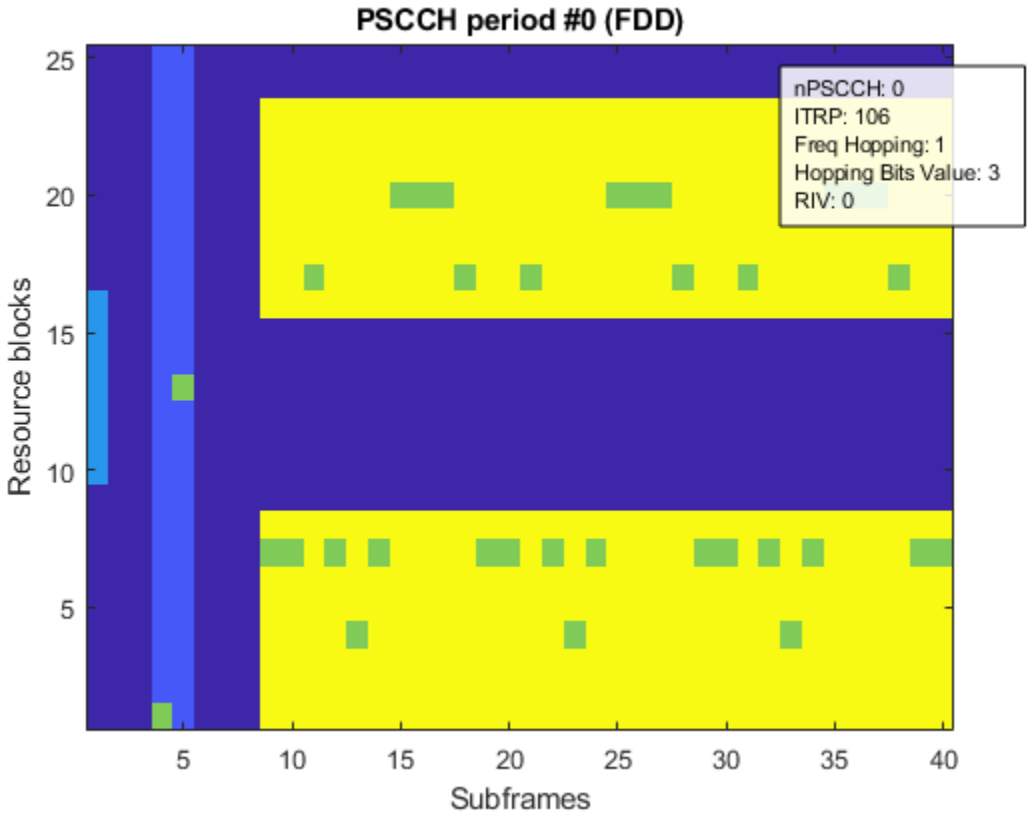
```
plot(abs(waveform)); title('PSCCH period baseband waveform');
snapnow;

subframes =

  Columns 1 through 13

     8     9    10    11    12    13    14    15    16    17    18    19    20

  Columns 14 through 26

    21    22    23    24    25    26    27    28    29    30    31    32    33

  Columns 27 through 32

    34    35    36    37    38    39

prb =

  1x32 uint64 row vector

  Columns 1 through 15

   12   12   11   12   23   12    0    0    0   11   12   12   11   12   23

  Columns 16 through 30

   12    0    0    0   11   12   12   11   12   23   12    0    0    0   11

  Columns 31 through 32

   12   12

poolindices =

  Columns 1 through 13

     0     1     2     3     4     5     6     7     8     9    10    11    12

  Columns 14 through 26

    13    14    15    16    17    18    19    20    21    22    23    24    25

  Columns 27 through 32

    26    27    28    29    30    31
```

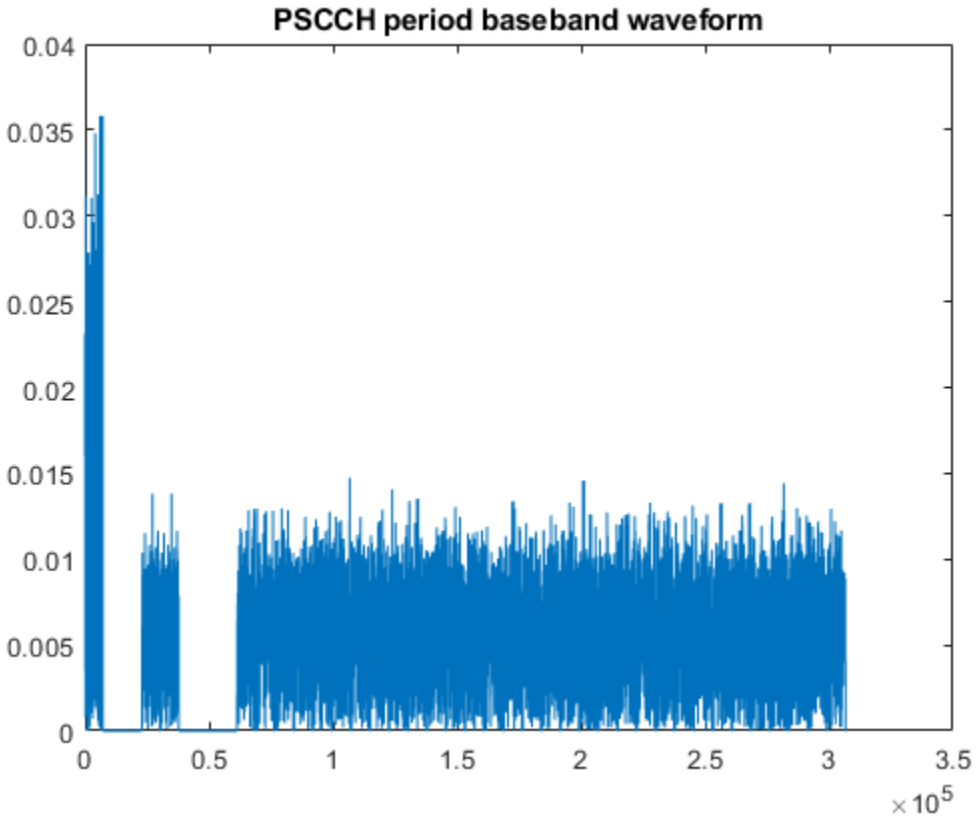



```
ans =

  struct with fields:

    hoppingParameter_r12: 504
         numSubbands_r12: 2
           rb_Offset_r12: 0
```




```
dataresconfig =

  struct with fields:

           prb_Num_r12: 13
         prb_Start_r12: 0
           prb_End_r12: 24
    offsetIndicator_r12: 0
     subframeBitmap_r12: '00000000111111111111111111111111111111111'
```


PSCCH period #0 (FDD)
nPSCCH: 0
ITRP: 106
Freq Hopping: 1
Hopping Bits Value: 3
RIV: 0
Resource blocks
Subframes
25
20
15
10
5
5
10
15
20
25
30
35
40



Appendix

This example uses this helper class.

- PSCCHPeriod

Selected Bibliography

3GPP TS 36.101 "User Equipment (UE) radio transmission and reception"

3GPP TS 36.211 "Physical channels and modulation"

3GPP TS 36.212 "Multiplexing and channel coding"

3GPP TS 36.213 "Physical layer procedures"

3GPP TS 36.321 "Medium Access Control (MAC) protocol specification"

3GPP TS 36.331 "Radio Resource Control (RRC) protocol specification"

Release 12 Sidelink PSCCH and PSSCH Throughput

This example shows how to perform a Block Error Ratio (BLER) simulation of the 3GPP Release 12 sidelink control and shared channels in frequency-selective fading and Additive White Gaussian Noise (AWGN) using LTE Toolbox™.

Introduction

3GPP Release 12 introduced the sidelink device-to-device communications for public safety in LTE. This example generates multiple Release 12 Physical Sidelink Control Channel (PSCCH) periods containing coded PSCCH and Physical Sidelink Shared Channel (PSSCH) transmissions. The resulting PSCCH period waveforms are then passed through a frequency-selective fading channel with AWGN, and the control channel and shared channel Block Error Ratios (BLERs) are calculated for a range of SNRs. The average number of transmission instances of the control and shared channels required for successful decoding at each SNR are also calculated. For information on how to model Release 14 V2X sidelink, check the following example: “Release 14 V2X Sidelink PSCCH and PSSCH Throughput” on page 2-263.

Number of PSCCH Periods to Simulate

The example is executed for a simulation length of 5 PSCCH periods to keep the simulation time low. A large number of periods, `nPeriods`, should be used to produce statistically meaningful results.

```
nPeriods = 5;
```

Fading Channel Configuration

A frequency-selective fading channel is configured according to TS 36.101 Table 12.2.1-1 [1]. An EVA delay profile with a Doppler frequency of 70Hz and 2 receiver antennas with low correlation is used. The channel seed is specified so that each execution of this example will use the same fading process realization. The Rayleigh fading model uses random phase initialization and the output is normalized so that the average power is unity.

```
channel.DelayProfile = 'EVA';
channel.DopplerFreq = 70.0;
channel.NRxAnts = 2;
channel.MIMOCorrelation = 'Low';
channel.Seed = 1;
channel.ModelType = 'GMEDS';
channel.NTerms = 16;
channel.NormalizeTxAnts = 'On';
channel.NormalizePathGains = 'On';
channel.InitPhase = 'Random';
```

SNR Configuration

Configure a range of SNR points, intended to cover both high and low BLER operating conditions.

```
SNRIn = [-10.0 -5.0 0.0 5.0];
```

Channel Estimator Configuration

The variable `perfectChanEstimator` controls channel estimator behavior. Valid values are `true` or `false`. When set to `true` a perfect channel response is used as the estimate, otherwise an imperfect estimation based on the values of received Demodulation Reference Signals (DRS) is obtained. If

`perfectChanEstimator` is set to `false`, a configuration structure `cec` is needed to parameterize the channel estimator.

```
% Perfect channel estimator flag
perfectChanEstimator = false;
```

The practical channel estimation configuration for the PSCCH and PSSCH channel estimators is defined below. A pilot averaging time window of 15 resource elements is used for both PSCCH and PSSCH, to ensure that the noise on the Demodulation Reference Signals (DRS) occurring in both slots is averaged. This averaging prioritizes pilot SNR improvement over accurate estimation of the channel time variation, which is reasonable as even at 70Hz Doppler frequency, the channel variation across a subframe is limited. A pilot averaging frequency window of 23 resource elements ensures that the noise on every DRS resource element across frequency is averaged for the PSCCH. This averaging prioritizes pilot SNR improvement over accurate estimation of the channel frequency selectivity, which is reasonable for the PSCCH as it is always QPSK modulated (so accuracy of channel estimation is not so critical) and it only occupies one resource block (the channel has limited frequency selectivity over this frequency span). For the PSSCH, this pilot averaging frequency window averages over roughly 2 resource blocks (10 resource blocks are allocated for the PSSCH for the parameters used in this example). Note that depending on the channel delay profile and Doppler frequency, the operating SNR, and the PSSCH modulation order and number of allocated resource blocks, different channel estimation parameters may give better performance for the PSSCH.

```
cec.PilotAverage = 'UserDefined';
cec.TimeWindow = 15;
cec.FreqWindow = 23;
cec.InterpType = 'linear';
```

Configure BLER measurements

The logical variables `measureBLERForSCI` and `measureBLERForSLSCH` allow the BLER measurements and all related receiver processing to be disabled for the SCI and SL-SCH respectively. This allows the simulation to be configured to measure BLER for only one of the channels. Disabling the receiver processing for the other channel improves the execution speed.

```
measureBLERForSCI = true;
measureBLERForSLSCH = true;
```

Create PSCCH Period Configuration

The class `PSCCHPeriod` represents the physical structure of a PSCCH period. See “LTE Sidelink Resource Pools and PSCCH Period” on page 2-215 and `PSCCHPeriod` for more information.

```
period = PSCCHPeriod;
```

Resource Pool Configuration

A resource pool configuration for the transmission of D2D sidelink communication consists of a large number of parameters which the UE receives via Radio Resource Control (RRC) messages [3]. Alternatively, in the case of a UE operating out of network coverage, these parameters are pre-configured within the UE. In this simulation, reference resource pool configuration #1-FDD from TS 36.101 Annex A.7.2.1-1 [1] is used. These parameters can be set using the `PSCCHPeriod.defaultConfig` function by specifying the reference resource pool configuration number and bandwidth.

```
% Configure the PSCCH period parameters for reference pool #1-FDD, 5MHz
period.Config = PSCCHPeriod.defaultConfig(1,'5MHz');
```

```
% Set the control and data channel cyclic prefix length
period.Config.sc_CP_Len_r12 = 'Normal';
period.Config.data_CP_Len_r12 = 'Normal';
```

Display Simulation Information

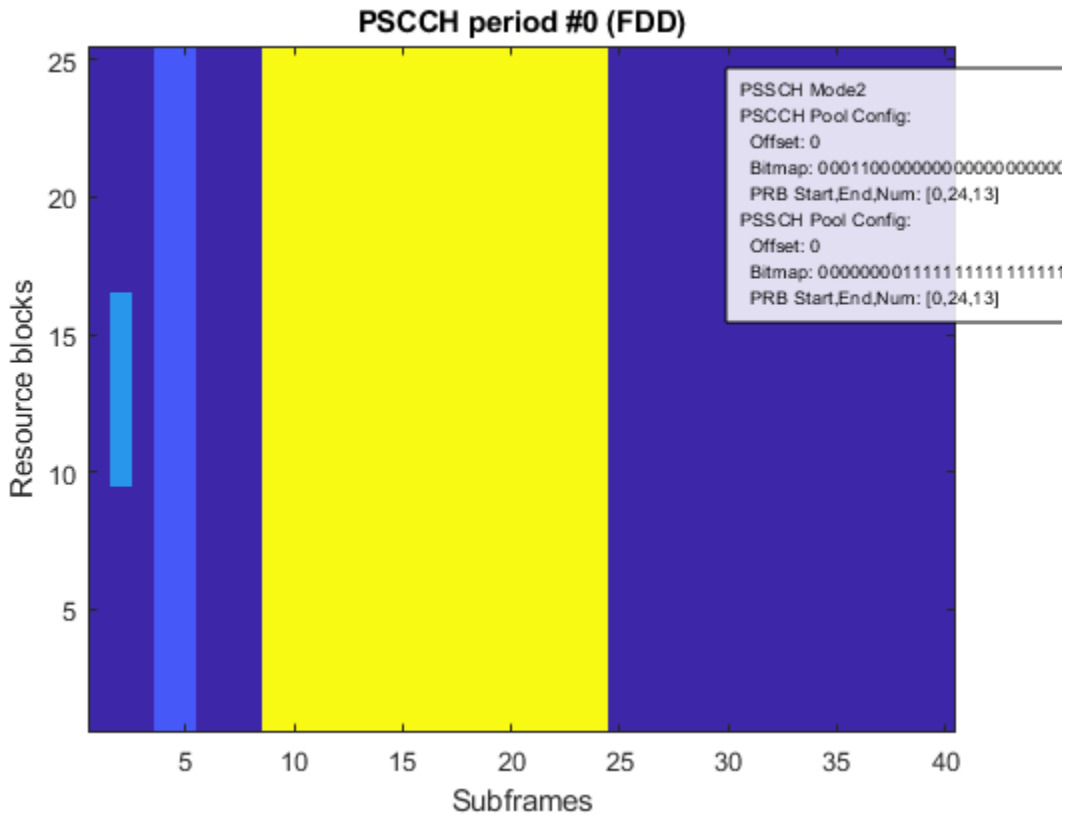
The variable `displaySimulationInformation` controls the display of simulation information such as the PSCCH resources used for each period, and whether or not SCI and SL-SCH decoding were successful.

```
displaySimulationInformation = true;
```

Display PSCCH Period

In order to illustrate the control and shared resource pools, the `period.displayPeriod` function is called which creates a plot showing the locations of the synchronization transmission (light blue), control resource pool (dark blue) and shared resource pool (yellow) within the PSCCH period.

```
if (displaySimulationInformation)
    figure;
    period.displayPeriod;
    drawnow;
end
```



Create Sidelink Control Information (SCI) Message

A Sidelink Control Information (SCI) message `sciMessage` is created according to sidelink reference measurement channel CC.3 FDD defined in TS 36.101 [1] Annex A.6.4 and CD.1 FDD defined in Annex A.6.5 as follows:

- The Transport Block Size for CD.1 FDD (872), used for the shared channel transmission in this example, is assigned to the variable `TBS`.
- The number of allocated resource blocks for CD.1 FDD (10) is assigned to the variable `NPRB`, which will be used to ensure that the resource allocation for the shared channel contains the correct number of resource blocks. Note that the code rate for the SL-SCH is a function of the PSSCH bit capacity (arising from `NPRB` and the modulation order) and the transport block size `TBS`.
- The number of resource blocks `NSLRB` in the UE-specific settings structure `ue` is set equal to that configured in the PSCCH period, `period.Config.NSLRB`.
- Frequency hopping is enabled by setting `sciMessage.FreqHopping` to 1.
- The SCI message `sciMessage` is then created using the `lteSCI` function. Enabling the frequency hopping in the SCI message structure prior to calling `lteSCI` ensures that the `HoppingBits` field of `sciMessage.Allocation` is created.
- The `HoppingBits` field is set to 1, which in conjunction with the `NSLRB` value configures Type 2 hopping as described in TS 36.213 Section 8.4 [2].
- The `TimeResourcePattern` is set as defined in TS 36.101 Table A.6.4-1 [1].
- The `ModCoding` field is set to the value corresponding to the Transport Block Size `TBS`.
- The `NSAID` field is set to an arbitrary 8-bit value (a value in the range from 0 to 255), representing the least significant 8 bits of the ProSe Layer-2 Group ID. TS 36.101 Table A.6.5-1 [1] states that the Group destination ID is "as set by higher layers". The value used is recorded in the variable `expectedNSAID` and is the only part of the SCI message content that the UE is aware of prior to reception (the UE obtains it from higher layer signaling). In this example, the NSAID value stored in `expectedNSAID` will be used by the receiver, and the receiver will not access `sciMessage` at all.

```
% Number of allocated resource blocks (NPRB) and Transport Block Size (TBS)
% for RMC CD.1 FDD, TS 36.101 Table A.6.5-1
NPRB = 10;
TBS = 872;

% Create UE configuration and SCI message with frequency hopping enabled
ue.NSLRB = period.Config.NSLRB;
sciMessage.FreqHopping = 1;
sciMessage.SCIFormat = 'Format0';
sciMessage = lteSCI(ue,sciMessage);

% Set the hopping bits for Type 2 hopping and the time resource pattern for
% RMC CC.3 FDD, TS 36.101 Table A.6.4-1
sciMessage.Allocation.HoppingBits = 1;
sciMessage.TimeResourcePattern = 8;

% Set the PSSCH MCS according to the TBS
ITBSs = lteMCS(0:28,'PUSCH');
sciMessage.ModCoding = find(lteTBS(NPRB,ITBSs)==TBS,1,'first') - 1;

% Set NSAID value, would be assigned by higher layers
```

```
sciMessage.NSAID = 117;

% Store NSAID value in 'expectedNSAID' so that the receiver won't have to
% access 'sciMessage' at all
expectedNSAID = sciMessage.NSAID;
```

Create Set of Valid Resource Indicator Values

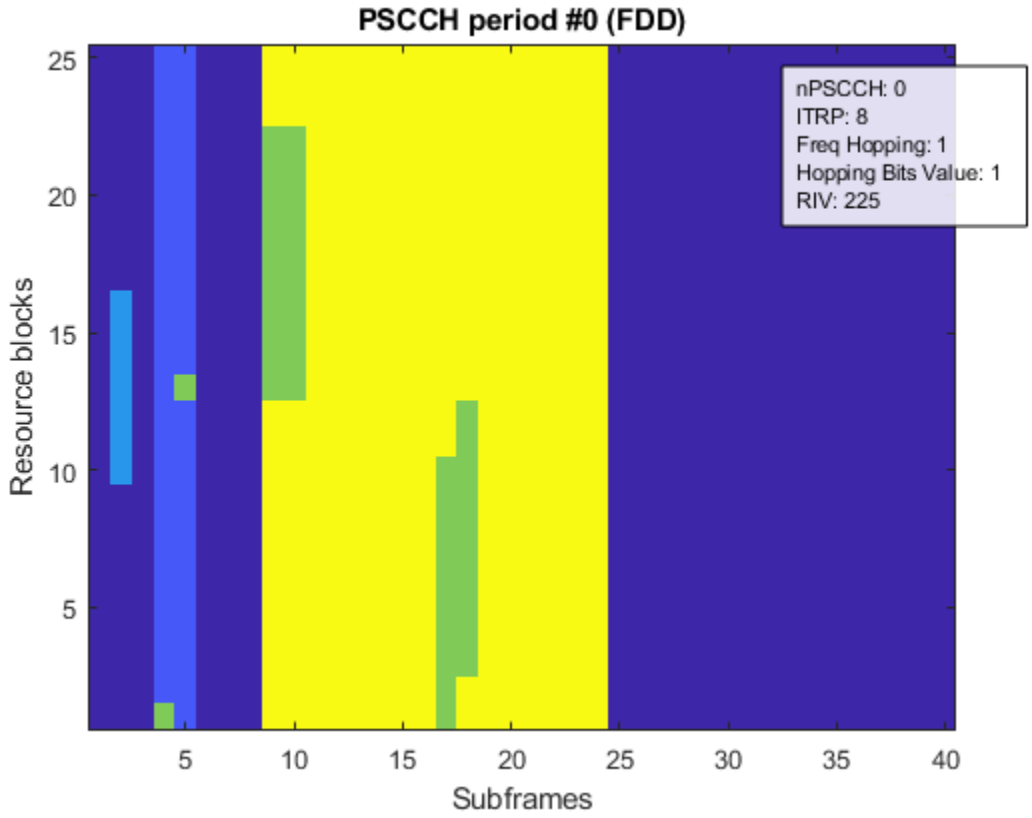
The set of Resource Indicator Values `RIVset` with the correct allocation size of 10 PRBs (given by RMC CD.1 FDD) are created. The `period.getAllowedRIV` function returns a vector of RIV values (`RIVset`) and a corresponding matrix of resource allocations (`range`) where each row contains the allocation length and starting resource block for each RIV. The `RIVset` vector is reduced to just the entries which have an allocation length equal to `NPRB` by performing logical indexing.

```
[RIVset,range] = period.getAllowedRIV(sciMessage);
RIVset = RIVset(range(:,1) == NPRB);
```

Display the PSCCH Period Including PSSCH Resources Selected by SCI Message

An example of the actual PSCCH and PSSCH subframes and resource blocks for transmission will be plotted. The SCI message is configured with the first PSCCH resource value `PSCCHResource=0` and the first PSSCH RIV value `Allocation.RIV = RIVset(1)`. The `period.displayPeriod` function is called, passing in the SCI message, and the sets of PSCCH and PSSCH subframes and resource blocks that will actually be used for transmission are plotted. The subframes and resource blocks are colored green, and are a subset of the overall control and shared resource pool, shown in blue and yellow respectively. See the “LTE Sidelink Resource Pools and PSCCH Period” on page 2-215 example for more information. In accordance with the definition in TS 36.101 Table 12.2.1-1 [1], the RIV value transmitted in the SCI message (which determines the PSSCH transmission resources within the pool) will be selected randomly (uniformly) from the set `RIVset`, for each PSCCH period. Similarly, the PSCCH resource value (which determines the PSCCH transmission resources within the pool) will be selected randomly (uniformly) across its valid range from 0 to `NumPSCCHReource-1`, for each PSCCH period.

```
% Select the first PSCCH and PSSCH resource
sciMessage.PSCCHResource = 0;
sciMessage.Allocation.RIV = RIVset(1);

% Display the subframes and resource blocks for transmission
if (displaySimulationInformation)
    figure;
    period.displayPeriod(sciMessage);
    drawnow;
end
```



Select Receiver Behavior When SCI Decoding Fails

The logical variable `sciAssumed` controls the simulation behavior in terms of the effect the control channel decoding has on the shared channel decoding. If `sciAssumed` is true, the receiver will assume that the SCI message was correctly decoded when attempting shared channel reception. This allows the performance of the shared channel reception to be measured independently from the performance of the control channel reception. If `sciAssumed` is false, a control channel decoding failure implies a shared channel decoding failure. Note that if the BLER measurement for the SCI is disabled above (`measureBLERForSCI=false`), and the BLER measurement for the SL-SCH is enabled (`measureBLERForSLSCH=true`), the receiver will assume that the SCI message was correctly decoded when measuring SL-SCH BLER regardless of the setting of `sciAssumed` here.

```
sciAssumed = true;
```

Create Variables to Record Simulated Performance

The performance of the receiver is recorded in the matrices `controlErrors` and `sharedErrors`. Each matrix has a row for each SNR point and a column for each PSCCH period simulated. The elements will be set to 1 for SNR points/PSCCH periods where the respective channel combination (PSCCH and SCI or PSSCH and SL-SCH) fails to decode successfully. Additionally, vectors `controlRxs` and `sharedRxs` record the number of transmission instances that were combined before successful decoding in each period. Note that the number of transmission instances is 2 for the control channel and 4 for the shared channel. The vectors `controlRxs` and `sharedRxs` are initialized with these maximum values, which will be the values returned in periods where the control and shared channel decoding fail.

```
controlErrors = zeros(length(SNRIn),nPeriods);
sharedErrors = zeros(length(SNRIn),nPeriods);

controlRxs = ones(length(SNRIn),nPeriods) * 2;
sharedRxs = ones(length(SNRIn),nPeriods) * 4;
```

Processing

For each PSCCH period, the following operations are performed:

- *Update the PSCCH period number*: The field `NPSCCHPeriod` of the period configuration is updated to match the currently transmitted period. This field is used in conjunction with the Sidelink Control (SC) period `sc_Period_r12` and offset indicator `sc_TF_ResourceConfig_r12.offsetIndicator_r12` to determine the Direct Frame Number (DFN) and subframe number for each subframe in the waveform.

- *Choose a random PSCCH resource*: For a sidelink UE in transmission mode 2 (e.g. out-of-coverage), the PSCCH resource `PSCCHResource` must be chosen randomly according to a uniform distribution between 0 and `period.NumPSCCHReource-1`. The PSCCH resource will select a random pair of subframes and Physical Resource Blocks (PRBs) in the PSCCH resource pool in which to transmit the two transmission instances of the SCI message. The `PSCCHResource` is added to the SCI message structure in order to pass it into the `period.generateWaveform` function. Note that for an in-coverage UE (i.e. operating in transmission mode 1), the control information is given by a DCI format 5 message sent by the eNodeB, which includes a `PSCCHResource` chosen by the eNodeB.

- *Choose a random PSSCH resource*: As given by the definition in TS 36.101 Table 12.2.1-1 [1], the Resource Indicator Value (RIV) transmitted in the SCI message will be selected randomly (uniformly) from the set `RIVset`, the RIVs whose number of allocated PRBs matches the configured RMC CD.1 FDD.

- *Generate PSCCH period waveform*: The PSCCH period waveform is generated by the `PSCCHPeriod.generateWaveform` function using its configuration parameters and the content of the SCI message passed in, plus the `PSCCHResource` field added to the message in the previous step.

- *Apply frequency-selective fading*: The PSCCH period waveform is passed through a frequency-selective fading channel using the EVA delay profile with a Doppler frequency of 70Hz and 2 receiver antennas with low correlation, as described in TS 36.101 Table 12.2.1-1 [1].

- *Add AWGN*: AWGN is added to achieve the specified SNR per resource element.

- *Receive the control and shared channels*: The receiver processing consists of PSCCH demodulation, SCI decoding, PSSCH demodulation and SL-SCH decoding. These steps are described in more detail below.

Note that the "Add AWGN" and "Receive control and shared channels" steps are repeated for each SNR point using the same faded waveform.

The receiver performs the following steps:

- *Synchronize and SC-FDMA demodulate each subframe in the PSCCH subframe pool*: Before attempting PSCCH demodulation and SCI decoding for each PSCCH resource, every subframe in the PSCCH subframe pool is SC-FDMA demodulated. For each subframe, the appropriate subframe of the waveform is synchronized using `lteSLFrameOffsetPSCCH`. The PSCCH

resource used in the transmitter is unknown, therefore the PRB used by the PSCCH in the subframe is unknown, so the synchronization is performed for each PRB in the PSCCH resource block pool and the timing offset for the correlation with the strongest peak is used. The synchronized waveform is then SC-FDMA demodulated using `lteSLSCFDMADemodulate`. Since the number of subframes in the PSCCH pool is likely to be small compared to the total number of PSCCH resource values (with multiple PSCCH resource values corresponding to the same subframe in the pool), it is more efficient to cache the SC-FDMA demodulated subframes and use the resulting resource grids indicated by the PSCCH subframe resources for a given PSCCH resource value. Therefore, the waveform is SC-FDMA demodulated outside of the PSCCH resource value loop. Note that the approach of choosing the timing offset corresponding to the PRB with the strongest correlation peak assumes that only one PSCCH is being transmitted. A more robust (but more expensive) approach would be to perform timing synchronization and SC-FDMA demodulation for each PSCCH resource.

For each PSCCH resource value until the SCI is decoded:

- *Get the PSCCH resources*: Within a period, the PSCCH is transmitted twice for a configured PSCCH resource `pscchResource`. The `period.getPSCCHResources` function provides the two subframes that carry PSCCH and corresponding PRBs allocated to PSCCH within those subframes. The subframe numbers are used to extract the appropriate subframes from the PSCCH period waveform, and the PRBs are used to configure the PSCCH resource extraction within those subframes. Note that the PSCCH will not be transmitted in synchronization subframes.

For each PSCCH transmission instance for the current PSCCH resource value:

- *Perform PSCCH channel estimation*: PSCCH channel estimation is performed using `lteSLChannelEstimatePSCCH` or the local function `perfectChannelEstimate` depending on the value of `perfectChanEstimator`. The channel estimator also produces an estimate of the noise power which can be used for MMSE equalization.

- *Extract the PSCCH symbols and channel estimate*: The received PSCCH symbols are extracted from the subframe resource grid and the corresponding channel estimates are extracted using `lteExtractResources` and the indices provided by `ltePSCCHIndices`.

- *Perform PSCCH equalization*: The PSCCH symbols are MMSE equalized using `lteEqualizeMMSE` with the channel estimate and noise estimate obtained above.

- *Perform PSCCH demodulation*: The equalized PSCCH symbols are demodulated using `ltePSCCHDecode`. This function performs the inversion of the transmitter modulation steps (SC-FDMA transform deprecoding, QPSK symbol demodulation and descrambling).

- *Perform SCI decoding*: SCI decoding is attempted using `lteSCIDecode`. The number of original information bits in the SCI message is given by `lteSCIInfo`. If the decoded CRC is zero, the decoded message bits are converted into the corresponding message structure using `lteSCI`. The `NSAID` message field, which is the eight LSBs of the group destination ID, is compared with the expected NSAID value. If they are equal, the SCI decoding is considered successful and the PSCCH resource value loop is terminated. The number of required transmission instances (1 or 2) is recorded in `controlRxs`.

The behavior of the shared channel decoding with respect to a failed SCI decoding is controlled by the variable `sciAssumed`. If `sciAssumed` is false, the failed SCI decoding immediately implies a failure to decode the SL-SCH and no further processing takes place for the current PSCCH period. If `sciAssumed` is true, the transmitted SCI message is assumed to be known to the receiver and will be used in place of the received SCI message. If `sciAssumed` is true or if SCI decoding was successful,

the receiver proceeds with SL-SCH decoding. Note that if SCI BLER measurement is disabled but SL-SCH BLER measurement is enabled then the transmitted SCI message is assumed to be known to the receiver.

The PSSCH / SL-SCH is transmitted four times, using HARQ where four redundancy versions (RV) with the sequence [0 2 3 1] are transmitted. There is no HARQ feedback as the sidelink communication transmissions are for a group of UEs (not for an individual receiving UE).

For each PSSCH transmission instance until the SL-SCH is decoded:

- *Get the PSSCH resources*: The `period.getPSSCHResources` function provides a vector of the subframes within the period that carry PSSCH and a matrix containing the PRB sets allocated to PSSCH within those subframes for the PSSCH configuration given by the decoded SCI message. This is similar to the resources described above for the PSCCH, with the difference being that the PSSCH can be sent on multiple PRBs but the PSCCH is only sent on a single PRB. As with the PSCCH, the subframe numbers are used to extract the appropriate subframes from the PSCCH period waveform, and the PRBs are used to configure the PSSCH resource extraction within those subframes. Additionally, `period.getPSSCHResources` returns a vector of PSCCH subframe numbers, which are the subframe numbers within the PSSCH resource pool for which this PSSCH is transmitted. These values are used to configure the descrambling of the PSSCH in each subframe. Since the PSCCH is not transmitted in synchronization subframes, any PSCCH subframes that correspond to synchronization subframes are configured to be skipped, by removing them from the vectors `sf` and `prb`. Prior to skipping any synchronization subframes, the redundancy version indices `IRV` for SL-SCH decoding are calculated for each resource. These indices must be computed prior to the skipping of any synchronization subframes because the index sequence is related solely to the position of the PSSCH subframes within the subframe pool.

- *Synchronize and SC-FDMA demodulate subframe carrying PSSCH*: The appropriate subframe of the waveform is synchronized using `lteSLFrameOffsetPSSCH`. The synchronized waveform is SC-FDMA demodulated using `lteSLSCFDMADemodulate`. Note that unlike the timing synchronization and SC-FDMA demodulation for the PSCCH, the synchronization and demodulation for the PSSCH is carried out for each transmission instance (whereas for the PSCCH, all subframes in the PSCCH resource pool were demodulated first). The reason is that unlike the PSCCH, the PSSCH transmission instances occur in distinct subframes and those subframes are likely to be a small subset of the overall PSSCH subframe pool. Therefore, it is efficient to only synchronize and SC-FDMA demodulate the subframes corresponding to a PSSCH transmission instance.

- *Perform PSSCH channel estimation*: PSSCH channel estimation is performed using `lteSLChannelEstimatePSSCH` or the local function `perfectChannelEstimate` depending on the value of `perfectChanEstimator`. The channel estimator also produces an estimate of the noise power which can be used for MMSE equalization.

- *Extract the PSSCH symbols and channel estimate*: The received PSSCH symbols are extracted from the subframe resource grid and the corresponding channel estimates are extracted using `lteExtractResources` and the indices provided by `ltePSSCHIndices`.

- *Perform PSSCH equalization*: The PSSCH symbols are MMSE equalized using `lteEqualizeMMSE` with the channel estimate and noise estimate obtained above.

- *Perform PSSCH demodulation*: The equalized PSSCH symbols are demodulated using `ltePSSCHDecode`. This function performs the inversion of the transmitter modulation steps (SC-FDMA transform deprecoding, QPSK or 16QAM symbol demodulation and descrambling).

- *Perform SL-SCH decoding*: The received redundancy versions are combined and decoded using `lteSLSCHDecode`. If the SL-SCH CRC is zero, the HARQ combining loop is terminated and the SL-SCH decoding is successful. The number of required redundancy versions (from 1 to 4) is recorded in `sharedRxs`.

```
% If SCI BLER is not being measured, 'sciAssumed' must be set so that the
% receiver can assume knowledge of the SCI
if (~measureBLERForSCI)
    sciAssumed = true;
end

rng('default');

% Repeat for each PSCCH period
p = 1;
while (p<=nPeriods)

    % Update PSCCH period number
    period.Config.NPSCCHPeriod = p - 1;

    % Choose a random PSCCH resource
    sciMessage.PSCCHResource = randi([0 period.NumPSCCHResource-1]);

    % Choose a random PSSCH resource
    sciMessage.Allocation.RIV = RIVset(randi(length(RIVset)));

    if (displaySimulationInformation)
        fprintf('\nSimulating PSCCH period %d of %d\n',p,nPeriods);
        fprintf('Randomly selected PSCCH resource = %d\n',sciMessage.PSCCHResource);
        fprintf('Randomly selected PSSCH RIV = %d\n',sciMessage.Allocation.RIV);
    end

    % Generate PSCCH period waveform
    txWaveform = period.generateWaveform(sciMessage);

    % Set the sampling rate and init time for frequency-selective fading
    ue.CyclicPrefixSL = period.Config.sc_CP_Len_r12;
    info = lteSLSCFDMAInfo(ue);
    channel.SamplingRate = info.SamplingRate;
    channel.InitTime = (p-1)*period.Config.sc_Period_r12*0.001;
    periodInitTime = channel.InitTime; % Cache the period init time
    % Apply frequency-selective fading
    [fadedWaveform, fadingInfo] = lteFadingChannel(channel,txWaveform);

    % For each SNR point, add AWGN and receive control and shared channels
    for i = 1:length(SNRIn)

        if (displaySimulationInformation && (measureBLERForSCI || measureBLERForSLSCH))
            fprintf('Receiving at %gdB SNR\n',SNRIn(i));
        end

        % Add AWGN
        SNR = 10^(SNRIn(i)/20);
        N = 1/(SNR*sqrt(double(info.Nfft)))/sqrt(2.0);
        noise = N*complex(randn(size(fadedWaveform)), randn(size(fadedWaveform)));
        rxWaveform = fadedWaveform + noise;
```

```
% Create PSCCH receiver configuration
pscch.SidelinkMode = 'D2D';
pscch.NSLRB = period.Config.NSLRB;
pscch.CyclicPrefixSL = period.Config.sc_CP_Len_r12;

% Establish the number of time-domain samples 'Nt' per subframe and
% number of SC-FDMA symbols 'L' per subframe
pscchGridInfo = lteSLSCFDMAInfo(pscch);
Nt = pscchGridInfo.SamplingRate * 1e-3;
L = numel(pscchGridInfo.CyclicPrefixLengths);

% If SCI BLER measurement is configured:
sciDecoded = false;
if (measureBLERForSCI)
    % Synchronize and extract each subframe of the PSCCH subframe
    % pool from the received waveform and perform SC-FDMA
    % demodulation. The timing offset used is the offset
    % corresponding to the strongest correlation peak when
    % correlating with the PSCCH DRS for each PRB in the PSCCH
    % subframe pool
    nSubframes = period.Config.sc_Period_r12;
    pscchPoolGrid = repmat(lteSLResourceGrid(pscch),1,nSubframes,size(rxWaveform,2));
    offsetpool = []; % Variable to store the offset for each PSCCH subframe
    for l = period.PSCCHSubframePool
        if(perfectChanEstimator)
            offset  = hPerfectTimingEstimate(fadingInfo);
        else
            subframeWaveform = rxWaveform(l*Nt + (1:Nt),:);
            bestCorr = 0.0;
            for prb = period.PSCCHResourceBlockPool.'
                pscch.PRBSet = prb;
                [thisoffset,thiscorr] = lteSLFrameOffsetPSCCH(pscch,subframeWaveform);
                % Find the best correlation across all antennas and
                % store the corresponding offset
                maxCorr = max(thiscorr(:));
                if maxCorr > bestCorr
                    bestCorr = maxCorr;
                    offset = max(0,thisoffset); % No negative offset
                end
            end
        end
        subframe = lteSLSCFDMADemodulate(pscch,rxWaveform(l*Nt + offset + (1:Nt),:));
        pscchPoolGrid(:,l*L + (1:L),:) = subframe;
        offsetpool = [offsetpool offset]; %#ok<AGROW>
    end

    % Repeat for each PSCCH resource until the SCI is decoded
    pscchResource = 0;
    while (pscchResource < period.NumPSCCHResource && ~sciDecoded)

        % Get the PSCCH subframes 'sf' and PRB allocations 'prb'
        % for the current PSCCH resource
        [sf,prb] = period.getPSCCHResources(pscchResource);

        % Remove any resources that overlap with synchronization
        % subframes. PSCCH will not be transmitted in these
        % subframes
        [~,syncIndex] = intersect(sf,period.SyncSubframes);
```

```
sf(syncIndex) = [];
prb(syncIndex) = [];

% Repeat for each PSCCH transmission instance until the SCI
% is decoded
tx = 1;
while (tx <= min(length(sf),2) && ~sciDecoded)

    % Configure the PSCCH receiver for the PRB allocation
    pscch.PRBSet = prb(:,tx);

    % Select the appropriate subframe resource grid from
    % the PSCCH subframe pool
    subframe = pscchPoolGrid(:,sf(tx)*L + (1:L),:);

    % Perform channel estimation
    if(perfectChanEstimator)
        % Update the init time to correspond to the current
        % subframe
        channel.InitTime = periodInitTime + (sf(tx)*Nt/channel.SamplingRate);
        [hest,nest] = perfectChannelEstimate(pscch,channel,noise,offsetpool(tx));
    else
        [hest,nest] = lteSLChannelEstimatePSCCH(pscch,cec,subframe);
    end
    % Extract the received PSCCH symbols and the
    % corresponding channel estimate, and perform
    % equalization
    [pscchIndices,pscchIndicesInfo] = ltePSCCHIndices(pscch);
    [pscchRx,pscchHest] = lteExtractResources(pscchIndices,subframe,hest);
    pscchSymbols = lteEqualizeMMSE(pscchRx,pscchHest,nest);

    % If we are receiving the first PSCCH transmission
    % instance, reset the receiver buffer
    if (tx==1)
        codedSciBits = zeros(pscchIndicesInfo.G,1);
    end

    % Demodulate the PSCCH and add the result into the
    % receiver buffer
    codedSciBits = codedSciBits + ltePSCCHDecode(pscchSymbols);

    % Decode the SCI message. If successful (CRC=0), check
    % the NSAID field of the decoded message against the
    % transmitted message
    sciInfo = lteSCIInfo(pscch);
    [sciBits,sciCRC] = lteSCIDecode(sciInfo.Format0,codedSciBits);
    if (sciCRC==0)
        sciMessageRx = lteSCI(pscch,sciBits);
        if (sciMessageRx.NSAID==expectedNSAID)
            sciDecoded = true;
            controlRxs(i,p) = tx;
            if (displaySimulationInformation)
                fprintf('   SCI decoded, transmissions combined = %d\n',tx);
            end
        else
            if (displaySimulationInformation)
                fprintf('   SCI decoded, but NSAID value (%d) did not match expe
            end
```

```
                end
            end

            % Increment the PSCCH transmission instance index
            tx = tx + 1;

        end

        % Increment the PSCCH resource number
        pscchResource = pscchResource + 1;

    end

end

% If SCI decoding failed and the SCI is assumed for SL-SCH
% decoding, set the decoded SCI message equal to the transmitted
% SCI message
if (~sciDecoded)
    if (displaySimulationInformation && measureBLERForSCI)
        fprintf('   SCI decoding failed\n');
    end
    if (sciAssumed)
        sciMessageRx = sciMessage;
    end
end

% If SL-SCH BLER measurement is configured and if the SCI was
% successfully decoded or if SCI decoding success is assumed,
% perform PSSCH reception and SL-SCH decoding
slschDecoded = false;
if (measureBLERForSLSCH && (sciDecoded || sciAssumed))

    % Create PSSCH receiver configuration
    pssch.SidelinkMode = 'D2D';
    pssch.NSLRB = period.Config.NSLRB;
    pssch.CyclicPrefixSL = period.Config.data_CP_Len_r12;
    pssch.NTurboDecIts = 5;

    % Establish the number of time-domain samples 'Nt' per subframe
    psschGridInfo = lteSLSCFDMAInfo(pssch);
    Nt = psschGridInfo.SamplingRate * 1e-3;

    % Get the PSSCH subframes 'sf', PRB allocations 'prb' and PSSCH
    % subframe numbers 'nsf' for the configuration given by the
    % decoded SCI message. 'sf' are the subframe numbers within the
    % SC period, 'nsf' are the subframe numbers within the PSSCH
    % subframe pool
    [sf,prb,nsf] = period.getPSSCHResources(sciMessageRx);

    % Create the redundancy version index sequence (IRV)
    % corresponding to the PSSCH transmission instances. This must
    % be computed "up front" prior to the removal of any
    % synchronization subframes in the next step, otherwise the RV
    % sequence is lost
    IRV = 0:length(sf)-1;

    % Remove any resources that overlap with synchronization
```

```
% subframes. PSSCH will not be transmitted in these subframes
[~,syncIndex] = intersect(sf,period.SyncSubframes);
sf(syncIndex) = [];
prb(:,syncIndex) = [];
nsf(syncIndex) = [];
IRV(syncIndex) = [];

% Configure the PSSCH receiver NSAID value and modulation from
% the decoded SCI message, and establish the SL-SCH transport
% block size (TBS)
pssch.NSAID = sciMessageRx.NSAID;
[ITBS,modulation] = lteMCS(sciMessageRx.ModCoding,'PUSCH');
if (strcmpi(modulation,'64QAM'))
    modulation = '16QAM';
end
pssch.Modulation = modulation;
NPRB = max(size(prb,1),1);
TBS = lteTBS(NPRB,ITBS);

% Repeat for each PSSCH transmission instance until the SL-SCH
% is decoded
tx = 1;
slschDecState = [];
rvsequence = [0 2 3 1];
while (tx <= min(length(sf),4) && ~slschDecoded)

    % Configure the PSSCH receiver for the PSSCH subframe
    % number, PRB allocation and redundancy version (RV). If
    % the RV index is zero (corresponding to the first PSSCH
    % transmission instance in a block of 4 transmissions),
    % reset the receiver buffer
    pssch.NSubframePSSCH = nsf(tx);
    pssch.PRBSet = prb(:,tx);
    pssch.RV = rvsequence(mod(IRV(tx),4)+1);
    if (IRV(tx)==0)
        slschDecState = [];
    end

    % Perform timing synchronization, extract the appropriate
    % subframe of the received waveform, and perform SC-FDMA
    % demodulation
    if(perfectChanEstimator)
        offset  = hPerfectTimingEstimate(fadingInfo);
    else
        offset = lteSLFrameOffsetPSSCH(pssch,rxWaveform(sf(tx)*Nt + (1:Nt),:));
        offset = max(0,offset); % No negative offset
    end
    subframeWaveform = rxWaveform(sf(tx)*Nt + offset + (1:Nt),:);
    subframe = lteSLSCFDMADemodulate(pssch,subframeWaveform);

    % Perform channel estimation, extract the received PSSCH
    % symbols and the corresponding channel estimate, and
    % perform equalization
    if(perfectChanEstimator)
        % Update the init time to correspond to the current
        % subframe
        channel.InitTime = periodInitTime + (sf(tx)*Nt/channel.SamplingRate);
        [hest,nest] = perfectChannelEstimate(pssch,channel,noise,offset);
```

```
                else
                    [hest,nest] = lteSLChannelEstimatePSSCH(pssch,cec,subframe);
                end
                psschIndices = ltePSSCHIndices(pssch);
                [psschRx,psschHest] = lteExtractResources(psschIndices,subframe,hest);
                psschSymbols = lteEqualizeMMSE(psschRx,psschHest,nest);

                % Demodulate the PSSCH
                codedSlschBits = ltePSSCHDecode(pssch,psschSymbols);

                % Decode the SL-SCH including soft combining into the
                % receiver buffer and check the CRC
                [~,slschCRC,slschDecState] = lteSLSCHDecode(pssch,TBS,codedSlschBits,slschDecSta
                if (slschCRC==0)
                    slschDecoded = true;
                    sharedRxs(i,p) = tx;
                    if (displaySimulationInformation)
                        fprintf('   SL-SCH decoded, transmissions combined = %d\n',tx);
                    end
                end

                % Increment the PSSCH transmission instance index
                tx = tx + 1;

            end

        end

        % Record control and shared channel errors
        controlErrors(i,p) = ~sciDecoded;
        sharedErrors(i,p) = ~slschDecoded;

        if (displaySimulationInformation && ~slschDecoded && measureBLERForSLSCH)
            fprintf('   SL-SCH decoding failed\n');
        end

    end

    % Update the PSCCH period number
    p = p + 1;

end
```

```
Simulating PSCCH period 1 of 5
Randomly selected PSCCH resource = 19
Randomly selected PSSCH RIV = 239
Receiving at -10dB SNR
   SCI decoding failed
   SL-SCH decoding failed
Receiving at -5dB SNR
   SCI decoded, transmissions combined = 2
   SL-SCH decoded, transmissions combined = 2
Receiving at 0dB SNR
   SCI decoded, transmissions combined = 1
   SL-SCH decoded, transmissions combined = 1
Receiving at 5dB SNR
   SCI decoded, transmissions combined = 1
```

```
   SL-SCH decoded, transmissions combined = 1

Simulating PSCCH period 2 of 5
Randomly selected PSCCH resource = 12
Randomly selected PSSCH RIV = 238
Receiving at -10dB SNR
   SCI decoding failed
   SL-SCH decoded, transmissions combined = 4
Receiving at -5dB SNR
   SCI decoding failed
   SL-SCH decoded, transmissions combined = 2
Receiving at 0dB SNR
   SCI decoded, transmissions combined = 1
   SL-SCH decoded, transmissions combined = 1
Receiving at 5dB SNR
   SCI decoded, transmissions combined = 1
   SL-SCH decoded, transmissions combined = 1

Simulating PSCCH period 3 of 5
Randomly selected PSCCH resource = 10
Randomly selected PSSCH RIV = 226
Receiving at -10dB SNR
   SCI decoding failed
   SL-SCH decoded, transmissions combined = 4
Receiving at -5dB SNR
   SCI decoded, transmissions combined = 2
   SL-SCH decoded, transmissions combined = 1
Receiving at 0dB SNR
   SCI decoded, transmissions combined = 1
   SL-SCH decoded, transmissions combined = 1
Receiving at 5dB SNR
   SCI decoded, transmissions combined = 1
   SL-SCH decoded, transmissions combined = 1

Simulating PSCCH period 4 of 5
Randomly selected PSCCH resource = 1
Randomly selected PSSCH RIV = 237
Receiving at -10dB SNR
   SCI decoded, transmissions combined = 1
   SL-SCH decoding failed
Receiving at -5dB SNR
   SCI decoded, transmissions combined = 1
   SL-SCH decoded, transmissions combined = 2
Receiving at 0dB SNR
   SCI decoded, transmissions combined = 1
   SL-SCH decoded, transmissions combined = 1
Receiving at 5dB SNR
   SCI decoded, transmissions combined = 1
   SL-SCH decoded, transmissions combined = 1

Simulating PSCCH period 5 of 5
Randomly selected PSCCH resource = 15
Randomly selected PSSCH RIV = 237
Receiving at -10dB SNR
   SCI decoding failed
   SL-SCH decoding failed
Receiving at -5dB SNR
   SCI decoding failed
```

```
   SL-SCH decoded, transmissions combined = 3
Receiving at 0dB SNR
   SCI decoded, transmissions combined = 1
   SL-SCH decoded, transmissions combined = 2
Receiving at 5dB SNR
   SCI decoded, transmissions combined = 1
   SL-SCH decoded, transmissions combined = 1
```

Calculate Control Channel and Shared Channel BLER

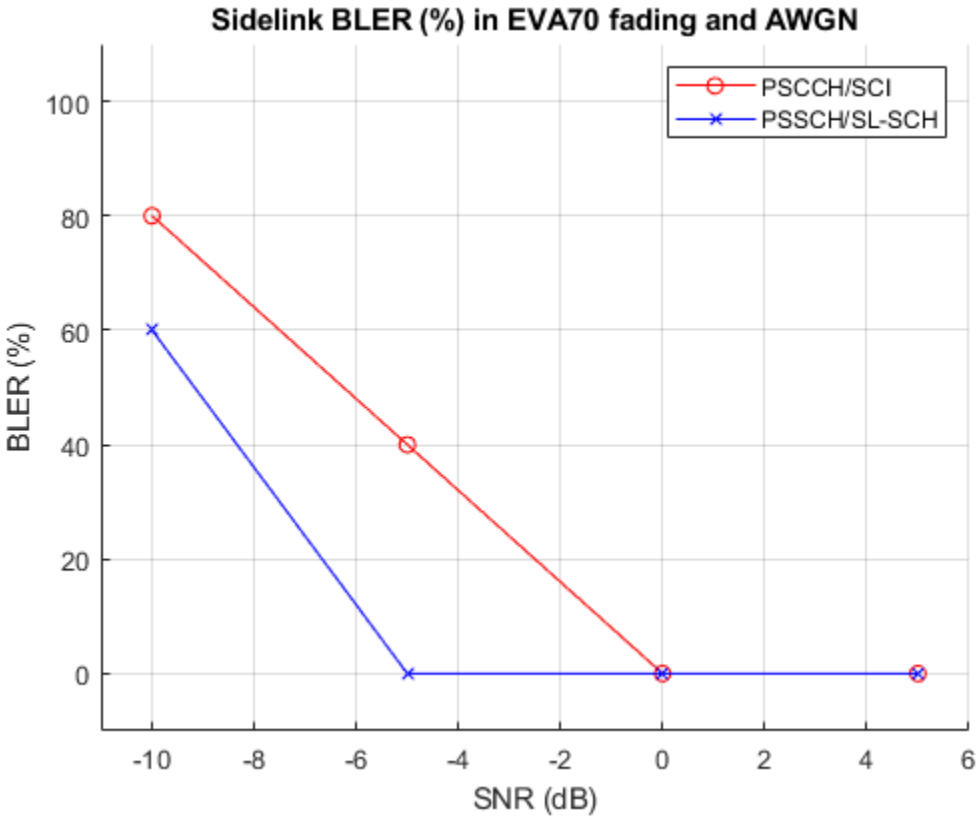
The overall BLER for the control channel (PSCCH and SCI) and shared channel (PSSCH and SL-SCH) are computed by averaging the `controlErrors` and `sharedErrors` matrices along their second dimension i.e. averaging each row, to produce vectors `controlBLER` and `sharedBLER` containing the BLER values for each SNR point tested. Similarly, `controlRxs` and `sharedRxs` are averaged to produce vectors `controlRxsAvg` and `sharedRxsAvg` containing the average number of transmission instances that needed to be combined for successful reception.

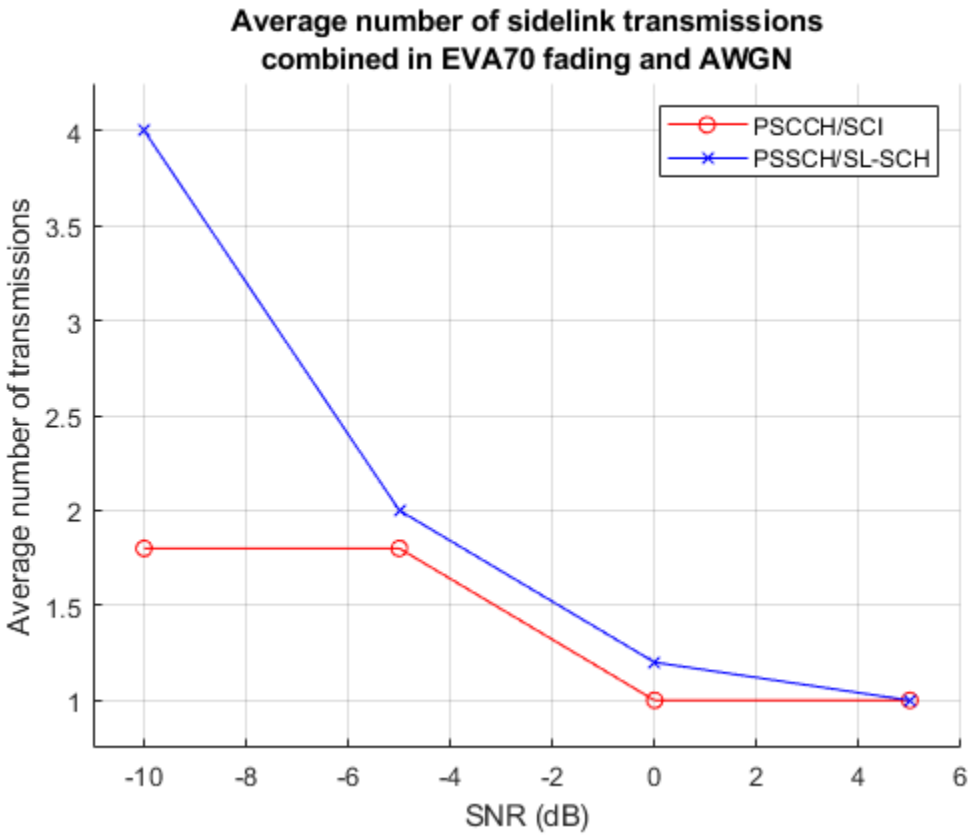
```
if (measureBLERForSCI)
    controlBLER = 100*mean(controlErrors,2);
    controlRxsAvg = mean(controlRxs,2);
else
    controlBLER = [];
    controlRxsAvg = [];
end

if (measureBLERForSLSCH)
    sharedBLER = 100*mean(sharedErrors,2);
    sharedRxsAvg = mean(sharedRxs,2);
else
    sharedBLER = [];
    sharedRxsAvg = [];
end
```

Plot Results

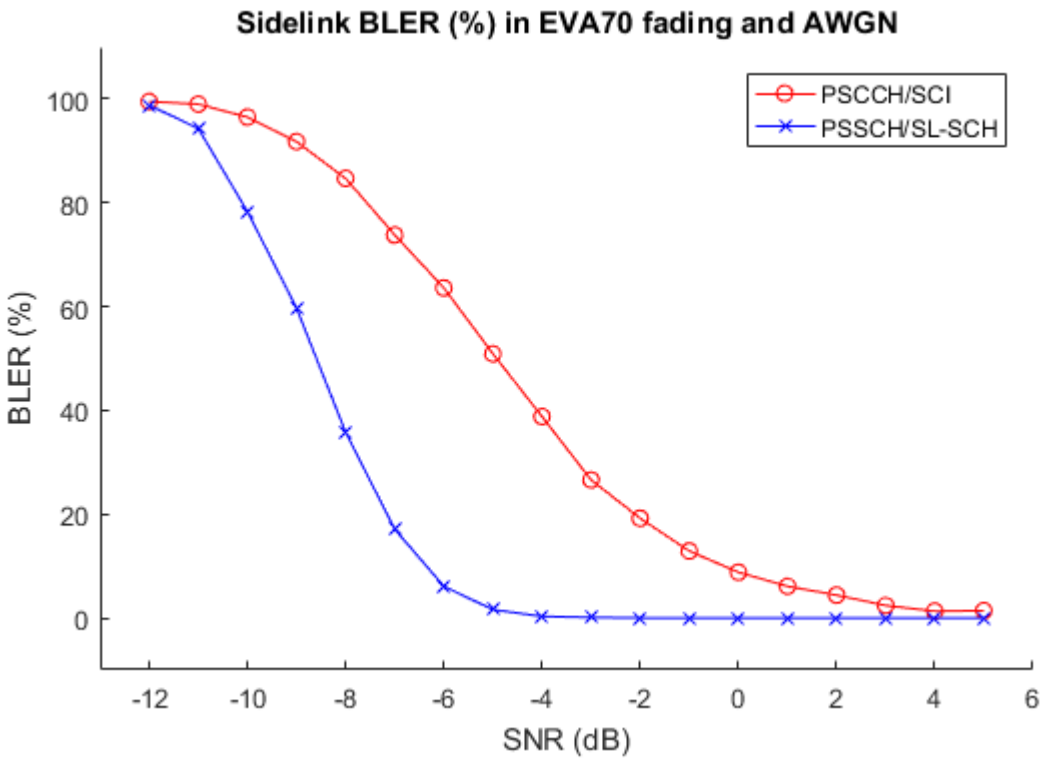
Finally the test results are plotted. The BLER for the control channel and shared channel are plotted for each SNR point. The average number of transmission instances that need to be combined for successful reception for each SNR are also plotted. Note that the number of transmission instances is 2 for the control channel and 4 for the shared channel.

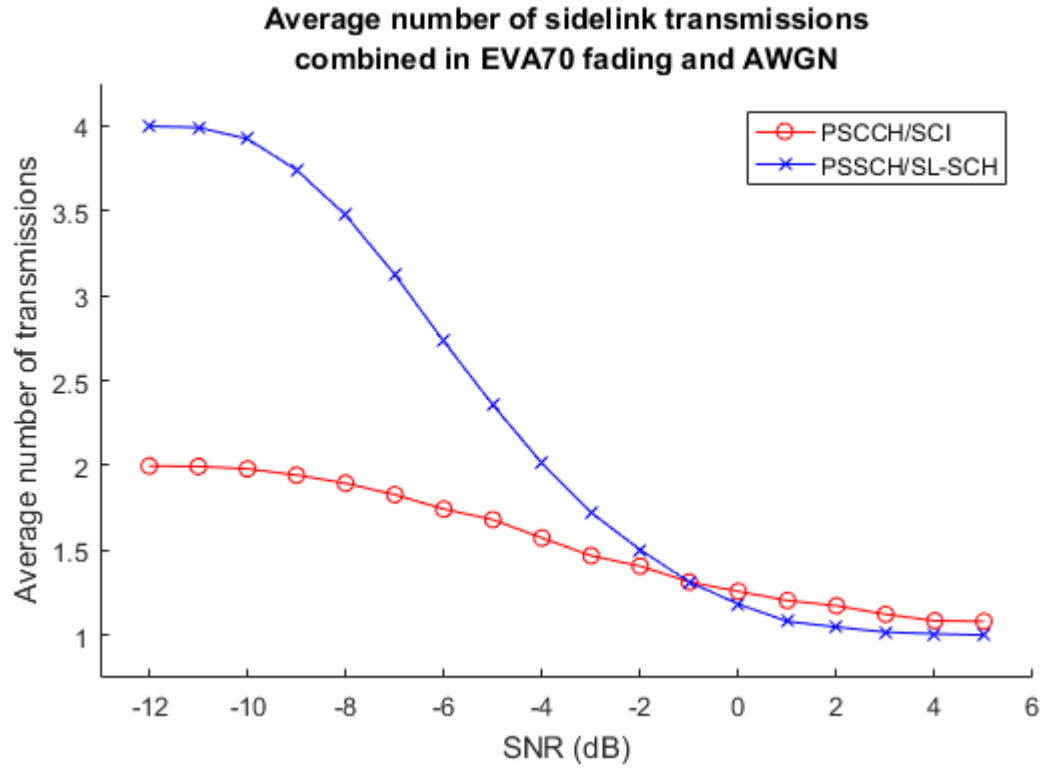
```
if (measureBLERForSCI || measureBLERForSLSCH)
    plotResults(channel,SNRIn,controlBLER,sharedBLER,controlRxsAvg,sharedRxsAvg);
end
```


Sidelink BLER (%) in EVA70 fading and AWGN
PSCCH/SCI
PSSCH/SL-SCH
BLER (%)
SNR (dB)
100
80
60
40
20
0
-10
-8
-6
-4
-2
0
2
4
6



Since the generated plots were obtained with a low number of PSCCH periods the results shown are not representative. A 1000 PSCCH period simulation with practical channel estimation including additional SNR points produced the results shown below.


Sidelink BLER (%) in EVA70 fading and AWGN
PSCCH/SCI
PSSCH/SL-SCH
BLER (%)
SNR (dB)
100
80
60
40
20
0
-12
-10
-8
-6
-4
-2
0
2
4
6


Average number of sidelink transmissions
combined in EVA70 fading and AWGN
PSCCH/SCI
PSSCH/SL-SCH
Average number of transmissions
SNR (dB)
4
3.5
3
2.5
2
1.5
1
-12
-10
-8
-6
-4
-2
0
2
4
6

Appendix

This example uses the helper function:

- hPerfectTimingEstimate.m

Selected Bibliography

1. 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception"
2. 3GPP TS 36.213 "Physical layer procedures"
3. 3GPP TS 36.331 "Radio Resource Control (RRC) Protocol specification"

Local Functions

The following local functions are used in this example:

- `perfectChannelEstimate`: perfect channel estimation
- `plotResults`: plot the results of the example

```
% Calculate the perfect channel estimate and the noise estimate
function [hest,nest] = perfectChannelEstimate(rxConfig,channel,noise,frameOffset)
    % Use the uplink perfect channel estimator to calculate
    % the sidelink estimate as both use SC-FDMA
    rxConfig.NTxAnts = 1;
    rxConfig.NULRB = rxConfig.NSLRB;
    rxConfig.CyclicPrefixUL = rxConfig.CyclicPrefixSL;
    rxConfig.TotSubframes = 1;
    hest = lteULPerfectChannelEstimate(rxConfig,channel,frameOffset);
    noiseGrid = lteSLSCFDMADemodulate(rxConfig,noise(1+frameOffset:end,:));
    nest = var(noiseGrid(:));
end

function plotResults(channel,SNRIn,controlBLER,sharedBLER,controlRxsAvg,sharedRxsAvg)

    fadingDescription = sprintf('%s%s',channel.DelayProfile,num2str(channel.DopplerFreq));

    figure;
    hold on;
    legends = {};
    if (~isempty(controlBLER))
        plot(SNRIn,controlBLER,'ro-');
        legends = [legends 'PSCCH/SCI'];
    end
    if (~isempty(sharedBLER))
        plot(SNRIn,sharedBLER,'bx-');
        legends = [legends 'PSSCH/SL-SCH'];
    end
    legend(legends);
    title(sprintf('Sidelink BLER (%%) in %s fading and AWGN',fadingDescription));
    ylabel('BLER (%)');
    xlabel('SNR (dB)');
    axis([SNRIn(1)-1 SNRIn(end)+1 -10 110]);
    grid on

    figure;
    hold on;
    if (~isempty(controlBLER))
```

```
            plot(SNRIn,controlRxsAvg,'ro-');
        end
        if (~isempty(sharedBLER))
            plot(SNRIn,sharedRxsAvg,'bx-');
        end
        legend(legends);
        title(sprintf('Average number of sidelink transmissions\ncombined in %s fading and AWGN',fadi
        ylabel('Average number of transmissions');
        xlabel('SNR (dB)');
        axis([SNRIn(1)-1 SNRIn(end)+1 0.75 4.25])
        grid on

    end
```

Release 14 V2X Sidelink PSCCH and PSSCH Throughput

This example demonstrates how to measure the physical sidelink shared channel (PSSCH) and physical sidelink control channel (PSCCH) throughput performance in frequency-selective fading and additive white Gaussian noise (AWGN).

Introduction

3GPP Release 14 introduced the support for LTE V2X (vehicle-to-everything) to enable connected vehicular services with the aim of providing a safer, cleaner, faster and more efficient transportation. V2X offers several modes of operation including vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I) and vehicle-to-pedestrian (V2P) direct communication without necessarily relying on network involvement for scheduling. Some of the distinguishing aspects of V2X over the Release 12 device-to-device sidelink are:

- Low latency and high reliability requirements
- Large Doppler shift due to high relative speeds
- Very large number of nodes and high node densities
- Challenges to synchronization especially when out-of-coverage

This example measures Release 14 V2X sidelink shared and control channel throughput for a number of SNR points. For information on how to model Release 12 sidelink device-to-device (D2D) interface (aimed primarily at allowing LTE to support public safety communication systems), see the “Release 12 Sidelink PSCCH and PSSCH Throughput” on page 2-241 example.

Operating on a subframe by subframe basis for each SNR point, the following steps are performed for the throughput and BLER calculation:

- A resource grid populated with PSCCH and/or PSSCH is generated and OFDM modulated to create the baseband waveform to transmit
- This waveform is passed through a noisy fading channel
- Receiver operations (SC-FDMA demodulation, channel estimation, and equalization) are performed
- The equalized symbols are decoded to obtain the block CRC
- The performance of the PSCCH and/or PSSCH is determined using the block CRC result at the output of the channel decoder

Simulation Configuration

By default, the example runs uses a simulation length of four frames over a range of SNR points. To produce meaningful throughput results, use a larger value of `NFrames`. You can specify `SNRIn` as an array of values or a scalar.

```
% Set the number of frames to simulate
NFrames = 4;
% Set a range of SNR points to cover both high and low BLER operating
% conditions
SNRIn = [-5 0 5];
```

Transmission Configuration

A set of top-level parameters is initially specified, these include the bandwidth, the cyclic prefix, the modulation scheme and the allocated set of resource blocks. The baseline configuration is taken from

the "Reference measurement channel for transmitter characteristics" as defined by TS 36.101 Table A.8.3-1 [2]. To simulate a more realistic V2X transmission, multiple HARQ processes and HARQ retransmissions have been introduced in this example. The data rate is defined in terms of the subframe gap `sfGap` and transmission time interval `tti`.

```
sfGap = 8; % Time in subframes between initial and retransmission
tti = 4;   % Time in subframes between different HARQ processes
if tti > sfGap
   error('tti cannot be greater than sfGap');
end

% Number of allocated resource blocks (NPRB) and Transport Block Size (TBS)
% as given by TS 36.101 Table A.8.3-1. Note that TBS must be chosen
% according to TS 36.213 section 14.1.1 and NPRB according to TS 36.213
% section 14.1.1.4C. If an invalid TBS/NPRB is specified, the transmitted
% ModCoding value in the SCI message would be incorrect and this could
% result in the SL-SCH decoding to fail. An invalid NPRB could also result
% in an SCI message and corresponding PSSCH transmission where the
% allocated resources are different from the NPRB specified here.
NPRB = 48;        % Number of allocated resource blocks (NPRB)
TBS = 3496;       % The transport block size
% Define the starting RB for the resource allocations. The initial and
% subsequent transmission can have different allocations
rbStart = [0 0];

% Define the UE configuration parameters
ueConfig = struct('SidelinkMode','V2X');% Release 14 V2X mode
ueConfig.NSLRB = 50;                    % 10MHz bandwidth
ueConfig.DuplexMode = 'FDD';            % Duplex mode
ueConfig.CyclicPrefixSL = 'Normal';     % Cyclic prefix length
ueConfig.Modulation = 'QPSK';           % Symbol modulation ('QPSK','16QAM')
```

Propagation Channel Configuration

The structure `channel` contains the channel model configuration parameters.

```
channel = struct;                    % Channel config structure
channel.Seed = 6;                    % Channel seed
channel.NRxAnts = 2;                 % Number of receive antennas
channel.DelayProfile ='EVA';         % Delay profile
channel.DopplerFreq = 500;           % Doppler frequency in Hz
channel.MIMOCorrelation = 'Low';     % Multi-antenna correlation
channel.NTerms = 16;                 % Oscillators used in fading model
channel.ModelType = 'GMEDS';         % Rayleigh fading model type
channel.InitPhase = 'Random';        % Random initial phases
channel.NormalizePathGains = 'On';   % Normalize delay profile power
channel.NormalizeTxAnts = 'On';      % Normalize for transmit antennas
% The channel sampling rate depends on the FFT size used in the OFDM
% modulator. This can be obtained using the function lteSLSCFDMAInfo.
ofdmInfo = lteSLSCFDMAInfo(ueConfig);
channel.SamplingRate = ofdmInfo.SamplingRate;
```

Channel Estimator Configuration

The variable `perfectChanEstimator` controls channel estimator behavior. Valid values are `true` or `false`. When set to `true`, a perfect channel response is used as estimate, otherwise an imperfect estimate based on the values of received DM-RS signals is obtained. If `perfectChanEstimator` is set to false a configuration structure `cec` is needed to parameterize the channel estimator.

```
% Perfect channel estimator flag
perfectChanEstimator = false;

% Define the practical channel estimator configuration structure. Note that
% depending on the channel delay profile and Doppler frequency, the
% operating SNR, PSSCH modulation order and the number of allocated
% resource blocks, different channel estimation parameters may give better
% performance.
cec = struct;                       % Channel estimation config structure
cec.PilotAverage = 'UserDefined';   % Type of pilot symbol averaging
cec.FreqWindow = 27;                % Frequency window size
cec.TimeWindow = 1;                 % Time window size
cec.InterpType = 'Cubic';           % 2D interpolation type
```

Configure Throughput Measurements

The logical variables `measureBLERForSCI` and `measureTputForSLSCH` allow the throughput measurements and all related receiver processing to be disabled for the SCI and SL-SCH respectively. This allows the simulation to be configured to measure throughput for only one of the channels. Disabling the receiver processing for the other channel improves the execution speed.

```
measureBLERForSCI = true;
measureTputForSLSCH = true;
```

Select Receiver Behavior When SCI Decoding Fails

The logical variable `sciAssumed` controls the simulation behavior in terms of the effect the control channel decoding has on the shared channel decoding. If `sciAssumed` is true, the receiver will assume that the SCI message was correctly decoded when attempting shared channel reception. This allows the performance of the shared channel reception to be measured independently from the performance of the control channel reception. If `sciAssumed` is false, a control channel decoding failure implies a shared channel decoding failure. Note that if the throughput measurement for the SCI is disabled above (`measureBLERForSCI=false`), and the throughput measurement for the SL-SCH is enabled (`measureTputForSLSCH=true`), the receiver will assume that the SCI message was correctly decoded when measuring SL-SCH throughput regardless of the setting of `sciAssumed` here.

```
sciAssumed = false;
```

Display Simulation Information

The variable `displaySimulationInformation` controls the display of simulation information such as the HARQ process ID used and the resource allocation plot for each subframe. For long simulations, it is recommended to turn off the display as it increases simulation time.

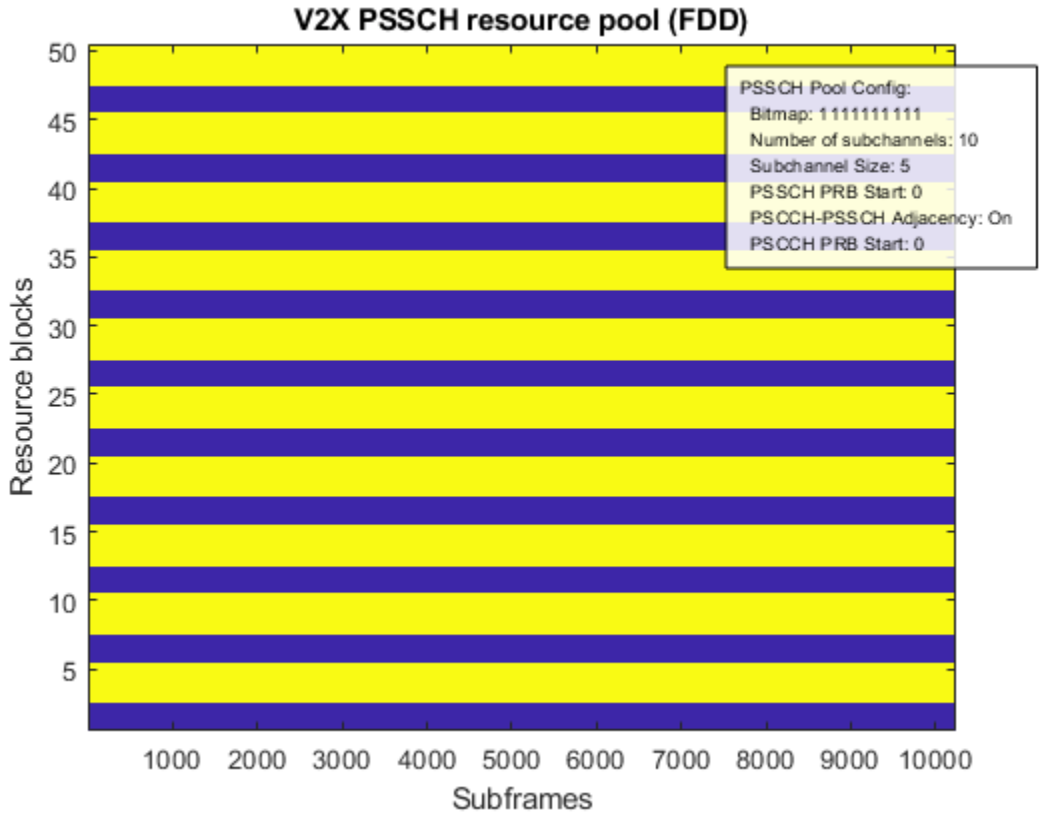
```
displaySimulationInformation = true;
```

Display V2X Sidelink Communication Resource Pool

Transmission and reception opportunities for sidelink direct communications are associated with a set of periodically occurring time-domain periods known as resource pools. A UE can be configured with multiple messages and therefore multiple resource pools for transmission and reception. A single resource pool contains both the shared and control subframe and resource pools and a specific transmission is defined by a combination of semi-static RRC parameters with dynamic DCI/SCI parameters. The PSCCH associated with a PSSCH is sent in the same subframe, on either adjacent or non-adjacent PRB. In order to illustrate the structure of V2X resource pools, the example uses the

class `V2XSidelinkResourcePool`, which models a resource pool and provides parameters and methods to define a specific transmission. The method `V2XSidelinkResourcePool.displayPeriod` creates a plot showing the locations of the control resource pool (dark blue) and shared resource pool (yellow) within the resource pool.

```
resourcePool = V2XSidelinkResourcePool;
resourcePool.Config.NSLRB = ueConfig.NSLRB;
resourcePool.Config.DuplexMode = ueConfig.DuplexMode;
if displaySimulationInformation
    resourcePool.displayPeriod;
    drawnow;
end
```



Processing Chain

To determine the throughput at each SNR point, the subframe by subframe processing chain includes:

- *Update configuration for current HARQ process* - The UE either carries new transport data or a retransmission of previously sent PSSCH transport data with a different redundancy version. All this is handled by the HARQ scheduler. The HARQ buffer `decState` stores the decoder state for soft combining.
- *Create and encode the SCI message* - The SCI message carries sidelink scheduling information. The SCI parameters are: `Priority` indicating ProSe Per-Packet Priority, `ResourceReservation` set by higher layers for reserving the resource for the next transmission of the transport block, `RIV` carrying allocation information, `TimeGap` signaling the time gap in subframes between the

initial transmission and retransmission, `ModCoding` signaling the modulation scheme are used for PSSCH reception and `RetransmissionIdx` indicating whether the transmission is the first or second (retransmission). The SCI message is then encoded and mapped on to the PSCCH for transmission. The CRC of the SCI message is the scrambling identity for PSSCH.

- *Plot the allocated resources* - The resource allocation for the first HARQ process is plotted if the `displaySimulationInformation` flag is enabled. All HARQ processes have the same PSSCH (orange) and PSCCH (green) resource block allocation. The PSCCH and PSSCH can be transmitted on adjacent or non-adjacent resource blocks.

- *Create Transmit Waveform* - Pass the data generated by the UE to the physical layer processing stage to produce an SC-FDMA modulated waveform, containing the PSCCH and PSSCH physical channels and DRS signals.

- *Noisy Channel Modeling* - Pass the waveform through a fading channel and add noise (AWGN).

- *Perform Blind Detection of PSCCH* - The PSCCH resources and cyclic shift used in the transmitter in the subframe is unknown, so the synchronization, channel estimation and decoding is performed for each PRB set in the PSCCH resource block pool (provided by `V2XSidelinkResourcePool.PSCCHResourceBlockPool` for all cyclic shifts until the SCI is successfully decoded or the entire resource pool is searched. The timing offset for the DM-RS correlation with the strongest peak is used for synchronization.

- *Perform PSCCH channel estimation*: PSCCH channel estimation is performed using `lteSLChannelEstimatePSCCH`. The channel estimator also produces an estimate of the noise power which can be used for MMSE equalization.

- *Extract the PSCCH symbols and channel estimate*: The received PSCCH symbols are extracted from the subframe resource grid, the corresponding channel estimates are extracted using `lteExtractResources` and the indices provided by `ltePSCCHIndices`.

- *Perform PSCCH equalization*: The PSCCH symbols are MMSE equalized using `lteEqualizeMMSE` with the channel estimate and noise estimate obtained above.

- *Perform PSCCH demodulation*: The equalized PSCCH symbols are demodulated using `ltePSCCHDecode`. This function performs the inverse of the transmitter modulation steps (SC-FDMA transform deprecoding, QPSK symbol demodulation and descrambling).

- *Perform SCI decoding*: SCI decoding is attempted using `lteSCIDecode`. The number of original information bits in the SCI message is given by `lteSCIInfo`. If the decoded CRC is zero and the recovered CRC mask is the expected value, the SCI decoding is considered successful and the decoded message bits are converted into the corresponding message structure using `lteSCI`. The CRC mask value provides the PSSCH scrambling identity, `NXID`.

The behavior of the shared channel decoding with respect to a failed SCI decoding is controlled by the variable `sciAssumed`. If `sciAssumed` is false, the failed SCI decoding immediately implies a failure to decode the SL-SCH and no further processing takes place for the current subframe. If `sciAssumed` is true, the transmitted SCI message is assumed to be known to the receiver and will be used in place of the received SCI message. If `sciAssumed` is true or if SCI decoding was successful, the receiver proceeds with SL-SCH decoding. Note that if SCI BLER measurement is disabled but SL-SCH throughput measurement is enabled then the transmitted SCI message is assumed to be known to the receiver.

For SL-SCH decoding:

- *Get the PSSCH resource allocation* - The RIV obtained from the recovered SCI message is then decoded to get the number of subchannels and the starting sub channel allocated for this transmission. This in combination with the PSCCH adjacency, size of each subchannel and first allocated resource block provides the set of allocated PSSCH RBs `PRBSet`.
- *Synchronize and SC-FDMA demodulate subframe carrying PSSCH*: The appropriate subframe of the waveform is synchronized using `lteSLFrameOffsetPSSCH`. The synchronized waveform is SC-FDMA demodulated using `lteSLSCFDMADemodulate`. Note that although the control and shared channel are transmitted in the same subframe, it is sufficient to perform the synchronization and SC-FDMA demodulation only for the PSCCH. However, in this example, since the shared and control channel can be independently received, PSSCH synchronization and SC-FDMA demodulation are performed. This is to enable PSSCH reception when the PSCCH reception is disabled.
- *Perform PSSCH channel estimation*: PSSCH channel estimation is performed using `lteSLChannelEstimatePSSCH`. The channel estimator also produces an estimate of the noise power which can be used for MMSE equalization.
- *Extract the PSSCH symbols and channel estimate*: The received PSSCH symbols are extracted from the subframe resource grid and the corresponding channel estimates are extracted using `lteExtractResources` and the indices provided by `ltePSSCHIndices`.
- *Perform PSSCH equalization*: The PSSCH symbols are MMSE equalized using `lteEqualizeMMSE` with the channel estimate and noise estimate obtained above.
- *Perform PSSCH demodulation*: The equalized PSSCH symbols are demodulated using `ltePSSCHDecode`. This function performs the inverse of the transmitter modulation steps (SC-FDMA transform deprecoding, QPSK or 16QAM symbol demodulation and descrambling). The descrambling uses the PSCCH CRC obtained from the SCI decoding stage.
- *Decoding the Sidelink Shared Channel (SL-SCH) and Storing the Block CRC Error for a UE* - Pass the vector of decoded soft bits to `lteSLSCHDecode`, which decodes the codeword and returns the block CRC error used to determine the throughput of the system. The contents of the new soft buffer, `harqProcesses(harqID).decState`, is available at the output of this function to be used when decoding the next subframe.

```
% Initialize variables used in the simulation and analysis
maxThroughputSLSCH = zeros(length(SNRIn),1);
simThroughputSLSCH = zeros(length(SNRIn),1);
simBLERSCI = zeros(length(SNRIn),1);

% Get the number of HARQ processes required
nHARQProcesses = floor(sfGap/tti);
% Calculate the HARQ ID sequence for each of the transmitting subframes
harqProcessSequence = zeros(1,NFrames*10);
for h = 1:nHARQProcesses
    harqProcessSequence(1+tti*(h-1):sfGap:NFrames*10) = h;
end

% Create the partial SCI message with the parameters that have fixed values
% for the simulation. Other parameters that have variable values will be
% set further down in the simulation loop
sciMessage = struct('SCIFormat','Format1');
sciMessage.TimeGap = sfGap;
% Set the PSSCH MCS according to the TBS and modulation scheme
[itbs,modn] = lteMCS(0:28,'PUSCH');
```

```
% According to TS 36.213 Section 14.1.1, for IMCS (0 to 28), the modulation
% order is set to Q' = min(4,Qm') where Qm' can be 2, 4 or 6. So change the
% cases where Qm' = 6 ('64QAM') to Qm' = 4 ('16QAM')
modn(strcmpi(modn, '64QAM')) = {'16QAM'};
% Filter valid transport block sizes according to modulation scheme used
possibleTBS = lteTBS(NPRB,itbs);
possibleTBS(~strcmpi(modn,ueConfig.Modulation)) = 0;
% Now set the MCS index according to the TBS
sciMessage.ModCoding = find(possibleTBS==TBS,1) - 1;

% Create new figure for displaying resource allocation, if required
if displaySimulationInformation
    figure;
end

% If SCI BLER is not being measured, 'sciAssumed' must be set so that the
% receiver can assume knowledge of the SCI
if (~measureBLERForSCI)
    sciAssumed = true;
end

rng('default');
for snrIdx = 1:numel(SNRIn)

    % Set the random number generator seed depending on the loop variable
    % to ensure independent random streams
    rng(snrIdx);

    % Initialize the state of all HARQ buffers
    harqProcess = struct('RVIdx',1,'data',[],'decState',[]);
    harqProcesses(1:nHARQProcesses) = harqProcess;
    RVSeq = [0 2];    % RV for initial transmission and retransmission

    % Set up variables for the main loop
    lastOffset = 0;       % Initialize overall frame timing offset
    frameOffset = 0;      % Initialize frame timing offset
    sharedbitTput = [];   % Number of successfully received bits per subframe
    txedTrBlkSizes = [];  % Number of transmitted bits per subframe
    controlErrors = [];   % Number of SCI block errors

    for subframeNo = 0:(NFrames*10-1)
        harqID = harqProcessSequence(subframeNo+1);
        if  harqID == 0
            % If there is no HARQ process transmitting in the current
            % subframe, continue to the next
            continue
        end

        % Update current HARQ process with new transport data and reset the
        % receive buffer if it is the initial transmission. The PSSCH is
        % transmitted twice with a gap of 'sfGap' subframes irrespective of
        % the success of the first transmission
        if harqProcesses(harqID).RVIdx == 1
            harqProcesses(harqID).data = randi([0 1], TBS, 1);
            harqProcesses(harqID).decState = [];
        end
        % Set the RV
        ueConfig.RV = RVSeq(harqProcesses(harqID).RVIdx);
```

```
% Set the subframe number
ueConfig.NSubframePSSCH = subframeNo;

% Display run time information
if displaySimulationInformation
    fprintf('Subframe: %d. HARQ process index: %d. Redundancy version: %d\n',subframeNo,h
end

% Channel time for the current subframe
channel.InitTime = subframeNo/1000;

% create an empty resource grid
slgrid = lteSLResourceGrid(ueConfig);

% Update the transmission config with the starting RB
startingRB = rbStart(harqProcesses(harqID).RVIdx);
resourcePool.Config.startRB_Subchannel_r14 = startingRB;

% Calculate the RIV from the number of allocated subchannels and
% the starting subchannel
ueConfig.FirstSubchannelIdx = 0;
beta = 0;
if strcmpi(resourcePool.Config.adjacencyPSCCH_PSSCH_r14,'On')
    beta = 2;
end
lsubCH = ceil((NPRB+beta)/resourcePool.Config.sizeSubchannel_r14);
sciMessage.RIV = resourcePool.encodeRIV(lsubCH,ueConfig.FirstSubchannelIdx);

% Set the retransmission index
sciMessage.RetransmissionIdx = harqProcesses(harqID).RVIdx - 1;

% Create the SCI and message bits
sciConfig = ueConfig;
sciConfig.PSSCHNSubchannels = resourcePool.Config.numSubchannel_r14;
[sciMessage, sciBits] = lteSCI(sciConfig,sciMessage);
% Perform SCI encoding
[cw,crc] = lteSCIEncode(sciConfig, sciBits);
% NXID is the 16 bit CRC associated with the PSCCH SCI grant and is
% used as the PSSCH scrambling identity
ueConfig.NXID = crc;
% Store NXID value in 'expectedNXID' so that the receiver won't
% have to access 'sciMessage' at all
expectedNXID = ueConfig.NXID;

% Display the subframes and resource blocks for transmission. Note
% that the allocations are the same for all HARQ processes and that
% the plot does not show the subframe offset for HARQ processes
% other than 1
if (displaySimulationInformation) && (harqID==1)
    % Specify the additional parameters required for the resource
    % pool display
    sciMessage.FirstSubchannelIdx = ueConfig.FirstSubchannelIdx;
    sciMessage.SLIndex = 0;
    resourcePool.displayPeriod(sciMessage,10);
    drawnow;
end

% Create the PSCCH codeword
```

```
pscchSymbols = ltePSCCH(cw);
% Get the resource allocation for the PSCCH
[pscchsf,pscchrb] = resourcePool.getPSCCHResources(setfield(sciMessage,'FirstSubchannelI
% Select the resource for this transmission
sciConfig.PRBSet = pscchrb(:,harqProcesses(harqID).RVIdx);
% Get the PSCCH symbol indices
pscchIndices = ltePSCCHIndices(sciConfig);
% Map the PSCCH symbols on to the grid
slgrid(pscchIndices) = pscchSymbols;

% Define a random cyclic shift from the set {0,3,6,9} for each
% PSCCH transmission
sciConfig.CyclicShift = randi([0 3])*3;
% Create the DM-RS symbols and indices
pscchdrssymbols = ltePSCCHDRS(sciConfig);
pscchdrsindices = ltePSCCHDRSIndices(sciConfig);
% Map the PSCCH DRS symbols on to the grid
slgrid(pscchdrsindices) = pscchdrssymbols;

% Get the resource allocations for the two transmissions
[psschsf,psschrb] = resourcePool.getPSSCHResources(setfield(sciMessage,'FirstSubchannelI
% Now select the resource for this transmission
ueConfig.PRBSet = psschrb(:,harqProcesses(harqID).RVIdx);
% Calculate the PSSCH resource indices
[psschIndices,psschinfo] = ltePSSCHIndices(ueConfig);

% SL-SCH and PSSCH
cw = lteSLSCH(ueConfig,psschinfo.G,harqProcesses(harqID).data);
psschSymbols = ltePSSCH(ueConfig,cw);
slgrid(psschIndices) = psschSymbols;

% PSSCH DRS and DRS indices
psschdrssymbols = ltePSSCHDRS(ueConfig);
psschdrsindices = ltePSSCHDRSIndices(ueConfig);
slgrid(psschdrsindices) = psschdrssymbols;

% The SC-FDMA modulation will not use windowing due to the
% piece-wise nature of the waveform construction
windowing = 0;
% perform sidelink SC-FDMA modulation
[txWaveform,scfdmaInfo] = lteSLSCFDMAModulate(ueConfig,slgrid,windowing);

% Add 25 sample padding. This is to cover the range of delays
% expected from channel modeling (a combination of
% implementation delay and channel delay spread)
ntxants = size(txWaveform,2);
txWaveform = [txWaveform; zeros(25, ntxants)]; %#ok<AGROW>

% Pass data through channel model
rxNoiselessWaveform = lteFadingChannel(channel,txWaveform);

% Calculate noise gain
SNR = 10^((SNRIn(snrIdx))/20);

% Normalize noise power to take account of sampling rate,
% which is a function of the IFFT size used in OFDM
% modulation, and the number of antennas
N0 = 1/(sqrt(2.0*ntxants*double(ofdmInfo.Nfft))*SNR);
```

```
% Create AWGN
noise = N0*complex(randn(size(rxNoiselessWaveform)),...
    randn(size(rxNoiselessWaveform)));

% Add AWGN to the received time domain waveform
rxWaveform = rxNoiselessWaveform + noise;

%-----------------------------------------------------------------
%            Receiver
%-----------------------------------------------------------------

% Define the receiver top-level configuration
rxConfig = [];
rxConfig.SidelinkMode = ueConfig.SidelinkMode;
rxConfig.CyclicPrefixSL = ueConfig.CyclicPrefixSL;
rxConfig.FirstSubchannelIdx = ueConfig.FirstSubchannelIdx;
rxConfig.NSLRB = resourcePool.Config.NSLRB;
rxConfig.DuplexMode = resourcePool.Config.DuplexMode;
rxConfig.TDDConfig = resourcePool.Config.TDDConfig;
rxConfig.PSSCHNSubchannels = resourcePool.Config.numSubchannel_r14;
rxConfig.PSSCHSubchannelSize = resourcePool.Config.sizeSubchannel_r14;
rxConfig.PSSCHAdjacency = resourcePool.Config.adjacencyPSCCH_PSSCH_r14;
rxConfig.PSSCHSubchannelPRBStart = resourcePool.Config.startRB_Subchannel_r14;
rxConfig.NSubframePSSCH = subframeNo; % Set the subframe number

% If SCI BLER measurement is configured:
sciDecoded = false;
if measureBLERForSCI

    % Blind decoding of the control channel
    % We know that the current subframe contains control and data,
    % but do not know the location. Check for the SCI message in
    % the resource pool for all cyclic shifts till it is found or
    % the whole resource pool is searched
    pscchPool = resourcePool.PSCCHResourceBlockPool;
    % Re-arrange the resources into the possible PRB pairs
    pscchPool = reshape(pscchPool,2,numel(pscchPool)/2);

    p = 1;
    while p <= size(pscchPool,2) && ~sciDecoded

        % PSCCH DRS for V2X is transmitted on two consecutive RBs
        rxConfig.PRBSet = pscchPool(:,p);
        for cs = [0 3 6 9]
            rxConfig.CyclicShift = cs;
            [frameOffset,corr] = lteSLFrameOffsetPSCCH(rxConfig,rxWaveform);

            % Perform subframe synchronization
            if (frameOffset > 25) || (frameOffset < 0)
                frameOffset = lastOffset;
            end
            lastOffset = frameOffset;

            % Perform SC-FDMA demodulation on the received data to obtain
            % the resource grid
            rxSubframe = lteSLSCFDMADemodulate(rxConfig,rxWaveform(1+frameOffset:end,:));
```

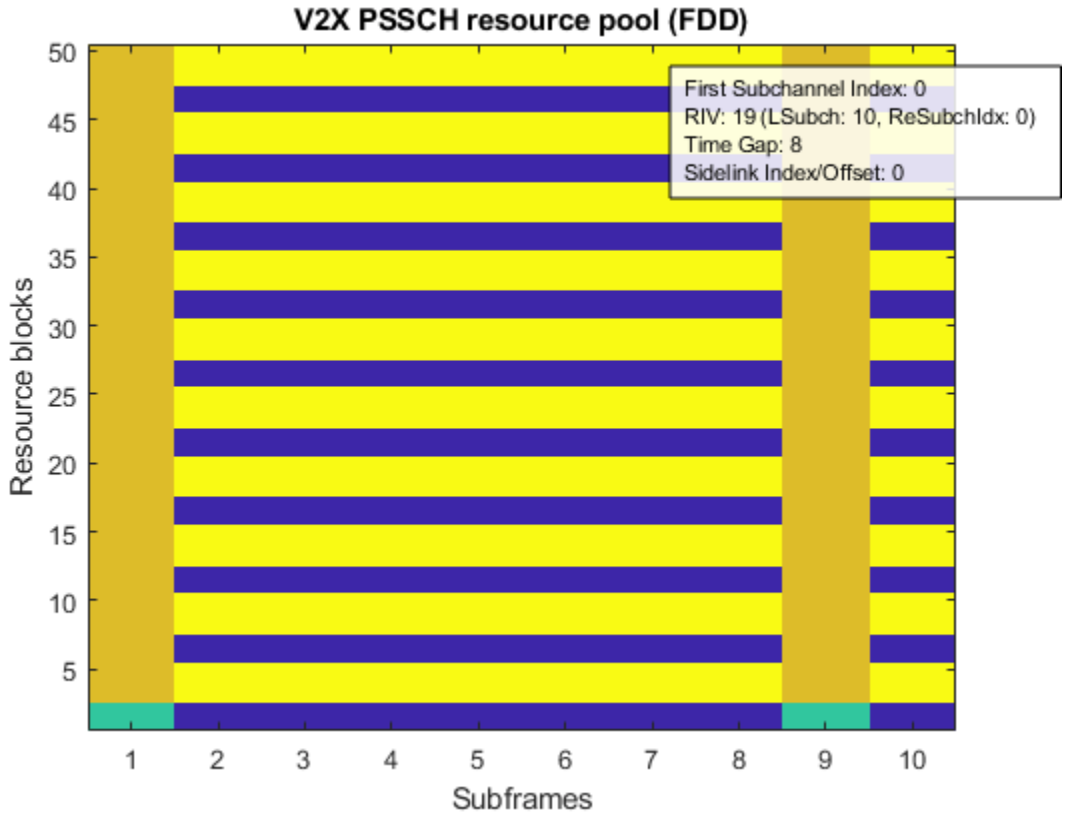
```
                % Perform channel estimation
                if(perfectChanEstimator)
                    [hest,nest] = perfectChannelEstimate(rxConfig,channel,noise,frameOffset);
                else
                    [hest,nest] = lteSLChannelEstimatePSCCH(rxConfig,cec,rxSubframe);
                end
                % Get the PSCCH candidate for the current PRBSet,
                % extract the corresponding received symbols and
                % channel estimate and perform equalization
                pscchCandidate = ltePSCCHIndices(rxConfig);
                [pscchRx,pscchHest] = lteExtractResources(pscchCandidate,rxSubframe,hest);
                pscchSymbols = lteEqualizeMMSE(pscchRx,pscchHest,nest);

                % Demodulate the PSCCH
                codedsciBits = ltePSCCHDecode(pscchSymbols);

                % Get the payload size for the Release 14 V2X SCI message
                scilengthinfo = lteSCIInfo(rxConfig);
                % Decode the SCI and recover the message
                [rxinfo,rxSCIErr,rxcrc] = lteSCIDecode(scilengthinfo.Format1,codedsciBits);
                if  ~rxSCIErr && (rxcrc==expectedNXID)
                    sciMessageRx = lteSCI(rxConfig,rxinfo);
                    rxConfig.NXID = rxcrc;
                    sciDecoded = true;
                    if (displaySimulationInformation)
                        fprintf('   SCI decoded\n');
                    end
                    break;
                end

            end
            % Increment the pool index
            p = p+1;
        end
        if ~sciDecoded
             if (displaySimulationInformation)
                fprintf('   SCI decoding failed\n');
             end
        end

        % Store values needed to calculate BLER
        controlErrors = [controlErrors ~sciDecoded];  %#ok<AGROW>
    end

    % If SCI decoding failed and the SCI is assumed for SL-SCH
    % decoding, set the decoded SCI message equal to the transmitted
    % SCI message. Also set the NXID to the expected value
    if (~sciDecoded) && (sciAssumed)
        sciMessageRx = sciMessage;
        rxConfig.NXID = expectedNXID;
    end

    % If SL-SCH throughput measurement is configured and if the SCI was
    % successfully decoded or if SCI decoding success is assumed,
    % perform PSSCH reception and SL-SCH decoding
    if measureTputForSLSCH
        if (sciDecoded || sciAssumed)
```

```
                % Get the PSSCH resource allocation from the decoded SCI
                % message and pool configuration
                rxConfig.PRBSet = lteSCIResourceAllocation(rxConfig,sciMessageRx);

                % Perform subframe synchronization
                frameOffset = lteSLFrameOffsetPSSCH(rxConfig,rxWaveform);
                if (frameOffset > 25) || (frameOffset < 0)
                    frameOffset = lastOffset;
                end
                lastOffset = frameOffset;

                % Perform SC-FDMA demodulation on the received data to obtain
                % the resource grid
                rxSubframe = lteSLSCFDMADemodulate(rxConfig,rxWaveform(1+frameOffset:end,:));

                % Perform channel estimation, extract the received PSSCH
                % symbols and the corresponding channel estimate, and
                % perform equalization
                if(perfectChanEstimator)
                    [hest,nest] = perfectChannelEstimate(rxConfig,channel,noise,frameOffset);
                else
                    [hest,nest] = lteSLChannelEstimatePSSCH(rxConfig,cec,rxSubframe);
                end
                [psschRx,psschHest] = lteExtractResources(psschIndices,rxSubframe,hest);
                psschSymbols = lteEqualizeMMSE(psschRx,psschHest,nest);

                % Demodulate the PSSCH
                [~,rxConfig.Modulation] = lteMCS(sciMessageRx.ModCoding,'PUSCH');
                if (strcmpi(rxConfig.Modulation,'64QAM'))
                    rxConfig.Modulation = '16QAM';
                end
                codedSlschBits = ltePSSCHDecode(rxConfig,psschSymbols);

                % Decode the SL-SCH, including soft combining into the
                % receiver buffer and check the CRC
                rxConfig.NTurboDecIts = 5;  % Turbo decoder iterations
                rxConfig.RV = RVSeq(harqProcesses(harqID).RVIdx);
                [~,slschCRC,harqProcesses(harqID).decState] = ...
                        lteSLSCHDecode(rxConfig,TBS,codedSlschBits, ...
                        harqProcesses(harqID).decState);
                slschDecoded = ~slschCRC;

            else
                % If SCI decoding failed, SL-SCH decoding also failed
                slschDecoded = false;
            end
            % Store values needed to calculate throughput
            sharedbitTput = [sharedbitTput TBS*slschDecoded]; %#ok<AGROW>
            txedTrBlkSizes = [txedTrBlkSizes TBS]; %#ok<AGROW>
            if (displaySimulationInformation)
                if  slschDecoded
                    fprintf('   SL-SCH decoded\n');
                else
                    fprintf('   SL-SCH decoding failed\n');
                end
            end

        end
```

```
            % Update the RV sequence index for the next transmission
            harqProcesses(harqID).RVIdx = mod(harqProcesses(harqID).RVIdx,size(RVSeq,2))+1;

        end

        if measureBLERForSCI

            % Calculate the SCI BLER
            simBLERSCI(snrIdx) = 100*mean(controlErrors,2);

            % Display the results dynamically in the command window
            fprintf('\nSNR = %.2f dB. SCI BLER(%%) for %d Frame(s) = %.4f %%\n',...
            SNRIn(snrIdx),NFrames,simBLERSCI(snrIdx));
        end

        if measureTputForSLSCH

            % Calculate the maximum and simulated throughput
            maxThroughputSLSCH(snrIdx) = sum(txedTrBlkSizes); % Max possible throughput
            simThroughputSLSCH(snrIdx) = sum(sharedbitTput);  % Simulated throughput

            % Display the results dynamically in the command window
            fprintf('\nSNR = %.2f dB. SL-SCH Throughput for %d Frame(s) = %.4f Mbps\n',...
                 SNRIn(snrIdx),NFrames,1e-6*simThroughputSLSCH(snrIdx)/(NFrames*10e-3));
            fprintf('SNR = %.2f dB. SL-SCH Throughput(%%) for %d Frame(s) = %.4f %%\n',...
            SNRIn(snrIdx),NFrames,simThroughputSLSCH(snrIdx)*100/maxThroughputSLSCH(snrIdx));
        end

    end

Subframe: 0. HARQ process index: 1. Redundancy version: 0
   SCI decoding failed
   SL-SCH decoding failed
Subframe: 4. HARQ process index: 2. Redundancy version: 0
   SCI decoded
   SL-SCH decoding failed
Subframe: 8. HARQ process index: 1. Redundancy version: 2
   SCI decoded
   SL-SCH decoding failed
Subframe: 12. HARQ process index: 2. Redundancy version: 2
   SCI decoded
   SL-SCH decoded
Subframe: 16. HARQ process index: 1. Redundancy version: 0
   SCI decoded
   SL-SCH decoding failed
Subframe: 20. HARQ process index: 2. Redundancy version: 0
   SCI decoded
   SL-SCH decoding failed
Subframe: 24. HARQ process index: 1. Redundancy version: 2
   SCI decoded
   SL-SCH decoded
Subframe: 28. HARQ process index: 2. Redundancy version: 2
   SCI decoded
   SL-SCH decoded
Subframe: 32. HARQ process index: 1. Redundancy version: 0
   SCI decoded
   SL-SCH decoding failed
```

```
Subframe: 36. HARQ process index: 2. Redundancy version: 0
   SCI decoded
   SL-SCH decoding failed

SNR = -5.00 dB. SCI BLER(%) for 4 Frame(s) = 10.0000 %

SNR = -5.00 dB. SL-SCH Throughput for 4 Frame(s) = 0.2622 Mbps
SNR = -5.00 dB. SL-SCH Throughput(%) for 4 Frame(s) = 30.0000 %
Subframe: 0. HARQ process index: 1. Redundancy version: 0
   SCI decoded
   SL-SCH decoded
Subframe: 4. HARQ process index: 2. Redundancy version: 0
   SCI decoded
   SL-SCH decoded
Subframe: 8. HARQ process index: 1. Redundancy version: 2
   SCI decoded
   SL-SCH decoded
Subframe: 12. HARQ process index: 2. Redundancy version: 2
   SCI decoded
   SL-SCH decoded
Subframe: 16. HARQ process index: 1. Redundancy version: 0
   SCI decoded
   SL-SCH decoded
Subframe: 20. HARQ process index: 2. Redundancy version: 0
   SCI decoded
   SL-SCH decoded
Subframe: 24. HARQ process index: 1. Redundancy version: 2
   SCI decoded
   SL-SCH decoded
Subframe: 28. HARQ process index: 2. Redundancy version: 2
   SCI decoded
   SL-SCH decoded
Subframe: 32. HARQ process index: 1. Redundancy version: 0
   SCI decoded
   SL-SCH decoded
Subframe: 36. HARQ process index: 2. Redundancy version: 0
   SCI decoded
   SL-SCH decoded

SNR = 0.00 dB. SCI BLER(%) for 4 Frame(s) = 0.0000 %

SNR = 0.00 dB. SL-SCH Throughput for 4 Frame(s) = 0.8740 Mbps
SNR = 0.00 dB. SL-SCH Throughput(%) for 4 Frame(s) = 100.0000 %
Subframe: 0. HARQ process index: 1. Redundancy version: 0
   SCI decoded
   SL-SCH decoded
Subframe: 4. HARQ process index: 2. Redundancy version: 0
   SCI decoded
   SL-SCH decoded
Subframe: 8. HARQ process index: 1. Redundancy version: 2
   SCI decoded
   SL-SCH decoded
Subframe: 12. HARQ process index: 2. Redundancy version: 2
   SCI decoded
   SL-SCH decoded
Subframe: 16. HARQ process index: 1. Redundancy version: 0
   SCI decoded
   SL-SCH decoded
```

```
Subframe: 20. HARQ process index: 2. Redundancy version: 0
   SCI decoded
   SL-SCH decoded
Subframe: 24. HARQ process index: 1. Redundancy version: 2
   SCI decoded
   SL-SCH decoded
Subframe: 28. HARQ process index: 2. Redundancy version: 2
   SCI decoded
   SL-SCH decoded
Subframe: 32. HARQ process index: 1. Redundancy version: 0
   SCI decoded
   SL-SCH decoded
Subframe: 36. HARQ process index: 2. Redundancy version: 0
   SCI decoded
   SL-SCH decoded

SNR = 5.00 dB. SCI BLER(%) for 4 Frame(s) = 0.0000 %

SNR = 5.00 dB. SL-SCH Throughput for 4 Frame(s) = 0.8740 Mbps
SNR = 5.00 dB. SL-SCH Throughput(%) for 4 Frame(s) = 100.0000 %
```

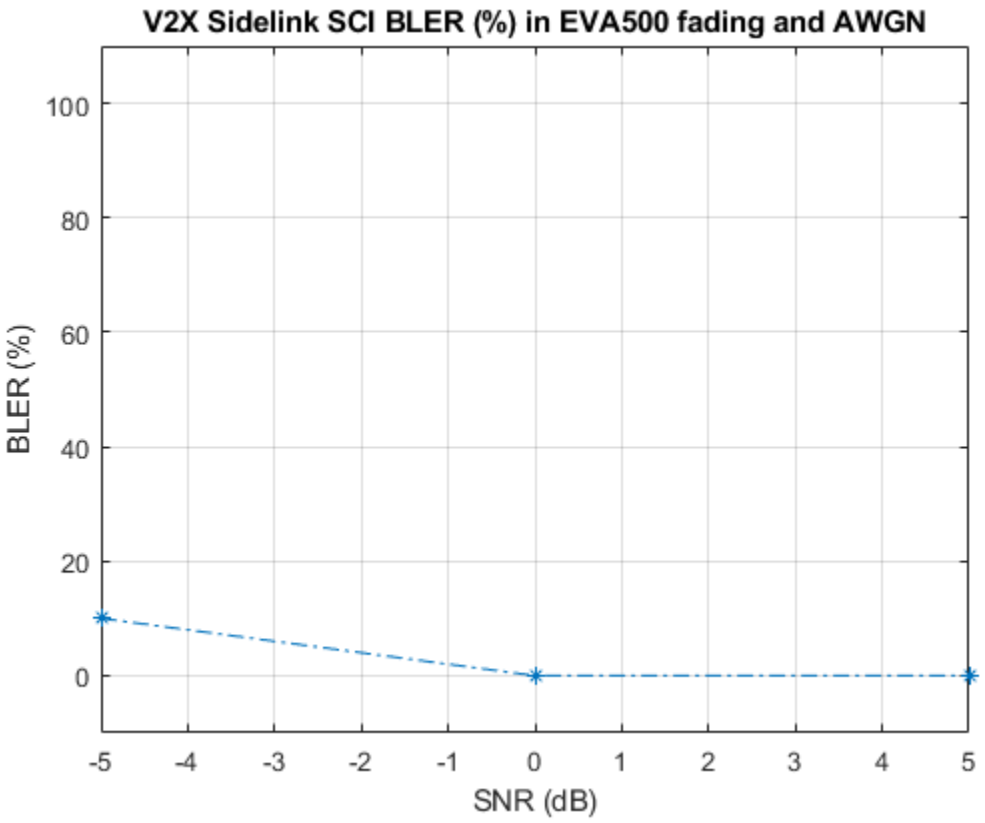


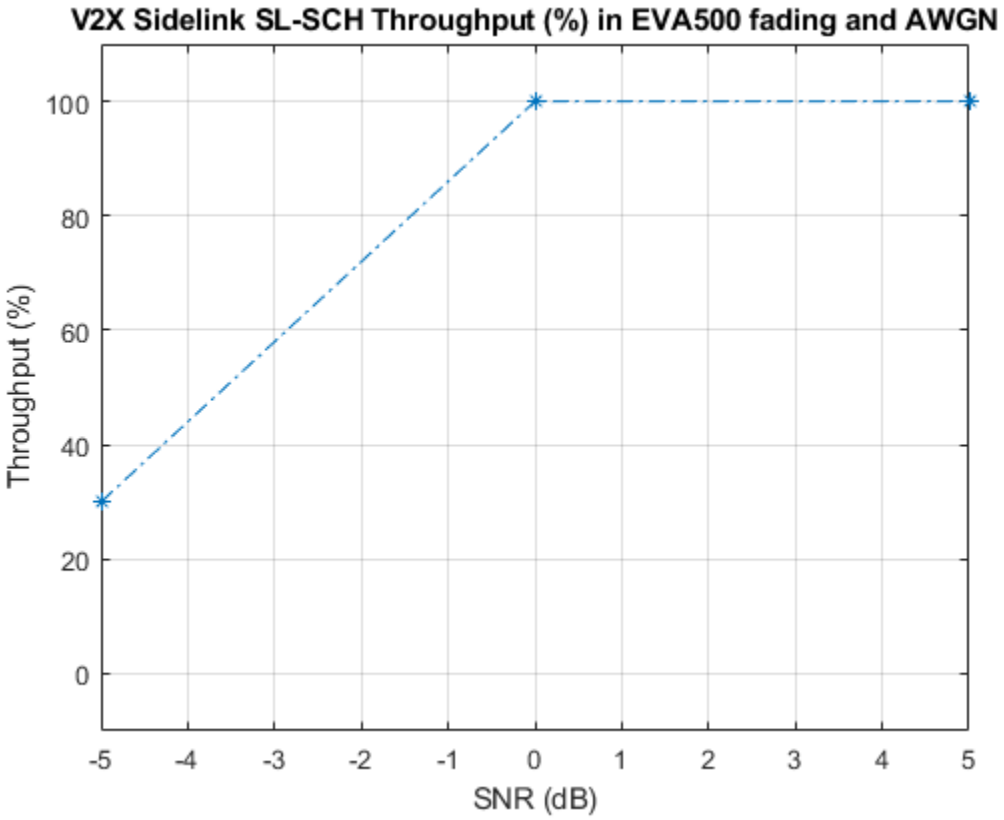
Throughput Results

The simulation results are displayed in the MATLAB® command window after each SNR point is completed. They are also captured in output arrays `simBLERSCI`, `simThroughputSLSCH` and `maxThroughputSLSCH`. For each simulated SNR point, `simBLERSCI` stores the measured SCI BLER,

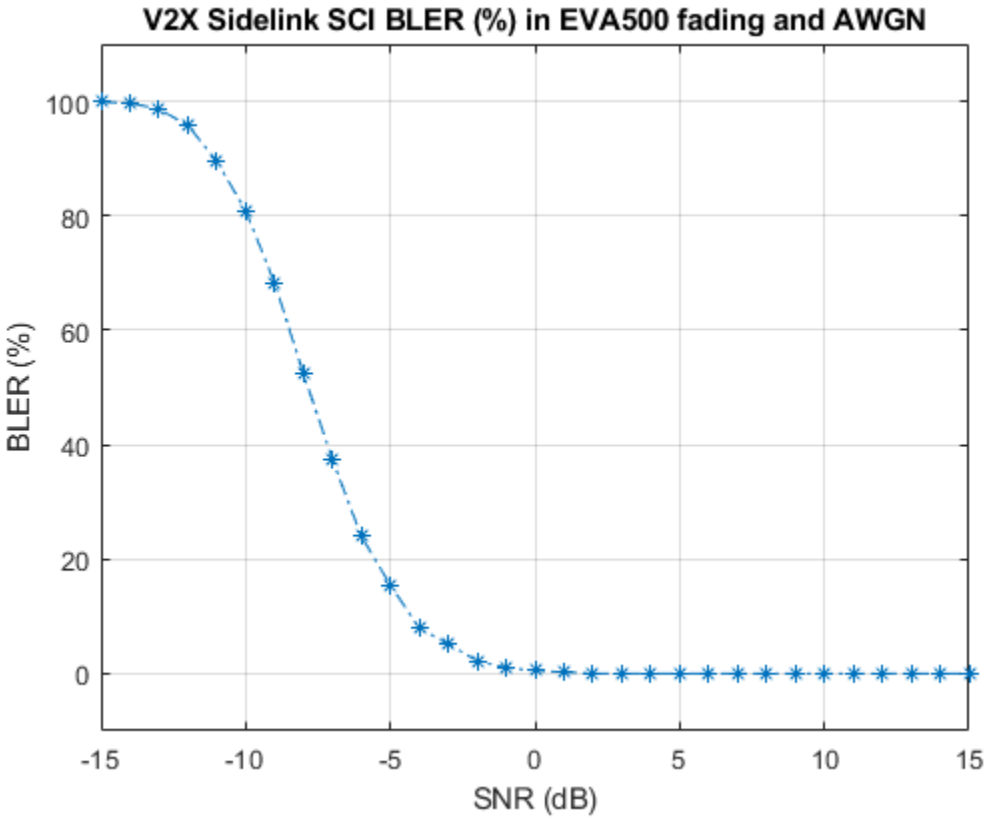
`simThroughputSLSCH` stores the measured SL-SCH throughput in number of bits and `maxThroughputSLSCH` stores the maximum possible SL-SCH throughput in number of bits.

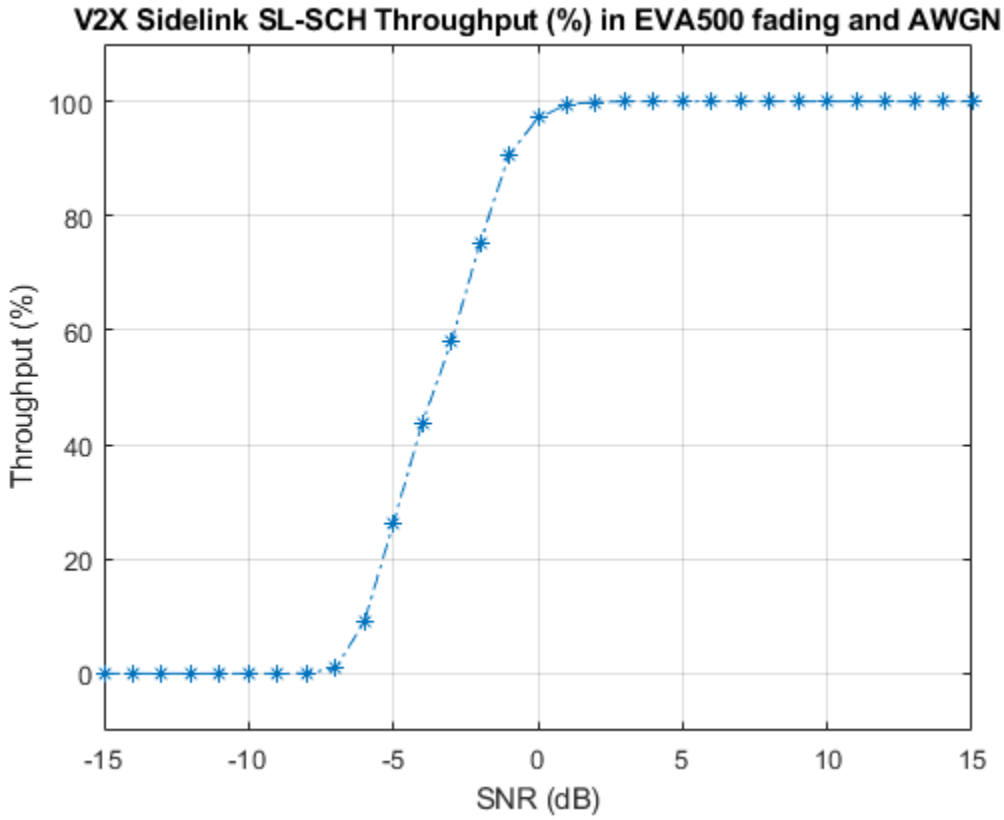
```
if measureBLERForSCI
    % Plot PSCCH/SCI BLER
    plotResults('SCI', 'BLER',channel,SNRIn,simBLERSCI);
end

if measureTputForSLSCH
    % Plot PSSCH/SL-SCH throughput
    plotResults('SL-SCH', 'Throughput',channel,SNRIn,simThroughputSLSCH*100./maxThroughputSLSCH)
end
```





For statistically valid results, run the simulation for a larger number of frames. The figures below show the SCI BLER and SL-SCH throughput results when simulating 1000 frames for an extended range of SNR values with practical channel estimation.


V2X Sidelink SCI BLER (%) in EVA500 fading and AWGN
BLER (%)
100
80
60
40
20
0
-15
-10
-5
0
5
10
15
SNR (dB)



References

1 3GPP TS 36.213 "Physical layer procedures"

2 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception"

Local Functions

This example uses these helper functions.

- `perfectChannelEstimate`: perfect channel estimation
- `plotResults`: plot the results of the example

```
% Calculate the perfect channel estimate and the noise estimate
function [hest,nest] = perfectChannelEstimate(rxConfig,channel,noise,frameOffset)
    % Use the uplink perfect channel estimator to calculate
    % the sidelink estimate as both use SC-FDMA
    rxConfig.NTxAnts = 1;
    rxConfig.NULRB = rxConfig.NSLRB;
    hest = lteULPerfectChannelEstimate(rxConfig,channel,frameOffset);
    noiseGrid = lteSLSCFDMADemodulate(rxConfig,noise(1+frameOffset:end ,:));
    nest = var(noiseGrid(:));
end

% Plot the SCI BLER or SL-SCH Throughput
function plotResults(chName,simOpt,channel,xvalues,yvalues)
    fadingDescription = sprintf('%s%s',channel.DelayProfile,num2str(channel.DopplerFreq));
    figure;
```

```
        plot(xvalues, yvalues,'*-.');
        v = axis;
        axis([v(1) v(2) -10 110])
        title(sprintf('V2X Sidelink %s %s (%%) in %s fading and AWGN',chName,simOpt,fadingDescriptio
        xlabel('SNR (dB)');
        ylabel([simOpt ' (%)']);
        grid on;
    end
```

NB-IoT Uplink Waveform Generation

This example shows how to generate LTE-Advanced Pro Release 13 Narrowband IoT (NB-IoT) uplink waveforms consisting of the Narrowband Physical Uplink Shared Channel (NPUSCH) and the associated demodulation reference signals for test and measurement applications using the LTE Toolbox™.

Introduction

3GPP introduced a new air interface, Narrowband IoT (NB-IoT) optimized for low data rate machine type communications in LTE-Advanced Pro Release 13. NB-IoT provides cost and power efficiency improvements as it avoids the need for complex signaling overhead required for LTE based systems.

The LTE Toolbox can be used to generate standard compliant NB-IoT uplink complex baseband waveforms representing the 180kHz narrowband carrier suitable for test and measurement applications. The LTE Toolbox supports all the NB-IoT modes of operation described below - standalone, guardband and in-band.

- Standalone: NB-IoT carrier deployed outside the LTE spectrum, e.g. the spectrum used for GSM or satellite communications
- Guardband: NB-IoT carrier deployed in the guardband between two LTE carriers
- In-band: NB-IoT carrier deployed in resource blocks of an LTE carrier

The NB-IoT uplink consists of the following physical layer channels and signals:

- Narrowband demodulation reference signal (DM-RS)
- Narrowband physical uplink shared channel (NPUSCH)
- Narrowband physical random access channel (NPRACH)

This example demonstrates the NB-IoT uplink resource element (RE) grid and waveform generation consisting of the NPUSCH and DM-RS signals. The sections below introduce these physical signals and channels that form the grid along with key concepts including subframe repetition, logical and transport channel mappings, and the corresponding grids for the different configurations.

The example outputs the complex baseband waveform along with the populated grid containing the NPUSCH and DM-RS signals. The waveform can be used for a range of applications from RF testing to simulation of receiver implementations.

NPUSCH Allocation

This section provides an overall description of how the NPUSCH gets mapped into the NB-IoT uplink slots.

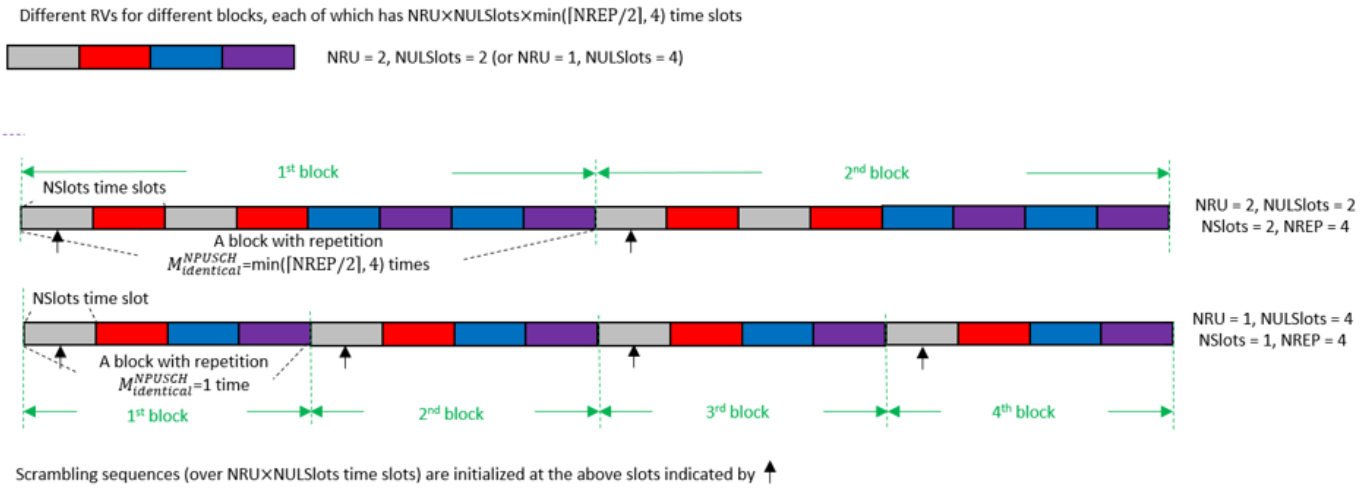
The NPUSCH can carry the uplink shared channel (UL-SCH) or the uplink control information according to the two formats:

- NPUSCH format 1, used to carry the uplink shared channel (UL-SCH)
- NPUSCH format 2, used to carry uplink control information

The NPUSCH is transmitted on one or more resource units and each of these resource units are repeated up to 128 times to improve transmission reliability and coverage without compromising on the low power and low complexity requirements to meet ultra low end IoT use cases.

The smallest mapping unit for the NPUSCH is a resource unit. It is defined as 7 * `NslotsUL` consecutive SC-FDMA symbols in the time domain and `NscRU` consecutive subcarriers in the frequency domain, where `NslotsUL` and `NscRU` are defined in TS 36.211 Table 10.1.2.3-1 [1]. The NB-IoT UL-SCH may carry Common Control Channel (CCCH), Dedicated Control Channel (DCCH) or Dedicated Traffic Channel (DTCH) and maps on to the NPUSCH physical channel (TS 36.300 section 6.1.3.1 and section 5.3.1a [6]). The NPUSCH can be mapped to one or more than one resource units, `NRU` as defined by TS 36.211 Section 10.1.3.6 [1] and each resource unit can be transmitted `Nrep` times.

The examples in the figure shows the repetition pattern with `NRep` = 4. The total duration for transmitting a block of data is given by `NRU` * `NULSlots` * `MidenticalNPUSCH` as specified in TS 36.211 Section 10.1.3.6 [1]. For the first case shown below, each transport block is transmitted over `NRU` = 2 and each of these `NRU` contains two UL slots indicated by `NULSlots`. After mapping to `Nslots`, these slots will be repeated `MidenticalNPUSCH` = 2 (assuming NscRU > 1) times. In the second case, we assume that `NscRU` is 1 and hence `MidenticalNPUSCH` = 1. This, combined with `Nslots` = 1 results in the transmission pattern where each block is transmitted without internal repetitions. In all cases, the scrambling sequence is reset at the start of the codeword transmission or retransmission (see TS 36.211 Section 10.1.3.1 [1]). The detailed specification of the repetition scheme can be found in TS 36.211 10.1.3 [1].



NB-IoT Uplink Slot Grid

In addition to the slot allocation described above, this section further explains the RE allocation in a slot. The grid consists of one or more frames containing NPUSCH and corresponding DM-RS.

- *DM-RS*: The DM-RS is transmitted in every NPUSCH slot with the same bandwidth as the associated NPUSCH. The reference signals depend on the number of subcarriers `NscRU`, the narrowband cell ID `NNcellID` and the NPUSCH format `NPUSCHFormat`. The RE positions depend on the NPUSCH format and subcarrier spacing. For NPUSCH format 1 with subcarrier spacing of 3.75kHz, the DM-RS is transmitted on symbol 4 and with subcarrier spacing of 15kHz, the DM-RS is transmitted on symbol 3. For NPUSCH format 2 with subcarrier spacing of 3.75kHz, the DM-RS is transmitted on symbols 0,1,2 and with subcarrier spacing of 15kHz, the DM-RS is transmitted on symbols 2,3 and 4 in the slot.

- *NPUSCH*: The NPUSCH supports single tone bandwidth in addition to the multitone (12 subcarriers) bandwidth. Single tone transmissions can use either the 15kHz or 3.75kHz subcarrier spacing whereas the multitone transmissions use the 15kHz subcarrier spacing. This means that the slot duration for 15kHz mode is 0.5ms and for 3.75kHz the slot duration is 2ms. The scrambling sequence is initialized in the first slot of the transmission of the codeword. If there are repetitions enabled, then the scrambling sequence is reinitialized after every `MidenticalNPUSCH` transmission of the codeword as described in TS 36.211 Section 10.1.3.1[1]. The codewords are BPSK/QPSK modulated on to a single layer and precoded before mapping to one or more resource units. All the resource elements other than those used for demodulation reference signals are used for NPUSCH transmission. If higher layer signaling (`npusch-AllSymbols` as described in TS 36.211 Section 10.1.3.6 [1]) indicates the presence of the SRS symbol, these symbols are counted in the NPUSCH mapping, but not used for the transmission of the NPUSCH (i.e. these NPUSCH positions are punctured by SRS).

NPUSCH Configuration

In this section, you configure the parameters required for NPUSCH generation. The UE uses the combination of MCS (modulation and coding scheme) and resource assignment signaled via the DCI to determine the transport block size from the set defined in TS 36.213 Table 16.5.1.2-2 [3] to use for NPUSCH transmission. In this example, this is specified via the parameter `tbs` and the duration of the generated waveform is controlled via the `totNumBlks` parameter.

```
tbs = 144;                          % The transport block size
totNumBlks = 1;                     % Number of simulated transport blocks

ue = struct();                      % Initialize the UE structure
ue.NBULSubcarrierSpacing = '15kHz'; % 3.75kHz, 15kHz
ue.NNCellID = 0;                    % Narrowband cell identity

chs = struct();
% NPUSCH carries data or control information
chs.NPUSCHFormat = 'Data'; % Payload type (Data or Control)
% The number of subcarriers used for NPUSCH 'NscRU' depends on the NPUSCH
% format and subcarrier spacing 'NBULSubcarrierSpacing' as shown in TS
% 36.211 Table 10.1.2.3-1. There are 1,3,6 or 12 contiguous subcarriers for
% NPUSCH
chs.NBULSubcarrierSet = 0;  % Range is 0-11 (15kHz); 0-47 (3.75kHz)
chs.NRUsc = length(chs.NBULSubcarrierSet);
chs.CyclicShift = 0;   % Cyclic shift required when NRUsc = 3 or 6
chs.RNTI = 0;          % RNTI value
chs.NLayers = 1;       % Number of layers
chs.NRU = 2;           % Number of resource units
chs.NRep = 4;          % Number of repetitions of the NPUSCH
chs.SlotIdx = 0;       % Start slot index in a bundle
% The symbol modulation depends on the NPUSCH format and NscRU as
% given by TS 36.211 Table 10.1.3.2-1
chs.Modulation = 'QPSK';
rvDCI = 0;             % RV offset signaled via DCI (See 36.213 16.5.1.2)

% Specify the NPUSCH and DM-RS power scaling in dB for plot visualization
chs.NPUSCHPower = 30;
chs.NPUSCHDRSPower = 34;
```

For DM-RS signals in NPUSCH format 1, sequence-group hopping can be enabled or disabled by the higher layer cell-specific parameter `groupHoppingEnabled`. Sequence-group hopping for a particular UE can be disabled through the higher layer parameter `groupHoppingDisabled` as

described in TS 36.211 Section 10.1.4.1.3 [1]. In this example, we use the `SeqGroupHopping` parameter to enable or disable sequence-group hopping.

```
chs.SeqGroupHopping = 'on'; % Enable/Disable Sequence-Group Hopping for UE
chs.SeqGroup = 0;           % Delta_SS. Higher-layer parameter groupAssignmentNPUSCH

% Get number of time slots in a resource unit NULSlots according to
% TS 36.211 Table 10.1.2.3-1
if strcmpi(chs.NPUSCHFormat,'Data')
    if chs.NRUsc == 1
        NULSlots = 16;
    elseif any(chs.NRUsc == [3 6 12])
        NULSlots = 24/chs.NRUsc;
    else
        error('Invalid number of subcarriers. NRUsc must be one of 1,3,6,12');
    end
elseif strcmpi(chs.NPUSCHFormat,'Control')
    NULSlots = 4;
else
    error('Invalid NPUSCH Format (%s). NPUSCHFormat must be ''Data'' or ''Control''',chs.NPUSCHF
end
chs.NULSlots = NULSlots;

NSlotsPerBundle = chs.NRU*chs.NULSlots*chs.NRep; % Number of slots in a codeword bundle
TotNSlots = totNumBlks*NSlotsPerBundle;   % Total number of simulated slots
```

NB-IoT Uplink Waveform Generation

In this section, you create the resource grid populated with the NPUSCH and the corresponding demodulation reference signals. This grid is then SC-FDMA modulated to generate the time domain waveform.

```
% Initialize the random generator to default state
rng('default');

% Get the slot grid and number of slots per frame
emptySlotGrid = lteNBResourceGrid(ue);
slotGridSize = size(emptySlotGrid);
NSlotsPerFrame = 20/(slotGridSize(1)/12);

state = [];    % NPUSCH encoder and DM-RS state, auto re-initialization in the function
trblk = [];    % Initialize the transport block
txgrid = [];   % Full grid initialization

% Display the number of slots being generated
fprintf('\nGenerating %d slots corresponding to %d transport block(s)\n',TotNSlots,totNumBlks);
for slotIdx = 0+(0:TotNSlots-1)
    % Calculate the frame number and slot number within the frame
    ue.NFrame = fix(slotIdx/NSlotsPerFrame);
    ue.NSlot = mod(slotIdx,NSlotsPerFrame);

    if isempty(trblk)
       if strcmpi(chs.NPUSCHFormat,'Data')
           % UL-SCH encoding is done for the two RV values used for
           % transmitting the codewords. The RV sequence used is determined
           % from the rvDCI value signaled in the DCI and alternates
           % between 0 and 2 as given in TS 36.213 Section 16.5.1.2
```

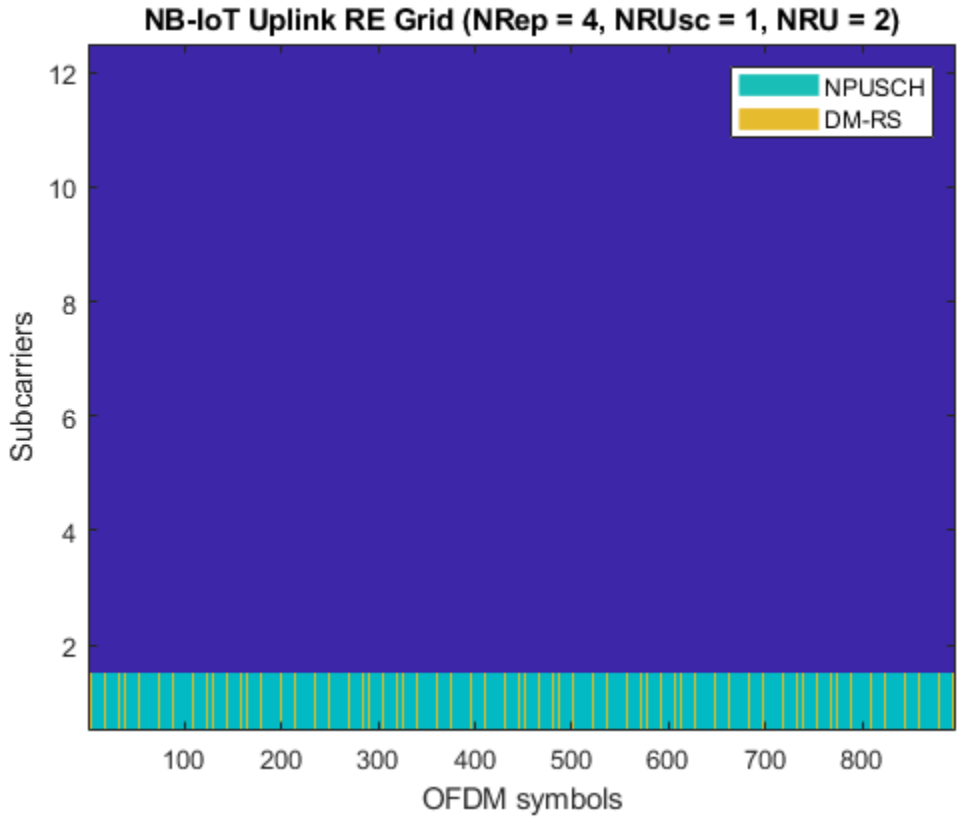
```
            % Define the transport block which will be encoded to create the
            % codewords for different RV
            trblk = randi([0 1],tbs,1);

            % Determine the coded transport block size
            [~, info] = lteNPUSCHIndices(ue,chs);
            outblklen = info.G;
            % Create the codewords corresponding to the two RV values used
            % in the first and second block, this will be repeated till all
            % blocks are transmitted
            chs.RV = 2*mod(rvDCI+0,2); % RV for the first block
            cw = lteNULSCH(chs,outblklen,trblk); % CRC and Turbo coding
            chs.RV = 2*mod(rvDCI+1,2); % RV for the second block
            cw = [cw lteNULSCH(chs,outblklen,trblk)]; %#ok<AGROW> % CRC and Turbo coding is repeat
        else
            trblk = randi([0 1],1); % 1 bit ACK
            % For ACK, the same codeword is transmitted every block as
            % defined in TS 36.212 Section 6.3.3
            cw = lteNULSCH(trblk);
        end
        blockIdx = 0; % First block to be transmitted
    end

    % Initialize grid
    slotGrid = emptySlotGrid;

    % NPUSCH encoding and mapping onto the slot grid
    txsym = lteNPUSCH(ue,chs,cw(:,mod(blockIdx,size(cw,2))+1),state);

    % Map NPUSCH symbols in the grid of a slot
    indicesNPUSCH = lteNPUSCHIndices(ue,chs);
    slotGrid(indicesNPUSCH) = txsym*db2mag(chs.NPUSCHPower);

    % Create DM-RS sequence and map to the slot grid
    [dmrs,state] = lteNPUSCHDRS(ue,chs,state);
    indicesDMRS = lteNPUSCHDRSIndices(ue,chs);
    slotGrid(indicesDMRS) = dmrs*db2mag(chs.NPUSCHDRSPower);

    % Concatenate this slot to the slot grid
    txgrid = [txgrid slotGrid]; %#ok<AGROW>

    % If a full block is transmitted, increment the clock counter so that
    % the correct codeword can be selected
    if state.EndOfBlk
        blockIdx = blockIdx + 1;
    end

    % trblk err count and re-initialization
    if state.EndOfTx
        % Re-initialize to enable the transmission of a new transport block
        trblk = [];
    end

end

% Perform SC-FDMA modulation to create time domain waveform
ue.CyclicPrefixUL = 'Normal'; % Normal cyclic prefix length for NB-IoT
[waveform,scfdmaInfo] = lteSCFDMAModulate(ue,chs,txgrid);
```

```
Generating 128 slots corresponding to 1 transport block(s)
```

Plot Transmitted Grid

Plot the populated grid and observe the NPUSCH and corresponding DM-RS. The positions of the NPUSCH and DM-RS depends on the number of subcarriers `chs.NRUsc` and the subcarriers used as specified by `chs.NBULSubcarrierSet`. Note that the resource grid plot uses the power levels of the PUSCH and the DM-RS to assign colors to the resource elements.

```
% Create an image of overall resource grid
figure
im = image(abs(txgrid));
cmap = parula(64);
colormap(im.Parent,cmap);
axis xy;
title(sprintf('NB-IoT Uplink RE Grid (NRep = %d, NRUsc = %d, NRU = %d)',chs.NRep,chs.NRUsc,chs.NR
xlabel('OFDM symbols')
ylabel('Subcarriers')
% Create the legend box to indicate the channel/signal types associated with the REs
reNames = {'NPUSCH';'DM-RS'};
clevels = round(db2mag([chs.NPUSCHPower chs.NPUSCHDRSPower]));
N = numel(reNames);
L = line(ones(N),ones(N), 'LineWidth',8); % Generate lines
% Set the colors according to cmap
set(L,{'color'},mat2cell(cmap( min(1+clevels,length(cmap) ),:),ones(1,N),3));
legend(reNames{:});
```



Selected Bibliography

1. 3GPP TS 36.211 "Physical channels and modulation"
2. 3GPP TS 36.212 "Multiplexing and channel coding"
3. 3GPP TS 36.213 "Physical layer procedures"
4. 3GPP TS 36.321 "Medium Access Control (MAC); Protocol specification"
5. 3GPP TS 36.331 "Radio Resource Control (RRC); Protocol specification"
6. 3GPP TS 36.300 "Overall description; Stage 2"
7. O. Liberg, M. Sundberg, Y.-P. Wang, J. Bergman and J. Sachs, Cellular Internet of Things: Technologies, Standards and Performance, Elsevier, 2018.

NB-IoT Downlink In-Band and Guardband Waveform Generation and Analysis

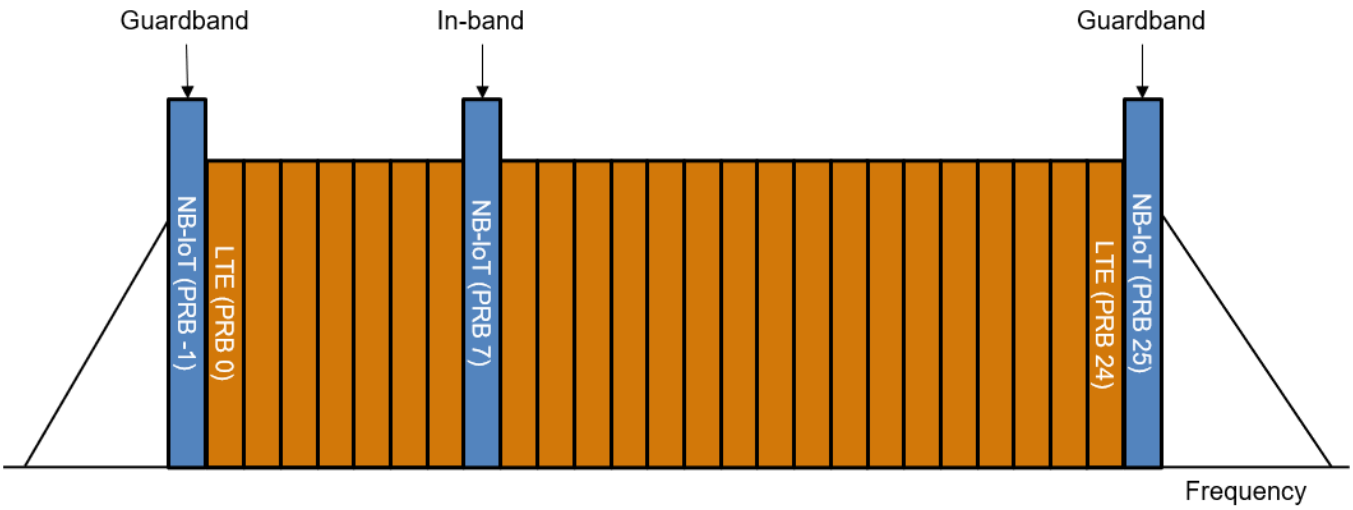
This example shows how to generate a narrowband internet of things (NB-IoT) downlink waveform for in-band and guardband operation modes on an LTE carrier by using the LTE Toolbox™. The example also shows an analysis of the quality of the recovered NB-IoT physical downlink shared channel (NPDSCH) through error vector magnitude (EVM) measurements.

Introduction

To support maximum flexibility of the NB-IoT deployment, 3GPP defines these three NB-IoT modes of operation:

- Standalone: NB-IoT carrier deployed outside the LTE spectrum, e.g., the spectrum used for GSM or satellite communications
- Guardband: NB-IoT carrier deployed in the guardband between two LTE carriers
- In-band: NB-IoT carrier deployed in resource blocks of an LTE carrier

This example focuses on the in-band and guardband modes. For more information on generating a standalone NB-IoT downlink waveform, see the “NB-IoT Downlink Waveform Generation” on page 2-104 example. This figure shows a sketch of in-band and guardband modes for a 5 MHz LTE carrier.



In-band mode can be further grouped depending on the physical cell identity (PCI) used, Inband-SamePCI and Inband-DifferentPCI. If Inband-SamePCI, the physical layer cell identity and PCI are the same and the user equipment (UE) can make assumptions about the ports and channel from LTE signals. The NB-IoT master information block (MIB-NB) indicates the operation mode. The network can use radio resource control information to allocate a non-anchor carrier to a UE operating in an anchor carrier, see Section 6.7.3.2 of TS 36.331 and sections 7.1.2.5 and 7.2.1.1 in [5 on page 2-302].

Table 16.8-1 of TS 36.213 restricts the allowed physical resource block (PRB) indices for in-band mode of an NB-IoT anchor carrier to these values. In-band mode of NB-IoT is not supported on a 1.4

MHz LTE carrier. For guardband mode of NB-IoT, the allowed PRB indices depend on the guardband size of the LTE carrier. Guardband mode of NB-IoT is not supported on a 1.4 MHz and 3 MHz LTE carrier. This table shows the allowed PRB indices for both in-band and guardband modes.

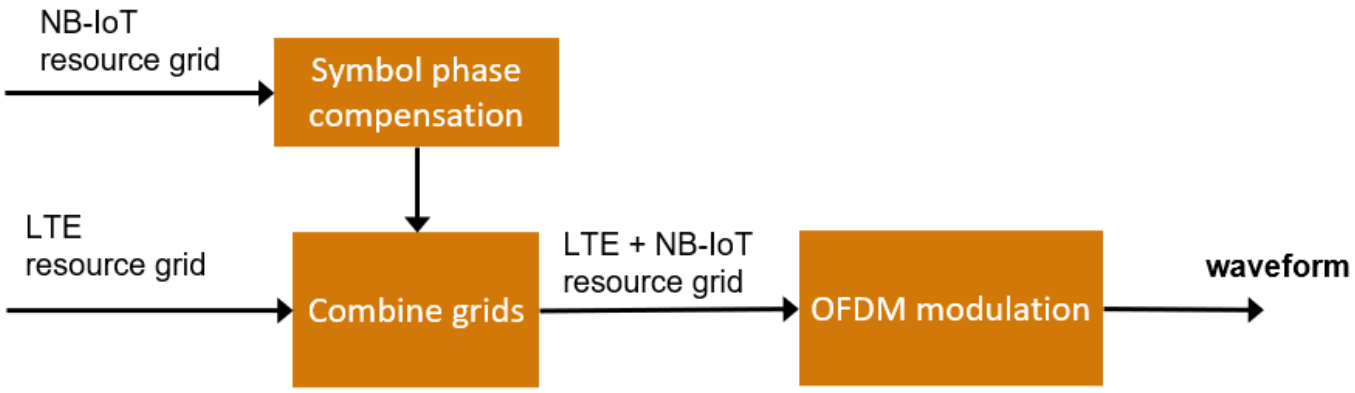
```
prbTable = nbAllowedPRB()
```

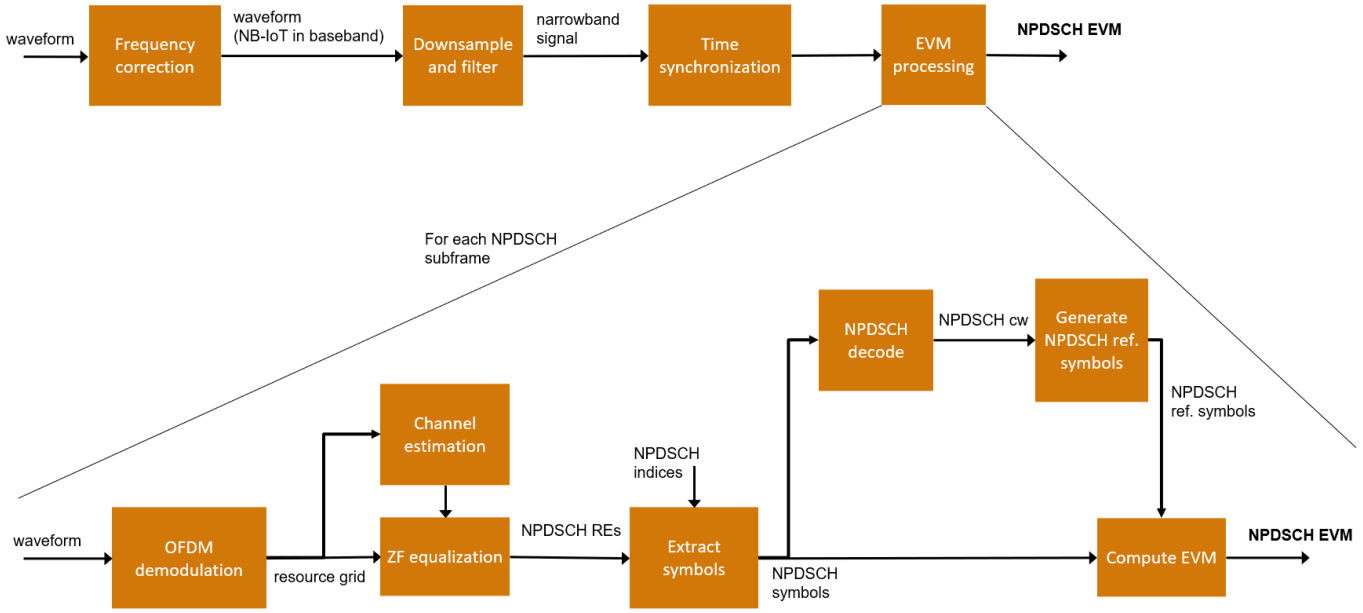
```
prbTable=6×3 table
       LTE Bandwidth             Allowed PRB indices for NB-IoT in-band mode          Allowed PRB
    ____________________    ____________________________________________________    ____________

    "1.4 MHz (  6 RBs)"    {1x0 double                                          }       {1x0 dou
    "  3 MHz ( 15 RBs)"    {[                                              2 12]}       {1x0 dou
    "  5 MHz ( 25 RBs)"    {[                                          2 7 17 22]}       {[
    " 10 MHz ( 50 RBs)"    {[                                4 9 14 19 30 35 40 45]}       {[
    " 15 MHz ( 75 RBs)"    {[            2 7 12 17 22 27 32 42 47 52 57 62 67 72]}       {[
    " 20 MHz (100 RBs)"    {[4 9 14 19 24 29 34 39 44 55 60 65 70 75 80 85 90 95]}       {[-5 -4
```

The example shows how to:

- Generate a waveform for an NB-IoT in-band operation mode
- Retrieve the original NB-IoT waveform from the received combined waveform containing both the NB-IoT and the LTE waveforms
- Measure NPDSCH EVM to verify the quality of the recovered NB-IoT waveform

These figures show the steps to generate the waveform and the processing steps at the receiver to compute the NPDSCH EVM. The frequency correction at the receiver introduces a phase offset to each OFDM symbol. For this reason, the example applies symbol phase compensation to the NB-IoT resource grid before combining it with the LTE grid.





Simulation Configuration

The example specifies an in-band operation mode of an NB-IoT anchor carrier on a 5 MHz LTE carrier. You can use the `nbPRB` parameter to change the index of the LTE PRB for NB-IoT allocation. To study the effect of the LTE interference on the NPDSCH EVM, run the example again without LTE interference by setting `lteDisabled` to `true`.

```
operationMode = Inband-SamePCI; % NB-IoT operation mode
ltecfg = struct(); % LTE configuration
ltecfg.TotSubframes = 10; % Number of subframes
ltecfg.NDLRB = 25; % Number of downlink resource blocks
nbPRB = 7; % PRB in which the NB-IoT is deployed (0-based)
lteDisabled = false; % Set to true to disable the LTE component
nbPowerBoost = 6; % NB-IoT power boost in dB

% Verify that the PRB chosen for the NB-IoT operation mode in the LTE carrier is allowed.
verifyNBPRB(nbPRB,ltecfg.NDLRB,operationMode);
```

NB-IoT Downlink Configuration and Resource Grid Generation

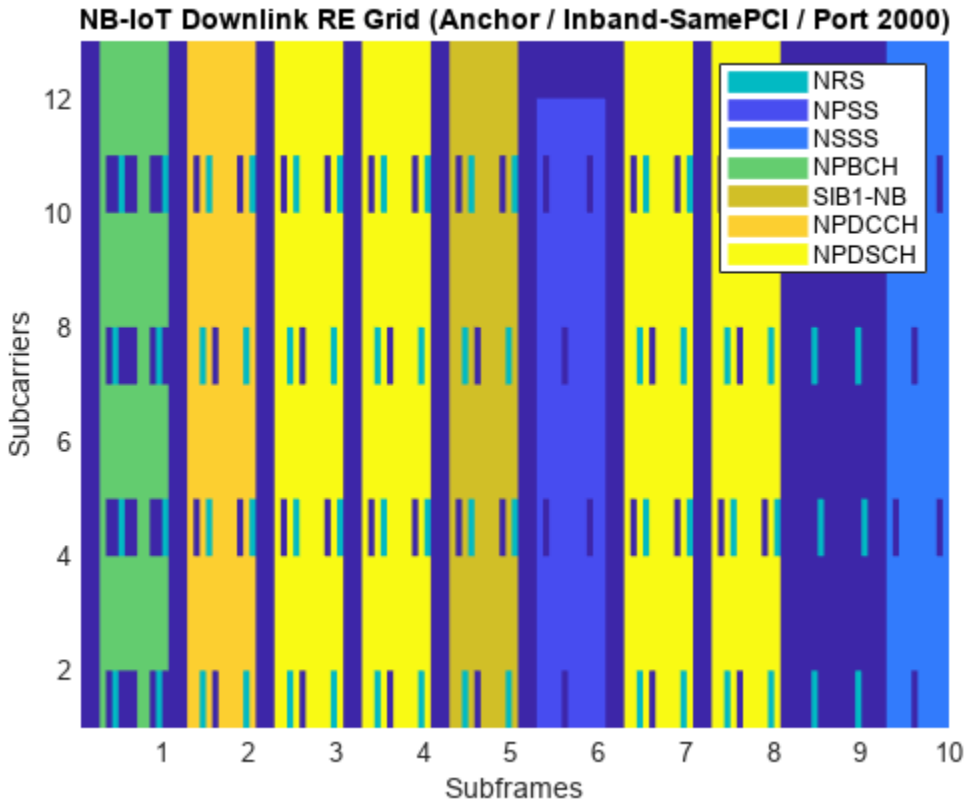
Configure the NB-IoT downlink waveform generator. The example uses the `NBIoTDownlinkWaveformGenerator` class to generate the NB-IoT resource grid. See the “NB-IoT Downlink Waveform Generation” on page 2-104 example for more information about `NBIoTDownlinkWaveformGenerator`.

```
ngen = NBIoTDownlinkWaveformGenerator;
ngen.Config.CellRefP = 1;
ngen.Config.NNCellID = 0;
ngen.Config.NFrame = 0;
ngen.Config.TotSubframes = ltecfg.TotSubframes;
ngen.Config.OperationMode = operationMode;
```

```
ngen.Config.NPDSCH.StartSubframe = 2;
ngen.Config.NPDSCH.NRep = 1;
ngen.Config.Windowing = 3; % Set Windowing to 3, as shown in Table E.5.1-1a of TS 36.104
```

Display the resource grid.

```
figure;
displayResourceGrid(ngen);
```



Generate the NB-IoT downlink waveform and resource grid.

```
[nbWaveform,nbGrid,nbInfo] = generateWaveform(ngen);
nbInfo
```

```
nbInfo = struct with fields:
            SamplingRate: 1920000
                    Nfft: 128
               Windowing: 3
     CyclicPrefixLengths: [10 9 9 9 9 9 9 10 9 9 9 9 9 9]
    SubframeChannelTypes: ["NPBCH"    "NPDCCH"    "NPDSCH"    "NPDSCH"    "SIB1-NB"    "NPSS"
```

LTE Downlink Configuration and Resource Grid Generation

Configure the LTE carrier. If you selected in-band operation mode, the example leaves the `nbPRB` PRB index empty in the LTE configuration.

```
nbPRB = double(nbPRB);
ltecfg.PDSCH.PRBSet = (0:ltecfg.NDLRB-1)'; % Full PRB allocation for LTE
```

```
if contains(ngen.Config.OperationMode,'Inband')
    ltecfg.PDSCH.PRBSet(nbPRB+1) = []; % Leave nbPRB empty for in-band NB-IoT
end
```

Generate the LTE resource grid. The example uses `lteRMCDLTool` to generate the wideband resource grid.

```
[~,lteGrid,lteInfo] = lteRMCDLTool(ltecfg,[1 0 0 1]);
```

Waveform Generation

The example combines LTE and NB-IoT resource grids before generating the waveform. The `hNBInbandGuardbandCombineGrid` function applies pre-OFDM symbol phase compensation to the NB-IoT grid to deal with the distortion that the receiver introduces when shifting the received waveform to bring NB-IoT in baseband.

Generate the combined resource grid and get the NB-IoT frequency offset.

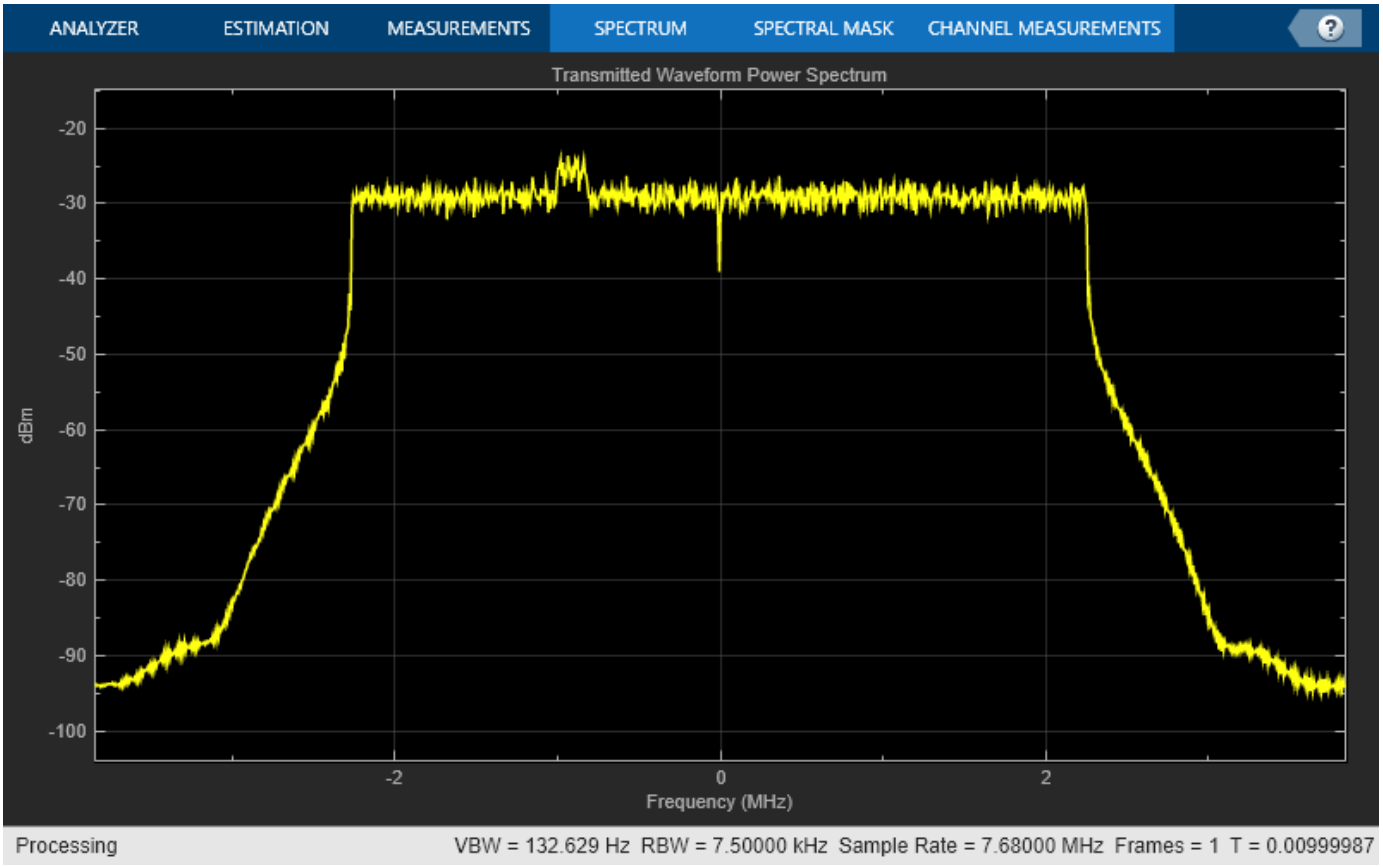
```
% Create the parameter structure needed to generate the combined grid
combGridParams = struct();
combGridParams.OperationMode = ngen.Config.OperationMode;
combGridParams.NBPRB = nbPRB;
combGridParams.OFDMInfo = nbInfo;
combGridParams.LTEDisabled = lteDisabled;
[combGrid,offset] = hNBInbandGuardbandCombinedGrid(combGridParams,lteGrid,nbGrid*db2mag(nbPowerBo
```

Generate the waveform.

```
OSR = lteInfo.SamplingRate/nbInfo.SamplingRate; % Oversampling ratio for NB-IoT
wavecfg.Windowing = ngen.Config.Windowing*OSR; % Use the same windowing as the NB-IoT waveform
txWaveform = lteOFDMModulate(wavecfg,combGrid);
```

Plot the power spectrum of the transmitted waveform.

```
txPlot = spectrumAnalyzer('SampleRate',lteInfo.SamplingRate,'Title','Transmitted Waveform Power S
txPlot(txWaveform);
```



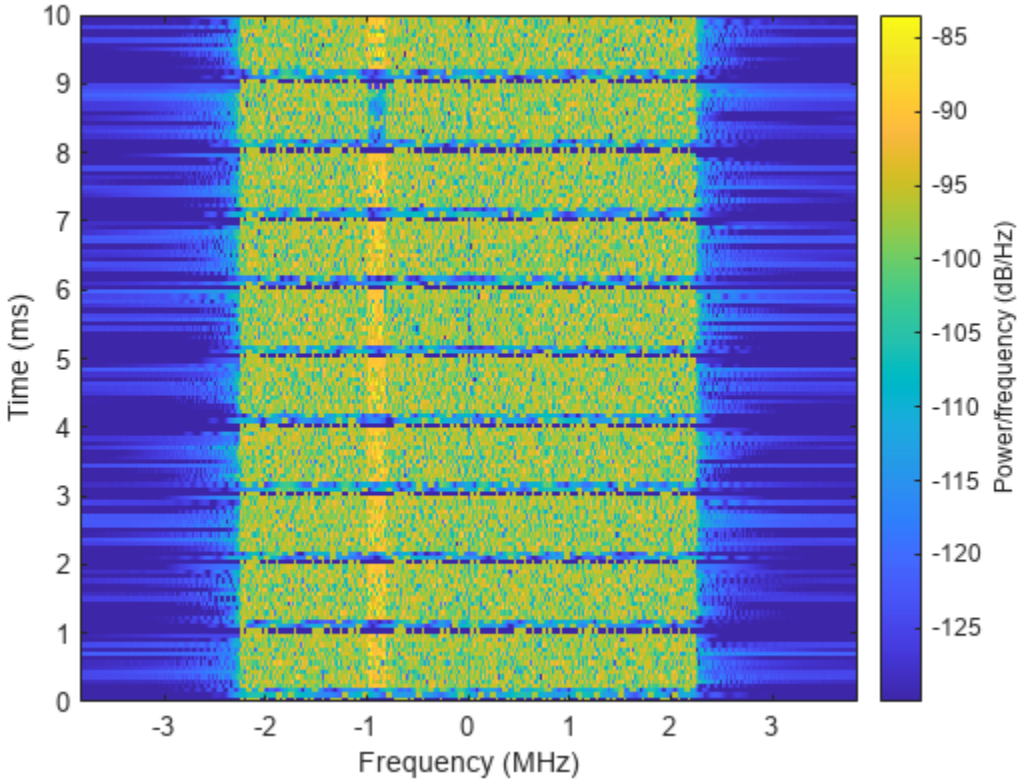
NB-IoT Synchronization and Filtering

This section follows these steps:

- Frequency correction. Shift the received waveform in frequency to bring the NB-IoT spectrum to baseband (0 Hz).
- Filter out the LTE component and downsample. Downsample the waveform to the NB-IoT sampling rate and filter out the frequencies outside the NB-IoT band.
- Time synchronization. Remove any potential time delay added by the filter.

Plot the spectrogram of the received waveform. This plot shows the frequency occupied by the NB-IoT component in the received wideband waveform.

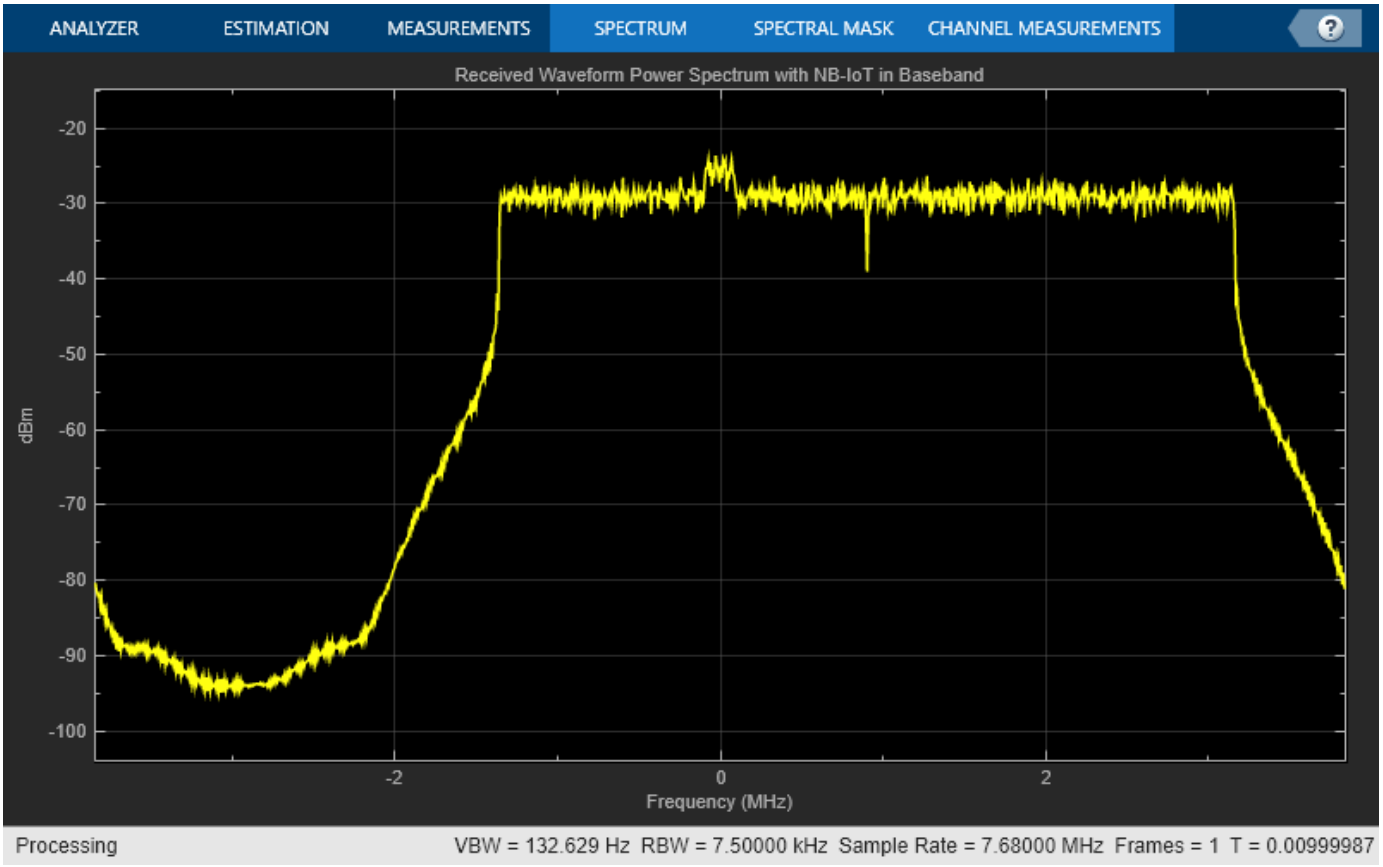
```
rxWaveform = txWaveform;
figure;
spectrogram(rxWaveform(:,1),ones(lteInfo.Nfft,1),0,lteInfo.Nfft,'centered',lteInfo.SamplingRate,
```



Frequency Correction

Shift the received waveform to bring the NB-IoT component to baseband. The plot shows the power spectrum of the received waveform after the frequency correction.

```
rxWaveformB = lteFrequencyCorrect(ltecfg,rxWaveform,offset); % Received waveform with NB-IoT in B
combinedSpecPlotB = spectrumAnalyzer('SampleRate',lteInfo.SamplingRate,'Title','Received Waveforr
combinedSpecPlotB(rxWaveformB);
```



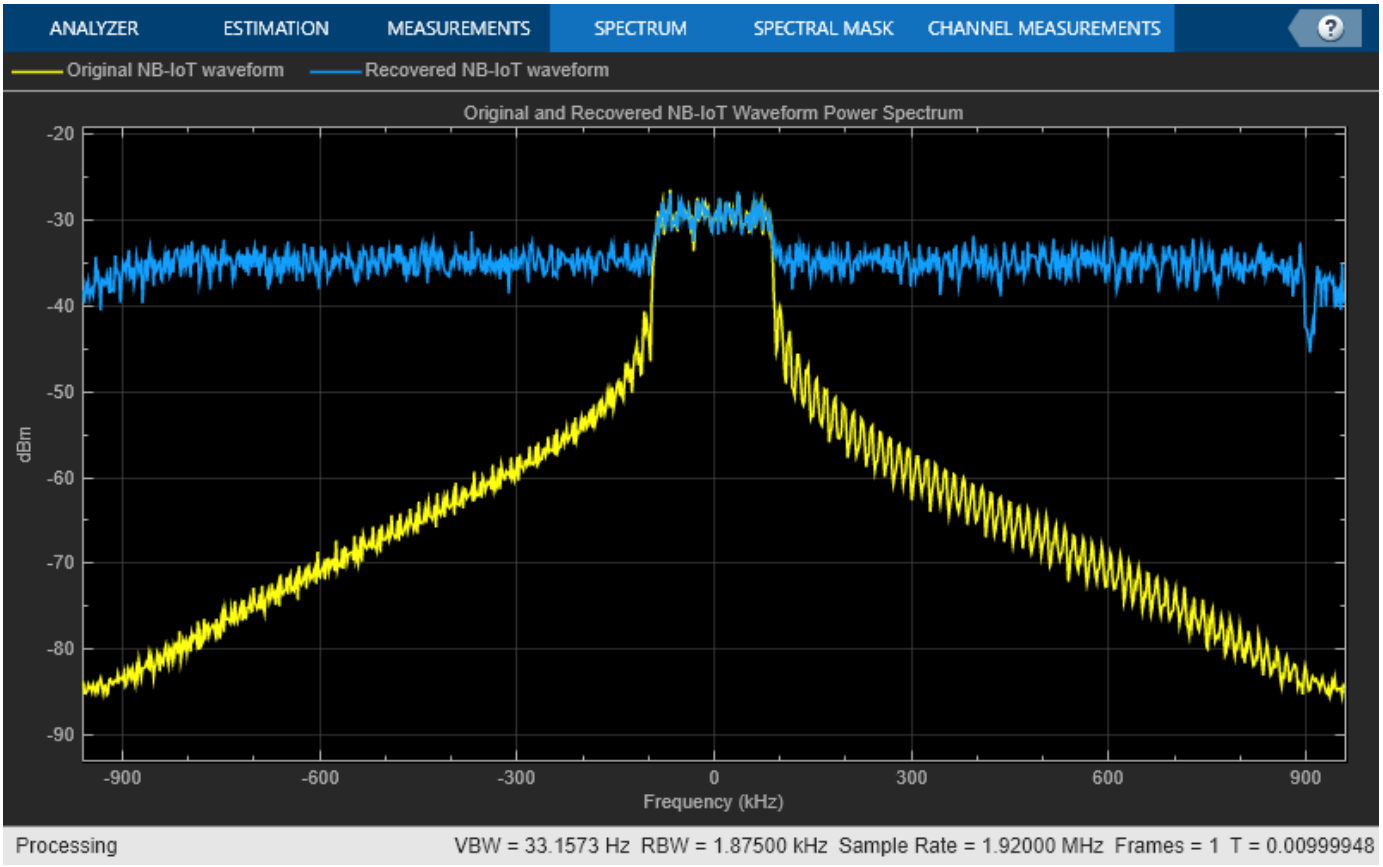
Filtering and Downsampling of NB-IoT Waveform

This example uses the `resample` function to downsample the wideband waveform to the nominal NB-IoT sampling rate of 1.92 MHz. Furthermore, the default FIR antialiasing lowpass filter implemented in the `resample` function filters out the unwanted LTE waveform, without the need of an additional lowpass filter.

```
rxwave = resample(rxWaveformB,1,OSR);
```

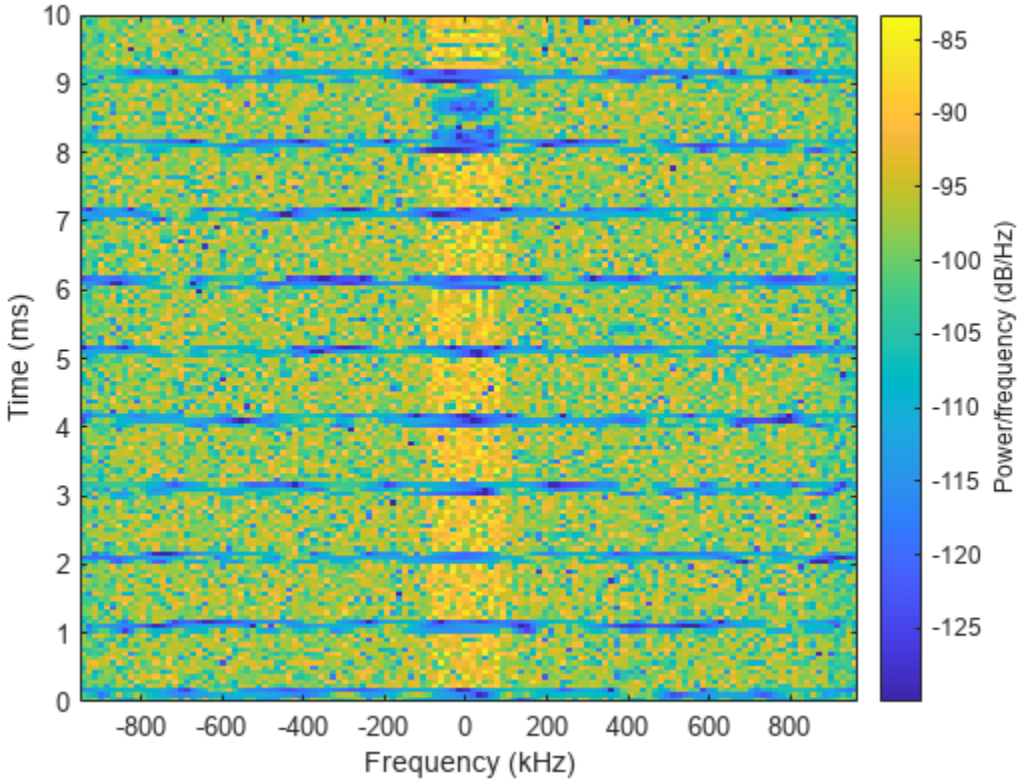
Plot the power spectrum of the recovered NB-IoT waveform against the power spectrum of the original NB-IoT waveform generated by the NB-IoT waveform generator `NBIoTDownlinkWaveformGenerator`. Because the power of the generated waveform depends on the FFT size, divide the original waveform by the oversampling rate so that both waveforms have a comparable power.

```
filteredSpecPlot = spectrumAnalyzer('SampleRate',nbInfo.SamplingRate,'Title','Original and Recove
    'ChannelNames',{'Original NB-IoT waveform','Recovered NB-IoT waveform'},'ShowLegend',true);
filteredSpecPlot([nbWaveform/OSR*db2mag(nbPowerBoost),rxwave]);
```



Plot the spectrogram of the recovered NB-IoT waveform. This plot shows the baseband narrowband waveform after extracting it from the received wideband waveform.

```
figure;
spectrogram(rxwave(:,1),ones(nbInfo.Nfft,1),0,nbInfo.Nfft,'centered',nbInfo.SamplingRate,'MinThre
```



Time Synchronization

Extract NB-IoT configuration parameters for time synchronization and EVM measurements.

```
enb = ngen.Config;
enb.NSubframe = lteInfo.NSubframe;
```

Apply time synchronization to the resulting waveform.

```
timeOffset = lteNBDLFrameOffset(enb,rxwave); % Time delay in samples
rxwave = rxwave(1+timeOffset:end,:);
```

EVM Measurements

Channel Estimator Configuration for EVM measurements

Parameterize the channel estimator at the receiver end using the structure `cec`.

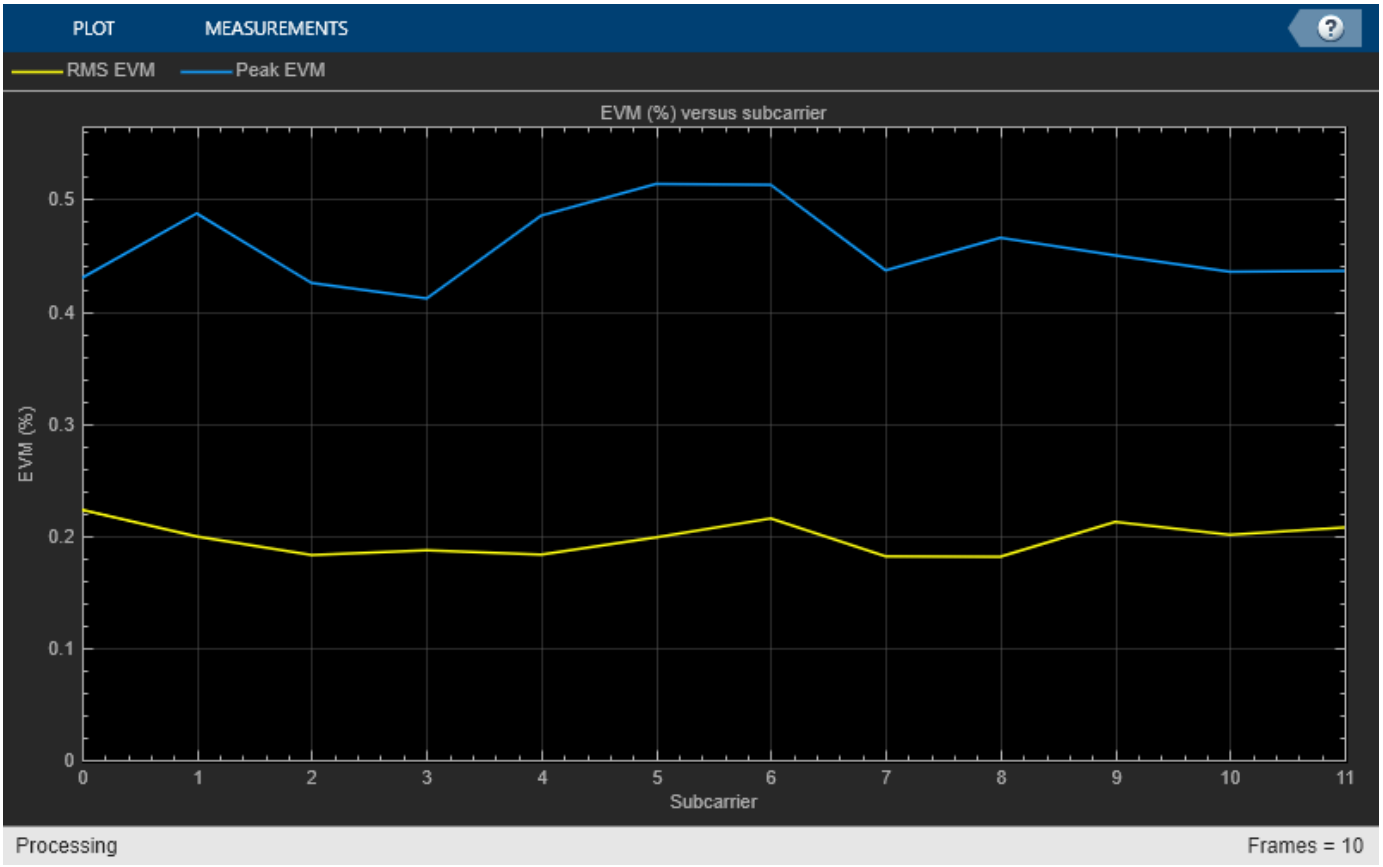
```
cec = struct;                       % Channel estimation config structure
cec.PilotAverage = 'UserDefined';   % Type of pilot symbol averaging
cec.FreqWindow = 13;                % Frequency window size
cec.TimeWindow = 9;                 % Time window size
cec.InterpType = 'Cubic';           % 2D interpolation type
cec.InterpWindow = 'Centered';      % Interpolation window type
cec.InterpWinSize = 1;              % Interpolation window size
cec.Reference = 'NRS';              % Reference signal for channel estimation
```

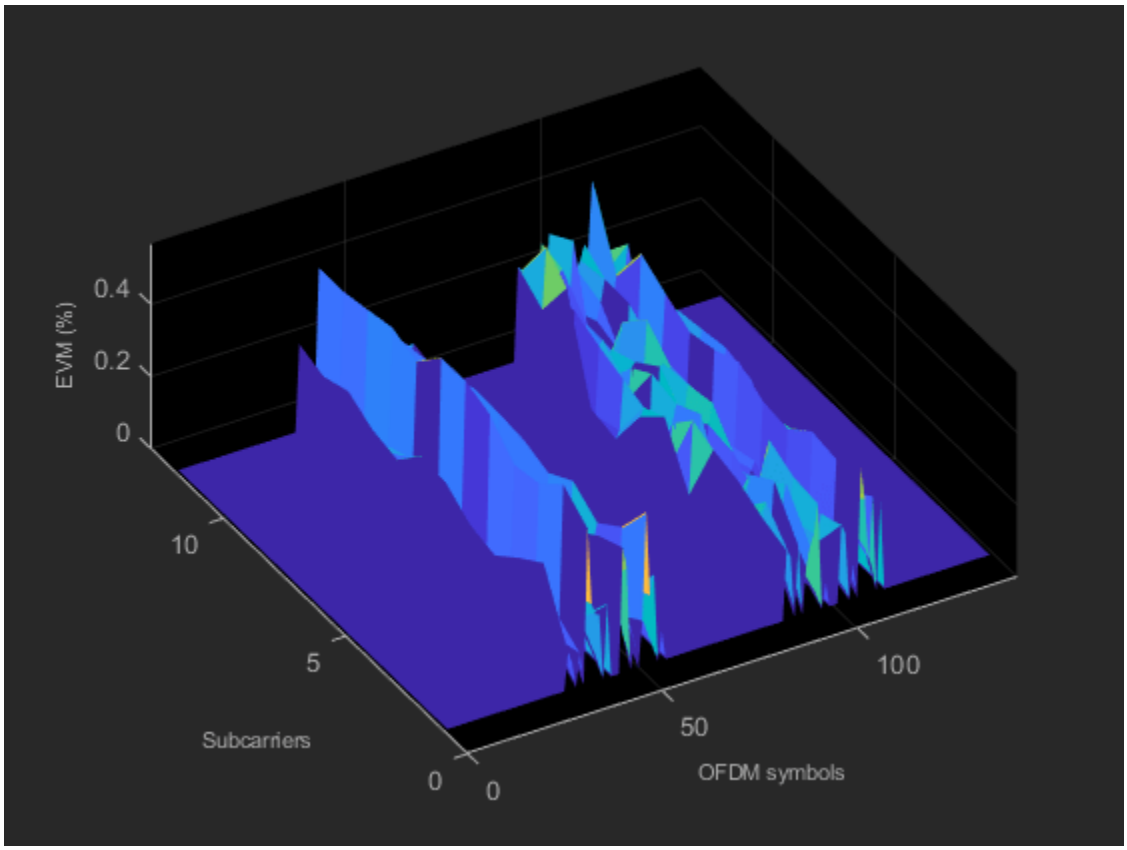
Process EVM

The `hNPDSCHEVM` function provides per subframe and average EVM measurements. The example also displays plots with the EVM versus time and subcarriers.

```
[evmmeas,plots] = hNPDSCHEVM(enb,cec,rxwave,nbInfo);
```

```
EVM subframe 2: 0.210%
EVM subframe 3: 0.176%
EVM subframe 6: 0.207%
EVM subframe 7: 0.192%
```


PLOT
MEASUREMENTS
RMS EVM
Peak EVM
EVM (%) versus subcarrier
EVM (%)
0
0.1
0.2
0.3
0.4
0.5
0
1
2
3
4
5
6
7
8
9
10
11
Subcarrier
Processing
Frames = 10




```
Averaged EVM frame 0: 0.197%
Averaged overall EVM: 0.197%
```

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 36.104. "Evolved Universal Terrestrial Radio Access (E-UTRA); Base Station (BS) Radio Transmission and Reception." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
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*.
4. 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*.
5. O. Liberg, M. Sundberg, Y.-P. Wang, J. Bergman, J. Sachs and G. Wikstrom, Cellular Internet of Things: From Massive Deployments to Critical 5G Applications, Elsevier, 2020.

Local Functions

```
function prbTable = nbAllowedPRB()
    % Generate a table showing the PRB indices allowed for NB-IoT in-band
    % and guardband operation mode for each LTE bandwidth.

    NDLRBList = [6 15 25 50 75 100];
    bwMHz = [1.4 3 5 10 15 20]; % Bandwidth in MHz
    bw = (pad(string(bwMHz),'left') + repmat(" MHz (",size(bwMHz)) + pad(string(NDLRBList),'left

    % In-band mode
    % The allowed PRB indices for in-band mode of NB-IoT anchor
    % carrier are derived from Table 16.8-1 of TS 36.213.
    prbIndexOdd = [-35:5:-5 5:5:35]; % DC (i.e., PRB 0) is excluded
    prbIndexEven = [-46:5:-6 5:5:45]; % DC (i.e., PRB 0) is excluded
    PRBInband = cell(length(NDLRBList),1);
    for ibw = 1:length(NDLRBList)
        if mod(NDLRBList(ibw),2) % Odd
            prbTemp = prbIndexOdd + floor(NDLRBList(ibw)/2);
        else % Even
            prbTemp = prbIndexEven + floor(NDLRBList(ibw)/2);
        end
        prbTemp(prbTemp<0 | prbTemp>=NDLRBList(ibw)) = [];
        PRBInband{ibw} = prbTemp;
    end
    PRBVariableNameInband = "Allowed PRB indices for NB-IoT in-band mode";

    % Guardband mode
    % The allowed PRB indices for NB-IoT guardband mode are derived
    % here for all channel bandwidths.
    totalGuardHz = bwMHz*1e6 - NDLRBList*12*15e3; % Total guardband in Hz
    NRBGuard = floor((totalGuardHz/2) / (12*15e3)); % Number of RBs allowed on each side of the g
    PRBGuard = cellfun(@(x,y)([-x:-1, y+(0:(x-1))]),num2cell(NRBGuard(:)),num2cell(NDLRBList(:))
    PRBVariableNameGuard = "Allowed PRB indices for NB-IoT guardband mode";

    prbTable = table(bw,PRBInband,PRBGuard);
    prbTable.Properties.VariableNames = ["LTE Bandwidth" PRBVariableNameInband PRBVariableNameGua
end

function verifyNBPRB(nbPRB,NDLRB,operationMode)
    % Check whether the PRB chosen for the NB-IoT operation mode in the LTE
    % carrier is allowed.
    prbTable = nbAllowedPRB();

    r = contains(prbTable{:,1},string(NDLRB)+" RBs"); % prbTable row associated to the given banc
    c = 1+contains(operationMode,'Inband')*1+contains(operationMode,'Guard')*2; % prbTable columr

    errorFlag = ismember(nbPRB,prbTable{r,c}{1}); % If true, the selected nbPRB is allowed
    tmp = extract(prbTable{r,1},digitsPattern);
    msg = "The chosen PRB index ("+string(nbPRB)+") is invalid for "+operationMode+" operation mo
        " MHz LTE carrier. See 'prbTable' for the allowed PRB indices for all combinations of LTE
    assert(errorFlag,msg); % Throws an error if the chosen nbPRB is invalid for the given combina
end
```

NB-IoT NPDSCH Block Error Rate Simulation

This example shows how LTE Toolbox™ can be used to create a NB-IoT Narrowband Physical Downlink Shared Channel (NPDSCH) Block Error Rate (BLER) simulation under frequency-selective fading and Additive White Gaussian Noise (AWGN) channel.

Introduction

3GPP Release 13 of LTE started to add support for Narrowband IoT applications. Release 13 defines a single NB-IoT UE Category, namely Cat-NB1, and Release 14 adds Cat-NB2 which allows for larger transport block sizes. This example focuses on Release 13 NB-IoT.

The example generates a NB-IoT NPDSCH BLER curve for a number of SNR points and transmission parameters. NPSS and NSSS are transmitted in appropriate subframes and the NPSS is used for practical timing synchronization. NPSS and NSSS subframes are not used for NPDSCH transmission. The NRS is transmitted in NPDSCH subframes and is used for practical channel estimation. NPBCH transmission gaps are not considered in this example.

Simulation Configuration

The simulation length is 4 DL-SCH transport blocks for a number of SNR points. A larger number of `numTrBlks` should be used to produce meaningful throughput results. `SNR` can be an array of values or a scalar. The simulation is performed over different repetition values to compare the performance improvement with repetitions.

```
numTrBlks = 4;        % Number of simulated transport blocks
SNRdB = -32:4:0;      % SNR range in dB
ireps = [0 5 9];      % Range of reps simulated
```

Setup Higher Layer Parameters

Setup the following higher layer parameters which are used to configure the NPDSCH in the next section:

- The variable `NPDSCHDataType` indicates whether the NPDSCH is carrying the SystemInformationBlockType1-NB (SIB1-NB) or not, and whether the NPDSCH is carrying the broadcast control channel (BCCH) or not. The allowed values of `NPDSCHDataType` are `'SIB1NB'`, `'BCCHNotSIB1NB'` and `'NotBCCH'`. Note that SIB1-NB belongs to the BCCH.
- The number of NPDSCH repetitions and the transport block size (TBS) are affected by whether NPDSCH is carrying SIB1-NB or not (see 3GPP TS 36.213 16.4.1.3 and 16.4.1.5 [2]). `NPDSCHDataType` set to `'SIB1NB'` indicates that the NPDSCH is carrying SIB1-NB; `NPDSCHDataType` set to either `'BCCHNotSIB1NB'` or `'NotBCCH'` indicates that the NPDSCH is not carrying SIB1-NB.
- The NPDSCH repetition pattern and the scrambling sequence generation is affected by whether NPDSCH is carrying BCCH or not (see 3GPP TS 36.211 10.2.3 [1]). `NPDSCHDataType` set to either `'SIB1NB'` or `'BCCHNotSIB1NB'` indicates that the NPDSCH is carrying BCCH; `NPDSCHDataType` set to `'NotBCCH'` indicates that the NPDSCH is not carrying BCCH.

```
NPDSCHDataType = 'NotBCCH';  % The allowed values are 'SIB1NB', 'BCCHNotSIB1NB' or 'NotBCCH'
```

- The variable `ISF` configures the number of subframes for a NPDSCH according to 3GPP TS 36.213 Table 16.4.1.3-1 [2]. Valid values for `ISF` are 0...7.

When the NPDSCH carries the SIB1-NB:

- The variable `SchedulingInfoSIB1` configures the number of NPDSCH repetitions according to 3GPP TS 36.213 Table 16.4.1.3-3 and the TBS according to Table 16.4.1.5.2-1 [2]. Valid values for `SchedulingInfoSIB1` are 0...11.

When the NPDSCH does not carry the SIB1-NB:

- The variable `IRep` configures the number of NPDSCH repetitions according to 3GPP TS 36.213 Table 16.4.1.3-2 [2]. Valid values for `IRep` are 0...15.
- The variable `IMCS` together with `IRep` configure the TBS according to 3GPP TS 36.213 Table 16.4.1.5.1-1 [2]. Valid values for `IMCS` are 0...13.

```
ISF = 0;                % Resource assignment field in DCI (DCI format N1 or N2)
SchedulingInfoSIB1 = 0; % Scheduling information field in MasterInformationBlock-NB (MIB-NB)
IMCS = 4;               % Modulation and coding scheme field in DCI (DCI format N1 or N2)
```

eNB Configuration

Configure the starting frame and subframe numbers (`enb.NFrame` and `enb.NSubframe`) in the simulation for each SNR point, the narrowband physical cell ID `enb.NNCellID`, the number of NRS antenna ports (`enb.NBRefP`, one antenna port indicates port 2000 is used, two antenna ports indicates port 2000 and port 2001 are used), the NB-IoT operation mode `enb.OperationMode` which can be any value as follows:

- `'Standalone'`: NB-IoT carrier deployed outside the LTE spectrum, e.g. the spectrum used for GSM or satellite communications
- `'Guardband'`: NB-IoT carrier deployed in the guardband between two LTE carriers
- `'Inband-SamePCI'`: NB-IoT carrier deployed in resource blocks of a LTE carrier, with `enb.NBRefP` the same as the number of CRS ports `enb.CellRefP`
- `'Inband-DifferentPCI'`: NB-IoT carrier deployed in resource blocks of a LTE carrier, with `enb.NBRefP` different as `enb.CellRefP`

`enb.CellRefP` is configured when the operation mode is `'Inband-DifferentPCI'`. The starting OFDM symbol index in a subframe for NPDSCH is configured using `enb.ControlRegionSize`, when the values of `NPDSCHDataType` and `enb.OperationMode` satisfy the following conditions:

- `NPDSCHDataType` is either `'BCCHNotSIB1NB'` or `'NotBCCH'`
- `enb.OperationMode` is either `'Inband-SamePCI'` or `'Inband-DifferentPCI'`

```
enb.NFrame = 0;     % Simulation starting frame number
enb.NSubframe = 0;  % Simulation starting subframe number
enb.NNCellID = 0;   % NB-IoT physical cell ID
enb.NBRefP = 2;     % Number of NRS antenna ports, should be either 1 or 2
enb.OperationMode = 'Inband-DifferentPCI';  % The allowed values are 'Inband-SamePCI', 'Inband-Di
if strcmpi(enb.OperationMode,'Inband-SamePCI')
    enb.CellRefP = enb.NBRefP;     % The allowed values are NBRefP or 4
    enb.NCellID = enb.NNCellID;
elseif strcmpi(enb.OperationMode,'Inband-DifferentPCI')
    enb.CellRefP = 4; % Number of Cell RS antenna ports (Must be equal to NBRefP or 4)
    enb.NCellID = 1;
end
if (strcmpi(NPDSCHDataType,'BCCHNotSIB1NB') || strcmpi(NPDSCHDataType,'NotBCCH')) && ...
        (strcmpi(enb.OperationMode,'Inband-SamePCI') || strcmpi(enb.OperationMode,'Inband-Differe
    enb.ControlRegionSize = 3;     % The allowed values are 0...13
end
```

Propagation Channel Model Configuration

The structure `channel` contains the channel model configuration parameters.

```
channel = struct;                    % Initialize channel config structure
channel.Seed = 6;                    % Channel seed
channel.NRxAnts = 1;                 % 1 receive antenna
channel.DelayProfile ='EPA';         % Delay profile
channel.DopplerFreq = 5;             % Doppler frequency in Hz
channel.MIMOCorrelation = 'Low';     % Multi-antenna correlation
channel.NTerms = 16;                 % Oscillators used in fading model
channel.ModelType = 'GMEDS';         % Rayleigh fading model type
channel.InitPhase = 'Random';        % Random initial phases
channel.NormalizePathGains = 'On';   % Normalize delay profile power
channel.NormalizeTxAnts = 'On';      % Normalize for transmit antennas
```

Channel Estimator Configuration

In this example the parameter `perfectChannelEstimator` controls channel estimator behavior. Valid values are `true` or `false`. When set to `true`, a perfect channel estimator is used otherwise a practical estimator is used, based on the values of the received NRS.

```
% Channel estimator behavior
perfectChannelEstimator = true;
```

The practical channel estimator is configured with a structure `cec`. An EPA delay profile with 5Hz Doppler causes the channel to change slowly over time. Therefore only frequency averaging is performed over pilot estimates by setting the time window to 1 Resource Element (RE) and frequency window to 25 to ensure averaging over all subcarriers for the resource block.

```
% Configure channel estimator
cec.PilotAverage = 'UserDefined';   % Type of pilot symbol averaging
cec.TimeWindow = 1;                 % Time window size in REs
cec.FreqWindow = 25;                % Frequency window size in REs
cec.InterpType = 'Cubic';           % 2D interpolation type
cec.InterpWindow = 'Centered';      % Interpolation window type
cec.InterpWinSize = 3;              % Interpolation window size
cec.Reference = 'NRS';              % Channel estimator reference signal
```

NPDSCH Configuration

Obtain the following NPDSCH parameters from the higher layer configurations defined above:

- The number of repetitions (`NRep`)
- The number of subframes used for a NPDSCH when there is no repetition (`NSF`)
- The transport block size (`TBS`)

These parameters can be obtained by using the class `hNPDSCHInfo`. `hNPDSCHInfo` also provides method `displaySubframePattern` to display the NPDSCH repetition pattern, which is shown in the next section.

```
for repIdx = 1:numel(ireps)

    npdschInfo = hNPDSCHInfo;
    npdschInfo.NPDSCHDataType = NPDSCHDataType;
    npdschInfo.ISF = ISF;
    if strcmpi(NPDSCHDataType,'SIB1NB')  % NPDSCH carrying SIB1-NB
```

```
        npdschInfo.SchedulingInfoSIB1 = SchedulingInfoSIB1;
    else % NPDSCH not carrying SIB1-NB
        npdschInfo.IRep = ireps(repIdx); % Repetition number field in DCI (DCI format N1 or N2)
        npdschInfo.IMCS = IMCS;          % Modulation and coding scheme field in DCI (DCI format
    end
```

Create the structure `npdsch` using the obtained number of repetitions (`npdschInfo.NRep`), the number of subframes of a NPDSCH (`npdschInfo.NSF`) from the class instance `npdschInfo`, input parameter `NPDSCHDataType` and the Radio Network Temporary Identifier RNTI. Note that `NSF = 8` is used when `NPDSCHDataType` is `'SIB1NB'`.

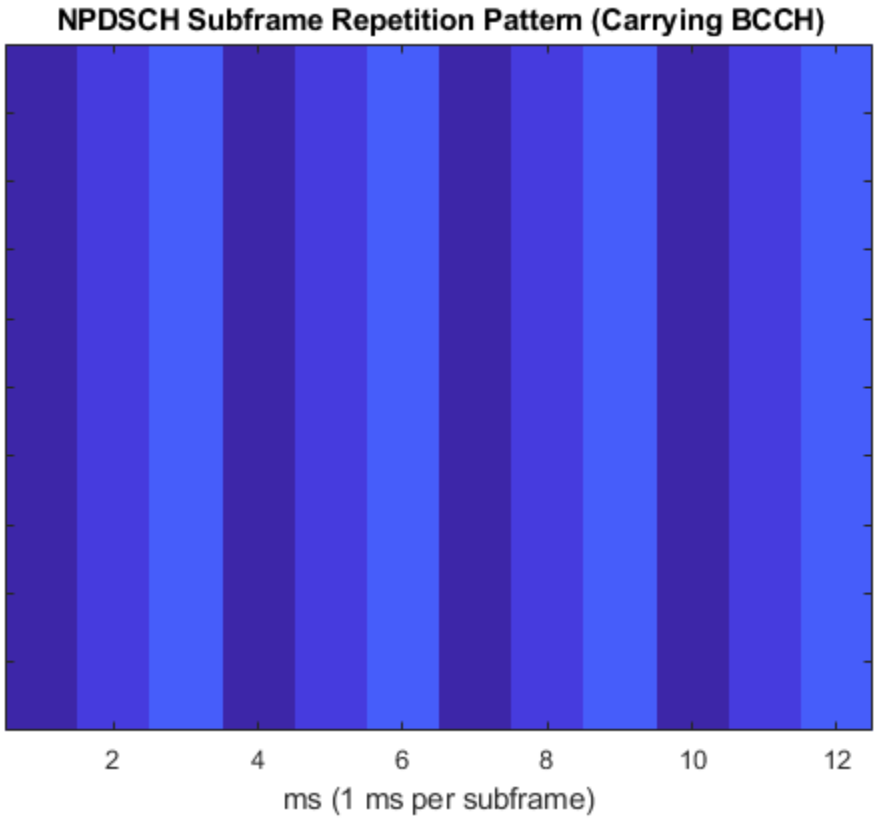
```
npdsch.NSF = npdschInfo.NSF;
npdsch.NRep = npdschInfo.NRep;
npdsch.NPDSCHDataType = NPDSCHDataType;
npdsch.RNTI = 1;
```

Compute codeword length and transport block size.

```
[~,info] = lteNPDSCHIndices(enb,npdsch);
rmoutlen = info.G;           % Bit length after rate matching, i.e. codeword length
trblklen = npdschInfo.TBS;   % Transport block size
```

Display Subframe Repetition Pattern

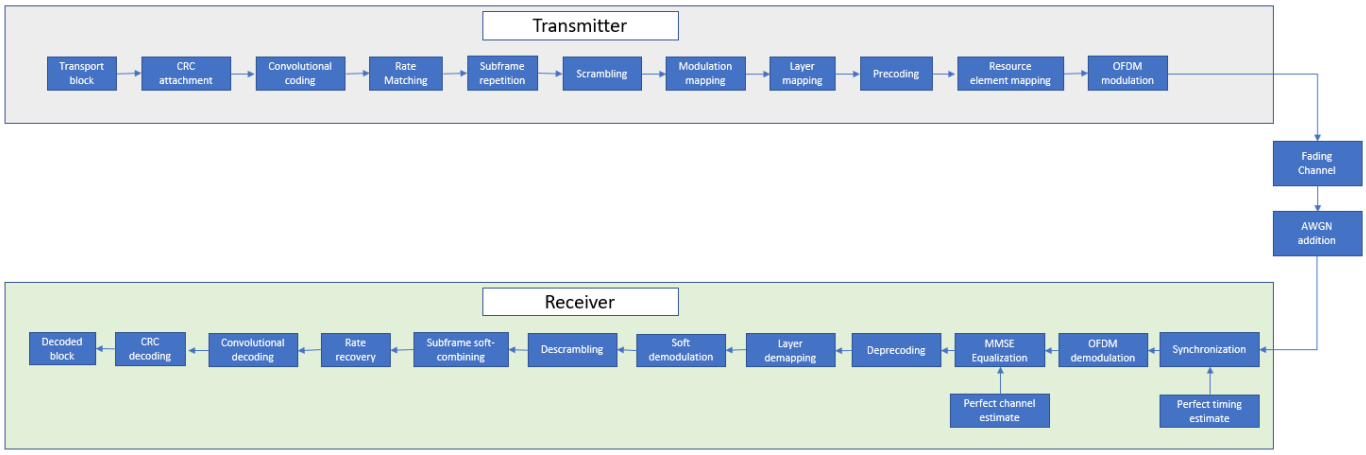
The variable `displayPattern` controls the display of the NPDSCH subframe repetition pattern. An example is shown in the following figure for the case when the NPDSCH carries the BCCH, the NPDSCH consists of `npdschInfo.NSF = 3` different subframes, each color represents a subframe which represents 1 ms. Each subframe is repeated `npdschInfo.NRep = 4` times, thus a total of 12 subframes are required to transmit the NPDSCH.




```
%  The NPDSCH repetition pattern for the current configuration is
%  displayed below
displayPattern = false;
% Display NPDSCH repetition pattern
if displayPattern == true
    npdschInfo.displaySubframePattern;
end
```

Block Error Rate Simulation Loop

This part of the example shows how to perform NB-IoT NPDSCH link level simulation and plot BLER results. The transmit and receive chain is depicted in the following figure.



A random stream of bits with the size of the desired transport block undergoes CRC encoding, convolutional encoding and rate matching to obtain the NPDSCH bits, which are repeated according to a specific subframe repetition pattern. Scrambling, modulation, layer mapping and precoding are then applied to form the complex NPDSCH symbols. These symbols along with the NRS signals are mapped to the grid and OFDM modulated to create the time domain waveform. This is then passed through a fading channel and AWGN is added. The noisy waveform is then synchronized and demodulated. Channel estimation and equalization is performed on the recovered NPDSCH symbols after which channel decoding and demodulation are performed to recover the transport block. After de-scrambling, the repetitive subframes are soft-combined before rate recover. The transport block error rate is calculated for each SNR point. The evaluation of the block error rate is based on the assumption that all the subframes in a bundle is used to decode the transport block at the UE. A bundle is defined in the MAC layer (see 3GPP TS 36.321 5.3.2.1 [3]) as the `npdsch.NSF` × `npdsch.NRep` subframes used to carry a transport block.

```
% Absolute subframe number at the starting point of the simulation
NSubframe = enb.NFrame*10+enb.NSubframe;

% Initialize BLER and throughput result
maxThroughput = zeros(length(SNRdB),1);
simThroughput = zeros(length(SNRdB),1);
bler = zeros(1,numel(SNRdB));

% The temporary variables 'enb_init' and 'channel_init' are used to create
% the temporary variable 'enb' and 'channel' within the SNR loop to create
% independent simulation loops for the 'parfor' loop
enb_init = enb;
channel_init = channel;

for snrIdx = 1:numel(SNRdB)
% parfor snrIdx = 1:numel(SNRdB)
% To enable the use of parallel computing for increased speed comment out
% the 'for' statement above and uncomment the 'parfor' statement below.
% This needs the Parallel Computing Toolbox. If this is not installed
% 'parfor' will default to the normal 'for' statement.

    % Set the random number generator seed depending to the loop variable
    % to ensure independent random streams
    rng(snrIdx,'combRecursive');
```

```
fprintf('\nSimulating %d transport blocks at %gdB SNR\n',numTrBlks,SNRdB(snrIdx));

enb = enb_init;         % Initialize eNodeB configuration
channel = channel_init; % Initialize fading channel configuration
txcw = [];              % Initialize the transmitted codeword
numBlkErrors = 0;       % Number of transport blocks with errors
estate = [];            % Initialize NPDSCH encoder state
dstate = [];            % Initialize NPDSCH decoder state
lastOffset = 0;         % Initialize overall frame timing offset
offset = 0;             % Initialize frame timing offset
subframeGrid = lteNBResourceGrid(enb); % Initialize the subframe grid

subframeIdx = NSubframe;
numRxTrBlks = 0;
while (numRxTrBlks < numTrBlks)

    % Set current subframe and frame numbers
    enb.NSubframe = mod(subframeIdx,10);
    enb.NFrame = floor((subframeIdx)/10);

    % Generate the NPSS symbols and indices
    npssSymbols = lteNPSS(enb);
    npssIndices = lteNPSSIndices(enb);
    % Map the symbols to the subframe grid
    subframeGrid(npssIndices) = npssSymbols;

    % Generate the NSSS symbols and indices
    nsssSymbols = lteNSSS(enb);
    nsssIndices = lteNSSSIndices(enb);
    % Map the symbols to the subframe grid
    subframeGrid(nsssIndices) = nsssSymbols;

    % Establish if either NPSS or NSSS is transmitted and if so,
    % do not transmit NPDSCH in this subframe
    isDataSubframe = isempty(npssSymbols) && isempty(nsssSymbols);

    % Create a new transport block and encode it when the
    % transmitted codeword is empty. The receiver sets the codeword
    % to empty to signal that all subframes in a bundle have been
    % received (it is also empty before the first transmission)
    if isempty(txcw)
        txTrBlk = randi([0 1],trblklen,1);
        txcw = lteNDLSCH(rmoutlen,txTrBlk);
    end

    if (isDataSubframe)
        % Generate NPDSCH symbols and indices for a subframe
        [txNpdschSymbols,estate] = lteNPDSCH(enb,npdsch,txcw,estate);
        npdschIndices = lteNPDSCHIndices(enb,npdsch);
        % Map the symbols to the subframe grid
        subframeGrid(npdschIndices) = txNpdschSymbols;
        % Generate the NRS symbols and indices
        nrsSymbols = lteNRS(enb);
        nrsIndices = lteNRSIndices(enb);
        % Map the symbols to the subframe grid
        subframeGrid(nrsIndices) = nrsSymbols;
    end
```

```
% Perform OFDM modulation to generate the time domain waveform
[txWaveform,ofdmInfo] = nbOFDMModulate(enb,subframeGrid);

% Add 25 sample padding. This is to cover the range of delays
% expected from channel modeling (a combination of
% implementation delay and channel delay spread)
txWaveform =  [txWaveform; zeros(25, enb.NBRefP)]; %#ok<AGROW>

% Initialize channel time for each subframe
channel.InitTime = subframeIdx/1000;

% Pass data through channel model
channel.SamplingRate = ofdmInfo.SamplingRate;
[rxWaveform,fadingInfo] = lteFadingChannel(channel, txWaveform);

% Calculate noise gain including compensation for downlink power
% allocation
SNR = 10^(SNRdB(snrIdx)/10);

% Normalize noise power to take account of sampling rate, which
% is a function of the IFFT size used in OFDM modulation, and
% the number of antennas
N0 = 1/sqrt(2.0*enb.NBRefP*double(ofdmInfo.Nfft)*SNR);

% Create additive white Gaussian noise
noise = N0*complex(randn(size(rxWaveform)), ...
                   randn(size(rxWaveform)));

% Add AWGN to the received time domain waveform
rxWaveform = rxWaveform + noise;

%-----------------------------------------------------------------
%            Receiver
%-----------------------------------------------------------------

% Perform timing synchronization, extract the appropriate
% subframe of the received waveform, and perform OFDM
% demodulation
if(perfectChannelEstimator)
    offset = hPerfectTimingEstimate(fadingInfo);
else
    % In this example, the subframe offset calculation relies
    % on NPSS present in subframe 5, so we need to pad the
    % subframes before it so that the frame offset returned by
    % lteNBDLFrameOffset is the offset for subframe 5
    sfTsamples = ofdmInfo.SamplingRate*1e-3;
    if (enb.NSubframe==5)
        padding = zeros([sfTsamples*5,size(rxWaveform,2)]);
        offset = lteNBDLFrameOffset(enb, [padding; rxWaveform]);
        if (offset > 25) || (offset < 0)
            offset = lastOffset;
        end
        lastOffset = offset;
    end
end

% Synchronize the received waveform
rxWaveform = rxWaveform(1+offset:end, :);
```

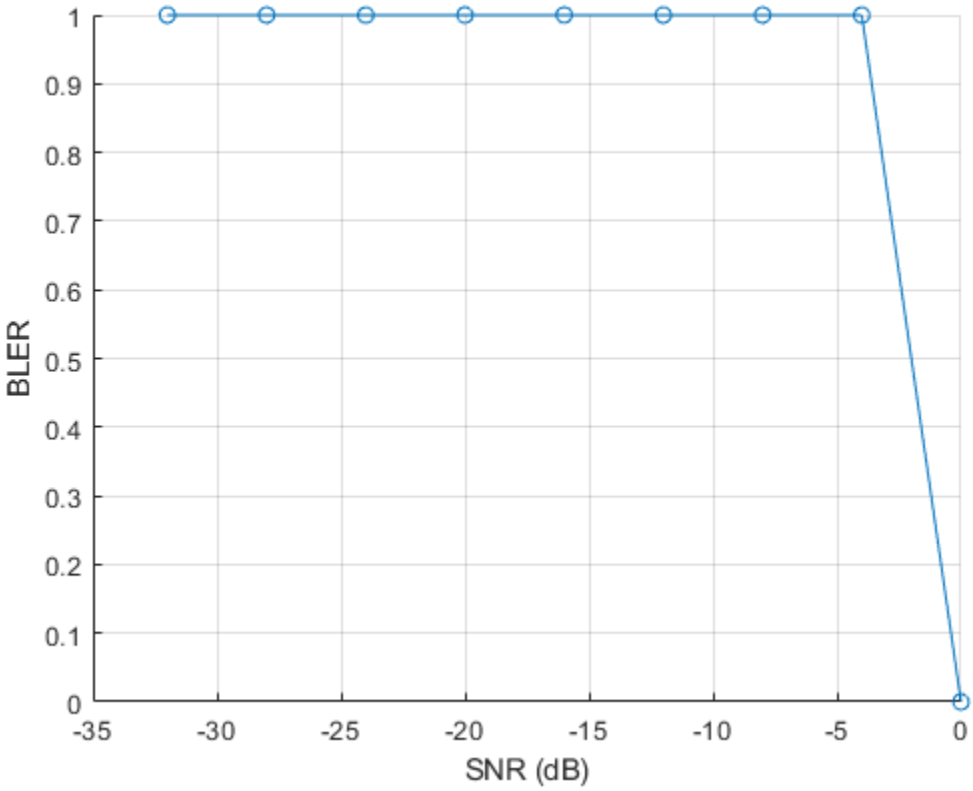
```
            % Perform OFDM demodulation on the received data to recreate the
            % resource grid
            rxSubframe = nbOFDMDemodulate(enb,rxWaveform);

            % Channel estimation
            if(perfectChannelEstimator)
                % Perfect channel estimation
                estChannelGrid = nbDLPerfectChannelEstimate(enb, channel, offset);
                noiseGrid = nbOFDMDemodulate(enb, noise(1+offset:end ,:));
                noiseEst = var(noiseGrid(:));
            else

                [estChannelGrid, noiseEst] = lteDLChannelEstimate( ...
                enb, cec, rxSubframe);
            end

            if (isDataSubframe)
                % Get NPDSCH indices
                npdschIndices = lteNPDSCHIndices(enb, npdsch);

                % Get PDSCH resource elements from the received subframe. Scale the
                % received subframe by the PDSCH power factor Rho. The PDSCH is
                % scaled by this amount, while the cell reference symbols used for
                % channel estimation (used in the PDSCH decoding stage) are not.
                [rxNpdschSymbols, npdschHest] = lteExtractResources(npdschIndices, ...
                    rxSubframe, estChannelGrid);

                % Decode NPDSCH
                [rxcw,dstate,symbols] = lteNPDSCHDecode(...
                                    enb, npdsch, rxNpdschSymbols, npdschHest, noiseEst,dstate);

                % Decode the transport block when all the subframes in a bundle
                % have been received
                if dstate.EndOfTx
                   [trblkout,blkerr] = lteNDLSCHDecode(trblklen,rxcw);
                   numBlkErrors = numBlkErrors + blkerr;
                   numRxTrBlks = numRxTrBlks + 1;
                   % Re-initialize to enable the transmission of a new transport block
                   txcw = [];
                end
            end

            subframeIdx = subframeIdx + 1;

        end

        % Calculate the block error rate
        bler(snrIdx) = numBlkErrors/numTrBlks;
        fprintf('NPDSCH BLER = %.4f \n',bler(snrIdx));
        % Calculate the maximum and simulated throughput
        maxThroughput(snrIdx) = trblklen*numTrBlks; % Max possible throughput
        simThroughput(snrIdx) = trblklen*(numTrBlks-numBlkErrors);  % Simulated throughput
        fprintf('NPDSCH Throughput(%%) = %.4f %%\n',simThroughput(snrIdx)*100/maxThroughput(snrId

    end
```

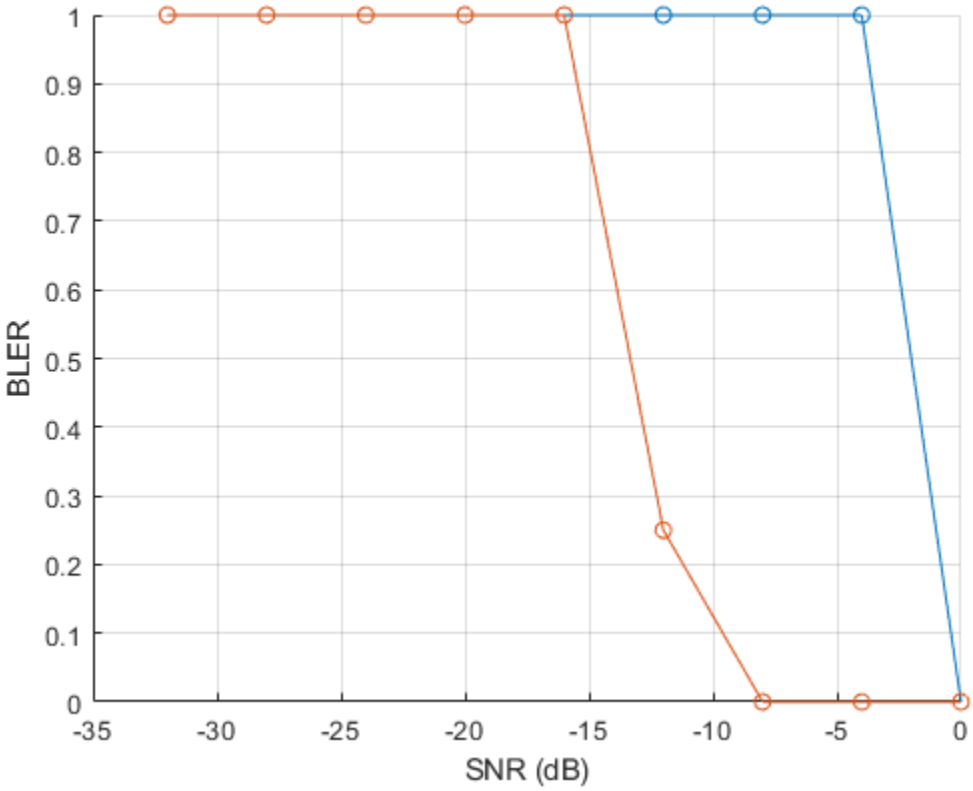
```
Simulating 4 transport blocks at -32dB SNR


Simulating 4 transport blocks at -32dB SNR
NPDSCH BLER = 1.0000
NPDSCH Throughput(%) = 0.0000 %

Simulating 4 transport blocks at -28dB SNR
NPDSCH BLER = 1.0000
NPDSCH Throughput(%) = 0.0000 %

Simulating 4 transport blocks at -24dB SNR
NPDSCH BLER = 1.0000
NPDSCH Throughput(%) = 0.0000 %

Simulating 4 transport blocks at -20dB SNR
NPDSCH BLER = 1.0000
NPDSCH Throughput(%) = 0.0000 %

Simulating 4 transport blocks at -16dB SNR
NPDSCH BLER = 1.0000
NPDSCH Throughput(%) = 0.0000 %

Simulating 4 transport blocks at -12dB SNR
NPDSCH BLER = 0.2500
NPDSCH Throughput(%) = 75.0000 %

Simulating 4 transport blocks at -8dB SNR
NPDSCH BLER = 0.0000
NPDSCH Throughput(%) = 100.0000 %

Simulating 4 transport blocks at -4dB SNR
NPDSCH BLER = 0.0000
NPDSCH Throughput(%) = 100.0000 %

Simulating 4 transport blocks at 0dB SNR
NPDSCH BLER = 0.0000
NPDSCH Throughput(%) = 100.0000 %


Simulating 4 transport blocks at -32dB SNR
NPDSCH BLER = 1.0000
NPDSCH Throughput(%) = 0.0000 %

Simulating 4 transport blocks at -28dB SNR
NPDSCH BLER = 1.0000
NPDSCH Throughput(%) = 0.0000 %

Simulating 4 transport blocks at -24dB SNR
NPDSCH BLER = 0.2500
NPDSCH Throughput(%) = 75.0000 %

Simulating 4 transport blocks at -20dB SNR
NPDSCH BLER = 0.0000
NPDSCH Throughput(%) = 100.0000 %

Simulating 4 transport blocks at -16dB SNR
```

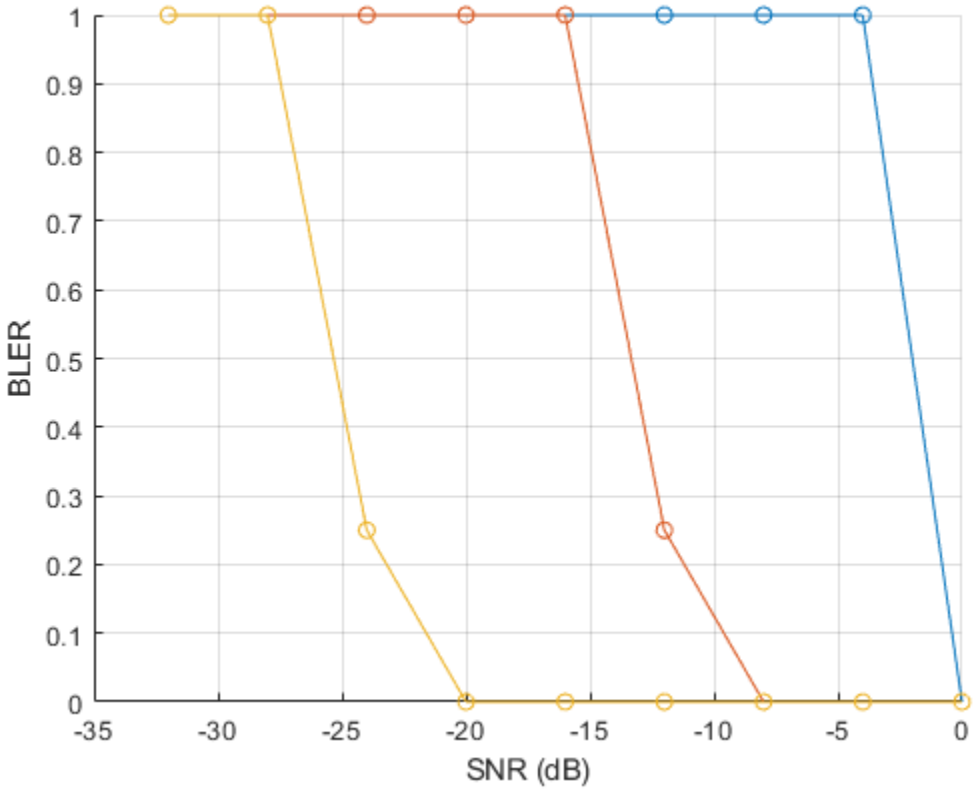
```
NPDSCH BLER = 0.0000
NPDSCH Throughput(%) = 100.0000 %

Simulating 4 transport blocks at -12dB SNR
NPDSCH BLER = 0.0000
NPDSCH Throughput(%) = 100.0000 %

Simulating 4 transport blocks at -8dB SNR
NPDSCH BLER = 0.0000
NPDSCH Throughput(%) = 100.0000 %

Simulating 4 transport blocks at -4dB SNR
NPDSCH BLER = 0.0000
NPDSCH Throughput(%) = 100.0000 %

Simulating 4 transport blocks at 0dB SNR
NPDSCH BLER = 0.0000
NPDSCH Throughput(%) = 100.0000 %
```

Plot Block Error Rate vs SNR results

```
    if repIdx == 1
        fh = figure;
        grid on;
        hold on;
        xlabel('SNR (dB)');
        ylabel('BLER');
        legendstr = {['NRep = ' num2str(npdsch.NRep)]};
    else
        legendstr = [legendstr ['NRep = ' num2str(npdsch.NRep)]]; %#ok<AGROW>
    end
    figure(fh);
    plot(SNRdB, bler, '-o');
```

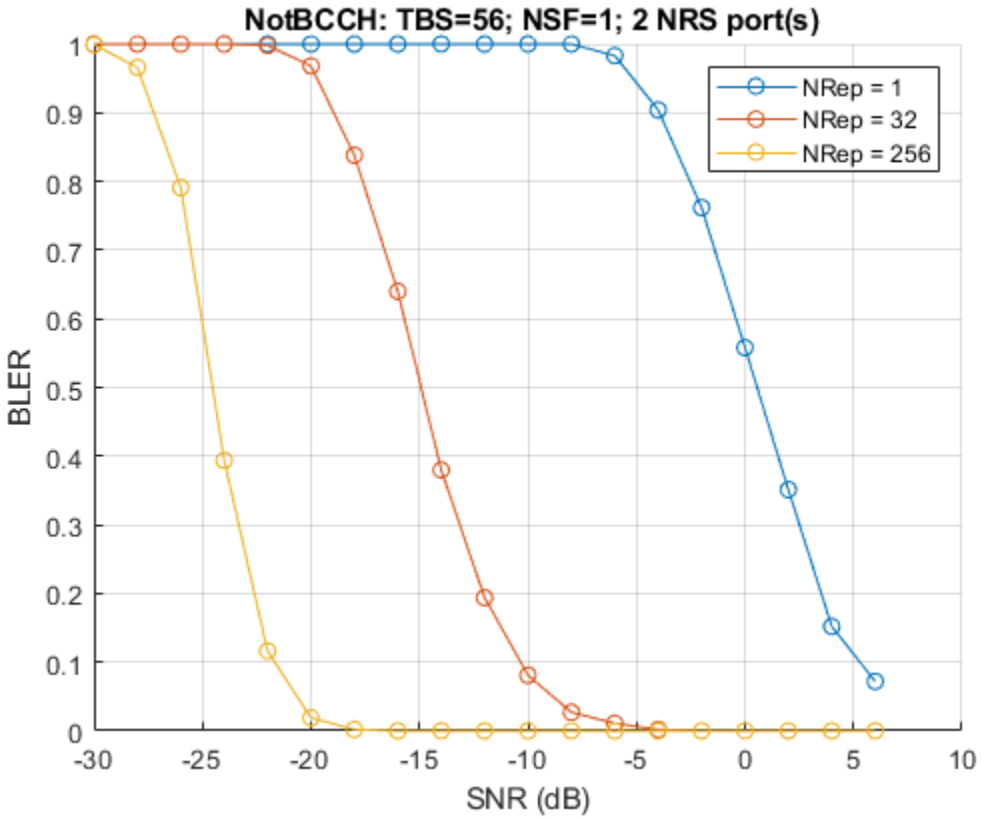

BLER
SNR (dB)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-35
-30
-25
-20
-15
-10
-5
0


BLER
SNR (dB)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-35
-30
-25
-20
-15
-10
-5
0




```
end
% Set figure title
if strcmpi(NPDSCHDataType,'SIB1NB')
    npdsch.NSF = 8;
end
title([' ' char(npdsch.NPDSCHDataType) ': TBS=' num2str(trblklen)...
    '; NSF=' num2str(npdsch.NSF) '; ' num2str(enb_init.NBRefP) ' NRS port(s)' ]);
legend(legendstr);
```

The following plot shows the simulation run with `numTrBlks` set to 1000 while using the perfect channel estimator.



Appendix

This example uses the helper functions:

- hPerfectTimingEstimate.m
- hNPDSCHInfo.m

Selected Bibliography

1 3GPP TS 36.211 "Physical channels and modulation"

2 3GPP TS 36.213 "Physical layer procedures"

3 3GPP TS 36.321 "Medium Access Control (MAC) protocol specification"

4 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception"

Local functions

```
% NB-IoT DL OFDM Modulator
function [waveform,info] = nbOFDMModulate(enb,grid)
    % Apply default window size according to TS 36.104 Table E.5.1-1a
    if(~isfield(enb,'Windowing'))
        enb.Windowing = 6;
    end
    % Use NB-IoT SC-FDMA to get the 1/2 subcarrier shift on the OFDM modulation
    enb.NBULSubcarrierSpacing = '15kHz';
    [waveform,info] = lteSCFDMAModulate(enb,grid);
end
```

```
% NB-IoT DL OFDM Demodulator
function grid = nbOFDMDemodulate(enb,rxWaveform)
    % Use NB-IoT SC-FDMA to get the 1/2 subcarrier shift on the OFDM modulation
    enb.NBULSubcarrierSpacing = '15kHz';
    grid = lteSCFDMADemodulate(enb,rxWaveform,0.55); % CP fraction of 0.55
end

% NB-IoT DL Perfect Channel Estimator
function H = nbDLPerfectChannelEstimate(enb,channel,timefreqoffset)
    % Reconfigure NB-IoT UL perfect channel estimator to perform DL perfect
    % channel estimation
    enb.NBULSubcarrierSpacing = '15kHz';
    enb.NTxAnts = enb.NBRefP;
    enb.TotSlots = 2;
    H = lteULPerfectChannelEstimate(enb, channel,timefreqoffset);
end

NPDSCH BLER = 1.0000
NPDSCH Throughput(%) = 0.0000 %

Simulating 4 transport blocks at -28dB SNR
NPDSCH BLER = 1.0000
NPDSCH Throughput(%) = 0.0000 %

Simulating 4 transport blocks at -24dB SNR
NPDSCH BLER = 1.0000
NPDSCH Throughput(%) = 0.0000 %

Simulating 4 transport blocks at -20dB SNR
NPDSCH BLER = 1.0000
NPDSCH Throughput(%) = 0.0000 %

Simulating 4 transport blocks at -16dB SNR
NPDSCH BLER = 1.0000
NPDSCH Throughput(%) = 0.0000 %

Simulating 4 transport blocks at -12dB SNR
NPDSCH BLER = 1.0000
NPDSCH Throughput(%) = 0.0000 %

Simulating 4 transport blocks at -8dB SNR
NPDSCH BLER = 1.0000
NPDSCH Throughput(%) = 0.0000 %

Simulating 4 transport blocks at -4dB SNR
NPDSCH BLER = 1.0000
NPDSCH Throughput(%) = 0.0000 %

Simulating 4 transport blocks at 0dB SNR
NPDSCH BLER = 0.0000
NPDSCH Throughput(%) = 100.0000 %
```

NB-IoT NPUSCH Block Error Rate Simulation

This example shows how to create an NB-IoT Narrowband Physical Uplink Shared Channel (NPUSCH) Block Error Rate (BLER) simulation in frequency-selective fading and Additive White Gaussian Noise (AWGN) using LTE Toolbox™.

Introduction

3GPP introduced a new air interface, Narrowband IoT (NB-IoT), optimized for low data rate machine-type communications in LTE-Advanced Pro Release 13. NB-IoT provides cost and power efficiency improvements as it avoids the need for complex signaling overhead required for LTE-based systems.

The example generates an NB-IoT NPUSCH BLER curve for a number of SNR points and transmission parameters. NPUSCH and narrowband demodulation reference signal (DRS) are transmitted in all slots. Operating on a slot-by-slot basis for each SNR point, the BLER calculation comprises these steps:

- Generate a resource grid and populate it with NPUSCH symbols
- Create the baseband waveform by SC-FDMA modulating the grid
- Pass the waveform through a noisy fading channel
- Perform receiver operations (SC-FDMA demodulation, channel estimation, and equalization)
- Obtain the block CRC by decoding the equalized symbols
- Determine the performance of the NPUSCH by using the block CRC result at the output of the channel decoder

Simulation Configuration

The simulation length is 5 UL-SCH transport blocks for a number of SNR points `SNRdB` for different repetitions `simReps`. To produce meaningful throughput results, you should use a larger number of transport blocks (`numTrBlks`). `SNRdB` and `simReps` can be a specified as a scalar or a numeric array.

```
numTrBlks = 5;              % Number of simulated transport blocks
SNRdB = [-20 -18 -15 -12.5 -10 -6.4 -3.5 0.7];  % Range of SNR values in dB
simReps = [2 16 64];        % Repetitions to simulate
```

NPUSCH Configuration

In this section we configure the parameters required for NPUSCH generation. There are two types of payload defined for NPUSCH transmission, format 1 ('Data') and format 2 ('Control'). For format 1, the UE uses the combination of modulation and coding scheme (MCS) and resource assignment signaled via the DCI to determine the transport block size from the set defined in TS 36.213 Table 16.5.1.2-2. For format 2, the NPUSCH carries the 1 bit ACK/NACK. The `chs.NPUSCHFormat` parameter specifies the format and `infoLen` specifies the transport block length. The parameters used in this example are as per the A16-5 FRC defined in TS 36.104 Annex A.16.

HARQ Operation NB-IoT has one or two UL HARQ processes and HARQ operation is asynchronous for NB-IoT UEs except for the repetitions within a bundle. Bundling operation relies on the HARQ entity for invoking the same HARQ process for each transmission that is part of the same bundle. Within a bundle, HARQ retransmissions are non-adaptive. They are triggered without waiting for feedback from the reception of previous repetitions. An uplink grant corresponding to a new transmission or a retransmission of the bundle is only received after the last repetition of the bundle.

A retransmission of a bundle is also a bundle. For more details, see TS 36.321 section 5.4.2. In this example the bundle retransmissions are not modeled.

```
ue = struct();                          % Initialize the UE structure
ue.NBULSubcarrierSpacing = '15kHz';     % 3.75kHz, 15kHz
ue.NNCellID = 10;                       % Narrowband cell identity

chs = struct();
chs.NPUSCHFormat = 'Data';        % NPUSCH payload type ('Data' or 'Control')
% The number of subcarriers used for NPUSCH 'NscRU' depends on the NPUSCH
% format and subcarrier spacing 'NBULSubcarrierSpacing' as shown in TS 36.211
% Table 10.1.2.3-1. There are 1,3,6 or 12 continuous subcarriers for NPUSCH
chs.NBULSubcarrierSet = 0:11;     % Range is 0-11 (15kHz); 0-47 (3.75kHz)
chs.NRUsc = length(chs.NBULSubcarrierSet);
% The symbol modulation depends on the NPUSCH format and NscRU as given by
% TS 36.211 Table 10.1.3.2-1
chs.Modulation = 'QPSK';
chs.CyclicShift = 0;        % Cyclic shift required when NRUsc = 3 or 6
chs.RNTI = 20;              % RNTI value
chs.NLayers = 1;            % Number of layers
chs.NRU = 1;                % Number of resource units
chs.SlotIdx = 0;            % The slot index
chs.NTurboDecIts = 5;       % Number of turbo decoder iterations
chs.CSI = 'On';             % Use channel CSI in PUSCH decoding

% RV offset signaled via DCI (See 36.213 16.5.1.2)
rvDCI = 0;
% Calculate the RVSeq used according to the RV offset
rvSeq = [2*mod(rvDCI+0,2)  2*mod(rvDCI+1,2)];

if strcmpi(chs.NPUSCHFormat,'Data')
    infoLen = 136;   % Transport block size for NPUSCH format 1
elseif strcmpi(chs.NPUSCHFormat,'Control')
    infoLen = 1;    % ACK/NACK bit for NPUSCH format 2
end
```

Propagation Channel Model Configuration

The structure `channel` contains the channel model configuration parameters.

```
channel = struct;                   % Initialize channel config structure
channel.Seed = 6;                   % Channel seed
channel.NRxAnts = 2;                % 2 receive antennas
channel.DelayProfile ='ETU';        % Delay profile
channel.DopplerFreq = 1;            % Doppler frequency in Hz
channel.MIMOCorrelation = 'Low';    % Multi-antenna correlation
channel.NTerms = 16;                % Oscillators used in fading model
channel.ModelType = 'GMEDS';        % Rayleigh fading model type
channel.InitPhase = 'Random';       % Random initial phases
channel.NormalizePathGains = 'On';  % Normalize delay profile power
channel.NormalizeTxAnts = 'On';     % Normalize for transmit antennas
```

Channel Estimator Configuration

In this example, the parameter `perfectChannelEstimator` controls channel estimator behavior. Valid values are `true` or `false`. When set to `true`, a perfect channel estimator is used. Otherwise a practical estimator is used, based on the values of the received NPUSCH DRS.

```
% Channel estimator behavior
perfectChannelEstimator = true;
```

The structure `cec` configures the practical channel estimator. An ETU delay profile with 1Hz Doppler causes the channel to change slowly over time. To ensure averaging over all subcarriers for the resource block, set the frequency window to 23 Resource Elements (REs). The variable `channelEstimationLength` configures the number of slots over which channel estimates are averaged, see TS 36.104 Table A.16.1-1 for suggested values for different NPUSCH configurations.

```
% Configure channel estimator
cec.PilotAverage = 'UserDefined';   % Type of pilot symbol averaging
cec.TimeWindow = 1;                 % Time window size in REs
cec.FreqWindow = 23;                % Frequency window size in REs
cec.InterpType = 'Cubic';           % 2D interpolation type
channelEstimationLength = 1;        % Channel estimation length in ms
```

For DRS signals in NPUSCH format 1, sequence-group hopping can be enabled or disabled by the higher layer cell-specific parameter `groupHoppingEnabled`. Sequence-group hopping for a particular UE can be disabled through the higher layer parameter `groupHoppingDisabled` as described in TS 36.211 Section 10.1.4.1.3. In this example, we use the `SeqGroupHopping` parameter to enable or disable sequence-group hopping

```
chs.SeqGroupHopping = 'on'; % Enable/Disable Sequence-Group Hopping for UE
chs.SeqGroup = 0;           % Higher-layer parameter groupAssignmentNPUSCH

% Get number of time slots in a resource unit NULSlots according to
% TS 36.211 Table 10.1.2.3-1
if strcmpi(chs.NPUSCHFormat,'Data')
    if chs.NRUsc == 1
        NULSlots = 16;
    elseif any(chs.NRUsc == [3 6 12])
        NULSlots = 24/chs.NRUsc;
    else
        error('Invalid number of subcarriers. NRUsc must be one of 1,3,6,12');
    end
elseif strcmpi(chs.NPUSCHFormat,'Control')
    NULSlots = 4;
else
    error('Invalid NPUSCH Format (%s). NPUSCHFormat must be ''Data'' or ''Control''',chs.NPUSCHF
end
chs.NULSlots = NULSlots;
```

Block Error Rate Simulation Loop

To perform NB-IoT NPUSCH link level simulation and plot BLER results for a number of repetition levels, this example performs the following steps:

For NPUSCH format 1 transmission for UL data transfer:

- Generate a random stream of bits with the size of the desired transport block
- Perform CRC encoding, turbo encoding and rate matching to create the NPUSCH bits
- Interleave the bits per resource unit to apply a time-first mapping and create the NPUSCH codeword

For NPUSCH format 2 used for signaling HARQ feedback for NPDSCH:

- Perform bit repetition of the HARQ indicator to create the NPUSCH codeword

Then for either NPUSCH format:

- Perform scrambling, modulation, layer mapping and precoding on the codeword to form the complex NPUSCH symbols
- Map the NPUSCH symbols and the corresponding DRS to the resource grid
- Generate the time domain waveform by performing SC-FDMA modulation of the resource grid
- Pass the waveform through a fading channel with AWGN
- Recover the transmitted grid by performing synchronization, channel estimation and MMSE equalization
- Extract the NPUSCH symbols
- Recover the transport block by demodulating the symbols and channel decoding the resulting bit estimates

Note that if practical channel estimation is configured (`perfectChannelEstimator = false`), practical timing estimation based on NPUSCH DRS correlation will also be performed. The timing offset is initialized to zero, intended to represent the initial synchronization after NPRACH reception. The timing estimate is then updated whenever the peak of the NPUSCH DRS correlation is sufficiently strong.

After de-scrambling, the repetitive slots are soft-combined before rate recovery. The transport block error rate is calculated for each SNR point. The evaluation of the block error rate is based on the assumption that all the slots in a bundle are used to decode the transport block at the UE. A bundle is defined in the MAC layer (see 3GPP TS 36.321 5.4.2.1) as the `NPUSCH.NRU` × `NPUSCH.NULSlots` × `NPUSCH.NRep` slots used to carry a transport block.

```
% Get the slot grid and number of slots per frame
emptySlotGrid = lteNBResourceGrid(ue); % Initialize empty slot grid
slotGridSize = size(emptySlotGrid);
NSlotsPerFrame = 20/(slotGridSize(1)/12);

tSlot = 10e-3/NSlotsPerFrame; % Slot duration
symbolsPerSlot = slotGridSize(2); % Number of symbols per slot

% Get a copy of the configuration variables ue, chs and channel to create
% independent simulation parfor loops
ueInit = ue;
chsInit = chs;
channelInit = channel;

for repIdx = 1:numel(simReps)

    chsInit.NRep = simReps(repIdx); % Number of repetitions of the NPUSCH
    NSlotsPerBundle = chsInit.NRU*chsInit.NULSlots*chsInit.NRep; % Number of slots in a codeword
    TotNSlots = numTrBlks*NSlotsPerBundle;   % Total number of simulated slots

    % Initialize BLER and throughput result
    maxThroughput = zeros(length(SNRdB),1);
    simThroughput = zeros(length(SNRdB),1);
    bler = zeros(1,numel(SNRdB)); % Initialize BLER result

    for snrIdx = 1:numel(SNRdB)
    % parfor snrIdx = 1:numel(SNRdB)
```

```
% To enable the use of parallel computing for increased speed comment out
% the 'for' statement above and uncomment the 'parfor' statement below.
% This needs the Parallel Computing Toolbox (TM). If this is not installed
% 'parfor' will default to the normal 'for' statement.

    % Set the random number generator seed depending on the loop variable
    % to ensure independent random streams
    rng(snrIdx,'combRecursive');

    ue = ueInit;    % Initialize ue configuration
    chs = chsInit;  % Initialize chs configuration
    channel = channelInit; % Initialize fading channel configuration
    numBlkErrors = 0;  % Number of transport blocks with errors
    estate = struct('SlotIdx',chs.SlotIdx);  % Initialize NPUSCH encoder state
    dstate = estate;   % Initialize NPUSCH decoder state
    offset = 0;        % Initialize overall frame timing offset
    trblk = [];        % Initialize the transport block
    npuschHest = [];   % Initialize channel estimate
    noiseEst = [];     % Initialize noise estimate

    % Display the number of slots being generated
    fprintf('\nGenerating %d slots corresponding to %d transport block(s) at %gdB SNR\n',TotN

    for slotIdx = 0+(0:TotNSlots-1)
        % Calculate the frame number and slot number within the frame
        ue.NFrame = fix(slotIdx/NSlotsPerFrame);
        ue.NSlot = mod(slotIdx,NSlotsPerFrame);
        % Create the slot grid
        slotGrid = emptySlotGrid;

        if isempty(trblk)

           % Initialize transport channel decoder state
           dstateULSCH = [];

           if strcmpi(chs.NPUSCHFormat,'Data')
               % UL-SCH encoding is performed for the two RV values used for
               % transmitting the codewords. The RV sequence used is determined
               % from the rvDCI value signaled in the DCI and alternates
               % between 0 and 2 as given in TS 36.213 Section 16.5.1.2

               % Define the transport block which will be encoded to create the
               % codewords for different RV
               trblk = randi([0 1],infoLen,1);

               % Determine the coded transport block size
               [~, info] = lteNPUSCHIndices(ue,chs);
               outblklen = info.G;
               % Create the codewords corresponding to the two RV values used
               % in the first and second block, this will be repeated till all
               % blocks are transmitted
               chs.RV = rvSeq(1); % RV for the first block
               cw = lteNULSCH(chs,outblklen,trblk); % CRC and Turbo coding is repeated
               chs.RV = rvSeq(2); % RV for the second block
               cw = [cw lteNULSCH(chs,outblklen,trblk)]; %#ok<AGROW> % CRC and Turbo coding i
           else
               trblk = randi([0 1],1); % 1 bit ACK
               % For ACK, the same codeword is transmitted every block as
```

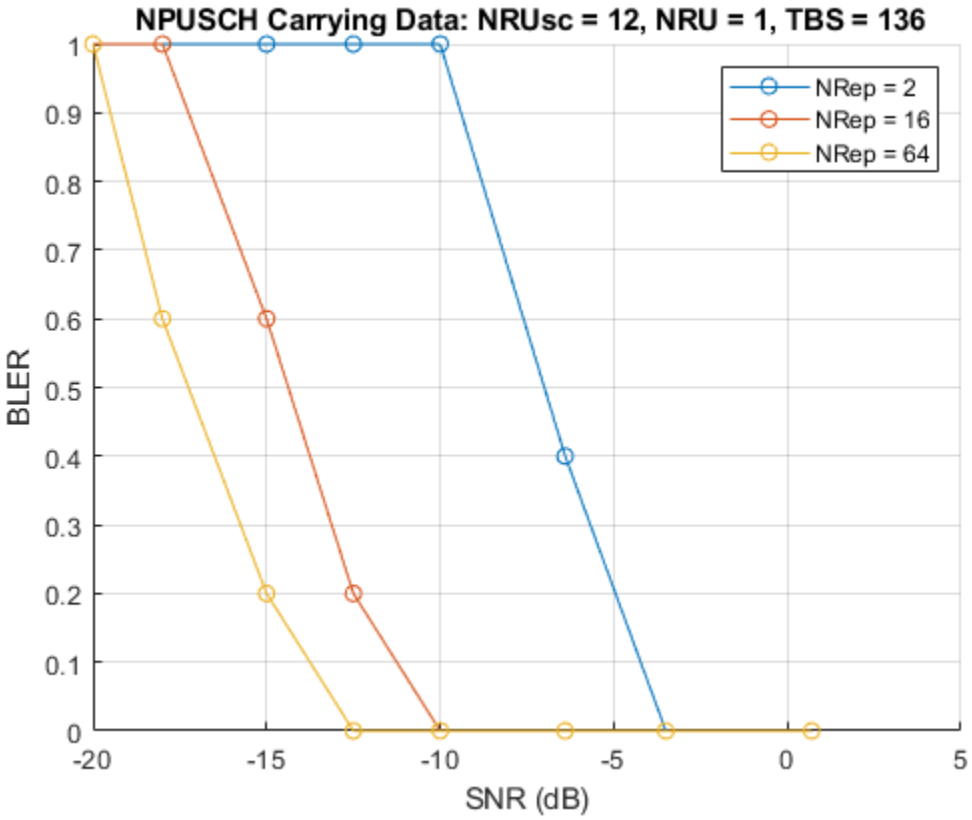
```
        % defined in TS 36.212 Section 6.3.3
        cw = lteNULSCH(trblk);
    end
    blockIdx = 0; % First block to be transmitted
end

% Copy SlotIdx for the SCFDMA modulator
chs.SlotIdx = estate.SlotIdx;

% Set the RV used for the current transport block
chs.RV = rvSeq(mod(blockIdx,size(rvSeq,2))+1);

% NPUSCH encoding and mapping onto the slot grid
txsym = lteNPUSCH(ue,chs,cw(:,mod(blockIdx,size(cw,2))+1),estate);
slotGrid(lteNPUSCHIndices(ue,chs)) = txsym;

% NPUSCH DRS and mapping on to the slot grid
[dmrs,estate] = lteNPUSCHDRS(ue,chs,estate);
slotGrid(lteNPUSCHDRSIndices(ue,chs)) = dmrs;

% If a full block is transmitted, increment the clock counter so that
% the correct codeword can be selected
if estate.EndOfBlk
    blockIdx = blockIdx + 1;
end

% Perform SC-FDMA modulation to create the time domain waveform
[txWaveform,scfdmaInfo] = lteSCFDMAModulate(ue,chs,slotGrid);

% Add 25 sample padding. This is to cover the range of delays
% expected from channel modeling (a combination of
% implementation delay and channel delay spread)
txWaveform =  [txWaveform; zeros(25, size(txWaveform,2))]; %#ok<AGROW>

% Initialize channel time for each slot
channel.InitTime = slotIdx*tSlot;

% Pass data through channel model
channel.SamplingRate = scfdmaInfo.SamplingRate;
[rxWaveform,fadingInfo] = lteFadingChannel(channel, txWaveform);

% Calculate noise gain
SNR = 10^(SNRdB(snrIdx)/10);

% Normalize noise power to take account of sampling rate, which is
% a function of the IFFT size used in SC-FDMA modulation
N0 = 1/sqrt(2.0*double(scfdmaInfo.Nfft)*SNR);

% Create additive white Gaussian noise
noise = N0*complex(randn(size(rxWaveform)), ...
                   randn(size(rxWaveform)));

% Add AWGN to the received time domain waveform
rxWaveform = rxWaveform + noise;

%----------------------------------------------------------------------
%           Receiver
%----------------------------------------------------------------------
```

```
% Perform timing synchronization, extract the appropriate
% subframe of the received waveform, and perform SC-FDMA
% demodulation
if (perfectChannelEstimator)
    offset = hPerfectTimingEstimate(fadingInfo);
else
    [t,mag] = lteULFrameOffsetNPUSCH(ue, chs, rxWaveform, dstate);
    % The function hSkipWeakTimingOffset is used to update the
    % receiver timing offset. If the correlation peak in 'mag'
    % is weak, the current timing estimate 't' is ignored and
    % the previous estimate 'offset' is used
    offset = hSkipWeakTimingOffset(offset,t,mag);
end

% Synchronize the received waveform
rxWaveform = rxWaveform(1+offset:end, :);

% Perform SC-FDMA demodulation on the received data to recreate
% the resource grid, including padding in the event that
% practical synchronization results in an incomplete slot being
% demodulated
rxSlot = lteSCFDMADemodulate(ue,chs,rxWaveform);
[K,L,R] = size(rxSlot);
if (L < symbolsPerSlot)
    rxSlot = cat(2,rxSlot,zeros(K,symbolsPerSlot-L,R));
end

% Channel estimation
if (perfectChannelEstimator)
    % Perfect channel estimation
    ue.TotSlots = 1; % Channel estimate for 1 slot
    estChannelGrid = lteULPerfectChannelEstimate(ue, chs, channel, offset);
    noiseGrid = lteSCFDMADemodulate(ue,chs,noise(1+offset:end ,:));
    noiseEstSlot = var(noiseGrid(:));
else
    [estChannelGrid, noiseEstSlot] = lteULChannelEstimateNPUSCH(ue, chs, cec, rxSlot
end

% Get NPUSCH indices
npuschIndices = lteNPUSCHIndices(ue,chs);

% Get NPUSCH resource elements from the received slot
[rxNpuschSymbols, npuschHestSlot] = lteExtractResources(npuschIndices, ...
    rxSlot, estChannelGrid);

% Perform channel estimate and noise estimate buffering in
% the case of practical channel estimation
if (perfectChannelEstimator)
    npuschHest = npuschHestSlot;
    noiseEst = noiseEstSlot;
else
    npuschHest = cat(3,npuschHest,npuschHestSlot);
    noiseEst = cat(1,noiseEst,noiseEstSlot);
    if (size(npuschHest,3) > channelEstimationLength)
        npuschHest = npuschHest(:,:,2:end);
        noiseEst = noiseEst(2:end);
    end
```

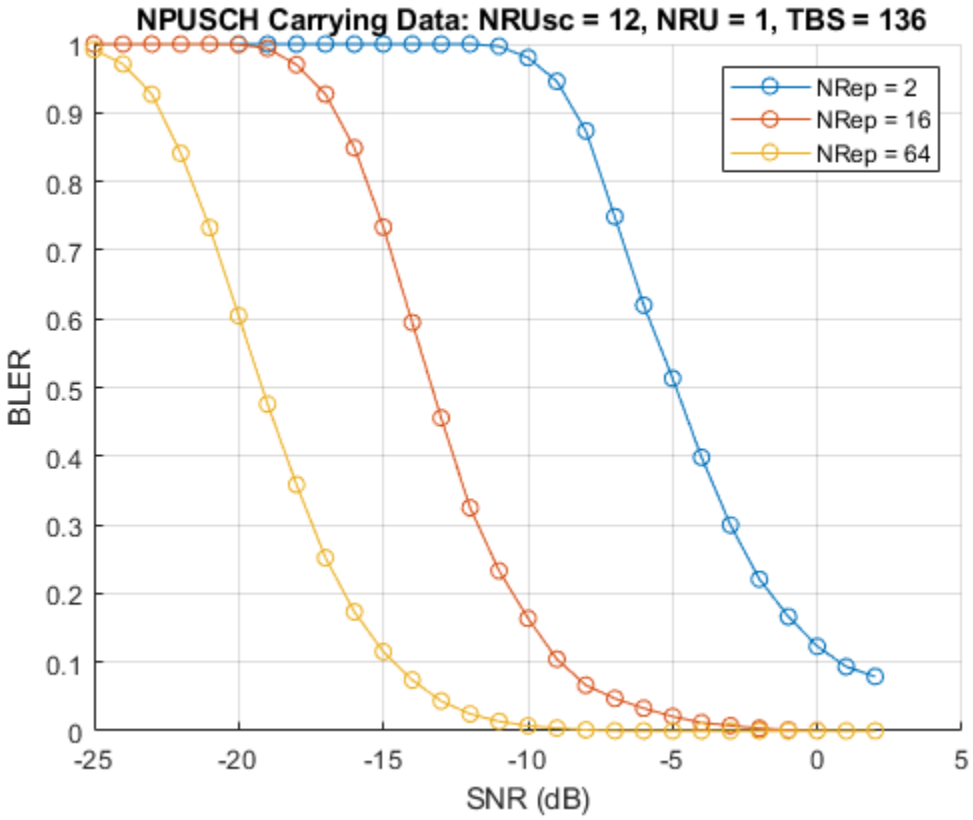
```
            end

            % Decode NPUSCH
            [rxcw,dstate,symbols] = lteNPUSCHDecode(...
                                ue, chs, rxNpuschSymbols, mean(npuschHest,3), mean(noiseEst),ds

            % Decode the transport block when all the slots in a block have
            % been received
            if dstate.EndOfBlk
                % Soft-combining at transport channel decoder
                [out, err, dstateULSCH] = lteNULSCHDecode(chs,infoLen,rxcw,dstateULSCH);
            end

            % If all the slots in the bundle have been received, count the
            % errors and reinitialize for the next bundle
            if dstate.EndOfTx
               if strcmpi(chs.NPUSCHFormat,'Control')
                   err = ~isequal(out,trblk);
               end
               numBlkErrors = numBlkErrors + err;
               % Re-initialize to enable the transmission of a new transport
               % block
               trblk = [];
            end
        end
        % Calculate the block error rate
        bler(snrIdx) = numBlkErrors/numTrBlks;
        fprintf('NPUSCH BLER = %.4f \n',bler(snrIdx));
        % Calculate the maximum and simulated throughput
        maxThroughput(snrIdx) = infoLen*numTrBlks; % Max possible throughput
        simThroughput(snrIdx) = infoLen*(numTrBlks-numBlkErrors);  % Simulated throughput
        fprintf('NPUSCH Throughput(%%) = %.4f %%\n',simThroughput(snrIdx)*100/maxThroughput(snrId

    end

    % Plot Block Error Rate vs SNR Results
    if repIdx == 1
        figure;
        grid on;
        hold on;
        xlabel('SNR (dB)');
        ylabel('BLER');
        legendstr = {['NRep = ' num2str(chsInit.NRep)]};
    else
        legendstr = [legendstr ['NRep = ' num2str(chsInit.NRep)]]; %#ok<AGROW>
    end
    plot(SNRdB, bler, '-o');

end
% Set figure title
title(sprintf(' NPUSCH Carrying %s: NRUsc = %d, NRU = %d, TBS = %d',...
    chsInit.NPUSCHFormat,chsInit.NRUsc,chsInit.NRU,infoLen));
legend(legendstr);


Generating 20 slots corresponding to 5 transport block(s) at -20dB SNR
NPUSCH BLER = 1.0000
NPUSCH Throughput(%) = 0.0000 %
```

```
Generating 20 slots corresponding to 5 transport block(s) at -18dB SNR
NPUSCH BLER = 1.0000
NPUSCH Throughput(%) = 0.0000 %

Generating 20 slots corresponding to 5 transport block(s) at -15dB SNR
NPUSCH BLER = 1.0000
NPUSCH Throughput(%) = 0.0000 %

Generating 20 slots corresponding to 5 transport block(s) at -12.5dB SNR
NPUSCH BLER = 1.0000
NPUSCH Throughput(%) = 0.0000 %

Generating 20 slots corresponding to 5 transport block(s) at -10dB SNR
NPUSCH BLER = 1.0000
NPUSCH Throughput(%) = 0.0000 %

Generating 20 slots corresponding to 5 transport block(s) at -6.4dB SNR
NPUSCH BLER = 0.4000
NPUSCH Throughput(%) = 60.0000 %

Generating 20 slots corresponding to 5 transport block(s) at -3.5dB SNR
NPUSCH BLER = 0.0000
NPUSCH Throughput(%) = 100.0000 %

Generating 20 slots corresponding to 5 transport block(s) at 0.7dB SNR
NPUSCH BLER = 0.0000
NPUSCH Throughput(%) = 100.0000 %

Generating 160 slots corresponding to 5 transport block(s) at -20dB SNR
NPUSCH BLER = 1.0000
NPUSCH Throughput(%) = 0.0000 %

Generating 160 slots corresponding to 5 transport block(s) at -18dB SNR
NPUSCH BLER = 1.0000
NPUSCH Throughput(%) = 0.0000 %

Generating 160 slots corresponding to 5 transport block(s) at -15dB SNR
NPUSCH BLER = 0.6000
NPUSCH Throughput(%) = 40.0000 %

Generating 160 slots corresponding to 5 transport block(s) at -12.5dB SNR
NPUSCH BLER = 0.2000
NPUSCH Throughput(%) = 80.0000 %

Generating 160 slots corresponding to 5 transport block(s) at -10dB SNR
NPUSCH BLER = 0.0000
NPUSCH Throughput(%) = 100.0000 %

Generating 160 slots corresponding to 5 transport block(s) at -6.4dB SNR
NPUSCH BLER = 0.0000
NPUSCH Throughput(%) = 100.0000 %

Generating 160 slots corresponding to 5 transport block(s) at -3.5dB SNR
NPUSCH BLER = 0.0000
NPUSCH Throughput(%) = 100.0000 %

Generating 160 slots corresponding to 5 transport block(s) at 0.7dB SNR
```

```
NPUSCH BLER = 0.0000
NPUSCH Throughput(%) = 100.0000 %

Generating 640 slots corresponding to 5 transport block(s) at -20dB SNR
NPUSCH BLER = 1.0000
NPUSCH Throughput(%) = 0.0000 %

Generating 640 slots corresponding to 5 transport block(s) at -18dB SNR
NPUSCH BLER = 0.6000
NPUSCH Throughput(%) = 40.0000 %

Generating 640 slots corresponding to 5 transport block(s) at -15dB SNR
NPUSCH BLER = 0.2000
NPUSCH Throughput(%) = 80.0000 %

Generating 640 slots corresponding to 5 transport block(s) at -12.5dB SNR
NPUSCH BLER = 0.0000
NPUSCH Throughput(%) = 100.0000 %

Generating 640 slots corresponding to 5 transport block(s) at -10dB SNR
NPUSCH BLER = 0.0000
NPUSCH Throughput(%) = 100.0000 %

Generating 640 slots corresponding to 5 transport block(s) at -6.4dB SNR
NPUSCH BLER = 0.0000
NPUSCH Throughput(%) = 100.0000 %

Generating 640 slots corresponding to 5 transport block(s) at -3.5dB SNR
NPUSCH BLER = 0.0000
NPUSCH Throughput(%) = 100.0000 %

Generating 640 slots corresponding to 5 transport block(s) at 0.7dB SNR
NPUSCH BLER = 0.0000
NPUSCH Throughput(%) = 100.0000 %
```



A larger number of transport blocks, `numTrBlks` should be used to produce meaningful throughput results. The following plot shows the simulation run with `numTrBlks` set to 5000 for different repetitions, with perfect channel estimator.



Selected Bibliography

1. 3GPP TS 36.211 "Physical channels and modulation"
2. 3GPP TS 36.212 "Multiplexing and channel coding"
3. 3GPP TS 36.213 "Physical layer procedures"
4. 3GPP TS 36.104 "Base Station (BS) radio transmission and reception"
5. 3GPP TS 36.321 "Medium Access Control (MAC); Protocol specification"
6. 3GPP TS 36.331 "Radio Resource Control (RRC); Protocol specification"
7. 3GPP TS 36.300 "Overall description; Stage 2"
8. O. Liberg, M. Sundberg, Y.-P. Wang, J. Bergman and J. Sachs, Cellular Internet of Things: Technologies, Standards and Performance, Elsevier, 2018.

Helper Functions

The following helper functions are used in this example:

- hPerfectTimingEstimate.m
- hSkipWeakTimingOffset.m

NB-IoT PRACH Waveform Generation

This example shows how to generate LTE-Advanced Pro Release 15 narrowband internet of things (NB-IoT) waveforms consisting of the narrowband physical random access channel (NPRACH) for frame structure type 1 by using LTE Toolbox™.

Introduction

3GPP introduced a new air interface, NB-IoT, optimized for low data rate machine type communications in LTE-Advanced Pro Release 13. NB-IoT provides cost and power efficiency improvements as it avoids the need for complex signaling overhead required for other LTE-based systems.

You can use LTE Toolbox to generate standard compliant NB-IoT uplink complex baseband waveforms, representing the 180 kHz narrowband carrier, suitable for test and measurement applications. The NB-IoT uplink physical layer channels and signals are:

- Narrowband demodulation reference signal (DM-RS)
- Narrowband physical uplink shared channel (NPUSCH)
- Narrowband physical random access channel (NPRACH)

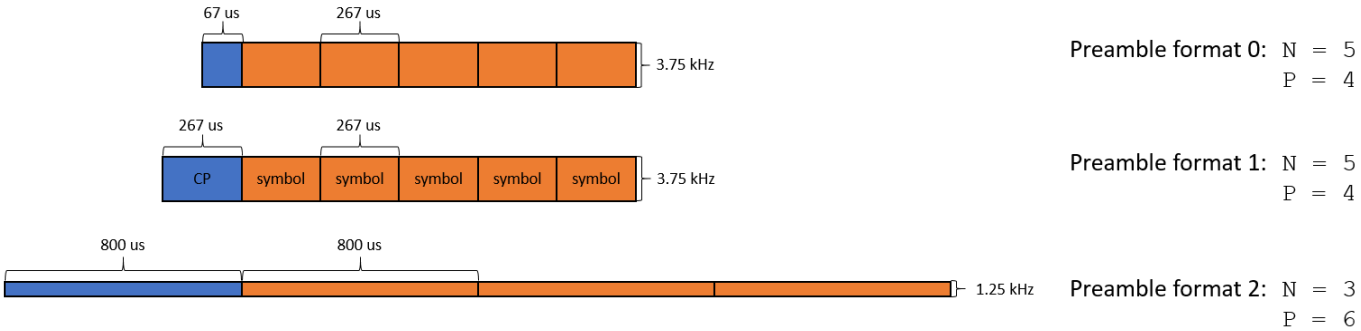
In this example, we demonstrate NPRACH allocation in the resource grid and the generation of NPRACH waveforms. Specifically, this example considers the frequency-division duplexing aspect of NB-IoT PRACH defined in LTE Advanced-Pro Release 15. This is referred to as frame structure type 1 in TS 36.211 Section 10.0.1 [1 on page 2-335].

The example generates the complex baseband waveform along with the populated grid containing the NPRACH signal.

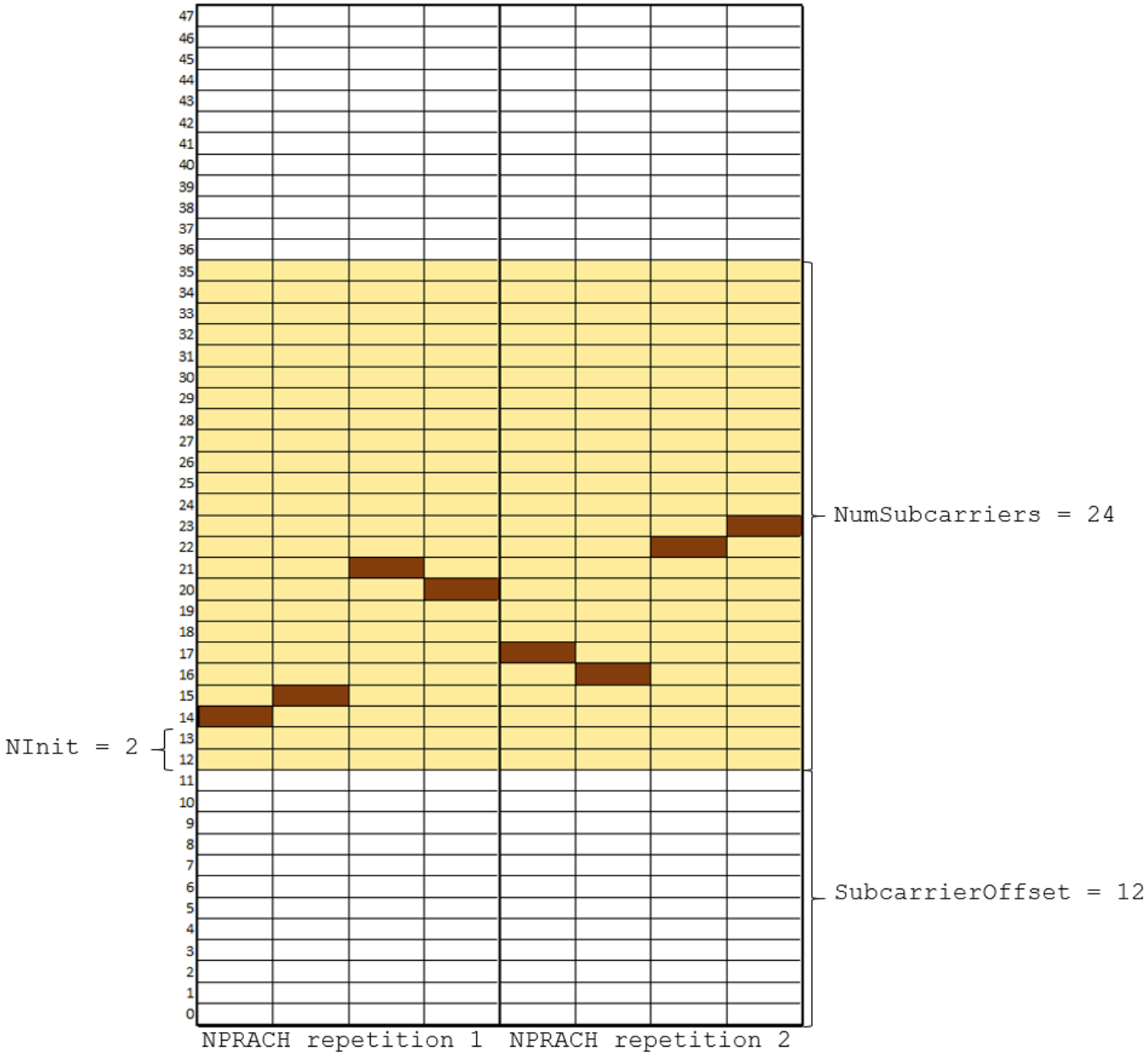
NPRACH Allocation

This section provides an overall description of the NPRACH allocation in time and frequency.

From a time point of view, an NPRACH transmission is defined as `NRep` repetitions of NPRACH preambles, in which an NPRACH preamble is defined as a set of `P` symbol groups. TS 36.211 Section 10.1.6.1 defines a symbol group as a sequence of `N` identical symbols preceded by a cyclic prefix. For this reason, the symbol group, rather than the OFDM symbol, can be considered as the atomic unit of NPRACH. The duration of an NPRACH symbol group is a function of the preamble format, as described in TS 36.211 Table 10.1.6.1-1. This figure shows a schematic of the symbol groups for the three NPRACH preamble formats.



From a frequency point of view, the NPRACH is characterized by single-tone transmission with frequency hopping and a subcarrier spacing defined in TS 36.211 Table 10.1.6.2-1. Each symbol group occupies a single subcarrier, following the frequency hopping pattern described in TS 36.211 Section 10.1.6.1. `SubcarrierOffset` defines the frequency location of the first subcarrier allocated to NPRACH. `NumSubcarriers` defines the total number of subcarriers allocated to NPRACH. Among those, the subcarrier location of the first symbol group of the first preamble is determined by `NInit`. For all subsequent preambles, the frequency of the first symbol group of each NPRACH preamble depends on the pseudorandom sequence defined in TS 36.211 Section 7.2 and initialized by `NNCellID`. This figure shows a schematic of the frequency hopping pattern for the first two repetitions of preamble format 0, given the NPRACH configuration used in this example. You can see that the first symbol groups of the two preambles occupy different subcarriers.



NPRACH Configuration

In this section, we configure the parameters required for NPRACH generation. For a list of all the allowed ranges for the parameters in the `ue` and `chs` structures, see `lteNPRACH`.

```
% Define the user equipment (UE) parameters
ue = struct();
ue.NBULSubcarrierSpacing = '15kHz'; % Uplink subcarrier spacing ('3.75kHz', '15kHz')
ue.NNCellID = 0;                    % Narrowband cell identity
ue.Windowing = 0;                   % Number of time-domain samples over which the function appli
                                    % Set Windowing to empty ([]) to use its default value.

% Define the channel parameters
chs = struct();
chs.NPRACHFormat = '0';    % Preamble format ('0','1','2')
chs.Periodicity = 80;      % NPRACH resource periodicity in ms
chs.SubcarrierOffset = 12; % Frequency location of the first subcarrier allocated to NPRACH
chs.NumSubcarriers = 24;   % Number of subcarriers allocated to NPRACH
chs.NRep = 8;              % Number of NPRACH repetitions
chs.StartTime = 8;         % NPRACH starting time in ms
chs.NInit = 2;             % Initial subcarrier for NPRACH
chs.NPRACHPower = 30;      % NPRACH power scaling in dB for plot visualization
```

NB-IoT PRACH Waveform Generation

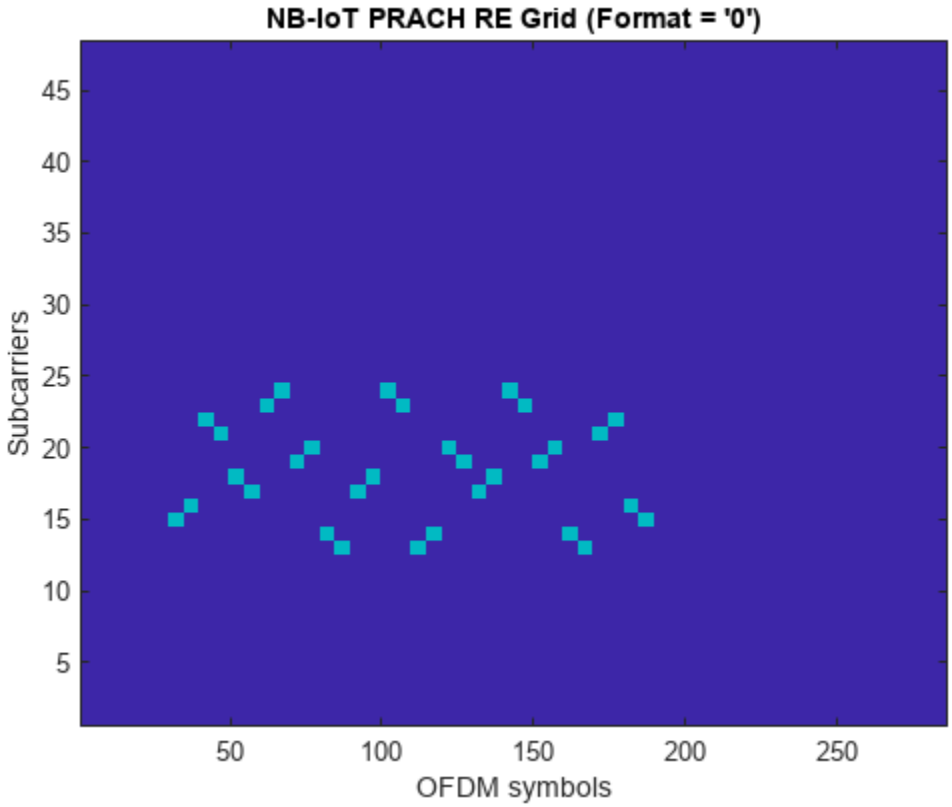
In this section, we create the NPRACH resource grid and the time-domain waveform. The example generates a waveform that spans `chs.Periodicity` milliseconds and is sampled with the same sample rate used for the uplink signals with subcarrier spacing given by `ue.NBULSubcarrierSpacing`.

```
[waveform,info,resourceGrid] = lteNPRACH(ue,chs);
```

Plot Transmitted Grid

Plot the NPRACH resource grid. You can observe the frequency hopping pattern of the NPRACH symbols. Note that the resource grid plot uses the power levels of the NPRACH to assign colors to the resource elements. To disable the plot of the resource grid, set the value of `displayGrid` to `'off'`.

```
displayGrid = 'on'; % Display the NPRACH resource grid ('on','off')
hNPRACHResourceGridPlot(chs,resourceGrid,displayGrid);
```



Summary and Further Exploration

This example shows how to generate a time-domain waveform and visualize the resource grid for a single NB-IoT PRACH configuration. The generated NPRACH waveform spans `chs.Periodicity` milliseconds and is sampled with the same sample rate used for the uplink signals with subcarrier spacing given by `ue.NBULSubcarrierSpacing`. This allows you to generate an NPRACH waveform and add it to an existing NB-IoT uplink waveform. For more information about the generation of an NB-IoT uplink waveform, see “NB-IoT Uplink Waveform Generation” on page 2-283.

Selected Bibliography

1 3GPP TS 36.211. "Physical channels and modulation", *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Local functions

This example uses this local function:

```
function hNPRACHResourceGridPlot(chs,resourceGrid,displayGrid)
%hNPRACHResourceGridPlot Plots the NPRACH resource grid
%   hNPRACHResourceGridPlot(CHS,RESOURCEGRID,DISPLAYGRID) plots the NPRACH
%   resource grid. Set DISPLAYGRID to 'off' to disable the plot of the
%   resource grid.

    if strcmpi(displayGrid,'on')
```

```
            figure

            % Create an image of the NPRACH resource grid
            im = image(resourceGrid);

            % Adjust plot colors, axes, and add title and labels
            cmap = parula(64);
            colormap(im.Parent,cmap);
            axis xy;
            xlabel('OFDM symbols')
            ylabel('Subcarriers')
            title(sprintf('NB-IoT PRACH RE Grid (Format = ''%s'')',chs.NPRACHFormat))

        end
    end
```

NB-IoT PRACH Detection and False Alarm Conformance Test

This example implements the narrowband physical random access channel (NPRACH) missed detection and false alarm conformance tests for frame structure type 1, as defined in TS 36.141. You can measure the probability of correct detection of the NPRACH preamble in the presence of a preamble signal or switch the NPRACH transmission off to measure the false alarm probability.

Introduction

The NPRACH is a narrowband internet of things (NB-IoT) uplink transmission used by the user equipment (UE) to initiate synchronization with the eNodeB. Section 8.5.3 of TS 36.141 defines that the probability of NPRACH detection must be greater than or equal to 99% at the SNR levels listed in Table 8.5.3.5-1 for several combinations of preamble format, number of repetitions, and channel propagation conditions. There are several detection error cases:

- Detecting an incorrect preamble
- Not detecting a preamble
- Detecting the correct preamble but with the wrong timing estimation

TS 36.141 states that a correct detection is achieved when the estimation error of the timing offset of the strongest path is less than 3.646 microseconds.

In this example, an NPRACH waveform is configured and passed through an appropriate channel. At the receiver side, the example performs NPRACH detection and calculates the NPRACH detection probability. The example considers the parameters defined in Table 8.5.3.1-1 and Table 8.5.3.5-1 of TS 36.141. These are: frequency-division duplexing (FDD), 2 receive antennas, 8 repetitions, EPA1 channel, preamble format 0, SNR 6.7 dB. If you change the simulation scenario to use one of the other NPRACH preamble formats, propagation channel, or number of repetitions listed in Table 8.5.3.5-1, you need to update the NPRACH configuration according to Table 8.5.3.1-1 and you need to update the values of the frequency offset and the SNR according to Table 8.5.3.5-1.

Simulation Configuration

The example considers 10 NPRACH transmissions at a number of SNRs. You should use a large number of `numNPRACHTransmissions` to produce meaningful results. You can set `SNRdB` as an array of values or a scalar. Table 8.5.3.5-1 of TS 36.141 specifies the frequency offset `foffset` that is modeled between the transmitter and receiver. The `timeErrorTolerance` variable specifies the time error tolerance, as defined in Section 8.5.3.1 of TS 36.141. You can set the detection threshold to a value in the range [0,1] or to empty to use the default value in the `lteNPRACHDetect` function. To simulate a false alarm test, disable the NPRACH transmission by setting `nprachEnabled` to `false` instead.

```
numNPRACHTransmissions = 10;         % Number of NPRACH transmissions to simulate at each SNR
SNRdB = [-5.3 -2.3 0.7 3.7 6.7 9.7]; % SNR range in dB
foffset = 200.0;                     % Frequency offset in Hz
timeErrorTolerance = 3.646;          % Time error tolerance in microseconds
threshold = [];                      % Detection threshold
nprachEnabled = true;                % Enable NPRACH transmission. To simulate false alarm test,
```

UE Configuration

UE settings are specified in the structure `ue`.

```
ue.NBULSubcarrierSpacing = '15kHz'; % Uplink subcarrier spacing ('3.75kHz', '15kHz')
ue.NNCellID = 0;                    % Narrowband cell identity
ue.Windowing = 0;                   % Windowing samples
```

NPRACH Configuration

Table 8.5.3.1-1 of TS 36.141 specifies the NPRACH configurations to use for the NPRACH detection conformance test.

Set the NPRACH configuration structure `nprach` and compute the NPRACH resource information `nprachInfo` by calling `lteNPRACHInfo`.

```
nprach = struct();
nprach.NPRACHFormat = '0';    % Preamble format ('0','1','2')
nprach.Periodicity = 80;      % Resource periodicity in ms
nprach.SubcarrierOffset = 0;  % Frequency location of the first allocated subcarrier
nprach.NumSubcarriers = 12;   % Number of allocated subcarriers
nprach.NRep = 8;              % Number of repetitions
nprach.StartTime = 8;         % Starting time in ms
nprach.NInit = 0;             % Initial subcarrier
nprach.NPRACHPower = 0;       % Power scaling in dB

nprachInfo = lteNPRACHInfo(ue,nprach); % NPRACH resource information
```

Propagation Channel Configuration

Configure the propagation channel model using the `chcfg` structure as described in Table 8.5.3.5-1 of TS 36.141.

```
chcfg.NRxAnts = 2;                          % Number of receive antennas
chcfg.DelayProfile = 'EPA';                 % Delay profile
chcfg.DopplerFreq = 1;                      % Doppler frequency
chcfg.MIMOCorrelation = 'Low';              % MIMO correlation
chcfg.Seed = 42;                            % Channel seed. Change this for different channel r
chcfg.NTerms = 16;                          % Oscillators used in fading model
chcfg.ModelType = 'GMEDS';                  % Rayleigh fading model type
chcfg.InitPhase = 'Random';                 % Random initial phases
chcfg.NormalizePathGains = 'On';            % Normalize delay profile power
chcfg.NormalizeTxAnts = 'On';               % Normalize for transmit antennas
chcfg.SamplingRate = nprachInfo.SamplingRate; % Sampling rate

% Compute the maximum channel delay
chcfg.InitTime = 0;
[~,chInfo] = lteFadingChannel(chcfg,0); % Get channel info
maxChDelay = ceil(max(chInfo.PathSampleDelays)) + chInfo.ChannelFilterDelay;
```

Loop for SNR Values

Use a loop to run the simulation for the set of SNR points given by the vector `SNRdB`. The SNR vector configured here is a range of SNR points including a point at 6.7 dB, the SNR at which the test requirement for NPRACH detection rate (99%) is to be achieved for preamble format 0, as discussed in Table 8.5.3.5-1 of TS 36.141.

`lteNPRACH` generates an output signal normalized to the same transmit power as for an uplink data transmission within the LTE Toolbox™. Therefore, the same normalization must take place on the noise added to the NPRACH. The noise added before OFDM demodulation will be amplified by the IFFT by a factor equal to the square root of the size of the IFFT (N_{FFT}). To ensure that the power of

the noise added is normalized after demodulation, and thus to achieve the desired SNR, the desired noise power is divided by N_{FFT}. In addition, as real and imaginary parts of the noise are created separately before being combined into complex additive white Gaussian noise, the noise amplitude is scaled by $1/\sqrt{2}$ so the generated noise power is 1.

At each SNR test point, calculate the probability of detection for each NPRACH transmission using these steps:

- *NPRACH Transmission:* Use `lteNPRACH` to generate an NPRACH waveform. Send the NPRACH preambles with a fixed timing offset of 50% of the number of cyclic shifts, as defined in Section 8.5.3.4.2 of TS 36.141.
- *Noisy Channel Modeling:* Pass the waveform through a fading channel and add additive white Gaussian noise. Add additional samples to the end of the waveform to cover the range of delays expected from the channel modeling (a combination of implementation delay and channel delay spread). This implementation delay is then removed to ensure the implementation delay is not interpreted as an actual timing offset in the preamble detector.
- *Application of Frequency Offset:* Apply the frequency offset to the received waveform as defined in Table 8.5.3.5-1 of TS 36.141.
- *NPRACH Detection:* Perform NPRACH detection using the `lteNPRACHDetect` function for all initial subcarrier indices `NInit`, each one of which uniquely defines an NPRACH preamble. Use the detected NPRACH index and offset returned by `lteNPRACHDetect` to determine whether a detection was successful according to the constraints discussed in the Introduction section.

```
% Initialize variables storing probability of detection at each SNR
pDetection = zeros(size(SNRdB));

% The temporary variables 'chcfg_init' and 'chInfo_init' are used to create
% the temporary variables 'chcfg' and 'chInfo' within the SNR loop to
% create independent instances in case of parallel simulation
chcfg_init = chcfg;
chInfo_init = chInfo;

for snrIdx = 1:length(SNRdB) % comment out for parallel computing
% parfor snrIdx = 1:numel(SNRdB) % uncomment for parallel computing
% To reduce the total simulation time, you can execute this loop in
% parallel by using the Parallel Computing Toolbox(TM). Comment out the
% 'for' statement and uncomment the 'parfor' statement. If the Parallel
% Computing Toolbox is not installed, 'parfor' defaults to normal 'for'
% statement

    % Display progress in the command window
    timeNow = char(datetime('now','Format','HH:mm:ss'));
    fprintf([timeNow,': Simulating SNR = %+5.1f dB...'],SNRdB(snrIdx));

    % Set the random number generator seed depending on the loop variable
    % to ensure independent random streams
    rng(snrIdx,'combRecursive');

    % Initialize variables for this SNR point, required for initialization
    % of variables when using the Parallel Computing Toolbox
    chcfg = chcfg_init;
    chInfo = chInfo_init;
```

```
% Normalize noise power to account for the sampling rate, which is a
% function of the IFFT size used in OFDM modulation. The SNR is defined
% per carrier resource element for each receive antenna.
ueInfo = lteSCFDMAInfo(ue);
SNR = 10^(SNRdB(snrIdx)/10);
N0 = 1/sqrt(2.0*double(ueInfo.Nfft)*SNR);

% Detected preamble count
detectedCount = 0;

% Loop for each NPRACH transmission
for nprachIdx = 1:numNPRACHTransmissions

    % Generate NPRACH waveform
    waveform = generateWaveform(ue,nprach,nprachInfo,nprachEnabled);

    % Set NPRACH timing offset in microseconds as per Section 8.5.3.4.2 of TS 36.141
    timingOffset = 0.5*nprachInfo.PreambleParameters.T_CP/30720*1e3; % (microseconds)
    sampleDelay = fix(timingOffset/1e6*nprachInfo.SamplingRate);

    % Generate transmit waveform
    txwave = [zeros(sampleDelay,1); waveform];

    % Pass data through channel model
    rxwave = applyChannel(txwave,chcfg,nprachInfo.Nfft,nprach.Periodicity,nprachIdx,maxChDela

    % Add noise
    noise = N0*complex(randn(size(rxwave)),randn(size(rxwave)));
    rxwave = rxwave + noise;

    % Apply frequency offset
    t = ((0:size(rxwave,1) - 1)/chcfg.SamplingRate).';
    rxwave = rxwave.*repmat(exp(1i*2*pi*foffset*t),1,size(rxwave,2));

    % NPRACH detection for all frequency hopping patterns
    [detected,offset,detinfo] = lteNPRACHDetect(ue,nprach,rxwave,threshold);

    % Test for frequency hopping pattern detection
    if (length(detected)==1)
        if ~nprachEnabled
            % For the false alarm test, any preamble detected is wrong
            detectedCount = detectedCount + 1;
        else
            % Test for correct frequency hopping pattern detection
            if (detected==nprach.NInit)

                % Calculate timing estimation error
                trueOffset = timingOffset/1e6; % (s)
                measuredOffset = offset/nprachInfo.SamplingRate;
                timingerror = abs(measuredOffset - trueOffset);

                % Test for acceptable timing error
                if (timingerror<=(timeErrorTolerance/1e6))
                    detectedCount = detectedCount + 1; % Detected preamble
                end
            end
        end
    end
```

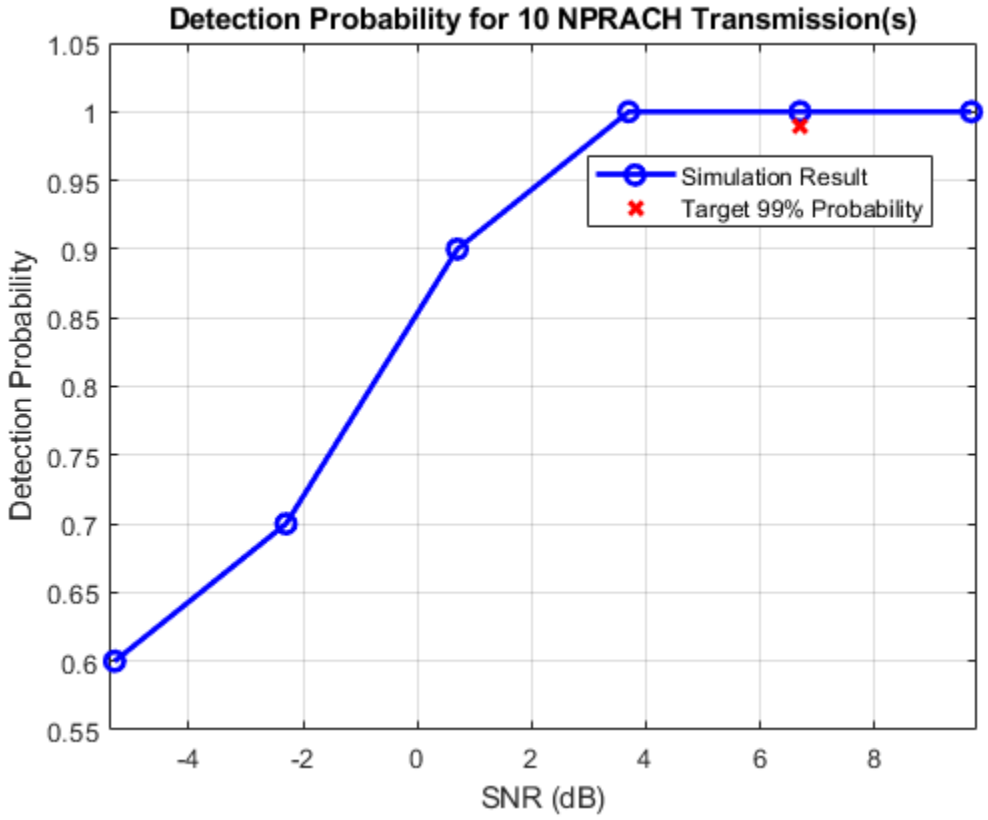
```
    end % of NPRACH transmission loop

    % Compute final detection probability for this SNR
    pDetection(snrIdx) = detectedCount/numNPRACHTransmissions;

    % Display the detection probability for this SNR
    fprintf('Detection probability: %4.2f%%\n',pDetection(snrIdx)*100);

end % of SNR loop
```

```
23:27:49: Simulating SNR =  -5.3 dB...
```

Analysis

At the end of the SNR loop, the example plots the calculated detection probabilities for each SNR value against the target probability.

```
targetSNR = 6.7;
NPRACHDetectionResults(pDetection,SNRdB,targetSNR,numNPRACHTransmissions,nprachEnabled);
```



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)*.

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).*

Local Functions

```
function waveform = generateWaveform(ue,nprach,nprachInfo,nprachEnabled)
    % Generate the waveform. If the NPRACH transmission is disabled, the
    % function generates a waveform of zeros.
    if nprachEnabled
        waveform = lteNPRACH(ue,nprach);
    else
        waveform = complex(zeros(nprach.Periodicity*nprachInfo.SamplingRate/1e3,1));
    end
end

function rxwave = applyChannel(txwave,chcfg,nFFT,periodicity,nprachIdx,maxChDelay)
    % Pass data through channel model

    % Resample the transmit waveform to speed up the channel
    nFFT_mod = nFFT/8;
    txwave = resample(txwave,nFFT_mod,nFFT);
    chcfg.SamplingRate = chcfg.SamplingRate*nFFT_mod/nFFT;

    % Pass data through channel model. Append zeros at the end of the
    % transmitted waveform to flush channel content. These zeros take into
    % account any delay introduced in the channel. This is a mix of
    % multipath delay and implementation delay. This value may change
    % depending on the sampling rate and delay profile.
    chcfg.InitTime = (nprachIdx - 1)*periodicity/1000;
    [rxwave,chInfo] = lteFadingChannel(chcfg,[txwave; zeros(ceil(maxChDelay*nFFT_mod/nFFT),1)]);

    % Remove the implementation delay of the channel filter
    rxwave = rxwave((chInfo.ChannelFilterDelay + 1):end,:);

    % Resample the receive waveform back to the original sampling rate
    rxwave = resample(rxwave,nFFT,nFFT_mod);
end

function NPRACHDetectionResults(pDetection,SNRdB,targetSNR,numNPRACHTransmissions,nprachEnabled)
    % Plot detection probability
    figure('NumberTitle','off','Name','NPRACH Detection Probability');
    plot(SNRdB,pDetection,'b-o','LineWidth',2,'MarkerSize',7);
    title(['Detection Probability for ',num2str(numNPRACHTransmissions),' NPRACH Transmission(s)
    xlabel('SNR (dB)'); ylabel('Detection Probability');
    grid on; hold on;

    % Plot target probability
    if nprachEnabled
        % For a missed detection test, detection probability should be >= 99%
        pTarget = 99;
    else
        % For a false alarm test, detection probability should be < 0.1%
        pTarget = 0.1;
    end
    plot(targetSNR,pTarget/100,'rx','LineWidth',2,'MarkerSize',7);
    legend('Simulation Result', ['Target ',num2str(pTarget),'% Probability'],'Location','best');
    minP = 0;
```

```
    if(~isnan(min(pDetection)))
        minP = min([pDetection(:); pTarget]);
    end
    axis([SNRdB(1)-0.1, SNRdB(end)+0.1, minP-0.05, 1.05]);
end
```

```
Detection probability: 60.00%
23:27:52: Simulating SNR =  -2.3 dB...Detection probability: 70.00%
23:27:56: Simulating SNR =  +0.7 dB...Detection probability: 90.00%
23:27:59: Simulating SNR =  +3.7 dB...Detection probability: 100.00%
23:28:02: Simulating SNR =  +6.7 dB...Detection probability: 100.00%
23:28:06: Simulating SNR =  +9.7 dB...Detection probability: 100.00%
```

See Also

Functions

`lteNPRACH` | `lteNPRACHInfo`

Related Examples

- “NB-IoT PRACH Waveform Generation” on page 2-332
- “PRACH False Alarm Probability Conformance Test” on page 2-559
- “PRACH Detection Conformance Test” on page 2-554

NB-IoT Cell Search and MIB Recovery

This example shows how the LTE Toolbox™ can be used to fully synchronize, demodulate and decode a live narrowband internet of things (NB-IoT) downlink signal. Before the user equipment (UE) can communicate with the network, it must perform cell search and selection procedures and obtain initial system information. This involves acquiring slot and frame synchronization, finding out the cell identity and decoding the MIB. This example demonstrates this process and decodes the MIB. To decode MIB a comprehensive receiver is required, capable of demodulating and decoding the downlink channels and signals.

Introduction

As NB-IoT is an extension of LTE infrastructure, the NB-IoT waveform has some similarities to that of LTE. Similarities between these two include:

- Sub-carrier spacing (15kHz)
- Frame structure (10 subframes with 2 slots each)
- Frame duration (10 ms)
- Physical channels

However, the number of subcarriers in NB-IoT is fixed to 12 (one resource block), unlike in LTE, where number of subcarriers vary with the bandwidth. Also, the physical channel mapping on the resource grid of NB-IoT is different from that of LTE. For more information see “NB-IoT Downlink Waveform Generation” on page 2-104.

In order to communicate with the network the UE must obtain some basic system information. This is carried by the MIB and SIBs. The MIB carries the most essential system information:

- System frame number (SFN)
- Scheduling information of SIB1-NB
- Operation mode information for LTE-NB
- Access barring

The MIB is carried on the broadcast channel (BCH) mapped into the narrowband physical broadcast channel (NPBCH) [1]. This is transmitted with a fixed coding and modulation scheme and can be decoded after the initial cell search procedure. The MIB corresponds to one BCH transport block. The BCH transmission time interval (TTI), or the time needed to transmit a single transport block, is 640 msec or 64 frames [3]. The BCH is transmitted in 64 parts, each part mapped to the first subframe (subframe 0) of a frame and it is possible that each transmission is independently decodable, depending on signal conditions. To ensure that subframe 0 is received, the capture should be at least 11 subframes long, to account for the possibility that the capture is started during subframe 0. For poor signal conditions, all 64 parts of the TTI may be required, in which case the capture should be at least 641 subframes long. For more details on NPBCH symbols generation, see “NB-IoT Downlink Waveform Generation” on page 2-104.

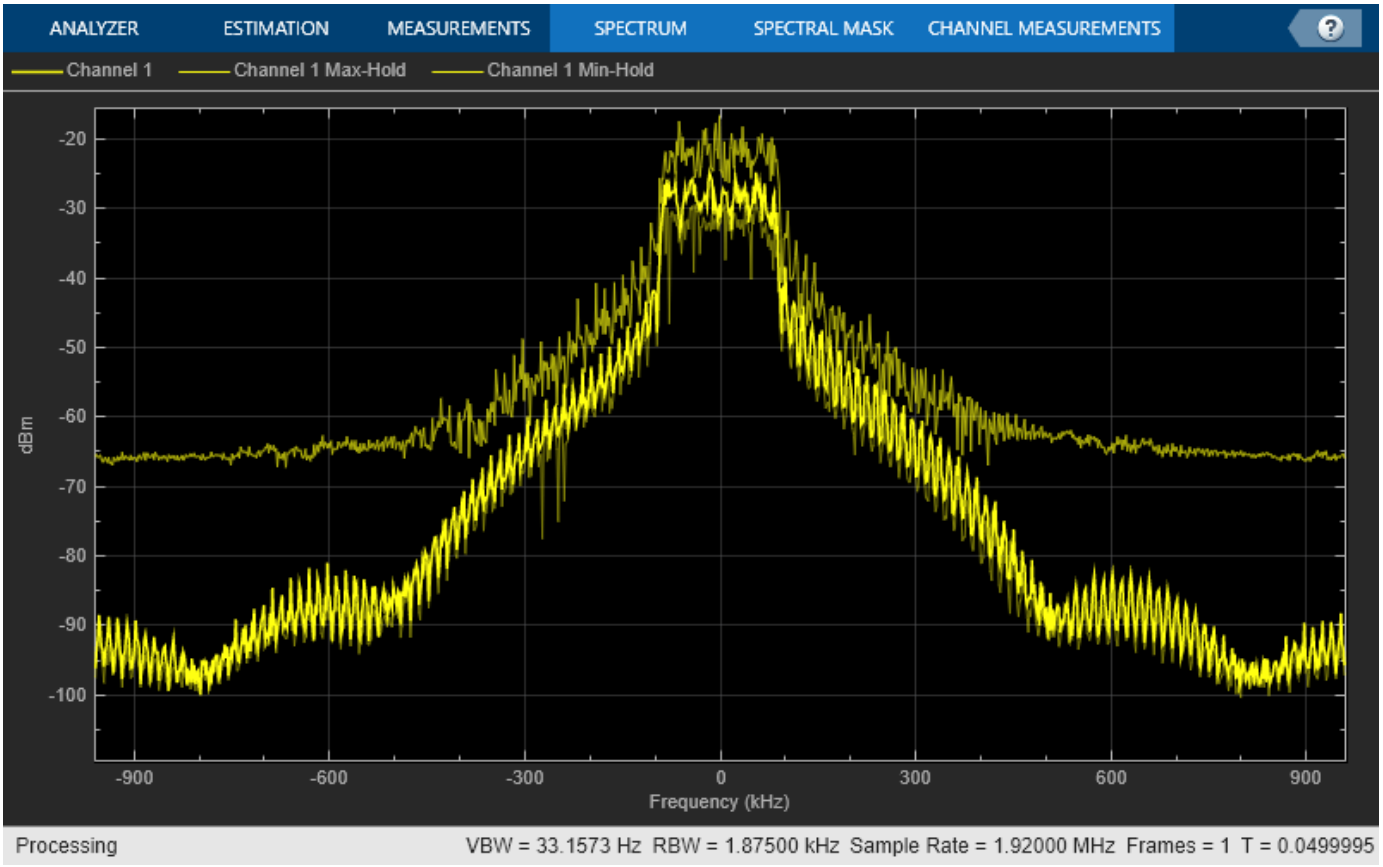
NB-IoT Downlink Waveform Generation and Initializing Parameters

Generate the time-domain waveform of a reference measurement channel (RMC) for narrowband physical downlink shared channel (NPDSCH) performance requirements, in accordance to TS 36.101 Annex A.3.12 [2]. The waveform is generated using the generateWaveform method in the class

NBIoTDownlinkWaveformGenerator. To call the method, an object `ngen` is created and the method is called by using `generateWaveform(ngen)`.

```
% Initialize NBIoTDownlinkWaveformGenerator object
%
% Use the 'rc' values of 'R.NB.5' and 'R.NB.6' for MIB decoding. Note that
% RMCs 'R.NB.5-1', 'R.NB.6-1' and 'R.NB.7' correspond to Non-Anchor type
% carriers which do not contain the required NPSS, NSSS and NPBCH.
rc = 'R.NB.5';
ngen = NBIoTDownlinkWaveformGenerator(rc);
ngen.Config.NNCellID = 120;
ngen.Config.NBRefP = 1;
ngen.Config.NFrame = 640;
ngen.Config.TotSubframes = 50;
ngen.Config.NPBCH.DataSource = 'MIB';

% Generate time domain NB-IoT waveform
[eNodeBOutput,grid,ofdmInfo] = generateWaveform(ngen);
sr = ofdmInfo.SamplingRate;

% Introduce a delay to simulate the unknown timing alignment associated
% with an arbitrary signal capture. This is compensated at the receiver
% while synchronizing the received signal.
delay = 50;
waveform = circshift(eNodeBOutput, delay);

% Initialize plots
if (~exist('channelFigure','var') || ~isvalid(channelFigure))
    channelFigure = figure('Visible','off');
end
[spectrumScope,synchCorrPlot,pdcchConstDiagram] = ...
    hSIB1RecoveryExamplePlots(channelFigure,sr);

% Display received signal spectrum
fprintf('\nPlotting received signal spectrum...\n');
spectrumScope(awgn(waveform, 100.0));
```

```
Plotting received signal spectrum...
```



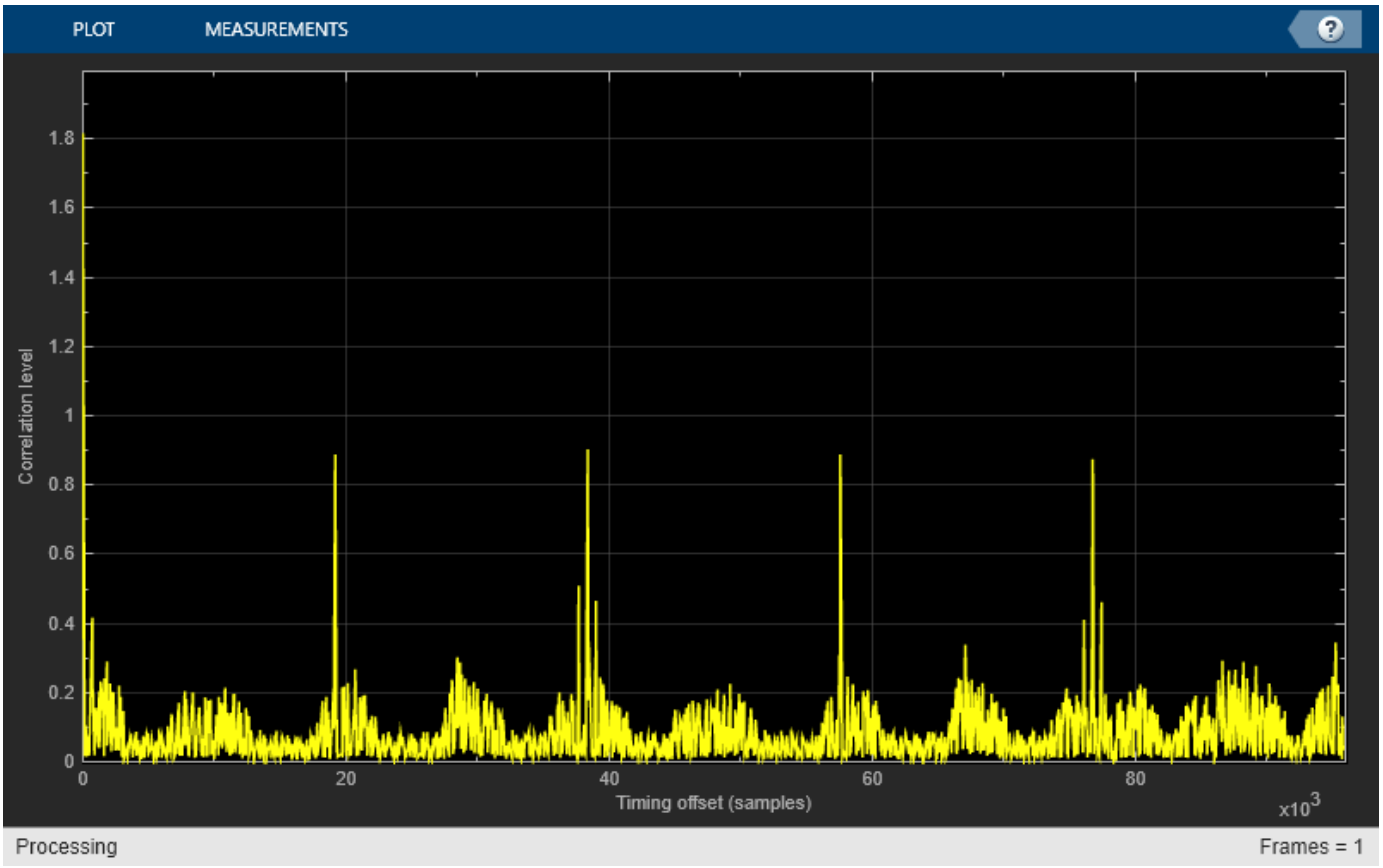
Cell Search and Timing Synchronization

Call the function hNBCellSearch to obtain the cell identity and timing offset `offset` to the first frame head. A plot of the correlation between the received signal and the narrowband primary synchronization signal (NPSS)/narrowband secondary synchronization signal (NSSS) for the detected cell identity is produced. The NPSS is detected using time-domain correlation and the NSSS is detected using frequency-domain correlation. Prior to NSSS detection, frequency offset estimation/correction using cyclic prefix correlation is performed. The time-domain NPSS detection is robust to small frequency offsets but larger offsets may degrade the NPSS correlation.

```
% Cell search
fprintf('\nPerforming cell search...\n');
alg.SSSDetection = 'PostFFT';
alg.MaxCellCount = 1;
[NNCellID, offset, peak] = hNBCellSearch(waveform, alg);
nbenb = struct;
nbenb.NNCellID = NNCellID;

[~,corr] = lteNBDLFrameOffset(setfield(nbenb,'OperationMode','Standalone'),waveform); %#ok<SFLD>

% As NB-IoT has similar waveform structure as LTE waveform, use functions
% pertaining to LTE to work with NB-IoT. For this purpose, create a
% cell-wide settings structure enb.
enb = nbenb;

% Plot NPSS/NSSS correlation and threshold
```

```
synchCorrPlot.YLimits = [0 max(corr(:))*1.1];
synchCorrPlot(corr);

% Perform timing synchronization
fprintf('Timing offset to frame start: %d samples\n',offset);
timesynced = waveform(1+offset:end,:);

% Show cell-wide settings
fprintf('Cell-wide settings after cell search:\n');
disp(nbenb);
```

```
Performing cell search...
Timing offset to frame start: 50 samples
Cell-wide settings after cell search:
    NNCellID: 120
```



Frequency Offset Estimation and Correction

Prior to OFDM demodulation, any significant frequency offset must be removed. The frequency offset in the I/Q waveform is estimated and corrected using `lteFrequencyOffset` and `lteFrequencyCorrect`. The frequency offset is estimated by means of correlation of the cyclic prefix and therefore can estimate offsets up to +/- half the subcarrier spacing i.e. +/- 7.5kHz. NB-IoT downlink waveform generation is similar to LTE uplink waveform generation as both the waveforms

have half sub-carrier shift. Hence, use LTE uplink functions to estimate the frequency offset and to demodulate the NB-IoT OFDM downlink received waveforms.

```
% Frequency offset estimation
fprintf('\nPerforming frequency offset estimation...\n');
enb.NULRB = 6; % For frequency offset computation
enb.DuplexMode = 'FDD';
foffset = lteFrequencyOffset(enb,timesynced,0);
fprintf('Frequency offset: %0.3fHz\n',foffset);

% Compensating for frequency offset
freqsynced = lteFrequencyCorrect(enb,timesynced,foffset);
```

```
Performing frequency offset estimation...
Frequency offset: -0.031Hz
```

OFDM Demodulation and Channel Estimation

The OFDM downsampled I/Q waveform is demodulated to produce a resource grid `rxgrid`. This is used to perform channel estimation. `hest` is the channel estimate, `nest` is an estimate of the noise (for MMSE equalization), and `cec` is the channel estimator configuration.

For channel estimation the example assumes four cell specific reference signals, and two narrowband reference signals. This means that channel estimates to each receiver antenna from all possible cell-specific reference signal ports are available. The true number of cell-specific reference signal ports is not yet known. The channel estimation is only performed on the first subframe, i.e. using the first `L` OFDM symbols in `rxgrid`.

A conservative 13-by-9 pilot averaging window is used, both in frequency and time, to reduce the impact of noise on pilot estimates during channel estimation.

```
% Channel estimator configuration
cec.PilotAverage = 'UserDefined';     % Type of pilot averaging
cec.FreqWindow = 13;                  % Frequency window size
cec.TimeWindow = 9;                   % Time window size
cec.InterpType = 'Cubic';             % 2D interpolation type
cec.InterpWindow = 'Centered';        % Interpolation window type
cec.InterpWinSize = 1;                % Interpolation window size
cec.Reference = 'NRS';                % NB-IoT downlink channel estimation

% Assume 4 cell specific and 2 narrowband reference signals for initial
% decoding attempt. This ensures channel estimates are available for all
% reference signals
enb.CellRefP = 4;
enb.NBRefP = 2;
enb.NTxAnts = 2;
nbenb.NBRefP = 2;

fprintf('Performing OFDM demodulation...\n\n');
griddims = lteResourceGridSize(enb); % Resource grid dimensions
L = griddims(2);                     % Number of OFDM symbols in a subframe

enb.NBULSubcarrierSpacing = '15kHz'; % Required for NB-IoT downlink OFDM demodulation
rxgrid = lteSCFDMADemodulate(enb,freqsynced); % For OFDM demodulation in NB-IoT
if (isempty(rxgrid))
    fprintf('After timing synchronization, signal is shorter than one subframe so no further demo
```

```
    return;
end

% Perform channel estimation
enb.NSubframe = 0; % Initialize NSubframe for channel estimation
[hest, nest] = lteDLChannelEstimate(enb, cec, rxgrid(:,1:L,:));

Performing OFDM demodulation...
```

NPBCH Demodulation, BCH Decoding, MIB Parsing

The MIB is now decoded along with the number of narrowband reference signal ports transmitted as a mask on the BCH CRC. The function `lteNPBCHDecode` establishes frame timing modulo 64 and returns this in the `nfmod64` parameter. It also returns the MIB bits in vector `mib` and the true number of narrowband reference signal ports which is assigned into `nbenb.NBRefP` at the output of this function call. If the number of narrowband reference signal ports is decoded as `nbenb.NBRefP=0`, this indicates a failure to decode the BCH.

```
% Decode the MIB
% Extract resource elements (REs) corresponding to the NPBCH from the first
% subframe across all receive antennas and channel estimates
separator = repmat('-',1,50);
fprintf('%s\n',separator);
fprintf('Performing MIB decoding...\n');
fprintf('%s\n\n',separator);
npbchIndices = lteNPBCHIndices(nbenb);
[npbchRx, npbchHest] = lteExtractResources(npbchIndices, ...
                               rxgrid(1:12,1:L,:), hest(:,1:L,:,:));

% Decode NPBCH
dstate = [];
[bchBits, dstate,npbchSymbols,nfmod64,mib,nbenb.NBRefP] = lteNPBCHDecode( ...
    nbenb, npbchRx, npbchHest, nest, dstate);

% Parse MIB bits
nbenb = hNBMIB(mib,nbenb);

if (nbenb.NBRefP == 0)
    fprintf('MIB decoding failed (nbenb.NBRefP=0).\n\n');
    return;
end

% Incorporate the nfmod64 value from the function lteNPBCHDecode, as
% the NFrame value established from the MIB is the SFN modulo 64 (it is
% stored in the MIB as floor(SFN/64)).
if nbenb.NBRefP % If NPBCH decoding is passed then set the NFrame value
    nbenb.NFrame = nbenb.NFrame+nfmod64;
end

if strcmpi(nbenb.OperationMode, 'Inband-SamePCI')
    nbenb.NCellID = NNCellID;
end

% Display cell-wide settings after MIB decoding
fprintf('Cell-wide settings after MIB decoding:\n');
disp(nbenb);
```

```
------------------------------------------------
Performing MIB decoding...
------------------------------------------------

Cell-wide settings after MIB decoding:
                      NNCellID: 120
                        NBRefP: 1
                        NFrame: 640
                      HyperSFN: 0
                     ABEnabled: 0
                 OperationMode: 'Inband-DifferentPCI'
    AdditionalTransmissionSIB1: 0
```

Appendix

This example uses the following helper files:

- NBIoTDownlinkWaveformGenerator
- hNBCellSearch.m
- hNBMIB.m
- hSIB1RecoveryExamplePlots.m

Selected Bibliography

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 36.101. "User Equipment (UE) radio transmission and reception". 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA). URL: https://www.3gpp.org.
3. O. Liberg, M. Sundberg, Y.-P. Wang, J. Bergman, and J. Sachs. Cellular Internet of Things: Technologies, Standards and Performance. Elsevier, 2018.

Cell Search, MIB and SIB1 Recovery

This example shows how to fully synchronize, demodulate and decode a live eNodeB signal by using LTE Toolbox™ software. Before the user equipment (UE) can communicate with the network, it must perform cell search and selection procedures, and then obtain initial system information. This process involves acquiring slot and frame synchronization, determining the cell identity and decoding the MIB and system information blocks (SIBs). This example demonstrates this process and decodes the MIB and SIB1, the first of the system information blocks. Decoding the MIB and SIB1 requires a comprehensive that is capable of demodulating and decoding the majority of the downlink channels and signals.

Introduction

In order to communicate with the network the UE must obtain some basic system information. This is carried by the MIB and SIBs. The MIB carries the most essential system information:

- System bandwidth
- System frame number (SFN)
- Physical hybrid automatic repeat request (HARQ) indicator channel (PHICH) configuration

The MIB is carried on the broadcast channel (BCH) mapped into the physical broadcast channel (PBCH). This is transmitted with a fixed coding and modulation scheme and can be decoded after the initial cell search procedure. With the information obtained from the MIB the UE can now decode the control format indicator (CFI), which indicates the physical downlink control channel (PDCCH) length. This allows the PDCCH to be decoded, and searched for downlink control information (DCI) messages. A DCI message CRC masked with system information radio network temporary identifier (SI-RNTI) indicates that a SIB is carried in the same subframe. The SIBs are transmitted in the broadcast control channel (BCCH) logical channel. Generally, BCCH messages are carried on the downlink shared channel (DL-SCH) and transmitted on the physical downlink shared channel (PDSCH). The format and resource allocation of the PDSCH transmission is indicated by a DCI message on the PDCCH.

This example decodes the MIB and SystemInformationBlockType1 (SIB1). The latter is transmitted to specify the scheduling of other system information, along with aspects of the cell identity such as Public Land Mobile Network (PLMN) identity. Although SIB1 is transmitted in a fixed time schedule, the resource allocation of the PDSCH carrying SIB1 is dynamic and is indicated in an associated DCI message.

Load and Process I/Q Waveform

MATLAB® can be used to acquire I/Q data from a wide range of instruments using the Instrument Control Toolbox™. In this example a single antenna I/Q capture of an eNodeB with two transmit antennas is used. The capture is performed at 15.36 Msamples/s which is sufficient to correctly sample all valid eNodeB bandwidths up to 10 MHz: 1.4 MHz, 3 MHz, 5 MHz, 10 MHz. The captured data is stored in the file eNodeBOutput.mat.

Alternatively, a suitable LTE signal can be generated using the LTE Toolbox. This can be controlled by the variable `loadFromFile`.

```
loadFromFile = 1; % Set to 0 to generate the eNodeB output locally
```

The MIB corresponds to one BCH transport block. The BCH Transmission Time Interval (TTI), or the time needed to transmit a single transport block, is 40msec or 4 frames. The BCH is transmitted in 4

parts, each part mapped to the first subframe (subframe 0) of a frame and it is possible that each transmission is independently decodable, depending on signal conditions. To ensure that subframe 0 is received, the capture should be at least 11 subframes long, to account for the possibility that the capture is started during subframe 0. For poor signal conditions, all 4 parts of the TTI may be required, in which case the capture should be at least 41 subframes long. A similar situation applies for SIB1; it is transmitted in subframe 5 of every even frame, with four different Redundancy Versions (RVs) being transmitted consecutively giving an overall period of 80msec or 8 frames. Therefore 21 subframes of capture are required to ensure reception of a single RV of SIB1 (in subframe 5 of an even frame), but up to 81 subframes of capture are required if signal conditions are such that all RVs need to be combined.

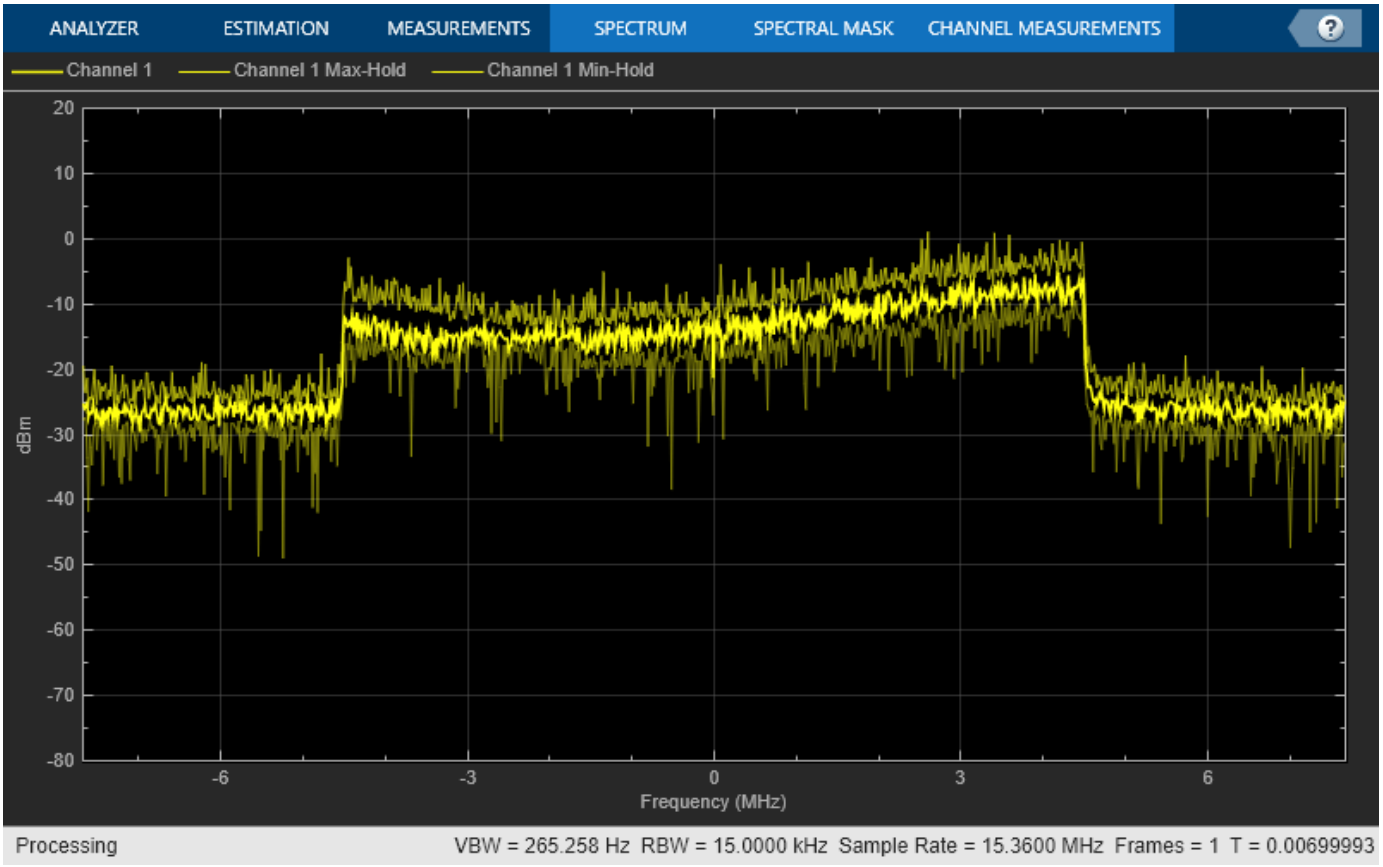
```
if loadFromFile
    load eNodeBOutput.mat           % Load I/Q capture of eNodeB output
    eNodeBOutput = double(eNodeBOutput)/32768; % Scale samples
    sr = 15.36e6;                   % Sampling rate for loaded samples
else
    rmc = lteRMCDL('R.3'); %#ok<UNRCH>
    rmc.NCellID = 17;
    rmc.TotSubframes = 41;
    rmc.PDSCH.RNTI = 61;
    % SIB parameters
    rmc.SIB.Enable = 'On';
    rmc.SIB.DCIFormat = 'Format1A';
    rmc.SIB.AllocationType = 0;
    rmc.SIB.VRBStart = 8;
    rmc.SIB.VRBLength = 8;
    rmc.SIB.Gap = 0;
    % SIB data field filled with random bits, this is not a valid SIB
    % message
    rmc.SIB.Data = randi([0 1],176,1);
    [eNodeBOutput,~,info] = lteRMCDLTool(rmc,[1;0;0;1]);
    sr = info.SamplingRate;     % Sampling rate of generated samples
end
```

Prior to decoding the MIB, the UE does not know the full system bandwidth. The primary and secondary synchronization signals (PSS and SSS) and the PBCH (containing the MIB) all lie in the central 72 subcarriers (6 resource blocks) of the system bandwidth, allowing the UE to initially demodulate just this central region. Therefore the bandwidth is initially set to 6 resource blocks. The I/Q waveform needs to be resampled accordingly. At this stage we also display the spectrum of the input signal `eNodeBOutput`.

```
% Set up some housekeeping variables:
% separator for command window logging
separator = repmat('-',1,50);
% plots
if (~exist('channelFigure','var') || ~isvalid(channelFigure))
    channelFigure = figure('Visible','off');
end
[spectrumScope,synchCorrPlot,pdcchConstDiagram] = ...
    hSIB1RecoveryExamplePlots(channelFigure,sr);
% PDSCH EVM
pdschEVM = comm.EVM();
pdschEVM.MaximumEVMOutputPort = true;

% The sampling rate for the initial cell search is established using
% lteOFDMInfo configured for 6 resource blocks. enb.CyclicPrefix is set
% temporarily in the call to lteOFDMInfo to suppress a default value
```

```
% warning (it does not affect the sampling rate).
enb = struct;                   % eNodeB config structure
enb.NDLRB = 6;                  % Number of resource blocks
ofdmInfo = lteOFDMInfo(setfield(enb,'CyclicPrefix','Normal')); %#ok<SFLD>

if (isempty(eNodeBOutput))
    fprintf('\nReceived signal must not be empty.\n');
    return;
end

% Display received signal spectrum
fprintf('\nPlotting received signal spectrum...\n');
spectrumScope(awgn(eNodeBOutput, 100.0));

if (sr~=ofdmInfo.SamplingRate)
    if (sr < ofdmInfo.SamplingRate)
        warning('The received signal sampling rate (%0.3fMs/s) is lower than the desired samplin
    end
    fprintf('\nResampling from %0.3fMs/s to %0.3fMs/s for cell search / MIB decoding...\n',sr/1e
else
    fprintf('\nResampling not required; received signal is at desired sampling rate for cell sea
end
% Downsample received signal
nSamples = ceil(ofdmInfo.SamplingRate/round(sr)*size(eNodeBOutput,1));
nRxAnts = size(eNodeBOutput, 2);
downsampled = zeros(nSamples, nRxAnts);
for i=1:nRxAnts
    downsampled(:,i) = resample(eNodeBOutput(:,i), ofdmInfo.SamplingRate, round(sr));
end
```

```
Plotting received signal spectrum...

Resampling from 15.360Ms/s to 1.920Ms/s for cell search / MIB decoding...
```



Cell Search, Cyclic Prefix Length and Duplex Mode Detection

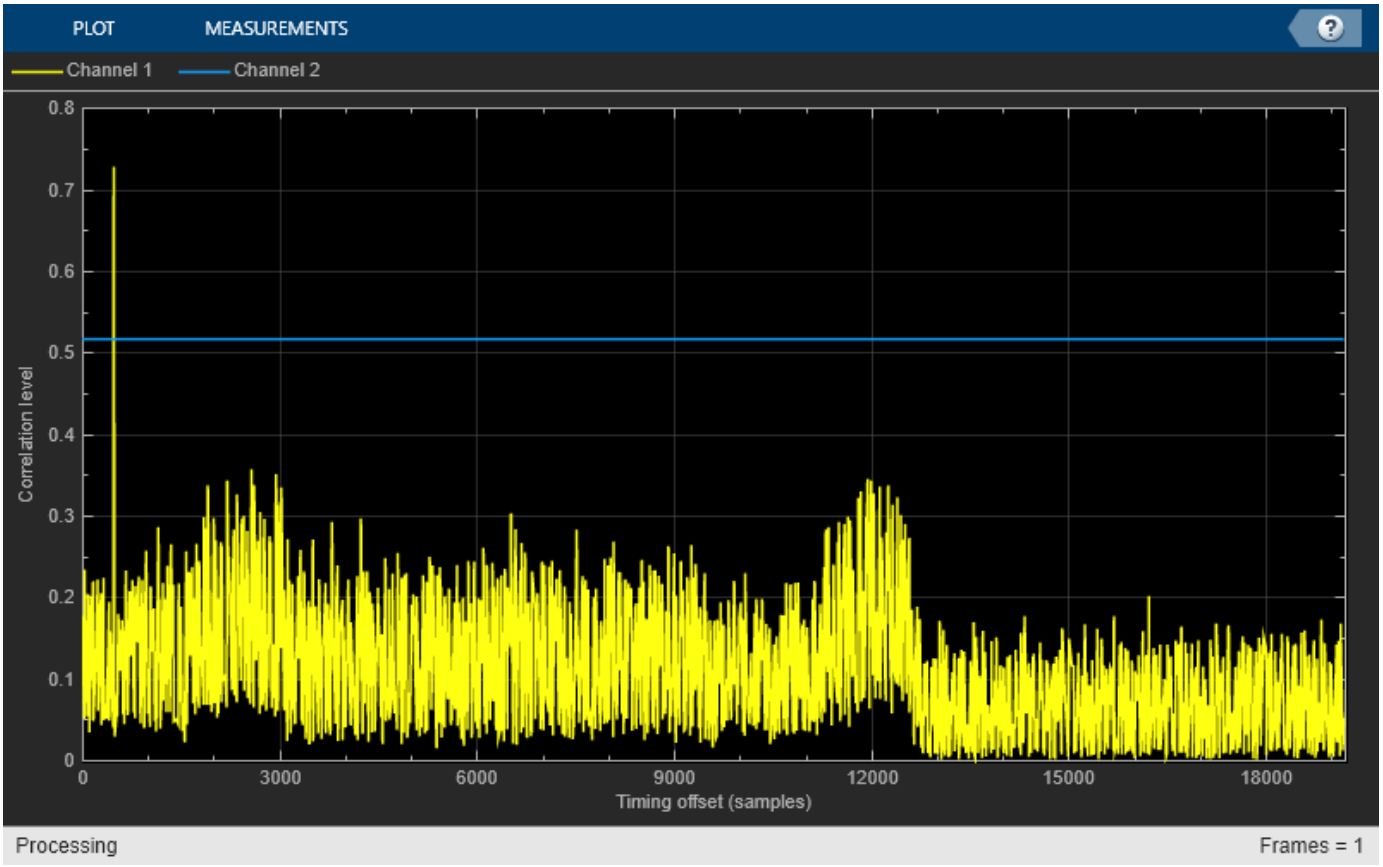
Call `lteCellSearch` to obtain the cell identity and timing offset `offset` to the first frame head. The cell search is repeated for each combination of cyclic prefix length and duplex mode, and the combination with the strongest correlation allows these parameters to be identified. A plot of the correlation between the received signal and the PSS/SSS for the detected cell identity is produced. The PSS is detected using time-domain correlation and the SSS is detected using frequency-domain correlation. Prior to SSS detection, frequency offset estimation/correction using cyclic prefix correlation is performed. The time-domain PSS detection is robust to small frequency offsets but larger offsets may degrade the PSS correlation.

```
fprintf('\nPerforming cell search...\n');

% Set up duplex mode and cyclic prefix length combinations for search; if
% either of these parameters is configured in |enb| then the value is
% assumed to be correct
if (~isfield(enb,'DuplexMode'))
    duplexModes = {'TDD' 'FDD'};
else
    duplexModes = {enb.DuplexMode};
end
if (~isfield(enb,'CyclicPrefix'))
    cyclicPrefixes = {'Normal' 'Extended'};
else
    cyclicPrefixes = {enb.CyclicPrefix};
end
```

```
% Perform cell search across duplex mode and cyclic prefix length
% combinations and record the combination with the maximum correlation; if
% multiple cell search is configured, this example will decode the first
% (strongest) detected cell
searchalg.MaxCellCount = 1;
searchalg.SSSDetection = 'PostFFT';
peakMax = -Inf;
for duplexMode = duplexModes
    for cyclicPrefix = cyclicPrefixes
        enb.DuplexMode = duplexMode{1};
        enb.CyclicPrefix = cyclicPrefix{1};
        [enb.NCellID, offset, peak] = lteCellSearch(enb, downsampled, searchalg);
        enb.NCellID = enb.NCellID(1);
        offset = offset(1);
        peak = peak(1);
        if (peak>peakMax)
            enbMax = enb;
            offsetMax = offset;
            peakMax = peak;
        end
    end
end

% Use the cell identity, cyclic prefix length, duplex mode and timing
% offset which gave the maximum correlation during cell search
enb = enbMax;
offset = offsetMax;

% Compute the correlation for each of the three possible primary cell
% identities; the peak of the correlation for the cell identity established
% above is compared with the peak of the correlation for the other two
% primary cell identities in order to establish the quality of the
% correlation.
corr = cell(1,3);
idGroup = floor(enbMax.NCellID/3);
for i = 0:2
    enb.NCellID = idGroup*3 + mod(enbMax.NCellID + i,3);
    [~,corr{i+1}] = lteDLFrameOffset(enb, downsampled);
    corr{i+1} = sum(corr{i+1},2);
end
threshold = 1.3 * max([corr{2}; corr{3}]); % multiplier of 1.3 empirically obtained
if (max(corr{1})<threshold)
    warning('Synchronization signal correlation was weak; detected cell identity may be incorrec
end
% Return to originally detected cell identity
enb.NCellID = enbMax.NCellID;

% Plot PSS/SSS correlation and threshold
synchCorrPlot.YLimits = [0 max([corr{1}; threshold])*1.1];
synchCorrPlot([corr{1} threshold*ones(size(corr{1}))]);

% Perform timing synchronization
fprintf('Timing offset to frame start: %d samples\n',offset);
downsampled = downsampled(1+offset:end,:);
enb.NSubframe = 0;

% Show cell-wide settings
```

```
fprintf('Cell-wide settings after cell search:\n');
disp(enb);
```

```
Performing cell search...
Timing offset to frame start: 481 samples
Cell-wide settings after cell search:
           NDLRB: 6
      DuplexMode: 'FDD'
    CyclicPrefix: 'Normal'
         NCellID: 17
       NSubframe: 0
```



Frequency Offset Estimation and Correction

Prior to OFDM demodulation, any significant frequency offset must be removed. The frequency offset in the I/Q waveform is estimated and corrected using `lteFrequencyOffset` and `lteFrequencyCorrect`. The frequency offset is estimated by means of correlation of the cyclic prefix and therefore can estimate offsets up to +/- half the subcarrier spacing i.e. +/- 7.5kHz.

```
fprintf('\nPerforming frequency offset estimation...\n');
% For TDD, TDDConfig and SSC are defaulted to 0. These parameters are not
% established in the system until SIB1 is decoded, so at this stage the
% values of 0 make the most conservative assumption (fewest downlink
% subframes and shortest special subframe).
if (strcmpi(enb.DuplexMode,'TDD'))
```

```
    enb.TDDConfig = 0;
    enb.SSC = 0;
end
delta_f = lteFrequencyOffset(enb, downsampled);
fprintf('Frequency offset: %0.3fHz\n',delta_f);
downsampled = lteFrequencyCorrect(enb, downsampled, delta_f);
```

```
Performing frequency offset estimation...
Frequency offset: -14.902Hz
```

OFDM Demodulation and Channel Estimation

The OFDM downsampled I/Q waveform is demodulated to produce a resource grid `rxgrid`. This is used to perform channel estimation. `hest` is the channel estimate, `nest` is an estimate of the noise (for MMSE equalization) and `cec` is the channel estimator configuration.

For channel estimation the example assumes 4 cell specific reference signals. This means that channel estimates to each receiver antenna from all possible cell-specific reference signal ports are available. The true number of cell-specific reference signal ports is not yet known. The channel estimation is only performed on the first subframe, i.e. using the first `L` OFDM symbols in `rxgrid`.

A conservative 13-by-9 pilot averaging window is used, in frequency and time, to reduce the impact of noise on pilot estimates during channel estimation.

```
% Channel estimator configuration
cec.PilotAverage = 'UserDefined';     % Type of pilot averaging
cec.FreqWindow = 13;                  % Frequency window size
cec.TimeWindow = 9;                   % Time window size
cec.InterpType = 'cubic';             % 2D interpolation type
cec.InterpWindow = 'Centered';        % Interpolation window type
cec.InterpWinSize = 1;                % Interpolation window size

% Assume 4 cell-specific reference signals for initial decoding attempt;
% ensures channel estimates are available for all cell-specific reference
% signals
enb.CellRefP = 4;

fprintf('Performing OFDM demodulation...\n\n');

griddims = lteResourceGridSize(enb); % Resource grid dimensions
L = griddims(2);                     % Number of OFDM symbols in a subframe
% OFDM demodulate signal
rxgrid = lteOFDMDemodulate(enb, downsampled);
if (isempty(rxgrid))
    fprintf('After timing synchronization, signal is shorter than one subframe so no further demo
    return;
end
% Perform channel estimation
[hest, nest] = lteDLChannelEstimate(enb, cec, rxgrid(:,1:L,:));
```

```
Performing OFDM demodulation...
```

PBCH Demodulation, BCH Decoding, MIB Parsing

The MIB is now decoded along with the number of cell-specific reference signal ports transmitted as a mask on the BCH CRC. The function `ltePBCHDecode` establishes frame timing modulo 4 and

returns this in the `nfmod4` parameter. It also returns the MIB bits in vector `mib` and the true number of cell-specific reference signal ports which is assigned into `enb.CellRefP` at the output of this function call. If the number of cell-specific reference signal ports is decoded as `enb.CellRefP=0`, this indicates a failure to decode the BCH. The function `lteMIB` is used to parse the bit vector `mib` and add the relevant fields to the configuration structure `enb`. After MIB decoding, the detected bandwidth is present in `enb.NDLRB`.

```
% Decode the MIB
% Extract resource elements (REs) corresponding to the PBCH from the first
% subframe across all receive antennas and channel estimates
fprintf('Performing MIB decoding...\n');
pbchIndices = ltePBCHIndices(enb);
[pbchRx, pbchHest] = lteExtractResources( ...
    pbchIndices, rxgrid(:,1:L,:), hest(:,1:L,:,:));

% Decode PBCH
[bchBits, pbchSymbols, nfmod4, mib, enb.CellRefP] = ltePBCHDecode( ...
    enb, pbchRx, pbchHest, nest);

% Parse MIB bits
enb = lteMIB(mib, enb);

% Incorporate the nfmod4 value output from the function ltePBCHDecode, as
% the NFrame value established from the MIB is the System Frame Number
% (SFN) modulo 4 (it is stored in the MIB as floor(SFN/4))
enb.NFrame = enb.NFrame+nfmod4;

% Display cell wide settings after MIB decoding
fprintf('Cell-wide settings after MIB decoding:\n');
disp(enb);

if (enb.CellRefP==0)
    fprintf('MIB decoding failed (enb.CellRefP=0).\n\n');
    return;
end
if (enb.NDLRB==0)
    fprintf('MIB decoding failed (enb.NDLRB=0).\n\n');
    return;
end
```

```
Performing MIB decoding...
Cell-wide settings after MIB decoding:
            NDLRB: 50
       DuplexMode: 'FDD'
     CyclicPrefix: 'Normal'
          NCellID: 17
        NSubframe: 0
          CellRefP: 2
    PHICHDuration: 'Normal'
               Ng: 'One'
           NFrame: 406
```

OFDM Demodulation on Full Bandwidth

Now that the signal bandwidth is known, the signal is resampled to the nominal sampling rate used by LTE Toolbox for that bandwidth (see `lteOFDMModulate` for details). Frequency offset estimation

and correction is performed on the resampled signal. Timing synchronization and OFDM demodulation are then performed.

```
fprintf('Restarting reception now that bandwidth (NDLRB=%d) is known...\n',enb.NDLRB);

% Resample now we know the true bandwidth
ofdmInfo = lteOFDMInfo(enb);
if (sr~=ofdmInfo.SamplingRate)
    if (sr < ofdmInfo.SamplingRate)
        warning('The received signal sampling rate (%0.3fMs/s) is lower than the desired sampling
    end
    fprintf('\nResampling from %0.3fMs/s to %0.3fMs/s...\n',sr/1e6,ofdmInfo.SamplingRate/1e6);
else
    fprintf('\nResampling not required; received signal is at desired sampling rate for NDLRB=%d
end
nSamples = ceil(ofdmInfo.SamplingRate/round(sr)*size(eNodeBOutput,1));
resampled = zeros(nSamples, nRxAnts);
for i = 1:nRxAnts
    resampled(:,i) = resample(eNodeBOutput(:,i), ofdmInfo.SamplingRate, round(sr));
end

% Perform frequency offset estimation and correction
fprintf('\nPerforming frequency offset estimation...\n');
delta_f = lteFrequencyOffset(enb, resampled);
fprintf('Frequency offset: %0.3fHz\n',delta_f);
resampled = lteFrequencyCorrect(enb, resampled, delta_f);

% Find beginning of frame
fprintf('\nPerforming timing offset estimation...\n');
offset = lteDLFrameOffset(enb, resampled);
fprintf('Timing offset to frame start: %d samples\n',offset);
% Aligning signal with the start of the frame
resampled = resampled(1+offset:end,:);

% OFDM demodulation
fprintf('\nPerforming OFDM demodulation...\n\n');
rxgrid = lteOFDMDemodulate(enb, resampled);

Restarting reception now that bandwidth (NDLRB=50) is known...

Resampling not required; received signal is at desired sampling rate for NDLRB=50 (15.360Ms/s).

Performing frequency offset estimation...
Frequency offset: 5.221Hz

Performing timing offset estimation...
Timing offset to frame start: 3848 samples

Performing OFDM demodulation...
```

SIB1 Decoding

The following steps are performed in this section:

- Physical Control Format Indicator Channel (PCFICH) demodulation, CFI decoding
- PDCCH decoding

- Blind PDCCH search
- SIB bits recovery: PDSCH demodulation and DL-SCH decoding
- Buffering and resetting of the DL-SCH HARQ state

After recovery the SIB CRC should be 0.

These decoding steps are performed in a loop for each occurrence of a subframe carrying SIB1 in the received signal. As mentioned above, the SIB1 is transmitted in subframe 5 of every even frame, so the input signal is first checked to establish that at least one occurrence of SIB1 is present. For each SIB1 subframe, the channel estimate magnitude response is plotted, as is the constellation of the received PDCCH.

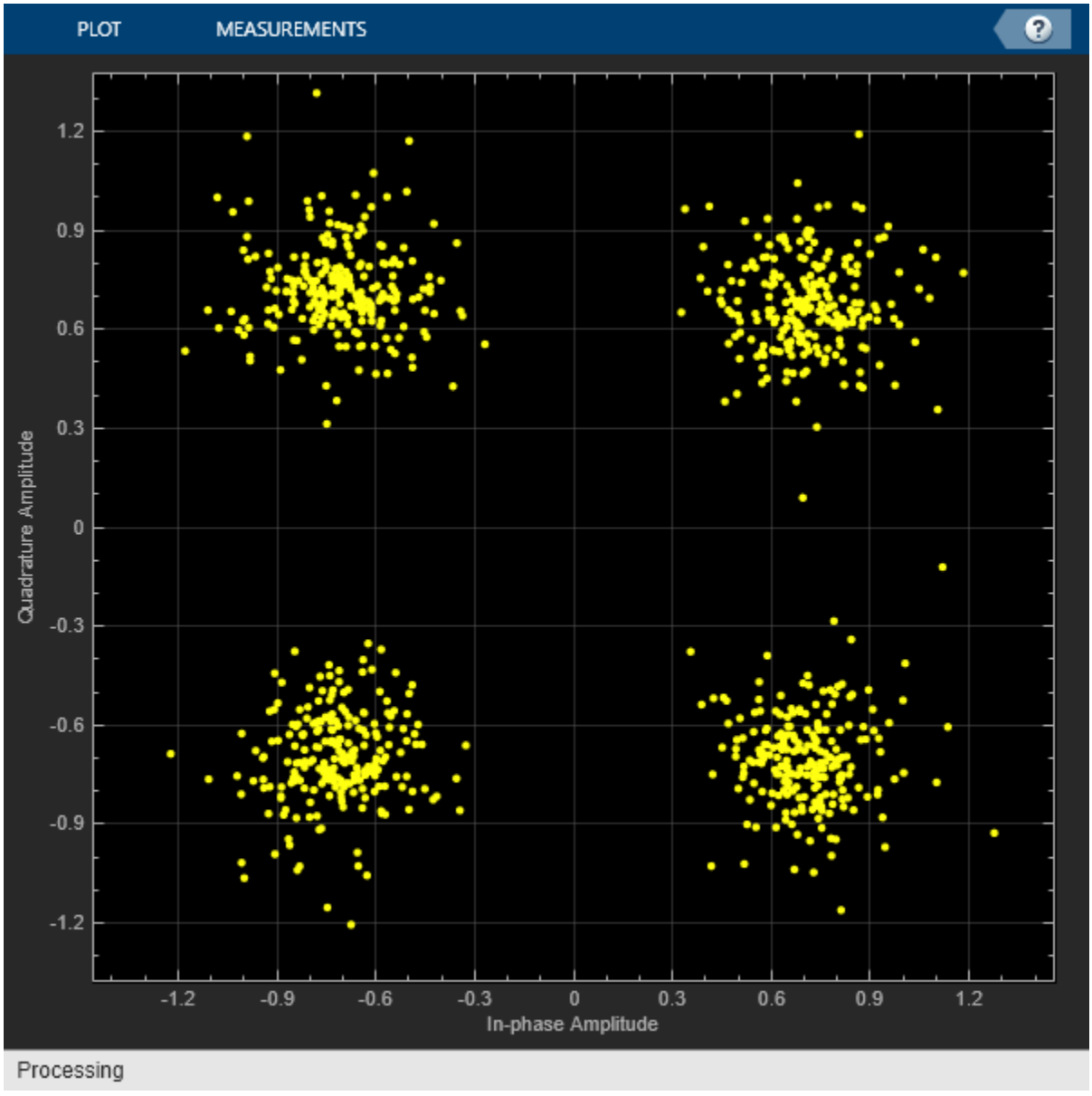
```
% Check this frame contains SIB1, if not advance by 1 frame provided we
% have enough data, terminate otherwise.
if (mod(enb.NFrame,2)~=0)
    if (size(rxgrid,2)>=(L*10))
        rxgrid(:,1:(L*10),:) = [];
        fprintf('Skipping frame %d (odd frame number does not contain SIB1).\n\n',enb.NFrame);
    else
        rxgrid = [];
    end
    enb.NFrame = enb.NFrame + 1;
end

% Advance to subframe 5, or terminate if we have less than 5 subframes
if (size(rxgrid,2)>=(L*5))
    rxgrid(:,1:(L*5),:) = [];   % Remove subframes 0 to 4
else
    rxgrid = [];
end
enb.NSubframe = 5;

if (isempty(rxgrid))
    fprintf('Received signal does not contain a subframe carrying SIB1.\n\n');
end

% Reset the HARQ buffers
decState = [];

% While we have more data left, attempt to decode SIB1
while (size(rxgrid,2) > 0)

    fprintf('%s\n',separator);
    fprintf('SIB1 decoding for frame %d\n',mod(enb.NFrame,1024));
    fprintf('%s\n\n',separator);

    % Reset the HARQ buffer with each new set of 8 frames as the SIB1
    % info may be different
    if (mod(enb.NFrame,8)==0)
        fprintf('Resetting HARQ buffers.\n\n');
        decState = [];
    end

    % Extract current subframe
    rxsubframe = rxgrid(:,1:L,:);

    % Perform channel estimation
```

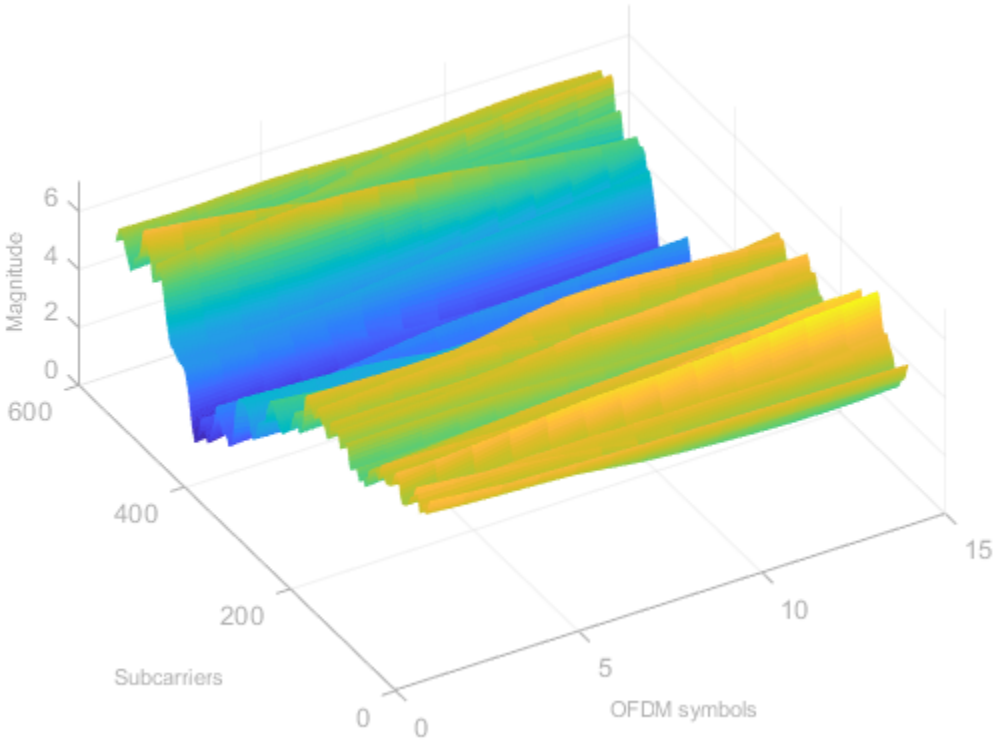
```
[hest,nest] = lteDLChannelEstimate(enb, cec, rxsubframe);

% PCFICH demodulation, CFI decoding. The CFI is now demodulated and
% decoded using similar resource extraction and decode functions to
% those shown already for BCH reception. lteExtractResources is used to
% extract REs corresponding to the PCFICH from the received subframe
% rxsubframe and channel estimate hest.
fprintf('Decoding CFI...\n\n');
pcfichIndices = ltePCFICHIndices(enb);  % Get PCFICH indices
[pcfichRx, pcfichHest] = lteExtractResources(pcfichIndices, rxsubframe, hest);
% Decode PCFICH
cfiBits = ltePCFICHDecode(enb, pcfichRx, pcfichHest, nest);
cfi = lteCFIDecode(cfiBits); % Get CFI
if (isfield(enb,'CFI') && cfi~=enb.CFI)
    release(pdcchConstDiagram);
end
enb.CFI = cfi;
fprintf('Decoded CFI value: %d\n\n', enb.CFI);

% For TDD, the PDCCH must be decoded blindly across possible values of
% the PHICH configuration factor m_i (0,1,2) in TS36.211 Table 6.9-1.
% Values of m_i = 0, 1 and 2 can be achieved by configuring TDD
% uplink-downlink configurations 1, 6 and 0 respectively.
if (strcmpi(enb.DuplexMode,'TDD'))
    tddConfigs = [1 6 0];
else
    tddConfigs = 0; % not used for FDD, only used to control while loop
end
alldci = {};
while (isempty(alldci) && ~isempty(tddConfigs))
    % Configure TDD uplink-downlink configuration
    if (strcmpi(enb.DuplexMode,'TDD'))
        enb.TDDConfig = tddConfigs(1);
    end
    tddConfigs(1) = [];
    % PDCCH demodulation. The PDCCH is now demodulated and decoded
    % using similar resource extraction and decode functions to those
    % shown already for BCH and CFI reception
    pdcchIndices = ltePDCCHIndices(enb); % Get PDCCH indices
    [pdcchRx, pdcchHest] = lteExtractResources(pdcchIndices, rxsubframe, hest);
    % Decode PDCCH and plot constellation
    [dciBits, pdcchSymbols] = ltePDCCHDecode(enb, pdcchRx, pdcchHest, nest);
    pdcchConstDiagram(pdcchSymbols);

    % PDCCH blind search for System Information (SI) and DCI decoding.
    % The LTE Toolbox provides full blind search of the PDCCH to find
    % any DCI messages with a specified RNTI, in this case the SI-RNTI.
    fprintf('PDCCH search for SI-RNTI...\n\n');
    pdcch = struct('RNTI', 65535);
    pdcch.ControlChannelType = 'PDCCH';
    pdcch.EnableCarrierIndication = 'Off';
    pdcch.SearchSpace = 'Common';
    pdcch.EnableMultipleCSIRequest = 'Off';
    pdcch.EnableSRSRequest = 'Off';
    pdcch.NTxAnts = 1;
    alldci = ltePDCCHSearch(enb, pdcch, dciBits); % Search PDCCH for DCI
end
```

```
% If DCI was decoded, proceed with decoding PDSCH / DL-SCH
for i = 1:numel(alldci)

    dci = alldci{i};
    fprintf('DCI message with SI-RNTI:\n');
    disp(dci);
    % Get the PDSCH configuration from the DCI
    [pdsch, trblklen] = hPDSCHConfiguration(enb, dci, pdcch.RNTI);

    % If a PDSCH configuration was created, proceed with decoding PDSCH
    % / DL-SCH
    if ~isempty(pdsch)

        pdsch.NTurboDecIts = 5;
        fprintf('PDSCH settings after DCI decoding:\n');
        disp(pdsch);

        % PDSCH demodulation and DL-SCH decoding to recover SIB bits.
        % The DCI message is now parsed to give the configuration of
        % the corresponding PDSCH carrying SIB1, the PDSCH is
        % demodulated and finally the received bits are DL-SCH decoded
        % to yield the SIB1 bits.

        fprintf('Decoding SIB1...\n\n');
        % Get PDSCH indices
        [pdschIndices,pdschIndicesInfo] = ltePDSCHIndices(enb, pdsch, pdsch.PRBSet);
        [pdschRx, pdschHest] = lteExtractResources(pdschIndices, rxsubframe, hest);
        % Decode PDSCH
        [dlschBits,pdschSymbols] = ltePDSCHDecode(enb, pdsch, pdschRx, pdschHest, nest);
        % Decode DL-SCH with soft buffer input/output for HARQ combining
        if ~isempty(decState)
            fprintf('Recombining with previous transmission.\n\n');
        end
        [sib1, crc, decState] = lteDLSCHDecode(enb, pdsch, trblklen, dlschBits, decState);

        % Compute PDSCH EVM
        recoded = lteDLSCH(enb, pdsch, pdschIndicesInfo.G, sib1);
        remod = ltePDSCH(enb, pdsch, recoded);
        [~,refSymbols] = ltePDSCHDecode(enb, pdsch, remod);
        [rmsevm,peakevm] = pdschEVM(refSymbols{1}, pdschSymbols{1});
        fprintf('PDSCH RMS EVM: %0.3f%%\n',rmsevm);
        fprintf('PDSCH Peak EVM: %0.3f%%\n\n',peakevm);

        fprintf('SIB1 CRC: %d\n',crc);
        if crc == 0
            fprintf('Successful SIB1 recovery.\n\n');
        else
            fprintf('SIB1 decoding failed.\n\n');
        end

    else
        % Indicate that creating a PDSCH configuration from the DCI
        % message failed
        fprintf('Creating PDSCH configuration from DCI message failed.\n\n');
    end

end
if (numel(alldci)==0)
```

```
        % Indicate that DCI decoding failed
        fprintf('DCI decoding failed.\n\n');
    end

    % Update channel estimate plot
    figure(channelFigure);
    surf(abs(hest(:,:,1,1)));
    hSIB1RecoveryExamplePlots(channelFigure);
    channelFigure.CurrentAxes.XLim = [0 size(hest,2)+1];
    channelFigure.CurrentAxes.YLim = [0 size(hest,1)+1];

    % Skip 2 frames and try SIB1 decoding again, or terminate if we
    % have less than 2 frames left.
    if (size(rxgrid,2)>=(L*20))
        rxgrid(:,1:(L*20),:) = [];   % Remove 2 more frames
    else
        rxgrid = []; % Less than 2 frames left
    end
    enb.NFrame = mod(enb.NFrame + 2,1024);

end

------------------------------------------------
SIB1 decoding for frame 406
------------------------------------------------

Decoding CFI...

Decoded CFI value: 2

PDCCH search for SI-RNTI...

DCI message with SI-RNTI:
           DCIFormat: 'Format1A'
                 CIF: 0
      AllocationType: 0
          Allocation: [1x1 struct]
           ModCoding: 6
              HARQNo: 0
             NewData: 0
                  RV: 1
            TPCPUCCH: 0
            TDDIndex: 0
          SRSRequest: 0
    HARQACKResOffset: 0

PDSCH settings after DCI decoding:
             RNTI: 65535
           PRBSet: [8x1 uint64]
          NLayers: 2
              CSI: 'On'
      Modulation: {'QPSK'}
               RV: 1
        TxScheme: 'TxDiversity'
    NTurboDecIts: 5

Decoding SIB1...
```

```
PDSCH RMS EVM: 27.072%
PDSCH Peak EVM: 75.981%

SIB1 CRC: 0
Successful SIB1 recovery.
```




Magnitude
6
4
2
0
600
400
200
0
Subcarriers
0
5
10
15
OFDM symbols

Scan and Decode LTE Waveform

This example shows how to capture and decode an LTE signal obtained either from a file or from radio by using LTE Toolbox™ software and various hardware support packages (HSPs). Before the user equipment (UE) can communicate with the network, it must perform cell search and selection procedures, and then obtain initial system information. This process involves acquiring slot and frame synchronization, determining the cell identity, and decoding the MIB and system information blocks (SIBs). This example launches a graphical user interface (GUI), which enables you to:

- Provide all the inputs for scanning LTE waveforms
- Capture live LTE waveforms using different radios
- Analyze decoded MIB and SIB1 information
- Visualize the reference signal measurements

Required Hardware and Software

To run this example using captured signals from a file, you need this file:

Baseband file (*.bb): Uses the `comm.BasebandFileReader` object to read previously captured LTE signals.

To receive signals in real time, you also need one of these software-defined radio (SDR) devices and the corresponding SDR support package add-on:

- Communications Toolbox Support Package for RTL-SDR Radio
- Communications Toolbox Support Package for ADALM-PLUTO Radio
- Communications Toolbox Support Package for USRP Radio
- Communications Toolbox Support Package for Zynq SDR Radio
- For a full list of Communications Toolbox supported SDR platforms, refer to the Supported Hardware section of Software Defined Radio (SDR) discovery page.

Main GUI Call

This code launches the GUI, which you can use to capture and decode an LTE waveform.

```
hScanLTEWaveformGUI;
```

User Inputs

These search settings need to be provided through the GUI to perform the LTE scanning operation:

Signal source: Signal source can be any one of these options. By default the 'Signal source' is set to 'File'.

1. **File**: Over-the-air signals written to a baseband file (*.bb).
2. **RTL-SDR**: RTL-SDR radio.
3. **ADALM-PLUTO**: ADALM-PLUTO radio.
4. **USRP-B200/B210**: USRP® B series (either B200 or B210) radio.
5. **USRP-N200/N210**: USRP® N series (either N200 or N210) radio.
6. **USRP-X300/X310**: USRP® X series (either X300 or X310) radio.
7. **ZedBoard-FMC2/3/4**: Zynq® Evaluation and Development Board (ZedBoard) with FMC 2/3/4.
8. **ZC706-FMC2/3/4**: Zynq®-7000 SoC ZC706 Evaluation Kit with FMC 2/3/4.

All of these radios are used to capture the live LTE waveforms for the specific radio settings.

Signal file name: If **Signal source** is a **File**, then you need to provide a baseband file (*.bb) that contains an LTE signal.

Center frequency: Center frequency of the saved LTE signal in Hz.

LTE frequency band: For **Signal source** other than **File**, you can search a range of frequencies by setting **LTE frequency band** to any one of the available 44 LTE frequency bands. Out of these 44 bands, RTL-SDR supports only 14 bands due to its **Center frequency** limitation.

User frequency (MHz): You can search for an LTE signal in a custom frequency band, which can be a single frequency or range of frequencies having minimum and maximum values. The supported formats for this field are: [Fmin, Fmax] and [Fmin1, Fmax1; Fmin2, Fmax2] where (Fmin, Fmin1 & Fmin2) and (Fmax, Fmax1 & Fmax2) represents the minimum and maximum frequencies in a specified range, respectively.

Frequency offset (ppm): For RTL-SDR and USRP radios as **Signal source**, you can provide carrier frequency offset in parts per million (ppm) within the range [-1e4, 1e4].

Information block type: You can select either **MIB** or **MIB & SIB1**. For RTL-SDR only MIB decoding is supported due to its bandwidth limitation.

After setting the search parameters, you need to click the **Start search** button to start searching. You can stop searching at any time by using **Stop** push button.

Decoding Procedure

The decoding procedure of this example can be described as:

1. Capture a suitable number of frames of an LTE signal using SDR hardware or from a baseband file.
2. Determine and correct the frequency offset of the received signal.
3. Perform a blind cell search to determine the cell identity.
4. Synchronize the captured signal to the start of an LTE frame.
5. OFDM demodulate the received signal to get an LTE resource grid.
6. Perform a channel estimation for the received signal.
7. Decode the MIB for each captured frame to determine cell-wide settings.
8. Decode the CFI and PDCCH for each subframe within the captured signal.
9. Plot the reference signal measurements (RSRQ, RSRP, RSSI) on the GUI. Using **Reference signal measurement plot** you can select one of the reference signals and plot it against frequency.
10. Based on **Information block type**, decode the MIB and SIB1 information and update these fields on GUI.

The MIB information fields are:

- DuplexMode - Frame structure type
- CyclicPrefix - Cyclic prefix length
- NDLRB - Number of downlink resource blocks
- CellRefP - Cell-specific reference signal antenna ports
- PHICHDuration - PHICH duration
- Ng - HICH group multiplier
- NFrame - Frame number

The SIB1 information fields are:

- RNTI - Radio network temporary identifier value
- PRBSet - Zero-based physical resource block (PRB) indices
- NLayers - Number of transmission layers
- Modulation - Modulation type
- RV - Redundancy version
- TxScheme - Transmission scheme
- CRC - Cyclic Redundancy check

Repeat all the steps in decoding procedure for each frequency in the provided range of frequencies.

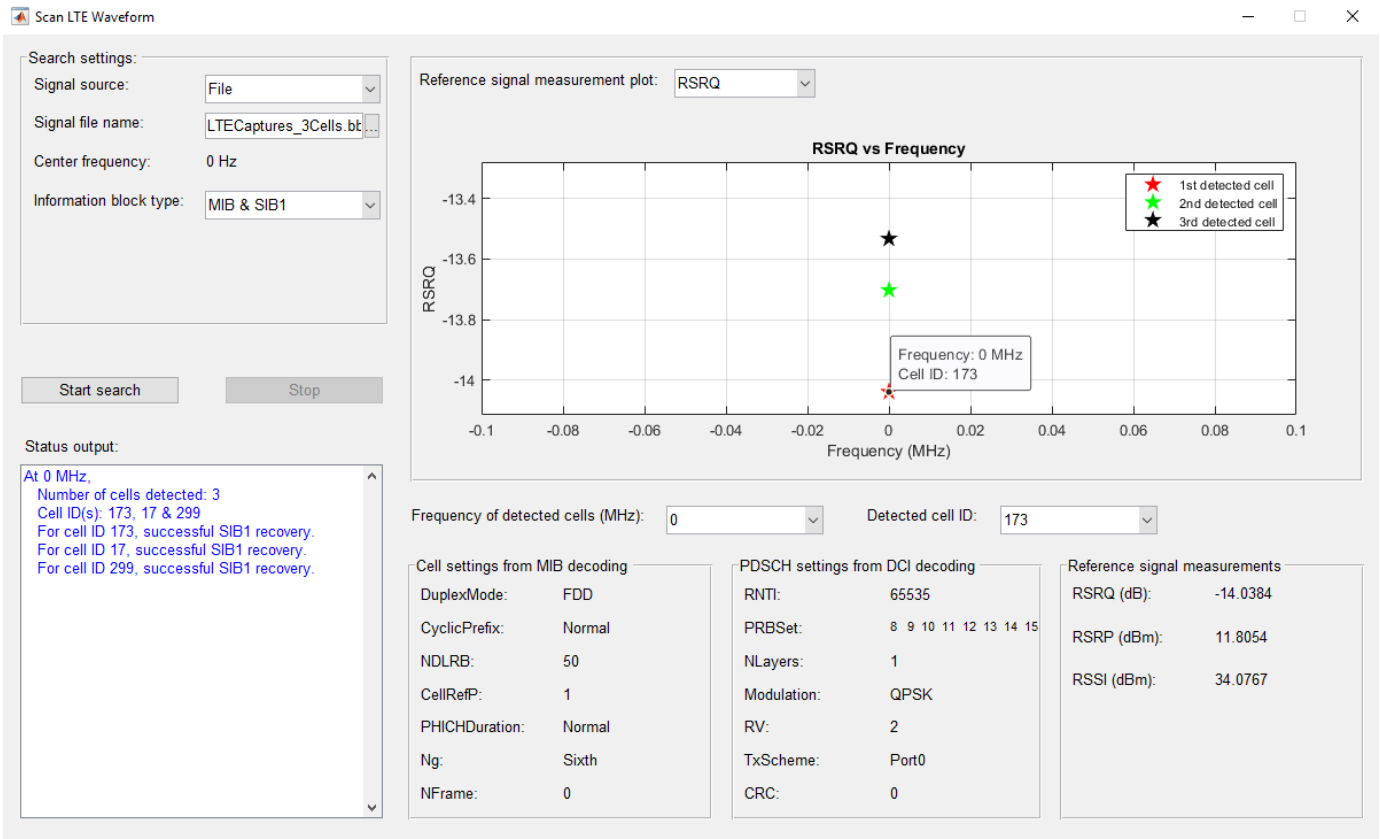
In order to get the decoded information specific to each detected cell, you can use these drop-downs:

- Frequency of detected cells (MHz): Lists the frequencies at which cells are detected.
- Detected cell ID: Lists the cell identities at an LTE frequency selected from **Frequency of detected cells (MHz)**.

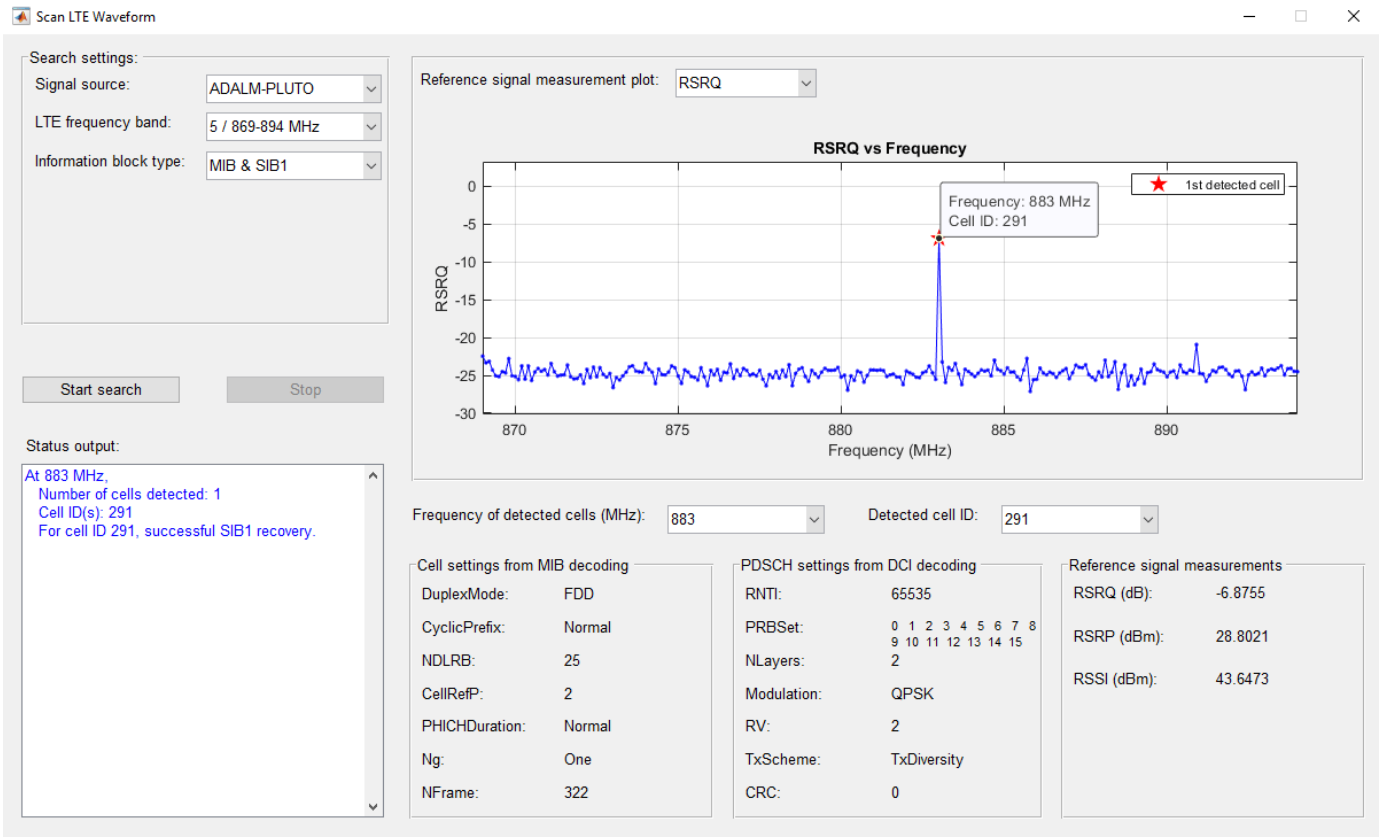
Detected cells are highlighted using markers on the reference signal measurement plot. You can view each cell ID and frequency by clicking on the corresponding marker.

You can observe the detected cell information on the GUI as shown in these figures,

For the default run using **File** as **Signal source**:



Searching over a frequency range using **SDR** as **Signal source**:



Appendix

This example uses this function:

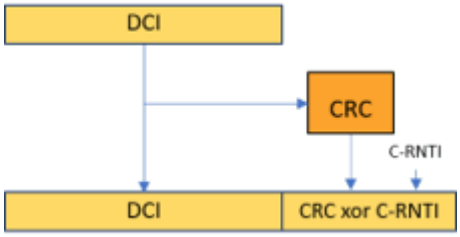
- hScanLTEWaveformGUI.m

UE Detection Using Downlink Signals

This example shows how LTE Toolbox™ can be used to detect the presence of UEs associated with an eNodeB. This is achieved by searching a downlink signal for DCI messages and establishing the set of unique identifiers (C-RNTIs) in use.

Introduction

In LTE, the Physical Downlink Control Channel (PDCCH) carries control information in the form of Downlink Control Information (DCI) messages. The DCI messages transmit uplink or downlink scheduling information from the eNodeB to destination UEs, in order that the UEs can identify the resources required to receive on the Physical Downlink Shared Channel (PDSCH) or to transmit on the Physical Uplink Shared Channel (PUSCH). Each UE is assigned an identifier known as a C-RNTI (Cell Radio Network Temporary Identifier). The C-RNTI is used to scramble the CRC bits of the DCI message for that UE, and also to determine the location of the DCI message within the PDCCH. For more information, see the “PDCCH Blind Search and DCI Decoding” on page 2-169 example.



The aim of this example is to decode the PDCCH and look for candidate DCI messages. The C-RNTI is then obtained from valid messages, giving an indication of the number of UEs being addressed, and their pattern over time. Note that in this example all detected RNTIs are returned, not just the RNTIs in the C-RNTI range defined in TS 36.321 Table 7.1-1 [1]. Keeping these other RNTIs allows this example to be used to identify occurrences of DCI messages associated with System Information (SI-RNTI) or Paging (P-RNTI).

Waveform Under Analysis

In this example, the waveform under analysis is stored in the file ueDetectionWaveform.mat as the variable `rxWaveform`. It is assumed that the bandwidth, duplexing mode, cyclic prefix length, cell identity, number of cell-specific reference signal ports and PHICH configuration are all known, and the corresponding eNodeB configuration variable `enb` is also loaded from the file. It is also possible to use a waveform whose eNodeB configuration is not known, in which case the eNodeB configuration would need to be decoded as shown in the “Cell Search, MIB and SIB1 Recovery” on page 2-351 example.

```
load ueDetectionWaveform.mat;
rxWaveform = double(rxWaveform) / 32768;
```

Synchronization

Frequency offset estimation and correction and timing synchronization are performed.

```
% Perform frequency offset estimation and correction
foffset = lteFrequencyOffset(enb,rxWaveform);
rxWaveform = lteFrequencyCorrect(enb,rxWaveform,foffset);
```

```
% Perform timing synchronization to the first whole subframe of the
% waveform
toffset = lteDLFrameOffset(enb,rxWaveform);
ofdmInfo = lteOFDMInfo(enb);
sfLength = ofdmInfo.SamplingRate * 1e-3;
offsetSubframes = floor(toffset / sfLength);
toffset = toffset - (offsetSubframes * sfLength);
rxWaveform = rxWaveform(1+toffset:end,:);
enb.NSubframe = mod(-offsetSubframes,10);
```

OFDM Demodulation

The I/Q waveform `rxWaveform` is OFDM demodulated to produce the received resource grid `rxgrid`.

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

Channel Estimator Configuration

Channel estimation and equalization will be needed for processing an I/Q waveform captured off-the-air, so the channel estimator parameters are configured.

```
cec.PilotAverage = 'UserDefined';     % Type of pilot averaging
cec.FreqWindow = 13;                  % Frequency window size
cec.TimeWindow = 9;                   % Time window size
cec.InterpType = 'cubic';             % 2D interpolation type
cec.InterpWindow = 'Centered';        % Interpolation window type
cec.InterpWinSize = 1;                % Interpolation window size
```

UE Detection

The downlink waveform under analysis may carry information for a number of UEs. This section attempts to find their RNTIs. The approach applied here uses the fact that for a valid DCI message, the CRC bits are masked with the RNTI. Therefore, assuming there are no errors (CRC = 0), the last 16 decoded bits correspond to the RNTI. This approach is described in Section 5 of [2].

This example implements the following algorithm:

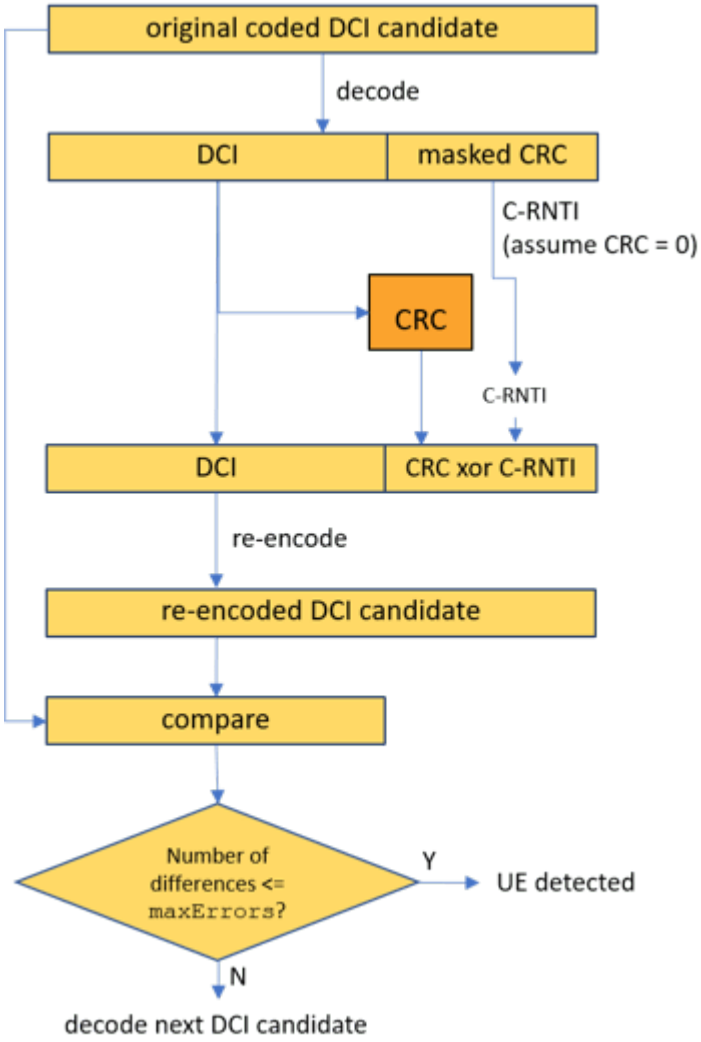
- Consider all PDCCH formats in the UE-specific search space (0...3)
- For each PDCCH format, calculate all possible candidates in the control region that may carry a DCI message.
- For each candidate, attempt to decode a DCI message (considering all possible DCI formats with unique DCI message length).

The following steps are performed to check if the decoded bits constitute a valid DCI message (a graphical representation of the algorithm is also provided below):

- Assume there are no errors in the transmission, i.e. assume CRC = 0. This means the last 16 decoded bits constitute the C-RNTI.
- Separate the DCI message from the 16 C-RNTI bits.
- Calculate new CRC bits and mask them with the C-RNTI.
- Re-encode the DCI message.
- Compare the re-encoded message with the original coded DCI candidate.

Lack of errors in the comparison in this last step means a valid DCI message and C-RNTI (UE identifier) have been detected. In some cases, especially with waveforms captured off-the air, some

errors may occur in the DCI decoding process. Therefore, this example uses a threshold (`maxErrors`) to control how many errors are allowed in the comparison between the received candidate bits and the re-encoded candidate bits.



Two post-processing steps are applied to the list of detected UEs in order to reduce the number of false detections:

- DCI messages having received candidate symbols whose power (in dB) is less than the threshold `minPDCCHPower` are excluded. Without this step, received candidate bits consisting solely of noise (even at a low power) can trigger false detections.
- In the case of overlapping results (i.e. where the candidate indices of one result are a subset of another), the result with the lowest number of bit errors will be selected.

Note that for completeness, the common search space is also searched for DCI format 1A and 1C messages allocating resources for System Information, Paging or Random Access Response messages.

```
% Maximum number of bit errors allowed to consider a detection as valid
maxErrors = 2;
```

```
% Minimum PDCCH power in dB (relative to the cell-specific reference signal
% used for channel estimation) to consider a detection as valid
minPDCCHPower = -5.0;

% Initialize results table
results = table;

% Get DCI formats and lengths for DCI messages in the common search space
pdcchCommon = struct('SearchSpace','Common');
pdcchCommon.ControlChannelType = 'PDCCH';
CIF = 'Off'; % enable or disable Carrier Indicator Field
pdcchCommon.EnableCarrierIndication = CIF;
[dciInfoCommon,pdcchCommon] = getUniqueDCILengths(enb,pdcchCommon);

% Get DCI formats and lengths for DCI messages in the UE-specific search
% space. The length of a DCI message of a given format may be affected by
% various UE-specific higher-layer parameters so the set of different
% unique lengths across those parameters is recorded, along with the
% corresponding parameter combinations
pdcchUESpecific = struct('SearchSpace','UESpecific');
pdcchUESpecific.ControlChannelType = 'PDCCH';
pdcchUESpecific.EnableCarrierIndication = CIF;
[dciInfoUESpecific,pdcchUESpecific] = getUniqueDCILengths(enb,pdcchUESpecific);

% Establish the number of subframes in the waveform
griddims = lteResourceGridSize(enb);
L = griddims(2);
nSubframes = floor(size(rxgrid,2) / L);

% For each subframe in the waveform, attempt to decode the PDCCH
startSubframe = enb.NSubframe;
CFIs = zeros(1,nSubframes);
for i = 0:nSubframes-1

    % Extract the current subframe
    rxSubframe = rxgrid(:,(i*L) + (1:L),:);

    % Perform channel estimation
    [hest,nest] = lteDLChannelEstimate(enb,cec,rxSubframe);

    % If the current subframe contains the first occurrence of the MIB in
    % the waveform, decode the MIB to establish the frame number
    if (enb.NSubframe==0 && i<10)
        startFrame = decodeMIB(enb,rxSubframe,hest,nest,i);
    end

    % Get PCFICH indices, extract received PCFICH symbols and corresponding
    % channel estimate, demodulate PCFICH, decode and record CFI
    pcfichIndices = ltePCFICHIndices(enb);
    [pcfichRx,pcfichHest] = lteExtractResources(pcfichIndices,rxSubframe,hest);
    pcfichBits = ltePCFICHDecode(enb,pcfichRx,pcfichHest,nest);
    enb.CFI = lteCFIDecode(pcfichBits);
    CFIs(i+1) = enb.CFI;

    % Get PDCCH indices
    pdcchIndices = ltePDCCHIndices(enb);
```

```
% Extract received PDCCH symbols and corresponding channel estimate
[pdcchRx,pdcchHest] = lteExtractResources(pdcchIndices,rxSubframe,hest);

% Perform PDCCH demodulation to obtain the received PDCCH bits, which
% may contain encoded DCI messages for one or more users
[pdcchBits,pdcchSymbols] = ltePDCCHDecode(enb,pdcchRx,pdcchHest,nest);

% Initialize array of PDCCH format / candidate / DCI format
% combinations, which represent possible UEs (i.e. possible locations
% for DCI messages providing a downlink or uplink grant for a UE)
possibleUEs = [];

% PDCCH format for UE-specific search space can be 0, 1, 2, or 3
for pdcchFormat = 0:3

    ue = struct();

    % Get all PDCCH candidate indices and the number of candidates M
    % for the current PDCCH format
    ue.PDCCHFormat = pdcchFormat;
    candidates = getPDCCHCandidates(enb,ue);
    M = size(candidates,1);

    % For each candidate
    for m = 1:M

        % Record PDCCH candidate indices
        ue.Candidate = candidates(m,:);

        % For each DCI format in the UE-specific search space
        dciFormats = fieldnames(dciInfoUESpecific);
        for dciFormatIdx = 1:length(dciFormats)

            % Record DCI format
            ue.DCIFormat = dciFormats{dciFormatIdx};

            % Record this PDCCH format / candidate / DCI format
            % combination as a possible "UE". For DCI formats that have
            % different lengths across the UE-specific higher-layer
            % parameters, each length is recorded as a different
            % possible UE
            dciLengths = dciInfoUESpecific.(ue.DCIFormat);
            for dciLengthIdx = 1:length(dciLengths)
                ue.DCILength = dciLengths(dciLengthIdx);
                ue.PDCCH = pdcchUESpecific.(ue.DCIFormat)(dciLengthIdx);
                possibleUEs = [possibleUEs ue]; %#ok<AGROW>
            end

        end

        % For PDCCH formats 2 or 3, record the possibility of a DCI
        % message in the common search space in this candidate location
        % with format 1A or 1C. Such DCI messages correspond to System
        % Information, Paging, or Random Access Response message rather
        % than uplink or downlink grants for a UE, but are searched for
        % in this example for completeness
        if (any(pdcchFormat==[2 3]))
            for dciFormat = {'Format1A' 'Format1C'}
```

```
                    ue.DCIFormat = dciFormat{1};
                    ue.DCILength = dciInfoCommon.(ue.DCIFormat);
                    ue.PDCCH = pdcchCommon.(ue.DCIFormat);
                    possibleUEs = [possibleUEs ue]; %#ok<AGROW>
                end
            end

        end

    end

    % For each possible UE
    for ue = possibleUEs

        % Get the received candidate bits from the received PDCCH bits (for
        % DCI decoding)
        bitIdx = ue.Candidate;
        candidateBits = pdcchBits(bitIdx(1):bitIdx(2));

        % Get the received candidate symbols from the received PDCCH
        % symbols (for PDCCH power estimation). The modulation order Qm = 2
        % is used to convert candidate bit indices into candidate symbol
        % indices (QPSK modulation is always used for PDCCH)
        Qm = 2;
        symIdx = [((bitIdx(1) - 1) / Qm) + 1 bitIdx(2)/Qm];
        candidateSymbols = pdcchSymbols(symIdx(1):symIdx(2));

        % Decode the received candidate bits and get the RNTI. Assuming a
        % CRC pass (CRC = 0), the 2nd output from lteDCIDecode will
        % correspond to the RNTI
        [dciMessageBits,RNTI] = lteDCIDecode(ue.DCILength,candidateBits);

        % Re-encode the decoded DCI message bits (using the detected RNTI)
        % and establish the number of bit errors between the re-encoded
        % candidate bits and received candidate bits
        ue.RNTI = RNTI;
        reencodedCandidateBits = lteDCIEncode(ue,dciMessageBits);
        numErrors = sum(int8(candidateBits > 0) ~= reencodedCandidateBits);

        % If the number of bit errors is greater than maxErrors, continue
        % to the next possible UE (the rest of the loop body here is not
        % executed)
        if (numErrors > maxErrors)
            continue;
        end

        % Create DCI message structure from message bit payload. This is
        % performed because for some DCI formats (e.g. format 0 versus
        % format 1A), the true DCI format amongst DCI formats with
        % the same length can be determined by the value of a message bit
        dci = lteDCI(enb,ue.PDCCH,dciMessageBits);

        % Create table entry for the current DCI message
        result = struct();
        result.Subframe = i;
        result.DetectedRNTI = string(dec2hex(ue.RNTI,4));
        result.NumErrors = numErrors;
        result.DCIFormat = string(dci.DCIFormat);
```

```
            result.PDCCHFormat = ue.PDCCHFormat;
            result.Candidate = ue.Candidate;
            result.PDCCHPower = round(10*log10(var(candidateSymbols)),2);
            result.DCI = {dci};
            result = struct2table(result);

            % Check if a DCI message has previously been successfully decoded
            % in this subframe with the same RNTI, DCI format and starting
            % candidate location as the current DCI message (but with a
            % different PDCCH format)
            if (~isempty(results))
                match = (results.Subframe == result.Subframe);
                match = match & (results.DetectedRNTI == result.DetectedRNTI);
                match = match & strcmpi(results.DCIFormat,result.DCIFormat);
                match = match & (results.Candidate(:,1) == result.Candidate(1));
                match = find(match~=0);
            else
                match = [];
            end

            % If a DCI message satisfies the criteria above, it must have been
            % for a smaller PDCCH format and therefore represents successful
            % decoding of part of the current message, i.e. decoding a subset
            % of the candidate Control Channel Elements (CCEs). Therefore the
            % previous result can be replaced with the current result. Note
            % that for larger PDCCH formats the number of bit errors will tend
            % to be bigger as there are more candidate bits, so at low SNRs the
            % detected PDCCH format here may be lower than the true PDCCH
            % format (i.e. the number of errors in the bits for the true PDCCH
            % format exceeds maxErrors)
            if (isempty(match))
                results = [results; result]; %#ok<AGROW>
            else
                results(match,:) = result;
            end

        end

        % Update the subframe number
        enb.NSubframe = mod(enb.NSubframe + 1,10);

    end

    % Remove results whose estimated PDCCH power (in dB) is lower than the
    % minimum power threshold. Without this step, received candidate symbols
    % consisting solely of noise (even at a low power) can trigger false
    % detections
    if (~isempty(results))
        results(results.PDCCHPower < minPDCCHPower,:) = [];
    end

    % Check if results are empty
    if (isempty(results))
        disp('No RNTIs found.');
        return;
    end

    % In any given subframe, in the case of overlapping results (i.e. where the
```

```
% candidate indices of one result are a subset of another), the result with
% the lowest number of bit errors will be selected
overlapping = [];
for i = unique(results.Subframe).'

    % Find result rows which correspond to the current subframe and extract
    % those results
    sf = find(results.Subframe == i).';

    % For each pair of results in this subframe
    for a = sf
        for b = sf

            ca = results{a,'Candidate'};
            cb = results{b,'Candidate'};

            % If the first result's candidate indices are a subset of the
            % second result's candidate indices
            if (ca(1)>=cb(1) && ca(2)<=cb(2))

                % If the first result has more errors
                if (results{a,'NumErrors'}>results{b,'NumErrors'})

                    % Mark the first result in the pair as overlapping
                    % (i.e. mark it for removal)
                    overlapping = [overlapping a]; %#ok<AGROW>

                end

            end

        end
    end

end

% Remove overlapping results
results(overlapping,:) = [];

% Add a label for each row of the results table
results.Properties.RowNames = strsplit(num2str(1:height(results)));
```

Display Results Table

The table of detected RNTIs and other information associated with the detected DCI messages is displayed. Each row of this table includes:

- `Subframe`: The subframe number (relative to the start of the received waveform) in which the detected RNTI occurs.
- `DetectedRNTI`: The detected RNTI.
- `NumErrors`: The number of bit errors between the re-encoded candidate bits and received candidate bits (less than or equal to `maxErrors`).
- `DCIFormat`: The DCI format of the DCI message. Note that during the UE detection process, only DCI formats with unique lengths were used, as only the length of the output DCI message influences the decoding process (specifically, the rate matching prior to convolutional decoding). Within the LTE system, multiple DCI messages can have the same length for the same eNodeB

configuration. This ambiguity is resolved in normal system operation by each UE being assigned a Transmission Mode (TM). TS 36.213 Table 7.1-5 [3] shows which DCI formats are applicable for a given TM, and all formats for a given TM will have unique lengths. Because this example operates without knowledge of the UEs and their TMs, the true DCI format detected for the UE may not match the DCI format here, but the length will match. The function `lteDCIInfo` can be used to establish which other DCI formats have the same length as the DCI format here.

- `PDCCHFormat`: The PDCCH format of the received candidate bits for the DCI message. For more information about PDCCH formats, see the “PDCCH Blind Search and DCI Decoding” on page 2-169 example.
- `Candidate`: The inclusive [begin,end] bit indices (1-based) of the PDCCH candidate location for the DCI message i.e. the location of the received candidate bits in the set of overall received PDCCH bits. The overall set of PDCCH search space candidates for a given eNodeB configuration, PDCCH format and RNTI can be found using the `ltePDCCHSpace` function.
- `PDCCHPower`: The power of the received candidate symbols (in dB) associated with this DCI message.
- `DCI`: The decoded DCI message structure, assuming that the DCI format noted above is the true DCI format for the message.

```
disp(results);
```

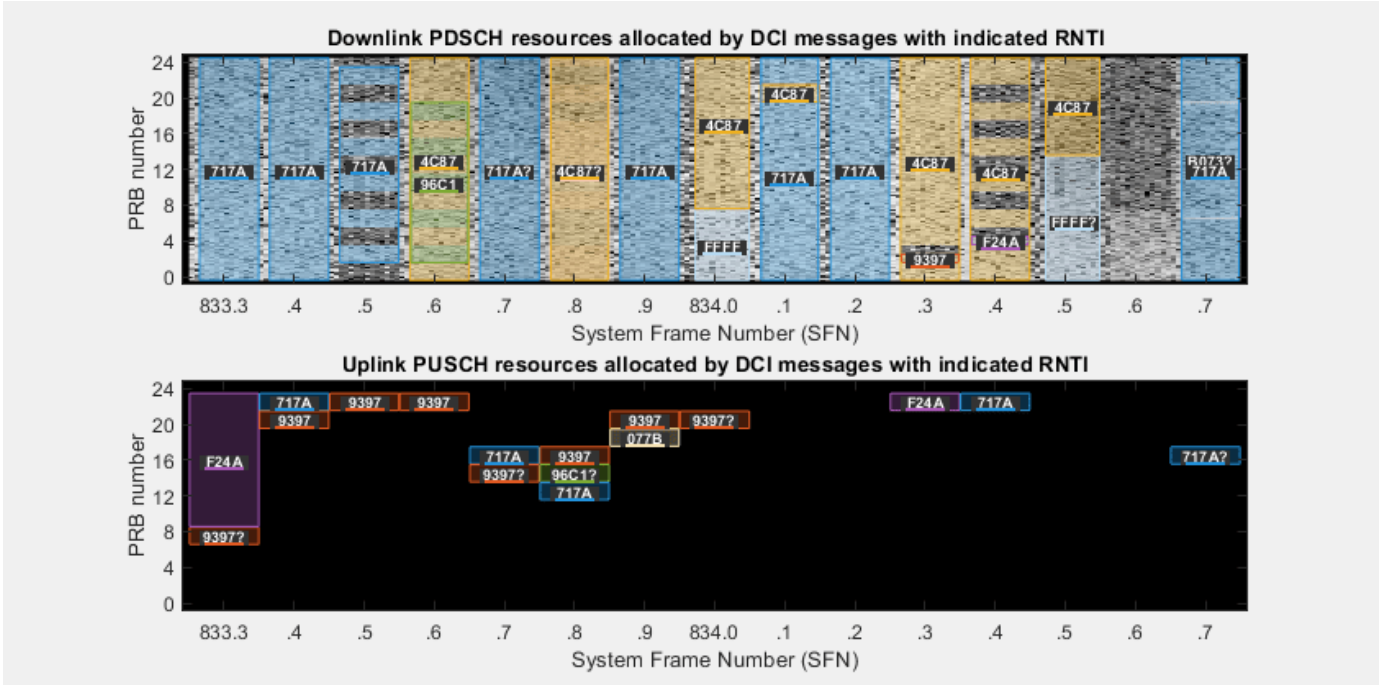
	Subframe	DetectedRNTI	NumErrors	DCIFormat	PDCCHFormat	Candidate	PD
1	0	"9397"	1	"Format0"	1	1 144	
2	0	"F24A"	0	"Format0"	2	289 576	
3	0	"717A"	0	"Format2A"	2	577 864	
4	1	"717A"	0	"Format2A"	2	1 288	
5	1	"717A"	0	"Format0"	2	289 576	
6	1	"9397"	0	"Format0"	1	577 720	
7	2	"717A"	0	"Format2A"	2	1 288	
8	2	"9397"	0	"Format0"	1	289 432	
9	3	"4C87"	0	"Format2A"	2	1 288	
10	3	"9397"	0	"Format0"	1	289 432	
11	3	"96C1"	0	"Format2A"	1	577 720	
12	4	"717A"	0	"Format0"	2	1 288	
13	4	"9397"	2	"Format0"	0	433 504	
14	4	"717A"	1	"Format2A"	1	577 720	
15	5	"9397"	0	"Format0"	0	1 72	
16	5	"96C1"	1	"Format0"	0	145 216	
17	5	"6C27"	2	"Format2C"	0	217 288	
18	5	"717A"	0	"Format0"	1	289 432	
19	5	"4C87"	2	"Format2A"	0	577 648	
20	6	"717A"	0	"Format2A"	2	1 288	
21	6	"9397"	0	"Format0"	1	289 432	
22	6	"077B"	0	"Format0"	1	433 576	
23	7	"FFFF"	0	"Format1A"	2	1 288	
24	7	"9397"	1	"Format0"	1	289 432	
25	7	"4C87"	0	"Format2A"	1	577 720	
26	8	"4C87"	0	"Format2A"	2	1 288	
27	8	"717A"	0	"Format2A"	2	577 864	
28	9	"717A"	0	"Format2A"	2	577 864	
29	10	"4C87"	0	"Format2A"	2	1 288	
30	10	"F24A"	0	"Format0"	2	289 576	
31	10	"9397"	0	"Format2A"	1	721 864	
32	11	"4C87"	0	"Format2A"	2	1 288	

```
    33        11        "717A"        0        "Format0"        2        289      576
    34        11        "F24A"        0        "Format2A"       2        577      864
    35        12        "FFFF"        2        "Format1A"       1          1      144
    36        12        "4C87"        0        "Format2A"       2        865     1152
    37        14        "717A"        2        "Format0"        1          1      144
    38        14        "B073"        2        "Format2"        0        145      216
    39        14        "717A"        0        "Format2A"       2        577      864
```

Plot Detected RNTIs

A plot is produced showing the detected RNTIs versus subframe number. The received downlink resource grid is displayed and the RNTIs are used to label the PDSCH resources signaled by the corresponding DCI messages. An empty uplink resource grid is also displayed and the PUSCH resources signaled by any DCI format 0 or format 4 messages are displayed and are labeled with the corresponding RNTIs. A question mark appears after any RNTI for which the number of bit errors between the re-encoded candidate bits and received candidate bits, `NumErrors`, is greater than zero (and less than or equal to `maxErrors`). The System Frame Numbers (SFNs) used here use integers to represent the SFN and decimals (tenths) to represent the subframe within the SFN.

```
enb.NFrame = startFrame;
enb.NSubframe = startSubframe;
enb.CFI = CFIs;
summary = hPlotDetectedRNTIs(results,enb,rxgrid);
```



Display Summary of Detected PDSCH or PUSCH Resource Allocations

Finally, a summary of the detected PDSCH or PUSCH resource allocations is displayed. For each detected DCI message the following is displayed:

- System Frame Number (SFN)

- RNTI
- Number of errors
- DCI format (and associated link direction)
- Allocated PRBs
- An estimate of the received power in the allocated PRBs

Note that because no uplink resource grid is present, the power estimated for uplink DCI messages will be -Inf. Note that if a summary per RNTI is required, the summary table can easily be sorted to group results for the same RNTI using `sortrows(summary,'RNTI')`.

```
disp(summary);
```

SFN	RNTI	NumErrors	DCIFormat	LinkDirection	PRBSet
833.3	"9397"	1	"Format0"	"Uplink"	"[7 8]"
833.3	"F24A"	0	"Format0"	"Uplink"	"[9...23]"
833.3	"717A"	0	"Format2A"	"Downlink"	"[0...24]"
833.4	"717A"	0	"Format2A"	"Downlink"	"[0...24]"
833.4	"717A"	0	"Format0"	"Uplink"	"[22 23]"
833.4	"9397"	0	"Format0"	"Uplink"	"[20 21]"
833.5	"717A"	0	"Format2A"	"Downlink"	"[2 3 6 7 10 11 14 15 18 19 22
833.5	"9397"	0	"Format0"	"Uplink"	"[22 23]"
833.6	"4C87"	0	"Format2A"	"Downlink"	"[0 1 4 5 8 9 12 13 16 17 20.
833.6	"9397"	0	"Format0"	"Uplink"	"[22 23]"
833.6	"96C1"	0	"Format2A"	"Downlink"	"[2 3 6 7 10 11 14 15 18 19]"
833.7	"717A"	0	"Format0"	"Uplink"	"[16 17]"
833.7	"9397"	2	"Format0"	"Uplink"	"[14 15]"
833.7	"717A"	1	"Format2A"	"Downlink"	"[0...24]"
833.8	"9397"	0	"Format0"	"Uplink"	"[16 17]"
833.8	"96C1"	1	"Format0"	"Uplink"	"[14 15]"
833.8	"6C27"	2	"Format2C"	"Downlink"	"[0...3 8 9 12...17 20 21 24]"
833.8	"717A"	0	"Format0"	"Uplink"	"[12 13]"
833.8	"4C87"	2	"Format2A"	"Downlink"	"[0...24]"
833.9	"717A"	0	"Format2A"	"Downlink"	"[0...24]"
833.9	"9397"	0	"Format0"	"Uplink"	"[20 21]"
833.9	"077B"	0	"Format0"	"Uplink"	"[18 19]"
834	"FFFF"	0	"Format1A"	"Downlink"	"[0...7]"
834	"9397"	1	"Format0"	"Uplink"	"[20 21]"
834	"4C87"	0	"Format2A"	"Downlink"	"[8...24]"
834.1	"4C87"	0	"Format2A"	"Downlink"	"[20 21]"
834.1	"717A"	0	"Format2A"	"Downlink"	"[0...19 22...24]"
834.2	"717A"	0	"Format2A"	"Downlink"	"[0...24]"
834.3	"4C87"	0	"Format2A"	"Downlink"	"[0 1 4...24]"
834.3	"F24A"	0	"Format0"	"Uplink"	"[22 23]"
834.3	"9397"	0	"Format2A"	"Downlink"	"[2]"
834.4	"4C87"	0	"Format2A"	"Downlink"	"[0...3 6 7 10 11 14 15 18 19
834.4	"717A"	0	"Format0"	"Uplink"	"[22 23]"
834.4	"F24A"	0	"Format2A"	"Downlink"	"[4]"
834.5	"FFFF"	2	"Format1A"	"Downlink"	"[0...13]"
834.5	"4C87"	0	"Format2A"	"Downlink"	"[14...24]"
834.7	"717A"	2	"Format0"	"Uplink"	"[16 17]"
834.7	"B073"	2	"Format2"	"Downlink"	"[7 11 15 19]"
834.7	"717A"	0	"Format2A"	"Downlink"	"[0...24]"

Observations on Detected RNTIs

The following observations can be made using the results summary and plot:

- *System Information (SI):* The RNTI used for System Information (SI-RNTI) is FFFF hex. DCI messages associated with System Information Blocks (SIBs) can be seen in SFN = 834.0 (NFrame = 834, NSubframe = 0) and SFN = 834.5 (NFrame = 834, NSubframe = 5). SystemInformationBlockType1 (SIB1) occurs in subframe 5 of even frames, so the occurrence in SFN = 834.5 is SIB1. Other SIBs are dynamically scheduled, with the scheduling information being contained in SIB1.
- *RNTIs occurring in multiple subframes:* The RNTIs 717A and 4C87 occur frequently throughout the waveform, typically with zero errors, so they are likely to correspond to active UEs.

The detection process can result in errors due to noise and distortion in the downlink signals. There are two main types of errors, these are described below using examples:

- *Missed detection:* Observe SFN 834.6. There is energy in PDSCH resource blocks 0 to 7. However, no DCI message has been detected. Rerun the simulation setting `maxErrors=3`. Now RNTI 4C87 is detected in this region. This RNTI is present in other subframes, which means that this is likely to be a genuine UE.
- *False positive:* Observe the PDSCH allocation with RNTI B073 in the final subframe of the waveform. It overlaps with PDSCH allocation 717A. Given that RNTI 717A occurs frequently throughout the waveform and that RNTI B073 decodes with 2 errors, it is likely that RNTI B073 is a "false positive" detection.

Appendix

This example uses the helper function:

- hPlotDetectedRNTIs.m

Selected Bibliography

1. 3GPP TS 36.321 "Medium Access Control (MAC) protocol specification"
2. Kumar, Hamed, Katabi and Li. "LTE Radio Analytics Made Easy and Accessible", SIGCOMM '14 (2014): 211-222
3. 3GPP TS 36.213 "Physical layer procedures"

Local Functions

This example uses the following local functions:

- `getUniqueDCILengths`: Get DCI message information for DCI formats with unique lengths
- `decodeMIB`: Decode the MIB to get frame number
- `getPDCCHCandidates`: Get all PDCCH candidates

```
% Get DCI message information for DCI formats with unique lengths, across
% all UE-specific higher-layer parameters
function [infoOut,pdcchOut] = getUniqueDCILengths(enb,pdcchIn)

    infoOut = [];

    for EnableSRSRequest = {'Off' 'On'}
```

```
pdcchIn.EnableSRSRequest = EnableSRSRequest{1};

for EnableMultipleCSIRequest = {'Off' 'On'}

    pdcchIn.EnableMultipleCSIRequest = EnableMultipleCSIRequest{1};

    for NTxAnts = [1 2 4]

        pdcchIn.NTxAnts = NTxAnts;

        % Get the DCI message lengths for all formats, as a
        % structure
        info = lteDCIInfo(enb,pdcchIn);

        % Convert the structure into a cell array of field names
        % and values
        formats = fieldnames(info);
        sizes = struct2cell(info);

        % Remove DCI format 3 and 3A as these DCI formats are used
        % for conveying power control commands rather than resource
        % allocations
        dci3idx = ismember(formats,{'Format3','Format3A'});
        formats(dci3idx) = [];
        sizes(dci3idx) = [];

        % Keep only DCI format 1A and 1C for the common search
        % space
        if (strcmpi(pdcchIn.SearchSpace,'Common'))
            commonidx = ismember(formats,{'Format1A','Format1C'});
            formats = formats(commonidx);
            sizes = sizes(commonidx);
        end

        % Find the indices of the unique sizes, the first
        % occurrence of each unique size will be retained in the
        % original order
        [~,idxs] = unique(cat(2,sizes{:}),'stable');

        % If the current UE-specific settings yield unique sizes
        % for any format (compared to all other formats and
        % UE-specific settings thus far), then record them
        if (isempty(infoOut))
            infoOut = cell2struct(sizes(idxs),formats(idxs));
            pdcchOut = cell2struct(repmat({pdcchIn},size(idxs)),formats(idxs));
        else
            sizes_out = struct2cell(infoOut);
            sizes_out = cat(2,sizes_out{:});
            for i = idxs.'
                if (~any(sizes_out==sizes{i}))
                    format = formats{i};
                    infoOut.(format) = [infoOut.(format) sizes{i}];
                    pdcchOut.(format) = [pdcchOut.(format) pdcchIn];
                end
            end
        end

    end
```

```
            end

        end

    end

    % Decode the MIB to get the frame number
    function startFrame = decodeMIB(enb,rxSubframe,hest,nest,i)

        pbchIndices = ltePBCHIndices(enb);
        [pbchRx, pbchHest] = lteExtractResources(pbchIndices,rxSubframe,hest);
        [~,~,nfmod4,mibbits] = ltePBCHDecode(enb,pbchRx,pbchHest,nest);
        mib = lteMIB(mibbits);
        startFrame = double(mib.NFrame) + nfmod4;
        if (i>0)
            startFrame = mod(startFrame - 1,1024);
        end

    end

    % Get all PDCCH candidates for a given eNodeB configuration and UE
    % configuration, without having to evaluate all possible RNTIs
    function pdcchCandidates = getPDCCHCandidates(enb,ue)

        % PDCCH dimensionality information
        pdcchInfo = ltePDCCHInfo(enb);

        % Aggregation level
        L = 2 ^ ue.PDCCHFormat;

        % Number of candidates
        M = floor(double(pdcchInfo.NCCE)/L);

        % 1 CCE = 9 REGs = 36 REs = 72 bits
        bitsPerCCE = 72;

        % PDCCH candidate indices within PDCCH bits (1-based)
        pdcchCandidates = L*bitsPerCCE*(0:M-1).' + [1 L*bitsPerCCE];

    end
```

PUSCH HARQ-ACK Detection Conformance Test

This example shows how to measure the false detection probability and missed detection probability of Hybrid Automatic Repeat Request ACK (HARQ-ACK) multiplexed on the Physical Uplink Shared Channel (PUSCH) using the LTE Toolbox™ under conformance test conditions as defined in TS36.104, Section 8.2.4.1.

Introduction

Uplink Control Information (UCI) can be carried on the Physical Uplink Shared Channel (PUSCH) and may contain Hybrid Automatic Repeat Request ACK (HARQ-ACK) information. This example uses the LTE Toolbox to perform the "HARQ-ACK Multiplexed on PUSCH" conformance test specified in TS36.104, Section 8.2.4.1 [1].

Two performance requirements are defined for HARQ-ACK multiplexed on the PUSCH:

- *ACK false detection probability* is the probability that ACK is detected when data is only sent on symbols where HARQ-ACK information can be allocated.
- *The ACK missed detection probability* is the conditional probability of not detecting an ACK when it was sent on PUSCH resources.

The test scenario implemented in this example specifies one layer, one transmit antenna, two receive antennas, normal cyclic prefix, ETU70 channel, using Fixed Reference Channel (FRC) A4-3. The target defined in TS 36.104, Section 8.2.4.1 [1] for 1.4 MHz bandwidth (6RB) is a false detection probability and missed detection probability of 1% at 13.8 dB.

The test is run on a subframe by subframe basis at each SNR test point. For each subframe an uplink waveform is generated with and without ACK information, passed through a fading channel and the HARQ-ACK decoded. False detections and missed detections are recorded and the probability of error displayed for the range of SNRs tested.

Simulation Configuration

The example is executed for a simulation length of 1 frame at a number of SNRs including the required 13.8 dB at which the test requirements for false and missed detection rates (1% in each case) are to be achieved as per TS36.104, Table 8.2.4.1-1 [1]. A large number of `numSubframes` should be used to produce meaningful throughput results. `SNRdB` can be an array of values or a scalar.

```
numSubframes = 10;  % Number of frames to simulate at each SNR
SNRdB = [4.8 7.8 10.8 13.8 16.8]; % SNR points to simulate
```

UE Configuration

To configure the transmitter, a few desired parameter fields are set in the structure `frc`, which is then passed to `lteRMCUL` which will set all the other required parameter fields. At this point, the structure `frc` represents the complete configuration of the required transmission.

```
frc.TotSubframes = 1;   % Total number of subframes to generate
frc.NCellID = 10;       % Cell identity
frc.RC = 'A4-3';        % FRC number

% Populate FRC configuration structure with default values for A4-3
frc = lteRMCUL(frc);
```

Propagation Channel Model Configuration

The propagation channel is configured via the structure `chcfg`, with settings per the test requirements. Note that the sampling rate of the propagation is determined from the sampling rate of the transmitted waveform, which can be established using `lteSCFDMAInfo`.

```
chcfg.NRxAnts = 2;                % Number of receive antennas
chcfg.DelayProfile = 'ETU';       % Delay profile
chcfg.DopplerFreq = 70;           % Doppler frequency
chcfg.MIMOCorrelation = 'Low';    % MIMO correlation
chcfg.Seed = 91;                  % Channel seed
chcfg.NTerms = 16;                % Oscillators used in fading model
chcfg.ModelType = 'GMEDS';        % Rayleigh fading model type
chcfg.InitPhase = 'Random';       % Random initial phases
chcfg.NormalizePathGains = 'On'; % Normalize delay profile power
chcfg.NormalizeTxAnts = 'On';     % Normalize for transmit antennas

% Set channel model sampling rate
info = lteSCFDMAInfo(frc);
chcfg.SamplingRate = info.SamplingRate;
```

Channel Estimator Configuration

The channel estimator is configured using the structure `cec`. An ETU delay profile causes the channel to change quickly over time. Therefore a small pilot averaging frequency window of 9 Resource Elements (REs) is used. The Demodulation Reference Signal (DRS) is contained in only one symbol per slot, therefore a time averaging window of 1 RE is used. This will not include any pilots from an adjacent slot when averaging.

```
cec.PilotAverage = 'UserDefined'; % Type of pilot averaging
cec.FreqWindow = 9;               % Frequency averaging windows in REs
cec.TimeWindow = 1;               % Time averaging windows in REs
cec.InterpType = 'cubic';         % Interpolation type
cec.Reference = 'Antennas';       % Reference for channel estimation
```

Loop for SNR Values

A loop is used to run the simulation for a set of SNR points, given by the vector `SNRdB`.

The noise added before SC-FDMA demodulation will be amplified by the IFFT. The amplification is the square root of the size of the IFFT (N_{FFT}). To ensure that the power of the additive noise is normalized after demodulation to achieve the desired SNR, the desired noise power is divided by N_{FFT}. In addition, as real and imaginary parts of the noise are created separately before being combined into complex Additive White Gaussian Noise (AWGN), the noise amplitude must be scaled by $1/\sqrt{2}$ so the generated noise power is 1.

At each SNR test point the probability of false and missed detection is calculated. These probabilities are calculated on a subframe by subframe basis using the following steps:

- *Create Transmit Waveform:* The Uplink Reference Measurement Channel (RMC) Tool `lteRMCULTool` is used to generate an uplink waveform containing random transport data and a HARQ-ACK bit in every odd subframe.
- *Noisy Channel Modeling:* The waveform is passed through a fading channel and AWGN added.

- *Perform Synchronization and SC-FDMA Demodulation:* The received symbols are synchronized to account for a combination of implementation delay and channel delay spread. The symbols are then SC-FDMA demodulated.
- *Perform Channel and Noise Power Spectral Density Estimation and Equalization:* The channel and noise power spectral density are estimated and the received PUSCH symbols equalized.
- *Decode the PUSCH:* The UpLink Shared Channel (UL-SCH) coding is determined and used to decode the PUSCH to recover the interleaved UL-SCH.
- *Recover UCI ACK Bit:* Deinterleave the PUSCH and decode the HARQ-ACK information to recover the UCI HARQ-ACK bit and detect HARQ-ACK false or missed detection based on whether an HARQ-ACK bit was encoded in the transmitted subframe or not.

```
% Initialize variables used in the simulation and analysis
pFalse = zeros(size(SNRdB));  % Probability of false detection at each SNR
pMissed = zeros(size(SNRdB)); % Probability of missed detection at each SNR

for nSNR = 1:length(SNRdB)

    % Initialize the random number generator stream
    rng('default');

    % Extract SNR to test
    SNR = 10^(SNRdB(nSNR)/20);

    % Scale noise to ensure the desired SNR after SC-FDMA demodulation
    N = 1/(SNR*sqrt(double(info.Nfft)))/sqrt(2.0);

    offsetUsed = 0;
    falseCount = 0;
    falseTests = 0;
    missCount = 0;
    missTests = 0;

    for subframeNo = 0:(numSubframes-1)

        % Updating subframe number
        frc.NSubframe = mod(subframeNo, 10);

        % Transmit ACK on every odd subframe
        if (mod(subframeNo, 2)==0)
            ACK = [];
            falseTests = falseTests + 1;
        else
            ACK = 1;
            missTests = missTests + 1;
        end

        % Create random data to transmit
        trblklen = frc.PUSCH.TrBlkSizes(frc.NSubframe+1);
        trblk = randi([0 1], trblklen, 1);

        % Transmit a waveform with an additional 25 samples to cover the
        % range of delays expected from the channel modeling (a
        % combination of implementation delay and channel delay spread)
        txWaveform = [lteRMCULTool(frc, trblk, [], [], ACK); zeros(25, 1)];
```

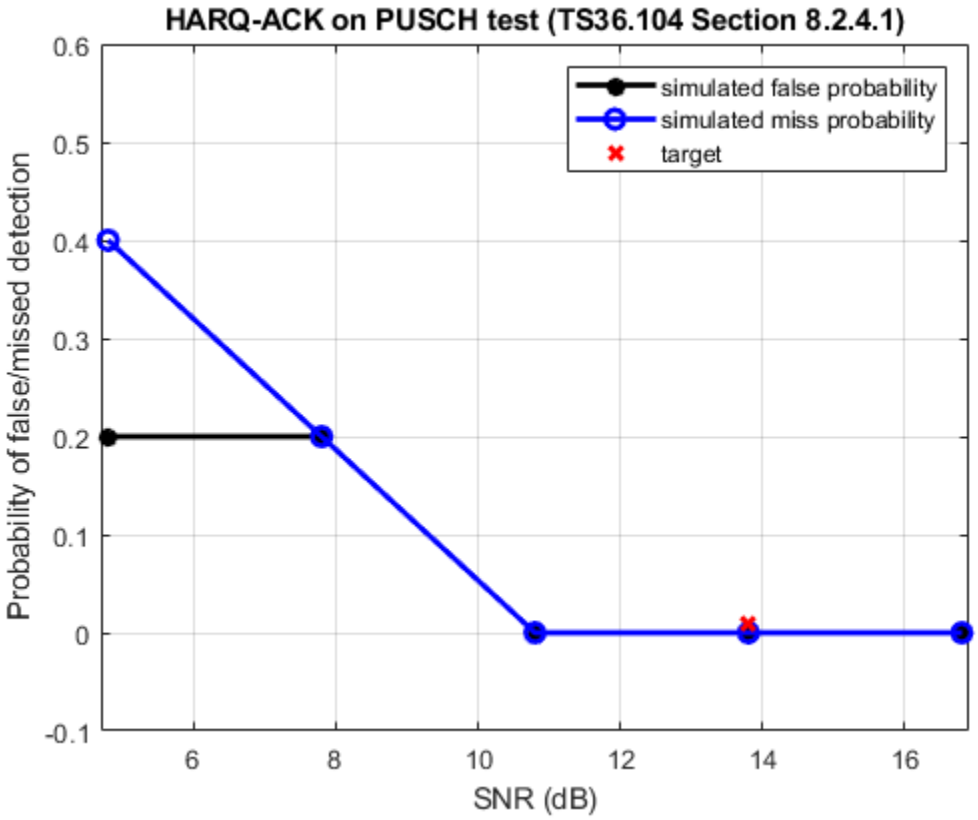
```
        % Pass waveform through fading channel model
        chcfg.InitTime = subframeNo/1000;
        rxWaveform = lteFadingChannel(chcfg, txWaveform);

        % Add noise
        noise = N*complex(randn(size(rxWaveform)), ...
            randn(size(rxWaveform)));
        rxWaveform = rxWaveform + noise;

        % Synchronization
        offset = lteULFrameOffset(frc, frc.PUSCH,rxWaveform);
        if (offset < 25)
            offsetUsed = offset;
        end

        % SC-FDMA demodulation
        rxSubframe = lteSCFDMADemodulate(frc, ...
            rxWaveform(1+offsetUsed:end, :));

        % Channel Estimation
        [estChannelGrid, noiseEst] = lteULChannelEstimate( ...
            frc, frc.PUSCH, cec, rxSubframe);

        % PUSCH indices for given subframe
        puschIndices = ltePUSCHIndices(frc,frc.PUSCH);

        % Minimum Mean Squared Error (MMSE) equalization
        rxSymbols = lteEqualizeMMSE(rxSubframe(puschIndices), ...
                        estChannelGrid(puschIndices), noiseEst);

        % Obtain UL-SCH coding information for current transport block and
        % ACK, and concatenate this information with the PUSCH / UL-SCH
        % configuration
        frc.PUSCH = lteULSCHInfo(frc, frc.PUSCH, trblklen, ...
            0, 0, 1, 'chsconcat');

        % Perform deprecoding, demodulation and descrambling on the
        % received data
        rxEncodedBits = ltePUSCHDecode(frc, frc.PUSCH, rxSymbols);

        % UL-SCH channel deinterleaving
        [deinterleavedData, ccqi, cri, cack] = ...
            lteULSCHDeinterleave(frc, frc.PUSCH, rxEncodedBits);

        % HARQ-ACK decoding
        rxACK = lteACKDecode(frc.PUSCH, cack);

        % Detect false or missed HARQ-ACK
        if (isempty(ACK) && ~isempty(rxACK))
            falseCount = falseCount + 1;
        end
        if (~isempty(ACK) && ~isequal(ACK,rxACK))
            missCount = missCount + 1;
        end

    end
```

```
    % Calculate false or missed HARQ-ACK probability
    pFalse(nSNR) = falseCount/falseTests;
    pMissed(nSNR) = missCount/missTests;

end
```

Analysis

Plot the simulated results against the target performance as stipulated in the standard using hHARQACKResults.m.

```
hHARQACKResults(SNRdB, pFalse, pMissed);
```



Appendix

This example uses this helper function.

- hHARQACKResults.m

Selected Bibliography

1 3GPP TS 36.104 "Base Station (BS) radio transmission and reception"

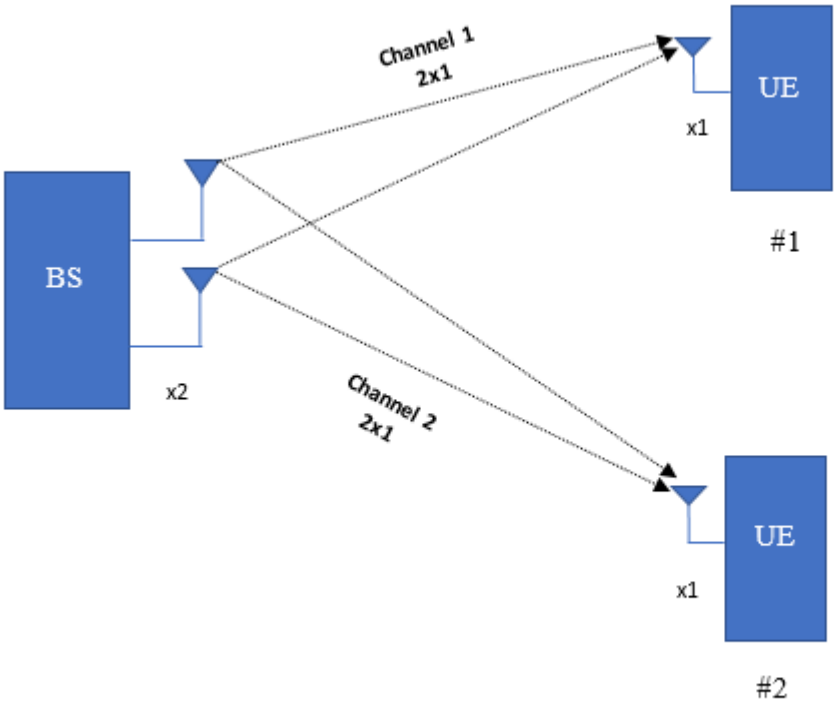
PDSCH Throughput for Non-Codebook Based MU-MIMO Transmission Mode 9 (TM9)

This example demonstrates how to measure the physical downlink shared channel (PDSCH) throughput performance in a multiuser multiple-input multiple-output (MU-MIMO) scenario with LTE Toolbox™. It models non-codebook based transmission mode, TM9, with block diagonalization [1]. This example supports both frequency division duplexing (FDD) and time division duplexing (TDD) schemes. It also supports the use of Parallel Computing Toolbox™ to reduce the effective simulation time.

Introduction

In a MU-MIMO scenario, due to the simultaneous transmission of data to multiple users, inter-user interference will be present at the receiver. Inter-user interference at the receiver can be canceled using precoding techniques at the transmitter. Two linear precoding techniques for MU-MIMO transmission are channel inversion and block diagonalization. This example uses block diagonalization precoding. This example measures the PDSCH throughput in a MU-MIMO scenario for a number of signal-to-noise ratio (SNR) points. For information on how to model single-user MIMO (SU-MIMO) in LTE look at the following example: “PDSCH Throughput for Non-Codebook Based Precoding Schemes: Port 5 (TM7), Port 7 or 8 or Port 7-8 (TM8), Port 7-14 (TM9 and TM10)” on page 2-424

A simple MU-MIMO block diagram with the default simulation configuration parameters used in the example is shown in the following figure.



Simulation Configuration

The simulation parameters for the base station and users are configured in this section. The example is executed for a simulation length of two frames for a number of SNR points. Increase `NFrames` to increase the simulation time and produce statistically significant throughput results. Use the variable `SNRIn` to set the SNR, it can be an array of values or a scalar. As per the constraints in LTE [2], this example supports a maximum of 4 users with a maximum of 4 layers across all users. The number of maximum layers per user is 2. The number of transmit antennas should be greater than or equal to the total number of receive antennas across all users.

```
NFrames = 2;              % Number of frames
SNRIn = [8 14];           % SNR range in dB
NUsers = 2;               % Number of active users
NTxAnts = 2;              % Number of antennas at eNodeB

% Specify UE-specific parameters
muNumLayers = [1 1 1 1];  % Number of layers for a maximum of 4 users
muNumRxAnts = [1 1 1 1];  % Number of receive antennas for a maximum of 4 users
muCodeRate = [0.5 0.5 0.5 0.5]; % Code rate for a maximum of 4 users
muModulation = {'16QAM';'16QAM';'16QAM';'16QAM'}; % Modulation for a maximum of 4 users
```

The set of parameters required for TM9 is specified below. This example does not perform DCI format decoding; the `DCIFormat` field is included for completeness. The cell array `muPDSCH` stores the PDSCH transmission configuration structure for all users.

```
% Initialize cell arrays of PDSCH transmission configuration structures,
% transport block sizes and coded transport block sizes.
```

```
muPDSCH = cell(NUsers,1);
trBlkSizes = cell(NUsers,1);
codedTrBlkSizes = cell(NUsers,1);

simulationParameters = []; % clear simulation parameters
simulationParameters.NDLRB = 50;
simulationParameters.PDSCH.PRBSet = (0)';
simulationParameters.PDSCH.DCIFormat = 'Format2C';
simulationParameters.PDSCH.TxScheme = 'Port7-14';
simulationParameters.PDSCH.NTxAnts = NTxAnts;
simulationParameters.DuplexMode = 'FDD'; % 'FDD', 'TDD'
simulationParameters.TotSubframes = 1;

% PDSCH configuration structure for users based on the common and
% user-specific parameters
ncw = zeros(NUsers,1);
for userIdx = 1:NUsers
    simulationParameters.PDSCH.TargetCodeRate = muCodeRate(userIdx);
    simulationParameters.PDSCH.Modulation = muModulation{userIdx};
    simulationParameters.PDSCH.NLayers = muNumLayers(userIdx);
    % Initialize W to zero
    simulationParameters.PDSCH.W = zeros(muNumLayers(userIdx),NTxAnts);
    % Downlink reference measurement channel configuration
    enb = lteRMCDL(simulationParameters);
    % PDSCH transmission configuration structure for users
    muPDSCH{userIdx}= enb.PDSCH;
    % Number of codewords for users
    ncw(userIdx) = length(muPDSCH{userIdx}.Modulation);
    % Store transport block sizes for users
    trBlkSizes{userIdx} = muPDSCH{userIdx}.TrBlkSizes;
end
% Assign redundancy version sequence
rvSequence = muPDSCH{1}.RVSeq;
```

Print a summary of some of the more relevant simulation parameters.

```
hMultiUserParameterSummary(enb,muPDSCH,muNumRxAnts);
```

```
 Parameter summary for TM9 MU-MIMO Transmission
-----------------------------------------------------------------
                     Duplexing mode: FDD
                  Transmission mode: TM9(MU-MIMO)
                Transmission scheme: Port7-14
  Number of downlink resource blocks: 50
 Number of allocated resource blocks: 1
         Number of transmit antennas: 2
-----------------------------------------------------------------
Number of Transmission layers for UE-1: 1
          Number of codewords for UE-1: 1
   Number of receive antennas for UE-1: 1
              Modulation codeword 1: 16QAM
    Transport block sizes codeword 1:      208     208     208     208     208     208     208
                Code rate codeword 1:   0.5088  0.5088  0.5088  0.5088  0.5088  0.5088  0.5088
-----------------------------------------------------------------
Number of Transmission layers for UE-2: 1
          Number of codewords for UE-2: 1
   Number of receive antennas for UE-2: 1
              Modulation codeword 1: 16QAM
```

```
    Transport block sizes codeword 1:      208     208     208     208     208     208     208
                Code rate codeword 1:   0.5088  0.5088  0.5088  0.5088  0.5088  0.5088  0.5088
```

Propagation Channel

The channel model configuration parameters for the channel between the eNodeB and the users are stored in the cell array `muChannel`. The set of common parameters for each channel is initially specified. The parameters defined here are used with `lteFadingChannel` during subframe processing.

```
muChannel  = cell(NUsers,1);
channel = struct;
channel.DelayProfile ='EPA';          % Delay profile
channel.MIMOCorrelation = 'Low';      % Multi-antenna correlation
channel.NTerms = 16;                  % Oscillators used in fading model
channel.ModelType = 'GMEDS';          % Rayleigh fading model type
channel.InitPhase = 'Random';         % Random initial phases
channel.NormalizePathGains = 'On';    % Normalize delay profile power
channel.NormalizeTxAnts = 'On';       % Normalize for transmit antennas
```

The channel sampling rate depends on the FFT size used in the OFDM modulator. This can be obtained using the function `lteOFDMInfo`.

```
ofdmInfo = lteOFDMInfo(enb);
channel.SamplingRate = ofdmInfo.SamplingRate;

% Independent channel configuration parameters for each user
chanSeeds = [1111 2222 3333 4444];  % Channel seed for a maximum of 4 users
dopplerFreq = [5 50 25 15];         % Doppler frequency for a maximum of 4 users
for userIdx = 1:NUsers
    muChannel{userIdx}= channel;
    muChannel{userIdx}.Seed = chanSeeds(userIdx);         % Channel seed
    muChannel{userIdx}.NRxAnts = muNumRxAnts(userIdx);    % Number of receive antennas
    muChannel{userIdx}.DopplerFreq = dopplerFreq(userIdx);% Doppler frequency
end
```

Processing Chain

To determine the throughput at each SNR point, the subframe-by-subframe PDSCH processing chain includes:

- *Calculating the Precoding Matrix* - A perfect channel estimate is used to calculate the precoding matrix for each user. A detailed explanation of this step is provided in the next section.
- *Updating Current HARQ Process* - Separate HARQ processes are used for each user.
- *Creating Transmit Waveform* - Separate PDSCH symbols are generated for each user. The PDSCH symbols are precoded with the calculated precoding matrix. The precoded PDSCH symbols corresponding to the UEs are combined and OFDM modulated.
- *Channel Modeling* - Pass the waveform through a fading channel to each user and add noise (AWGN)
- *Performing Synchronization and OFDM Demodulation* - Separately carried out for each user. Offset the received symbols to account for a combination of implementation delay and channel delay spread. OFDM demodulate the symbols.

- *Decoding the PDSCH* - Separately carried out for each user. Perfect channel estimate is assumed at the receiver for decoding operations. Obtain an estimate of the received codewords using `ltePDSCHDecode` to demodulate and descramble the recovered PDSCH symbols for all transmit and receive antenna pairs.
- *Decoding the DL-SCH and Storing the Block CRC* - Separately carried out for each user. Pass the vector of decoded soft bits to `lteDLSCHDecode`, which decodes the codeword and returns the block CRC error used to determine the throughput of the system.

Precoding Matrix Calculation

The precoding matrix for each user needs to be computed based on the channel estimates between the eNodeB and users. The precoding matrix is calculated using the block diagonalization method. For the computation of the precoding matrix, channel state information (CSI) is required at the transmitter. In this example, for the sake of simplicity, perfect knowledge of the channel between the eNodeB and the users is assumed at the base station.

For TDD, the channel estimates between the eNodeB and the users are estimated in the last UL subframe before a DL subframe. These channel estimates are used to calculate the precoding matrix, `W`. All subsequent DL subframes (including special subframes) until the next UL subframe are precoded with matrix `W`.

For FDD, there is a delay of one subframe between the calculation of `W` and the subframe where it is used. For example, `W` used in DL subframe `n` has been calculated with the channel estimates obtained in DL subframe `n-1`.

The function hMultiUserPrecodingMatrix calculates `W` using the following steps:

- Obtain a perfect channel estimate for the considered subframe for all users
- Average the channel estimates for all the allocated RBs
- Compute the precoding matrix using the block diagonalization method as per the `blkdiagbfweights` function.

Note that for an allocation of a single resource block, the precoding matrix will usually be well matched to the channel conditions, with little deviation from the optimal precoding. But as the allocation size increases, the precoding matrix takes into account the average of the channel conditions over the whole allocation. This averaging causes a deviation from the optimal precoding matrix. Therefore, you can expect a degradation in performance as the size of the resource allocation increases.

Processing Loop

The 'for' loop for SNR points processing is included below. To enable the use of parallel computing for increased speed use 'parfor' instead of 'for' in the loop. This requires the Parallel Computing Toolbox. If this is not installed 'parfor' will default to the normal 'for' statement.

```
% Initialize variables used in the simulation and analysis
maxThroughput = zeros(length(SNRIn),NUsers);
simThroughput = zeros(length(SNRIn),NUsers);
harqProcesses = cell(NUsers,1);
% Initialize cell array for constellation plot
rxConstellation = cell(numel(SNRIn),NUsers,2);
% Copy the channel cell array and cell array of PDSCH transmission
% configuration structure to optimize parallel processing (only if running
% the example with Parallel Computing Toolbox)
```

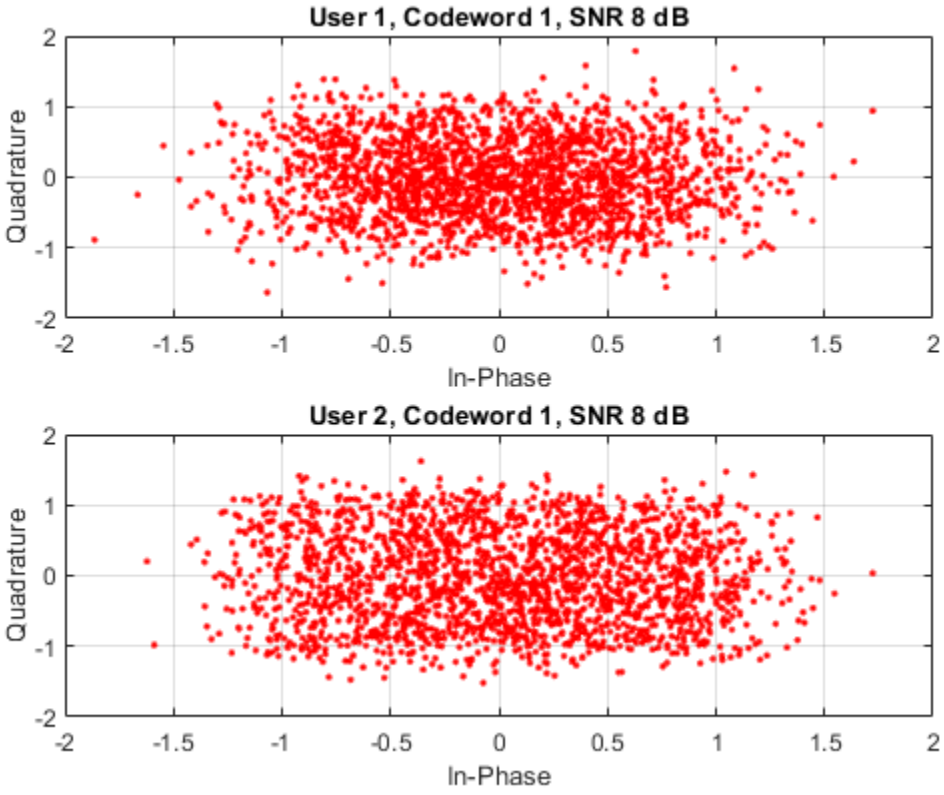
```
muChannelInit = muChannel;
muPDSCHInit = muPDSCH;
% During the simulation, some fields of enb will be updated, make a copy to
% reinitialize it when simulating each SNR point
enbInit = enb;

% For TDD precalculate vector of subframe types: D, S and U for downlink,
% special, and uplink, respectively
if strcmpi(enb.DuplexMode,'TDD')
    subframeType = char(10,1);
    initialSubframeNo = enb.NSubframe;
    for sNo=0:9 % for all subframes in a frame
        enb.NSubframe = sNo;
        duplexInfo = lteDuplexingInfo(enb);
        subframeType(sNo+1) = duplexInfo.SubframeType(1); % first char: D, S or U
    end
    enb.NSubframe = initialSubframeNo;
end

% CFI can be a scalar, or a vector of length 10 (corresponding to a frame)
% if the CFI varies per subframe. If CFI is scalar, create a local copy of
% CFI as a vector (one value per subframe).
if numel(enb.CFI) == 1
    CFI = repmat(enb.CFI,1,10);
else
    CFI = enb.CFI;
end

for snrIdx = 1:numel(SNRIn)          % Comment out for parallel computing
%parfor snrIdx = 1:numel(SNRIn)      % Uncomment for parallel computing

    % Set the random number generator seed depending on the loop variable
    % to ensure independent random streams
    rng(snrIdx,'combRecursive');

    % Reinitialize enb structures (they are modified during
    % each SNR point simulation)
    enb = enbInit;
    % Reinitialize muChannel and muPDSCH cell array
    muChannel = muChannelInit;
    muPDSCH = muPDSCHInit;

    % Initialize the state of all HARQ processes
    harqProcesses = cell(NUsers,1);
    for userIdx = 1:NUsers
        harqProcesses{userIdx} = hNewHARQProcess(enb,muPDSCH{userIdx});
    end
    harqProcessSequence = 1;

    % Set up variables for the main loop
    lastOffset = zeros(NUsers,1);
    frameOffset = zeros(NUsers,1);
    blkCRC = [];
    rxSymbols = cell(NUsers,2); % DL-SCH symbols for constellation plot
    bitTput = cell(NUsers,1);
    txedTrBlkSizes = cell(NUsers,1);
    W = cell(NUsers,1);
    pdschIndices = [];
```

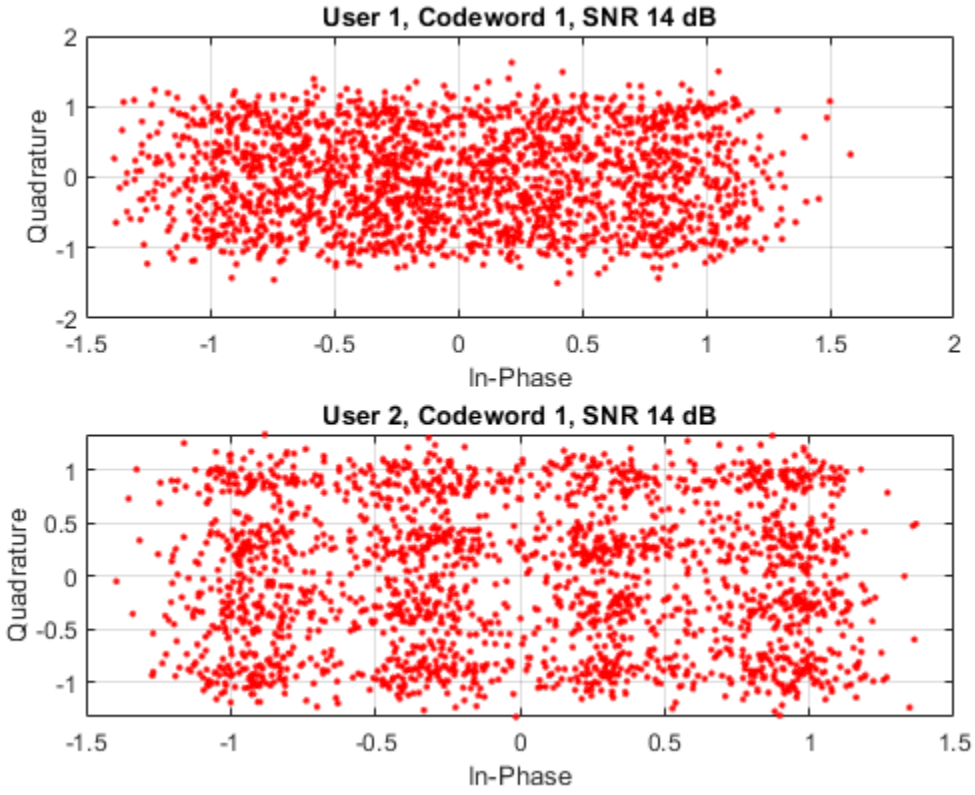
```
pdschRho = 0;

% Flag to indicate if a precoding matrix cell array W is available.
isWready = false;
% Flag to indicate if a subframe is to be processed. Set to true if
% there is data to be processed in the subframe, i.e. non-zero transport
% block size.
processSubframe = false;

% Main for loop: for all subframes
for subframeNo = 0:(NFrames*10-1)

    % Update subframe number
    enb.NSubframe = subframeNo;

    % Load CFI for current subframe
    enb.CFI = CFI(mod(subframeNo,length(CFI))+1);

    % Get HARQ process ID for the subframe from HARQ process sequence
    harqID = harqProcessSequence(mod(subframeNo,...
                                    length(harqProcessSequence))+1);

    % Channel fading process time offset for the current subframe and
    % transport block size(s)
    trBlk = zeros(NUsers,2); % User can have maximum 2 transport block
    trBlkNext = zeros(NUsers,2);
    for userIdx = 1:NUsers
        % Initialize channel fading process time offset for each subframe
        muChannel{userIdx}.InitTime = subframeNo/1000;
        trBlk(userIdx,1:ncw(userIdx)) = trBlkSizes{userIdx}(:,mod(subframeNo, 10)+1).';
        % Get transport block for next subframe
        trBlkNext(userIdx,1:ncw(userIdx))= trBlkSizes{userIdx}(:,mod(subframeNo+1,10)+1).';
    end

    % Set the flag to trigger subframe processing
    if isWready && any(trBlk(:))
        processSubframe = true;
    else
        processSubframe = false;
    end

    % Precoding matrix calculation
    if strcmpi(enb.DuplexMode,'TDD')
        % Estimate channel in UL subframe
        if strcmp(subframeType(mod(subframeNo,10)+1),'U')
            processSubframe = false; % UL subframe, no DL data
            % Only perform channel estimate if next subframe is DL
            if strcmp(subframeType(mod((subframeNo+1),10)+1),'D')
                W = hMultiUserPrecodingMatrix(enb,muPDSCH,muChannel);
                isWready = true;
            end
        end
    else %FDD

        % Calculate the precoding matrix for next subframe only if it
        % carries data (i.e. non-zero trBlkNext)
        if any(trBlkNext(:))
            W = hMultiUserPrecodingMatrix(enb,muPDSCH,muChannel);
```

```
            isWready = true;
        else
            isWready = false;
        end
    end

    % Subframe processing
    if processSubframe

        % In this example, the variables pdschRho and pdschIndices will
        % have the same values for all users
        codedTrBlk = zeros(NUsers,2);
        for userIdx = 1:NUsers
            % Update current HARQ process for all users
            harqProcesses{userIdx}(harqID) = hHARQScheduling( ...
                harqProcesses{userIdx}(harqID), subframeNo, rvSequence);
            % Map precoding matrix of all users into PDSCH configuration
            % cell array
            muPDSCH{userIdx}.W = W{userIdx};
            % PDSCH resource element power allocation in dB
            pdschRho = muPDSCH{userIdx}.Rho;
            % Generate indices for mapping of PDSCH symbols on resource
            % grid
            [pdschIndices,pdschInfo] = ltePDSCHIndices(enb,...
                muPDSCH{userIdx},muPDSCH{userIdx}.PRBSet);
            % Obtain coded transport block size
            codedTrBlk(userIdx,1:ncw(userIdx)) =  pdschInfo.G;
        end

        % Generate grid without any PDSCH mapped
        [~,txGrid,enbOut] = lteRMCDLTool(enb,[]);

        % Get the HARQ ID sequence from 'enbOut' for HARQ processing
        harqProcessSequence = enbOut.PDSCH.HARQProcessSequence;

        % Generate complex-valued modulated symbol for PDSCH in multi-user
        % MIMO transmission with block-diagonalization precoding
        pdschSymbols = hMultiUserPDSCH(enb,muPDSCH,codedTrBlk,...
            harqProcesses,harqID);
        powerAdjPerRE = 10^(pdschRho/20);

        % Perform PDSCH symbols mapping on resource grid
        txGrid(pdschIndices) = pdschSymbols*powerAdjPerRE;

        % Perform OFDM modulation
        [waveform,ofdmInfo] = lteOFDMModulate(enb,txGrid);

        % Add 25 sample padding. This is to cover the range of delays
        % expected from channel modeling (a combination of
        % implementation delay and channel delay spread)
        txWaveform = [waveform; zeros(25,NTxAnts)];

        % Calculate noise gain including compensation for downlink
        % power allocation
        SNR = 10^((SNRIn(snrIdx)-muPDSCH{userIdx}.Rho)/20);

        % Normalize noise power to take account of sampling rate,
        % which is a function of the IFFT size used in OFDM
```

```
            % modulation, and the number of antennas
            N0 = 1/(sqrt(2.0*NTxAnts*double(ofdmInfo.Nfft))*SNR);

            % Pass the waveform through noisy fading channels and receiver
            % operations for each user
            for userIdx = 1:NUsers

                % Pass data through channel model
                rxWaveform = lteFadingChannel(muChannel{userIdx},txWaveform);

                % Create additive white Gaussian noise
                noise = N0*complex(randn(size(rxWaveform)), ...
                    randn(size(rxWaveform)));

                % Add AWGN to the received time domain waveform
                rxWaveform = rxWaveform + noise;

                % Receiver
                % Synchronization offset, OFDM demodulation,
                % perfect channel estimation, PDSCH and DL-SCH Decoding
                [harqProcesses{userIdx},dlschSymbols,lastOffset(userIdx)]...
                    = hReceiverOperations(enb,muPDSCH{userIdx},rxWaveform,...
                   muChannel{userIdx},harqProcesses{userIdx},trBlk(userIdx,...
                   1:ncw(userIdx)),lastOffset(userIdx),harqID,subframeNo,noise);

                % Store the decoded DLSCH symbols for constellation
                % plotting
                rxSymbols{userIdx,1} = [rxSymbols{userIdx,1}; dlschSymbols{1}(:)];
                if ncw(userIdx)>1
                    rxSymbols{userIdx,2} = [rxSymbols{userIdx,2}; dlschSymbols{2}(:)];
                end

                % Store values to calculate throughput
                % Only for subframes with data
                if(any(trBlk(userIdx,1:ncw(userIdx))) ~= 0)
                    blkCRC = [blkCRC harqProcesses{userIdx}(harqID).blkerr];
                    bitTput{userIdx} = [bitTput{userIdx} ...
                        trBlk(userIdx,1:ncw(userIdx)).*(1-harqProcesses{userIdx}(harqID).blkerr)];
                    txedTrBlkSizes{userIdx} = [txedTrBlkSizes{userIdx} ...
                        trBlk(userIdx,1:ncw(userIdx))];
                end
            end
        end
    end

    % Calculate the maximum and simulated throughput
    maxTput = zeros(NUsers,1);
    simTput = zeros(NUsers,1);
    for userIdx = 1:NUsers
        maxTput(userIdx) = sum(txedTrBlkSizes{userIdx}); % Max possible throughput
        simTput(userIdx) = sum(bitTput{userIdx},2);      % Simulated throughput
        % Display the results dynamically in the command window
        fprintf('\nSNR = %.2f dB. Throughput for UE-%d  %d Frame(s) = %.4f Mbps\n',...
            SNRIn(snrIdx),userIdx,NFrames,1e-6*simTput(userIdx)/(NFrames*10e-3));
        fprintf('SNR = %.2f dB. Throughput(%%) for UE-%d %d Frame(s) = %.4f %%\n',...
            SNRIn(snrIdx),userIdx, NFrames,simTput(userIdx)*100/maxTput(userIdx));
    end
```

```
    maxThroughput(snrIdx,:) = maxTput;
    simThroughput(snrIdx,:) = simTput;
    rxConstellation(snrIdx,:,:)= rxSymbols;

end

% Plot received symbol constellation
for snrIdx = 1:numel(SNRIn)
    ii = 1;
    figure;
    for userIdx = 1:NUsers
        subplot(NUsers,max(ncw),ii);
        plot(rxConstellation{snrIdx,userIdx,1},'.r');
        title(['User ' num2str(userIdx) ', Codeword 1, SNR '...
            num2str(SNRIn(snrIdx)) ' dB']);
        xlabel('In-Phase');
        ylabel('Quadrature');
        grid on;
        if size(rxConstellation{snrIdx,userIdx,2})~=0
            ii = ii+1;
            subplot(NUsers,max(ncw),ii);
            plot(rxConstellation{snrIdx,userIdx,2},'.r');
            title(['User ' num2str(userIdx) ', Codeword 2, SNR '...
                num2str(SNRIn(snrIdx)) ' dB']);
            xlabel('In-Phase');
            ylabel('Quadrature');
            grid on;
        end
        ii = ii+1;
    end
end


SNR = 8.00 dB. Throughput for UE-1  2 Frame(s) = 0.0520 Mbps
SNR = 8.00 dB. Throughput(%) for UE-1 2 Frame(s) = 26.3158 %

SNR = 8.00 dB. Throughput for UE-2  2 Frame(s) = 0.1352 Mbps
SNR = 8.00 dB. Throughput(%) for UE-2 2 Frame(s) = 68.4211 %

SNR = 14.00 dB. Throughput for UE-1  2 Frame(s) = 0.1456 Mbps
SNR = 14.00 dB. Throughput(%) for UE-1 2 Frame(s) = 73.6842 %

SNR = 14.00 dB. Throughput for UE-2  2 Frame(s) = 0.1976 Mbps
SNR = 14.00 dB. Throughput(%) for UE-2 2 Frame(s) = 100.0000 %
```

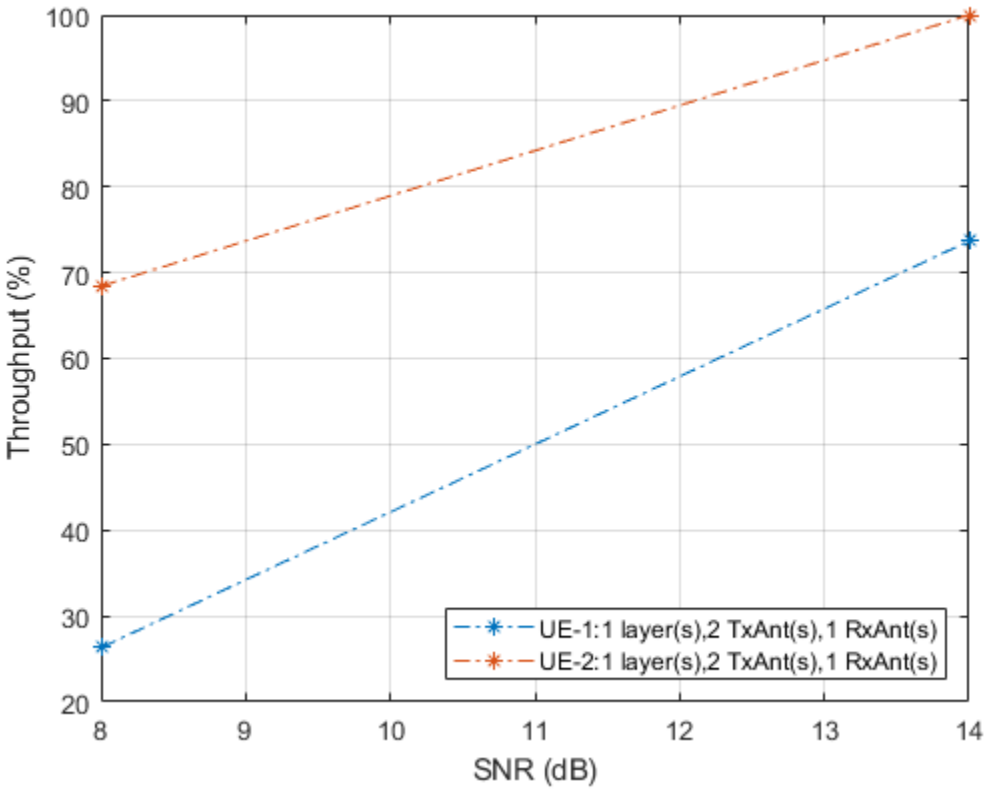

User 1, Codeword 1, SNR 8 dB
Quadrature
In-Phase
User 2, Codeword 1, SNR 8 dB
Quadrature
In-Phase



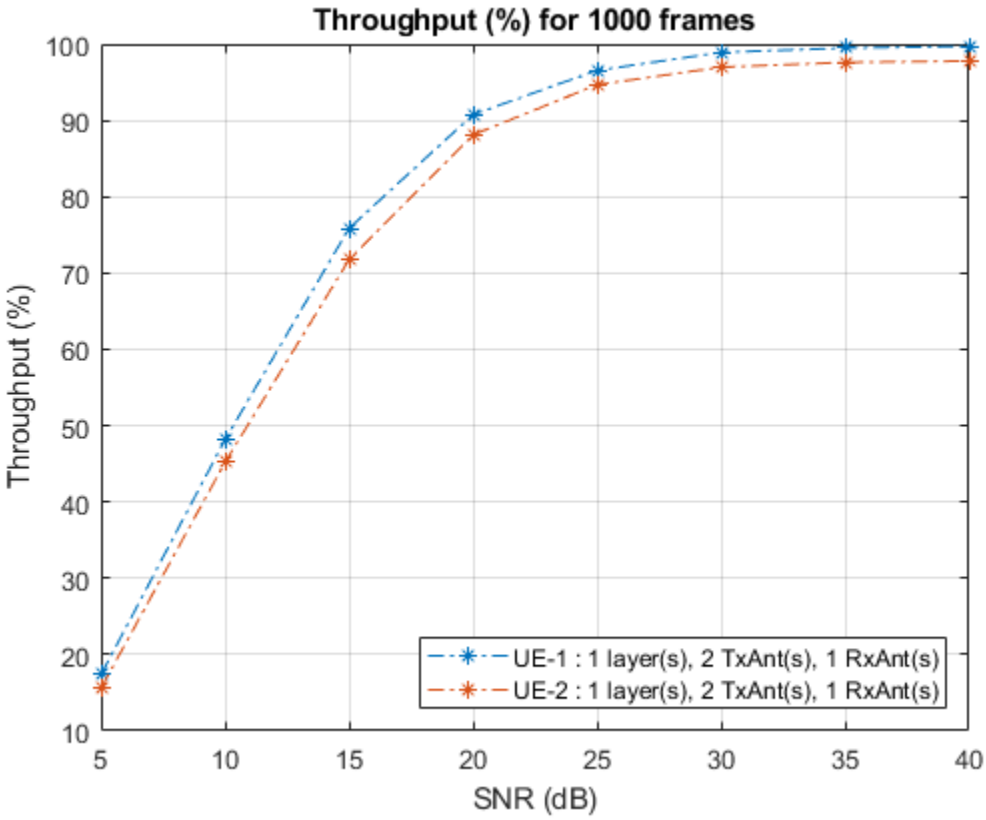
Throughput Results

The throughput results for all users are displayed in the MATLAB® command window after the simulation for each SNR point is completed. They are also captured in output arrays `simThroughput` and `maxThroughput`.

```
legendString = cell(NUsers,1);
figure;
for userIdx = 1:NUsers
    plot(SNRIn, simThroughput(:,userIdx)*100./maxThroughput(:,userIdx),'*-.');
    hold on;
    legendString{userIdx} = strcat('UE-' ,num2str(userIdx), ': ', ...
        num2str(muNumLayers(userIdx)), ' layer(s), ' ,num2str(NTxAnts), ...
        ' TxAnt(s), ', num2str(muNumRxAnts(userIdx)), ' RxAnt(s)');
end
grid on;
xlabel('SNR (dB)');
ylabel('Throughput (%)');
legend(legendString,'Location','SouthEast');
```



For statistically valid results, the simulation should be run for a larger number of frames. The figure below shows the throughput results when simulating 1000 frames.



Appendix

This example uses the following helper functions:

- hMultiUserPrecodingMatrix.m
- hMultiUserPDSCH.m
- hReceiverOperations.m
- hMultiUserParameterSummary.m
- hNewHARQProcess.m
- hHARQScheduling.m

Selected Bibliography

1 Spencer Q., A. Swindlehurst, M. Haardt. "Zero-Forcing Methods for Downlink Spatial Multiplexing in Multiuser MIMO Channels." IEEE Transactions on Signal Processing, Vol. 52, No. 2, February 2004, pp. 461-471.

2 Lim C., T. Yoo, B. Clerckx, B. Lee, B. Shim. "Recent trend of multi-user MIMO in LTE-advanced." IEEE Communications Magazine, March 2013, pp. 127-135.

PDSCH Throughput Conformance Test for Single Antenna (TM1), Transmit Diversity (TM2), Open Loop (TM3) and Closed Loop (TM4/6) Spatial Multiplexing

This example demonstrates how to measure the Physical Downlink Shared Channel (PDSCH) throughput performance using LTE Toolbox™ for the following transmission modes (TM):

- TM1: Single antenna (Port 0)
- TM2: Transmit diversity
- TM3: Open loop codebook based precoding: Cyclic Delay Diversity (CDD)
- TM4: Closed loop codebook based spatial multiplexing
- TM6: Single layer closed loop codebook based spatial multiplexing

The example also shows how to parameterize and customize the settings for the different TMs. It also supports the use of Parallel Computing Toolbox™ to reduce effective simulation time.

Introduction

This example measures the throughput for a number of SNR points. The provided code can operate under a number of transmission modes: TM1, TM2, TM3, TM4 and TM6. For information on how to model TM7, TM8, TM9 and TM10 check the following example: “PDSCH Throughput for Non-Codebook Based Precoding Schemes: Port 5 (TM7), Port 7 or 8 or Port 7-8 (TM8), Port 7-14 (TM9 and TM10)” on page 2-424

The example works on a subframe by subframe basis. For each of the considered SNR points a populated resource grid is generated and OFDM modulated to create a transmit waveform. The generated waveform is passed through a noisy fading channel. The following operations are then performed by the receiver: channel estimation, equalization, demodulation and decoding. The throughput performance of the PDSCH is determined using the block CRC result at the output of the channel decoder.

Precoder Matrix Indication (PMI) feedback is implemented for the TMs requiring the feedback of a precoding matrix (TM4 and TM6).

A `parfor` loop can be used instead of the `for` loop for the SNR calculation. This is indicated within the example. The `parfor` statement is part of the Parallel Computing Toolbox and executes the SNR loop in parallel to reduce the total simulation time.

Simulation Configuration

The example is executed for a simulation length of 2 frames for a number of SNR points. A large number of `NFrames` should be used to produce meaningful throughput results. `SNRIn` can be an array of values or a scalar. Some TMs and certain modulation schemes are more robust to noise and channel impairments than others, therefore different values of SNR may have to be used for different parameter sets.

```
NFrames = 2;                % Number of frames
SNRIn = [10.3 12.3 14.3];   % SNR range in dB
```

eNodeB Configuration

This section selects the TM of interest and sets the eNodeB parameters. The TM is selected using the variable `txMode`, which can take the values TM1, TM2, TM3, TM4 and TM6.

```
txMode = 'TM4'; % TM1, TM2, TM3, TM4, TM6
```

For simplicity all TMs modeled in this example have a bandwidth of 50 resource blocks with a full allocation and a code rate of 0.5. Not specifying an RMC number ensures that all downlink subframes are scheduled. If RMC is specified (e.g. 'R.0'), the subframe scheduling is as defined in TS 36.101 where subframe 5 is not scheduled in most cases.

The variable `txMode` selects the TM via a switch statement. For each TM, the required parameters are specified. This example does not perform DCI format decoding, so the `DCIFormat` field is not strictly necessary. However, since the DCI format is closely linked to the TM, it is included for completeness.

```
simulationParameters = []; % clear simulationParameters
simulationParameters.NDLRB = 50;
simulationParameters.PDSCH.TargetCodeRate = 0.5;
simulationParameters.PDSCH.PRBSet = (0:49)';

switch txMode
% Single antenna (Port0) mode (TM1)
    case 'TM1'
        fprintf('\nTM1 - Single antenna (port 0)\n');
        simulationParameters.PDSCH.TxScheme = 'Port0';
        simulationParameters.PDSCH.DCIFormat = 'Format1';
        simulationParameters.CellRefP = 1;
        simulationParameters.PDSCH.Modulation = {'16QAM'};

% Transmit diversity mode (TM2)
    case 'TM2'
        fprintf('\nTM2 - Transmit diversity\n');
        simulationParameters.PDSCH.TxScheme = 'TxDiversity';
        simulationParameters.PDSCH.DCIFormat = 'Format1';
        simulationParameters.CellRefP = 2;
        simulationParameters.PDSCH.Modulation = {'16QAM'};
        simulationParameters.PDSCH.NLayers = 2;

% CDD mode (TM3)
    case 'TM3'
        fprintf('\nTM3 - CDD\n');
        simulationParameters.PDSCH.TxScheme = 'CDD';
        simulationParameters.PDSCH.DCIFormat = 'Format2A';
        simulationParameters.CellRefP = 2;
        simulationParameters.PDSCH.Modulation = {'16QAM', '16QAM'};
        simulationParameters.PDSCH.NLayers = 2;

% Spatial multiplexing mode (TM4)
    case 'TM4'
        fprintf('\nTM4 - Codebook based spatial multiplexing\n');
        simulationParameters.CellRefP = 2;
        simulationParameters.PDSCH.Modulation = {'16QAM', '16QAM'};
        simulationParameters.PDSCH.DCIFormat = 'Format2';
        simulationParameters.PDSCH.TxScheme = 'SpatialMux';
        simulationParameters.PDSCH.NLayers = 2;
```

```
        % No codebook restriction
        simulationParameters.PDSCH.CodebookSubset = '';

% Single layer spatial multiplexing mode (TM6)
    case 'TM6'
        fprintf(...
        '\nTM6 - Codebook based spatial multiplexing with single layer\n');
        simulationParameters.CellRefP = 4;
        simulationParameters.PDSCH.Modulation = {'QPSK'};
        simulationParameters.PDSCH.DCIFormat = 'Format2';
        simulationParameters.PDSCH.TxScheme = 'SpatialMux';
        simulationParameters.PDSCH.NLayers = 1;
        % No codebook restriction
        simulationParameters.PDSCH.CodebookSubset = '';

    otherwise
        error('Transmission mode should be one of TM1, TM2, TM3, TM4 or TM6.')
end

% Set other simulationParameters fields applying to all TMs
simulationParameters.TotSubframes = 1; % Generate one subframe at a time
simulationParameters.PDSCH.CSI = 'On'; % Soft bits are weighted by CSI
```

```
TM4 - Codebook based spatial multiplexing
```

Call `lteRMCDL` to generate the default eNodeB parameters not specified in `simulationParameters`. These will be required later to generate the waveform using `lteRMCDLTool`.

```
enb = lteRMCDL(simulationParameters);
```

The output `enb` structure contains, among other fields, the transport block sizes and redundancy version sequence for each codeword subframe within a frame. These will be used later in the simulation.

```
rvSequence = enb.PDSCH.RVSeq;
trBlkSizes = enb.PDSCH.TrBlkSizes;
```

The number of codewords, `ncw`, is the number of entries in the `enb.PDSCH.Modulation` field.

```
ncw = length(string(enb.PDSCH.Modulation));
```

Set the PMI delay for the closed-loop TMs (TM4 and TM6). This is the delay between a PMI being passed from UE to eNodeB as defined in TS 36.101, Table 8.2.1.4.2-1.

```
pmiDelay = 8;
```

Next we print a summary of some of the more relevant simulation parameters. Check these values to make sure they are as expected. The code rate displayed can be useful to detect problems if manually specifying the transport block sizes. Typical values are 1/3, 1/2 and 3/4.

```
hDisplayENBParameterSummary(enb, txMode);
```

```
-- Parameter summary: -------------------------------------------
                     Duplexing mode: FDD
                  Transmission mode: TM4
```

```
                Transmission scheme: SpatialMux
  Number of downlink resource blocks: 50
 Number of allocated resource blocks: 50
Cell-specific reference signal ports: 2
         Number of transmit antennas: 2
                 Transmission layers: 2
                 Number of codewords: 2
               Modulation codeword 1: 16QAM
    Transport block sizes codeword 1:    11448   11448   11448   11448   11448   11448   11448
                 Code rate codeword 1:    0.515    0.48    0.48    0.48    0.48  0.4918    0.48
               Modulation codeword 2: 16QAM
    Transport block sizes codeword 2:    11448   11448   11448   11448   11448   11448   11448
                 Code rate codeword 2:    0.515    0.48    0.48    0.48    0.48  0.4918    0.48
------------------------------------------------------------------
```

Propagation Channel Model Configuration

The structure `channel` contains the channel model configuration parameters.

```
channel.Seed = 6;                    % Channel seed
channel.NRxAnts = 2;                 % 2 receive antennas
channel.DelayProfile = 'EPA';        % Delay profile
channel.DopplerFreq = 5;             % Doppler frequency
channel.MIMOCorrelation = 'Low';     % Multi-antenna correlation
channel.NTerms = 16;                 % Oscillators used in fading model
channel.ModelType = 'GMEDS';         % Rayleigh fading model type
channel.InitPhase = 'Random';        % Random initial phases
channel.NormalizePathGains = 'On';   % Normalize delay profile power
channel.NormalizeTxAnts = 'On';      % Normalize for transmit antennas
```

The sampling rate for the channel model is set using the value returned from `lteOFDMInfo`.

```
ofdmInfo = lteOFDMInfo(enb);
channel.SamplingRate = ofdmInfo.SamplingRate;

% Maximum channel delay (multipath and channel filtering)
channel.InitTime = 0;
[~,chInfo] = lteFadingChannel(channel,0); % get channel info
maxChDelay = ceil(max(chInfo.PathSampleDelays)) + chInfo.ChannelFilterDelay;
```

Channel Estimator Configuration

The variable `perfectChanEstimator` controls channel estimator behavior. Valid values are `true` or `false`. When set to `true` a perfect channel response is used as estimate, otherwise an imperfect estimation based on the values of received pilot signals is obtained.

```
% Perfect channel estimator flag
perfectChanEstimator = false;
```

If `perfectChanEstimator` is set to false a configuration structure `cec` is needed to parameterize the channel estimator. The channel changes slowly in time and frequency, therefore a large averaging window is used in order to average the noise out.

```
% Configure channel estimator
cec.PilotAverage = 'UserDefined';   % Type of pilot symbol averaging
cec.FreqWindow = 41;                % Frequency window size in REs
cec.TimeWindow = 27;                % Time window size in REs
cec.InterpType = 'Cubic';           % 2D interpolation type
```

```
cec.InterpWindow = 'Centered';      % Interpolation window type
cec.InterpWinSize = 1;              % Interpolation window size
```

Display Simulation Information

The variable `displaySimulationInformation` controls the display of simulation information such as the HARQ process ID used for each subframe. In case of CRC error the value of the index to the RV sequence is also displayed.

```
displaySimulationInformation = true;
```

Processing Loop

To determine the throughput at each SNR point, the PDSCH data is analyzed on a subframe by subframe basis using the following steps:

- *Update Current HARQ Process.* The HARQ process either carries new transport data or a retransmission of previously sent transport data depending upon the Acknowledgment (ACK) or Negative Acknowledgment (NACK) based on CRC results. All this is handled by the HARQ scheduler, `hHARQScheduling`. The PDSCH data is updated based on the HARQ state.
- *Set PMI.* This step is only applicable to TM4 and TM6 (closed loop spatial multiplexing and single layer closed loop spatial multiplexing). A PMI is taken sequentially from a set of PMIs, `txPMIs`, each subframe and used by the eNodeB to select a precoding matrix. The PMI recommended by the UE is used by the eNodeB for data transmission. There is a delay of `pmiDelay` subframes between the UE recommending the PMI and the eNodeB using it to select a precoding matrix. Initially a set of `pmiDelay` random PMIs is used.
- *Create Transmit Waveform.* The data generated by the HARQ process is passed to `lteRMCDLTool` which produces an OFDM modulated waveform, containing the physical channels and signals.
- *Noisy Channel Modeling.* The waveform is passed through a fading channel and noise (AWGN) is added.
- *Perform Synchronization and OFDM Demodulation.* The received symbols are offset to account for a combination of implementation delay and channel delay spread. The symbols are then OFDM demodulated.
- *Perform Channel Estimation.* The channel response and noise levels are estimated. These estimates are used to decode the PDSCH.
- *Decode the PDSCH.* The recovered PDSCH symbols for all transmit and receive antenna pairs, along with a noise estimate, are demodulated and descrambled by `ltePDSCHDecode` to obtain an estimate of the received codewords.
- *Decode the Downlink Shared Channel (DL-SCH) and Store the Block CRC Error for a HARQ Process.* The vector of decoded soft bits is passed to `lteDLSCHDecode`; this decodes the codeword and returns the block CRC error used to determine the throughput of the system. The contents of the new soft buffer, `harqProc(harqID).decState`, is available at the output of this function to be used when decoding the next subframe.
- *Update PMI.* A PMI is selected and fed back to the eNodeB for future use. This step is only applicable to TM4 and TM6 (close loop spatial multiplexing and single layer closed loop spatial multiplexing).

```
% The number of transmit antennas P is obtained from the resource grid
% dimensions. 'dims' is M-by-N-by-P where M is the number of subcarriers, N
```

```
% is the number of symbols and P is the number of transmit antennas.
dims = lteDLResourceGridSize(enb);
P = dims(3);

% Initialize variables used in the simulation and analysis
% Array to store the maximum throughput for all SNR points
maxThroughput = zeros(length(SNRIn),1);
% Array to store the simulation throughput for all SNR points
simThroughput = zeros(length(SNRIn),1);

% Get the HARQ ID sequence for HARQ processing. This is a list of indices
% for HARQ process scheduling.
[~,~,enbOut] = lteRMCDLTool(enb, []);
harqProcessSequence = enbOut.PDSCH.HARQProcessSequence;

% Temporary variables 'enb_init', 'channel_init' and
% 'harqProcessSequence_init' are used to optimize parallel processing (only
% if running the example with Parallel Computing Toolbox)
enb_init = enb;
channel_init = channel;
harqProcessSequence_init = harqProcessSequence;
legendString = ['Throughput: ' char(enb.PDSCH.TxScheme)];
allRvSeqPtrHistory = cell(1,numel(SNRIn));
nFFT = ofdmInfo.Nfft;

for snrIdx = 1:numel(SNRIn)
% parfor snrIdx = 1:numel(SNRIn)
% To enable the use of parallel computing for increased speed comment out
% the 'for' statement above and uncomment the 'parfor' statement below.
% This needs the Parallel Computing Toolbox. If this is not installed
% 'parfor' will default to the normal 'for' statement. If 'parfor' is
% used it is recommended that the variable 'displaySimulationInformation'
% above is set to false, otherwise the simulation information displays for
% each SNR point will overlap.

    % Set the random number generator seed depending to the loop variable
    % to ensure independent random streams
    rng(snrIdx,'combRecursive');

    SNRdB = SNRIn(snrIdx);
    fprintf('\nSimulating at %g dB SNR for %d Frame(s)\n' ,SNRdB, NFrames);

    % Initialize variables used in the simulation and analysis
    offsets = 0;            % Initialize frame offset value
    offset = 0;             % Initialize frame offset value for radio frame
    blkCRC = [];            % Block CRC for all considered subframes
    bitTput = [];           % Number of successfully received bits per subframe
    txedTrBlkSizes = [];    % Number of transmitted bits per subframe
    enb = enb_init;         % Initialize RMC configuration
    channel = channel_init; % Initialize channel configuration
    harqProcessSequence = harqProcessSequence_init; % Initialize HARQ process sequence
    pmiIdx = 0;             % PMI index in delay queue

    % The variable harqPtrTable stores the history of the value of the
    % pointer to the RV sequence values for all the HARQ processes.
    % Pre-allocate with NaNs as some subframes do not have data
    rvSeqPtrHistory = NaN(ncw, NFrames*10);
```

```
% Initialize state of all HARQ processes
harqProcesses = hNewHARQProcess(enb);

% Use random PMIs for the first 'pmiDelay' subframes until feedback is
% available from the UE; note that PMI feedback is only applicable for
% spatial multiplexing TMs (TM4 and TM6), but the code here is required
% for complete initialization of variables in the SNR loop when using
% the Parallel Computing Toolbox.
pmidims = ltePMIInfo(enb,enb.PDSCH);
txPMIs = randi([0 pmidims.MaxPMI], pmidims.NSubbands, pmiDelay);

for subframeNo = 0:(NFrames*10-1)

    % Update subframe number
    enb.NSubframe = subframeNo;

    % Get HARQ process ID for the subframe from HARQ process sequence
    harqID = harqProcessSequence(mod(subframeNo, length(harqProcessSequence))+1);

    % If there is a transport block scheduled in the current subframe
    % (indicated by non-zero 'harqID'), perform transmission and
    % reception. Otherwise continue to the next subframe
    if harqID == 0
        continue;
    end

    % Update current HARQ process
    harqProcesses(harqID) = hHARQScheduling( ...
        harqProcesses(harqID), subframeNo, rvSequence);

    % Extract the current subframe transport block size(s)
    trBlk = trBlkSizes(:, mod(subframeNo, 10)+1).';

    % Display run time information
    if displaySimulationInformation
        disp(' ');
        disp(['Subframe: ' num2str(subframeNo)...
                        '. HARQ process ID: ' num2str(harqID)]);
    end

    % Update RV sequence pointer table
    rvSeqPtrHistory(:,subframeNo+1) = ...
                            harqProcesses(harqID).txConfig.RVIdx.';

    % Update the PDSCH transmission config with HARQ process state
    enb.PDSCH = harqProcesses(harqID).txConfig;
    data = harqProcesses(harqID).data;

    % Set the PMI to the appropriate value in the delay queue
    if strcmpi(enb.PDSCH.TxScheme,'SpatialMux')
        pmiIdx = mod(subframeNo, pmiDelay);  % PMI index in delay queue
        enb.PDSCH.PMISet = txPMIs(:, pmiIdx+1); % Set PMI
    end

    % Generate transmit waveform
    txWaveform = lteRMCDLTool(enb, data);

    % Add maxChDelay sample padding. This is to cover the range of
```

```
% delays expected from channel modeling (a combination of
% implementation delay and channel delay spread)
txWaveform =  [txWaveform; zeros(maxChDelay, P)]; %#ok<AGROW>

% Get the HARQ ID sequence from 'enbOut' for HARQ processing
harqProcessSequence = enbOut.PDSCH.HARQProcessSequence;

% Initialize channel time for each subframe
channel.InitTime = subframeNo/1000;

% Pass data through channel model
rxWaveform = lteFadingChannel(channel, txWaveform);

% Calculate noise gain including compensation for downlink power
% allocation
SNR = 10^((SNRdB-enb.PDSCH.Rho)/20);

% Normalize noise power to take account of sampling rate, which is
% a function of the IFFT size used in OFDM modulation, and the
% number of antennas
N0 = 1/(sqrt(2.0*enb.CellRefP*double(nFFT))*SNR);

% Create additive white Gaussian noise
noise = N0*complex(randn(size(rxWaveform)), ...
                   randn(size(rxWaveform)));

% Add AWGN to the received time domain waveform
rxWaveform = rxWaveform + noise;

% Once every frame, on subframe 0, calculate a new synchronization
% offset
if (mod(subframeNo,10) == 0)
    offset = lteDLFrameOffset(enb, rxWaveform);
    if (offset > maxChDelay)
        offset = offsets(end);
    end
    offsets = [offsets offset]; %#ok
end

% Synchronize the received waveform
rxWaveform = rxWaveform(1+offset:end, :);

% Perform OFDM demodulation on the received data to recreate the
% resource grid
rxSubframe = lteOFDMDemodulate(enb, rxWaveform);

% Channel estimation
if(perfectChanEstimator)
    estChannelGrid = lteDLPerfectChannelEstimate(enb, channel, offset); %#ok
    noiseGrid = lteOFDMDemodulate(enb, noise(1+offset:end ,:));
    noiseEst = var(noiseGrid(:));
else
    [estChannelGrid, noiseEst] = lteDLChannelEstimate( ...
        enb, enb.PDSCH, cec, rxSubframe);
end

% Get PDSCH indices
pdschIndices = ltePDSCHIndices(enb, enb.PDSCH, enb.PDSCH.PRBSet);
```

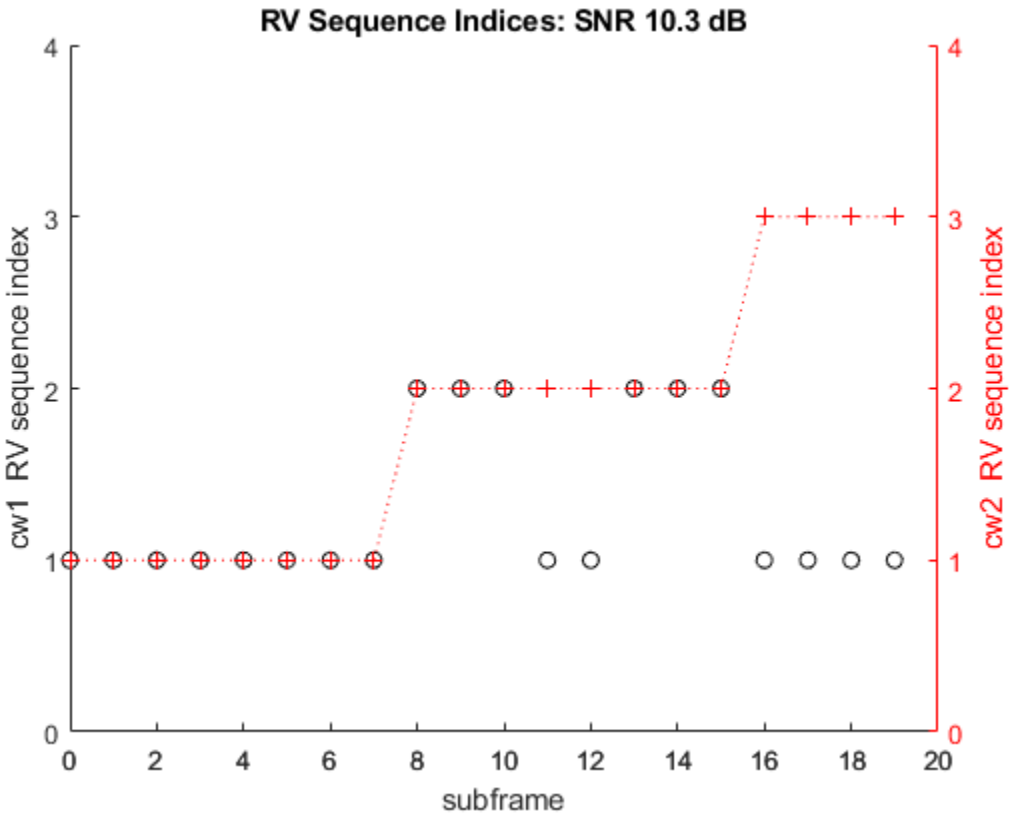
```
            % Get PDSCH resource elements from the received subframe. Scale the
            % received subframe by the PDSCH power factor Rho. The PDSCH is
            % scaled by this amount, while the cell reference symbols used for
            % channel estimation (used in the PDSCH decoding stage) are not.
            [pdschRx, pdschHest] = lteExtractResources(pdschIndices, ...
                rxSubframe*(10^(-enb.PDSCH.Rho/20)), estChannelGrid);

            % Decode PDSCH
            dlschBits = ltePDSCHDecode(...
                                enb, enb.PDSCH, pdschRx, pdschHest, noiseEst);

            % Decode the DL-SCH
            [decbits, harqProcesses(harqID).blkerr,harqProcesses(harqID).decState] = ...
                lteDLSCHDecode(enb, enb.PDSCH, trBlk, dlschBits, ...
                               harqProcesses(harqID).decState);

            % Display block errors
            if displaySimulationInformation
                if any(harqProcesses(harqID).blkerr)
                    disp(['Block error. RV index: ' num2str(harqProcesses(harqID).txConfig.RVIdx)...
                          ', CRC: ' num2str(harqProcesses(harqID).blkerr)])
                else
                    disp(['No error. RV index: ' num2str(harqProcesses(harqID).txConfig.RVIdx)...
                          ', CRC: ' num2str(harqProcesses(harqID).blkerr)])
                end
            end

            % Store values to calculate throughput
            % Only for subframes with data
            if any(trBlk)
                blkCRC = [blkCRC harqProcesses(harqID).blkerr]; %#ok<AGROW>
                bitTput = [bitTput trBlk.*(1- ...
                    harqProcesses(harqID).blkerr)]; %#ok<AGROW>
                txedTrBlkSizes = [txedTrBlkSizes trBlk]; %#ok<AGROW>
            end

            % Provide PMI feedback to the eNodeB
            if strcmpi(enb.PDSCH.TxScheme,'SpatialMux')
                PMI = ltePMISelect(enb, enb.PDSCH, estChannelGrid, noiseEst);
                txPMIs(:, pmiIdx+1) = PMI;
            end
        end

        % Calculate maximum and simulated throughput
        maxThroughput(snrIdx) = sum(txedTrBlkSizes); % Max possible throughput
        simThroughput(snrIdx) = sum(bitTput,2);      % Simulated throughput

        % Display the results dynamically in the command window
        fprintf([['\nThroughput(Mbps) for ', num2str(NFrames) ' Frame(s) '],...
            '= %.4f\n'], 1e-6*simThroughput(snrIdx)/(NFrames*10e-3));
        fprintf(['Throughput(%%) for ', num2str(NFrames) ' Frame(s) = %.4f\n'],...
            simThroughput(snrIdx)*100/maxThroughput(snrIdx));

        allRvSeqPtrHistory{snrIdx} = rvSeqPtrHistory;

    end
```

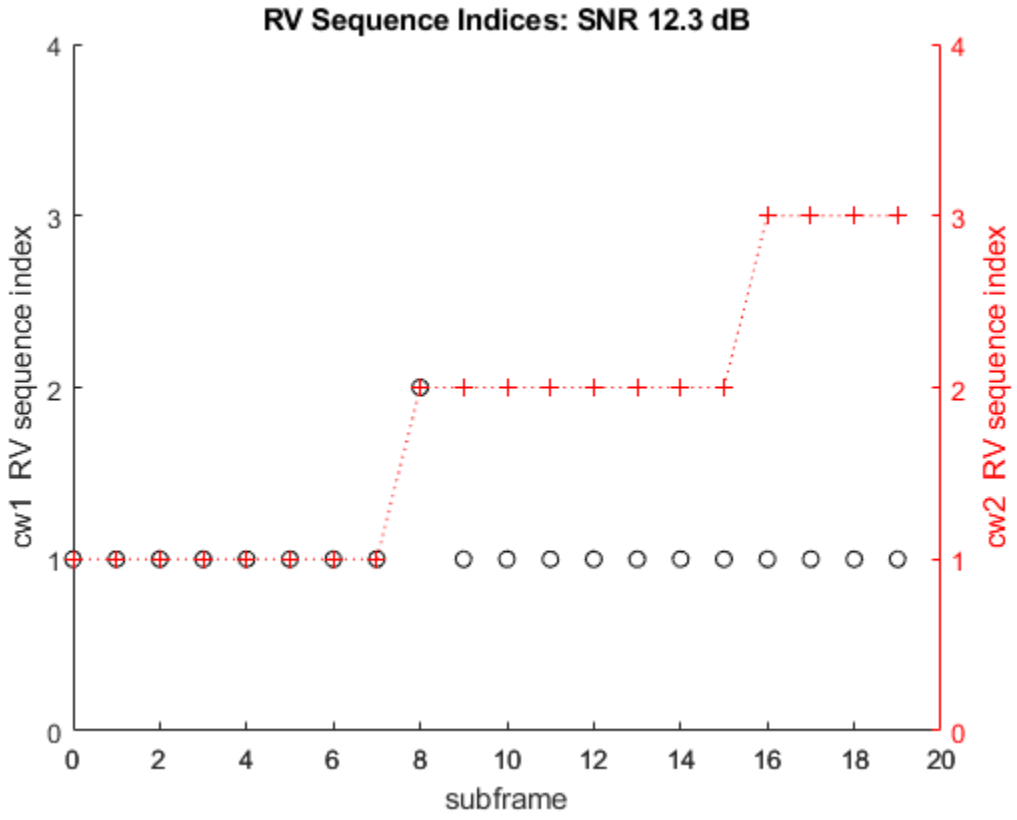
```
% Plot the RV sequence for all HARQ processes
hPlotRVSequence(SNRIn,allRvSeqPtrHistory,NFrames);


Simulating at 10.3 dB SNR for 2 Frame(s)

Subframe: 0. HARQ process ID: 1
Block error. RV index: 1  1, CRC: 1  1

Subframe: 1. HARQ process ID: 2
Block error. RV index: 1  1, CRC: 1  1

Subframe: 2. HARQ process ID: 3
Block error. RV index: 1  1, CRC: 1  1

Subframe: 3. HARQ process ID: 4
Block error. RV index: 1  1, CRC: 0  1

Subframe: 4. HARQ process ID: 5
Block error. RV index: 1  1, CRC: 0  1

Subframe: 5. HARQ process ID: 6
Block error. RV index: 1  1, CRC: 1  1

Subframe: 6. HARQ process ID: 7
Block error. RV index: 1  1, CRC: 1  1

Subframe: 7. HARQ process ID: 8
Block error. RV index: 1  1, CRC: 1  1

Subframe: 8. HARQ process ID: 1
Block error. RV index: 2  2, CRC: 0  1

Subframe: 9. HARQ process ID: 2
Block error. RV index: 2  2, CRC: 0  1

Subframe: 10. HARQ process ID: 3
Block error. RV index: 2  2, CRC: 0  1

Subframe: 11. HARQ process ID: 4
Block error. RV index: 1  2, CRC: 0  1

Subframe: 12. HARQ process ID: 5
Block error. RV index: 1  2, CRC: 0  1

Subframe: 13. HARQ process ID: 6
Block error. RV index: 2  2, CRC: 0  1

Subframe: 14. HARQ process ID: 7
Block error. RV index: 2  2, CRC: 0  1

Subframe: 15. HARQ process ID: 8
Block error. RV index: 2  2, CRC: 0  1

Subframe: 16. HARQ process ID: 1
No error. RV index: 1  3, CRC: 0  0

Subframe: 17. HARQ process ID: 2
```

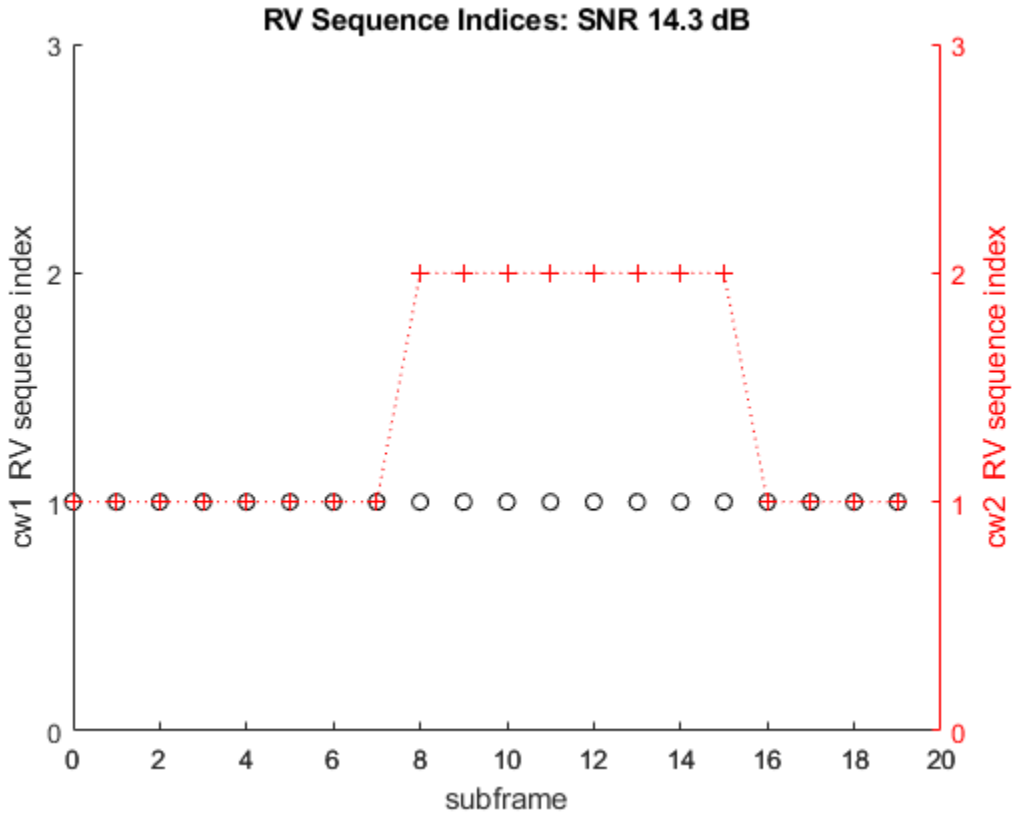
```
No error. RV index: 1  3, CRC: 0  0

Subframe: 18. HARQ process ID: 3
No error. RV index: 1  3, CRC: 0  0

Subframe: 19. HARQ process ID: 4
Block error. RV index: 1  3, CRC: 0  1

Throughput(Mbps) for 2 Frame(s) = 9.7308
Throughput(%) for 2 Frame(s) = 42.5000

Simulating at 12.3 dB SNR for 2 Frame(s)

Subframe: 0. HARQ process ID: 1
Block error. RV index: 1  1, CRC: 1  1

Subframe: 1. HARQ process ID: 2
Block error. RV index: 1  1, CRC: 0  1

Subframe: 2. HARQ process ID: 3
Block error. RV index: 1  1, CRC: 0  1

Subframe: 3. HARQ process ID: 4
Block error. RV index: 1  1, CRC: 0  1

Subframe: 4. HARQ process ID: 5
Block error. RV index: 1  1, CRC: 0  1

Subframe: 5. HARQ process ID: 6
Block error. RV index: 1  1, CRC: 0  1

Subframe: 6. HARQ process ID: 7
Block error. RV index: 1  1, CRC: 0  1

Subframe: 7. HARQ process ID: 8
Block error. RV index: 1  1, CRC: 0  1

Subframe: 8. HARQ process ID: 1
Block error. RV index: 2  2, CRC: 0  1

Subframe: 9. HARQ process ID: 2
Block error. RV index: 1  2, CRC: 0  1

Subframe: 10. HARQ process ID: 3
Block error. RV index: 1  2, CRC: 0  1

Subframe: 11. HARQ process ID: 4
Block error. RV index: 1  2, CRC: 0  1

Subframe: 12. HARQ process ID: 5
Block error. RV index: 1  2, CRC: 0  1

Subframe: 13. HARQ process ID: 6
Block error. RV index: 1  2, CRC: 0  1

Subframe: 14. HARQ process ID: 7
No error. RV index: 1  2, CRC: 0  0
```

```
Subframe: 15. HARQ process ID: 8
No error. RV index: 1  2, CRC: 0  0

Subframe: 16. HARQ process ID: 1
No error. RV index: 1  3, CRC: 0  0

Subframe: 17. HARQ process ID: 2
No error. RV index: 1  3, CRC: 0  0

Subframe: 18. HARQ process ID: 3
No error. RV index: 1  3, CRC: 0  0

Subframe: 19. HARQ process ID: 4
No error. RV index: 1  3, CRC: 0  0

Throughput(Mbps) for 2 Frame(s) = 14.3100
Throughput(%) for 2 Frame(s) = 62.5000

Simulating at 14.3 dB SNR for 2 Frame(s)

Subframe: 0. HARQ process ID: 1
Block error. RV index: 1  1, CRC: 0  1

Subframe: 1. HARQ process ID: 2
Block error. RV index: 1  1, CRC: 0  1

Subframe: 2. HARQ process ID: 3
Block error. RV index: 1  1, CRC: 0  1

Subframe: 3. HARQ process ID: 4
Block error. RV index: 1  1, CRC: 0  1

Subframe: 4. HARQ process ID: 5
Block error. RV index: 1  1, CRC: 0  1

Subframe: 5. HARQ process ID: 6
Block error. RV index: 1  1, CRC: 0  1

Subframe: 6. HARQ process ID: 7
Block error. RV index: 1  1, CRC: 0  1

Subframe: 7. HARQ process ID: 8
Block error. RV index: 1  1, CRC: 0  1

Subframe: 8. HARQ process ID: 1
No error. RV index: 1  2, CRC: 0  0

Subframe: 9. HARQ process ID: 2
No error. RV index: 1  2, CRC: 0  0

Subframe: 10. HARQ process ID: 3
No error. RV index: 1  2, CRC: 0  0

Subframe: 11. HARQ process ID: 4
No error. RV index: 1  2, CRC: 0  0

Subframe: 12. HARQ process ID: 5
No error. RV index: 1  2, CRC: 0  0
```

```
Subframe: 13. HARQ process ID: 6
No error. RV index: 1  2, CRC: 0  0

Subframe: 14. HARQ process ID: 7
No error. RV index: 1  2, CRC: 0  0

Subframe: 15. HARQ process ID: 8
No error. RV index: 1  2, CRC: 0  0

Subframe: 16. HARQ process ID: 1
Block error. RV index: 1  1, CRC: 0  1

Subframe: 17. HARQ process ID: 2
Block error. RV index: 1  1, CRC: 0  1

Subframe: 18. HARQ process ID: 3
Block error. RV index: 1  1, CRC: 0  1

Subframe: 19. HARQ process ID: 4
Block error. RV index: 1  1, CRC: 0  1

Throughput(Mbps) for 2 Frame(s) = 16.0272
Throughput(%) for 2 Frame(s) = 70.0000
```




RV Sequence Indices: SNR 12.3 dB
cw1 RV sequence index
cw2 RV sequence index
subframe
0
1
2
3
4
0
2
4
6
8
10
12
14
16
18
20

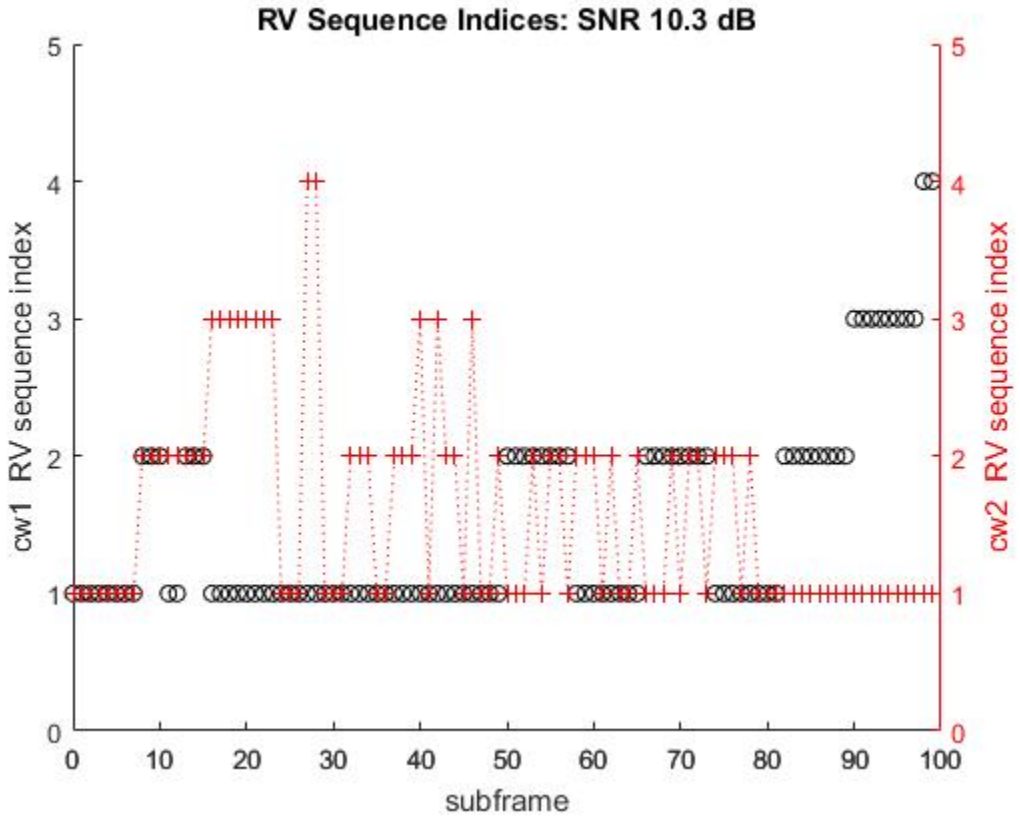


RV Sequence Pointer Plots

The code above also generates plots with the value of the pointers to the elements in the RV sequence for the simulated subframes. This provides an idea of the retransmissions required. We plot the pointers and note the RV values used in case these are not organized in ascending order. For example, in some cases the RV sequence can be [0, 2, 3, 1]. Plotting these values as they are used will not provide a clear idea of the number of retransmissions needed.

When transmitting a new transport block, the first element of the RV sequence is used. In the plots above a value of 1 is shown for that subframe. This is the case at the beginning of the simulation. If a retransmission is required, the next element in the RV sequence is selected and the pointer is increased. A value of 2 will be plotted for the subframe where the retransmission takes place. If further retransmissions are required, the pointer value will increase further. Note that the plots do not show any value in subframe 5 of consecutive frames. This is because no data is transmitted in those subframes.

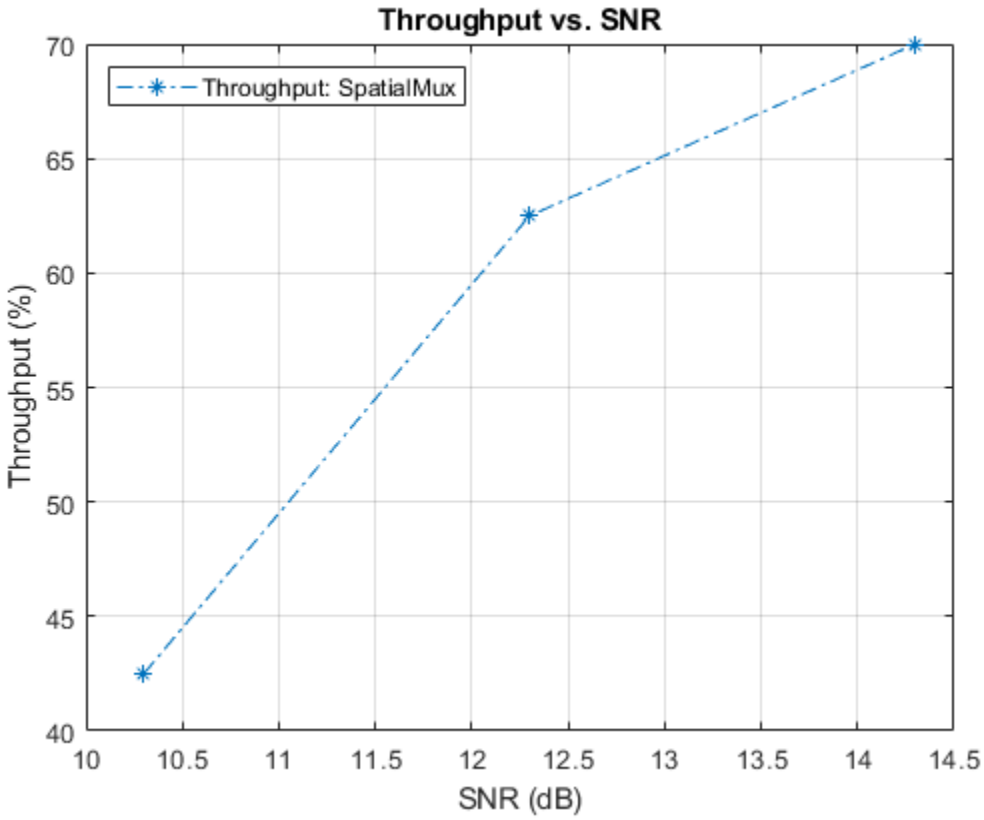
The figure shown below was obtained simulating 10 frames. Note how in some cases up to 3 retransmissions are required.



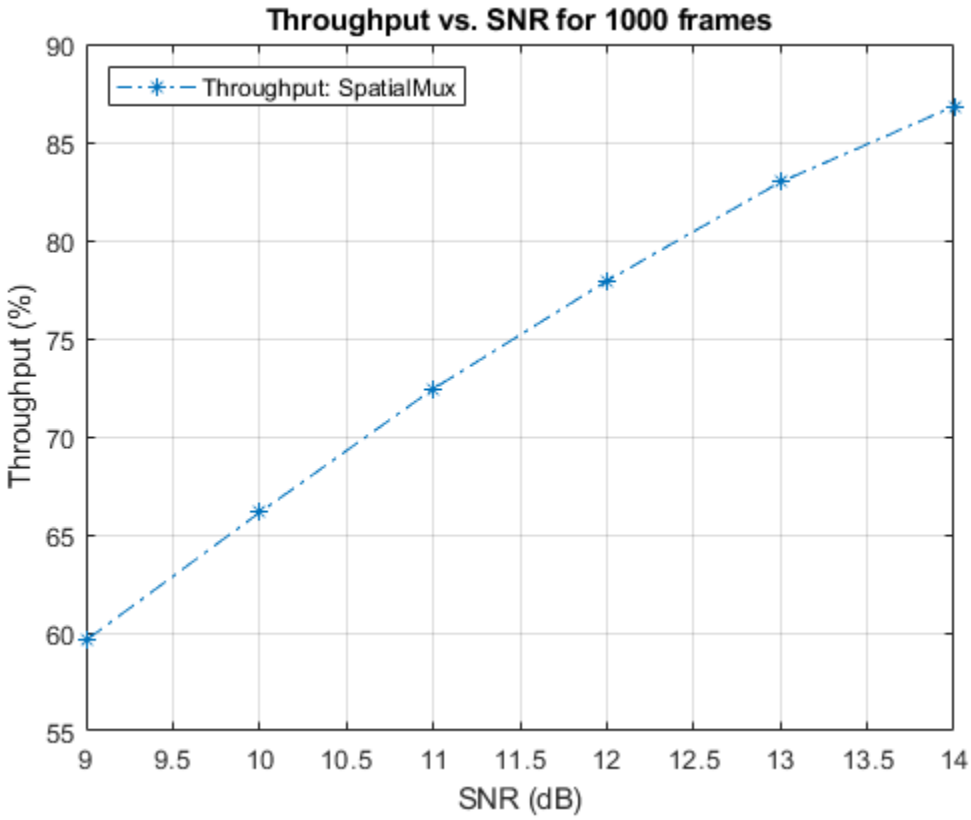
Throughput Results

The throughput results for the simulation are displayed in the MATLAB® command window after each SNR point is completed. They are also captured in `simThroughput` and `maxThroughput`. `simThroughput` is an array with the measured throughput in number of bits for all simulated SNR points. `maxThroughput` stores the maximum possible throughput in number of bits for each simulated SNR point.

```
% Plot throughput
figure
plot(SNRIn, simThroughput*100./maxThroughput,'*-.');
xlabel('SNR (dB)');
ylabel('Throughput (%)');
title('Throughput vs. SNR')
legend(legendString,'Location','NorthWest');
grid on;
```



The generated plot has been obtained with a low number of frames, therefore the results shown are not representative. A longer simulation obtained with 1000 frames produced the results shown below.



Appendix

This example uses these helper functions.

- hDisplayENBParameterSummary.m
- hHARQScheduling.m
- hNewHARQProcess.m
- hPlotRVSequence.m

Selected Bibliography

1 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception"

PDSCH Bit Error Rate Curve Generation

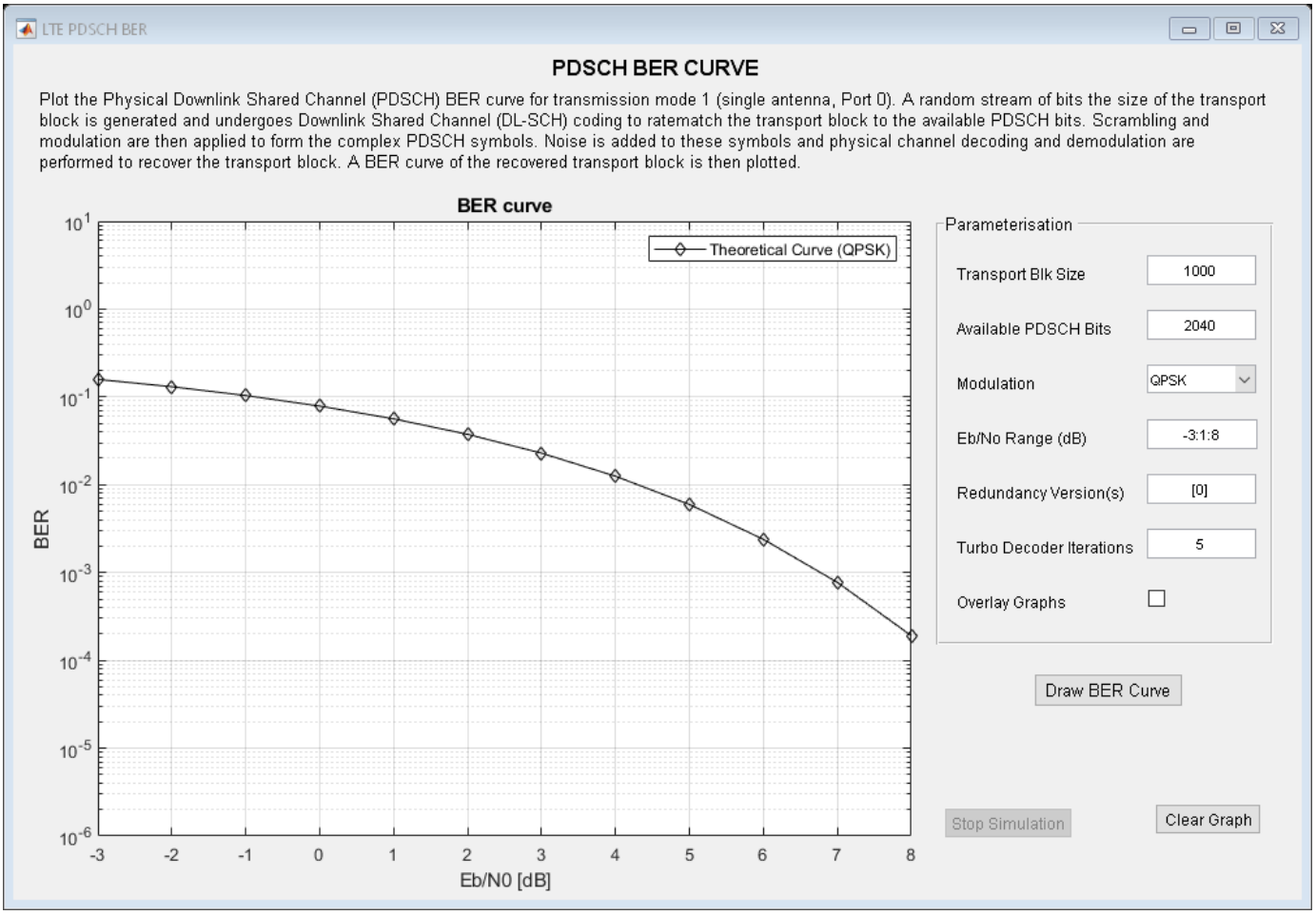
This example shows how the LTE Toolbox™ can be used to create Physical Downlink Shared Channel (PDSCH) Bit Error Rate (BER) curves under Additive White Gaussian Noise (AWGN) in a simple Graphical User Interface (GUI).

Introduction

hPDSCHBER.m provides a simple GUI to draw different BER curves for given SNR values and modulation schemes.

This example plots the Physical Downlink Shared Channel (PDSCH) BER curve for transmission mode 1 (single antenna, Port 0). A random stream of bits the size of the transport block is generated and undergoes Downlink Shared Channel (DL-SCH) coding to ratematch the transport block to the available PDSCH bits. Scrambling and modulation are then applied to form the complex PDSCH symbols. AWGN is added to these symbols after which channel decoding and demodulation are performed to recover the transport block. Using the recovered transport block a BER curve is plotted for a given range of SNR values.

```
hPDSCHBER; % Launch GUI
```



Parameters

The following GUI parameters are available:

- `TransportBlockSize` - Size of transport block
- `AvailablePDSCHBits` - Size of coded transport block after rate matching (codeword size)
- `Modulation` - Modulation scheme, one of {'QPSK', '16QAM', '64QAM', '256QAM'}
- `SNRRange` - Eb/No range in dB
- `RVSeq` - Redundancy version indicators sequence
- `NTurboDecIts` - Number of turbo decoder iteration cycles
- `OverlayGraphs` - Holds the previous graphs when checked, thus overlays new curve on previously drawn curves

Altering the various input parameters will affect the shape of the BER curve in different ways. The ratio of values assigned to the transport block size and available PDSCH bits should fit the range of target turbo code rates defined by LTE (1/3, 1/2, 3/4). Furthermore the value assigned to the available PDSCH bits are governed by the modulation scheme selected, e.g. for 16QAM a value which is a multiple of 4 must be chosen. Higher symbol modulation orders are more sensitive to noise interference and thus will suffer degradation in performance when compared to lower symbol modulation orders schemes at similar SNR values. The redundancy version must come from the range {0,1,2,3}. It can be a single value or a vector of values from the defined set.

The GUI also provides control over the configuration of the number of turbo decoder iteration cycles to be used in the decoder algorithm. This helps to perform an extended performance analysis of turbo decoder algorithm under AWGN. To compare the effect of altering the various parameters, all the curves can be plotted onto the same graph by checking the `OverlayGraphs` check box.

Appendix

This example uses the following helper functions:

- hPDSCHBER.m

PDSCH Throughput for Non-Codebook Based Precoding Schemes: Port 5 (TM7), Port 7 or 8 or Port 7-8 (TM8), Port 7-14 (TM9 and TM10)

This example demonstrates how to measure the Physical Downlink Shared Channel (PDSCH) throughput performance using LTE Toolbox™ for the following non-codebook based precoding transmission modes (TM):

- TM7: non-codebook based precoding for single layer (Port 5)
- TM8: non-codebook based precoding for up to two layers (dual layer Port 7-8, or single antenna port, port 7 or 8)
- TM9 and TM10: non-codebook based precoding for up to eight layers Port 7-14 or single antenna port, port 7 or 8

FDD and TDD duplexing schemes are supported. The example also shows how to parameterize and customize the settings for the different transmission modes. It also supports the use of Parallel Computing Toolbox™ to reduce effective simulation time.

Introduction

This example measures throughput for a number of SNR points. The provided code can operate under a number of transmission modes: TM7, TM8, TM9 and TM10. FDD and TDD duplexing schemes are supported. For information on how to model TM1, TM2, TM3, TM4 and TM6 check the following example: “PDSCH Throughput Conformance Test for Single Antenna (TM1), Transmit Diversity (TM2), Open Loop (TM3) and Closed Loop (TM4/6) Spatial Multiplexing” on page 2-404

Since non-codebook based TMs are considered, the precoding (beamforming) matrix must be calculated. This is achieved using a singular value decomposition (SVD) approach. Perfect knowledge of the channel is assumed.

For each of the considered SNR points, operating on a subframe by subframe basis:

- The precoding (beamforming) matrix is calculated from a perfect channel estimate.
- A populated resource grid is generated and OFDM modulated to create the waveform to transmit.
- The generated waveform is passed through a noisy fading channel.
- Receiver operations (channel estimation, equalization, demodulation and decoding) are performed.
- The throughput performance of the PDSCH is determined using the block CRC result at the output of the channel decoder.

A `parfor` loop can be used instead of the `for` loop for the SNR calculation. This is indicated within the example. The `parfor` statement is part of the Parallel Computing Toolbox and executes the SNR loops in parallel to reduce the total simulation time.

Simulation Configuration

The example is executed for a simulation length of `NFrames` = 2 frames for a number of SNR points. A large number of `NFrames` should be used to produce meaningful throughput results. `SNRIn` can be an array of values or a scalar. Some TMs and certain modulation schemes are more robust to noise

and channel impairments than others, therefore different values of SNR may have to be used for different parameter sets.

```
NFrames = 2;        % Number of frames

SNRIn = [-4 1];     % SNR range in dB
```

A set of common parameters for all TMs is initially specified, these include the bandwidth, the desired code rate, the modulation scheme and the allocated set of resource blocks. Not specifying an RMC number ensures that all downlink subframes are scheduled. If RMC is specified (e.g. 'R.0'), the subframe scheduling is as defined in TS 36.101 [1].

The variable `txMode` selects the TM via a switch statement. For each TM, the required parameters are specified. This example does not perform DCI format decoding, so the `DCIFormat` field is not strictly necessary. However, since the DCI format is closely linked to the TM, it is included for completeness.

For the purpose of this example, TM9 and TM10 are the same, except for the DCI format, however, they are separated in the switch statement for completeness.

Once the basic set of parameters have been specified, a call to `lteRMCDL` is needed to fully populate the downlink parameter structure `enb`.

```
simulationParameters = []; % clear simulationParameters
simulationParameters.NDLRB = 50;
simulationParameters.PDSCH.TargetCodeRate = 0.5;
simulationParameters.PDSCH.Modulation = {'16QAM'};
simulationParameters.PDSCH.PRBSet = (0:9)';

txMode = 'TM8';     % TM7, TM8, TM9, TM10
DuplexMode = 'TDD'; % 'FDD', 'TDD'

switch txMode
    case 'TM7' % single layer (Port 5)
        simulationParameters.PDSCH.DCIFormat = 'Format1';
        simulationParameters.PDSCH.TxScheme = 'Port5';
        simulationParameters.PDSCH.NLayers = 1;
        ntxants = 4;
    case 'TM8' % up to two layers (dual layer Port 7-8, or single antenna
               % port, port 7 or 8)
        simulationParameters.PDSCH.DCIFormat = 'Format2B';
        simulationParameters.PDSCH.TxScheme = 'Port7-8'; %'Port7-8','Port8'
        simulationParameters.PDSCH.NLayers = 2;
        ntxants = 4;
        simulationParameters.PDSCH.NSCID = 0;
    case 'TM9' % up to eight layers (up to eight layers Port 7-14 or single
               % antenna port, port 7 or 8)
        simulationParameters.PDSCH.DCIFormat = 'Format2C';
        simulationParameters.PDSCH.TxScheme = 'Port7-14';
        simulationParameters.PDSCH.NLayers = 4;
        ntxants = 8;
        simulationParameters.CSIRefP = ntxants;
    case 'TM10' % up to eight layers (up to eight layers Port 7-14 or
                % single antenna port, port 7 or 8)
        simulationParameters.PDSCH.DCIFormat = 'Format2D';
        simulationParameters.PDSCH.TxScheme = 'Port7-14';
        simulationParameters.PDSCH.NLayers = 4;
```

```
        ntxants = 8;
        simulationParameters.CSIRefP = ntxants;
    otherwise
        error('Transmission mode should be one of TM7, TM8, TM9 or TM10.')
end

% For 2 codewords (more than 4 layers), the modulation field needs two values, one per codeword
if (simulationParameters.PDSCH.NLayers > 4) && (length(simulationParameters.PDSCH.Modulation)<2)
    simulationParameters.PDSCH.Modulation = repmat(simulationParameters.PDSCH.Modulation,1,2);
end
```

The size of the precoding matrix `W` (beamforming matrix) is `NLayers` -by- `ntxants`. The number of transmit antennas is taken from `W` in subsequent toolbox function calls. The number of receive antennas is specified later on as part of the propagation channel.

```
% Initialize W to zero
simulationParameters.PDSCH.W = zeros(simulationParameters.PDSCH.NLayers, ntxants);
```

Set the duplexing mode and specify the number of subframes to 1. The code generates one subframe at a time repeatedly until a total of `NFrames` are generated.

```
simulationParameters.DuplexMode = DuplexMode;
simulationParameters.TotSubframes = 1;
```

Call the `lteRMCDL` function to generate the default eNodeB parameters not specified in `simulationParameters`. These will be required later to generate the waveform using `lteRMCDLTool`.

```
enb = lteRMCDL(simulationParameters);
```

The output `enb` structure contains, amongst other fields, the transport block sizes and redundancy version sequence for each codeword subframe within a frame. These are used later in the simulation.

```
trBlkSizes = enb.PDSCH.TrBlkSizes;
rvSequence = enb.PDSCH.RVSeq;
```

The number of codewords, `ncw`, is the number of entries in the `enb.PDSCH.Modulation` field.

```
ncw = length(string(enb.PDSCH.Modulation));
```

Print a summary of some of the more relevant simulation parameters. Confirm the parameter settings align with expected values. The reported code rate displayed can be useful to detect problems when manually specifying the transport block sizes. Typical values are 1/3, 1/2 and 3/4.

```
hDisplayENBParameterSummary(enb, txMode);
```

```
-- Parameter summary: -------------------------------------------
                     Duplexing mode: TDD
                  Transmission mode: TM8
                Transmission scheme: Port7-8
  Number of downlink resource blocks: 50
 Number of allocated resource blocks: 10
Cell-specific reference signal ports: 1
         Number of transmit antennas: 4
                 Transmission layers: 2
                 Number of codewords: 1
               Modulation codeword 1: 16QAM
```

```
    Transport block sizes codeword 1:     4584    4008       0       0    4584    4584    4008
                 Code rate codeword 1:   0.5053  0.4941       0       0  0.5053  0.5053  0.4941
----------------------------------------------------------------------
```

Propagation Channel and Channel Estimator Configuration

The structure `channel` contains the channel model configuration parameters.

```
channel = struct;                   % Channel config structure
channel.Seed = 6;                   % Channel seed
channel.NRxAnts = 8;                % Number of receive antennas
channel.DelayProfile ='EPA';        % Delay profile
channel.DopplerFreq = 5;            % Doppler frequency
channel.MIMOCorrelation = 'Low';    % Multi-antenna correlation
channel.NTerms = 16;                % Oscillators used in fading model
channel.ModelType = 'GMEDS';        % Rayleigh fading model type
channel.InitPhase = 'Random';       % Random initial phases
channel.NormalizePathGains = 'On';  % Normalize delay profile power
channel.NormalizeTxAnts = 'On';     % Normalize for transmit antennas
% The channel sampling rate depends on the FFT size used in the OFDM
% modulator. This can be obtained using the function lteOFDMInfo.
ofdmInfo = lteOFDMInfo(enb);
channel.SamplingRate = ofdmInfo.SamplingRate;

% Maximum channel delay (multipath and channel filtering)
channel.InitTime = 0;
[~,chInfo] = lteFadingChannel(channel,0); % get channel info
maxChDelay = ceil(max(chInfo.PathSampleDelays)) + chInfo.ChannelFilterDelay;
```

Define the channel estimator configuration structure. Since non-codebook based TMs are used, set the DMRS as reference signal for channel estimation.

```
cec = struct;                       % Channel estimation config structure
cec.PilotAverage = 'UserDefined';   % Type of pilot symbol averaging
cec.FreqWindow = 41;                % Frequency window size
cec.TimeWindow = 27;                % Time window size
cec.InterpType = 'Cubic';           % 2D interpolation type
cec.InterpWindow = 'Centered';      % Interpolation window type
cec.InterpWinSize = 1;              % Interpolation window size
cec.Reference = 'DMRS';             % Use DMRS as reference signal
```

Display Simulation Information

The variable `displaySimulationInformation` controls the display of simulation information such as the HARQ process ID used for each subframe. In case of CRC error, the value of the index to the RV sequence is also displayed.

```
displaySimulationInformation = true;
```

Processing Chain

To determine the throughput at each SNR point, the subframe by subframe PDSCH processing chain includes:

- *Calculating the Precoding Matrix* - A perfect channel estimate is used to calculate the precoding matrix. The calculation process differs slightly between FDD and TDD. A detailed explanation of this step is provided in the next section.

- *Updating Current HARQ Process* - The HARQ process either carries new transport data or a retransmission of previously sent transport data depending upon the Acknowledgment (ACK) or Negative Acknowledgment (NACK) based on CRC results. All this is handled by the HARQ scheduler, hHARQScheduling.m. The PDSCH data is updated based on the HARQ state.
- *Creating Transmit Waveform* - Pass the data generated by the HARQ process to the `lteRMCDLTool` function to produce an OFDM modulated waveform, containing the physical channels and signals.
- *Noisy Channel Modeling* - Pass the waveform through a fading channel and add noise (AWGN).
- *Performing Synchronization and OFDM Demodulation* - Offset the received symbols to account for a combination of implementation delay and channel delay spread. OFDM demodulate the symbols.
- *Performing Channel Estimation* - Estimate the channel response and noise level. Use these estimates to decode the PDSCH.
- *Decoding the PDSCH* - Obtain an estimate of the received codewords using `ltePDSCHDecode` to demodulate and descramble the recovered PDSCH symbols for all transmit and receive antenna pairs.
- *Decoding the Downlink Shared Channel (DL-SCH) and Storing the Block CRC Error for a HARQ Process* - Pass the vector of decoded soft bits to `lteDLSCHDecode`, which decodes the codeword and returns the block CRC error used to determine the throughput of the system. The contents of the new soft buffer, `harqProc(harqID).decState`, is available at the output of this function to be used when decoding the next subframe.

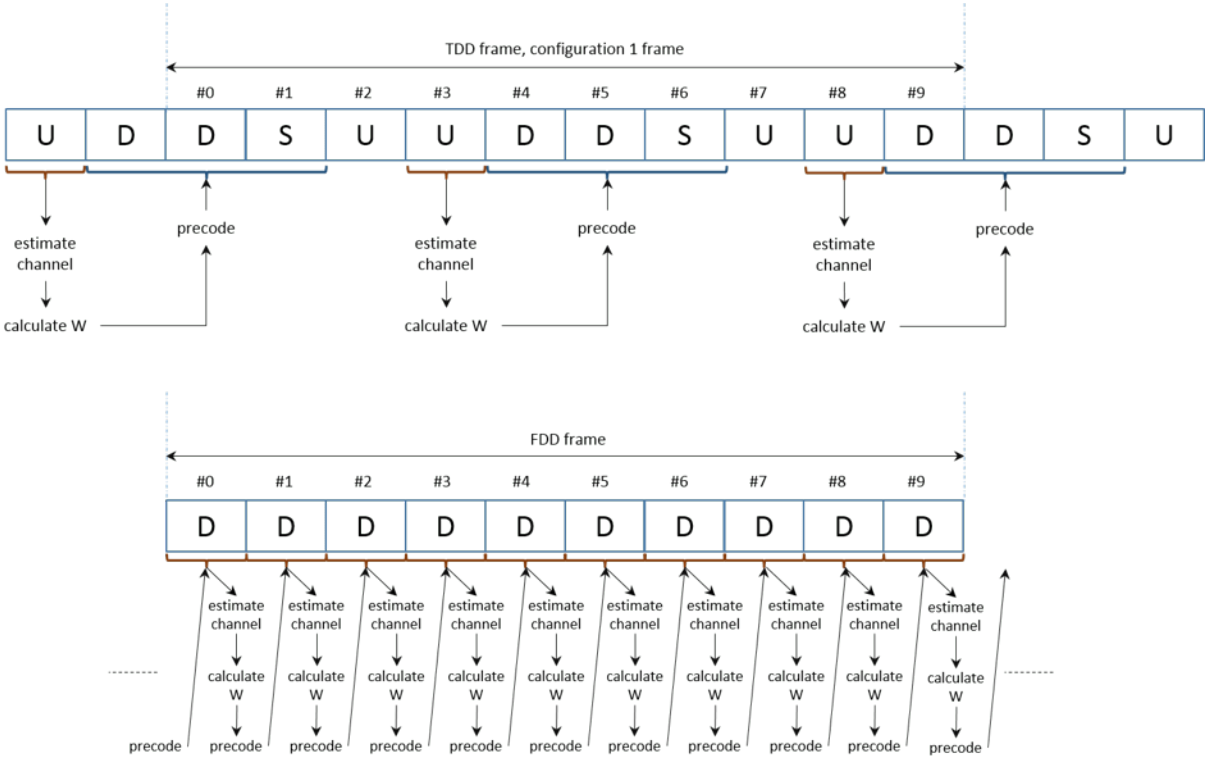
Precoding Matrix Calculation

Since non-codebook based TMs are modeled, the precoding (beamforming) matrix must be computed, which is based on the channel estimate. For simplicity, perfect knowledge of the channel is assumed.

The figure shows the timing associated with the channel estimation, precoding matrix `W` calculation and precoding operation for both FDD and TDD modes.

For TDD (uplink-downlink configuration 1 frame structure), the channel is estimated in the last UL subframe before a DL subframe. This channel estimate is used to calculate the precoding matrix `W`. All subsequent DL subframes (including special subframes) until the next UL subframe are precoded with this matrix `W`.

For FDD, there is a delay of one subframe between the calculation of the precoding matrix `W` and the subframe where it is used. For example, the precoding matrix used in subframe `n` has been calculated with the channel estimate obtained in subframe `n-1`, as shown in the figure.

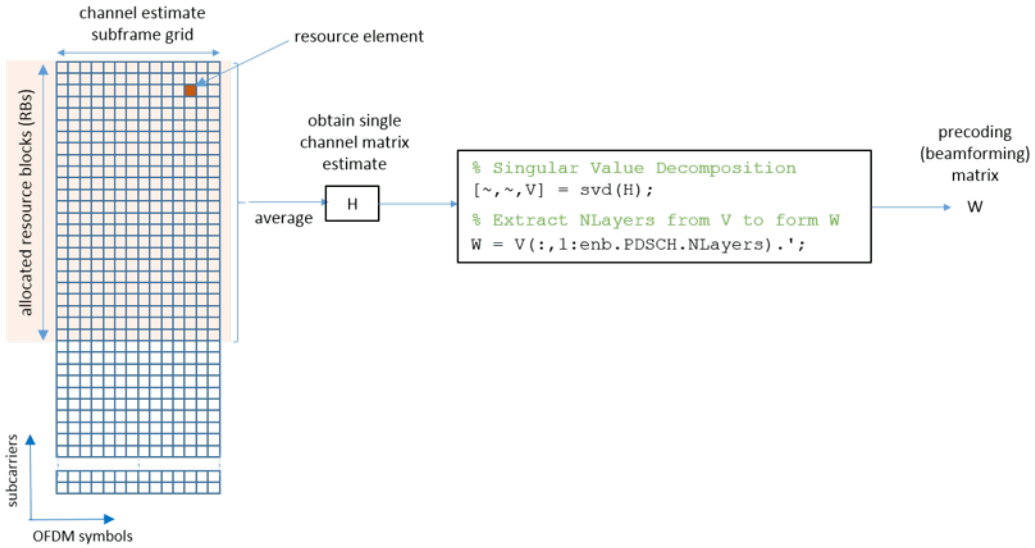


The function hCalculatePrecodingMatrix calculates `W` by:

- Obtaining a perfect channel estimate for the considered subframe
- Averaging the channel estimates for all the allocated RBs
- Calculating the singular value decomposition and extracting the first `NLayers` columns to obtain the precoding vector.

If desired, a different calculation or value of `W` can be used here instead of the call to hCalculatePrecodingMatrix.m.

This figure shows the channel estimation averaging process and `W` calculation.



The process depicted shows how a single channel matrix estimate, `H`, is obtained from the whole set of allocated resource blocks by averaging over time and frequency. Using this channel estimate, `H`, a precoding matrix, `W`, is obtained via singular value decomposition.

Note that for an allocation of a single resource block, the precoding matrix `W` will usually be well matched to the channel conditions, with little deviation from the optimal precoding. But as the allocation size increases, the precoding matrix takes into account the average of the channel conditions over the whole allocation. This averaging causes a deviation from the optimal precoding matrix. Therefore, you can expect a degradation in performance as the size of the resource allocation increases.

Processing Loop

The 'for' loop for SNR points processing is included below. To enable the use of parallel computing for increased speed use 'parfor' instead of 'for' in the loop. This requires the Parallel Computing Toolbox. If this is not installed 'parfor' will default to the normal 'for' statement. If 'parfor' is used, it is recommended that the variable `displaySimulationInformation` above is set to false, otherwise the simulation information displays for each SNR point will overlap.

```
% Initialize variables used in the simulation and analysis
% Array to store the maximum throughput for all SNR points
maxThroughput = zeros(length(SNRIn),1);
% Array to store the simulation throughput for all SNR points
simThroughput = zeros(length(SNRIn),1);
% Array to store RV sequence index history
allRvSeqIdxHistory = cell(1,numel(SNRIn));
% Copy of channel structure to optimize parallel processing (only if
% running the example with Parallel Computing Toolbox)
channelInit = channel;
% During simulation some fields of enb will be updated, make a copy to
% reinitialize it when simulating each SNR point
enbInit = enb;

% For TDD precalculate vector of subframe types: D, S and U for downlink,
% special and uplink respectively
if strcmpi(DuplexMode,'TDD')
```

```
    subframeType = char(10,1);
    initialSubframeNo = enb.NSubframe;
    for sNo=0:9 % for all subframes in a frame
        enb.NSubframe = sNo;
        duplexInfo = lteDuplexingInfo(enb);
        subframeType(sNo+1) = duplexInfo.SubframeType(1); % first char: D, S or U
    end
    enb.NSubframe = initialSubframeNo;
end

% CFI can be a scalar or a vector, create a local copy of CFI as a vector
% (one value per subframe)
if numel(enb.CFI) == 1
    CFI = repmat(enb.CFI,1,10);
else
    CFI = enb.CFI;
end

% Get the HARQ ID sequence for HARQ processing. This is a list of indices
% for HARQ process scheduling.
[~,~,enbOut] = lteRMCDLTool(enb, []);
harqProcessSequenceInit = enbOut.PDSCH.HARQProcessSequence;

for snrIdx = 1:numel(SNRIn)     % comment out for parallel computing
%parfor snrIdx = 1:numel(SNRIn) % uncomment for parallel computing

    % Set the random number generator seed depending on the loop variable
    % to ensure independent random streams
    rng(snrIdx,'combRecursive');

    % Reinitialize variables (they are modified during each SNR point
    % simulation)
    enb = enbInit;
    channel = channelInit;
    harqProcessSequence = harqProcessSequenceInit;

    % Initialize the state of all HARQ processes
    harqProcesses = hNewHARQProcess(enb);

    % Set up variables for the main loop
    lastOffset = 7;       % Initialize overall frame timing offset
                          % (set equal to channel implementation delay)
    frameOffset = 7;      % Initialize frame timing offset
                          % (set equal to channel implementation delay)
    blkCRC = [];          % Block CRC for all considered subframes
    bitTput = [];         % Number of successfully received bits per subframe
    txedTrBlkSizes = [];  % Number of transmitted bits per subframe
    precodingMatrix = []; % Precoding matrix
    rxSymbols = [];       % DL-SCH symbols for constellation plot (codeword 1)
    rxSymbols2 = [];      % DL-SCH symbols for constellation plot (codeword 2)

    % The variable rvSeqIdxHistory stores the history of the value of the
    % RV index sequence for all the HARQ processes
    rvSeqIdxHistory = nan(ncw, NFrames*10);

    % Flag to indicate if a precoding matrix W is available. This is useful
    % if no W has been calculated at the beginning of the simulation.
    isWready = false;
```

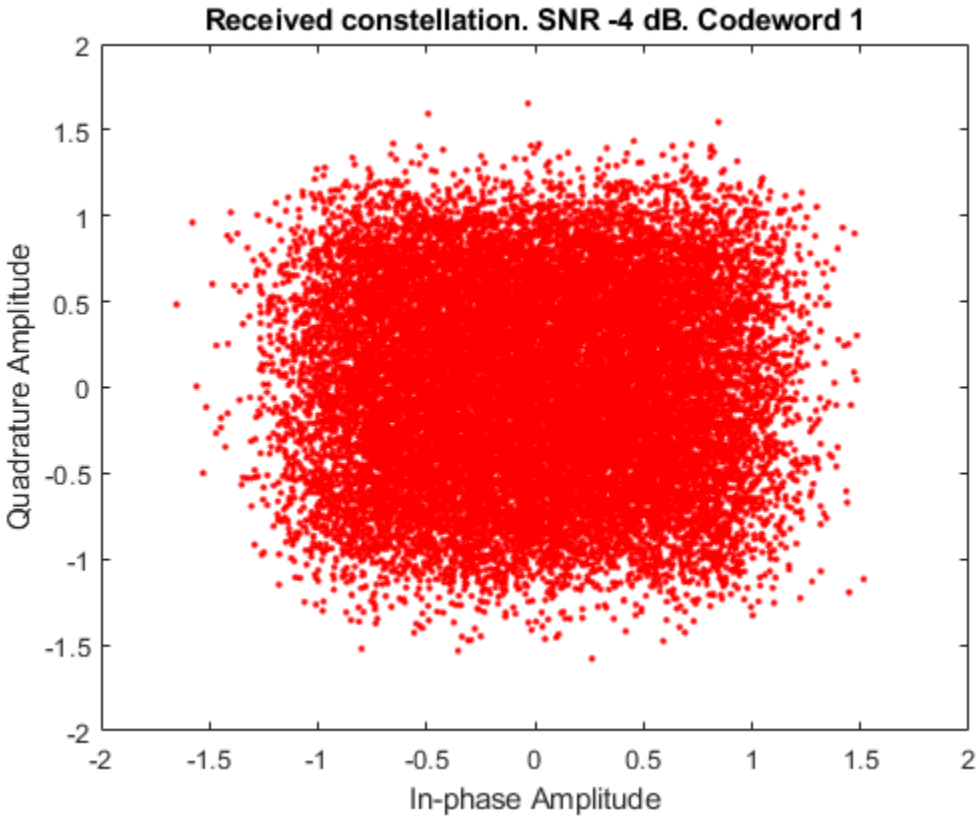
```
% Flag to indicate is a subframe is to be processed. Set to true if
% there is data to be processed in the subframe, i.e. DL subframe or
% non-zero transport block size.
processSubframe = false;

% Main for loop: for all subframes
for subframeNo = 0:(NFrames*10-1)

    % Update subframe number
    enb.NSubframe = subframeNo;

    % Load CFI for current subframe
    enb.CFI = CFI(mod(subframeNo,length(CFI))+1);

    % Channel time for the current subframe
    channel.InitTime = subframeNo/1000;

    % Get HARQ process ID for the subframe from HARQ process sequence
    harqID = harqProcessSequence(mod(subframeNo, length(harqProcessSequence))+1);

    % Extract the current subframe transport block size(s)
    trBlk = trBlkSizes(:, mod(subframeNo, 10)+1).';

    % Load precoding matrix calculated in previous subframe if it
    % exists and if there is data to transmit. Set the flag to trigger
    % subframe processing.
    if isWready && any(trBlk)
        harqProcesses(harqID).txConfig.W = precodingMatrix;
        processSubframe = true;
    else
        processSubframe = false;
    end

    % Precoding matrix calculation
    if strcmpi(DuplexMode,'TDD')
        % Estimate channel in UL subframe
        if strcmp(subframeType(mod(subframeNo,10)+1),'U')
            processSubframe = false; % UL subframe, no DL data
            % Only perform channel estimate if next subframe is DL
            if strcmp(subframeType(mod((subframeNo+1),10)+1),'D')
                precodingMatrix = hCalculatePrecodingMatrix(enb, channel);
                isWready = true;
            end
        end
    else %FDD
        % Get transport block for next subframe
        trBlkNext = trBlkSizes(:, mod(subframeNo+1, 10)+1).';

        % Calculate the precoding matrix for next subframe only if it
        % carries data (i.e. non-zero trBlkNext)
        if any(trBlkNext)
            precodingMatrix = hCalculatePrecodingMatrix(enb, channel);
            isWready = true;
        else
            isWready = false;
        end
    end
```

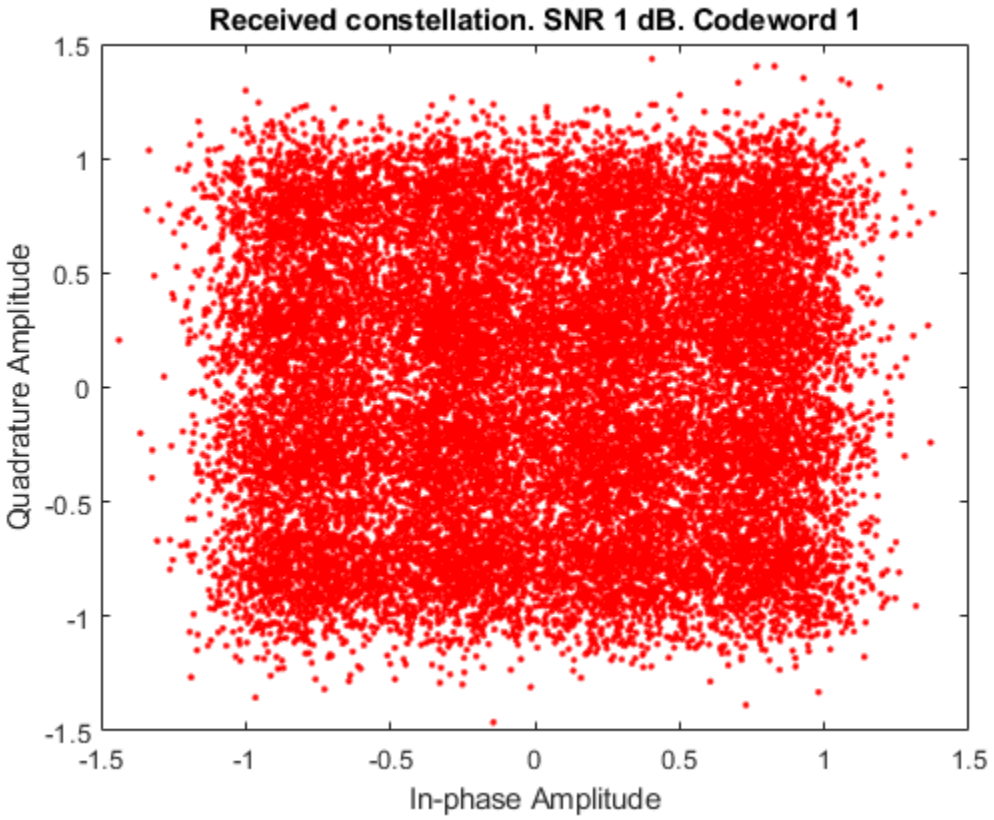
```
% Subframe processing
if processSubframe

    % Update current HARQ process
    harqProcesses(harqID) = hHARQScheduling( ...
        harqProcesses(harqID), subframeNo, rvSequence);

    % Display run time information
    if displaySimulationInformation && any(trBlk)
        disp(' ');
        disp(['Subframe: ' num2str(subframeNo) '. HARQ process index: ' num2str(harqID)])
    end

    % Update RV sequence index table
    rvSeqIdxHistory(:,subframeNo+1) = ...
        harqProcesses(harqID).txConfig.RVIdx.';

    % Update the PDSCH transmission configuration with HARQ
    % process state
    enb.PDSCH = harqProcesses(harqID).txConfig;

    % Cell payload
    dlschTransportBlk = harqProcesses(harqID).data;

    % Create transmit waveform
    txWaveform = lteRMCDLTool(enb, dlschTransportBlk);

    % Add maxChDelay sample padding. This is to cover the range of delays
    % expected from channel modeling (a combination of
    % implementation delay and channel delay spread)
    txWaveform =  [txWaveform; zeros(maxChDelay, ntxants)]; %#ok<AGROW>

    % Pass data through channel model
    rxNoiselessWaveform = lteFadingChannel(channel,txWaveform);

    % Calculate noise gain including compensation for downlink
    % power allocation
    SNR = 10^((SNRIn(snrIdx)-enb.PDSCH.Rho)/20);

    % Normalize noise power to take account of sampling rate,
    % which is a function of the IFFT size used in OFDM
    % modulation, and the number of antennas
    N0 = 1/(sqrt(2.0*ntxants*double(ofdmInfo.Nfft))*SNR);

    % Create additive white Gaussian noise
    noise = N0*complex(randn(size(rxNoiselessWaveform)),...
        randn(size(rxNoiselessWaveform)));

    % Add AWGN to the received time domain waveform and scale
    % for required power
    rxWaveform = rxNoiselessWaveform + noise;

    % Receiver
    % Once every frame, on subframe 0, calculate a new
    % synchronization offset. An offset within the range of
    % delays expected from the channel modeling (a combination
    % of implementation delay and channel delay spread)
    % indicates success
```

```
if (mod(subframeNo,10)==0)
    frameOffset = lteDLFrameOffset(enb,rxWaveform);
    if (frameOffset > maxChDelay)
        frameOffset = lastOffset;
    end
    lastOffset = frameOffset;
end
rxWaveform = rxWaveform(1+frameOffset:end,:);

% Perform OFDM demodulation on the received data to obtain
% the resource grid
rxSubframe = lteOFDMDemodulate(enb,rxWaveform);

% Channel estimation
[estChannelGrid,noiseEst] = ...
    lteDLChannelEstimate(enb,enb.PDSCH,cec,rxSubframe);

% Perform equalization, deprecoding, layer demapping,
% demodulation and descrambling on the received data using the
% channel estimate.

% Get PDSCH indices
pdschIndices = ltePDSCHIndices(enb,enb.PDSCH,enb.PDSCH.PRBSet);

% Get PDSCH resource elements. Scale the received subframe
% by the PDSCH power factor Rho. The PDSCH is scaled by
% Rho, while the cell reference symbols used for channel
% estimation (used in the PDSCH decoding stage) are not.
[pdschRx, pdschHest] = lteExtractResources(pdschIndices, ...
    rxSubframe*(10^(-enb.PDSCH.Rho/20)),estChannelGrid);

% Decode PDSCH
[dlschBits,dlschSymbols] = ltePDSCHDecode(enb,enb.PDSCH,...
    pdschRx,pdschHest,noiseEst);

% Store the decoded DLSCH symbols for constellation
% plotting
rxSymbols = [rxSymbols; dlschSymbols{1}(:)]; %#ok<AGROW>
if ncw>1
    rxSymbols2 = [rxSymbols2; dlschSymbols{2}(:)]; %#ok<AGROW>
end

% Decode the DL-SCH
[decbits,harqProcesses(harqID).blkerr,harqProcesses(harqID).decState] = ...
    lteDLSCHDecode(enb,enb.PDSCH,trBlk,dlschBits, ...
    harqProcesses(harqID).decState);

% Display block errors
if displaySimulationInformation
    if any(harqProcesses(harqID).blkerr)
        disp(['Block error. RV index: ' num2str(harqProcesses(harqID).txConfig.RVIdx)
    else
        disp(['No error. RV index: ' num2str(harqProcesses(harqID).txConfig.RVIdx) '
    end
end

% Store values needed to calculate throughput
% Only for subframes with data
```

```
            if(any(trBlk))
                blkCRC = [blkCRC harqProcesses(harqID).blkerr]; %#ok<AGROW>
                bitTput = [bitTput trBlk.*(1- ...
                    harqProcesses(harqID).blkerr)]; %#ok<AGROW>
                txedTrBlkSizes = [txedTrBlkSizes trBlk]; %#ok<AGROW>
            end
        end
    end

    % Plot received symbol constellation
    if displaySimulationInformation
        figure;plot(rxSymbols,'.r'); title(['Received constellation. SNR ' ...
                                num2str(SNRIn(snrIdx)) ' dB. Codeword 1'])
        xlabel('In-phase Amplitude');
        ylabel('Quadrature Amplitude');
        if ncw>1
            figure;plot(rxSymbols2,'.r'); title(['Received constellation. SNR ' ...
                                num2str(SNRIn(snrIdx)) ' dB. Codeword 2'])
            xlabel('In-phase Amplitude');
            ylabel('Quadrature Amplitude');
        end
    end

    % Calculate the maximum and simulated throughput
    maxThroughput(snrIdx) = sum(txedTrBlkSizes); % Max possible throughput
    simThroughput(snrIdx) = sum(bitTput,2);      % Simulated throughput

    % Display the results dynamically in the command window
    fprintf('\nSNR = %.2f dB. Throughput for %d Frame(s) = %.4f Mbps\n',...
         SNRIn(snrIdx),NFrames,1e-6*simThroughput(snrIdx)/(NFrames*10e-3));
    fprintf('SNR = %.2f dB. Throughput(%%) for %d Frame(s) = %.4f %%\n',...
    SNRIn(snrIdx),NFrames,simThroughput(snrIdx)*100/maxThroughput(snrIdx));

    allRvSeqIdxHistory{snrIdx} = rvSeqIdxHistory;

end


Subframe: 4. HARQ process index: 3
Block error. RV index: 1, CRC: 1

Subframe: 5. HARQ process index: 4
Block error. RV index: 1, CRC: 1

Subframe: 6. HARQ process index: 5
Block error. RV index: 1, CRC: 1

Subframe: 9. HARQ process index: 6
Block error. RV index: 1, CRC: 1

Subframe: 10. HARQ process index: 7
Block error. RV index: 1, CRC: 1

Subframe: 11. HARQ process index: 2
Block error. RV index: 1, CRC: 1

Subframe: 14. HARQ process index: 1
Block error. RV index: 1, CRC: 1
```

```
Subframe: 15. HARQ process index: 3
No error. RV index: 2, CRC: 0

Subframe: 16. HARQ process index: 5
No error. RV index: 2, CRC: 0

Subframe: 19. HARQ process index: 4
No error. RV index: 2, CRC: 0

SNR = -4.00 dB. Throughput for 2 Frame(s) = 0.6588 Mbps
SNR = -4.00 dB. Throughput(%) for 2 Frame(s) = 29.8694 %

Subframe: 4. HARQ process index: 3
No error. RV index: 1, CRC: 0

Subframe: 5. HARQ process index: 4
No error. RV index: 1, CRC: 0

Subframe: 6. HARQ process index: 5
No error. RV index: 1, CRC: 0

Subframe: 9. HARQ process index: 6
No error. RV index: 1, CRC: 0

Subframe: 10. HARQ process index: 7
No error. RV index: 1, CRC: 0

Subframe: 11. HARQ process index: 2
No error. RV index: 1, CRC: 0

Subframe: 14. HARQ process index: 1
No error. RV index: 1, CRC: 0

Subframe: 15. HARQ process index: 3
No error. RV index: 1, CRC: 0

Subframe: 16. HARQ process index: 5
No error. RV index: 1, CRC: 0

Subframe: 19. HARQ process index: 4
No error. RV index: 1, CRC: 0

SNR = 1.00 dB. Throughput for 2 Frame(s) = 2.2056 Mbps
SNR = 1.00 dB. Throughput(%) for 2 Frame(s) = 100.0000 %
```


Received constellation. SNR -4 dB. Codeword 1
Quadrature Amplitude
In-phase Amplitude
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

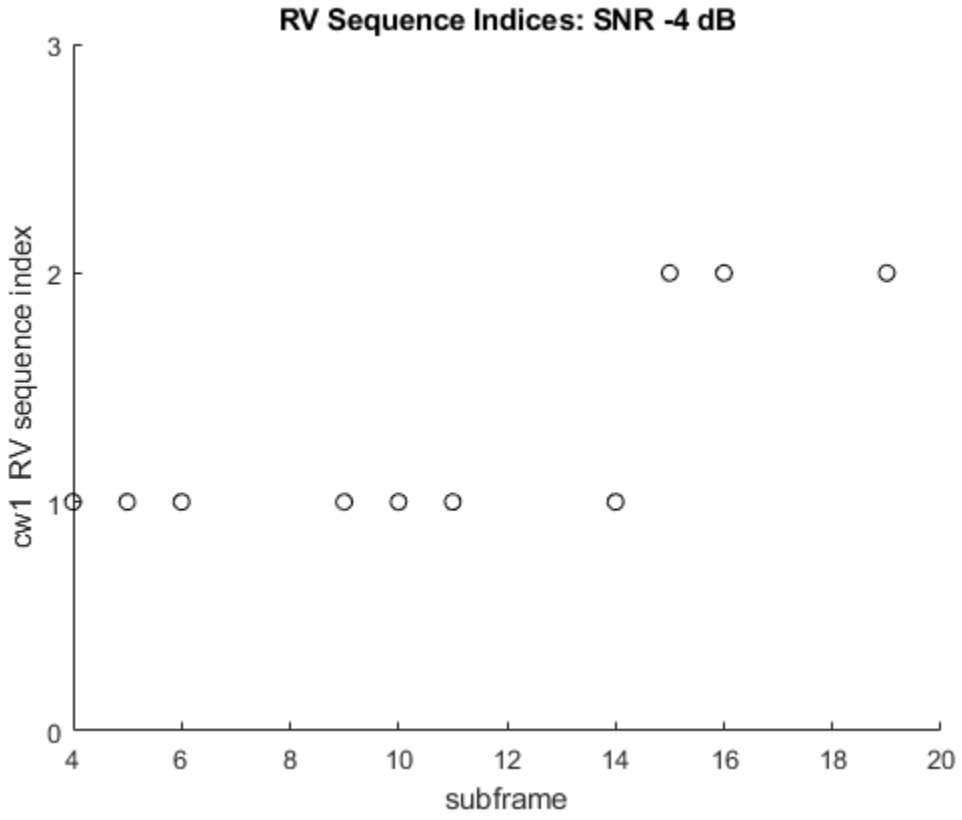


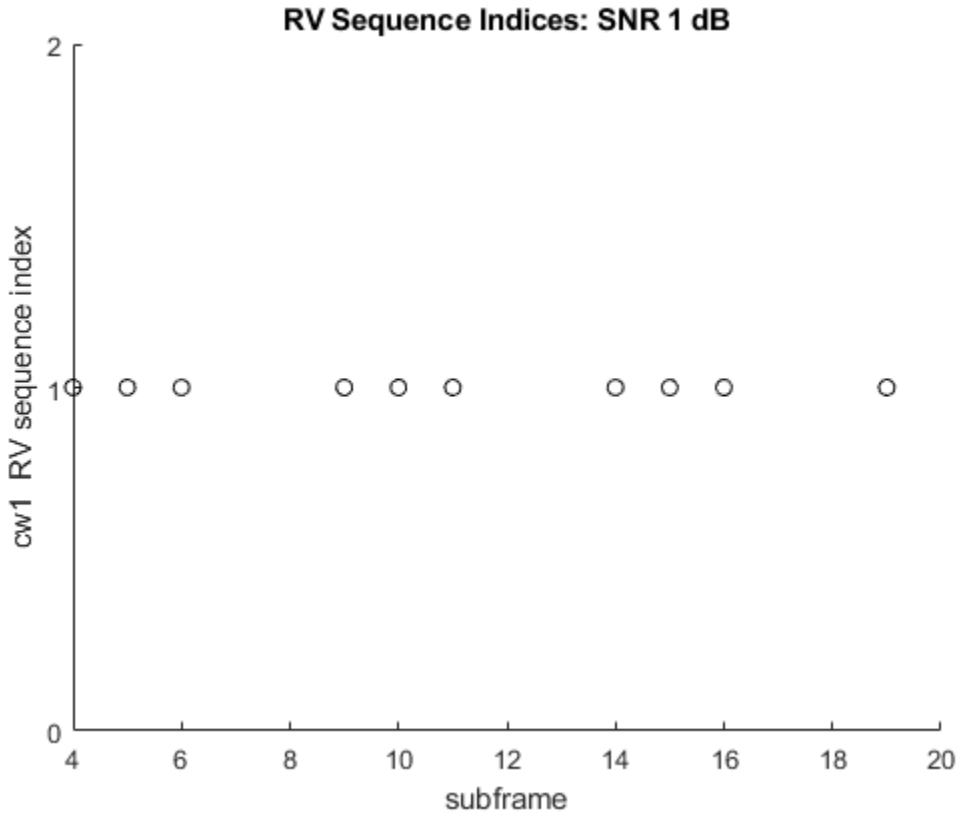
RV Sequence Index Plots

The code below generates plots with the value of the indices to the elements in the RV sequence for the simulated subframes. This provides an idea of the retransmissions required. The reason for plotting the indices instead of the RV values is that the latter may not be organized in ascending order. For example, in some cases the RV sequence can be [0, 2, 3, 1]. Plotting these values as they are used will not provide a clear idea of the number of retransmissions needed.

When transmitting a new transport block, the first element of the RV sequence is used, and a value of 1 is shown for that subframe. This is the case at the beginning of the simulation. If a retransmission is required, the next element in the RV sequence is selected and the index is increased. A value of 2 will be plotted for the subframe where the retransmission takes place. If further retransmissions are required, the index value will increase further. The plots do not show any value in subframe 5 of consecutive frames. This is because no data is transmitted in those subframes for the setup used in this example.

```
hPlotRVSequence(SNRIn,allRvSeqIdxHistory,NFrames);
```

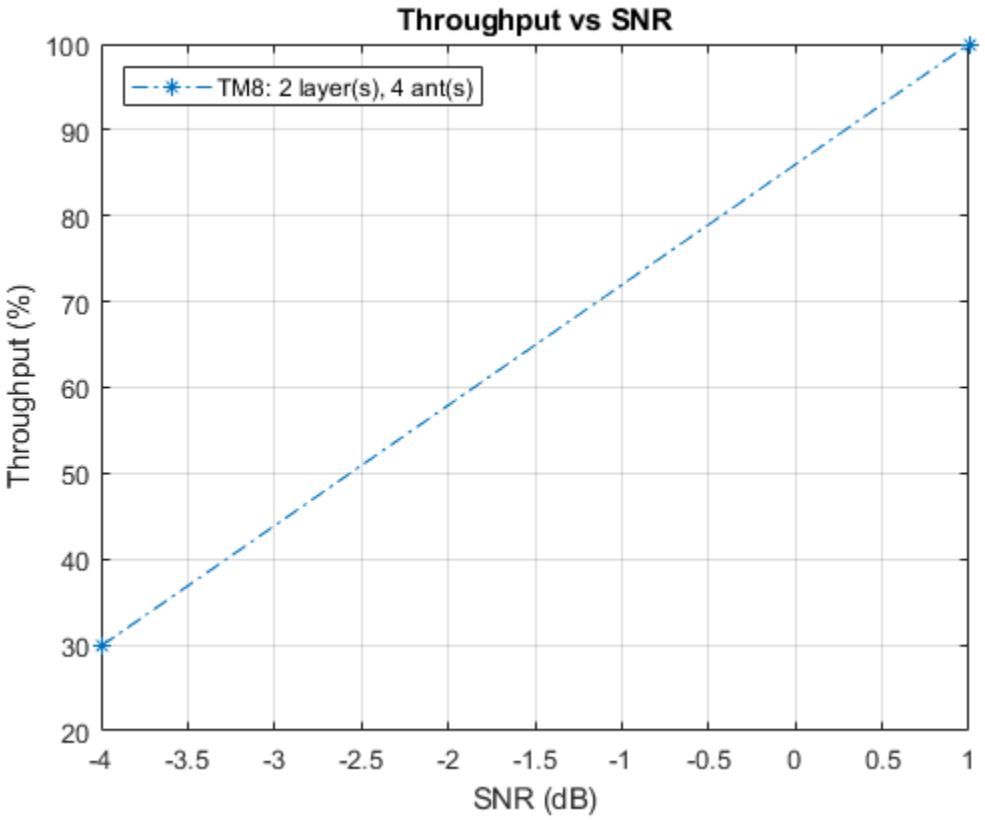

RV Sequence Indices: SNR -4 dB
cw1 RV sequence index
subframe
0
1
2
3
4
6
8
10
12
14
16
18
20



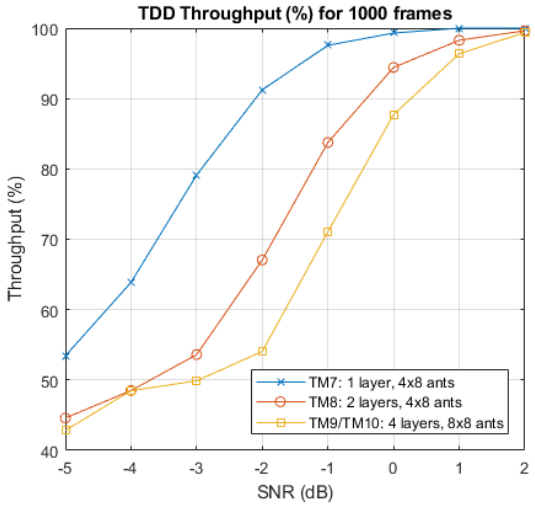
Throughput Results

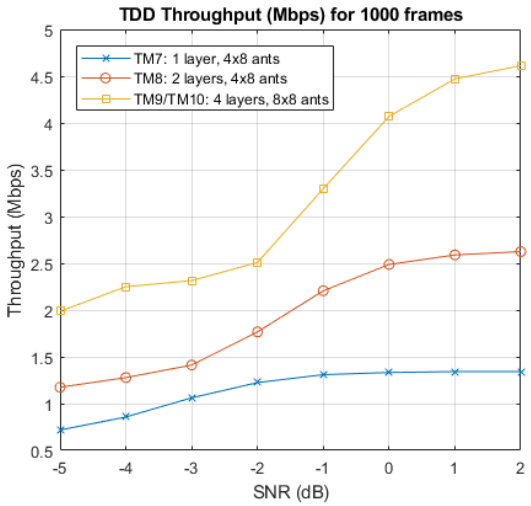
The throughput results are displayed in the MATLAB® command window after each SNR point is completed. They are also captured in output arrays `simThroughput` and `maxThroughput`. `simThroughput` stores the measured throughput in number of bits for all simulated SNR points. `maxThroughput` stores the maximum possible throughput in number of bits for each simulated SNR point.

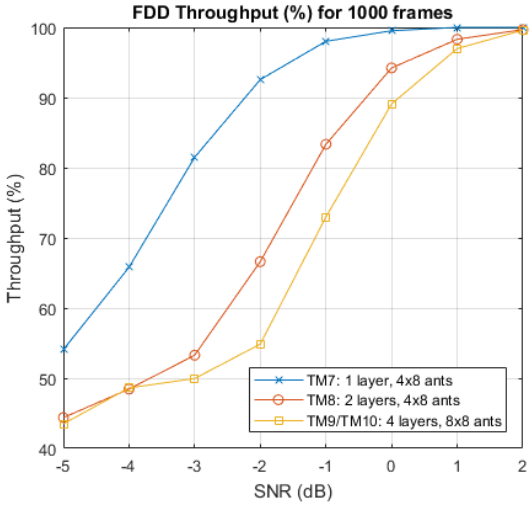
```
% Plot throughput
figure
legendString = [char(txMode) ': ' num2str(simulationParameters.PDSCH.NLayers) ...
                ' layer(s), ' num2str(ntxants) ' ant(s)'];
plot(SNRIn, simThroughput*100./maxThroughput,'*-.');
xlabel('SNR (dB)');
ylabel('Throughput (%)');
title('Throughput vs SNR');
legend(legendString,'Location','NorthWest');
grid on;
```

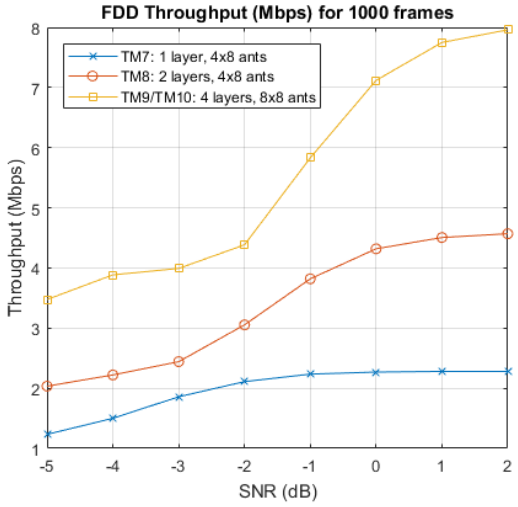


For statistically valid results, the simulation should be run for a larger number of frames. The figure below compares the throughput results for TDD and FDD when simulating 1000 frames.









Appendix

This example uses the following helper functions:

- hDisplayENBParameterSummary.m
- hCalculatePrecodingMatrix.m
- hNewHARQProcess.m
- hHARQScheduling.m
- hPlotRVSequence.m

Selected Bibliography

1 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception"

CoMP Dynamic Point Selection with Multiple CSI Processes

This example shows how multiple Channel State Information (CSI) processes provide the network with feedback for Coordinated Multipoint (CoMP) operation. In this example User Equipment (UE) data is transmitted from one of two cooperating eNodeB as part of a Dynamic Point Selection (DPS) scheme. The transmission decision is based on Channel Quality Indicator (CQI) reports from the UE.

Introduction

Coordinated multipoint (CoMP) is a term used to describe schemes in which a group of base stations dynamically cooperate to mitigate interference, or even turn this interference into a useful signal. The group of coordinating base stations is termed the cooperating set. CoMP in LTE Release 11 is designed to be able to take advantage of low latency and high capacity backhaul between base stations within a cooperating set. Therefore data for a User Equipment (UE) may be available at one or more cooperating base stations.

There are three categories of downlink CoMP:

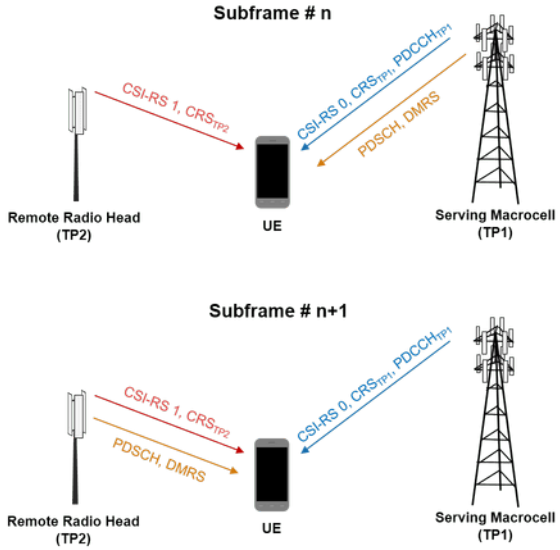
- In Coordinated Scheduling and Beamforming (CS/CB), UE data is only available at a single base station within the coordinating set, therefore the PDSCH is transmitted from a single base station. Scheduling and link adaptation are coordinated using information from other base stations within the cooperating set. Other cooperating base stations can also coordinate their scheduling and beamforming decisions to mitigate interference.
- In Dynamic Point Selection (DPS), UE data is available at multiple base stations within the coordinating set but data is only transmitted from one base station at a time. The base station transmitting to the UE, named the transmission point (TP), can be changed subframe-to-subframe, to provide the best transmission for a UE with varying channel conditions. This scenario is most likely at a cell edge, where long term channel characteristics favor the serving base station, but short term characteristics may favor other cooperating base stations.
- In Joint Transmission (JT), UE data is transmitted from multiple base stations simultaneously. This can be coherent or non-coherent. Coherent JT jointly precodes transmissions from multiple TPs to allow the receiver to achieve coherent combining of the transmission. In non-coherent JT, each TP precodes the transmission independently, therefore only a power gain is available to the receiver.

The network uses Channel State Information (CSI), reported by the UE, or inferred from TDD uplink transmissions, to make CoMP transmission decisions. The UE feeds back multiple reports, each of which correspond to different hypotheses regarding the transmission decisions of cooperating base stations. To provide a report, a UE is configured with a CSI process. A CSI process consists of a CSI Reference Signal (CSI-RS) resource, a CSI interference Measurement resource (CSI-IM) and a reporting mechanism. For CSI reporting, the network can configure a UE with up to four CSI processes. For each CSI process the UE reports calculated CSI indicators as requested by the network:

- Channel Quality Indicator (CQI)
- Rank Indicator (RI)
- Precoder Matrix Indicator (PMI)

For more information about CQI/RI/PMI reporting see the examples “Reporting of Rank Indicator (RI) Conformance Test” on page 2-495 and “Reporting of Channel Quality Indicator (CQI) Conformance Test” on page 2-487.

This example shows a simple DPS scenario for PDSCH transmission to a UE. The cooperating set contains two transmission points, TP1 and TP2, each of which is capable of transmitting the PDSCH to the UE. TP1 is the serving cell for the UE. The network selects the PDSCH transmission point and the modulation and coding scheme using the CQI reported by the UE. The following diagram shows the principle of DPS operation in this example. Both transmission points transmit a CSI-RS and cell-specific reference signal (CRS). The serving cell also transmits downlink control information for the UE in the PDCCH. The PDSCH transmission point can change subframe-to-subframe to take advantage of instantaneous channel conditions. In the diagram the PDSCH transmission point changes from TP1 in subframe `n` to TP2 in subframe `n+1`.



In this example the CSI processes and transmission points are configured as required by the conformance test "CQI Reporting requirement with multiple CSI processes" in TS36.101 Section 9.3.6.1 [1]. This example differs from this conformance test by selecting one of two possible PDSCH transmission points based on the highest reported wideband CQI from either transmission point. In this conformance test the PDSCH transmission point is fixed. Additionally no CQI reporting delay is implemented.

Simulation Controls

In this example the PDSCH transmission point can be either dynamically selected or fixed to the serving cell, TP1, using the parameter `dpsOperation`. Use this parameter to explore the impact of DPS on throughput.

```
dpsOperation = true; % Enable DPS {true,false}
totSubframes = 150;  % Number of subframes to simulate
```

Transmission Point Configurations

Two transmission points are defined and configured as per TS36.101 Table 9.3.6.1-1 [1]: TP1 is a macro cell (the serving cell) and TP2 is a cooperating base station such as a remote radio head. A structure array `enb` contains the parameters for both transmission points.

```
% Transmission point 1 cell-wide settings
enb = struct;
enb.NDLRB = 50;
enb.CellRefP = 2;
```

```
enb.DuplexMode = 'FDD';
enb.CFI = 3;
enb.CyclicPrefix = 'Normal';
enb.NFrame = 0;
enb.NCellID = 0;

% PDSCH configuration for transmission mode 10 (TM10)
enb.PDSCH.TxScheme = 'Port7-14';
enb.PDSCH.NLayers = 1;
enb.PDSCH.RNTI = 1;
enb.PDSCH.NSCID = 0;
enb.PDSCH.Modulation = {'16QAM'};
enb.PDSCH.Rho = 0;
enb.PDSCH.RV = 0;
enb.PDSCH.NTurboDecIts = 5;
enb.PDSCH.PRBSet = (0:enb.NDLRB-1).';
enb.PDSCH.NTxAnts = 4;
enb.PDSCH.W = lteCSICodebook(enb.PDSCH.NLayers,enb.PDSCH.NTxAnts,0).';
enb.PDSCH.AltCodebook4Tx = 'Off';
enb.PDSCH.CSI = 'On';
```

The configuration for TP2 is based on TP1. Different cell-specific settings are configured as required.

```
enb = repmat(enb,2,1);
enb(2).NCellID = 6;
enb(2).PDSCH.NTxAnts = 2;
enb(2).PDSCH.W = lteCSICodebook(enb(2).PDSCH.NLayers,enb(2).PDSCH.NTxAnts,0).';
```

Transmission Hypothesis, CSI Resources and CSI Processes

When the two coordinating transmission points use DPS, the PDSCH can be transmitted from either TP1 or TP2. When the PDSCH is transmitted by one transmission point to a UE, for example TP1, there are two transmission options for the other transmission point, TP2. The first option is to serve other UEs using the same resources thereby interfering with the PDSCH transmission from TP1. The second option is to mute transmission in these resources thereby not interfering with the PDSCH transmission from TP1. These options are grouped into transmission hypotheses. In this example four transmission hypotheses are tested by the network:

```
               TP1 Hypothesis       TP2 Hypothesis
-----------------------------------------------------
Hypothesis 0:  Transmitting PDSCH   Muting
Hypothesis 1:  Muting               Transmitting PDSCH
Hypothesis 2:  Transmitting PDSCH   Interfering
Hypothesis 3:  Interfering          Transmitting PDSCH
```

Although four hypotheses are tested in this example, PDSCH transmissions made to the UE are only consistent with hypotheses 2 or 3.

To provide the network with CSI for these transmission hypotheses, two CSI-RS resources and three CSI-IM resources are configured at the UE. CSI processes use these resources to report CSI for each hypothesis.

In the following sections the resources and CSI processes are configured to test these four transmission hypotheses.

CSI-RS Resources

A unique CSI-RS is transmitted by each cooperating base station. The UE is configured with two CSI-RS resources to provide channel quality estimates, one for each transmission point:

- CSI-RS #0: Transmission from TP1
- CSI-RS #1: Transmission from TP2

Each CSI-RS is defined by a configuration, a period and a CSI-RS scrambling identity. The number of CSI reference ports is the number of transmit antennas. For this simulation the periods of CSI-RS and CSI-IM resources must be the same. These are parameterized by `SimCSIPeriod`.

```
SimCSIPeriod = [5 1]; % [Tcsi-rs Dcsi-rs]

% CSI-RS resource: {CSI-RS #0, CSI-RS #1}
SimCSIRS.CSIRSConfig = [0 5];                        % CSI-RS configuration
SimCSIRS.CSIRSPeriod = {SimCSIPeriod,SimCSIPeriod}; % CSI-RS period
SimCSIRS.NCSIID = [10 16];                           % CSI-RS scrambling identity
SimCSIRS.CSIRefP = [enb(1).PDSCH.NTxAnts enb(2).PDSCH.NTxAnts];
```

CSI-IM Resources

CSI-IM resources describe a set of Resource Elements (REs) over which the average power is measured by the UE. These measurements are used to estimate the interference for CSI calculations. Three CSI-IM are required to measure interference when the TPs are transmitting:

- CSI-IM #0: Measure background noise when both TPs are muted
- CSI-IM #1: Measure TP2 interference
- CSI-IM #2: Measure TP1 interference

Each CSI-IM is defined by a configuration and a period. Note the configurations differ from the CSI-RS configurations but the periods are the same.

```
% CSI-IM resource: {CSI-IM #0, CSI-IM #1, CSI-IM #2}
SimCSIIM.ZeroPowerCSIRSConfig = [2 6 1];
SimCSIIM.ZeroPowerCSIRSPeriod = {SimCSIPeriod,SimCSIPeriod,SimCSIPeriod};
```

CSI Processes

Four processes are configured to test the four transmission hypotheses. These use the CSI-RS and CSI-IM resources described above:

```
            TP1 Hypothesis      TP2 Hypothesis      CSI-RS     CSI-IM
-------------------------------------------------------------------------
Process 0:  Transmitting PDSCH  Muting              CSI-RS #0  CSI-IM #0
Process 1:  Muting              Transmitting PDSCH  CSI-RS #1  CSI-IM #0
Process 2:  Transmitting PDSCH  Interfering         CSI-RS #0  CSI-IM #1
Process 3:  Interfering         Transmitting PDSCH  CSI-RS #1  CSI-IM #2
```

A process is defined by a CSI-RS resource, a CSI-IM resource and a reporting mode. The CSI reporting mode, PMI reporting mode and codebook subset restriction for each process are configured as per TS36.101 Table 9.3.6.1-1 [1]. The codebook subset restriction for each process restricts PMI selection to a single PMI therefore PMI and RI reporting is not required.

```
% {CSI Process #0, CSI Process #1, CSI Process #2, CSI Process #3}
SimCSIProcess.CSIRSResource  = [1 2 1 2]; % CSI-RS resources index (1 based)
SimCSIProcess.CSIIMResource  = [1 1 2 3]; % CSI-IM resources index (1 based)
SimCSIProcess.CSIMode        = {'PUCCH 1-1','PUSCH 3-1','PUSCH 3-1','PUSCH 3-1'}; % CSI reporting
```

```
SimCSIProcess.PMIMode        = {'Wideband' ,'Wideband' ,'Wideband','Wideband'};   % PMI reporting
SimCSIProcess.CodebookSubset = {'0x0000000000000001','000001','0x0000000000000001','000001'}; % C
```

In this simulation only two of the four hypotheses are implemented by the network; PDSCH transmission from either TP1 or TP2 with the other TP interfering. Therefore only the feedback from CSI processes 2 and 3 is used for the transmission decision.

Fading Channel and SNR Configuration

The SNR for each transmission to the UE is defined in TS36.101 Table 9.3.6.1-1 [1]. The noise power is defined using `Noc`.

```
snrTP = [11 8]; % SNR of received transmission from TP1 and TP2
Noc = -98;      % dBm/15kHz average power spectral density
```

A fading channel is configured between TP1 and the UE, and TP2 and the UE. The structure array `chcfg` parameterizes the channels from TP1 and TP2.

```
% Channel between TP1 and UE
ofdmInfo = lteOFDMInfo(enb(1));
chcfg = struct;
chcfg.DelayProfile = 'EPA';
chcfg.NRxAnts = 2;
chcfg.DopplerFreq = 5;
chcfg.MIMOCorrelation = 'Low';
chcfg.SamplingRate = ofdmInfo.SamplingRate;
chcfg.InitPhase = 'Random';
chcfg.ModelType = 'GMEDS';
chcfg.NTerms = 16;
chcfg.NormalizeTxAnts = 'On';
chcfg.NormalizePathGains = 'On';
chcfg.Seed = 1;

% Channel between TP2 and UE
chcfg = repmat(chcfg,2,1);
chcfg(2).Seed = 2;

% Calculate the size of the received waveform
rxWaveformSize = [chcfg(1).SamplingRate*1e-3+15 chcfg(1).NRxAnts];
```

Channel Estimation Configuration

Two reference signals must be used by the UE: the CSI-RS and DM-RS. Two separate channel estimation configurations are required to estimate each reference signal. Here cubic interpolation will be used with an averaging window of 1-by-2 REs. This configures the channel estimator to use a special mode which ensures the ability to despread and orthogonalize the CSI-RS and DMRS transmissions.

```
% CSI-RS estimation
ceccsi.FreqWindow = 1;
ceccsi.TimeWindow = 2;
ceccsi.InterpType = 'cubic';
ceccsi.PilotAverage = 'UserDefined';
ceccsi.InterpWinSize = 1;
ceccsi.InterpWindow = 'Causal';
ceccsi.Reference = 'CSIRS';

% DM-RS estimation
```

```
cecdmrs = ceccsi;
cecdmrs.Reference = 'DMRS';
```

Transmission Point Parameterization for CSI Estimation and Reporting

In this section the transmission points are parameterized from the configured CSI-RS resources and CSI-IM resources for CSI-RS generation and PDSCH mapping. The appropriate parameters are set in the configuration structure array `enb`.

CSI-RS Resources

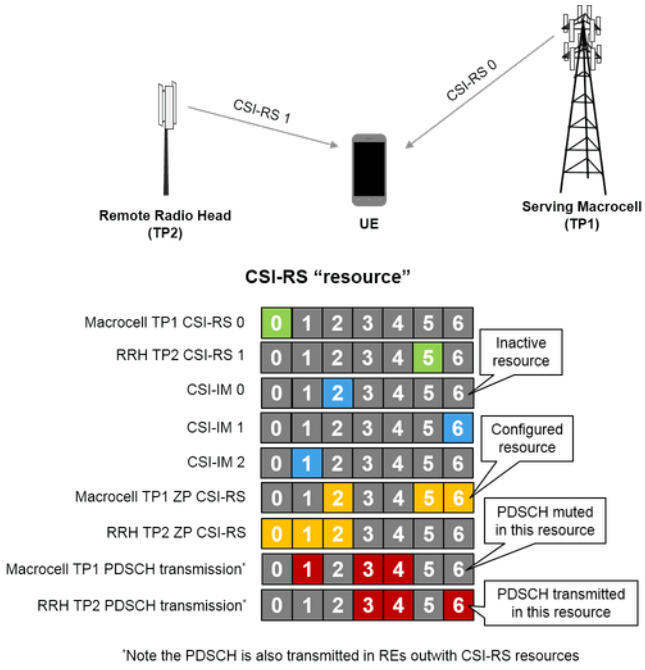
Both transmission points are parameterized with all CSI-RS configured at the UE.

```
% Set TP1 and TP2 CSI-RS configuration
for enbIdx = 1:2
    enb(enbIdx).CSIRefP = SimCSIRS.CSIRefP;
    enb(enbIdx).CSIRSConfig = SimCSIRS.CSIRSConfig;
    enb(enbIdx).CSIRSPeriod = SimCSIRS.CSIRSPeriod;
    enb(enbIdx).NCSIID = SimCSIRS.NCSIID;
end
```

ZP CSI-RS Resources

Zero Power (ZP) CSI-RS resources prevent the PDSCH from being mapped to a set of REs. Therefore ZP CSI-RS are used to mute REs within each TP PDSCH transmission to allow configured CSI-IM to measure the interference for different hypotheses.

The diagram below illustrates how ZP CSI-RS are configured to allow CSI-IM resources to measure interference. CSI-IM #0 measures the noise when neither TP is transmitting in resource 2. Therefore to prevent the TPs transmitting in this resource, the ZP CSI-RS is configured in resource 2 for both TPs. This stops the PDSCH being mapped to this resource, muting the transmission. The ZP CSI-RS for each TP is also configured to map around the CSI-RS of the coordinating TP to prevent interference. CSI-IM #1 and CSI-IM #2 measure the individual interference caused by TP2 and TP1 in resources 6 and 1. Therefore the ZP CSI-RS for TP1 is configured to prevent PDSCH transmission in resource 6 to allow measurement with CSI-IM #1. The ZP CSI-RS for TP2 is configured to prevent PDSCH transmission in resource 1 to allow measurement with CSI-IM #2.



The ZP CSI-RS are configured with a 16-bit bitmap. Each bit controls whether a set of REs should be muted (1) or unmuted (0). For each transmission point, a ZP CSI-RS configuration is created from the required CSI-IM configurations and the CSI-RS configuration used by the coordinating transmission point.

```
% ZP CSI-RS resource for TP1
zp1 = '0000000000000000';                              % Assume all CSI configurations unmuted
zp1(SimCSIIM.ZeroPowerCSIRSConfig([1 2])+1) = '1'; % Mute CSI-IM #0,1 (background & TP2 interfere
zp1(SimCSIRS.CSIRSConfig(2)+1) = '1';                  % Mute CSI-RS #1 (TP2 transmission)

% Add ZP CSI-RS resource to TP1 parameters
enb(1).ZeroPowerCSIRSConfig = zp1;
enb(1).ZeroPowerCSIRSPeriod = SimCSIPeriod;

% ZP CSI-RS resource for TP2
zp2 = '0000000000000000';                              % Assume all CSI configurations unmuted
zp2(SimCSIIM.ZeroPowerCSIRSConfig([1 3])+1) = '1'; % Mute CSI-IM #0,2 (background & TP1 interfere
zp2(SimCSIRS.CSIRSConfig(1)+1) = '1';                  % Mute CSI-RS #0 (TP1 transmission)

% Add ZP CSI-RS resource to TP2 parameters
enb(2).ZeroPowerCSIRSConfig = zp2;
enb(2).ZeroPowerCSIRSPeriod = SimCSIPeriod;
```

UE Parameterization for CSI Estimation and Reporting

In this example the CSI-RS, CSI-IM and CSI processes are represented at the UE as structure arrays. Each element of the structure array configures a single resource or process. This section creates these structure arrays from the configured CSI-RS resources and CSI-IM resources.

The structure array `csirs` contains the CSI-RS resource parameterization. This is based on the parameters of the serving cell but the CSI-RS parameters are configured to use the appropriate resource given in the simulation settings.

```
numCSIRS = numel(SimCSIRS.CSIRSConfig);
csirs = repmat(enb(1),numCSIRS,1);
for idx = 1:numCSIRS
    csirs(idx).CSIRefP = SimCSIRS.CSIRefP(idx);
    csirs(idx).CSIRSConfig = SimCSIRS.CSIRSConfig(idx);
    csirs(idx).CSIRSPeriod = SimCSIRS.CSIRSPeriod{idx};
    csirs(idx).NCSIID = SimCSIRS.NCSIID(idx);
end
```

The structure array `csiim` contains the CSI-IM resource parameterization. This is based on the parameters of the serving cell but the CSI-IM parameters are configured to use the appropriate resource give in the simulation settings. As a CSI-IM resource is a ZP CSI-RS configuration the `CSIRSPeriod` parameter is set to `'Off'` so only ZP CSI-RS REs are used to measure interference.

```
numCSIIM = numel(SimCSIIM.ZeroPowerCSIRSConfig);
csiim = repmat(enb(1),numCSIIM,1);
for idx = 1:numCSIIM
    csiim(idx).ZeroPowerCSIRSConfig = SimCSIIM.ZeroPowerCSIRSConfig(idx);
    csiim(idx).ZeroPowerCSIRSPeriod = SimCSIIM.ZeroPowerCSIRSPeriod{idx};
    csiim(idx).CSIRSPeriod = 'Off';
end
```

The structure array `process` contains the CSI process parameterization. This is based on the parameters of the serving cell. The parameters `CSIRSIdx` and `CSIIMIdx` are used to index the CSI-RS and CSI-IM resources for calculating CSI. The reporting modes are configured for each process from the simulation settings. The CQI reported by the UE for each CSI process is selected using the estimated Signal to Interference plus Noise Ratio (SINR). The lowest SINRs recommended to achieve 90% throughput for each CQI index in this scenario are defined by the vector `SINRs`. This vector is used to parameterize the CQI selection for each CSI process.

```
SINRs = [1.3 1.3 2.3 3.7 5 6.8 9.2 10.9 13 14.8 17.1 18.9 21 23.9 24.3];
numCSIProcesses = numel(SimCSIProcess.CSIRSResource);
process = repmat(enb(1),numCSIProcesses,1);
for idx = 1:numCSIProcesses
    % Index CSI-RS and CSI-IM resources used by the process
    process(idx).CSIRSIdx = SimCSIProcess.CSIRSResource(idx);
    process(idx).CSIIMIdx = SimCSIProcess.CSIIMResource(idx);

    % Reporting configuration
    process(idx).PDSCH.CSIMode = SimCSIProcess.CSIMode{idx};
    process(idx).PDSCH.PMIMode = SimCSIProcess.PMIMode{idx};
    process(idx).PDSCH.CodebookSubset = SimCSIProcess.CodebookSubset{idx};
    process(idx).PDSCH.SINRs90pc = SINRs;

    % CSI-RS configuration for CSI estimation
    process(idx).CSIRefP = SimCSIRS.CSIRefP(process(idx).CSIRSIdx);
    process(idx).CSIRSConfig = SimCSIRS.CSIRSConfig(process(idx).CSIRSIdx);
    process(idx).CSIRSPeriod = SimCSIRS.CSIRSPeriod(process(idx).CSIRSIdx);
    process(idx).NCSIID = SimCSIRS.NCSIID(process(idx).CSIRSIdx);
end
```

Simulation Setup

The required signal power to satisfy the SNR for each TP is calculated below.

```
% Convert to linear and Watts
nocLin = 10.^(Noc/10)*(1e-3); % linear in Watts
```

```
% Take into account number of antennas and FFT (OFDM) scaling
No = sqrt(nocLin/(2*double(ofdmInfo.Nfft)));
NocW = 10.^((Noc-30)/10); % convert to W/15kHz

% SINR = Es/Noc TS 36.101 Sec. 8.1.1
NocTot = NocW; % W/15kHz
snrLin = 10.^(snrTP/10);
Es = snrLin*NocTot; % W/15kHz

% Amplitude scaling factors
K = sqrt(Es);
```

Variables required for the simulation are initialized in this section.

```
% Set the random number seed
rng('default');

% Initialize containers which store the channel estimates interference
% measurements for CSI-RS and CSI-IM resources
csirshest = cell(numCSIRS,1);
csiimnest = zeros(numCSIIM,1);

% Initialize a buffer to store CQI reports for each process. Size buffer
% for subband CQI reporting.
numCSIReports = ceil(totSubframes/SimCSIPeriod(1));
pmiInfo = ltePMIInfo(process(1),setfield(process(1).PDSCH,'PMIMode','Subband')); %#ok<SFLD>
cqiBuffer = ones(numCSIReports,pmiInfo.NSubbands+1,numCSIProcesses);

% Initialize buffers to store the CRC, BER and TP selected
crcBuffer = cell(totSubframes,1);
berBuffer =  zeros(totSubframes,2);
tpBuffer = zeros(totSubframes,1);

csiReportIdx = 1; % Index of CSI report
lastOffset = 0;   % Initialize overall frame timing offset
frameOffset = 0;  % Initialize frame timing offset
```

Simulation Loop

The simulation is run subframe-by-subframe. For each subframe the following steps are carried out:

- A time domain waveform `rxWaveform` is initialized with noise. The received waveform from TP1 and TP2 will be added to this.
- The PDSCH TP is selected using the wideband CQI reported by the UE.
- For each TP in turn a subframe is generated containing the relevant synchronizing and reference signals and TM10 OCNG.
- The PDSCH for the UE is generated from either TP1 or TP2 in a single subband.
- The two subframes are OFDM modulated, passed through a fading channel, and combined.
- The received waveform at the UE is synchronized and OFDM demodulated.
- The configured CSI-RS and CSI-IM measurements are performed by the UE
- A CSI report is generated using the CSI-RS and CSI-IM resources for configured CSI processes.
- The PDSCH is demodulated.

```
for nsf = 0:totSubframes-1
    % Initialize UE receive waveform with noise
```

```
rxWaveform = No*complex(randn(rxWaveformSize),randn(rxWaveformSize));

% Select PDSCH TP based on the highest reported wideband CQI
bufferIdx = mod(csiReportIdx-2,numCSIReports)+1; % Buffered CQI to use
if dpsOperation
    widebandCQI = permute(cqiBuffer(:,1,:),[1 3 2]);
    if (widebandCQI(bufferIdx,3)>=widebandCQI(bufferIdx,4))
        pdschTransmissionPoint = 1; % TP1
    else
        pdschTransmissionPoint = 2; % TP2
    end
else
    pdschTransmissionPoint = 1; %#ok<UNRCH> % TP1
end

% Generate TP1 and TP2 transmissions.
% Each transmission contains cell-specific reference signal,
% synchronizing signals, CSI-RS and TM10 OCNG. One transmission
% contains the PDSCH for the UE.
for enbIdx = 1:2
    % Update subframe number for each transmission
    enb(enbIdx).NSubframe = nsf;

    % Turn off the CSI resource not transmitted by this TP
    tpenb = enb(enbIdx);
    tpenb.CSIRSPeriod{mod(enbIdx,2)+1} = 'Off';

    % Blank subframe; CRS, PSS and SSS
    sf = lteResourceGrid(tpenb,tpenb.PDSCH.NTxAnts);
    crsInd = lteCellRSIndices(tpenb); % Cell-specific reference signal
    sf(crsInd) = lteCellRS(tpenb);
    pssInd = ltePSSIndices(tpenb);    % Primary synchronizing signal
    sf(pssInd) = ltePSS(tpenb);
    sssInd = lteSSSIndices(tpenb);    % Secondary synchronizing signal
    sf(sssInd) = lteSSS(tpenb);

    % CSI-RS resource
    csitp = tpenb;
    csitp.ZeroPowerCSIRSPeriod = 'Off';
    csiInd = lteCSIRSIndices(csitp);
    sf(csiInd) = lteCSIRS(csitp);

    % TM10 OCNG transmitted apart from subframes containing PSS/SSS/PBCH
    if isempty(pssInd)
        tpenb.PDSCH.RNTI = 0;

        % Add OCNG at DMRS locations
        oncngInd = lteDMRSIndices(tpenb,tpenb.PDSCH);
        ocngSym = lteDMRS(tpenb,tpenb.PDSCH);
        sf(oncngInd) = ocngSym;

        % Add OCNG for PDSCH symbols
        [oncngInd,ocngInfo] = ltePDSCHIndices(tpenb,tpenb.PDSCH,tpenb.PDSCH.PRBSet);
        ocngSym = ltePDSCH(tpenb,tpenb.PDSCH,randi([0 1],ocngInfo.G,1));
        sf(oncngInd) = ocngSym;
    end

    % PDSCH and DMRS Transmission to UE from either TP1 or TP2.
```

```
% The transport block size and modulation scheme for transmission
% are selected using the reported CQI for the appropriate CSI
% process. The PDSCH must be mapped around the ZP CSI-RS configured
% for the UE and the ZP CSI-RS of the TP. Only transmit PDSCH for
% subframes not containing CSI resources or PSS/SSS as per TS36.101
% Table 9.3.6.1-1.
if ~isempty(pssInd)||~isempty(csiInd)
    tbs = 0; % Transport block size is 0 as no PDSCH transmitted
elseif (pdschTransmissionPoint==enbIdx)
    tpBuffer(nsf+1) = pdschTransmissionPoint;
    % Get relevant configuration for transmission point and CQI to
    % use.
    txenb = enb(enbIdx);
    cqi = cqiBuffer(bufferIdx,:,enbIdx+2);

    % Select subband for PDSCH transmission.
    % The PDSCH is transmitted in the highest differential CQI
    % subband. Subbands less than full size are excluded.
    partialSubband = (pmiInfo.k*pmiInfo.NSubbands>txenb.NDLRB);
    [~,idx] = max(cqi(2:(end-partialSubband))); % Maximum differential
    sbCandidates = find(cqi(2:(end-partialSubband))==cqi(idx+1));
    sb = sbCandidates(randi([1 numel(sbCandidates)],1,1));
    cqi = cqi(1)+cqi(1+sb); % Calculate SB PMI from wideband and differential
    txenb.PDSCH.PRBSet = ((sb-1)*pmiInfo.k+(0:(pmiInfo.k-1))).'; % PRB allocation for sub

    % Select MCS according to CQI using TS36.101 Table A.4-1 CSI
    % RMC RC.12 FDD (MCS.13), which defines the relationship
    % between CQI indices and MCS indices
    imcsTable = [-1 0 0 1 3 5 7 10 12 14 17 19 21 22 24 25];
    imcs = imcsTable(cqi+1);

    % Determine TBS and modulation order, fixed RI (1) and PMI (0).
    % Generate PDSCH for only a non-zero transport block size.
    [itbs,modulation] = lteMCS(imcs);
    tbs = double(lteTBS(size(txenb.PDSCH.PRBSet,1),itbs));
    if any(tbs)
        if ~iscell(modulation)
            modulation = {modulation};
        end
        txenb.PDSCH.NLayers = 1;
        txenb.PDSCH.Modulation = modulation;
        txenb.PDSCH.W = lteCSICodebook(txenb.PDSCH.NLayers,txenb.PDSCH.NTxAnts,0).';

        % PDSCH mapping
        [pdschInd,pdschInfo] = ltePDSCHIndices(txenb,txenb.PDSCH,txenb.PDSCH.PRBSet);

        % Generate DL-SCH data
        txtrblk = arrayfun(@(x)randi([0 1],x,1),tbs,'UniformOutput',false);
        cw = lteDLSCH(txenb,txenb.PDSCH,pdschInfo.G,txtrblk);

        % Generate PDSCH symbols with cell identity of serving cell
        % for correct scrambling
        txenb.NCellID = enb(1).NCellID;
        pdschSym = ltePDSCH(txenb,txenb.PDSCH,cw);

        % Create UE specific DMRS configuration to allow for
        % scrambling code to change depending on transmission point
        txenb.NCellID = enb(1).NCellID;
```

```
                if pdschTransmissionPoint == 2
                    txenb.NCellID = enb(2).NCellID;
                end
                dmrsInd = lteDMRSIndices(txenb,txenb.PDSCH);
                dmrsSym = lteDMRS(txenb,txenb.PDSCH);

                % Map PDSCH and DMRS
                sf(pdschInd) = pdschSym;
                sf(dmrsInd) = dmrsSym;
            end
        end

        % OFDM modulate, pass through a fading channel, scale for SNR and
        % add to existing receive waveform
        txWaveform = lteOFDMModulate(tpenb,sf);
        txWaveform = [txWaveform; zeros(15,size(txWaveform,2))]; %#ok<AGROW>
        chcfg(enbIdx).InitTime = nsf/1e3;
        rxWaveform = rxWaveform + K(enbIdx)*lteFadingChannel(chcfg(enbIdx),txWaveform);
    end

    % Receiver Synchronization and OFDM demodulation
    % Synchronize using the PSS/SSS of the serving cell (TP1) and OFDM
    % demodulate
    if ~isempty(pssInd)
        frameOffset = lteDLFrameOffset(enb(1),rxWaveform);
        if (frameOffset > 25)
            frameOffset = lastOffset;
        end
        lastOffset = frameOffset;
    end
    rxWaveform = rxWaveform(1+frameOffset:end,:);
    rxsf = lteOFDMDemodulate(enb(1),rxWaveform);

    % Calculate CSI-RS Estimates
    % Generate channel estimates for CSI-RS resources configured at the UE.
    for idx = 1:numCSIRS
        % Calculate channel estimate
        csirs(idx).NSubframe = nsf;
        csirshest{idx} = lteDLChannelEstimate(csirs(idx),csirs(idx).PDSCH,ceccsi,rxsf);
    end

    % Calculate interference using CSI-IM
    % For each CSI-IM resource calculate the energy in resource elements.
    % This is the noise+interference estimate.
    for idx = 1:numCSIIM
        % Calculate noise and interference estimate
        csiim(idx).NSubframe = nsf;
        imIndices = lteCSIRSIndices(csiim(idx));
        imSym = lteExtractResources(imIndices,rxsf);
        csiimnest(idx) = mean(abs(imSym(:)).^2);
    end

    % CSI Process Reporting
    % When estimated CSI resource elements are not zero calculate CSI
    if ~isempty(csiInd)
        % For each CSI process calculate CSI feedback
        for idx = 1:numCSIProcesses
            % Update subframe number
```

```
            process(idx).NSubframe = nsf;

            % Extract CSI-RS estimate and CSI-IM estimate for process
            hest = csirshest{process(idx).CSIRSIdx};
            nest = csiimnest(process(idx).CSIIMIdx);

            % Calculate CQI/PMI/RI, condition CQI on PMI/RI selection
            [ri,PMISet] = lteRISelect(process(idx),process(idx).PDSCH,hest,nest);
            process(idx).PDSCH.PMISet = PMISet;
            process(idx).PDSCH.NLayers = ri;
            process(idx).PDSCH.NCodewords = min(ri,2);
            [cqi,sinrs] = lteCQISelect(process(idx),process(idx).PDSCH,hest,nest);
            cqiBuffer(csiReportIdx,1:numel(cqi),idx) = cqi;
        end

        % New CSI report
        csiReportIdx = csiReportIdx+1;
    end

    % PDSCH Demodulation
    % If the transport block size is not 0, a PDSCH exists to decode
    if any(tbs)
        % Estimate channel using DMRS. To use the correct scrambling
        % sequence for the DMRS use the configuration for the active TP.
        [dmrshest,dmrsnest] = lteDLChannelEstimate(txenb,txenb.PDSCH,cecdmrs,rxsf);

        % Extract PDSCH symbols from received grid and channel estimate
        [sym,symhest] = lteExtractResources(pdschInd,rxsf,dmrshest);

        % Scale the received symbols by the PDSCH power factor Rho and
        % decode the PDSCH with the CellID for the serving cell
        sym = sym*(10^(-txenb.PDSCH.Rho/20));
        txenb.NCellID = enb(1).NCellID;
        [cws,recsym] = ltePDSCHDecode(txenb,txenb.PDSCH,sym,symhest,dmrsnest);
        % Scale cws by 1/K(1) to avoid numerical issues with DLSCH decoding
        cws = cellfun(@(x) x*(1/K(1)),cws,'UniformOutput',false);
        [trblk,crc] = lteDLSCHDecode(txenb,txenb.PDSCH,tbs,cws,[]);

        % Store CRC and BER
        crcBuffer{nsf+1} = double(crc);
        berBuffer(nsf+1,:) = [sum(trblk{1}~=txtrblk{1}),numel(trblk{1})];
    end
end
```

Conclusion and Results

The Block Error Rate (BLER) and PDSCH throughput are displayed for the simulation duration. Two figures are also created:

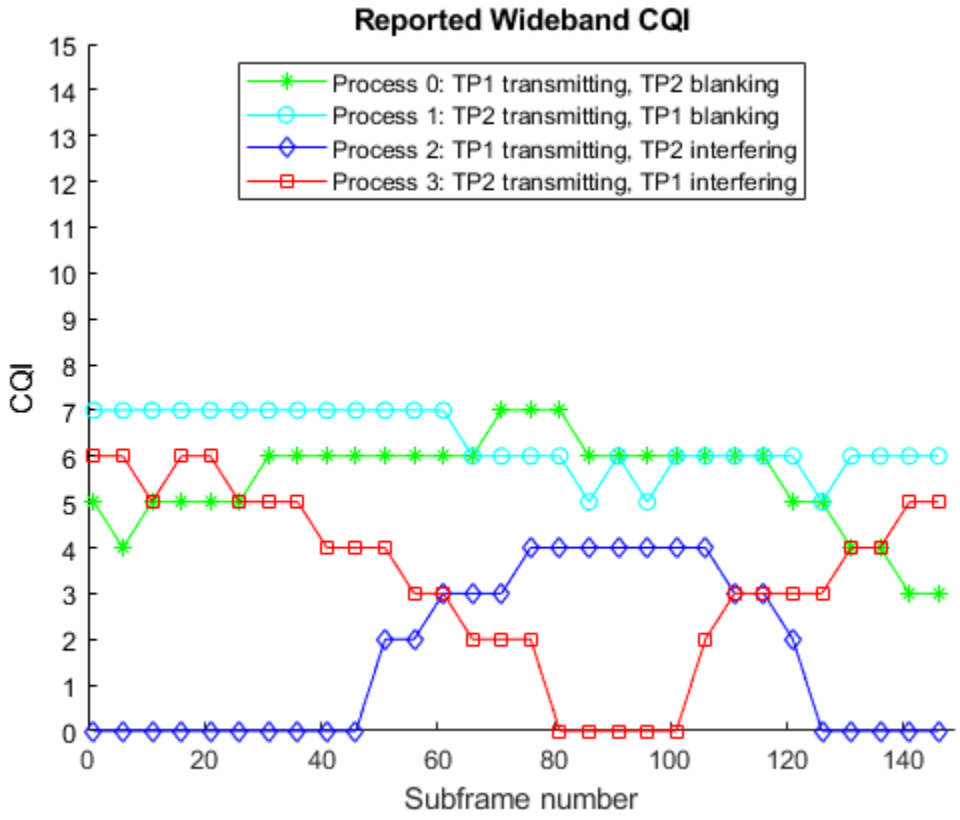
The first figure plots the reported wideband CQI for each CSI process over the duration of the simulation. The reported CQI of process 0 and process 1 show the channel conditions favor TP1 during the middle of the simulation but TP2 otherwise, as the reported CQI exceeds than of TP1. The reported CQI of processes 2 and 3 show a similar pattern, but the reported CQI is lower than for processes 0 and 1. This is because these processes assume added interference from the TPs.

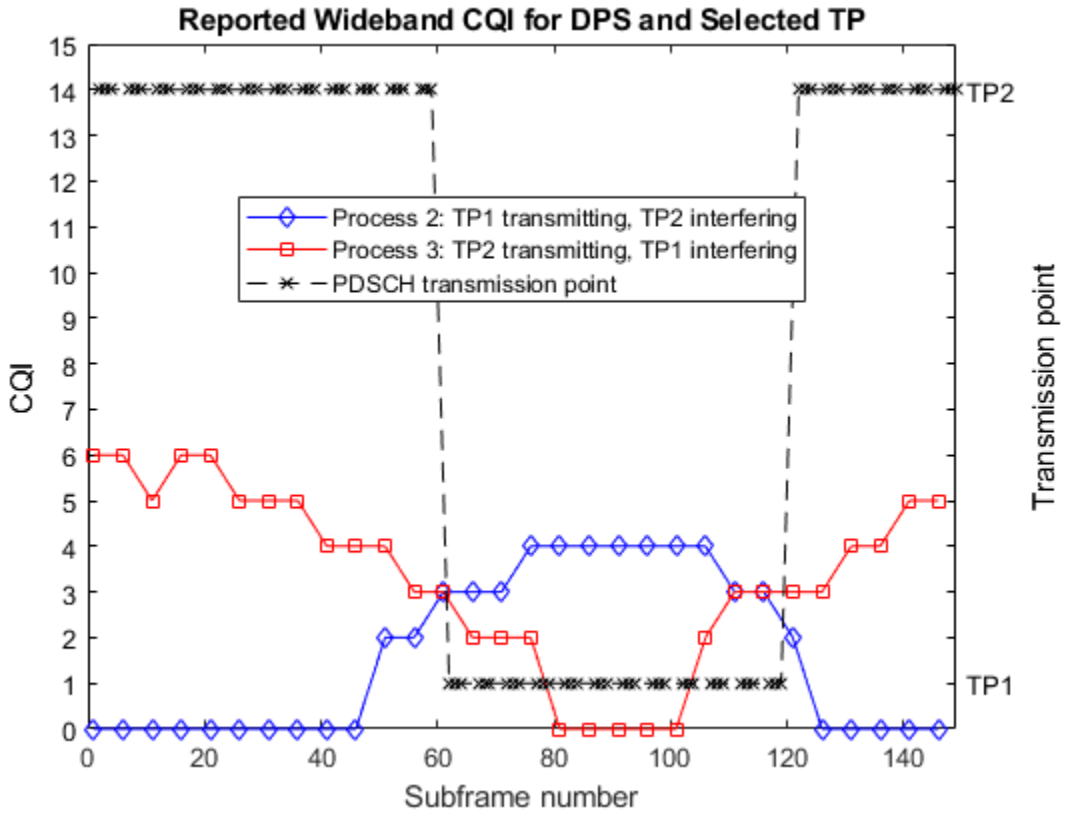
The second figure plots the PDSCH transmission point selected and the reported wideband CQI of the two processes used to make the transmission point decision. This figure shows TP1 was selected for

PDSCH transmission during the middle of the simulation as the reported CQI favored this transmission point.

```
hCoMPResults(totSubframes,SimCSIPeriod,crcBuffer,berBuffer,cqiBuffer,tpBuffer);
```

```
BLER: 0.044444 (target is 0.1)
Throughput: 266.875 kbps
```





This example showed how multiple CSI processes provide feedback for DPS CoMP operation. UE data was transmitted from one of two cooperating eNodeB, based on the wideband CQI reported by the UE. This examples simulated a cell-edge scenario where DPS provides a throughput gain for the UE. Try disabling DPS by setting `dpsOperation = false` and note the decrease in throughput.

Appendix

This example uses the following helper function:

- hCoMPResults.m

Selected Bibliography

1 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception"

2 3GPP TS 36.211 "Physical channels and modulation"

3 Erik Dahlman, 4G: LTE/LTE-Advanced for Mobile Broadband, Elsevier 2014

4 Joydeep Acharya, Heterogeneous Networks in LTE-Advanced, Wiley 2014

PUSCH Throughput Conformance Test

This example demonstrates how to measure the Physical Uplink Shared Channel (PUSCH) throughput performance using the LTE Toolbox™ under conformance test conditions as defined in TS36.104.

Introduction

TS 36.104 [1] defines the performance requirements for physical uplink shared channel (PUSCH) as a minimum throughput for a given SNR assuming hybrid automatic repeat request (HARQ) retransmissions. This example demonstrates how the conformance test can be constructed using the LTE Toolbox.

Transmission is simulated using the Extended Pedestrian A (EPA) propagation channel model using 8 HARQ processes. Channel noise is added to the received waveform which is then SC-FDMA demodulated, resulting in a received resource grid for each receive antenna. Channel estimation is performed to determine the channel between each transmit/receive antenna pair. Minimum Mean Square Error (MMSE) equalization is performed on the received resource grid using the estimated channel to recover the resource grid. PUSCH data is then extracted and decoded from this recovered resource grid. Using the result of the block CRC, the throughput performance of the transmit/receive chain is determined.

Simulation Configuration

The example is executed for a simulation length of 1 frame at an SNR of -4.1 dB, -2.0 dB and 0.1 dB as per TS 36.104, Table 8.2.1.1-1 [1]. A large number of `NFrames` should be used to produce meaningful throughput results. `SNRIn` can be an array of values or a scalar.

```
NFrames = 1;                % Number of frames to simulate at each SNR
SNRIn = [-4.1, -2.0, 0.1];  % SNR points to simulate
```

UE Configuration

User Equipment (UE) settings are specified in a structure form.

```
ue.TotSubframes = 1;    % Total number of subframes to generate a waveform for
ue.NCellID = 10;        % Cell identity
ue.RC = 'A3-2';         % FRC number
ue.PUSCH.NLayers = 1;   % Number of layers
ue.NTxAnts = 1;         % Number of transmit antennas
```

Propagation Channel Model Configuration

Propagation channel model characteristics are set using a structure containing the fields specified below. These are set according to TS 36.104, Table 8.2.1.1-1 [1].

```
chcfg.NRxAnts = 2;               % Number of receive antennas
chcfg.DelayProfile = 'EPA';      % Delay profile
chcfg.DopplerFreq = 5.0;         % Doppler frequency
chcfg.MIMOCorrelation = 'Low';   % MIMO correlation
chcfg.Seed = 100;                % Channel seed
chcfg.NTerms = 16;               % Oscillators used in fading model
chcfg.ModelType = 'GMEDS';       % Rayleigh fading model type
chcfg.InitPhase = 'Random';      % Random initial phases
chcfg.NormalizePathGains = 'On'; % Normalize delay profile power
chcfg.NormalizeTxAnts = 'On';    % Normalize for transmit antennas
```

Channel Estimator Configuration

Channel estimation settings are defined using a structure `cec`. An EPA delay profile causes the channel response to change slowly over frequency. Therefore, a large frequency averaging window of 13 Resource Elements (REs) is used. The Demodulation Reference Signal (DRS) is contained in only one symbol per slot, therefore a time averaging window of 1 RE is used. This will not include any pilots from an adjacent slot when averaging.

```
if ue.PUSCH.NLayers == 1
   freqWindow = 13;
else
   freqWindow = 12; % Averaging to take advantage of orthogonal DMRS for multilayer
end
cec.FreqWindow = freqWindow;      % Frequency averaging windows in REs
cec.PilotAverage = 'UserDefined'; % Type of pilot averaging
cec.TimeWindow = 1;               % Time averaging windows in REs
cec.InterpType = 'cubic';         % Interpolation type
cec.Reference = 'Antennas';       % Reference for channel estimation
```

Uplink RMC Configuration

To generate the uplink Reference Model Channel (RMC) the LTE Toolbox functions `lteRMCUL` and `lteRMCULTool` are used. `lteRMCUL` creates a configuration structure for given UE settings; specific to a given Fixed Reference Channel (FRC). This configuration structure is constructed as per TS36.104 Annex A [1] and is used by `lteRMCULTool` to generate an SC-FDMA modulated waveform. The sub-structure `PUSCH` defines the parameters associated with PUSCH; containing the vector defining the transport data capacity per subframe. These lengths are used when decoding Uplink Shared Channel (UL-SCH).

```
% Generate FRC configuration structure for A3-2
frc = lteRMCUL(ue);

% Transport block sizes for each subframe within a frame
trBlkSizes = frc.PUSCH.TrBlkSizes;
codedTrBlkSizes = frc.PUSCH.CodedTrBlkSizes;

% Redundancy version sequence
rvSequence = frc.PUSCH.RVSeq;
```

Set Propagation Channel Model Sampling Rate

The sampling rate for the channel model is set using the value returned from `lteSCFDMAInfo`.

```
info = lteSCFDMAInfo(frc);
chcfg.SamplingRate = info.SamplingRate;
```

Processing Loop

The throughput test is carried out over a number of SNR points. To determine the throughput at an SNR point, the PUSCH data is analyzed on a subframe by subframe basis using the following steps:

- *Update Current HARQ Process.* After every 8 subframes, the given HARQ process either carries new transport data or a retransmission of previously sent transport data depending upon the Acknowledgment (ACK) or Negative Acknowledgment (NACK) based on CRC results. All this is handled by the HARQ scheduler, hHARQScheduling.m.

- *Create Transmit Waveform.* Using the input data generated by the HARQ scheduler and the `frc` structure, `lteRMCULTool` produces an SC-FDMA modulated waveform and a populated resource grid containing the physical channels and signals.
- *Noisy Channel Modeling.* The waveform is passed through a fading channel and Additive White Gaussian Noise (AWGN) added.
- *Perform Synchronization and SC-FDMA Demodulation.* The received symbols are synchronized to account for a combination of implementation delay and channel delay spread. The symbols are then SC-FDMA demodulated.
- *Perform Channel and Noise Power Spectral Density Estimation.* The channel and noise power spectral density are estimated to aid in equalization and decoding.
- *Perform MMSE Equalization.* The channel and noise estimates are used to equalize the received PUSCH symbols.
- *Decode the PUSCH.* The recovered PUSCH symbols for all transmit and receive antenna pairs, along with a noise estimate, are demodulated and descrambled by `ltePUSCHDecode` to obtain an estimate of the received codeword.
- UL-SCH Channel Decoding._ The vector of decoded soft bits is passed to `lteULSCHDecode`; this decodes the codeword and returns the block CRC error and this is used to determine the throughput of the system. The contents of the new soft buffer, `harqProc(harqID).decState`, is available at the output of this function to be used for the next subframe. The transport block size is obtained from a lookup table of sizes for each subframe.

```
% Initialize variables used in the simulation and analysis
totalBLKCRC = zeros(numel(SNRIn), NFrames*10);   % Total block CRC vector
bitThroughput = zeros(numel(SNRIn), NFrames*10); % Total throughput vector
resultIndex = 1;        % Initialize frame counter index

for SNRdB = SNRIn

    fprintf('\nSimulating at %g dB SNR for a total %d Frame(s)', ...
        SNRdB, NFrames);

    % Calculate required AWGN channel noise
    SNR = 10^(SNRdB/20);
    N = 1/(SNR*sqrt(double(info.Nfft)))/sqrt(2.0);
    rng('default');

    % Store results for every subframe at SNR point
    bitTp = zeros(1, NFrames*10);  % Intermediate bit throughput vector
    blkCRC = zeros(1, NFrames*10); % Intermediate block CRC vector

    % Initialize state of all HARQ processes
    harqProcesses = hNewHARQProcess(frc);
    % Initialize HARQ process IDs to 1 as the first non-zero transport
    % block will always be transmitted using the first HARQ process. This
    % will be updated with the full sequence output by lteRMCULTool after
    % the first call to the function
    harqProcessSequence = 1;

    offsetused = 0;
    for subframeNo = 0:(NFrames*10-1)
```

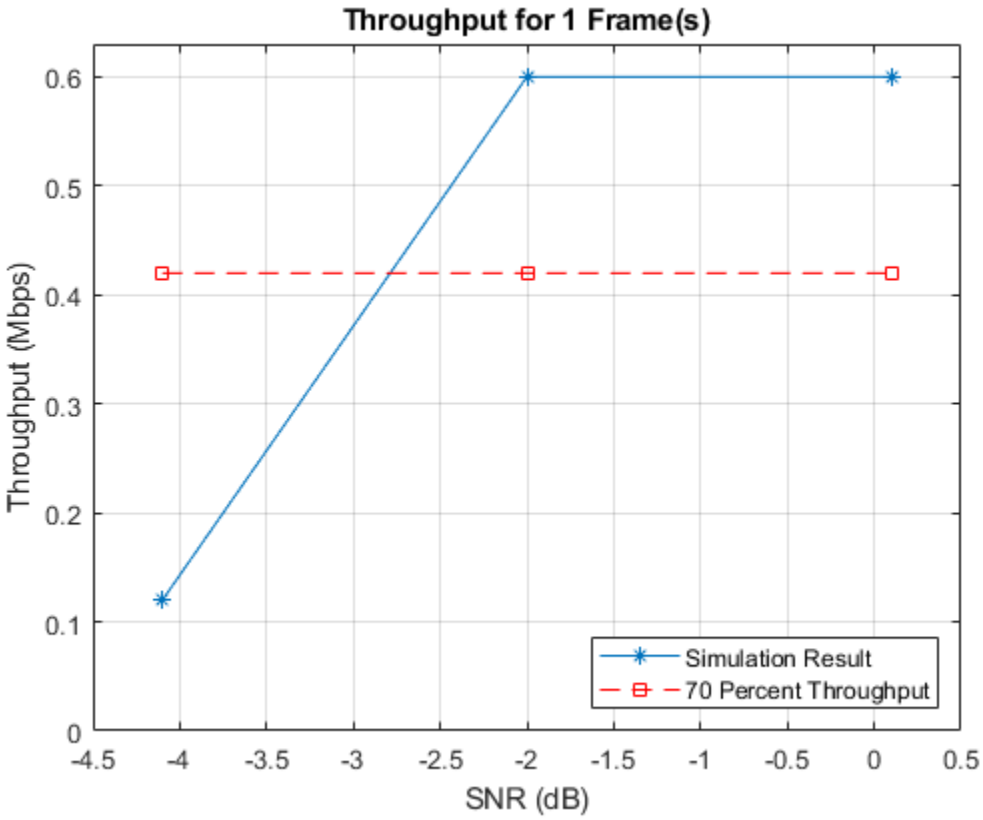
```
% Update subframe number
frc.NSubframe = subframeNo;

% Get HARQ process ID for the subframe from HARQ process sequence
harqID = harqProcessSequence(mod(subframeNo, length(harqProcessSequence))+1);

% If there is a transport block scheduled in the current subframe
% (indicated by non-zero 'harqID'), perform transmission and
% reception. Otherwise continue to the next subframe
if harqID == 0
    continue;
end

% Update current HARQ process
harqProcesses(harqID) = hHARQScheduling(...
    harqProcesses(harqID), subframeNo, rvSequence);

% Update the PUSCH transmission config with HARQ process state
frc.PUSCH = harqProcesses(harqID).txConfig;
data = harqProcesses(harqID).data;

% Create transmit waveform and get the HARQ scheduling ID sequence
% from 'frcOut' structure output which also contains the waveform
% configuration and OFDM modulation parameters
[txWaveform,~,frcOut] = lteRMCULTool(frc, data);

% Add 25 sample padding. This is to cover the range of delays
% expected from channel modeling (a combination of
% implementation delay and channel delay spread)
txWaveform =  [txWaveform; zeros(25, frc.NTxAnts)]; %#ok<AGROW>

% Get the HARQ ID sequence from 'frcOut' for HARQ processing
harqProcessSequence = frcOut.PUSCH.HARQProcessSequence;

% The initialization time for channel modeling is set each subframe
% to simulate a continuously varying channel
chcfg.InitTime = subframeNo/1000;

% Pass data through channel model
rxWaveform = lteFadingChannel(chcfg, txWaveform);

% Add noise at the receiver
v = N*complex(randn(size(rxWaveform)), randn(size(rxWaveform)));
rxWaveform = rxWaveform+v;

% Calculate synchronization offset
offset = lteULFrameOffset(frc, frc.PUSCH, rxWaveform);
if (offset < 25)
    offsetused = offset;
end

% SC-FDMA demodulation
rxSubframe = lteSCFDMADemodulate(frc, ...
    rxWaveform(1+offsetused:end, :));

% Channel and noise power spectral density estimation
[estChannelGrid, noiseEst] = lteULChannelEstimate(frc, ...
    frc.PUSCH, cec, rxSubframe);
```

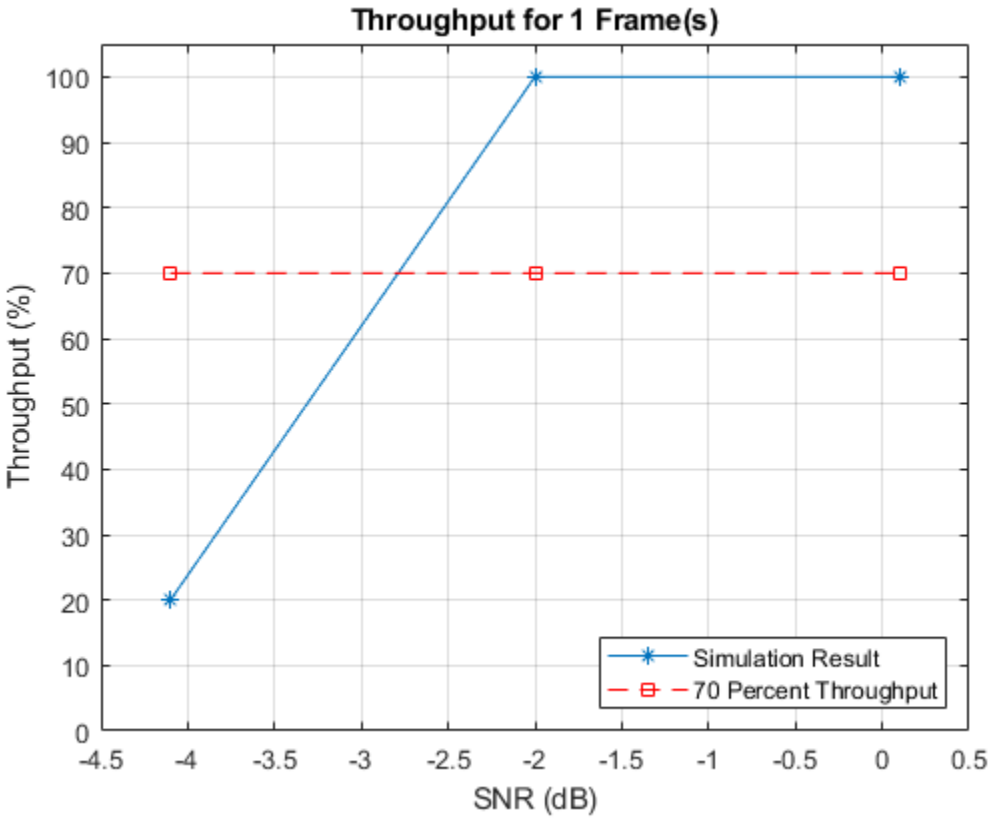
```
        % Extract REs corresponding to the PUSCH from the given subframe
        % across all receive antennas and channel estimates
        puschIndices = ltePUSCHIndices(frc, frc.PUSCH);
        [puschRx, puschEstCh] = lteExtractResources( ...
            puschIndices, rxSubframe, estChannelGrid);

        % MMSE equalization
        rxSymbols = lteEqualizeMMSE(puschRx, puschEstCh, noiseEst);

        % Update frc.PUSCH to carry complete information of the UL-SCH
        % coding configuration
        trBlkSize = trBlkSizes(mod(subframeNo, 10)+1);
        frc.PUSCH = lteULSCHInfo(frc, frc.PUSCH, trBlkSize, 'chsconcat');

        % Decode the PUSCH
        rxEncodedBits = ltePUSCHDecode(frc, frc.PUSCH, rxSymbols);

        % Decode the UL-SCH channel and store the block CRC error for given
        % HARQ process
        [rxDecodedBits, harqProcesses(harqID).blkerr, ...
            harqProcesses(harqID).decState] = lteULSCHDecode(...
            frc, frc.PUSCH, trBlkSize, ...
            rxEncodedBits, harqProcesses(harqID).decState);

        % Store the CRC calculation and total number of bits per subframe
        % successfully decoded
        blkCRC(subframeNo+1) = harqProcesses(harqID).blkerr;
        bitTp(subframeNo+1) = trBlkSize.*(1-harqProcesses(harqID).blkerr);

    end

    % Record the block CRC error and bit throughput for the total number of
    % frames simulated at a particular SNR
    totalBLKCRC(resultIndex, :) = blkCRC;
    bitThroughput(resultIndex, :) = bitTp;
    resultIndex = resultIndex + 1;
end


Simulating at -4.1 dB SNR for a total 1 Frame(s)
Simulating at -2 dB SNR for a total 1 Frame(s)
Simulating at 0.1 dB SNR for a total 1 Frame(s)
```

Display Throughput Results

The throughput results are plotted as a percentage of total capacity and actual bit throughput for the range of SNR values input using hPUSCHResults.m.

```
% Throughput calculation as a percentage
throughput = 100*(1-mean(totalBLKCRC, 2)).';

hPUSCHResults(SNRIn, NFrames, trBlkSizes, throughput, bitThroughput);
```


Throughput for 1 Frame(s)
Throughput (Mbps)
SNR (dB)
Simulation Result
70 Percent Throughput
0
0.1
0.2
0.3
0.4
0.5
0.6
-4.5
-4
-3.5
-3
-2.5
-2
-1.5
-1
-0.5
0
0.5



Appendix

This example uses these helper functions.

- hHARQScheduling.m
- hNewHARQProcess.m
- hPUSCHResults.m

Selected Bibliography

1 3GPP TS 36.104 "Base Station (BS) radio transmission and reception"

Effect of Inter-Cell Interference on PDSCH Throughput with MMSE-IRC Receiver

This example demonstrates the effect of inter-cell interference on PDSCH throughput with minimum mean square error (MMSE) and minimum mean square error - interference rejection combining (MMSE-IRC) receiver. A serving cell and two interfering eNodeBs are considered. The conditions specified in TS 36.101, Section 8.2.1.4.1B [1] are used.

Introduction

This example measures the achieved throughput for a user equipment (UE) in the serving cell with inter-cell interference from two dominant interfering cells. The serving cell uses RMC R.47 in FDD mode. The parameters for the serving and interfering cells including power levels and noise levels are described in TS 36.101, Section 8.2.1.4.1B [1].

Simulation Settings

The default simulation length is set to four frames to keep the simulation time low. Increase `NFrames` to increase the simulation time and produce statistically significant throughput results. Use the variable `eqMethod` to set the receiver equalization, which can take the values 'MMSE' and 'MMSE_IRC'.

```
NFrames = 4;                % Number of frames
eqMethod = 'MMSE_IRC';      % MMSE, MMSE_IRC
```

The signal, interferers, and noise power levels are specified in the test (TS 36.101, Section 8.2.1.4.1B [1]) using the following parameters: signal-to-interference-plus-noise ratio (SINR), dominant interferer proportion (DIP) and noise power spectral density.

```
SINR = 0.8;         % SINR in dB
DIP2 = -1.73;       % DIP in dB for cell 2
DIP3 = -8.66;       % DIP in dB for cell 3
Noc = -98;          % dBm/15kHz average power spectral density
```

The DIP characterizes each of the interfering cells and is defined as:

$$DIP2 = Ior2/(Ior2 + Ior3 + Noc)$$

$$DIP3 = Ior3/(Ior2 + Ior3 + Noc)$$

where $Ior2$ and $Io3$ are the average received power spectral density from cells 2 and 3, respectively. Noc is the average power spectral density of a white noise source (average power per resource element normalized with respect to the subcarrier spacing).

Serving eNodeB Configuration

The test considered uses reference channel R.47 in FDD mode. The parameters associated with this reference channel are specified in (TS 36.101, Table A.3.3.2.1-2 [1]). The structure `enb1` characterizes the serving cell.

```
% Set the random number generator seed
rng('default');
```

```
% Set cell 1 eNodeB configuration according to R.47
simulationParameters = struct;
simulationParameters.NDLRB = 50;
simulationParameters.CellRefP = 2;
simulationParameters.NCellID = 0;
simulationParameters.CFI = 2;
simulationParameters.DuplexMode = 'FDD';
simulationParameters.TotSubframes = 1; % This is not the total number of
% subframes used in the simulation, just the number of subframes we
% generate per call to the waveform generator.

% Specify PDSCH configuration substructure
simulationParameters.PDSCH.TxScheme = 'SpatialMux';
simulationParameters.PDSCH.Modulation = {'16QAM'};
simulationParameters.PDSCH.NLayers = 1;
simulationParameters.PDSCH.Rho = -3;
simulationParameters.PDSCH.PRBSet = (0:49)';
% Table A.3.3.2.1-2, TS 36.101
simulationParameters.PDSCH.TrBlkSizes = [8760 8760 8760 8760 8760 0 8760 8760 8760 8760];
% Table A.3.3.2.1-2, TS 36.101
simulationParameters.PDSCH.CodedTrBlkSizes = [24768 26400 26400 26400 26400 0 26400 26400 26400 2
simulationParameters.PDSCH.CSIMode = 'PUCCH 1-1';
simulationParameters.PDSCH.PMIMode = 'Wideband';
simulationParameters.PDSCH.CSI = 'On';
simulationParameters.PDSCH.W = [];
simulationParameters.PDSCH.CodebookSubset = '1111';

% Specify PDSCH OCNG configuration
simulationParameters.OCNGPDSCHEnable = 'On';             % Enable OCNG fill
simulationParameters.OCNGPDSCHPower = -3;                % OCNG power same as PDSCH Rho
simulationParameters.OCNGPDSCH.RNTI = 0;                 % Virtual UE RNTI
simulationParameters.OCNGPDSCH.Modulation = 'QPSK';      % OCNG symbol modulation
simulationParameters.OCNGPDSCH.TxScheme = 'TxDiversity'; % OCNG transmission mode 2
```

Call `lteRMCDLTool` to generate the default eNodeB parameters not specified in `simulationParameters`.

```
enb1 = lteRMCDL(simulationParameters);
```

Interfering eNodeBs Configurations

The two interfering cells are characterized by the structures `enb2` and `enb3`. These have the same field values as the serving cell (`enb1`) with the following exceptions:

- Cell Id takes the values 1 and 2 for `enb2` and `enb3`, respectively.
- The PDSCH modulation scheme is specified by the transmission mode 4 (TM4) interference model (TS 36.101, B.5.3 [1]). This value changes on a subframe-by-subframe basis and is modified in the main processing loop.

```
% Cell 2
enb2 = enb1;
enb2.NCellID = 1;
enb2.OCNGPDSCHEnable = 'Off';

% Cell 3
enb3 = enb1;
```

```
enb3.NCellID = 2;
enb3.OCNGPDSCHEnable = 'Off';
```

Propagation Channel and Channel Estimator Configurations

This section sets up the parameters for three propagation channels

- Serving cell to UE
- First interfering cell to UE
- Second interfering cell to UE

As specified in (TS 36.101, Table 8.2.1.4.1B-2 [1]) EVA5 channel conditions are used.

```
% eNodeB1 to UE propagation channel
channel1 = struct;                      % Channel config structure
channel1.Seed = 20;                     % Channel seed
channel1.NRxAnts = 2;                   % 2 receive antennas
channel1.DelayProfile ='EVA';           % Delay profile
channel1.DopplerFreq = 5;               % Doppler frequency
channel1.MIMOCorrelation = 'Low';       % Multi-antenna correlation
channel1.NTerms = 16;                   % Oscillators used in fading model
channel1.ModelType = 'GMEDS';           % Rayleigh fading model type
channel1.InitPhase = 'Random';          % Random initial phases
channel1.NormalizePathGains = 'On';     % Normalize delay profile power
channel1.NormalizeTxAnts = 'On';        % Normalize for transmit antennas
```

The channel sampling rate depends on the FFT size used in the OFDM modulator. This can be obtained using the function `lteOFDMInfo`.

```
ofdmInfo = lteOFDMInfo(enb1);
channel1.SamplingRate = ofdmInfo.SamplingRate;

% eNodeB2 (interference) to UE propagation channel
channel2 = channel1;
channel2.Seed = 122;                    % Channel seed

% eNodeB3 (interference) to UE propagation channel
channel3 = channel1;
channel3.Seed = 36;                     % Channel seed
```

The variable `perfectChanEstimator` controls channel estimator behavior. Valid values are `true` or `false`. When set to `true` a perfect channel estimate is used otherwise an imperfect estimate is used, based on the values of received pilot signals.

```
% Channel estimator behavior
perfectChanEstimator = true;

% The channel estimator configuration structure is defined below. The
% frequency and time averaging window sizes are chosen to span a relatively
% large number of resource elements. The large window sizes are chosen to
% average as much as possible noise and interference in the resource
% elements. Note that too large an averaging window in time and/or frequency
% will cause loss of information due to averaging out the channel variations.
% This produces an increasingly imperfect channel estimate which can affect
% the performance of the equalizer.
```

```
cec = struct;                           % Channel estimation config structure
cec.PilotAverage = 'UserDefined';       % Type of pilot symbol averaging
cec.FreqWindow = 31;                    % Frequency window size
cec.TimeWindow = 23;                    % Time window size
cec.InterpType = 'Cubic';               % 2D interpolation type
cec.InterpWindow = 'Centered';          % Interpolation window type
cec.InterpWinSize = 1;                  % Interpolation window size
```

Signal, Interference and Noise Power Levels

From the SINR, DIP and `Noc` values we can calculate the scaling factors to apply to the signals from the serving and interfering cells, as well as the noise level.

The function hENBscalingFactors.m calculates the scaling factors `K1`, `K2` and `K3` to apply to the channel filtered waveforms from the three cells considered. The scaling factor `No` to apply to the white Gaussian noise is also calculated. These values ensure that the signal power, interference power and noise power are as per the specified SINR and DIP values.

```
% Channel noise setup
nocLin = 10.^(Noc/10)*(1e-3); % linear  in Watts
% Take into account FFT (OFDM) scaling
No = sqrt(nocLin/(2*double(ofdmInfo.Nfft)));

% Signal and interference amplitude scaling factors calculation
[K1, K2, K3] = hENBscalingFactors(DIP2, DIP3, Noc, SINR, enb1, enb2, enb3);
```

Main Loop Initialization

Before the main processing loop we need to set up the hybrid automatic repeat request (HARQ) processes and initialize intermediate variables. A HARQ process ID sequence corresponding to the configuration is output by the `lteRMCDLTool` function. A HARQ process (with IDs 1 to 8) is associated to each subframe that has data scheduled. A value of 0 in the sequence indicates that data is not transmitted in the corresponding subframe. This can be because it is an uplink subframe or because no data is scheduled in a downlink subframe (similar to subframe 5 in this example).

```
% Initialize state of all HARQ processes
harqProcesses = hNewHARQProcess(enb1);
% Initialize HARQ process IDs to 1 as the first non-zero transport
% block will always be transmitted using the first HARQ process. This
% will be updated with the full sequence output by lteRMCDLTool after
% the first call to the function
harqProcessSequence = 1;

% Set up variables for the main loop
lastOffset = 0;       % Initialize frame timing offset from previous frame
frameOffset = 0;      % Initialize frame timing offset
nPMI = 0;             % Initialize the number of precoder matrix indication
                      % (PMI) set calculated
blkCRC = [];          % Block CRC for all considered subframes
bitTput = [];         % Number of successfully received bits per subframe
txedTrBlkSizes = [];  % Number of transmitted bits per subframes
% Vector of total number of bits transmitted, calculated for each subframe.
runningMaxThPut = [];
% Vector storing the number of successfully received bits, calculated for
% each subframe.
runningSimThPut = [];
```

```
% Obtain the number of transmit antennas.
dims = lteDLResourceGridSize(enb1);
P = dims(3);

% Assign the redundancy version sequence for each codeword and transport
% block sizes for each subframe
rvSequence = enb1.PDSCH.RVSeq;
trBlkSizes = enb1.PDSCH.TrBlkSizes;

% Set the PMI delay for the closed-loop spatial multiplexing
pmiDelay = 8; % As specified in TS 36.101, Table 8.2.1.4.1B-1

% Initialize PMIs for the first 'pmiDelay' subframes
pmiDims = ltePMIInfo(enb1,enb1.PDSCH);
txPMIs = zeros(pmiDims.NSubbands, pmiDelay);

% Flag to indicate if a valid PMI feedback is available from the UE
pmiReady = false;
```

Main Loop

The main loop iterates over the specified number of subframes. For each downlink subframe with data the following operations are performed:

- Check the HARQ processes and determine whether to send a new packet or if a retransmission is required
- Generate downlink waveforms from serving cell and interfering cells
- Filter waveforms with propagation channels and add white Gaussian noise
- Synchronize and OFDM demodulate the signal from the serving cell
- Estimate the propagation channel for the serving cell
- Equalize and decode the PDSCH
- Decode the DL-SCH
- Determine throughput performance using the block CRC obtained

```
fprintf('\nSimulating %d frame(s)\n',NFrames);

% Main for loop: for all subframes
for subframeNo = 0:(NFrames*10-1)

    % Reinitialize channel seed for each subframe to increase variability
    channel1.Seed = 1+subframeNo;
    channel2.Seed = 1+subframeNo+(NFrames*10);
    channel3.Seed = 1+subframeNo+2*(NFrames*10);

    % Update subframe number
    enb1.NSubframe = subframeNo;
    enb2.NSubframe = subframeNo;
    enb3.NSubframe = subframeNo;

    duplexInfo = lteDuplexingInfo(enb1);
```

```
if  duplexInfo.NSymbolsDL ~= 0 % target only downlink subframes

    % Get HARQ process ID for the subframe from HARQ process sequence
    harqID = harqProcessSequence(mod(subframeNo, length(harqProcessSequence))+1);
    % If there is a transport block scheduled in the current subframe
    % (indicated by non-zero 'harqID'), perform transmission and
    % reception. Otherwise, continue to the next subframe.
    if harqID == 0
        continue;
    end

    % Update current HARQ process
    harqProcesses(harqID) = hHARQScheduling( ...
        harqProcesses(harqID), subframeNo, rvSequence);

    % Extract the current subframe transport block size(s)
    trBlk = trBlkSizes(:, mod(subframeNo, 10)+1).';

    % Update the PDSCH transmission config with HARQ process state
    enb1.PDSCH.RVSeq = harqProcesses(harqID).txConfig.RVSeq;
    enb1.PDSCH.RV = harqProcesses(harqID).txConfig.RV;
    dlschTransportBlk = harqProcesses(harqID).data;

    % Set the PMI to the appropriate value in the delay queue
    if strcmpi(enb1.PDSCH.TxScheme,'SpatialMux')
        pmiIdx = mod(subframeNo, pmiDelay);      % PMI index in delay queue
        enb1.PDSCH.PMISet = txPMIs(:, pmiIdx+1); % Set PMI
    end

    % Create transmit waveform
    [tx,~,enbOut] = lteRMCDLTool(enb1, dlschTransportBlk);

    % Pad 25 samples to cover the range of delays expected from channel
    % modeling (a combination of implementation delay and channel delay
    % spread)
    txWaveform1 = [tx; zeros(25, P)];

    % Get the HARQ ID sequence from 'enbOut' for HARQ processing
    harqProcessSequence = enbOut.PDSCH.HARQProcessSequence;

    % Generate interferer model as per as per TS 36.101, B.5.3. The
    % function hTM4InterfModel generates the interferer transmit signal.
    txWaveform2 = [hTM4InterfModel(enb2); zeros(25,P)];
    txWaveform3 = [hTM4InterfModel(enb3); zeros(25,P)];

    % Specify channel time for the present subframe
    channel1.InitTime = subframeNo/1000;
    channel2.InitTime = channel1.InitTime;
    channel3.InitTime = channel1.InitTime;

    % Pass data through the channel
    rxWaveform1 = lteFadingChannel(channel1,txWaveform1);
    rxWaveform2 = lteFadingChannel(channel2,txWaveform2);
    rxWaveform3 = lteFadingChannel(channel3,txWaveform3);

    % Generate noise
    noise = No*complex(randn(size(rxWaveform1)), ...
        randn(size(rxWaveform1)));
```

```
% Add AWGN to the received time domain waveform
rxWaveform = K1*rxWaveform1 + K2*rxWaveform2 + K3*rxWaveform3 + noise;

% Receiver
% Once every frame, on subframe 0, calculate a new synchronization
% offset
if (mod(subframeNo,10) == 0)
    frameOffset = lteDLFrameOffset(enb1, rxWaveform);
    if (frameOffset > 25)
        frameOffset = lastOffset;
    end
    lastOffset = frameOffset;
end

% Synchronize the received waveform
rxWaveform = rxWaveform(1+frameOffset:end, :);

% Scale rxWaveform by 1/K1 to avoid numerical issues with
% channel decoding stages
rxWaveform = rxWaveform/K1;

% Perform OFDM demodulation on the received data to obtain the
% resource grid
rxSubframe = lteOFDMDemodulate(enb1, rxWaveform);

% Perform channel estimation
if(perfectChanEstimator)
    estChannelGrid = lteDLPerfectChannelEstimate(enb1, channel1, frameOffset);
    noiseInterf = K2*rxWaveform2 + K3*rxWaveform3 + noise;
    noiseInterf = noiseInterf/K1;
    noiseGrid = lteOFDMDemodulate(enb1, noiseInterf(1+frameOffset:end ,:));
    noiseEst = var(noiseGrid(:));
else
    [estChannelGrid, noiseEst] = lteDLChannelEstimate( ...
        enb1, enb1.PDSCH, cec, rxSubframe);
end

% Get PDSCH indices
pdschIndices = ltePDSCHIndices(enb1,enb1.PDSCH,enb1.PDSCH.PRBSet);
% Get PDSCH resource elements. Scale the received subframe by
% the PDSCH power factor Rho.
[pdschRx, pdschHest] = lteExtractResources(pdschIndices, ...
    rxSubframe*(10^(-enb1.PDSCH.Rho/20)), estChannelGrid);

% Perform equalization and deprecoding
if strcmp(eqMethod,'MMSE')
    % MIMO equalization and deprecoding (MMSE based)
    [rxDeprecoded,csi] = lteEqualizeMIMO(enb1,enb1.PDSCH,...
        pdschRx,pdschHest,noiseEst);
else
    % MIMO equalization and deprecoding (MMSE-IRC based)
    [rxDeprecoded,csi] = hEqualizeMMSEIRC(enb1,enb1.PDSCH,...
        rxSubframe,estChannelGrid,noiseEst);
end

% Perform layer demapping, demodulation and descrambling
cws = ltePDSCHDecode(enb1,setfield(enb1.PDSCH,'TxScheme',...
```

```
            'Port7-14'),rxDeprecoded); % The PDSCH transmission scheme is
        % modified into port7-14, in order to skip the deprecoding operation

        % Scaling LLRs by CSI
        cws = hCSIscaling(enb1.PDSCH,cws,csi);

        % Decode DL-SCH
        [decbits, harqProcesses(harqID).blkerr,harqProcesses(harqID).decState] = ...
            lteDLSCHDecode(enb1, enb1.PDSCH, trBlk, cws, ...
            harqProcesses(harqID).decState);

        % Store values to calculate throughput
        % Only for subframes with data and valid PMI feedback
        if any(trBlk) && pmiReady
            blkCRC = [blkCRC harqProcesses(harqID).blkerr];
            txedTrBlkSizes =  [txedTrBlkSizes trBlk];
            bitTput = [bitTput trBlk.*(1-harqProcesses(harqID).blkerr)];
        end
        runningSimThPut = [runningSimThPut sum(bitTput,2)];
        runningMaxThPut = [runningMaxThPut sum(txedTrBlkSizes,2)];

        % Provide PMI feedback to the eNodeB
        if strcmpi(enb1.PDSCH.TxScheme,'SpatialMux')
            PMI = ltePMISelect(enb1, enb1.PDSCH, estChannelGrid, noiseEst);
            txPMIs(:, pmiIdx+1) = PMI;
            nPMI = nPMI+1;
            if nPMI>=pmiDelay
                pmiReady = true;
            end
        end
    end
end
```

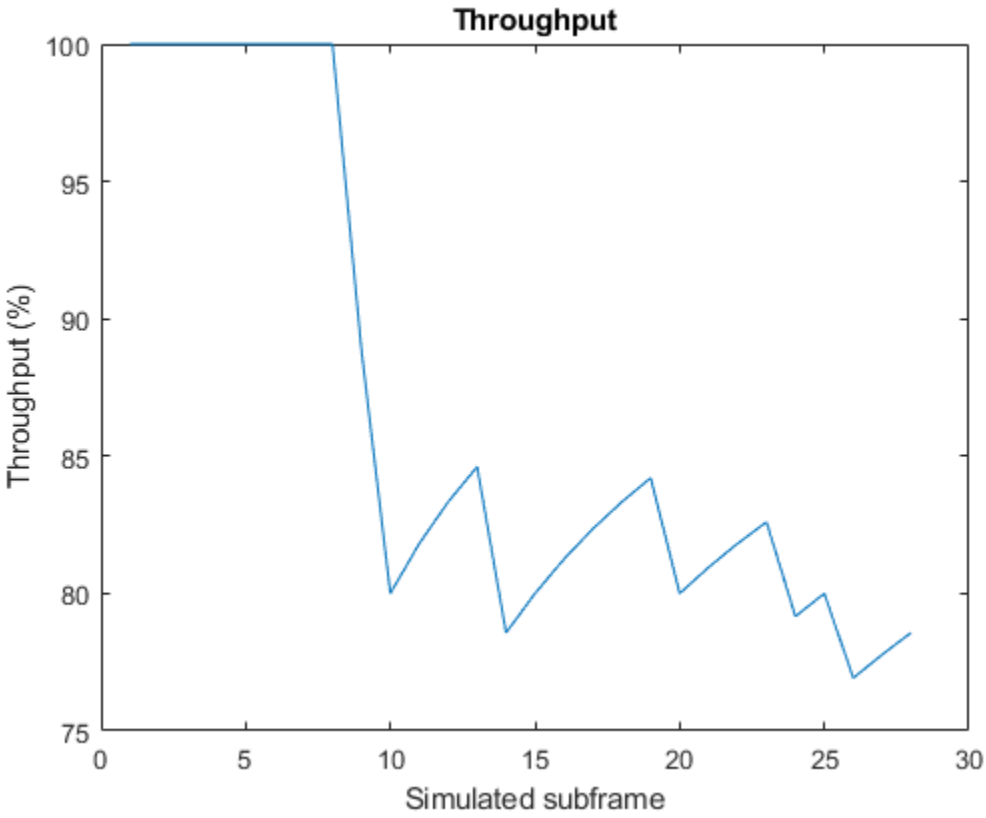
```
Simulating 4 frame(s)
```

Results

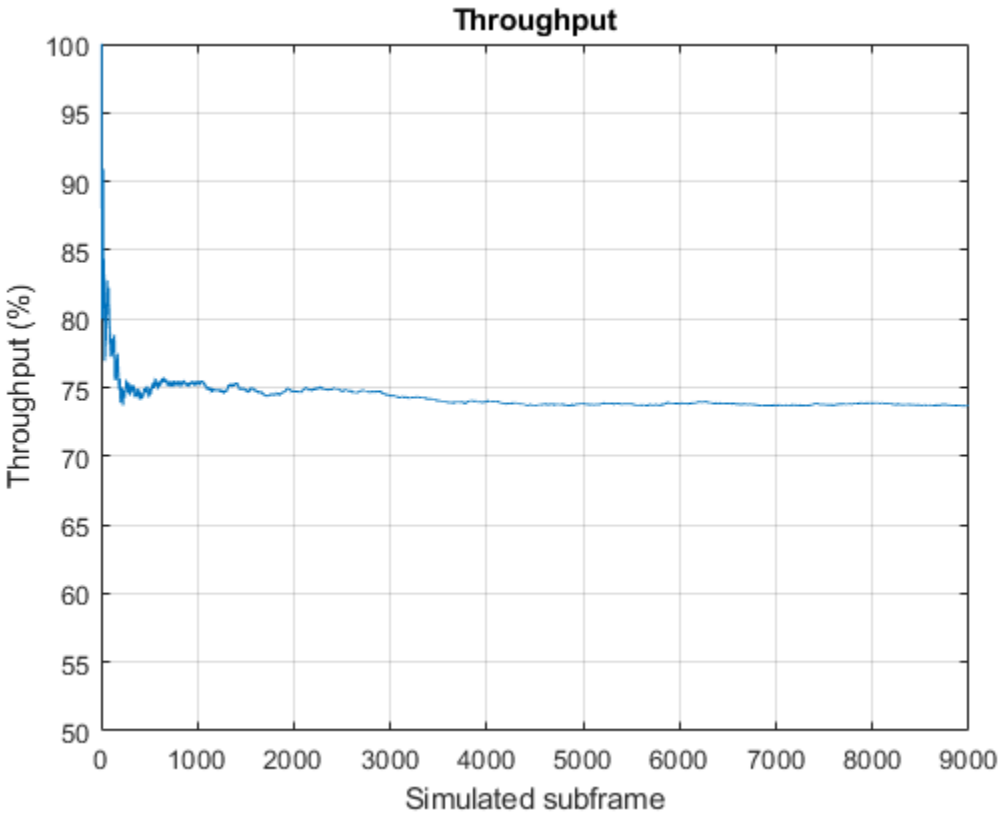
This section calculates the achieved throughput. A figure with the running measured throughput for all simulated subframes is also provided.

```
maxThroughput = sum(txedTrBlkSizes); % Maximum possible throughput
simThroughput = sum(bitTput,2);      % Simulated throughput

% Display achieved throughput percentage
disp(['Achieved throughput ' num2str(simThroughput*100/maxThroughput) '%'])

% Plot running throughput
figure;plot(runningSimThPut*100./runningMaxThPut)
ylabel('Throughput (%)');
xlabel('Simulated subframe');
title('Throughput');
```

```
Achieved throughput 78.5714%
```



For statistically valid results, the simulation should be run for a larger number of frames. The figure below shows the throughput results when simulating 1000 frames.



Appendix

This example uses the following helper functions:

- hENBscalingFactors.m
- hEqualizeMMSEIRC.m
- hTM4InterfModel.m
- hCSIscaling.m
- hNewHARQProcess.m
- hHARQScheduling.m

Selected Bibliography

1 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception"

2 3GPP TR 36.829 "Enhanced performance requirement for LTE User Equipment (UE)"

PDSCH Throughput Performance in Simulink

This example demonstrates how to measure the physical downlink shared channel (PDSCH) throughput performance in Simulink® using LTE Toolbox™ for the following transmission modes (TM):

- TM1: Single antenna (Port 0)
- TM2: Transmit diversity
- TM3: Open loop codebook based precoding: Cyclic Delay Diversity (CDD)

Introduction

The model in this example measures the throughput of an end-to-end PDSCH simulation. The Simulink model can operate under TM1, TM2 and TM3 transmission modes without implementing hybrid automatic repeat request (Hybrid ARQ). For an equivalent simulation in MATLAB®, see the example "PDSCH Throughput Conformance Test for Single Antenna (TM1), Transmit Diversity (TM2), Open Loop (TM3) and Closed Loop (TM4/6) Spatial Multiplexing" on page 2-404, which additionally covers TM4/6 and enables Hybrid ARQ. For information on modeling TM7, TM8, TM9 and TM10, see the following MATLAB example: "PDSCH Throughput for Non-Codebook Based Precoding Schemes: Port 5 (TM7), Port 7 or 8 or Port 7-8 (TM8), Port 7-14 (TM9 and TM10)" on page 2-424.

The example works on a *subframe by subframe* basis. For each subframe, the model generates and OFDM modulates a populated resource grid to create a transmit waveform. The generated waveform is then passed through a noisy fading channel. The receiver then performs channel estimation, equalization, demodulation, and decoding. The block CRC result at the output of the channel decoder is used to determine the throughput performance of the PDSCH.

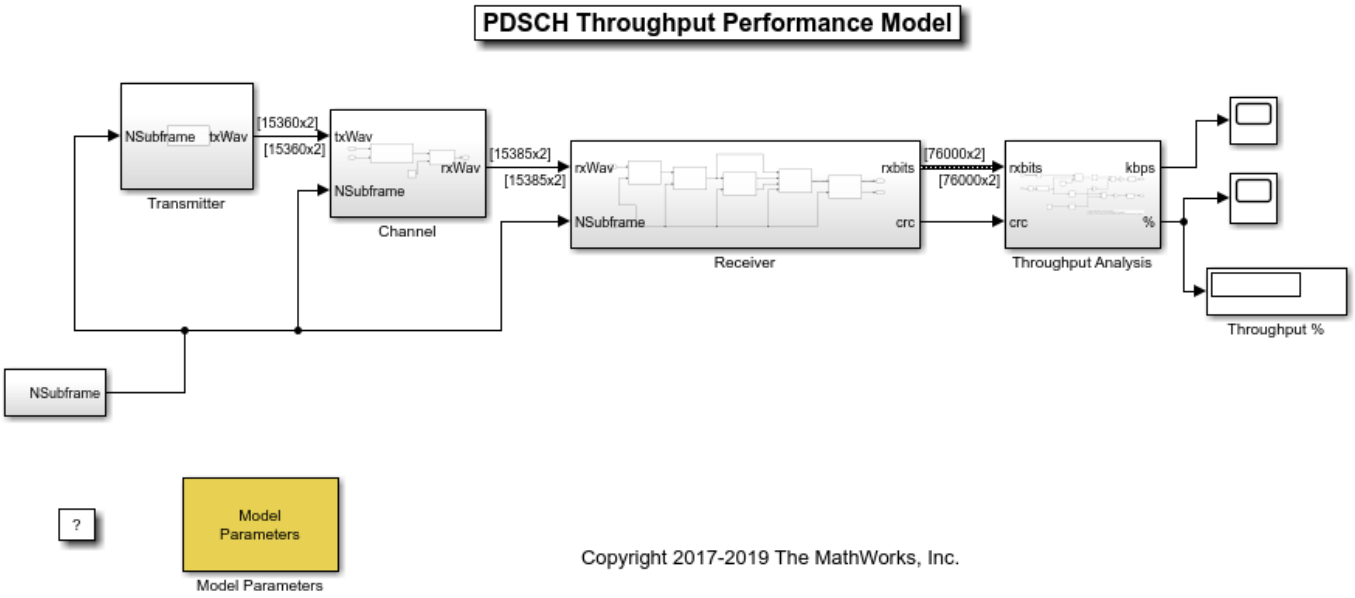
The MATLAB Function block enables the use of MATLAB functions in a Simulink model. In this example, the end-to-end simulation is modeled in Simulink by using MATLAB Function blocks to call LTE Toolbox functions. The `coder.extrinsic` (MATLAB Coder) construct is used at the top of every MATLAB Function block to declare the function extrinsic during simulation. This construct enables you to call MATLAB functions in Simulink that do not support code generation.

Structure of the Model

The model has four main parts:

- Transmitter: Generates random codewords and a populated resource grid, which is OFDM modulated to create a transmit waveform.
- Channel: Filters the transmitted waveform through a multipath Rayleigh fading channel with AWGN.
- Receiver: Recovers the transmitted sequence of bits by performing synchronization, channel estimation, equalization, demodulation, and decoding.
- Throughput analysis: Calculates the throughput performance with the block CRC decoding result.

Finally, the Model Parameters block enables you to vary the most common parameters for the simulation, channel modeling, and channel estimation.

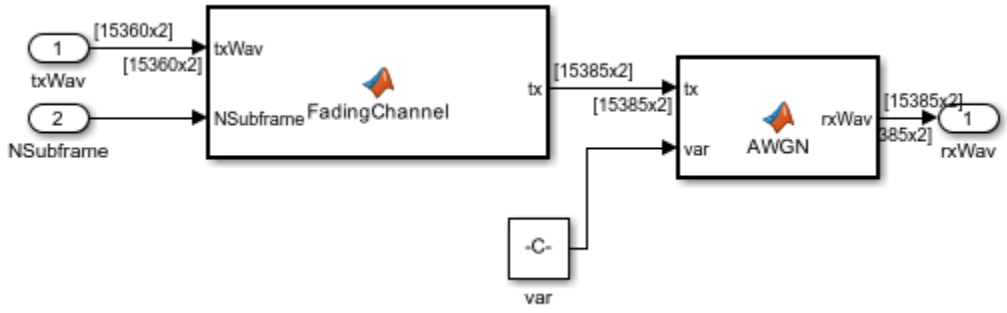


Transmitter

The Transmitter block creates one or two random codewords with information bits, depending on the transmission mode. Then the call to the `lteRMCDLTool` function produces an *OFDM modulated* waveform from the information bits. This waveform contains the physical channels and signals. Since the model does not specify an RMC, all downlink subframes are scheduled.

Channel

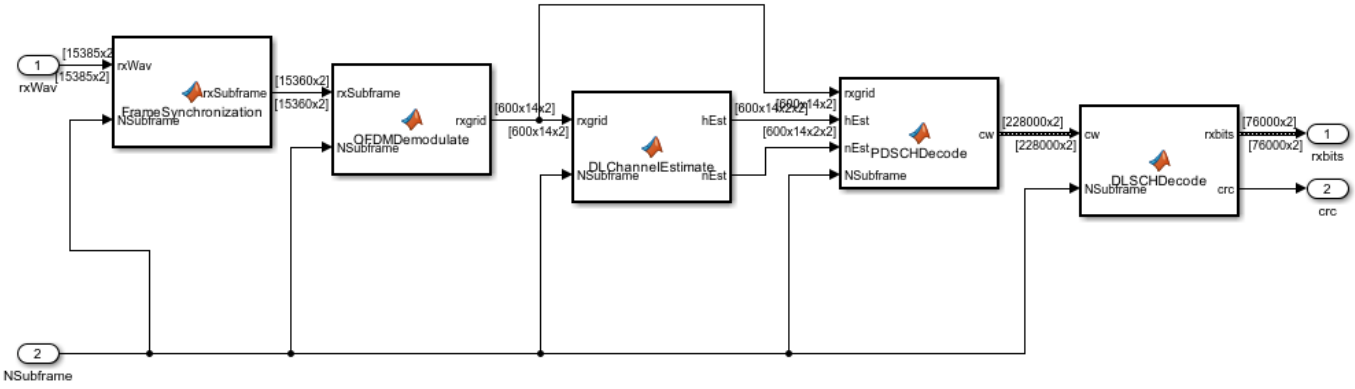
In the channel subsystem, the waveform is passed through a multipath Rayleigh fading channel and AWGN noise is added. The noise power is controlled by varying the *signal to noise* ratio (SNR) parameter. The parameters of the fading process can be controlled from the Channel Parameters tab in the Model Parameters block.



Receiver

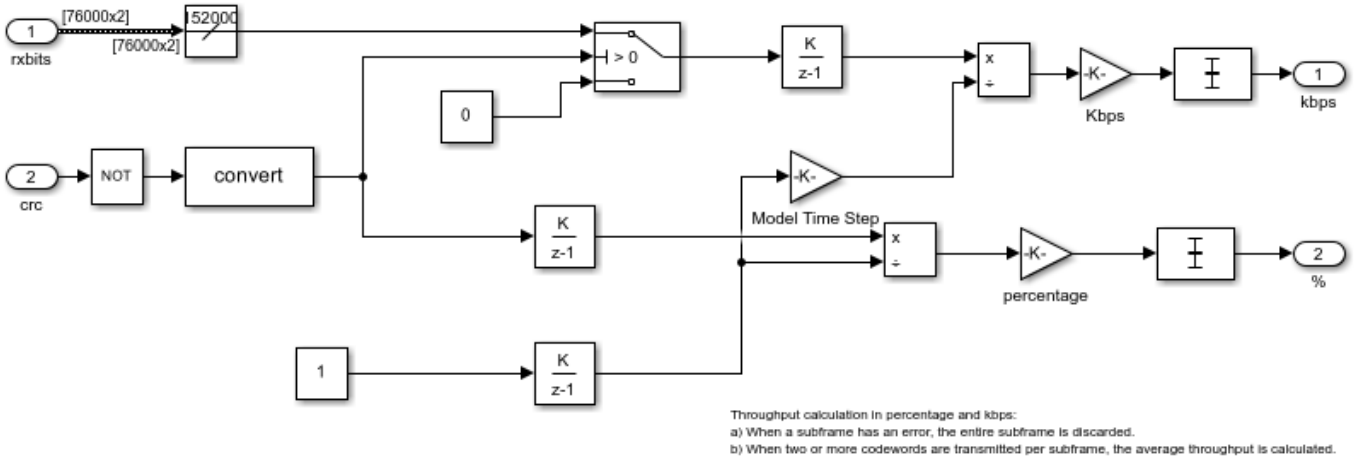
The receiver recovers the PDSCH data from the channel and computes the block CRC. This process consists of the following steps:

1. Synchronization: The received symbols are offset to account for a combination of implementation delay and channel delay spread.
2. OFDM Demodulation: The received symbols are OFDM demodulated.
3. Channel Estimation: The channel response and noise levels are estimated. These estimates are used to decode the PDSCH.
4. PDSCH Decoding: The recovered PDSCH symbols for each transmit antenna from the received grid, along with a channel estimate, are demodulated and descrambled to obtain an estimate of the received codewords.
5. Downlink Shared Channel (DL-SCH) decoding and block CRC error computing: The vector of decoded soft bits is passed to `lteDLSCHDecode`. This function decodes the codeword and returns the block CRC error used to determine the throughput of the system.



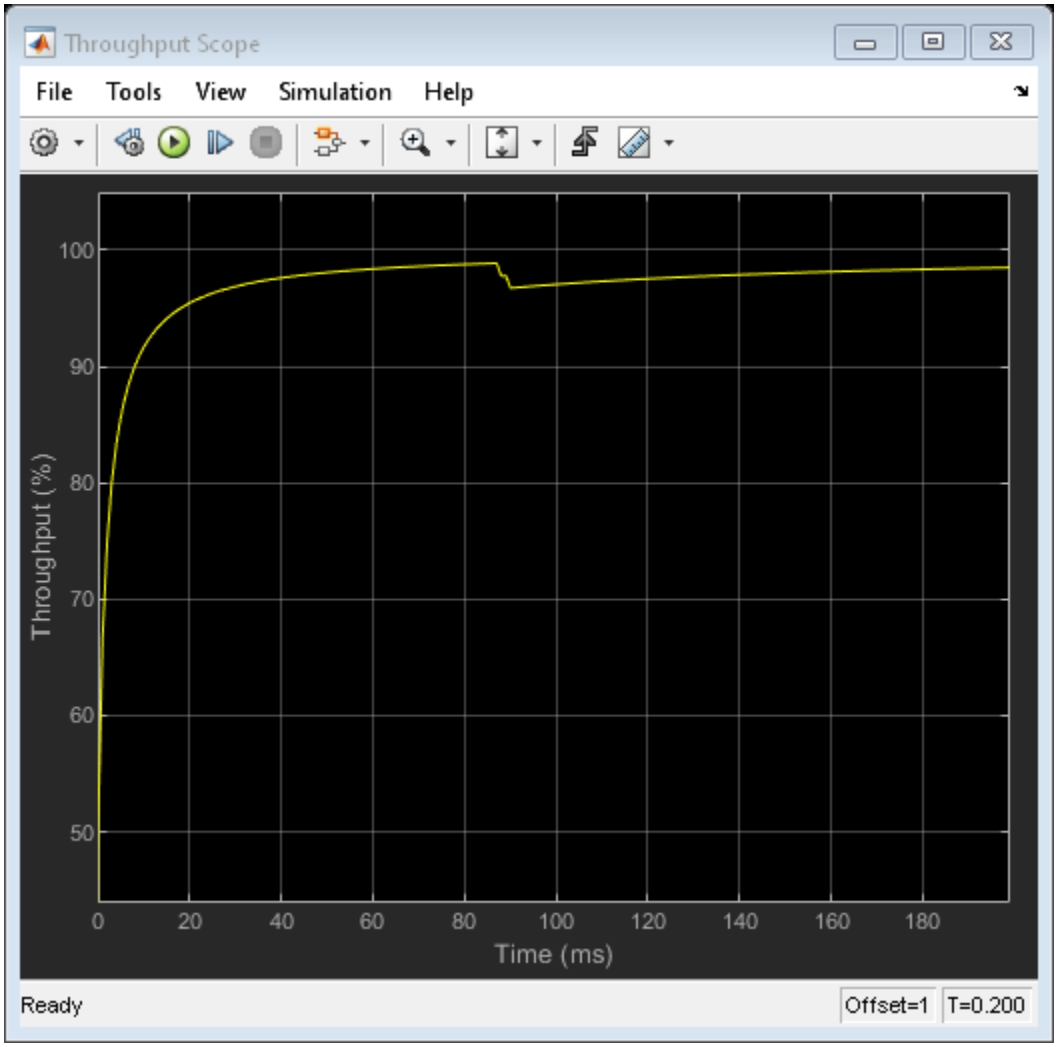
Throughput Analysis

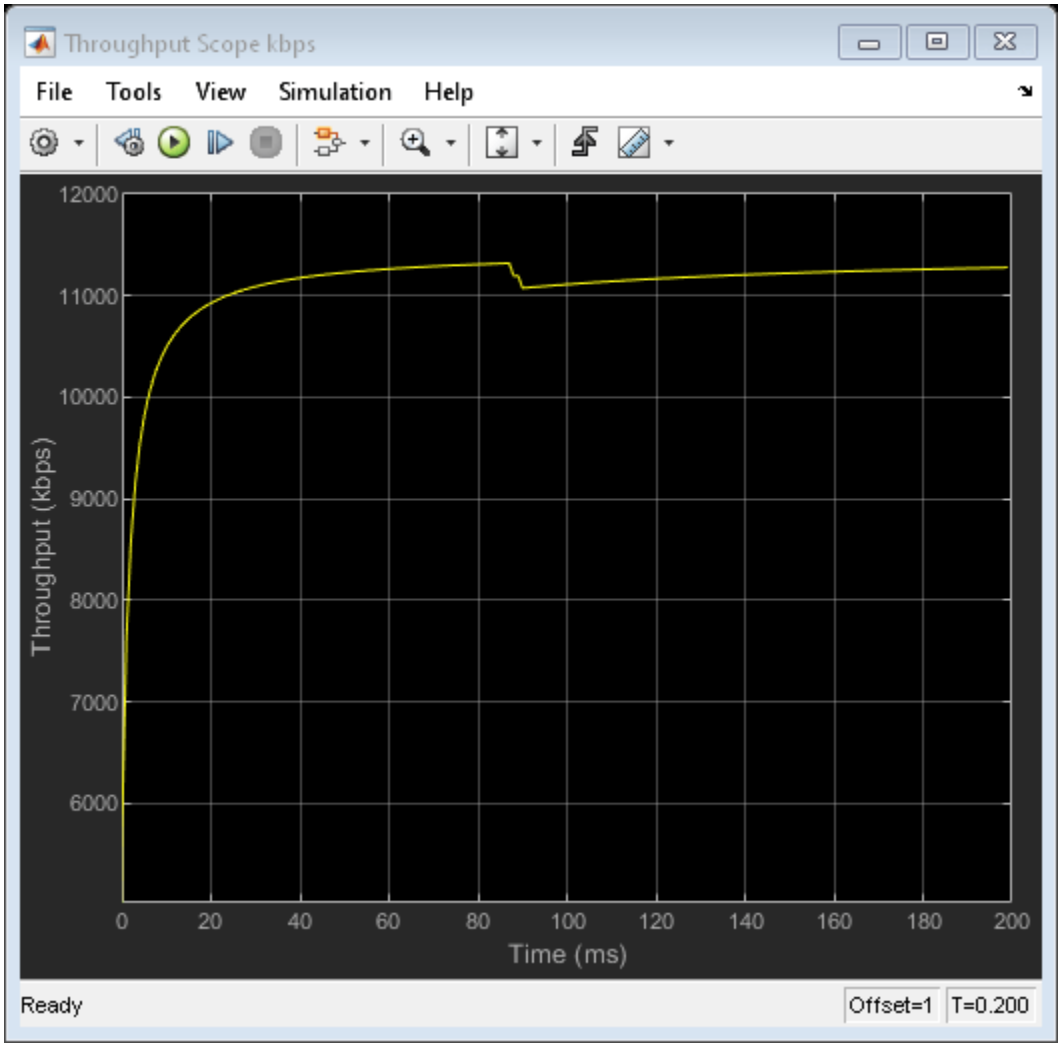
In this block, the throughput performance of the link is calculated, both in kbps and in percentage, by using the block CRC result from the receiver.



Results and Displays

When the simulation is run, the throughput per codeword is displayed both in percentage and in kbps. To obtain representative results, run the simulation long enough for the throughput results to reach a steady state. By default, 20 frames (0.2 s of simulation time) are simulated.





Exploring the Example

Try changing the SNR and the transmission mode in the Model Parameters block. Decreasing the SNR will decrease the throughput of the system since more subframes will be lost. Simulating different TMs will result in different throughput in kbps and percentage for the same SNR.

You can also try changing several channel parameters, such as the number of receive antennas or the MIMO correlation. Other parameters to try changing include the channel estimator parameters, such as the window type and window size.

PDCCH Conformance Test

Example showing how to generate the PDCCH/PCFICH RMCs defined in TS 36.101 and use in conformance tests.

Introduction

This example shows the demodulation performance of PDCCH/PCFICH and determination of the probability of miss-detection of the downlink scheduling grant (Pm-dsg) test as defined in TS 36.101/TS 36.521-1 Section 8.4. The PDCCH and PCFICH are tested jointly and a miss-detection of PCFICH implies a miss-detection of PDCCH. The 1x2 transmit/receive antenna port performance test in TS 36.101 Section 8.4.1.1 Table 8.4.1.1-1[1] for R.15 PDCCH/PCFICH reference channel (FDD duplex mode) defines a required Pm-dsg of 1% at an SNR of -1.7 dB.

The LTE Toolbox™ allows configuration of the downlink waveform generator, `lteRMCDL`, for PDCCH conformance tests. Working on a frame by frame basis when simulating at a particular SNR, a populated resource grid is generated and OFDM modulated to create a transmit waveform. The generated waveform is passed through a noisy Extended Typical Urban (ETU) channel. Channel estimation, equalization, demodulation and decoding are performed at receiver. The average miss-detection probability of the PDCCH/PCFICH is determined using the combination of the received CFI and received DCI message.

Simulation Setup

The example is executed for a simulation length of 1 frame to keep the simulation time low. SNR values include -1.7 dB specified for the R.15 RMC single port test. A large number of frames, `NFrames`, should be used to produce meaningful results. `SNRIn` can be an array of values or a scalar.

```
nFrames = 1;                       % Number of frames to simulate
snrIn = [-3.1 -2.4 -1.7 -1 -0.3]; % SNR points to simulate
```

PDCCH/PCFICH Configuration

The downlink waveform generator `lteRMCDLTool` generates the PDSCH, PDCCH, PCFICH and other physical channels and signals. Here we perform the conformance testing for the R.15 FDD RMC by configuring the PCFICH and PDCCH channels and generating the waveform. The parameters for the generator are specified via a structure `rmc`. The `lteRMCDL` function generates PDSCH RMC configurations including associated PDCCH/PCFICH and we can update the PDCCH/PCFICH parameters to generate the PDCCH/PCFICH RMCs. Here we start by using a PDSCH RMC that closely matches the PDSCH setup mentioned in TS 36.521-1 Section 8.4.1.1.4 [2] for the R.15 FDD RMC and then setting the DCI and PDCCH parameters. No data and control is transmitted on subframe 5. Therefore it is not decoded and does not count towards the missed detection calculation.

```
% Setup the PDSCH according to TS 36.521-1 Table A.3.5.1-2 for single port
% which corresponds to R.2 FDD PDSCH RMC
rmc = lteRMCDL('R.2','FDD');

% Setup DCI and PDCCH according to TS 36.101 Section 8.4.1.1 for R.15 FDD
% RMC which requires the CFI, DCI format, HICH group multiplier,
% aggregation level and power to be set as defined for the test
rmc.CFI = 2;                      % OFDM symbols for PDCCH
rmc.Ng = 'One';                   % HICH group multiplier
rmc.PDSCH.DCIFormat = 'Format1'; % Set the DCI Format
rmc.PDSCH.PDCCHFormat = 3;        % Set the aggregation level to be 8
```

```
rmc.PDSCH.PDCCHPower = 0;        % Relative power is 0dB for single port

% Setup the OCNG to fill all unused data and control region REs
rmc.OCNGPDSCHEnable = 'On';
rmc.OCNGPDSCHPower = 0;
rmc.OCNGPDCCHEnable = 'On';
rmc.OCNGPDCCHPower = 0;
```

Propagation Channel Model Configuration

```
cfg = struct;                    % Initialize channel config structure
cfg.Seed = 6;                    % Channel seed
cfg.NRxAnts = 2;                 % 2 receive antennas
cfg.DelayProfile ='ETU';         % Delay profile
cfg.DopplerFreq = 70;            % Doppler frequency in Hz
cfg.MIMOCorrelation = 'Low';     % Multi-antenna correlation
cfg.NTerms = 16;                 % Oscillators used in fading model
cfg.ModelType = 'GMEDS';         % Rayleigh fading model type
cfg.InitPhase = 'Random';        % Random initial phases
cfg.NormalizePathGains = 'On';   % Normalize delay profile power
cfg.NormalizeTxAnts = 'On';      % Normalize for transmit antennas
```

Channel Estimator Configuration

The variable `perfectChannelEstimator` controls channel estimator behavior. Valid values are `true` or `false`. When set to `true` a perfect channel estimate is used otherwise an imperfect estimate is used, based on the values of received pilot signals.

```
% Channel estimator behavior
perfectChannelEstimator = false;
```

The imperfect channel estimator is configured with a structure `cec`. An ETU delay profile causes the channel to change quickly over time. Therefore only time averaging is performed over pilot estimates by setting the frequency window to 1 Resource Element (RE). A Doppler frequency of 70 Hz causes the channel fading to be slow enough to allow a large time averaging window. Pilots in the whole frame are included in the average by setting the time window to 31 REs.

```
% Configure channel estimator
cec.PilotAverage = 'UserDefined';   % Type of pilot symbol averaging
cec.FreqWindow = 1;                 % Frequency window size in REs
cec.TimeWindow = 31;                % Time window size in REs
cec.InterpType = 'Cubic';           % 2D interpolation type
cec.InterpWindow = 'Centered';      % Interpolation window type
cec.InterpWinSize = 3;              % Interpolation window size
```

Processing

For each SNR point the following operations are performed for each frame:

- *RMC Generation and OFDM Modulation*: Generate the OFDM modulated waveform as defined for the PDCCH/PCFICH RMC.
- *Propagation Channel Model*: The OFDM modulated waveform is transmitted through the propagation channel. The output of the channel `rxWaveform` has two columns, one per receive antenna.
- *Add Channel Noise*: The channel noise is modeled by AWGN.

- *Receiver Synchronization and OFDM Demodulation*: Determine the delay suffered during propagation. This is calculated by calling `lteDLFrameOffset`, which uses the primary and secondary synchronization signals. OFDM demodulation is performed after synchronization.
- *Channel Estimation*: Provides an estimate of the channel response at each element of the resource grid for a transmit/receive antenna pair. This estimate is then used to remove the effect of the channel on the transmitted signal. The channel estimation also aims to reduce the channel noise experienced during transmission by averaging the reference signals (pilot symbols).
- *Miss-Detection Measurement*: PCFICH and PDCCH data is analyzed on a subframe by subframe basis. A specific frame worth of data for all receive antennas is extracted from the array of received grids, the same defined frame worth of information is extracted from the estimate of channel response for all transmit and receive antenna pairs. These, along with the noise estimate, are input into the function `ltePCFICHDecode`, which decodes the CFI data. If the CFI is correctly detected, the PCFICH reception is successful and the next stage is to receive the DCI. Using the channel estimate and noise estimate, the PDCCH message is recovered using the `ltePDCCHDecode` function; blind search is then performed on this decoded bits to recover the DCI. If the DCI message is also correctly received, the transmission is successful and used to determine the detection of the system.

```
% dims is a three element vector [K L P]: where K = no of subcarriers,
% L = no of OFDM symbols and P = no of transmit antennas
dims = lteDLResourceGridSize(rmc);  % Determine resource grid dimensions
L = dims(2);                        % Number of OFDM symbols per subframe
P = dims(3);                        % Number of transmit antennas

% Initialize the variables used in the simulation and analysis
resultIndex = 1;                    % Initialize SNR index
initOffsets = 0;                    % Initialize frame offset

% Set the random number generator to default value
rng('default');

% Number of subframes carrying transport blocks with data and control
% within a frame. For RMC R.15 this should be 9, as out of the 10 subframes
% in a frame subframe 5 does not carry any data or control
nDataTBSPerFrame = sum(rmc.PDSCH.TrBlkSizes(:) ~= 0);

% Total Pmdsg vector
totalPmdsg = zeros(numel(snrIn),nDataTBSPerFrame*nFrames);

for snrdb = snrIn
    fprintf('\nSimulating at %gdB SNR for a total %d Frame(s)\n',...
    snrdb,nFrames);

    % Initialize result vector for each SNR point
    totalPmdsgSNR = zeros(nFrames,nDataTBSPerFrame);% Total block CRC

    % Initialize offset vector for each frame
    offsets = initOffsets;

    for FrameNo = 1:nFrames

        % Set the subframe number
        rmc.NSubframe = 0;

        % Generate the waveform
```

```
[txWaveform,txGrid,info] = lteRMCDLTool(rmc,[1 0 0 1]);

% Initialize result store for frame
pmdsg = zeros(nDataTBSPerFrame,1);
dataSubframeIndex = 1;

% Set sampling rate of channel to that of OFDM modulation
cfg.SamplingRate = info.SamplingRate;

% Set channel offset to current frame (1 frame = 10ms)
cfg.InitTime = (FrameNo-1)*(rmc.TotSubframes)/1000;

% Pass data through the fading channel model.
% An additional 25 samples are added to the end of the waveform.
% These are to cover the range of delays expected from the channel
% modeling (a combination of implementation delay and channel
% delay spread).
rxWaveform = lteFadingChannel(cfg,[txWaveform ; zeros(25,P)]);

% Noise setup
SNR = 10^((snrdb-rmc.PDSCH.PDCCHPower)/20);    % Linear SNR

% Normalize noise power to take account of sampling rate, which is
% a function of the IFFT size used in OFDM modulation, and the
% number of antennas
N0 = 1/(sqrt(2.0*rmc.CellRefP*double(info.Nfft))*SNR);

% Create additive white Gaussian noise
noise = N0*complex(randn(size(rxWaveform)), ...
                   randn(size(rxWaveform)));

% Add AWGN to the received time domain waveform
rxWaveform = rxWaveform + noise;

% Perform receiver synchronization
offset = lteDLFrameOffset(rmc,rxWaveform);

% Determine if frame synchronization was successful
if (offset > 25)
    offset = offsets(end);
else
    offsets = [offsets offset]; %#ok
end
if (offset>0)
    rxWaveform = rxWaveform(1+offset:end,:);
end

% Perform OFDM demodulation on the received data to recreate the
% resource grid
rxGrid = lteOFDMDemodulate(rmc,rxWaveform);

% Channel estimation
if(perfectChannelEstimator)
    estChannelGrid = lteDLPerfectChannelEstimate(rmc, cfg, offset);  %#ok<UNRCH>
    n = lteOFDMDemodulate(rmc, noise(1+offset:end ,:));
    noiseest = var(n(:));
else
    [estChannelGrid, noiseest] = lteDLChannelEstimate( ...
```

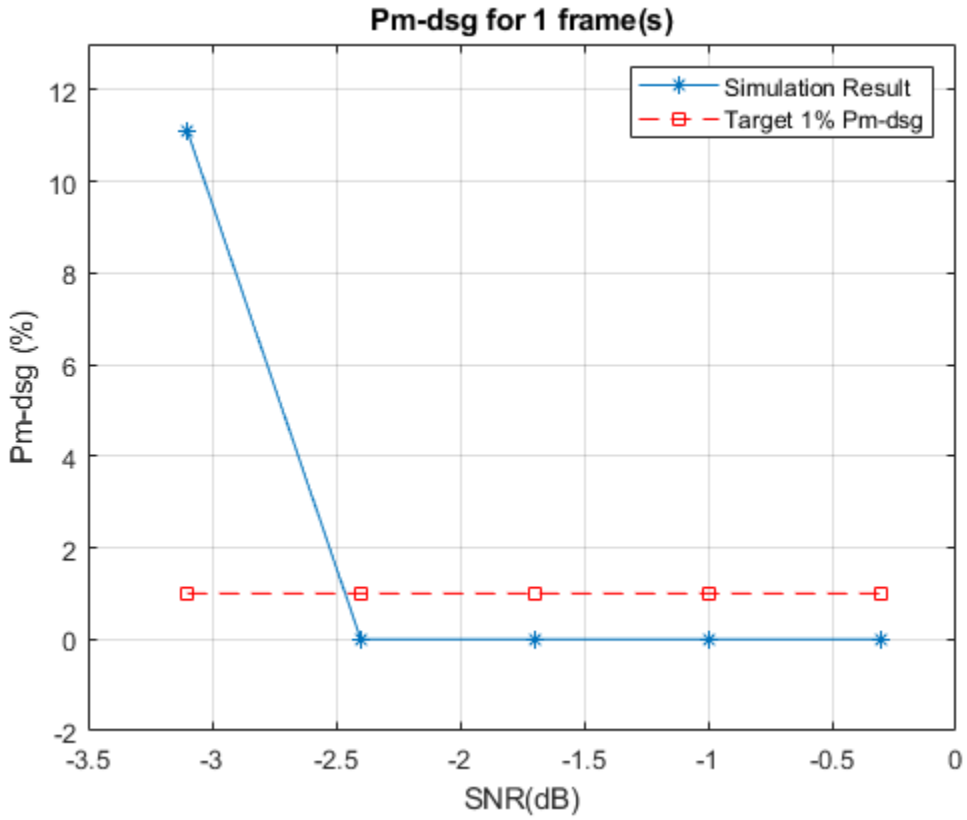
```
        rmc, cec, rxGrid);
end

% Process subframes 0 to 9 within received frame. Test for PCFICH
% and if successful, test for PDCCH. If both PCFICH and PDCCH are
% successfully received, the transmission is successful
for sf = 0:rmc.TotSubframes-1

    % Increment subframe number for correct decoding of data
    rmc.NSubframe = mod(sf,10);

    % No data is transmitted in subframe 5, so skip this
    if(rmc.NSubframe ~= 5)

        % Extract one subframe for analyzing at a time from the
        % received grid
        rxSubframe = rxGrid(:,L*sf+1:L*(sf+1),:);

        % Extract the estimated channel response for the subframe
        % being analyzed
        chSubframe = estChannelGrid(:,L*sf+1:L*(sf+1),:,:);

        % Perform PCFICH decoding of received data using the estimate
        % of the channel
        pcfichIndices = ltePCFICHIndices(rmc);
        [rxPcfichSym,pcfichHestSym] = lteExtractResources(pcfichIndices,rxSubframe,chSub
        rxPcfich = ltePCFICHDecode(rmc,rxPcfichSym,pcfichHestSym,noiseest);
        rxCFI = lteCFIDecode(rxPcfich);

        if rxCFI ~= rmc.CFI
            rxFail = 1;
        else
           % CFI decoded fine, now check if DCI can be decoded

           % Extract and decode PDCCH bits
           pdcchIndices = ltePDCCHIndices(rmc);
           [rxPdcchSym,pdcchHestSym] = lteExtractResources(pdcchIndices,rxSubframe,chSub
           rxPdcchBits = ltePDCCHDecode(rmc,rxPdcchSym,pdcchHestSym,noiseest);

           % PDCCH blind search, demask PDCCH candidate using RNTI
           ueConfig.RNTI = rmc.PDSCH.RNTI;
           ueConfig.ControlChannelType = 'PDCCH';
           ueConfig.EnableCarrierIndication = 'Off';
           ueConfig.SearchSpace = 'UESpecific';
           ueConfig.EnableMultipleCSIRequest = 'Off';
           ueConfig.EnableSRSRequest = 'Off';
           ueConfig.NTxAnts = 1;
           [rxDCI,rxDCIBits] = ltePDCCHSearch(rmc,ueConfig,rxPdcchBits);

           if ~isempty(rxDCI) && isfield(rxDCI{1},'DCIFormat') && strcmpi(rxDCI{1}.DCIFo
               rxFail = 0;
           else
               rxFail = 1;
           end

        end
        % Store the values and increment the subframe index
        pmdsg(dataSubframeIndex, :) = rxFail;
```

```
                dataSubframeIndex = dataSubframeIndex + 1;
            end
        end
        % Store resulting pmdsg results
        totalPmdsgSNR(FrameNo,:) = pmdsg(:);

    end
    % Record the missed detection for the total number of frames simulated
    % at a particular SNR
    totalPmdsg(resultIndex,:) = totalPmdsgSNR(:); % CRC
    resultIndex = resultIndex + 1;

end
```

```
Simulating at -3.1dB SNR for a total 1 Frame(s)

Simulating at -2.4dB SNR for a total 1 Frame(s)

Simulating at -1.7dB SNR for a total 1 Frame(s)

Simulating at -1dB SNR for a total 1 Frame(s)

Simulating at -0.3dB SNR for a total 1 Frame(s)
```

Plot Results

```
% Pm-dsg should be less than 1% at -1.7 dB SNR for single port
pmdsgVal = 100*mean(totalPmdsg,2);
figure
plot(snrIn,pmdsgVal,'-*')
title(['Pm-dsg for ', num2str(nFrames) ' frame(s)'] )
xlabel('SNR(dB)'); ylabel('Pm-dsg (%)');
grid on;
hold on;
plot(snrIn, ones(1,numel(snrIn)),'--rs');
legend('Simulation Result','Target 1% Pm-dsg','Location',...
       'NorthEast');
ylim([fix(min(pmdsgVal)-2) fix(max(pmdsgVal)+2)]);
```



Further Exploration

You can modify parts of this example to generate other PDCCH/PCFICH RMCs (as per TS 36.101 Section 8.4 [1] and 36.521-1 Section 8.4 [2]). For example, to generate the R.16 FDD RMC from TS 36.101 Section 8.4.2.2.1 [1] which is defined for transmit diversity performance with 2x2 transmit/receive antenna configuration, update the `rmc` structure as below:

```
rmc.CellRefP = 2;                   % Number of transmit antenna ports
rmc.CFI = 2;                        % OFDM symbols for PDCCH
rmc.Ng = 'One';                     % HICH group multiplier
rmc.PDSCH.DCIFormat = 'Format2';    % Set the DCI Format
rmc.PDSCH.PDCCHFormat = 2;          % Set the aggregation level to be 4
rmc.PDSCH.PDCCHPower = -3;          % Relative power is -3dB
rmc.PDSCH.TxScheme = 'TxDiversity'; % PDSCH transmission scheme
rmc.PDSCH.NLayers = 2;              % Number of transmission layers
rmc.OCNGPDSCH.TxScheme = 'TxDiversity'; % PDSCH OCNG transmission scheme
rmc.OCNGPDCCHPower = -3;            % Relative power is -3dB for PDCCH OCNG
```

Selected Bibliography

1 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception"

2 3GPP TS 36.521-1 "User Equipment (UE) conformance specification; Radio transmission and reception; Part 1: Conformance testing"

Reporting of Channel Quality Indicator (CQI) Conformance Test

This example demonstrates how to measure the Channel Quality Indicator (CQI) reporting performance using the LTE Toolbox™ under conformance test conditions as defined in TS36.101 Section 9.3.2.1.1.

Introduction

This example highlights the use of the `lteCQISelect` function which provides estimation of the CQI. The performance of the CQI estimation is also tested. This example provides a testbench which shows that the LTE Toolbox can satisfy the CQI reporting performance test defined in TS36.101 Section 9.3.2.1.1 [1]. The performance requirements of the test are as follows:

- a CQI index not in the set {median CQI - 1, median CQI + 1} shall be reported at least 20% of the time;
- the ratio of the throughput obtained when transmitting the transport format indicated by each reported wideband CQI index and that obtained when transmitting a fixed transport format configured according to the wideband CQI median shall be >= 1.05;
- when transmitting the transport format indicated by each reported wideband CQI index, the average BLER for the indicated transport formats shall be greater than or equal to 0.02.

This example tests that these requirements are met.

Simulation Configuration

The example is executed for a simulation length of 10 frames at an SNR of 6.0dB. A large number of `NFrames` should be used to produce meaningful results.

```
NFrames = 10;
SNRdB = 6.0;
```

eNodeB Configuration

eNodeB settings are specified in a structure `enb`. This includes a substructure `PDSCH` to configure the PDSCH according to the conformance test requirements: HARQ is disabled by setting the RV sequence to zero and the value of `CSIMode` is configured according to TS36.101 Table 9.3.2.1.1-1 [1].

```
enb = struct('RC','R.3');          % Set up parameters of RMC R.3
enb = lteRMCDL(enb);
enb.CFI = 3;                       % Reconfigure Control Format Indicator
enb.OCNGPDSCHEnable = 'On';        % Enable OCNG for unallocated PDSCH REs
enb.TotSubframes = 1;              % Reconfigure for a single subframe
enb.PDSCH.RVSeq = 0;               % Disable HARQ
enb.PDSCH.CSIMode = 'PUCCH 1-0';  % Configure the CSI reporting mode
enb.PDSCH.CSI = 'On';              % CSI scaling of soft bits
```

Propagation Channel Model Configuration

The structure, `channel`, contains the channel model configuration parameters.

```
channel.Seed = 10;                  % Channel seed
channel.NRxAnts = 2;                % 2 receive antennas
```

```
channel.DelayProfile = 'EPA';       % Delay profile
channel.DopplerFreq = 5.0;          % Doppler frequency
channel.MIMOCorrelation = 'High';   % Multi-antenna correlation
channel.ModelType = 'GMEDS';        % Rayleigh fading model type
channel.NormalizeTxAnts = 'On';     % Normalize for transmit antennas
channel.NormalizePathGains = 'On';  % Normalize delay profile power
channel.InitPhase = 'Random';       % Random initial phases
channel.NTerms = 16;                % Oscillators used in fading model

 % Set channel model sampling rate
ofdmInfo = lteOFDMInfo(enb);
channel.SamplingRate = ofdmInfo.SamplingRate;
```

Channel Estimator Configuration

The channel estimator is configured with a structure `cec`. The variable `perfectChanEstimator` controls channel estimator behavior. Valid values are `true` or `false`. When set to `true` a perfect channel estimator is used otherwise an imperfect estimate of the channel is used, based on the values of received pilot signals. In this example, we enable the perfect channel estimator.

```
% Configure channel estimator
cec.PilotAverage = 'UserDefined';   % Type of pilot symbol averaging
cec.FreqWindow = 9;                 % Frequency window size in REs
cec.TimeWindow = 9;                 % Time window size in REs
cec.InterpType = 'Cubic';           % 2D interpolation type
cec.InterpWindow = 'Centered';      % Interpolation window type
cec.InterpWinSize = 1;              % Interpolation window size

% Channel estimator behavior
perfectChanEstimator = true;
```

Set CQI Delay

Set the CQI delay in subframes. This is the delay in a CQI being passed from UE to eNodeB as defined in TS36.101 Table 9.3.2.1.1-1 [1]. Note that the feedback of the CQI is assumed to be perfect, with the values being fed back in a buffer rather than being fed back in an uplink transmission.

```
cqiDelay = 8; % subframes
```

System Processing

The main processing is split into two phases, configured via the `cqiConfig` loop variable. These phases implement the two measurements required in the performance test defined in TS36.101 Section 9.3.2.1.1 [1]:

UE reported CQI. The first phase (`cqiConfig=1`) performs PDSCH transmission and reception where the Modulation and Coding Scheme (MCS) is selected on the basis of the UE reported CQI, with the reported CQI being updated every 2 subframes and fed back with a delay of 8 subframes. The final throughput, BLER and median CQI are recorded, and the BLER (`measuredBLER`) and deviation from the median CQI (`measuredAlpha`, in percent) are checked against the specified performance requirements.

Median CQI. In the second phase (`cqiConfig=2`), PDSCH transmission and reception are performed using the median CQI (`medianCQI`) determined in the first phase. The final throughput is recorded and the throughput ratio (`measuredGamma`) between using the UE reported CQI phase and the median CQI phase is reported and checked against the specified performance requirement.

The processing is performed on a subframe by subframe basis using the following steps:

- *Select CQI.* For UE reported CQI, the current CQI is read from the oldest value in the CQI buffer `cqiBuffer`; for median CQI, the CQI is always set to `medianCQI` (this is achieved by filling the CQI buffer with the median CQI value and the buffer will not be updated).
- *Select MCS according to CQI.* The Modulation and Coding Scheme (MCS) index corresponding to the CQI is selected by means of a lookup table defined by TS36.101 Table A.4-1 CSI RMC RC.1 FDD (MCS.1).
- *Determine Transport Block Size and modulation order.* The MCS index is passed to the `lteMCS` function which calculates the corresponding Transport Block Size (TBS) index and modulation order; the `lteTBS` function is then used to calculate the TBS from the TBS index and the number of resource blocks allocated to the PDSCH.
- *Transmit and receive waveform.* Transport block data is generated and passed to `lteRMCDLTool` to create a transmitted downlink waveform. This waveform is then passed through a fading channel and AWGN noise is added. The received signal is synchronized and OFDM demodulated and channel estimation is performed.
- *Measure PDSCH throughput.* The PDSCH and DL-SCH are decoded and the CRC pass/fail is recorded to determine the data throughput.
- *Update CQI.* If a CQI update is scheduled in this subframe, use the channel estimate to update the CQI with the `lteCQISelect` function. The updated CQI value is recorded in the CQI buffer. If a CQI update is not scheduled in this subframe, the previous CQI value is reused.

```
% Initialize variables used for results recording
CQIReport = [];     % reported CQI values
SINRReport = [];    % corresponding SINR values
xaxis = [];         % corresponding subframe numbers

% For each CQI configuration (UE reported and median):
for cqiConfig = 1:2

    if (cqiConfig==1)
        cqiConfigStr = 'UE reported';
    else
        cqiConfigStr = 'median';
    end
    fprintf('\nSimulating with %s CQI at %gdB SNR for %d Frame(s)\n', ...
        cqiConfigStr,SNRdB,NFrames);

    % Initialize CQI values: for UE reported, set to all ones; for median,
    % set to the median of the CQI values for the UE reported run
    if (cqiConfig==1)
        cqiBuffer = ones(1,cqiDelay);
    else
        cqiBuffer = ones(1,cqiDelay)*medianCQI;
    end

    % Initialize variables
    rng('default');     % Default random number generator seed
    totalCRC = [];      % CRC values, used for throughput calculation
    totalTBS = [];      % TBS values, used for throughput calculation
    offsets = 0;        % Initialize frame offset value
```

```
% For each subframe:
for subframeNo = 0:(NFrames*10-1)

    % Update subframe number
    enb.NSubframe = mod(subframeNo,10);

    % Select CQI, reading the oldest value from the CQI buffer
    cqiPtr = mod(subframeNo,cqiDelay);
    CQI = cqiBuffer(cqiPtr+1);

    % Select MCS according to CQI using TS36.101 Table A.4-1 CSI RMC
    % RC.1 FDD (MCS.1), which defines the relationship between CQI
    % indices and MCS indices
    IMCSTable = [-1 0 0 2 4 6 8 11 13 16 18 21 23 25 27 27];
    IMCS = IMCSTable(CQI+1);

    % Determine TBS and modulation order
    [ITBS,modulation] = lteMCS(IMCS);
    enb.PDSCH.Modulation = {modulation};
    if (mod(enb.NSubframe,5)==0)
        TBS = 0;
    else
        TBS = double(lteTBS(size(enb.PDSCH.PRBSet,1),ITBS));
    end
    enb.PDSCH.TrBlkSizes(enb.NSubframe+1) = TBS;

    % Determine if a CQI update is required in this subframe, according
    % to reporting periodicity N_pd = 2ms and configuration index
    % cqi-pmi-ConfigurationIndex = 1 from TS36.101 Table 9.3.2.1.1-1
    cqiPeriod = 2; % periodicity N_pd
    cqiOffset = 1; % offset deriving from cqi-pmi-ConfigurationIndex
    cqiUpdate = (mod(subframeNo,cqiPeriod)==cqiOffset);

    % Establish if this subframe actually needs executed for PDSCH
    % reception, CQI estimation or initial timing offset estimation:
    if((TBS~=0 && subframeNo>=(cqiDelay+cqiOffset)) || ...
            (cqiConfig==1 && cqiUpdate) || subframeNo==0)

        % Generate random bits for the subframe
        data = randi([0 1],TBS,1);

        % Create OFDM resource grid containing RMC transmission and
        % perform OFDM modulation.
        txWaveform = lteRMCDLTool(enb,data);

        % The initialization time for channel modeling is set each
        % subframe to simulate a continuously varying channel
        channel.InitTime = subframeNo/1000;

        % Pass data through the fading channel model.
        % An additional 25 samples are added to the end of the
        % waveform. These are to cover the range of delays expected
        % from the channel modeling (a combination of implementation
        % delay and channel delay spread)
        rxWaveform = lteFadingChannel(channel, ...
                        [txWaveform ; zeros(25,size(txWaveform,2))]);

        % Calculate noise gain including compensation for downlink
```

```
% power allocation
SNR = 10^((SNRdB-enb.PDSCH.Rho)/20);

% Normalize noise power to take account of sampling rate, which
% is a function of the IFFT size used in OFDM modulation, and
% the number of antennas
N0 = 1/(sqrt(2.0*enb.CellRefP*double(ofdmInfo.Nfft))*SNR);

% Create additive white Gaussian noise
noise = N0*complex(randn(size(rxWaveform)), ...
                   randn(size(rxWaveform)));

% Add AWGN to the received time domain waveform
rxWaveform = rxWaveform + noise;

% Perform synchronization
% An offset within the range of delays expected from the
% channel modeling (a combination of implementation delay and
% channel delay spread) indicates success
if (mod(subframeNo,10)==0)
    offset = lteDLFrameOffset(enb,rxWaveform);
    if (offset > 25)
        offset = offsets(end);
    end
    offsets = [offsets offset]; %#ok<AGROW>
end
rxWaveform = rxWaveform(1+offset:end,:);

% Perform OFDM demodulation on the received data to create
% the received resource grid
rxGrid = lteOFDMDemodulate(enb,rxWaveform);

% Channel estimation
if (perfectChanEstimator)
    chEstGrid = ...
       lteDLPerfectChannelEstimate(enb,channel,offset);
    n = lteOFDMDemodulate(enb,noise(1+offset:end,:));
    noiseEst = var(reshape(n,numel(n),1));
else
    [chEstGrid,noiseEst] = ...
        lteDLChannelEstimate(enb,enb.PDSCH, ...
            cec,rxGrid); %#ok<UNRCH>
end

% If this subframe requires PDSCH reception:
if(TBS~=0 && subframeNo>=(cqiDelay+cqiOffset))

    % Decode the PDSCH
    ind = ltePDSCHIndices(enb,enb.PDSCH,enb.PDSCH.PRBSet);
    pdschRx = lteExtractResources(ind,rxGrid) * ...
            (10^(-enb.PDSCH.Rho/20));
    pdschChEst = lteExtractResources(ind,chEstGrid);
    [rxBits,rxSymbols] = ltePDSCHDecode(enb,enb.PDSCH, ...
                            pdschRx,pdschChEst,noiseEst);

    % Decode the DL-SCH
    [decbits,crc] = lteDLSCHDecode(enb,enb.PDSCH,TBS,rxBits);
```

```
                % Record the CRC and TBS values for final throughput
                % calculation
                totalCRC = [totalCRC crc]; %#ok<AGROW>
                totalTBS = [totalTBS TBS]; %#ok<AGROW>

            end

            % Update CQI:
            if (cqiConfig==1 && cqiUpdate)

                % Perform CQI selection
                [thisCQI,thisSINR] = ...
                    lteCQISelect(enb,enb.PDSCH,chEstGrid,noiseEst);

                % Feed the CQI value back to UE (in a buffer)
                cqiBuffer(cqiPtr+1) = thisCQI;

                % Record values for plotting
                CQIReport = [CQIReport thisCQI];    %#ok<AGROW>
                SINRReport = [SINRReport thisSINR]; %#ok<AGROW>
                xaxis = [xaxis subframeNo];         %#ok<AGROW>

            end

        end

        % For subframes where CQI was not updated, re-use the previous
        % value in the buffer
        if (cqiConfig==1 && ~cqiUpdate)
            cqiBuffer(cqiPtr+1) = cqiBuffer(mod(cqiPtr-1,cqiDelay)+1);
        end

    end

    % Display results for the current CQI configuration
    fprintf('\nResults with %s CQI:\n',cqiConfigStr);
    tputTotal = sum(totalTBS);
    if (cqiConfig==1)

        % Compute and display throughput
        tputUEReported = sum(totalTBS.*(1-totalCRC));
        fprintf('Throughput: %d bits (%0.2f%%)\n', ...
            tputUEReported,tputUEReported/tputTotal*100);

        % Compute and display BLER
        measuredBLER = mean(totalCRC);
        fprintf('BLER: %0.3f (requirement is >= 0.02)\n',measuredBLER);

        % Compute and display median CQI
        medianCQI = ceil(median(CQIReport));
        fprintf('Median CQI: %d\n',medianCQI');

        % Compute and display proportion of CQI values
        % outside +/- 1 of the median
        measuredAlpha = (sum(CQIReport<(medianCQI-1)) + ...
             sum(CQIReport>(medianCQI+1)))/length(CQIReport)*100;
        fprintf(['Percentage of CQI indices outside +/- 1 of median:' ...
            ' %0.2f%% (requirement is >= 20%%)\n'],measuredAlpha);
```

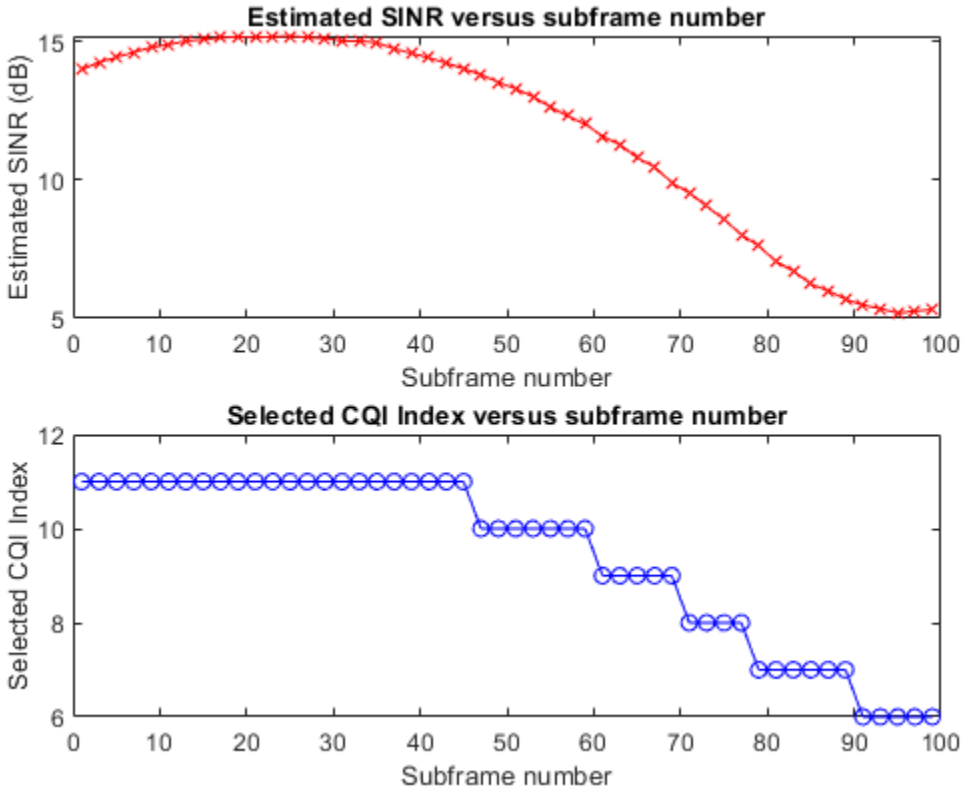
```
    else

        % Compute and display throughput
        tputMedian = sum(totalTBS.*(1-totalCRC));
        fprintf('Throughput: %d bits (%0.2f%%)\n', ...
            tputMedian,tputMedian/tputTotal*100);

        % Compute and display throughput ratio
        measuredGamma = tputUEReported/tputMedian;
        fprintf(['Throughput ratio (gamma): %0.3f' ...
            ' (requirement is >= 1.05)'],measuredGamma);

    end

end
```

```
Simulating with UE reported CQI at 6dB SNR for 10 Frame(s)

Results with UE reported CQI:
Throughput: 980384 bits (77.98%)
BLER: 0.219 (requirement is >= 0.02)
Median CQI: 10
Percentage of CQI indices outside +/- 1 of median: 30.00% (requirement is >= 20%)

Simulating with median CQI at 6dB SNR for 10 Frame(s)

Results with median CQI:
Throughput: 722304 bits (60.27%)
Throughput ratio (gamma): 1.357 (requirement is >= 1.05)
```

Plot Results

A figure with two subplots is produced. The first subplot shows the estimated SINR for each subframe; the second subplot shows the reported CQI for each subframe. This illustrates how the SINR and corresponding reported CQI vary over time due to the fading channel.

```
figure;
subplot(2,1,1);
plot(xaxis,SINRReport,'rx-');
xlabel('Subframe number');
ylabel('Estimated SINR (dB)');
title('Estimated SINR versus subframe number');
hold on;
subplot(2,1,2);
plot(xaxis,CQIReport,'bo-');
xlabel('Subframe number');
ylabel('Selected CQI Index');
title('Selected CQI Index versus subframe number');
```



Selected Bibliography

1 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception"

Reporting of Rank Indicator (RI) Conformance Test

This example demonstrates how to measure the Rank Indicator (RI) reporting performance using the LTE Toolbox™ under conformance test conditions as defined in TS36.101 Section 9.5.1.1 [1].

Introduction

This example highlights the use of the `lteRISelect` function which provides estimation of the RI. The performance of the RI estimation is also tested. The performance requirement defined in TS36.101 Section 9.5.1.1 [1] is as follows:

- the ratio of the throughput obtained when transmitting based on UE reported RI and that obtained when transmitting with fixed rank 2 shall be >= 1 (Test 1)
- the ratio of the throughput obtained when transmitting based on UE reported RI and that obtained when transmitting with fixed rank 1 shall be >= 1.05 (Test 2)

This example tests that these requirements are met.

Simulation Configuration

The example is executed for a simulation length of 10 frames at an SNR of 0.0dB. The fixed RI value is configured to be 2, therefore simulating TS36.101 Section 9.5.1.1 Test 1 [1]. A large number of `NFrames` should be used to produce meaningful results. The variable `FixedRI` controls which of the performance requirements described in the introduction are tested: `FixedRI=2` corresponds to Test 1 and `FixedRI=1` corresponds to Test 2. Note that a target SNR of 20.0dB rather than 0.0dB applies for Test 2.

```
NFrames = 10;
SNRdB = 0.0;
FixedRI = 2;
```

Propagation Channel Model Configuration

The structure, `channel`, contains the channel model configuration parameters. Note that TS36.101 Section 9.5.1.1 Test 3 [1] can also be implemented, by setting `channel.MIMOCorrelation = 'High'` (a target SNR of 20.0dB applies to Test 3, and the throughput ratio target is different).

```
channel.Seed = 1;                   % Channel seed
channel.NRxAnts = 2;                % 2 receive antennas
channel.DelayProfile = 'EPA';       % Delay profile
channel.DopplerFreq = 5.0;          % Doppler frequency
channel.MIMOCorrelation = 'Low';    % Multi-antenna correlation
channel.ModelType = 'GMEDS';        % Rayleigh fading model type
channel.NormalizeTxAnts = 'On';     % Normalize for transmit antennas
channel.NormalizePathGains = 'On';  % Normalize delay profile power
channel.InitPhase = 'Random';       % Random initial phases
channel.NTerms = 16;                % Oscillators used in fading model
```

Channel Estimator Configuration

The channel estimator is configured with a structure `cec`. The variable `perfectChanEstimator` controls channel estimator behavior. Valid values are `true` or `false`. When set to `true` a perfect channel estimator is used otherwise an imperfect estimate of the channel is used, based on the values of received pilot signals. In this example, we enable the perfect channel estimator.

```
% Configure channel estimator
cec.PilotAverage = 'UserDefined';   % Type of pilot symbol averaging
cec.FreqWindow = 9;                 % Frequency window size in REs
cec.TimeWindow = 9;                 % Time window size in REs
cec.InterpType = 'Cubic';           % 2D interpolation type
cec.InterpWindow = 'Centered';      % Interpolation window type
cec.InterpWinSize = 1;              % Interpolation window size

% Channel estimator behavior
perfectChanEstimator = true;
```

Set RI and CQI/PMI Delays

Initialize RI and CQI/PMI reporting delays in subframes; the RI delay is 9 subframes rather than 8 subframes in accordance with Note 5 of TS36.101 Table 9.5.1.1-1 [1]. Note that the feedback of the RI, CQI and PMI is assumed to be perfect, with the values being fed back in buffers rather than being fed back in an uplink transmissions.

```
riDelay = 9; % subframes
cqipmiDelay = 8; % subframes
```

Codebook Subset Restriction Bitmaps

In order to control the PMI selection which underlies the rank selection, codebook subset restriction bitmaps are used as described in TS36.213 Section 7.2 [2]. `codebookSubsetRI1` configures the codebook subset restriction to the two precoders for rank 1, used for `FixedRI=1`. `codebookSubsetRI2` configures the codebook subset restriction to the single precoder of rank 2, used for `FixedRI=2`. `codebookSubsetUEReported` is the union of these codebook subset restrictions and allows the rank to be selected dynamically in the case of UE reported RI.

```
codebookSubsetRI1 = '000011';
codebookSubsetRI2 = '010000';
codebookSubsetUEReported = '010011';
```

System Processing

The main processing is split into two phases, configured via the `riConfig` loop variable. These phases implement the two measurements required in the performance test defined in TS36.101 Section 9.5.1.1 [1]:

- *Fixed RI*. The first phase (`riConfig=1`) performs PDSCH transmission and reception where the rank is set to a fixed value (either 1 or 2, selected by the variable `FixedRI` above). The final throughput is recorded.
- *UE reported RI*. In the second phase (`riConfig=2`), the number of transmission layers is selected on the basis of UE reported RI, with the RI being updated every 5 subframes. The final throughput is recorded. The throughput ratio (`measuredGamma`) between the fixed RI phase and the UE reported phase is calculated and checked against the specified performance requirement.

Note that CQI and PMI are also updated every 5 subframes during both simulation phases.

The processing is performed on a subframe by subframe basis using the following steps:

- *Select RI*. The current RI is read from the oldest value in the RI buffer `riBuffer`. The current RI is used to configure the number of transmission layers and codewords for the PDSCH.

- *Select CQI/PMI.* The current CQI and PMI are read from the oldest values in the CQI and PMI buffers `cqiBuffer` and `pmiBuffer`. The current PMI is used to configure the precoding for the PDSCH. Note that for RI=2, there are two CQI values, one for each codeword; for RI=1 there is one CQI value for the single codeword.
- *Select MCS according to CQI.* The Modulation and Coding Scheme (MCS) index corresponding to the CQI is selected for each codeword by means of a lookup table defined in TS36.101 Table A.4-1 [1] CSI RMC RC.2 FDD (MCS.2).
- *Determine Transport Block Size and modulation order.* The MCS index for each codeword is passed to the `lteMCS` function which calculates the corresponding Transport Block Size (TBS) index and modulation order; the `lteTBS` function is then used to calculate the TBS for each codeword from the TBS index and the number of resource blocks allocated to the PDSCH.
- *Transmit and receive waveform.* Transport block data is generated for one or two codewords as appropriate and passed to `lteRMCDLTool` to create a transmitted downlink waveform. This waveform is then passed through a fading channel and AWGN noise is added. The received signal is synchronized and OFDM demodulated and channel estimation is performed.
- *Measure PDSCH throughput.* The PDSCH and DL-SCH are decoded and the CRC pass/fail for each codeword is recorded to determine the data throughput.
- *Update RI.* If an RI update is scheduled in this subframe, use the channel estimate to update the RI with the `lteRISelect` function. The updated RI value is recorded in the RI buffer. The codebook subset restriction bitmap ensures that the reported RI will remain fixed for the fixed RI phase but will be dynamically selected for the UE reported RI phase. If an RI update is not scheduled in this subframe, the previous RI value is reused.
- *Update CQI/PMI.* If a CQI/PMI update is scheduled in this subframe, use the channel estimate to update the CQI and PMI with the `lteCQISelect` and `ltePMISelect` functions. The updated CQI and PMI values are recorded in the CQI and PMI buffers. If a CQI/PMI update is not scheduled in this subframe, the previous CQI and PMI values are reused.

```
% Check that fixed RI valid is either 1 or 2
if (~any(FixedRI==[1 2]))
    error('Fixed RI value specified must be 1 or 2.');
end

% For each RI configuration (fixed RI and UE reported):
for riConfig = 1:2

    if (riConfig==1)
        riConfigStr = sprintf('fixed RI=%d',FixedRI);
    else
        riConfigStr = 'UE reported RI';
    end
    fprintf('\nSimulating with %s at %gdB SNR for %d Frame(s)\n', ...
        riConfigStr,SNRdB,NFrames);

    % Set up eNodeB settings for 2 codewords
    enb = struct('RC','R.3');
    enb.CellRefP = 2;
    enb.CFI = 3;
    enb.OCNGPDSCHEnable = 'On';
    enb.TotSubframes = 1;
    enb.PDSCH.RVSeq = 0;
```

```
enb.PDSCH.CSIMode = 'PUCCH 1-1';
enb.PDSCH.TxScheme = 'SpatialMux';
enb.PDSCH.NLayers = 2;
enb.PDSCH.Rho = -3.0;
enb.PDSCH.CSI = 'On';
ncw = 2;
enb = lteRMCDL(enb,ncw);

% Set channel model sampling rate
ofdmInfo = lteOFDMInfo(enb);
channel.SamplingRate = ofdmInfo.SamplingRate;

% Configure appropriate codebook subset restriction and target
% throughput ratio
if (riConfig==1)
    if (FixedRI==1)
        enb.PDSCH.CodebookSubset = codebookSubsetRI1;
        targetGamma = 1.05;
    else
        enb.PDSCH.CodebookSubset = codebookSubsetRI2;
        targetGamma = 1.0;
    end
else
    enb.PDSCH.CodebookSubset = codebookSubsetUEReported;
end

% Initialize variables
totalCRC = [];      % CRC values, used for throughput calculation
totalTBS = [];      % TBS values, used for throughput calculation
RIReport = [];      % reported RI values
riXaxis = [];       % corresponding subframe numbers
PMIReport = [];     % reported PMI values
CQIReport = [];     % reported CQI values
SINRReport = [];    % corresponding SINR values
cqipmiXaxis = [];   % corresponding subframe numbers
offsets = 0;        % Initialize frame offset value
rng('default');     % Default random number generator seed

% Initialize RI/PMI/CQI buffers; note that the PDSCH throughput
% resulting from these initial values is ignored, as the throughput
% recording waits until the buffers have been filled with meaningful
% reports
riBuffer = ones(1,riDelay);
pmiBuffer = ones(1,cqipmiDelay);
cqiBuffer = ones(ncw,cqipmiDelay);

% For each subframe:
for subframeNo = 0:(NFrames*10-1)

    % Update subframe number
    enb.NSubframe = mod(subframeNo,10);

    % Update PMI, reading the oldest value from the PMI buffer
    cqipmiPtr = mod(subframeNo,cqipmiDelay);
    enb.PDSCH.PMISet = pmiBuffer(cqipmiPtr+1);

    % Update number of transmission layers based on RI, reading the
    % oldest value from the RI buffer
```

```
riPtr = mod(subframeNo,riDelay);
enb.PDSCH.NLayers = riBuffer(riPtr+1);

% Update MCS according to CQI, reading the oldest value from the
% CQI buffer
CQI = cqiBuffer(:,cqipmiPtr+1);
% Select MCS according to CQI using TS36.101 Table A.4-1 CSI RMC
% RC.2 FDD (MCS.2), which defines the relationship between CQI
% indices and MCS indices
IMCSTable = [-1 0 0 2 4 6 8 11 13 15 18 20 22 24 26 27];
IMCS = IMCSTable(CQI+1);

% Determine TBS and modulation order
[ITBS,modulation] = lteMCS(IMCS);
if (mod(enb.NSubframe,5)==0)
    TBS = [0; 0];
else
    TBS = double(lteTBS(size(enb.PDSCH.PRBSet,1),ITBS));
end
enb.PDSCH.TrBlkSizes(:,enb.NSubframe+1) = TBS;

% Determine if an RI or PMI/CQI update is required in this
% subframe, according to reporting periodicity N_pd = 5ms and
% configuration indices cqi-pmi-ConfigurationIndex = 6 and
% ri-ConfigurationInd = 1 from TS36.101 Table 9.5.1.1-1
riPeriod = 5; % periodicity N_pd
riOffset = 3; % offset deriving from configuration indices
riUpdate = (mod(subframeNo,riPeriod)==riOffset);
cqipmiPeriod = 5; % periodicity N_pd
cqipmiOffset = 4; % offset deriving from cqi-pmi-ConfigurationIndex
cqipmiUpdate = (mod(subframeNo,cqipmiPeriod)==cqipmiOffset);

% Establish if this subframe actually needs executed for PDSCH
% reception, RI/PMI/CQI estimation or initial timing offset
% estimation:
firstRxSubframe = max(cqipmiDelay+cqipmiOffset,riDelay+riOffset);
if((any(TBS) && subframeNo>=firstRxSubframe) || ...
        (riUpdate || cqipmiUpdate) || subframeNo==0)

    % Configure data for transmission
    % (1 codeword on 1 layer, 2 codewords on 2 layers)
    data = {randi([0 1],TBS(1),1)};
    if (enb.PDSCH.NLayers==2)
        data = {data{1} randi([0 1],TBS(2),1)};
    end
    enb.PDSCH.Modulation = modulation(1:numel(data));

    % Create OFDM resource grid containing RMC transmission and
    % perform OFDM modulation
    txWaveform = lteRMCDLTool(enb,data);

    % The initialization time for channel modeling is set each
    % subframe to simulate a continuously varying channel
    channel.InitTime = subframeNo/1000;

    % Pass data through the fading channel model.
    % An additional 25 samples are added to the end of the
    % waveform. These are to cover the range of delays expected
```

```
% from the channel modeling (a combination of implementation
% delay and channel delay spread)
rxWaveform = lteFadingChannel(channel, ...
                [txWaveform ; zeros(25,size(txWaveform,2))]);

% Calculate noise gain including compensation for downlink
% power allocation
SNR = 10^((SNRdB-enb.PDSCH.Rho)/20);

% Normalize noise power to take account of sampling rate, which
% is a function of the IFFT size used in OFDM modulation, and
% the number of antennas
N0 = 1/(sqrt(2.0*enb.CellRefP*double(ofdmInfo.Nfft))*SNR);

% Create additive white Gaussian noise
noise = N0*complex(randn(size(rxWaveform)), ...
                   randn(size(rxWaveform)));

% Add AWGN to the received time domain waveform
rxWaveform = rxWaveform + noise;

% Perform synchronization
% An offset within the range of delays expected from the
% channel modeling (a combination of implementation delay and
% channel delay spread) indicates success
if (mod(subframeNo,10)==0)
    offset = lteDLFrameOffset(enb,rxWaveform);
    if (offset > 25)
        offset = offsets(end);
    end
    offsets = [offsets offset]; %#ok<AGROW>
end
rxWaveform = rxWaveform(1+offset:end,:);

% Perform OFDM demodulation on the received data to create the
% received resource grid
rxGrid = lteOFDMDemodulate(enb,rxWaveform);

% Channel estimation
if (perfectChanEstimator)
    chEstGrid = ...
       lteDLPerfectChannelEstimate(enb,channel,offset);
    n = lteOFDMDemodulate(enb,noise(1+offset:end,:));
    noiseEst = var(reshape(n,numel(n),1));
else
    [chEstGrid,noiseEst] = ...
        lteDLChannelEstimate(enb,enb.PDSCH, ...
            cec,rxGrid); %#ok<UNRCH>
end

% If this subframe requires PDSCH reception:
if(any(TBS) && subframeNo>=firstRxSubframe)

    % Decode the PDSCH
    ind = ltePDSCHIndices(enb,enb.PDSCH,enb.PDSCH.PRBSet);
    pdschRx = lteExtractResources(ind,rxGrid) * ...
                (10^(-enb.PDSCH.Rho/20));
    pdschChEst = lteExtractResources(ind,chEstGrid);
```

```
        [rxBits,rxSymbols] = ltePDSCHDecode(enb,enb.PDSCH, ...
                                pdschRx,pdschChEst,noiseEst);

        % Decode the DL-SCH
        [decbits,crc] = lteDLSCHDecode(enb,enb.PDSCH, ...
            enb.PDSCH.TrBlkSizes(1:numel(rxBits),enb.NSubframe+1),rxBits);

        % Record the CRC and TBS values for final throughput
        % calculation
        totalCRC = [totalCRC crc];                  %#ok<AGROW>
        totalTBS = [totalTBS TBS(1:numel(crc)).'];  %#ok<AGROW>

    end

    % Update RI:
    if (riUpdate)

        % Update RI
        thisRI = lteRISelect(enb,enb.PDSCH,chEstGrid,noiseEst);

        % Feed the value back to UE (in a buffer)
        riBuffer(riPtr+1) = thisRI;

        % Record values for plotting
        if (riConfig==2)
            RIReport = [RIReport thisRI];           %#ok<AGROW>
            riXaxis = [riXaxis subframeNo];         %#ok<AGROW>
        end

    end

    % Update CQI and PMI:
    if (cqipmiUpdate)

        % Update PMI conditioned on the most recently reported RI,
        % or the fixed RI if RI reporting is not configured
        if (isempty(RIReport))
            enb.PDSCH.NLayers = FixedRI;
        else
            enb.PDSCH.NLayers = RIReport(end);
        end
        thisPMI = ltePMISelect(enb,enb.PDSCH,chEstGrid,noiseEst);

        % Update CQI conditioned on the most recently reported PMI
        % and RI; this includes configuring the number of codewords
        % for CQI selection based on the RI
        enb.PDSCH.PMISet = thisPMI;
        enb.PDSCH.NCodewords = enb.PDSCH.NLayers;
        [thisCQI,thisSINR] = ...
            lteCQISelect(enb,enb.PDSCH,chEstGrid,noiseEst);

        % Feed the values back to UE (in buffers)
        pmiBuffer(:,cqipmiPtr+1) = thisPMI;
        % For CSIMode='PUCCH 1-1', the CQI for the second codeword
        % is reported as a differential from the first codeword;
        % here we convert to absolute CQI indices for both
        % codewords.
        if (numel(thisCQI)==2)
```

```
                thisCQI(2) = thisCQI(1) - thisCQI(2);
            end
            % Transmit with lowest CQI=1 when CQI selection reports
            % CQI=0 (out of range)
            thisCQI(thisCQI==0) = 1;

            cqiBuffer(:,cqipmiPtr+1) = thisCQI;

            % Record values for plotting
            if (riConfig==2)
                PMIReport = [PMIReport thisPMI];        %#ok<AGROW>
                CQIReport = [CQIReport thisCQI(1)];     %#ok<AGROW>
                SINRReport = [SINRReport thisSINR(1)];  %#ok<AGROW>
                cqipmiXaxis = [cqipmiXaxis subframeNo]; %#ok<AGROW>
            end

        end

    end

    % For subframes where RI was not updated, re-use the previous value
    % in the buffer
    if (~riUpdate)
        riBuffer(riPtr+1) = riBuffer(mod(riPtr-1,riDelay)+1);
    end

    % For subframes where PMI/CQI were not updated, re-use the previous
    % values in the buffers
    if (~cqipmiUpdate)
        pmiBuffer(:,cqipmiPtr+1) = ...
            pmiBuffer(:,mod(cqipmiPtr-1,cqipmiDelay)+1);
        cqiBuffer(:,cqipmiPtr+1) = ...
            cqiBuffer(:,mod(cqipmiPtr-1,cqipmiDelay)+1);
    end

end

% Display results for the current RI configuration
fprintf('\nResults with %s:\n',riConfigStr);
tputTotal = sum(totalTBS);
if (riConfig==1)

    % Compute and display throughput
    tputFixedRI = sum(totalTBS.*(1-totalCRC));
    fprintf('Throughput: %d bits (%0.2f%%)\n', ...
        tputFixedRI,tputFixedRI/tputTotal*100);

else

    % Compute and display throughput
    tputUEReported = sum(totalTBS.*(1-totalCRC));
    fprintf('Throughput: %d bits (%0.2f%%)\n', ...
        tputUEReported,tputUEReported/tputTotal*100);

    % Compute and display throughput ratio
    measuredGamma = tputUEReported/tputFixedRI;
    fprintf(['Throughput ratio (gamma): %0.3f' ...
        ' (requirement is >= %0.2f)'],measuredGamma,targetGamma);
```

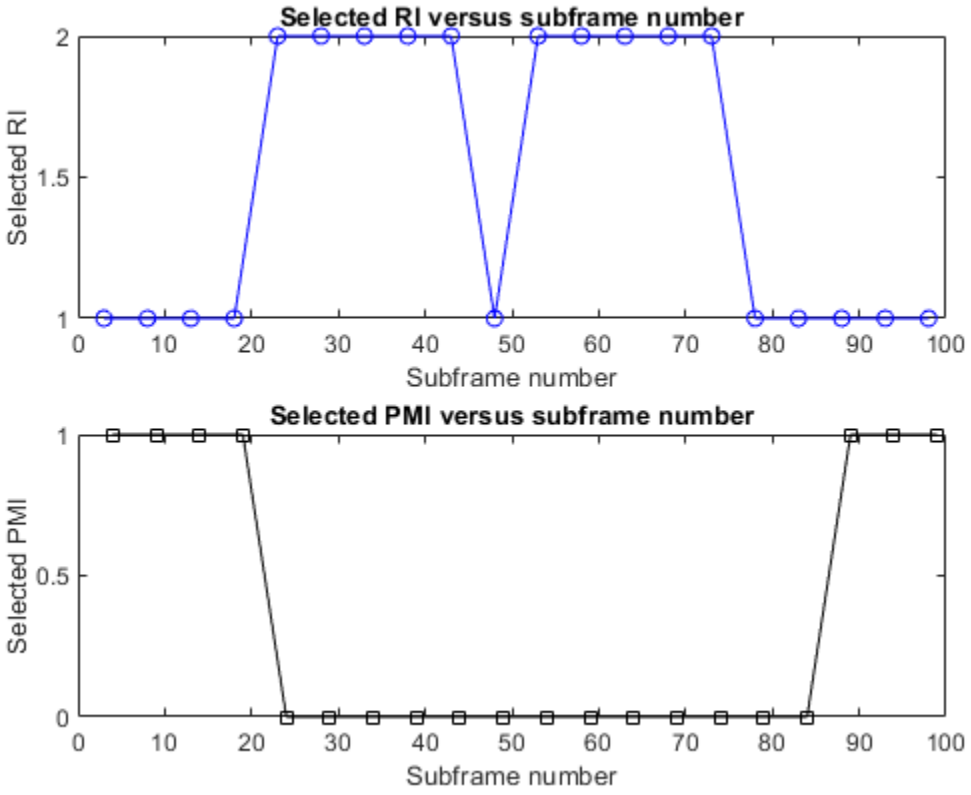
```
    end

end
```

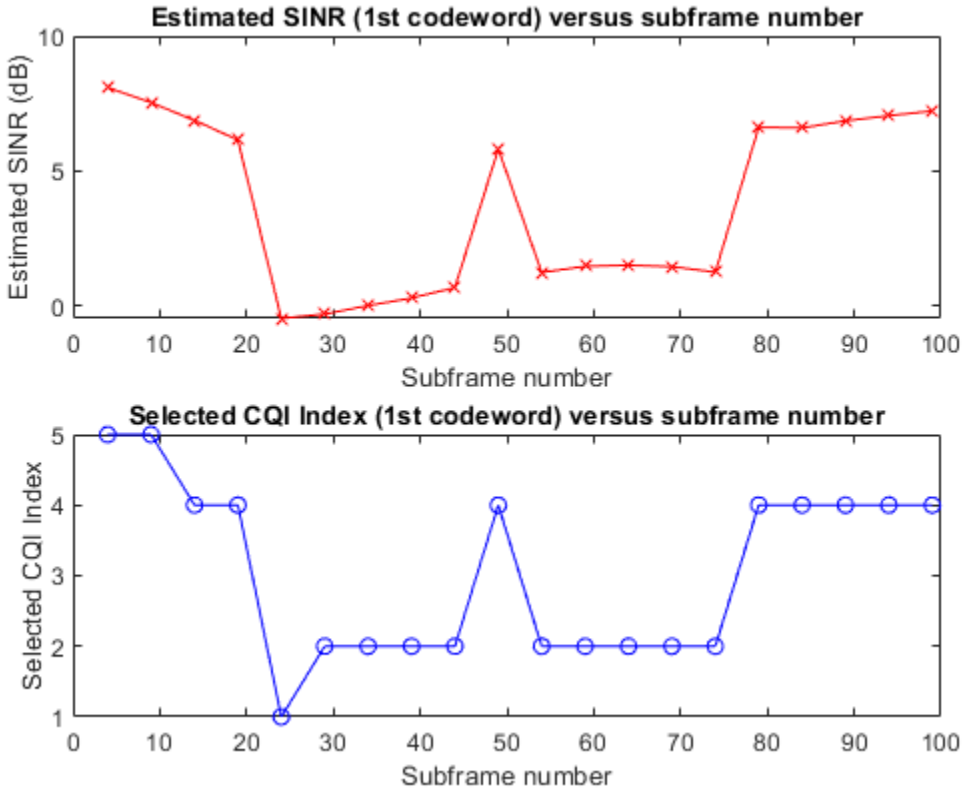
```
Simulating with fixed RI=2 at 0dB SNR for 10 Frame(s)

Results with fixed RI=2:
Throughput: 203184 bits (100.00%)

Simulating with UE reported RI at 0dB SNR for 10 Frame(s)

Results with UE reported RI:
Throughput: 235352 bits (100.00%)
Throughput ratio (gamma): 1.158 (requirement is >= 1.00)
```

Plot Results

Two figures are produced. The first figure has two subplots: the first subplot shows the reported RI for each subframe; the second subplot shows the reported PMI for each subframe. The second figure also has two subplots: the first subplot shows the estimated SINR for each subframe; the second subplot shows the reported CQI for each subframe. These plots illustrate how the RI, PMI and CQI vary over time due to the fading channel.

```
figure;
subplot(2,1,1);
plot(riXaxis,RIReport,'bo-');
xlabel('Subframe number');
ylabel('Selected RI');
title('Selected RI versus subframe number');
hold on;
subplot(2,1,2);
plot(cqipmiXaxis,PMIReport,'ks-');
xlabel('Subframe number');
ylabel('Selected PMI');
title('Selected PMI versus subframe number');

figure;
subplot(2,1,1);
plot(cqipmiXaxis,SINRReport,'rx-');
xlabel('Subframe number');
ylabel('Estimated SINR (dB)');
title('Estimated SINR (1st codeword) versus subframe number');
hold on;
subplot(2,1,2);
plot(cqipmiXaxis,CQIReport,'bo-');
xlabel('Subframe number');
ylabel('Selected CQI Index');
title('Selected CQI Index (1st codeword) versus subframe number');
```


Selected RI versus subframe number
Selected RI
Subframe number
Selected PMI versus subframe number
Selected PMI
Subframe number



Selected Bibliography

1 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception"

2 3GPP TS 36.213 "Physical layer procedures"

Enhanced Physical Downlink Control Channel (EPDCCH) Conformance Test

This example shows how to measure EPDCCH/PCFICH demodulation performance using LTE Toolbox™ under the test conditions defined in TS 36.101 [1] Section 8.8 "Demodulation of EPDCCH".

Introduction

This example shows the demodulation performance of EPDCCH/PCFICH in terms of the probability of misdetection of the downlink scheduling grant (Pm-dsg) as defined in TS 36.101 [1] Section 8.8 "Demodulation of EPDCCH". The EPDCCH and PCFICH are tested jointly and a misdetection of PCFICH implies a misdetection of EPDCCH. This example works on a subframe by subframe basis when simulating at a particular SNR. A resource grid populated with EPDCCH, EPDCCH DM-RS, and PCFICH is generated and OFDM modulated to create a transmit waveform. The generated waveform is passed through a noisy Extended Vehicular A (EVA) channel. Channel estimation, equalization, demodulation and decoding are performed at the receiver. The average misdetection probability of the EPDCCH/PCFICH is determined using a combination of the received CFI and received DCI message.

LTE Toolbox supports the transmission and reception of EPDCCH via these functions.

- `lteDCI` and `lteDCIEncode`: Create and encode downlink control information (DCI) messages
- `lteEPDCCHSpace`: Create EPDCCH search space candidate
- `lteEPDCCH` and `lteEPDCCHIndices`: Generate EPDCCH symbols and map to resource elements
- `lteEPDCCHDMRS` and `lteEPDCCHDMRSIndices`: Create EPDCCH demodulation reference signal (DM-RS) and map to resource elements
- `lteDLChannelEstimate`: channel estimation including support for estimation using the EPDCCH DM-RS
- `lteEPDCCHDecode`: Decode EPDCCH symbols
- `lteEPDCCHSearch`: Perform EPDCCH DCI blind search

Simulation Setup

The variable `Test` allows selection of Test number 1 or 2 from TS 36.101 [1] Table 8.8.1.1-2. For Test 1, the target SNR for a probability of missed downlink scheduling grant (Pm-dsg) of <= 1% is 2.6dB. For Test 2, the target SNR for Pm-dsg <= 1% is -3.2dB. The code below sets up the target SNR depending on the Test number, and it also chooses a range of SNR points `SNRIn` to simulate, where the range covers the target SNR. `SNRIn` can be an array of values or a scalar.

```
Test = 1; % Test number (1 or 2)

if (Test==1)
    targetSNR = 2.6;
    SNRIn = [-1.5 1.0 3.5];
else
    targetSNR = -3.2;
    SNRIn = [-7.3 -4.8 2.3];
end
```

The example is executed for a simulation length of 1 frame to keep the simulation time low. A large number of frames, `nFrames`, should be used to produce meaningful results.

```
nFrames = 1;
```

eNodeB Configuration

Create a structure `enb` representing the cell-wide (eNodeB) settings specified for the test. The number of downlink resource blocks (50) corresponds to the test bandwidth (10MHz) defined in TS 36.101 [1] Table 8.8.1.1-2. The duplexing mode is indicated in Table 8.8.1.1-2, as part of the description of the Reference Channel: "R.55 FDD" for Test 1 and "R.56 FDD" for Test 2. See TS 36.101 Appendix A.3.10.1 for additional R.55 FDD and R.56 FDD reference channel settings.

```
% eNodeB configuration according to TS 36.101 Table 8.8.1.1-2
enb = struct;
enb.NDLRB = 50;
enb.DuplexMode = 'FDD';

% eNodeB configuration according to TS 36.101 Table 8.8.1.1-1
enb.CyclicPrefix = 'Normal';
enb.CellRefP = 2;
enb.CFI = 2 - (enb.NDLRB <= 10); % 2 symbols for PDCCH
enb.NCellID = 0;
```

EPDCCH Configuration

Create a structure `epdcch` representing the EPDCCH settings specified for the test.

For these tests, two EPDCCH sets are configured as noted in TS 36.101 [1] Table 8.8.1.1-1. `EPDCCHTypeList` and `EPDCCHPRBSetList` give the EPDCCH types and EPDCCH PRB pairs for each EPDCCH set. Note that the EPDCCH sets must be configured in the order given by `EPDCCHPRBSetList`. Although TS 36.101 Table 8.8.1.1-1 specifies that the first set has 4 PRB pairs and the second set has 8 PRB pairs, the definition of the number of PRB pairs in the EPDCCH candidate sets in TS 36.213 [3] Table 9.1.4-3a has an entry for 8 pairs in set 1 and 4 pairs in set 2, but not the other way around.

Set `ControlChannelType` to `'EPDCCH'`, so that `lteDCIInfo` reports the correct DCI message sizes for the DCI messages transmitted on the EPDCCH. The DCI Format `DCIFormat` is also set as described in TS 36.101 Table 8.8.1.1-1

The EPDCCH format `EPDCCHFormat` is set for the particular Test number according to the "Aggregation level" noted in TS 36.101 Table 8.8.1.1-2. The relationship between EPDCCH format and aggregation level is given by TS 36.211 [2] Table 6.8A.1-2, where the aggregation level is described as the "Number of ECCEs for one EPDCCH". For Test 1 the EPDCCH format is format 1, corresponding to an aggregation level of 4 ECCE (Enhanced Control Channel Elements), and for Test 2 the EPDCCH format is format 3, corresponding to an aggregation level of 16 ECCE.

```
% EPDCCH configuration according to TS 36.101 Table 8.8.1.1-1
epdcch = struct;
epdcch.EPDCCHTypeList = {'Distributed' 'Distributed'};
epdcch.EPDCCHPRBSetList = {[0; 7; 14; 21; 28; 35; 42; 49] [3; 17; 31; 45]};
epdcch.ControlChannelType = 'EPDCCH';
epdcch.DCIFormat = 'Format2A';

% EPDCCH configuration according to TS 36.101 Table 8.8.1.1-2
if (Test==1)
```

```
    epdcch.EPDCCHFormat = 1; % Aggregation level L = 4 ECCE
else
    epdcch.EPDCCHFormat = 3; % Aggregation level L = 16 ECCE
end

% Configure the EPDCCH scrambling identity and RNTI. These values are not
% mandated by the test configuration, but the parameters are required.
epdcch.EPDCCHNID = 0;
epdcch.RNTI = 1;
```

Propagation Channel Model Configuration

Create a propagation channel configuration structure, `channel`, according to TS 36.101 [1] Table 8.8.1.1-2. This configuration structure will be used by the function `lteFadingChannel`.

```
channel = struct;
channel.NRxAnts = 2;
channel.DelayProfile = 'EVA';
channel.MIMOCorrelation = 'Low';
if (Test==1)
    channel.DopplerFreq = 5;
else
    channel.DopplerFreq = 70;
end
```

Channel Estimator Configuration

The logical variable `perfectChannelEstimator` controls channel estimator behavior. When set to `true`, a perfect channel estimator is used otherwise an imperfect estimator is used, based on the values of the received EPDCCH DM-RS. Set perfectChannelEstimator to `false`, and create channel estimation configuration structure, `cec`, which the `lteDLChannelEstimate` function will use to generate an imperfect channel estimate.

```
perfectChannelEstimator = false;

if (~perfectChannelEstimator)
    cec.PilotAverage = 'UserDefined';   % Type of pilot symbol averaging
    cec.FreqWindow = 1;                 % Frequency window size in REs
    cec.TimeWindow = 2;                 % Time window size in REs
    cec.InterpType = 'linear';          % 2D interpolation type
    cec.InterpWindow = 'Centered';      % Interpolation window type
    cec.InterpWinSize = 1;              % Interpolation window size
    cec.Reference = 'EPDCCHDMRS';       % EPDCCH DM-RS used as reference
end
```

Select EPDCCH Transmission Set

The EPDCCH set for transmission `txSetIndex` is configured in the code below as a function of the Test number. Note that because the EPDCCH receiver (channel estimation, demodulation, EPDCCH search) runs over both EPDCCH sets, the receiver does not need to be configured specifically for either Test 1 or Test 2, it will automatically decode the transmitted DCI message in the relevant EPDCCH set.

```
txSetIndex = 3-Test; % Test=1 -> 2nd set, Test=2 -> 1st set

% Configure EPDCCH set for transmission from the two EPDCCH sets according
% to 'txSetIndex'
```

```
epdcch.EPDCCHType = epdcch.EPDCCHTypeList{txSetIndex};
epdcch.EPDCCHPRBSet = epdcch.EPDCCHPRBSetList{txSetIndex};
```

Create Variables to Record Simulated Performance

The performance of the receiver is recorded in the matrix `mdsg`, representing missed downlink scheduling grants. This matrix has a row for each SNR point and a column for each subframe simulated. The elements will be set to 1 for SNR points/subframes where the downlink scheduling grant was missed. As described earlier, the EPDCCH and PCFICH are tested jointly and a misdetection of PCFICH implies a misdetection of EPDCCH. Therefore an element of `mdsg` is set to 1 if either the PCFICH or EPDCCH could not be demodulated. For convenience, a separate matrix `mcfi` is created to record the misdetections of PCFICH only.

```
% The 'mdsg' matrix will contain ones in the locations of missed downlink
% scheduling grants; the 'mcfi' matrix will contain ones in the locations
% of missed CFI decoding
mdsg = zeros(numel(SNRIn),10*nFrames);
mcfi = zeros(numel(SNRIn),10*nFrames);
```

Establish Number of Transmit Antennas

Establish the number of transmit antennas `ntxants`, which is the maximum of the number of antenna ports used for each set of reference signals configured in the example. Note that although the EPDCCH has 4 reference signal ports, only a maximum of 2 of these will be used for any given configuration (2 for distributed EPDCCH type, 1 for localized EPDCCH type), so only 2 physical antennas are required to support the transmission of EPDCCH.

```
epdcchports = 2;
portcounts = [enb.CellRefP epdcchports];
if (isfield(enb,'CSIRefP'))
    portcounts = [portcounts enb.CSIRefP];
end
ntxants = max(portcounts);
```

Print Parameter Summary

Print a summary of the simulation parameters, for the list of parameters in TS 36.101 [1] Annex A.3.10 "Reference Measurement Channels for EPDCCH performance requirements". Note that the aggregation level (number of ECCEs per EPDCCH) can be obtained for a given configuration by using the field `NECCEPerEPDCCH` of the 2nd output of the `lteEPDCCHSpace` function. `enb.NSubframe` is set here because it is a required parameter of `lteEPDCCHSpace` however it only affects the candidates (1st output), not the dimensionality information obtained from the 2nd output. (`enb.NSubframe` will be updated in the main subframe processing loop, meaning that over time the EPDCCH candidate locations used for transmission can vary.)

```
enb.NSubframe = 0;
[~,candidateInfo] = lteEPDCCHSpace(enb,epdcch);

fprintf('\n-- Parameter summary: ------------------------------------------\n');
fprintf('  Number of transmitter antennas: %d\n',ntxants);
fprintf('                Channel bandwidth: %gMHz\n',hNRBToBandwidth(enb.NDLRB));
fprintf('Number of OFDM symbols for PDCCH: %d symbols\n',enb.CFI + (enb.NDLRB <= 10));
fprintf('                Aggregation level: %d ECCE \n',candidateInfo.NECCEPerEPDCCH);
fprintf('                        DCI format: %s\n',epdcch.DCIFormat);
disp('-----------------------------------------------------------------');
```

```
-- Parameter summary: ------------------------------------------
```

```
  Number of transmitter antennas: 2
               Channel bandwidth: 10MHz
Number of OFDM symbols for PDCCH: 2 symbols
               Aggregation level: 4 ECCE
                      DCI format: Format2A
------------------------------------------------------------------
```

Processing

For each SNR point, the following operations are performed for each subframe:

- *EPDCCH candidate creation*: The `lteEPDCCHSpace` function is used to create the possible EPDCCH candidates, and a candidate is selected for transmission.
- *DCI message creation*: A set of random DCI message bits is created of the length given by `lteDCIInfo` for the DCI format specified in `epdcch.DCIFormat`.
- *DCI message encoding*: `lteDCIEncode` is used to encode the DCI message bits, with the coded capacity given by the `EPDCCHG` field of the dimensionality information output of `lteEPDCCHIndices`. The EPDCCH indices are also stored for later use in mapping the EPDCCH.
- *EPDCCH modulation*: `lteEPDCCH` is used to create modulation symbols for the encoded DCI message.
- *EPDCCH DM-RS creation*: `lteEPDCCHDMRS` is used to create the EPDCCH DM-RS sequence, and `lteEPDCCHDMRSIndices` is used to create the corresponding indices for use in mapping the EPDCCH DM-RS.
- *EPDCCH beamforming*: The EPDCCH symbols/indices are concatenated with the EPDCCH DM-RS symbols/indices to facilitate beamforming. Beamforming is performed according to TS 36.101 [1] Annex B.4.4 for Distributed type and Annex B.4.5 for Localized type.
- *CFI coding / PCFICH modulation*: `lteCFIDecode`, `ltePCFICH` and `ltePCFICHIndices` are used to encode the CFI, modulate the PCFICH and map the modulated symbols to the subframe resource grid.
- *Cell-Specific Reference Signal (CRS) addition*: `lteCellRS` and `lteCellRSIndices` are used to create the CRS sequence and map it to the subframe resource grid.
- *Primary Synchronization Signal (PSS) and Secondary Synchronization Signal (SSS) addition*: `ltePSS`, `lteSSS`, `ltePSSIndices`, and `lteSSSIndices` are used to create the PSS and SSS sequences and map them to the subframe resource grid.
- *OCNG addition*: As specified in the test, OP.7 OCNG (TS 36.101 [1] Annex A.5.1.7) is added, consisting of a single PDSCH in all resource blocks not used by the EPDCCH. A control region OCNG reference channel (TS 36.101 Annex A.5) is also added, including the PDCCH and PHICH locations but avoiding the PCFICH locations, to allow the CFI to be transmitted.
- *OFDM modulation*: The subframe resource grid is OFDM modulated using `lteOFDMModulate`.
- *Propagation Channel Model*: The OFDM modulated waveform is transmitted through the propagation channel. The output of the channel `rxwaveform` has two columns, one per receive antenna.
- *Add Channel Noise*: The channel noise is modeled by AWGN.
- *Receiver Synchronization and OFDM Demodulation*: Determine the delay suffered during propagation. This is calculated by calling `lteDLFrameOffset`, which uses the PSS and SSS. OFDM demodulation is performed after synchronization.
- *Channel estimation for PCFICH reception*: CRS-based channel estimation (or perfect channel estimation) is performed to provide a channel estimation for PCFICH equalization and demodulation

- *PCFICH / CFI reception*: The PCFICH is demodulated (including equalization) and the CFI is decoded. If the decoded CFI does not match the transmitted CFI, a misdetection of downlink scheduling grant is noted (as is a misdetection of CFI), otherwise reception proceeds to EPDCCH reception

EPDCCH reception / EPDCCH blind search is attempted for each of the two EPDCCH sets configured in this test. For each EPDCCH set:

- *Configure the receiver for the EPDCCH set*: An EPDCCH configuration structure `epdcchRx` is created by copying the structure used in the transmitter, `epdcch`. `epdcchRx.EPDCCHType` and `epdcchRx.EPDCCHPRBSet` are configured for the EPDCCH set to be received by setting them to the appropriate element of `epdcchRx.EPDCCHTypeList` and `epdcchRx.EPDCCHPRBSetList` respectively. The resulting configuration structure `epdcchRx` is then used to configure the channel estimation, EPDCCH demodulation and EPDCCH blind search steps. The fields in `epdcchRx` would be known to the UE by higher-layer signaling of the *EPDCCH-Config* Information Element (IE) in the Radio Resource Control (RRC) protocol TS 36.331 [4].
- *Channel estimation for EPDCCH reception*: If perfect channel estimation is used, the perfect channel estimate previously obtained for PCFICH reception is beamformed according to the transmit beamforming used for the EPDCCH, to create a channel estimate for EPDCCH reception. Otherwise, if practical channel estimation is used, an EPDCCH DM-RS-based channel estimation is performed.
- *EPDCCH demodulation*: `lteEPDCCHDecode` is used to demodulate the EPDCCH for all possible EPDCCH candidate locations, returning soft bit estimates for all possible candidates.
- *EPDCCH blind search*: `lteEPDCCHSearch` is used to search for DCI messages by attempting DCI message decoding in all the possible candidate locations given by `lteEPDCCHSpace`, for all the possible DCI formats. If the search fails to decode a message of the format given by the transmitter configuration in the test, a misdetection of the downlink scheduling grant is noted.

For a more detailed description of the steps involved in EPDCCH transmission, see “Enhanced Physical Downlink Control Channel (EPDCCH) Generation” on page 2-175.

```
% For each SNR point:
for SNRIdx = 1:numel(SNRIn)

    % Set the random number generator to its default value
    rng('default');

    % Get current SNR value
    SNRdB = SNRIn(SNRIdx);

    fprintf('\nSimulating at %gdB SNR for a total of %d Frame(s)\n',SNRdB,nFrames);

    % Initialize timing offset vector
    offsets = 0;

    % For each subframe:
    for sf = 0:(nFrames*10)-1

        % Update frame and subframe numbers
        enb.NFrame = floor(sf/10);
        enb.NSubframe = mod(sf,10);

        % Configure EPDCCH candidate; the first candidate from the set of
        % search space candidates is used
```

```
candidates = lteEPDCCHSpace(enb,epdcch);
epdcch.EPDCCHECCE = candidates(1,:);

% Create random DCI message bits from the configured DCI format
dciInfo = lteDCIInfo(enb,epdcch);
[dci,dciBits] = lteDCI(enb,epdcch,randi([0 1],dciInfo.(epdcch.DCIFormat),1));

% Create EPDCCH indices; the 2nd output contains the EPDCCH bit
% capacity EPDCCHG
[epdcchIndices,epdcchInfo] = lteEPDCCHIndices(enb,epdcch);

% Encode the DCI message bits to match the bit capacity EPDCCHG
codedDciBits = lteDCIEncode(epdcch,dciBits,epdcchInfo.EPDCCHG);

% Modulate the coded DCI message on the EPDCCH
epdcchSymbols = lteEPDCCH(enb,epdcch,codedDciBits);

% Create EPDCCH Demodulation Reference Signal (DM-RS)
epdcchDmrsSymbols = lteEPDCCHDMRS(enb,epdcch);
epdcchDmrsIndices = lteEPDCCHDMRSIndices(enb,epdcch);

% Create empty subframe resource grid
subframe = lteDLResourceGrid(enb,ntxants);

% Concatenate EPDCCH symbols/indices with EPDCCH DM-RS
% symbols/indices to facilitate beamforming
allSymbols = [epdcchSymbols; epdcchDmrsSymbols];
allIndices = [epdcchIndices; epdcchDmrsIndices];

% Perform EPDCCH beamforming according to TS 36.101 Annex B.4.4 for
% Distributed type and Annex B.4.5 for Localized type
[K,L,P] = size(subframe);
[resubs,~,portsubs] = ind2sub([K L 4],allIndices);
rbsubs = floor((resubs-1)/12)+1;
rbs = unique(rbsubs);
ports = unique(portsubs.');
if (strcmpi(epdcch.EPDCCHType,'Localized'))
    codebookIdx = randi([0 3],1);
    W = lteDLPrecode(1,ntxants,'SpatialMux',codebookIdx);
end
codebookIdxs = zeros(length(rbs),length(ports));
for p=1:length(ports)
    thisport = (portsubs==ports(p));
    for r = 1:length(rbs)
        thisrb = (rbsubs==rbs(r));
        if (strcmpi(epdcch.EPDCCHType,'Distributed'))
            unusedIdxs = setxor(0:3,codebookIdxs(r,1:p-1));
            codebookIdx = unusedIdxs(randi(length(unusedIdxs),1));
            W = lteDLPrecode(1,ntxants,'SpatialMux',codebookIdx);
        end
        codebookIdxs(r,p) = codebookIdx;
        bfSymbols = allSymbols(thisport & thisrb) * W;
        [~,bfIndices] = lteExtractResources(allIndices(thisport & thisrb),subframe);
        subframe(bfIndices) = subframe(bfIndices) + bfSymbols;
    end
end

% Transmit CFI on the PCFICH
```

```
subframe(ltePCFICHIndices(enb)) = ltePCFICH(enb,lteCFI(enb));

% Transmit Cell-Specific Reference Signals (CRS)
subframe(lteCellRSIndices(enb)) = lteCellRS(enb);

% Transmit PSS and SSS
subframe(ltePSSIndices(enb)) = ltePSS(enb);
subframe(lteSSSIndices(enb)) = lteSSS(enb);

% Add OCNG
subframe = addOCNG(enb,epdcch,subframe);

% Perform OFDM modulation
[txwaveform,info] = lteOFDMModulate(enb,subframe);

% Set sampling rate of channel to that of the OFDM modulation
channel.SamplingRate = info.SamplingRate;

% Set channel time to current subframe
channel.InitTime = sf/1000;

% Reinitialize channel seed for each subframe to increase
% variability
channel.Seed = sf + 1;

% Pass data through the fading channel model.
% An additional 25 samples are added to the end of the waveform.
% These are to cover the range of delays expected from the channel
% modeling (a combination of implementation delay and channel
% delay spread).
rxwaveform = lteFadingChannel(channel,[txwaveform; zeros(25,ntxants)]);

% Linear SNR
SNR = 10^(SNRdB/20);

% Normalize noise power to take account of sampling rate, which is
% a function of the IFFT size used in OFDM modulation, and the
% number of antennas
N0 = 1/(sqrt(2.0*ntxants*double(info.Nfft))*SNR);

% Create additive white Gaussian noise
noise = N0*complex(randn(size(rxwaveform)),randn(size(rxwaveform)));

% Add AWGN to the received time domain waveform
rxwaveform = rxwaveform + noise;

% Perform receiver synchronization
if (enb.NSubframe==0)
    offset = lteDLFrameOffset(enb,rxwaveform);
    % Determine if frame synchronization was successful
    if (offset > 25)
        offset = offsets(end);
    else
        offsets = [offsets offset]; %#ok
    end
end
if (offset>0)
    rxwaveform = rxwaveform(1+offset:end,:);
```

```
end

% Perform OFDM demodulation
rxsubframe = lteOFDMDemodulate(enb,rxwaveform);

% Perform CRS-based (or perfect) channel estimation for PCFICH /
% CFI reception
if (perfectChannelEstimator)
    % Perform perfect channel estimation assuming the number of
    % transmit antennas is CellRefP (i.e. for CRS-based reception)
    hest = lteDLPerfectChannelEstimate(enb,channel,offset); %#ok<UNRCH>
    noiseGrid = lteOFDMDemodulate(enb,noise(1+offset:end ,:));
    nest = var(noiseGrid(:));
else
    % Perform channel estimation using the CRS. The channel
    % estimator averaging parameters are adjusted to use a
    % rectangular averaging window suited to the CRS, rather than
    % the 1-by-2 (frequency-by-time) window configured in 'cec'
    % which is suited to the EPDCCH DM-RS symbols (which are
    % adjacent in time).
    cec_crs = cec;
    cec_crs.Reference = 'CellRS';
    cec_crs.FreqWindow = 31;
    cec_crs.TimeWindow = 15;
    [hest,nest] = lteDLChannelEstimate(enb,cec_crs,rxsubframe);
end

% Receive PCFICH / CFI
pcfichIndices = ltePCFICHIndices(enb);
[pcfichRx,pcfichHest] = lteExtractResources(pcfichIndices,rxsubframe,hest);
rxCFI = lteCFIDecode(ltePCFICHDecode(enb,pcfichRx,pcfichHest,nest));

% If the CFI was incorrectly decoded:
if (rxCFI~=enb.CFI)

    % Record a misdetection of the CFI as a misdetection of the DCI
    % message (in 'mdsg'), in accordance with the test definition
    % in TS 36.101 Section 8.8. It is also recorded separately as a
    % misdetection of the CFI (in 'mcfi'). 'mcfi' can be used to
    % establish the probability of misdetection of the CFI on its
    % own. At low SNRs and for robust EPDCCH configurations (high
    % aggregation level), misdetection of the CFI may dominate the
    % overall misdetection of the DCI message.
    mdsg(SNRIdx,sf+1) = 1;
    mcfi(SNRIdx,sf+1) = 1;

else % Proceed to EPDCCH / DCI decoding

    if (perfectChannelEstimator)
        if (ntxants~=enb.CellRefP) %#ok<UNRCH>
            % Perform perfect channel estimation for 'ntxants'
            % antennas (i.e. for EPDCCH reception) if this differs
            % from CellRefP (used earlier for PCFICH/CFI reception)
            hest = lteDLPerfectChannelEstimate(enb,channel,offset,ntxants);
        end
        % Apply EPDCCH beamforming to the perfect channel estimate
        hest = beamformPerfectChannelEstimate(hest,rbs,ports,codebookIdxs);
    end
```

```
            % Initialize array for received DCI message
            rxDCI = {};

            % Copy the structure 'epdcch' to 'epdcchRx' and remove fields
            % of 'epdcchRx' which are not required (they will be set during
            % decoding and search). This is not essential but provides
            % realism because |epdcchRx| then only contains system
            % information conveyed to the UE from the RRC
            epdcchRx = epdcch;
            epdcchRx = rmfield(epdcchRx,{'DCIFormat','EPDCCHFormat','EPDCCHECCE'});

            % For each EPDCCH set:
            for rxSetIndex = 1:numel(epdcchRx.EPDCCHTypeList)

                % Configure reception for current set
                epdcchRx.EPDCCHType = epdcchRx.EPDCCHTypeList{rxSetIndex};
                epdcchRx.EPDCCHPRBSet = epdcchRx.EPDCCHPRBSetList{rxSetIndex};

                if (~perfectChannelEstimator)
                    % Perform channel estimation using the EPDCCH DM-RS
                    [hest,nest] = lteDLChannelEstimate(enb,epdcchRx,cec,rxsubframe);
                end

                % Perform EPDCCH demodulation
                [bits,symbols] = lteEPDCCHDecode(enb,epdcchRx,rxsubframe,hest,nest);

                % Perform EPDCCH blind search and DCI message decoding
                rxDCI = [rxDCI lteEPDCCHSearch(enb,epdcchRx,bits)]; %#ok<AGROW>

            end

            % Record if the DCI message was missed; specifically if no DCI
            % messages were decoded or if none of the decoded messages
            % matched the transmitted message
            mdsg(SNRIdx,sf+1) = (isempty(rxDCI) || ~any(cellfun(@(x)isequal(x,dci),rxDCI)));

        end

    end

end
```

```
Simulating at -1.5dB SNR for a total of 1 Frame(s)

Simulating at 1dB SNR for a total of 1 Frame(s)

Simulating at 3.5dB SNR for a total of 1 Frame(s)
```

Calculate Probability of Missed Downlink Scheduling Grant (Pm-dsg)

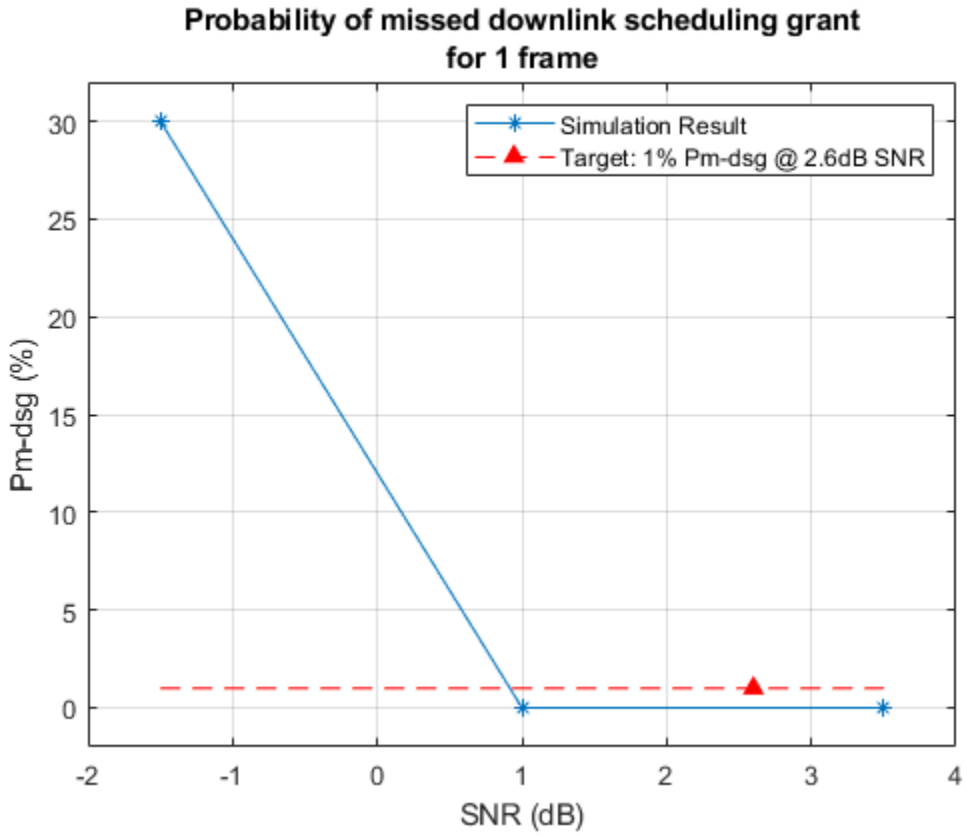
The overall probability of missed downlink scheduling grant (Pm-dsg) is computed by averaging the `mdsg` matrix along its second dimension i.e. averaging each row, to produce a vector `Pmdsg` containing the Pm-dsg value for each SNR point tested.

```
Pmdsg = 100*mean(mdsg,2);
```

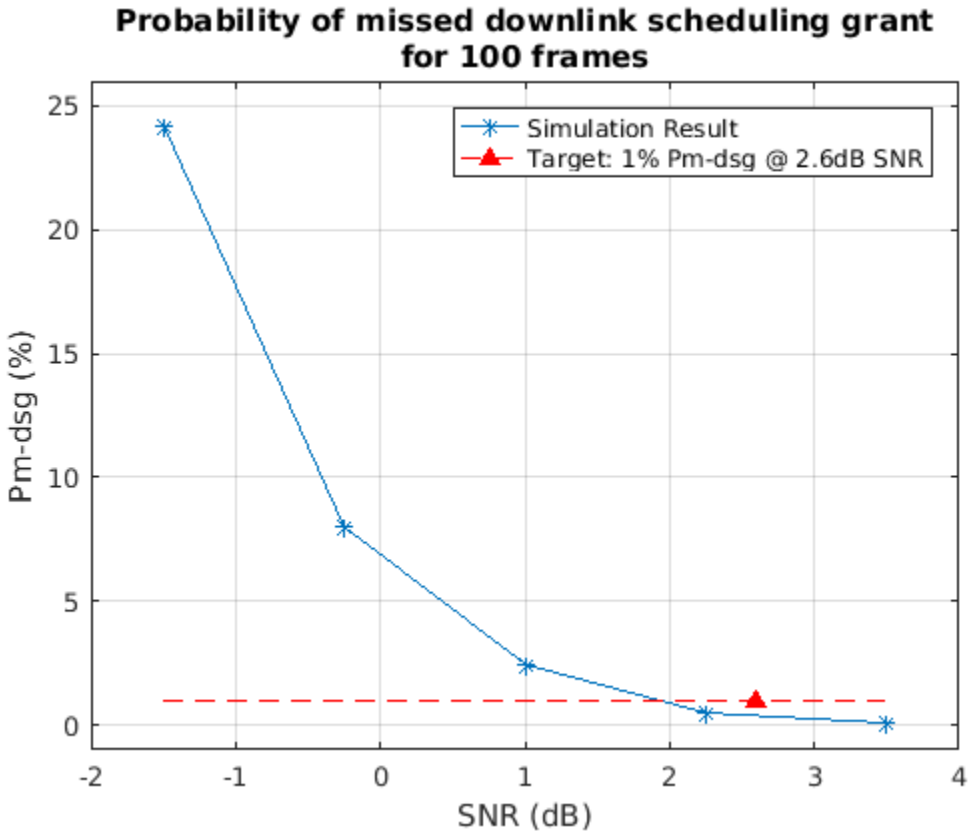
Plot Results

Finally, the test results are plotted. The probability of a missed downlink scheduling grant Pm-dsg is plotted for each SNR point. A marker is added at the target SNR and a Pm-dsg of 1%. A reference line is added for Pm-dsg of 1%, and observing the SNR at which the simulated performance crosses this line gives an indication of the SNR margin between the simulated performance and the test requirement.

```
plotResults(SNRIn,targetSNR,nFrames,Pmdsg);
```



The generated plot has been obtained with a low number of frames, therefore the results shown are not representative. A longer simulation obtained with 100 frames and including some additional SNR points produced the results shown below.



Selected Bibliography

1. 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception"
2. 3GPP TS 36.211 "Physical channels and modulation"
3. 3GPP TS 36.213 "Physical layer procedures"
4. 3GPP TS 36.331 "Radio Resource Control (RRC) Protocol specification"

Local Functions

The following local functions are used in this example:

- `beamformPerfectChannelEstimate`: apply EPDCCH beamforming to the perfect channel estimate
- `addOCNG`: add EPDCCH OCNG to the subframe resource grid
- `plotResults`: plot the results of the test

```
function hest = beamformPerfectChannelEstimate(hestperfect,rbs,ports,codebookIdxs)

    [K,L,nrxants,ntxants] = size(hestperfect);
    hest = zeros([K L nrxants 4]);
    for r = 1:length(rbs)
        resubs = (rbs(r)-1)*12 + (1:12).';
        allportW = zeros(4,ntxants);
        activeportW = arrayfun(@(x)lteDLPrecode(1,ntxants,'SpatialMux',x),codebookIdxs(r,:),'Uni
        allportW(ports,:) = cat(1,activeportW{:});
```

```
            hest(resubs,:,:,:) = reshape(reshape(hestperfect(resubs,:,:,:),[12*L*nrxants ntxants])*a
        end

    end

    function subframe = addOCNG(enb,epdcch,subframe)

        % Add OP.7 OCNG (TS 36.101 Annex A.5.1.7). Consists of a single PDSCH
        % in all resource blocks not used by the EPDCCH.
        ocngPdsch.Modulation = 'QPSK';
        ocngPdsch.RNTI = 0;
        if (enb.CellRefP==1)
            ocngPdsch.TxScheme = 'Port0';
        else
            ocngPdsch.TxScheme = 'TxDiversity';
        end
        ocngPRBs = setdiff((0:enb.NDLRB-1).',epdcch.EPDCCHPRBSet);
        [ocngPdschInd,ocngPdschInfo] = ltePDSCHIndices(enb,ocngPdsch,ocngPRBs);
        ocngPdschData = randi([0 1],1,ocngPdschInfo.G);
        ocngPdschPower = -3.0; % rho + sigma = -3.0 + 0.0
        ocngPdschSym = ltePDSCH(enb,ocngPdsch,ocngPdschData)*(10^(ocngPdschPower/20));
        subframe(ocngPdschInd) = ocngPdschSym;

        % Add control region OCNG reference channel (TS 36.101 Annex A.5).
        % Includes PDCCH and PHICH locations but avoids PCFICH locations, to
        % allow the CFI to be transmitted
        pcfichRegInd = ltePCFICHIndices(enb,'reg');
        enbPdcch = enb;
        enbPdcch.PHICHDuration = 'Normal';
        enbPdcch.Ng = 'Sixth';
        exreg = pcfichRegInd(:,1);
        ocngPdcchInd = ltePDCCHIndices(enbPdcch,exreg);
        NRE = size(ocngPdcchInd,1);
        ocngPdcchData = randi([0 1],1,NRE*2);
        ocngPdcchPower = -3.0; % rho + sigma = -3.0 + 0.0
        ocngPdcchSym = ltePDCCH(enbPdcch,ocngPdcchData,NRE/4)*(10^(ocngPdcchPower/20));
        subframe(ocngPdcchInd) = ocngPdcchSym;

    end

    function plotResults(SNRIn,targetSNR,nFrames,Pmdsg)

        figure;
        plot(SNRIn,Pmdsg,'-*')
        if (nFrames==1)
            frameStr = '1 frame';
        else
            frameStr = [num2str(nFrames) ' frames'];
        end
        title(sprintf('Probability of missed downlink scheduling grant\nfor %s',frameStr));
        xlabel('SNR (dB)'); ylabel('Pm-dsg (%)');
        grid on;
        hold on;
        plot(targetSNR,1,'--r^','MarkerFaceColor','r')
        plot(SNRIn, ones(1,numel(SNRIn)),'--r');
        legend('Simulation Result',sprintf('Target: 1%% Pm-dsg @ %0.1fdB SNR',targetSNR),'Location',
        xlim([min(SNRIn)-0.5 max(SNRIn)+0.5]);
        ylim([fix(min(Pmdsg)-2) fix(max(Pmdsg)+2)]);
```

```
end
```

Release 12 Downlink Carrier Aggregation Waveform Generation, Demodulation and Analysis

This example shows how multiple downlink carriers can be generated, aggregated and further demodulated using the LTE Toolbox™.

Introduction

This example models an LTE Release 12 waveform with Carrier Aggregation (CA). The number of subcarriers and their bandwidth can be specified as a parameter. An intra-band contiguous CA case is considered.

To generate an aggregated downlink waveform an eNodeB is configured for each component carrier. The component carrier parameters are calculated and used to generate a modulated waveform for each eNodeB configuration. All the CC modulated waveforms are resampled to a common sampling rate so they can be combined to create an aggregated waveform.

Number of Component Carriers and Bandwidths

A vector `NDLRB` specifies the number of resource blocks (RBs) for each component carrier (CC). The length of this vector corresponds to the number of CCs. The elements of `NDLRB` must be in the set {6, 15, 25, 50, 75, 100} RBs. TS 36.101 Table 5.6A.1-1 [1] lists the valid combinations of bandwidths for carrier aggregation.

```
% Configure the set of NDLRB values to describe the carriers to be
% aggregated
NDLRB = [50 75 100];

% Establish the number of component carriers
numCC = numel(NDLRB);

if numCC<2
    error('Please specify more than one CC bandwidth value.');
end
```

Component Carrier Configuration

A configuration structure is generated for each CC using `lteRMCDL`. The configuration structures for all CCs are stored in a cell array.

```
% Configure the number of subframes to generate
numSubframes = 10;

% CC configuration
enb = cell(1,numCC);
for i = 1:numCC
    switch NDLRB(i)
        case 6
            enb{i} = lteRMCDL('R.4');
        case 15
            enb{i} = lteRMCDL('R.5');
        case 25
            enb{i} = lteRMCDL('R.6');
        case 50
            enb{i} = lteRMCDL('R.7');
```

```
        case 75
            enb{i} = lteRMCDL('R.8');
        case 100
            enb{i} = lteRMCDL('R.9');
        otherwise
            fprintf('Not a valid number of resource blocks: %d\n',...
                NDLRB(i));
            return
    end
    enb{i}.NDLRB = NDLRB(i);
    enb{i}.Bandwidth = hNRBToBandwidth(NDLRB(i));
    enb{i}.TotSubframes = numSubframes;
    enb{i}.PDSCH.PRBSet = (0:enb{i}.NDLRB-1).';
    enb{i}.PDSCH.RVSeq = 0;
    enb{i}.NCellID = 10;
end
```

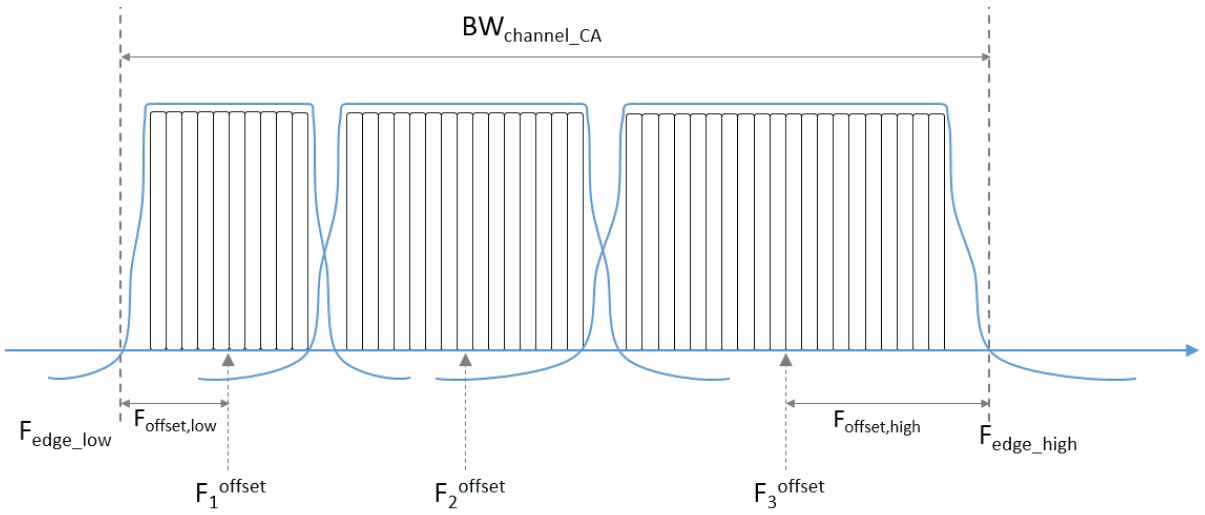
Channel Estimator Configuration

The channel estimator at the receiver end is parameterized using the structure `cec` defined below.

```
cec = struct;                           % Channel estimation config structure
cec.PilotAverage = 'UserDefined';       % Type of pilot symbol averaging
cec.FreqWindow = 15;                    % Frequency window size
cec.TimeWindow = 15;                    % Time window size
cec.InterpType = 'Cubic';               % 2D interpolation type
cec.InterpWindow = 'Centered';          % Interpolation window type
cec.InterpWinSize = 1;                  % Interpolation window size
```

Carrier Aggregation Parameter Calculation

To perform carrier aggregation the frequency parameters described in TS 36.101, Sections 5.6 and 5.7 [1] are calculated. These parameters are summarized in the following figure.



This results in three variables:

- `F_c` is a vector containing the center frequency of each CC
- `ccSpacing` contains the spacing between CC in MHz
- `BW_channel_CA` is the aggregated channel bandwidth of all CCs

In the code below we first calculate the value for all the CCs assuming the lower one is centered at baseband (0 Hz). Once the `BW_channel_CA` is calculated all the values are shifted so the center of the aggregated bandwidth is located at baseband (0 Hz).

```
% Define Delta_f1 parameter in MHz for nominal guard band and frequency
% offset calculations. In the downlink Delta_f1 is the subcarrier spacing
% (TS 36.101, Table 5.6A-1)
Delta_f1 = 0.015; % in MHz, LTE subcarrier spacing in MHz
maxBW = hNRBToBandwidth(max(NDLRB));

F_c = zeros(1,numCC);

ccSpacing = zeros(1,numCC-1); % CC spacing

%  Calculate nominal guard band, TS 36.101 5.6A-1
nominalGuardBand = 0.05*maxBW-0.5*Delta_f1;

% Initially assume lower carrier frequency is at baseband
F_c(1) = 0;

% Calculate CC spacing and carrier values
for k = 2:numCC
    ccSpacing(k-1) = hCarrierAggregationChannelSpacing( ...
        enb{k-1}.Bandwidth, enb{k}.Bandwidth);
    F_c(k) = F_c(k-1) + ccSpacing(k-1);
end

% Calculate lower and higher frequency offsets, TS 36.101 5.6A
F_offset_low = (0.18*NDLRB(1)+Delta_f1)/2 + nominalGuardBand;
F_offset_high = (0.18*NDLRB(end)+Delta_f1)/2 + nominalGuardBand;

% Calculate lower and higher frequency edges, TS 36.101 5.6A
F_edge_low = F_c(1) - F_offset_low;
F_edge_high = F_c(end) + F_offset_high;

% Calculate aggregated channel bandwidth, TS 36.101 5.6A
BW_channel_CA = F_edge_high - F_edge_low;
fprintf('BW_channel_CA: %0.4f MHz\n',BW_channel_CA);

% Calculate shift to center baseband
shiftToCenter = -1*(BW_channel_CA/2 + F_edge_low);

% Aggregated bandwidth centered at baseband
F_c = F_c + shiftToCenter;
F_edge_low = F_c(1) - F_offset_low;
F_edge_high = F_c(end) + F_offset_high;

% Display frequency band edges
fprintf('F_edge_low:  %0.4f MHz\n',F_edge_low);
fprintf('F_edge_high: %0.4f MHz\n',F_edge_high);
fprintf('F_offset_low:  %0.4f MHz\n',F_offset_low);
fprintf('F_offset_high: %0.4f MHz\n',F_offset_high);
```

```
% Display carrier frequencies
fprintf('\n');
for i = 1:numCC
    fprintf('Component Carrier %d:\n',i);
    fprintf('   Fc: %0.4f MHz\n', F_c(i));
end
```

```
BW_channel_CA: 44.6000 MHz
F_edge_low:  -22.3000 MHz
F_edge_high: 22.3000 MHz
F_offset_low:  5.5000 MHz
F_offset_high: 10.0000 MHz

Component Carrier 1:
   Fc: -16.8000 MHz
Component Carrier 2:
   Fc: -4.8000 MHz
Component Carrier 3:
   Fc: 12.3000 MHz
```

Required Oversampling Rate Calculation

The required oversampling factors for each component carrier `OSRs` are calculated for a common sampling rate for the aggregated signal.

```
% Bandwidth utilization of 85%
bwfraction = 0.85;

% Calculate sampling rates of the component carriers
CCSR = zeros(1,numCC);
for i = 1:numCC
    info = lteOFDMInfo(enb{i});
    CCSR(i) = info.SamplingRate;
end

% Calculate overall sampling rate for the aggregated signal

% Calculate the oversampling ratio (for the largest BW CC) to make sure the
% signal occupies 85% (bwfraction) of the total bandwidth
OSR = (BW_channel_CA/bwfraction)/(max(CCSR)/1e6);
% To simplify the resampling operation choose an oversampling ratio which
% is a power of 2: calculate the next power of two above OSR
OSR = 2^ceil(log2(OSR));
SR = OSR*max(CCSR);
fprintf('\nOutput sample rate: %0.4f Ms/s\n\n',SR/1e6);

% Calculate individual oversampling factors for the component carriers
OSRs = SR./CCSR;
```

```
Output sample rate: 61.4400 Ms/s
```

Waveform Generation and Carrier Aggregation

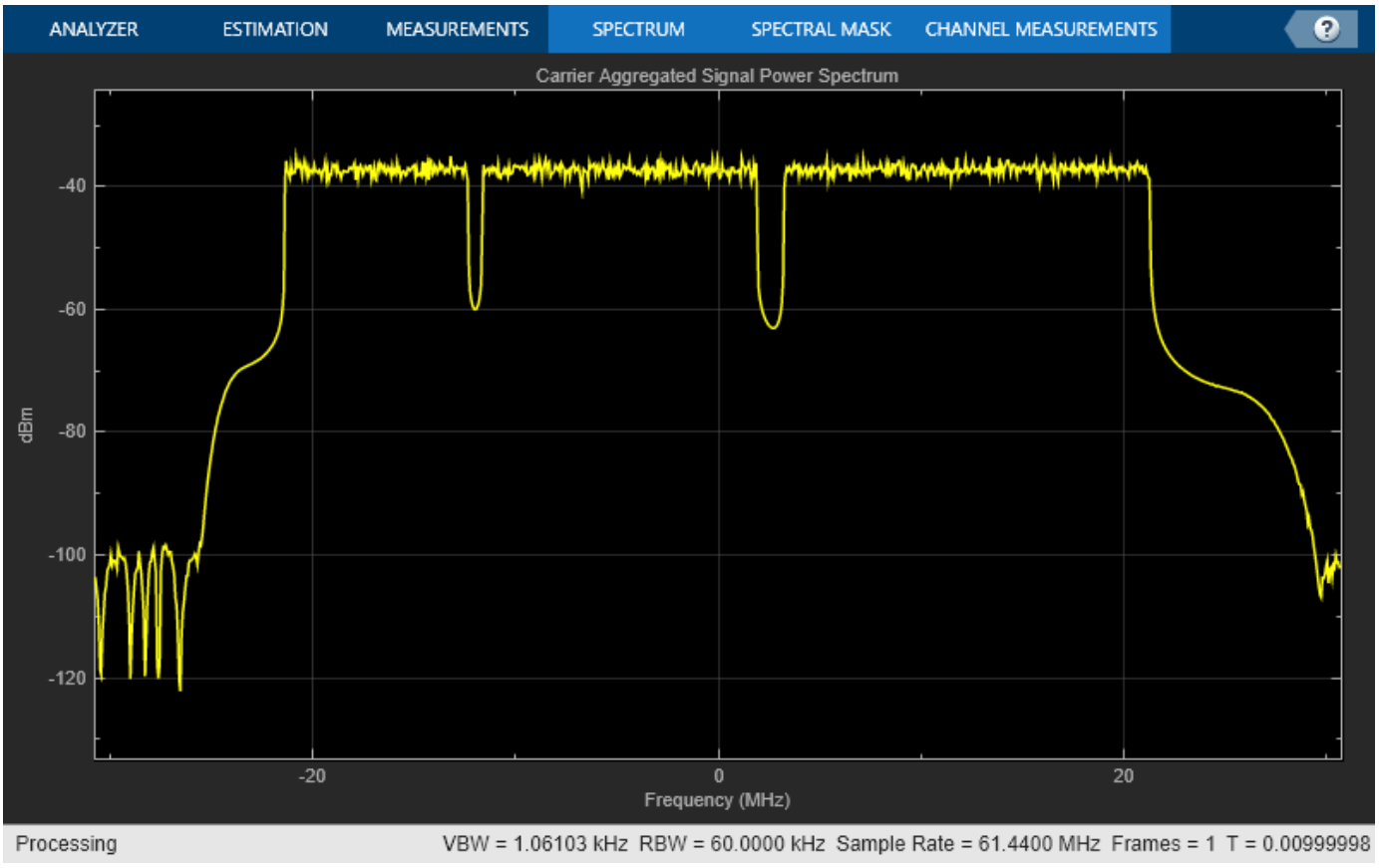
`lteRMCDLTool` is used to generate the waveform for each CC. Each of them is resampled to a common sampling rate. Finally, Multiband Combiner is used to frequency modulate each CC to the appropriate center frequency and add them together to form the aggregated signal.

```
% Generate component carriers
tx = cell(1,numCC);
for i = 1:numCC
    tx{i} = lteRMCDLTool(enb{i},randi([0 1],1000,1));
    tx{i} = resample(tx{i},OSRs(i),1)/OSRs(i);
end

% Aggregate all CC. comm.MultibandCombiner applies the frequency offsets
% and adds the resulting signals together.
sigAggregator = comm.MultibandCombiner(InputSampleRate = SR, ...
    FrequencyOffsets = F_c*1e6, ...
    OutputSampleRateSource = 'Property', ...
    OutputSampleRate = SR);

waveform = sigAggregator([tx{:}]);
```

Plot Carrier Aggregation Waveform Spectrum

Display the power spectrum of the carrier aggregated signal. In the spectrum the individual carrier bandwidths are visible. Note that the center of the aggregated bandwidth is located at baseband (0 Hz), i.e. in this example the signal is not modulated to RF.

```
specPlot = spectrumAnalyzer('SampleRate',SR,...
                            'Title','Carrier Aggregated Signal Power Spectrum');
specPlot(waveform);
```



Demodulation and Filtering of CC of Interest

This section demodulates, filters and downsamples one of the CCs. The steps followed are:

- Demodulate the CC of interest and bring it to baseband (0 Hz).
- Filter out neighboring CCs and downsample. A suitable filter is designed to remove the unwanted neighboring CCs in the downsampling process. The filter design will have an impact on the quality and EVM of the recovered signal. Filters may need to be tweaked for different values of bandwidth and CC to demodulate.

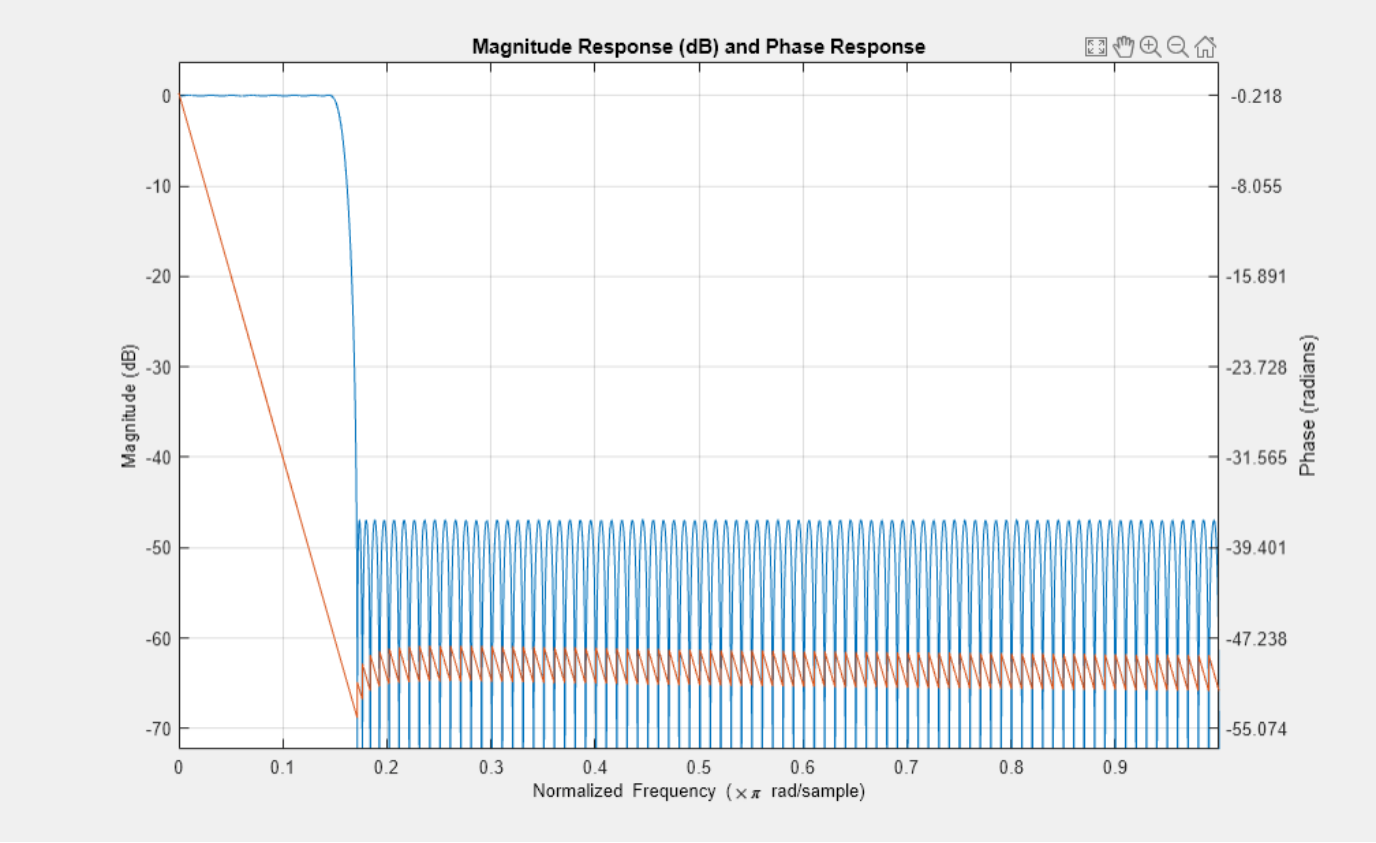
We start by selecting the CC to demodulate and designing the appropriate downsampling filter. Passband and stopband frequencies are calculated.

```
% Select CC to demodulate
CCofInterest = 1;
if CCofInterest > numCC || CCofInterest <= 0
    error('Cannot demodulate CC number %d, there are %d CCs\n',...
        CCofInterest, numCC) ;
end

% Define downsampling filter order
filterOrder = 201;

% Precalculate passband and stopband values for all CC
firPassbandVec = (0.18*NDLRB+Delta_f1)/2 / (SR/1e6/2);
firStopbandVec = hCarrierAggregationStopband(ccSpacing,NDLRB,SR);

% Define passband and stopband for carrier of interest
firPassband = firPassbandVec(CCofInterest);
firStopband = firStopbandVec(CCofInterest);

% Extract and decode CC of interest
fprintf(1,'Extracting CC number %d: \n', CCofInterest);

% Pad signal with zeros to take into account filter transient response
% length
waveform = [waveform; zeros(filterOrder+1,size(waveform,2))];

% Demodulate carrier of interest
demodulatedCC = ...
    hCarrierAggregationModulate(waveform,SR,-F_c(CCofInterest)*1e6);

% Downsampling is done in two stages if the filter is too narrow. This
% eases the filter design requirements. If this is the case an initial
% downsampling factor of 4 is applied. You may want to consider a different
% filter design in this initial stage if the quality of the resulting
% signal is not acceptable.
if (firStopband < 0.1)
    % Down-sample by 4
    SRC = 4;
    demodulatedCC = resample(demodulatedCC,1,SRC);
    % Update pass and stopband to take first downsampling into account
    firPassband = firPassband * SRC;
    firStopband = firStopband * SRC;
else
    % No pre-filter
    SRC = 1;
```

```
end

% Design lowpass filter to filter out CC of interest
frEdges = [0 firPassband firStopband 1];
firFilter = dsp.FIRFilter;
firFilter.Numerator = firpm(filterOrder,frEdges,[1 1 0 0]);

Extracting CC number 1:
```

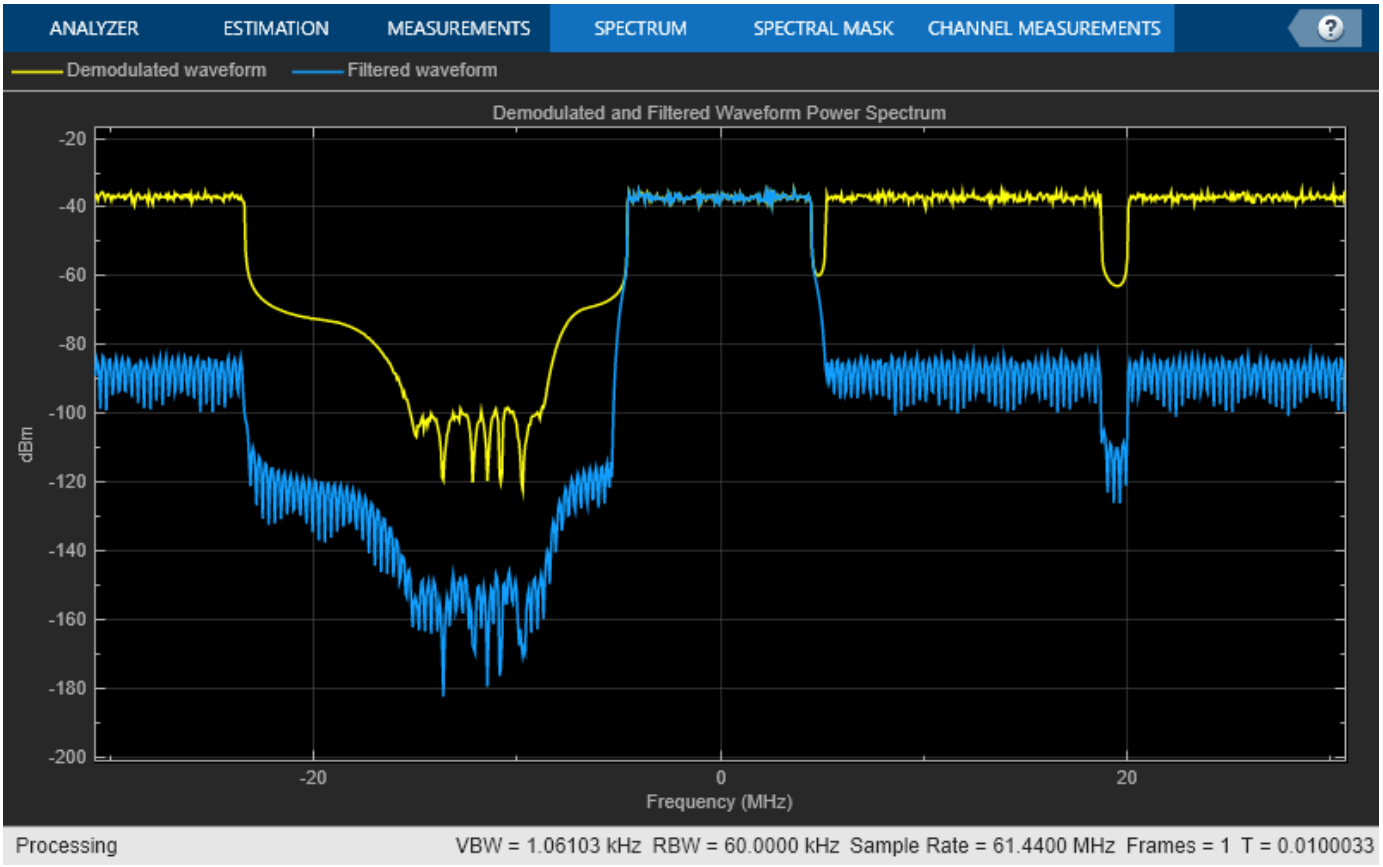
The response of the designed filter is displayed.

```
fvtool(firFilter,'Analysis','freq');
```



The demodulated waveform is then filtered to extract the CC of interest. The spectrum of the demodulated waveform before and after the filtering is plotted.

```
% Filter the signal to extract the component of interest
rxWaveform  = firFilter(demodulatedCC);

% Plot the demodulated and filtered signal spectra
filteredSpecPlot = spectrumAnalyzer('SampleRate',SR,'Title','Demodulated and Filtered Waveform Po
                                    'ChannelNames',{'Demodulated waveform' 'Filtered waveform'},
filteredSpecPlot([demodulatedCC, rxWaveform]);
```



At this point the filtered waveform can be downsampled to its baseband rate.

```
rxWaveform = downsample(rxWaveform,OSRs(CCofInterest)/SRC);
```

Synchronization

Synchronization is applied to the resulting signal.

```
% Parameters for CC of interest
eNodeB = enb{CCofInterest};
PDSCH = eNodeB.PDSCH;

% Synchronize received waveform
offset = lteDLFrameOffset(eNodeB, rxWaveform);
rxWaveform = rxWaveform(1+offset:end, :);
```

EVM Measurements and PDSCH Decoding

The code below provides per subframe and average EVM measurements. Plots with the EVM versus time, resource block and subcarriers are also displayed.

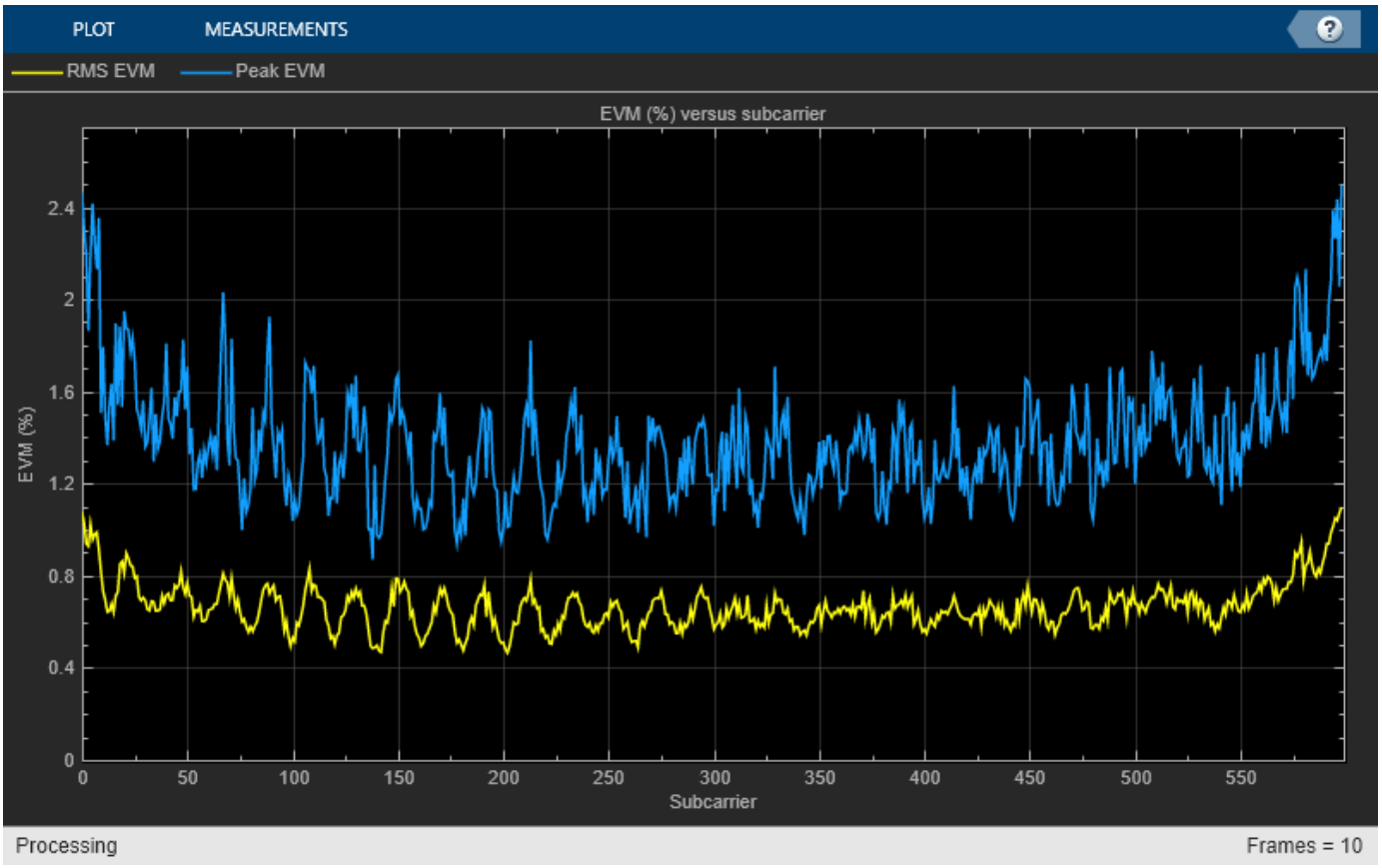
The PDSCH of the recovered signal is decoded and the resulting CRC is checked for errors.

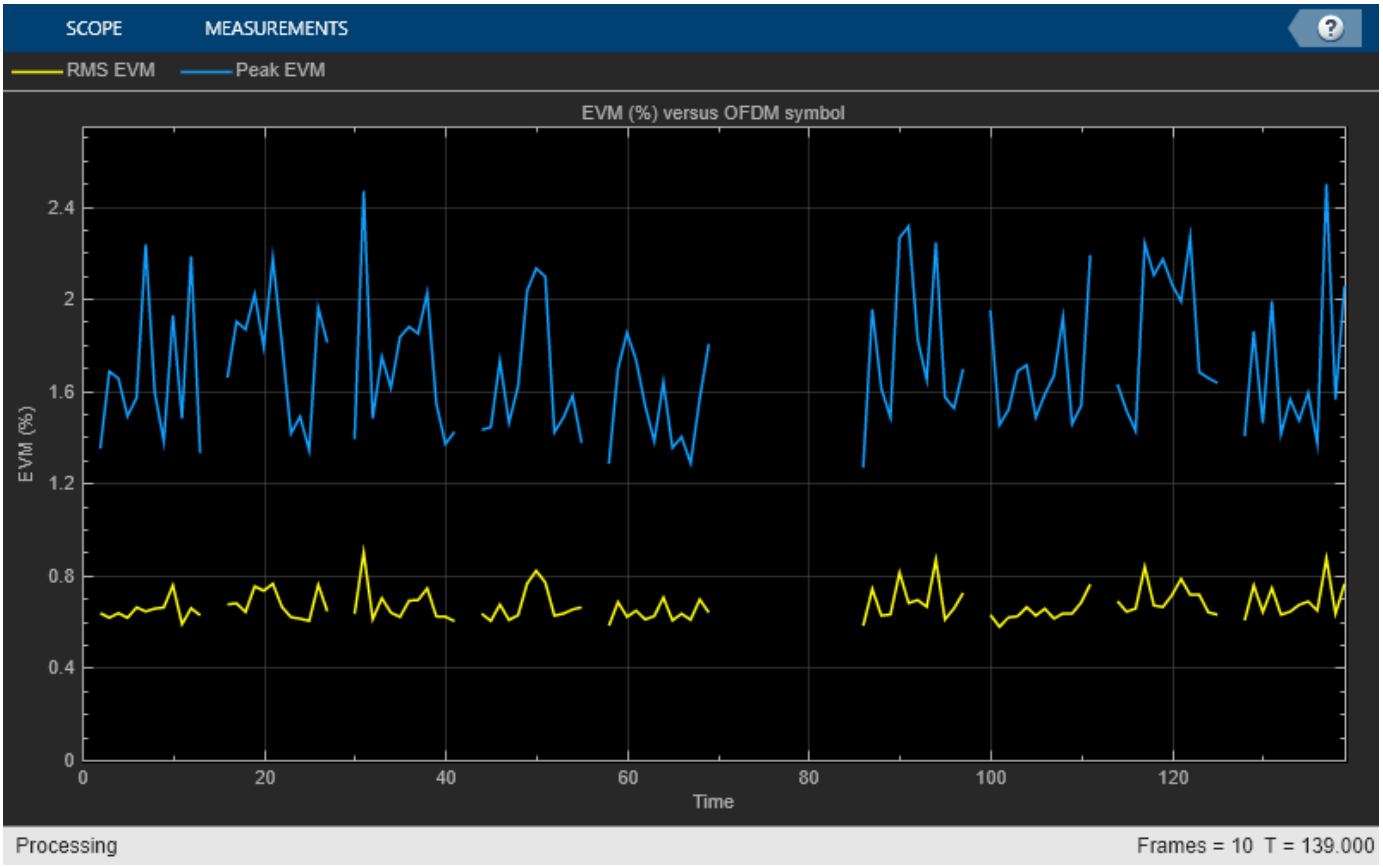
```
% channel estimation structure for EVM measurement
cecEVM = cec;
cecEVM.PilotAverage = 'TestEVM';
[evmmeas, plots] = hPDSCHEVM(enb{CCofInterest},cecEVM,rxWaveform);
```

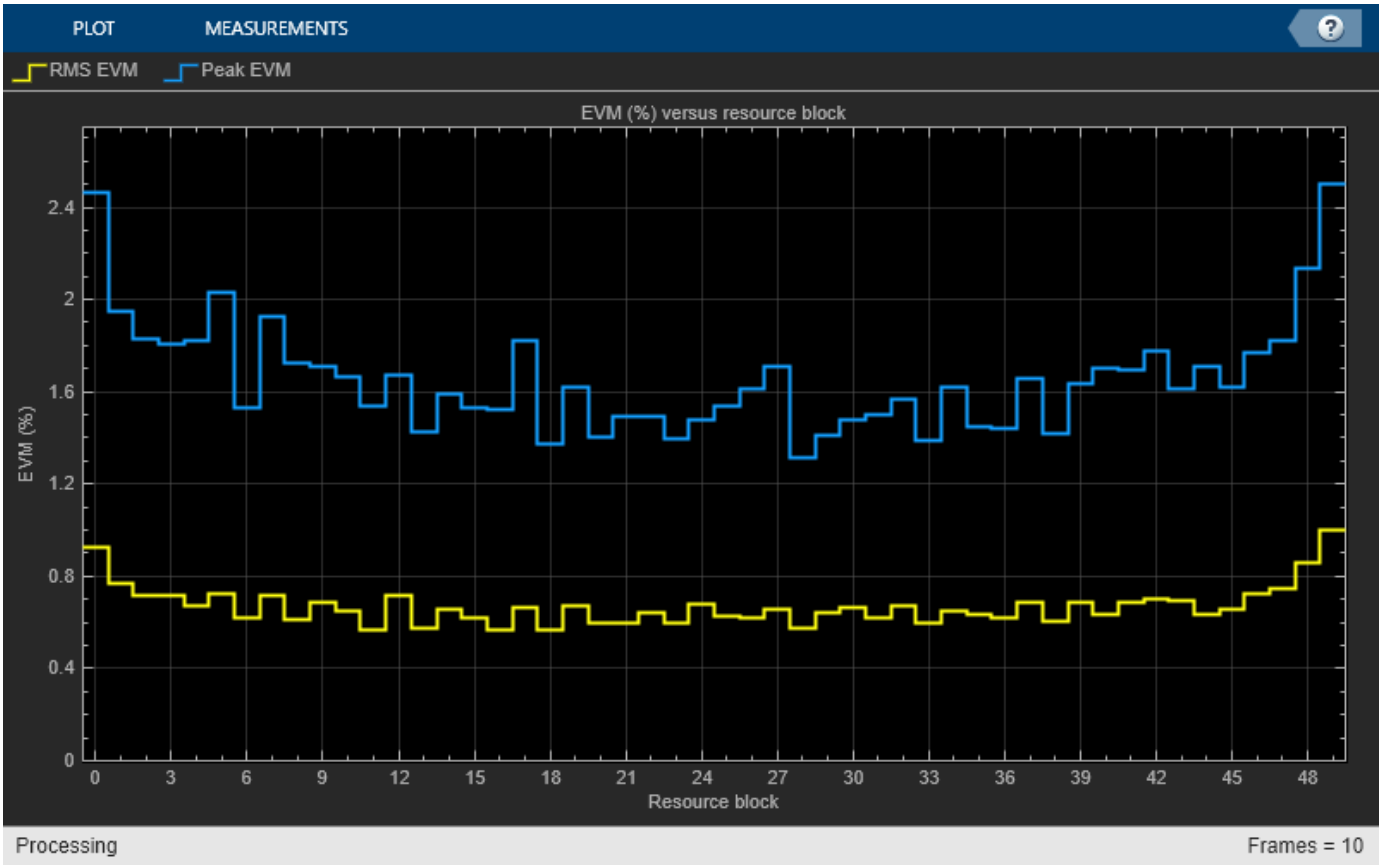
```
% OFDM demodulation
rxGrid = lteOFDMDemodulate(eNodeB,rxWaveform);

% Get the number of received subframes and OFDM symbols per subframe
dims = lteOFDMInfo(eNodeB);
samplesPerSubframe = dims.SamplingRate/1000;
nRxSubframes = floor(size(rxWaveform, 1)/samplesPerSubframe);
eNodeB.TotSubframes = 1;
resGridSize = lteResourceGridSize(eNodeB);
L = resGridSize(2);

disp('Decode transmitted subframes and check CRC.');

for n=0:nRxSubframes-1

    % extract subframe
    rxSubframe = rxGrid(:,(1:L)+(n*L),:);

    % transport block size for current subframe
    eNodeB.NSubframe = n;
    trBlkSize = PDSCH.TrBlkSizes(n+1);

    % Channel estimation
    [estChannelGrid, noiseEst] = lteDLChannelEstimate( ...
        eNodeB, cec, rxSubframe);

    % Perform deprecoding, layer demapping, demodulation and descrambling
    % on the received data using the channel estimate
    [rxEncodedBits, pdschsymbols] = ltePDSCHDecode(eNodeB,PDSCH, ...
        rxSubframe*(10^(-PDSCH.Rho/20)),estChannelGrid,noiseEst);

    % Decode DownLink Shared Channel (DL-SCH)
    [decbits,crc] = ...
        lteDLSCHDecode(eNodeB,PDSCH,trBlkSize,rxEncodedBits{1});

    if crc
        disp(['Subframe ' num2str(n) ': CRC failed']);
    else
        disp(['Subframe ' num2str(n) ': CRC passed']);
    end
end
```

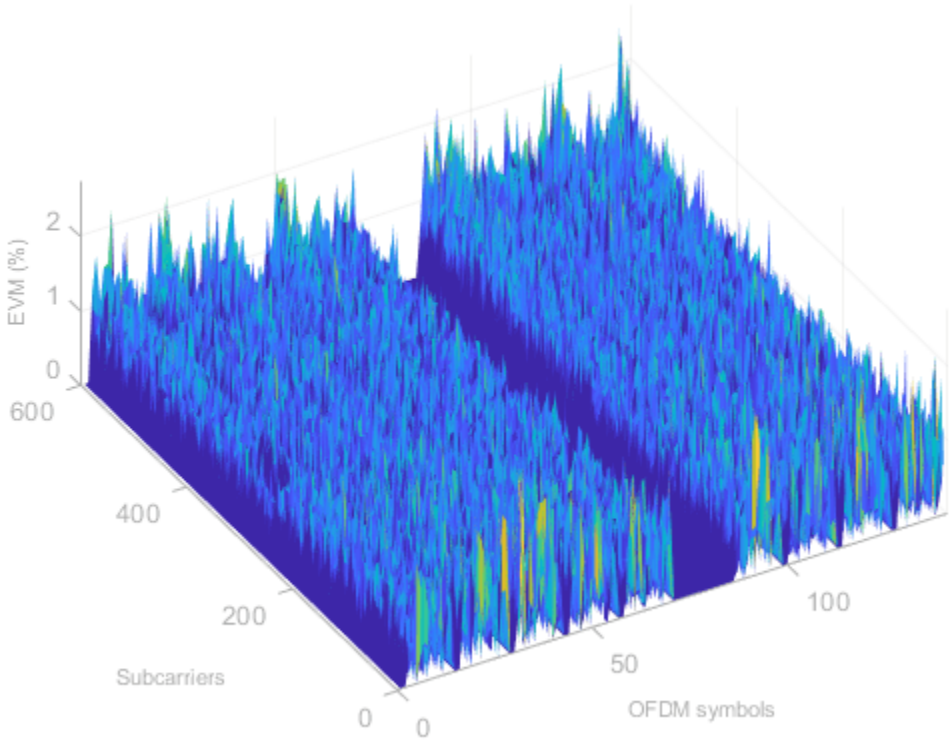
```
Low edge EVM, subframe 0: 0.647%
High edge EVM, subframe 0: 0.629%
Low edge EVM, subframe 1: 0.682%
High edge EVM, subframe 1: 0.650%
Low edge EVM, subframe 2: 0.680%
High edge EVM, subframe 2: 0.676%
Low edge EVM, subframe 3: 0.676%
High edge EVM, subframe 3: 0.641%
Low edge EVM, subframe 4: 0.640%
High edge EVM, subframe 4: 0.648%
Low edge EVM, subframe 6: 0.698%
High edge EVM, subframe 6: 0.642%
Low edge EVM, subframe 7: 0.645%
High edge EVM, subframe 7: 0.649%
Low edge EVM, subframe 8: 0.700%
```

```
High edge EVM, subframe 8: 0.647%
Low edge EVM, subframe 9: 0.694%
High edge EVM, subframe 9: 0.685%
Averaged low edge EVM, frame 0: 0.674%
Averaged high edge EVM, frame 0: 0.652%
Averaged EVM frame 0: 0.674%
Averaged overall EVM: 0.674%
Decode transmitted subframes and check CRC.
Subframe 0: CRC passed
Subframe 1: CRC passed
Subframe 2: CRC passed
Subframe 3: CRC passed
Subframe 4: CRC passed
Subframe 5: CRC passed
Subframe 6: CRC passed
Subframe 7: CRC passed
Subframe 8: CRC passed
Subframe 9: CRC passed
```




SCOPE
MEASUREMENTS
RMS EVM
Peak EVM
EVM (%) versus OFDM symbol
EVM (%)
2.4
2
1.6
1.2
0.8
0.4
0
0
20
40
60
80
100
120
Time
Processing
Frames = 10 T = 139.000


PLOT
MEASUREMENTS
RMS EVM
Peak EVM
EVM (%) versus resource block
EVM (%)
2.4
2
1.6
1.2
0.8
0.4
0
0
3
6
9
12
15
18
21
24
27
30
33
36
39
42
45
48
Resource block
Processing
Frames = 10



Bibliography

1 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception"

PUCCH1a Multi User ACK Missed Detection Probability Conformance Test

This example shows how to use the LTE Toolbox™ to measure the probability of Acknowledgment (ACK) missed detection for multiuser Physical Uplink Control Channel (PUCCH) 1a. The test conditions are defined in TS36.104 Section 8.3.4.1 [1].

Introduction

In this example, four different UEs are configured, each of which transmits a PUCCH format 1a signal. Appropriate Demodulation Reference Signals (DRS) are also generated. For each considered SNR value, the transmitted signals are fed through different channels and added, together with Gaussian noise. This simulates the reception of the signals from four different UEs at a base station. The receiver decodes the PUCCH1a for the user of interest and the probability of ACK missed detection is measured. This example uses a simulation length of 10 subframes. This value has been chosen to speed up the simulation. A larger value should be chosen to obtain more accurate results. The target defined in TS36.104 Section 8.3.4.1 [1] for 1.4 MHz bandwidth (6 Resource Blocks-RBs) and a single transmit antenna is an ACK missed detection probability not exceeding 1% at an SNR of -4.1 dB. The test is defined for 1 transmit antenna.

```
numSubframes = 10;                              % Number of subframes
SNRdB = [-16.1 -12.1 -8.1 -4.1 -0.1];            % SNR range
NTxAnts = 1;                                    % Number of transmit antennas
```

UE 1 Configuration

Create a User Equipment (UE) configuration structure. These parameters are common for all the users.

```
ue = struct;                    % UE config structure
ue.NULRB = 6;                   % 6 resource blocks (1.4 MHz)
ue.CyclicPrefixUL = 'Normal'; % Normal cyclic prefix
ue.Hopping = 'Off';           % No frequency hopping
ue.NCellID = 150;             % Cell id as specified in TS36.104 Appendix A9
ue.Shortened = 0;             % No SRS transmission
ue.NTxAnts = NTxAnts;
```

PUCCH 1a Configuration

We intend to transmit an ACK via a PUCCH of Format 1, so we create an appropriate configuration structure `pucch`. We give the cell an arbitrary identification number, and set up PUCCH Resource Indices, transmission powers and channel seeds for each user. A different random channel seed for each user ensures that each experiences different channel conditions.

```
% Hybrid Automatic Repeat Request (HARQ)  indicator bit set to one. Only
% one bit is required for PUCCH 1a
ACK = 1;
pucch = struct;  % PUCCH config structure
% Set the size of resources allocated to PUCCH format 2. This affects the
% location of PUCCH 1 transmission
pucch.ResourceSize = 0;
% Delta shift PUCCH parameter as specified in TS36.104 Appendix A9 [ <#8 1> ]
pucch.DeltaShift = 2;
% Number of cyclic shifts used for PUCCH format 1 in resource blocks with a
```

```
% mixture of formats 1 and 2. This is the N1cs parameter as specified in
% TS36.104 Appendix A9
pucch.CyclicShifts = 0;
% Vector of PUCCH resource indices for all UEs as specified in TS36.104
% Appendix A9
usersPUCCHindices = [2 1 7 14];
% PUCCH power for all UEs as specified in TS36.104 Appendix A9
usersPUCCHpower = [0 0 -3 3];
```

Propagation Channel Configuration

This section of the code configures the propagation channels for the four UEs. The parameters are specified in the tests described in TS36.104 Section 8.3.4.1 [1] and are: ETU 70Hz and 2 receive, i.e. base station, antennas is configured to be 2. Each UE will see a different channel, therefore a different seed is used in each case. This is specified in the `ueChannelSeed` parameter.

```
channel = struct;                   % Channel config structure
channel.NRxAnts = 2;                % Number of receive antennas
channel.DelayProfile = 'ETU';       % Channel delay profile
channel.DopplerFreq = 70.0;         % Doppler frequency in Hz
channel.MIMOCorrelation = 'Low';    % Low MIMO correlation
channel.NTerms = 16;                % Oscillators used in fading model
channel.ModelType = 'GMEDS';        % Rayleigh fading model type
channel.InitPhase = 'Random';       % Random initial phases
channel.NormalizePathGains = 'On';  % Normalize delay profile power
channel.NormalizeTxAnts = 'On';     % Normalize for transmit antennas

% SC-FDMA modulation information: required to get the sampling rate
info = lteSCFDMAInfo(ue);
channel.SamplingRate = info.SamplingRate;   % Channel sampling rate

% Channel seeds for each of the 4 UEs (arbitrary)
ueChannelSeed = [11 1 7 14];
```

Channel Estimator Configuration

The channel estimator is configured using a structure. Here cubic interpolation will be used with an averaging window of 9x9 resource elements.

```
cec = struct;        % Channel estimation config structure
cec.TimeWindow = 9;  % Time averaging window size in resource elements
cec.FreqWindow = 9;  % Frequency averaging window size in resource elements
cec.InterpType = 'cubic';         % Cubic interpolation
cec.PilotAverage = 'UserDefined'; % Type of pilot averaging
```

Simulation Loop for Configured SNR Points

A loop is used to run the simulation for a set of SNR points, given by the vector `SNRdB`. The SNR vector configured here is a range of SNR points including an SNR point at -4.1dB, the SNR at which the test requirement for ACK detection rate (99%) is to be achieved.

```
% Preallocate memory for missed detection probability vector
PMISS = zeros(size(SNRdB));
for nSNR = 1:length(SNRdB)

    % Detection failures counter
    missCount = 0;
    falseCount = 0;
```

```
% Noise configuration
SNR = 10^(SNRdB(nSNR)/20);              % Convert dB to linear
% The noise added before SC-FDMA demodulation will be amplified by the
% IFFT. The amplification is the square root of the size of the IFFT.
% To achieve the desired SNR after demodulation the noise power is
% normalized by this value. In addition, because real and imaginary
% parts of the noise are created separately before being combined into
% complex additive white Gaussian noise, the noise amplitude must be
% scaled by 1/sqrt(2*ue.NTxAnts) so the generated noise power is 1.
N = 1/(SNR*sqrt(double(info.Nfft)))/sqrt(2.0*ue.NTxAnts);
% Set the type of random number generator and its seed to the default
% value
rng('default');

% Subframe and user loops
% We now enter two further loops to process multiple subframes and
% create each of the users' transmissions. The fading process time
% offset, InitTime, is also generated for the current subframe

offsetused = 0;
for nsf = 1:numSubframes

    % Channel state information: set the init time to the correct value
    % to guarantee continuity of the fading waveform
    channel.InitTime = (nsf-1)/1000;

    % Loop for each user
    for user = 1:4

        % Create resource grid
        ue.NSubframe = mod(nsf-1,10);
        txgrid = lteULResourceGrid(ue);

        % Configure resource index for this user
        pucch.ResourceIdx = usersPUCCHindices(user);

        % ACK bit to transmit for the 1st (target) user, the PUCCH
        % Format 1 carries the Hybrid ARQ (HARQ) indicator ACK and for
        % other users it carries a random HARQ indicator. As there is a
        % single indicator, the transmissions will be of Format 1a. The
        % PUCCH Format 1 DRS carries no data.
        if (user==1)
            txACK = ACK;
        else
            txACK = randi([0 1],1,1);
        end

        % Generate PUCCH 1 and its DRS
        % Different users have different relative powers
        pucch1Sym = ltePUCCH1(ue,pucch,txACK)* ...
            10^(usersPUCCHpower(user)/20);
        pucch1DRSSym = ltePUCCH1DRS(ue,pucch)* ...
            10^(usersPUCCHpower(user)/20);

        % Generate indices for PUCCH 1 and its DRS
        pucch1Indices = ltePUCCH1Indices(ue,pucch);
        pucch1DRSIndices = ltePUCCH1DRSIndices(ue,pucch);
```

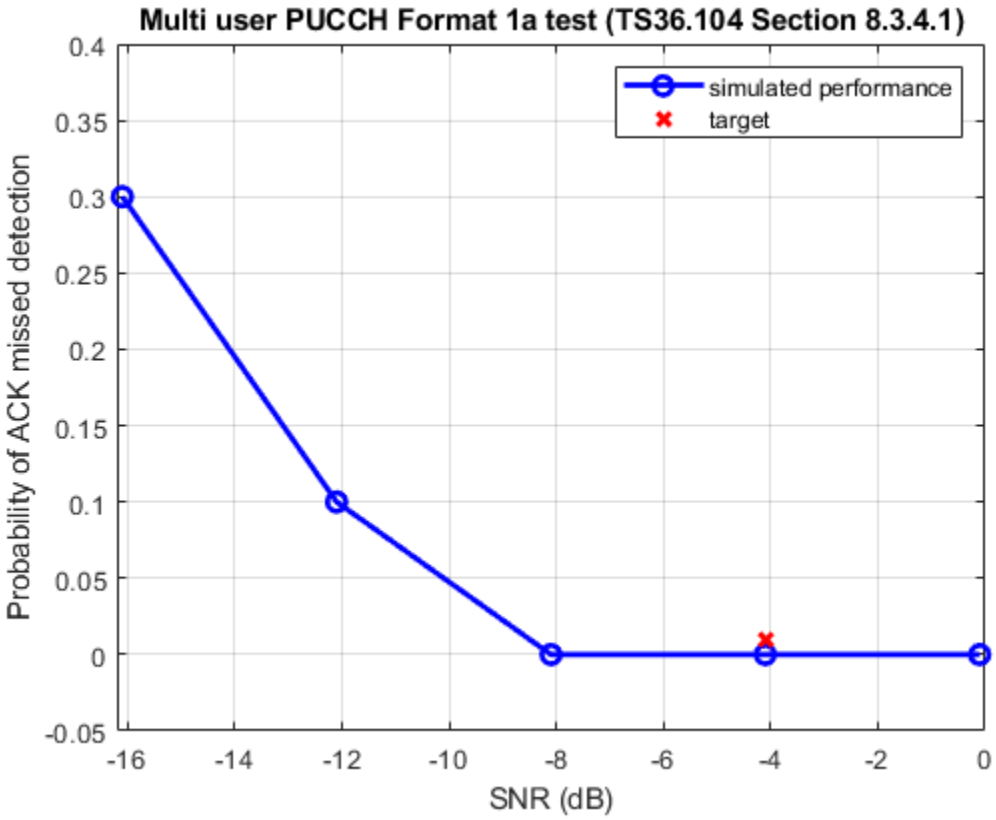
```
    % Map PUCCH 1 and PUCCH 1 DRS to the resource grid
    if (~isempty(txACK))
        txgrid(pucch1Indices) = pucch1Sym;
        txgrid(pucch1DRSIndices) = pucch1DRSSym;
    end

    % SC-FDMA modulation
    txwave = lteSCFDMAModulate(ue,txgrid);

    % Channel modeling and superposition of received signals.
    % The additional 25 samples added to the end of the waveform
    % are to cover the range of delays expected from the channel
    % modeling (a combination of implementation delay and channel
    % delay spread). On each iteration of the loop we accumulate
    % the sum of each transmitted signal, simulating the reception
    % of all four users at the base station.
    channel.Seed = ueChannelSeed(user);
    if (user==1)
        rxwave = lteFadingChannel(channel,[txwave; zeros(25,NTxAnts)]);
    else
        rxwave = rxwave + ...
            lteFadingChannel(channel,[txwave; zeros(25,NTxAnts)]);
    end
end

% Receiver

% Add AWGN noise at the receiver
noise = N*complex(randn(size(rxwave)),randn(size(rxwave)));
rxwave = rxwave + noise;

% Use the resource indices for the user of interest
pucch.ResourceIdx = usersPUCCHindices(1);

% Synchronization
% The uplink frame timing estimate for UE1 is calculated using
% the PUCCH 1 DRS signals and then used to demodulate the
% SC-FDMA signal.
% An offset within the range of delays expected from the channel
% modeling (a combination of implementation delay and channel
% delay spread) indicates success.
offset = lteULFrameOffsetPUCCH1(ue,pucch,rxwave);
if (offset<25)
    offsetused = offset;
end

% SC-FDMA demodulation
% The resulting grid (rxgrid) is a 3-dimensional matrix. The number
% of rows represents the number of subcarriers. The number of
% columns equals the number of SC-FDMA symbols in a subframe. The
% number of subcarriers and symbols is the same for the returned
% grid from lteSCFDMADemodulate as the grid passed into
% lteSCFDMAModulate. The number of planes (3rd dimension) in the
% grid corresponds to the number of receive antenna.
rxgrid = lteSCFDMADemodulate(ue,rxwave(1+offsetused:end,:));

% Channel estimation
```

```
            [H,n0] = lteULChannelEstimatePUCCH1(ue,pucch,cec,rxgrid);

            % PUCCH 1 indices for UE of interest
            pucch1Indices = ltePUCCH1Indices(ue,pucch);

            % Extract resource elements (REs) corresponding to the PUCCH 1 from
            % the given subframe across all receive antennas and channel
            % estimates
            [pucch1Rx,pucch1H] = lteExtractResources(pucch1Indices,rxgrid,H);

            % Minimum Mean Squared Error (MMSE) Equalization
            eqgrid = lteULResourceGrid(ue);
            eqgrid(pucch1Indices) = lteEqualizeMMSE(pucch1Rx,pucch1H,n0);

            % PUCCH 1 decoding
            rxACK = ltePUCCH1Decode(ue,pucch,1,eqgrid(pucch1Indices));

            % Detect missed (empty rxACK) or incorrect HARQ-ACK (compare
            % against transmitted ACK.
            if (isempty(rxACK) || any(rxACK~=ACK))
                missCount = missCount + 1;
            end


        end

        PMISS(nSNR) = missCount/numSubframes;

    end
```

Results

Finally we plot the simulated results against the target performance as stipulated in the standard.

```
plot(SNRdB,PMISS,'b-o','LineWidth',2,'MarkerSize',7);
grid on;
hold on;
plot(-4.1,0.01,'rx','LineWidth',2,'MarkerSize',7);
xlabel('SNR (dB)');
ylabel('Probability of ACK missed detection');
title('Multi user PUCCH Format 1a test (TS36.104 Section 8.3.4.1)');
axis([SNRdB(1)-0.1 SNRdB(end)+0.1 -0.05 0.4]);
legend('simulated performance','target');
```



Selected Bibliography

1 3GPP TS 36.104 "Base Station (BS) radio transmission and reception"

PUCCH1a ACK Missed Detection Probability Conformance Test

This example measures the Acknowledgment (ACK) missed detection probability using the LTE Toolbox™ under the single user Physical Uplink Control Channel (PUCCH1a) conformance test conditions as defined in TS 36.104 Section 8.3.2.1.

Introduction

This example uses a simulation length of 10 subframes. This value has been chosen to speed up the simulation. A higher value should be chosen to obtain more accurate results. The probability of erroneous ACK detection is calculated for a number of SNR point. The target defined in TS36.104 Section 8.3.2.1 [1] for 1.4 MHz bandwidth (6 RBs) and a single transmit antenna is an ACK missed detection probability not exceeding 1% at an SNR of -4.2 dB. The test is defined for 1 transmit antenna.

```
NSubframes = 10;                              % Number of subframes
SNRIn = [-10.2 -8.2 -6.2 -4.2 -2.2];          % SNR range in dB
NTxAnts = 1;                                  % Number of transmit antennas
```

UE Configuration

```
ue = struct;                                  % UE config structure
ue.NULRB = 6;                                 % 6 resource blocks (1.4 MHz)
ue.CyclicPrefixUL = 'Extended';               % Extended cyclic prefix
ue.Hopping = 'Off';                           % No frequency hopping
ue.NCellID =  9;
ue.Shortened = 0;                             % No SRS transmission
ue.NTxAnts = NTxAnts;
```

PUCCH 1a Configuration

```
% Hybrid ARQ indicator bit set to one. Only one bit is required for PUCCH
% 1a
ACK = 1;
pucch = struct;                             % PUCCH config structure
% Set the size of resources allocated to PUCCH format 2. This affects the
% location of PUCCH 1 transmission
pucch.ResourceSize = 0;
pucch.DeltaShift = 1;                         % Delta shift PUCCH parameter
% Number of cyclic shifts used for PUCCH format 1 in resource blocks with a
% mixture of formats 1 and 2. This is the N1cs parameter
pucch.CyclicShifts = 0;
% Vector of PUCCH resource indices, one per transmission antenna. This is
% the n2pucch parameter
pucch.ResourceIdx = 0:ue.NTxAnts-1;
```

Propagation Channel Configuration

Configure the channel model with the parameters specified in the tests described in TS36.104 Section 8.3.2.1 [1].

```
channel = struct;                           % Channel config structure
channel.NRxAnts = 2;                        % Number of receive antennas
channel.DelayProfile = 'ETU';               % Channel delay profile
channel.DopplerFreq = 70.0;                 % Doppler frequency in Hz
channel.MIMOCorrelation = 'Low';            % Low MIMO correlation
```

```
channel.NTerms = 16;                    % Oscillators used in fading model
channel.ModelType = 'GMEDS';            % Rayleigh fading model type
channel.Seed = 13;                      % Channel seed
channel.InitPhase = 'Random';           % Random initial phases
channel.NormalizePathGains = 'On';      % Normalize delay profile power
channel.NormalizeTxAnts = 'On';         % Normalize for transmit antennas

% SC-FDMA modulation information: required to get the sampling rate
scfdmaInfo = lteSCFDMAInfo(ue);
channel.SamplingRate = scfdmaInfo.SamplingRate;   % Channel sampling rate
```

Channel Estimator Configuration

The channel estimator is configured using a structure `cec`. Here cubic interpolation will be used with an averaging window of 12-by-1 Resource Elements (REs). This configures the channel estimator to use a special mode which ensures the ability to despread and orthogonalize the different overlapping PUCCH transmissions.

```
cec = struct;                     % Channel estimation config structure
cec.PilotAverage = 'UserDefined'; % Type of pilot averaging
cec.FreqWindow = 12;              % Frequency averaging window in REs (special mode)
cec.TimeWindow = 1;               % Time averaging window in REs (Special mode)
cec.InterpType = 'cubic';         % Cubic interpolation
```

Simulation Loop for Configured SNR Points

For each SNR point the loop below calculates the probability of successful ACK detection using information obtained from `NSubframes` consecutive subframes. The following operations are performed for each subframe and SNR values:

- Create an empty resource grid
- Generate and map PUCCH 1 and its Demodulation Reference Signal (DRS) to the resource grid
- Apply SC-FDMA modulation
- Send the modulated signal through the channel
- Receiver synchronization
- SC-FDMA demodulation
- Channel estimation
- Minimum Mean Squared Error (MMSE) equalization
- PUCCH 1 demodulation/decoding
- Measure missing or incorrect Hybrid Automatic Repeat Request (HARQ)-ACK

```
% Preallocate memory for probability of detection vector
PMISS = zeros(size(SNRIn));

for nSNR = 1:length(SNRIn)

    % Missed or incorrect ACK detection counter
    missCount = 0;

    % Noise configuration
    SNR = 10^(SNRIn(nSNR)/20);              % Convert dB to linear
    % The noise added before SC-FDMA demodulation will be amplified by the
    % IFFT. The amplification is the square root of the size of the IFFT.
    % To achieve the desired SNR after demodulation the noise power is
```

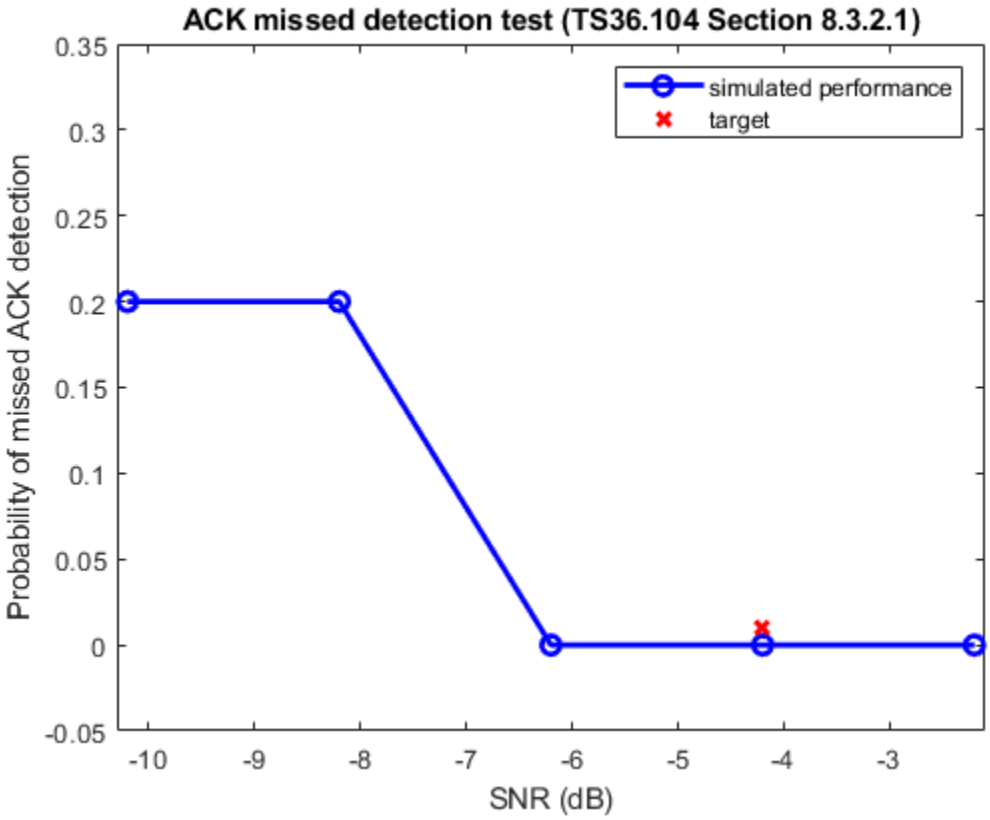
```
% normalized by this value. In addition, because real and imaginary
% parts of the noise are created separately before being combined into
% complex additive white Gaussian noise, the noise amplitude must be
% scaled by 1/sqrt(2*ue.NTxAnts) so the generated noise power is 1
N = 1/(SNR*sqrt(double(scfdmaInfo.Nfft)))/sqrt(2.0*ue.NTxAnts);
% Set the type of random number generator and its seed to the default
% value
rng('default')

% Loop for subframes
offsetused = 0;
for nsf = 1:NSubframes

    % Create resource grid
    ue.NSubframe = mod(nsf-1,10);
    reGrid = lteULResourceGrid(ue);

    % Create PUCCH 1 and its DRS
    pucch1Sym = ltePUCCH1(ue,pucch,ACK);
    pucch1DRSSym = ltePUCCH1DRS(ue,pucch);

    % Generate indices for PUCCH 1 and its DRS
    pucch1Indices = ltePUCCH1Indices(ue,pucch);
    pucch1DRSIndices = ltePUCCH1DRSIndices(ue,pucch);

    % Map PUCCH 1 and PUCCH 1 DRS to the resource grid
    reGrid(pucch1Indices) = pucch1Sym;
    reGrid(pucch1DRSIndices) = pucch1DRSSym;

    % SC-FDMA modulation
    txwave = lteSCFDMAModulate(ue,reGrid);

    % Channel state information: set the init time to the correct value
    % to guarantee continuity of the fading waveform
    channel.InitTime = (nsf-1)/1000;

    % Channel modeling
    % The additional 25 samples added to the end of the waveform are to
    % cover the range of delays expected from the channel modeling (a
    % combination of implementation delay and channel delay spread)
    rxwave = lteFadingChannel(channel,[txwave; zeros(25,ue.NTxAnts)]);

    % Add noise at receiver
    noise = N * complex(randn(size(rxwave)),randn(size(rxwave)));
    rxwave = rxwave + noise;

    % Receiver

    % Synchronization
    % An offset within the range of delays expected from the channel
    % modeling (a combination of implementation delay and channel
    % delay spread) indicates success
    offset = lteULFrameOffsetPUCCH1(ue,pucch,rxwave);
    if (offset < 25)
        offsetused = offset;
    end

    % SC-FDMA demodulation
```

```
            rxgrid = lteSCFDMADemodulate(ue,rxwave(1+offsetused:end,:));

            % Channel estimation
            [H,n0] = lteULChannelEstimatePUCCH1(ue,pucch,cec,rxgrid);

            % Extract REs corresponding to the PUCCH 1 from the given subframe
            % across all receive antennas and channel estimates
            [pucch1Rx,pucch1H] = lteExtractResources(pucch1Indices,rxgrid,H);

            % MMSE equalization
            eqgrid = lteULResourceGrid(ue);
            eqgrid(pucch1Indices) = lteEqualizeMMSE(pucch1Rx,pucch1H,n0);

            % PUCCH 1 demodulation/decoding
            rxACK = ltePUCCH1Decode(ue,pucch,length(ACK),eqgrid(pucch1Indices));

            % Detect missed (empty rxACK) or incorrect HARQ-ACK (compare
            % against transmitted ACK
            if (isempty(rxACK) || any(rxACK ~= ACK))
                missCount = missCount + 1;
            end

        end

        PMISS(nSNR) = (missCount/NSubframes);

    end
```

Results

The graph shows the simulation result for ACK missed detection test

```
plot(SNRIn,PMISS,'b-o','LineWidth',2,'MarkerSize',7);
hold on;
plot(-4.2,0.01,'rx','LineWidth',2,'MarkerSize',7);
xlabel('SNR (dB)');
ylabel('Probability of missed ACK detection');
title('ACK missed detection test (TS36.104 Section 8.3.2.1)');
axis([SNRIn(1)-0.1 SNRIn(end)+0.1 -0.05 .35]);
legend('simulated performance','target');
```



Selected Bibliography

1 3GPP TS 36.104 "Base Station (BS) radio transmission and reception"

PUCCH2 CQI BLER Conformance Test

This example shows how to measure the channel quality indicator (CQI) block error rate (BLER), which indicates the probability of incorrectly decoding CQI information sent using physical uplink control channel (PUCCH) format 2. The CQI BLER performance requirements are defined in TS 36.104 Section 8.3.3.1.

Introduction

This example uses a simulation length of 10 subframes. This value has been chosen to speed up the simulation. A larger value should be chosen to obtain more accurate results. The CQI BLER is calculated for a number of SNR points. The target defined in TS 36.104 Section 8.3.3.1 [1] for 1.4 MHz bandwidth (6 RBs) and a single transmit antenna is a CQI BLER of 1% (i.e. probability of erroneous block detection P = 0.01) at an SNR of -3.9 dB. The test is defined for 1 transmit antenna.

```
numSubframes = 10;                          % Number of subframes
SNRdB = [-9.9 -7.9 -5.9 -3.9 -1.9];         % SNR range
NTxAnts = 1;                                % Number of transmit antennas
```

UE Configuration

```
ue = struct;                                % UE config structure
ue.NULRB = 6;                               % 6 resource blocks
ue.CyclicPrefixUL = 'Normal';               % Normal cyclic prefix
ue.Hopping = 'Off';                         % No frequency hopping
ue.NCellID = 9;
ue.RNTI = 1;                                % Radio network temporary id
ue.NTxAnts = NTxAnts;
```

PUCCH 2 Configuration

```
% Empty hybrid ACK vector is used for Physical Uplink Control Channel
% (PUCCH) 2
ACK = [];

pucch = struct; % PUCCH config structure
% Vector of PUCCH resource indices, one per transmission antenna. This is
% the n2pucch parameter
pucch.ResourceIdx = 0:ue.NTxAnts-1;
% Set the size of resources allocated to PUCCH format 2
pucch.ResourceSize = 0;
% Number of cyclic shifts used for PUCCH format 1 in resource blocks with a
% mixture of formats 1 and 2. This is the N1cs parameter
pucch.CyclicShifts = 0;
```

Propagation Channel Configuration

Configure the channel model with the parameters specified in the tests described in TS 36.104 Section 8.3.3.1 [1].

```
channel = struct;                   % Channel config structure
channel.NRxAnts = 2;                % Number of receive antennas
channel.DelayProfile = 'ETU';       % Channel delay profile
channel.DopplerFreq = 70.0;         % Doppler frequency in Hz
channel.MIMOCorrelation = 'Low';    % Low MIMO correlation
channel.NTerms = 16;                % Oscillators used in fading model
```

```
channel.ModelType = 'GMEDS';        % Rayleigh fading model type
channel.Seed = 3;                   % Channel seed
channel.InitPhase = 'Random';       % Random initial phases
channel.NormalizePathGains = 'On';  % Normalize delay profile power
channel.NormalizeTxAnts = 'On';     % Normalize for transmit antennas

% SC-FDMA modulation information: required to get the sampling rate
info = lteSCFDMAInfo(ue);
channel.SamplingRate = info.SamplingRate;   % Channel sampling rate
```

Channel Estimator Configuration

The channel estimator is configured using a structure `cec`. Here cubic interpolation will be used with an averaging window of 12-by-1 Resource Elements (REs). This configures the channel estimator to use a special mode which ensures the ability to despread and orthogonalize the different overlapping PUCCH transmissions.

```
cec = struct;                     % Channel estimation config structure
cec.PilotAverage = 'UserDefined'; % Type of pilot averaging
cec.FreqWindow = 12;              % Frequency averaging window in REs (special mode)
cec.TimeWindow = 1;               % Time averaging window in REs (Special mode)
cec.InterpType = 'cubic';         % Cubic interpolation
```

Simulation Loop for Configured SNR Points

For each SNR point the loop below calculates the CQI BLER using information obtained from `NSubframes` consecutive subframes. The following operations are performed for each subframe and SNR values:

- Create an empty resource grid
- Generate and map PUCCH 2 and its Demodulation Reference Signal (DRS) to the resource grid
- SC-FDMA modulation
- Send the modulated signal through the channel
- Receiver synchronization
- SC-FDMA demodulation
- Channel estimation
- Minimum Mean Squared Error (MMSE) equalization
- PUCCH 2 demodulation/decoding
- Record decoding failures
- PUCCH 2 DRS decoding. This is not required as part of this test but is included to illustrate the steps involved

```
% Preallocate memory for vector of BLERs versus SNR
BLER = zeros(size(SNRdB));
for nSNR = 1:length(SNRdB)

    % Detection failures counter
    failCount = 0;

    % Noise configuration
    SNR = 10^(SNRdB(nSNR)/20);              % Convert dB to linear
    % The noise added before SC-FDMA demodulation will be amplified by the
    % IFFT. The amplification is the square root of the size of the IFFT.
```

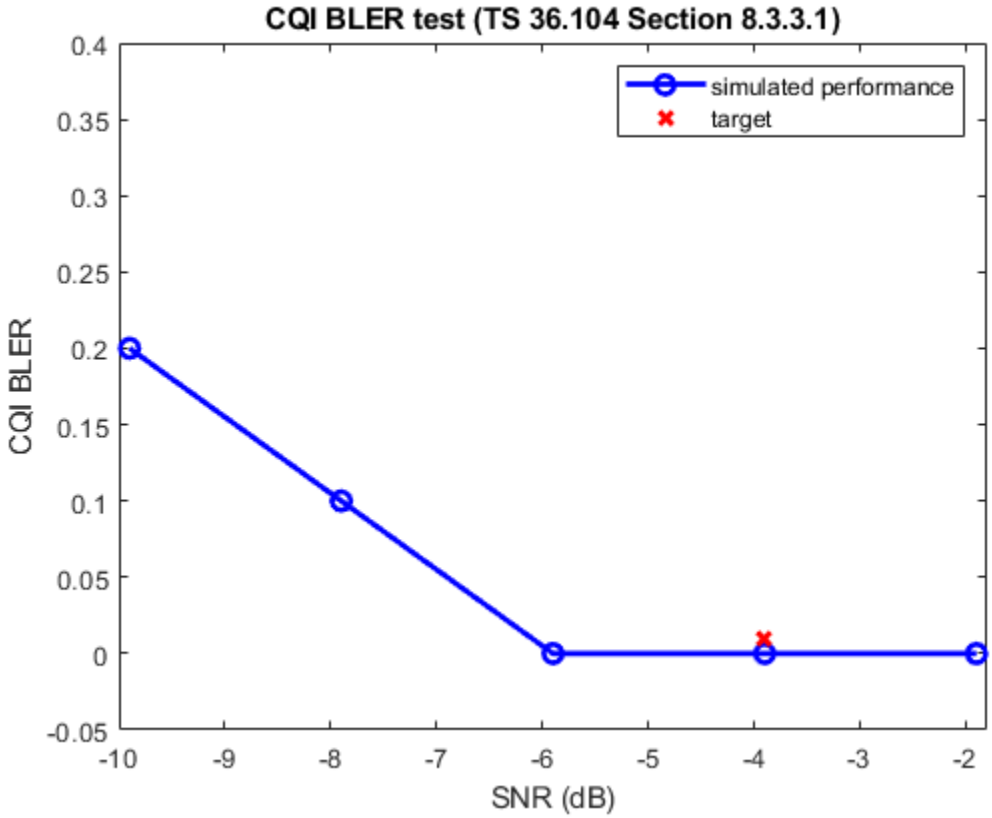
```
% To achieve the desired SNR after demodulation the noise power is
% normalized by this value. In addition, because real and imaginary
% parts of the noise are created separately before being combined into
% complex additive white Gaussian noise, the noise amplitude must be
% scaled by 1/sqrt(2*ue.NTxAnts) so the generated noise power is 1.
N = 1/(SNR*sqrt(double(info.Nfft)))/sqrt(2.0*ue.NTxAnts);
% Set the type of random number generator and its seed to the default
% value
rng('default');

% Loop for subframes
offsetused = 0;
for nsf = 1:numSubframes

    % Create resource grid
    ue.NSubframe = mod(nsf-1, 10);   % Subframe number
    reGrid = lteULResourceGrid(ue);  % Resource grid

    % Create PUCCH 2 and its DRS
    CQI = randi([0 1], 4, 1);              % Generate 4 CQI bits to send
    % Encode CQI bits to produce 20 bits
    coded = lteUCIEncode(CQI);
    pucch2Sym = ltePUCCH2(ue, pucch, coded);     % PUCCH 2 modulation
    pucch2DRSSym = ltePUCCH2DRS(ue, pucch, ACK); % PUCCH 2 DRS creation

    % Generate indices for PUCCH 2 and its DRS
    pucch2Indices = ltePUCCH2Indices(ue, pucch);
    pucch2DRSIndices = ltePUCCH2DRSIndices(ue, pucch);

    % Map PUCCH 2 and its DRS to the resource grid
    reGrid(pucch2Indices) = pucch2Sym;
    reGrid(pucch2DRSIndices) = pucch2DRSSym;

    % SC-FDMA modulation
    txwave = lteSCFDMAModulate(ue, reGrid);

    % Channel state information: set the init time to the correct value
    % to guarantee continuity of the fading waveform
    channel.InitTime = (nsf-1)/1000;

    % Channel modeling
    % The additional 25 samples added to the end of the waveform are to
    % cover the range of delays expected from the channel modeling (a
    % combination of implementation delay and channel delay spread)
    rxwave = lteFadingChannel(channel, [txwave;zeros(25, ue.NTxAnts)]);

    % Add noise at receiver
    noise = N*complex(randn(size(rxwave)), randn(size(rxwave)));
    rxwave = rxwave + noise;

    % Receiver

    % Synchronization
    % An offset within the range of delays expected from the channel
    % modeling (a combination of implementation delay and channel
    % delay spread) indicates success
    [offset, rxACK] = lteULFrameOffsetPUCCH2( ...
        ue, pucch, rxwave, length(ACK));
```

```
            if (offset<25)
                offsetused = offset;
            end

            % SC-FDMA demodulation
            rxgrid = lteSCFDMADemodulate(ue, rxwave(1+offsetused:end, :));

            % Channel estimation
            [H, n0] = lteULChannelEstimatePUCCH2(ue, pucch, cec, rxgrid, rxACK);

            % Extract REs corresponding to the PUCCH 2 from the given subframe
            % across all receive antennas and channel estimates
            [pucch2Rx, pucch2H] = lteExtractResources(pucch2Indices, rxgrid, H);

            % MMSE Equalization
            eqgrid = lteULResourceGrid(ue);
            eqgrid(pucch2Indices) = lteEqualizeMMSE(pucch2Rx, pucch2H, n0);

            % PUCCH 2 demodulation
            rxBits = ltePUCCH2Decode(ue, pucch, eqgrid(pucch2Indices));

            % PUCCH 2 decoding
            decoded = lteUCIDecode(rxBits, length(CQI));

            % Record any decoding failures
            if (sum(decoded~=CQI)~=0)
                failCount = failCount + 1;
            end

            % Perform PUCCH 2 DRS decoding. This is not required as part of
            % this test, but illustrates the steps involved.

            % Extract REs corresponding to the PUCCH 2 DRS from the given
            % subframe across all receive antennas and channel estimates
            [drsRx, drsH] = lteExtractResources(pucch2DRSIndices, rxgrid, H);

            % PUCCH 2 DRS Equalization
            eqgrid(pucch2DRSIndices) = lteEqualizeMMSE(drsRx, drsH, n0);

            % PUCCH 2 DRS decoding
            rxACK = ltePUCCH2DRSDecode( ...
                ue, pucch, length(ACK), eqgrid(pucch2DRSIndices));

        end

        % Probability of erroneous block detection
        BLER(nSNR) = (failCount/numSubframes);

    end
```

Results

```
plot(SNRdB, BLER, 'b-o', 'LineWidth', 2, 'MarkerSize', 7);
hold on;
plot(-3.9, 0.01, 'rx', 'LineWidth', 2, 'MarkerSize', 7);
xlabel('SNR (dB)');
ylabel('CQI BLER');
title('CQI BLER test (TS 36.104 Section 8.3.3.1)');
```

```
axis([SNRdB(1)-0.1 SNRdB(end)+0.1 -0.05 0.4]);
legend('simulated performance', 'target');
```



Selected Bibliography

1 3GPP TS 36.104 "Base Station (BS) radio transmission and reception"

PUCCH3 ACK Missed Detection Probability Conformance Test

This example measures the ACK missed detection probability using the LTE Toolbox™ under the single user Physical Uplink Control Channel (PUCCH3) conformance test conditions as defined in TS 36.104 Section 8.3.6.1.

Introduction

This example uses a simulation length of 10 subframes. This value has been chosen to speed up the simulation. A larger value should be chosen to obtain more accurate results. The target defined in TS36.104 Section 8.3.6.1 [1] for 10 MHz bandwidth (50 resource blocks) and a single transmit antenna is an Acknowledgment (ACK) missed detection probability not exceeding 1% at an SNR of -3.7 dB. The test is defined for 1 transmit antenna.

```
numSubframes = 10;                              % Number of subframes
SNRdB = [-9.7 -7.7 -5.7 -3.7 -1.7];             % SNR range
NTxAnts = 1;                                    % Number of transmit antennas
```

UE Configuration

```
ue = struct;                                    % UE config structure
ue.NULRB = 50;                                  % 50 resource blocks (10 MHz)
ue.CyclicPrefixUL = 'Normal';                   % Normal cyclic prefix
ue.NTxAnts = NTxAnts;
ue.NCellID = 9;
ue.RNTI = 1;                                    % Radio network temporary id
ue.Hopping = 'Off';                             % No frequency hopping
ue.Shortened = 0;                               % No SRS transmission
```

PUCCH 3 Configuration

```
% Vector of PUCCH resource indices, one per transmission antenna. This is
% the n3pucch parameter.
pucch = struct;
pucch.ResourceIdx = 0:ue.NTxAnts-1;
```

Propagation Channel Configuration

Configure the channel model with the parameters specified in the tests described in TS36.104 Section 8.3.6.1 [1].

```
channel = struct;                % Channel config structure
channel.NRxAnts = 2;             % Number of receive antennas
channel.DelayProfile = 'EPA';    % Channel delay profile
channel.DopplerFreq = 5.0;       % Doppler frequency in Hz
channel.MIMOCorrelation = 'Low'; % Low MIMO correlation
channel.NTerms = 16;             % Oscillators used in fading model
channel.ModelType = 'GMEDS';     % Rayleigh fading model type
channel.Seed = 4;                % Channel seed
channel.InitPhase = 'Random';    % Random initial phases
channel.NormalizePathGains = 'On';% Normalize delay profile power
channel.NormalizeTxAnts = 'On';  % Normalize for transmit antennas

% SC-FDMA modulation information: required to get the sampling rate
info = lteSCFDMAInfo(ue);
channel.SamplingRate = info.SamplingRate;
```

Channel Estimator Configuration

The channel estimator is configured using a structure `cec`. Here cubic interpolation will be used with an averaging window of 12-by-1 Resource Elements (REs). This configures the channel estimator to use a special mode which ensures the ability to despread and orthogonalize the different overlapping PUCCH transmissions.

```
cec = struct;                       % Channel estimation config structure
cec.PilotAverage = 'UserDefined'; % Type of pilot averaging
cec.FreqWindow = 12;                % Frequency averaging window in REs (special mode)
cec.TimeWindow = 1;                 % Time averaging window in REs (Special mode)
cec.InterpType = 'cubic';           % Cubic interpolation
```

Simulation Loop for Configured SNR Points

For each SNR point the loop below calculates the probability of successful ACK detection using information obtained from `NSubframes` consecutive subframes. The following operations are performed for each subframe and SNR values:

- Create an empty resource grid
- Generate and map PUCCH 3 and its Demodulation Reference Signal (DRS) to the resource grid
- Apply SC-FDMA modulation
- Send the modulated signal through the channel
- Receiver synchronization
- Apply SC-FDMA demodulation
- Estimate the channel
- Minimum Mean Squared Error (MMSE) equalization
- PUCCH 3 demodulation/decoding
- Record decoding failures

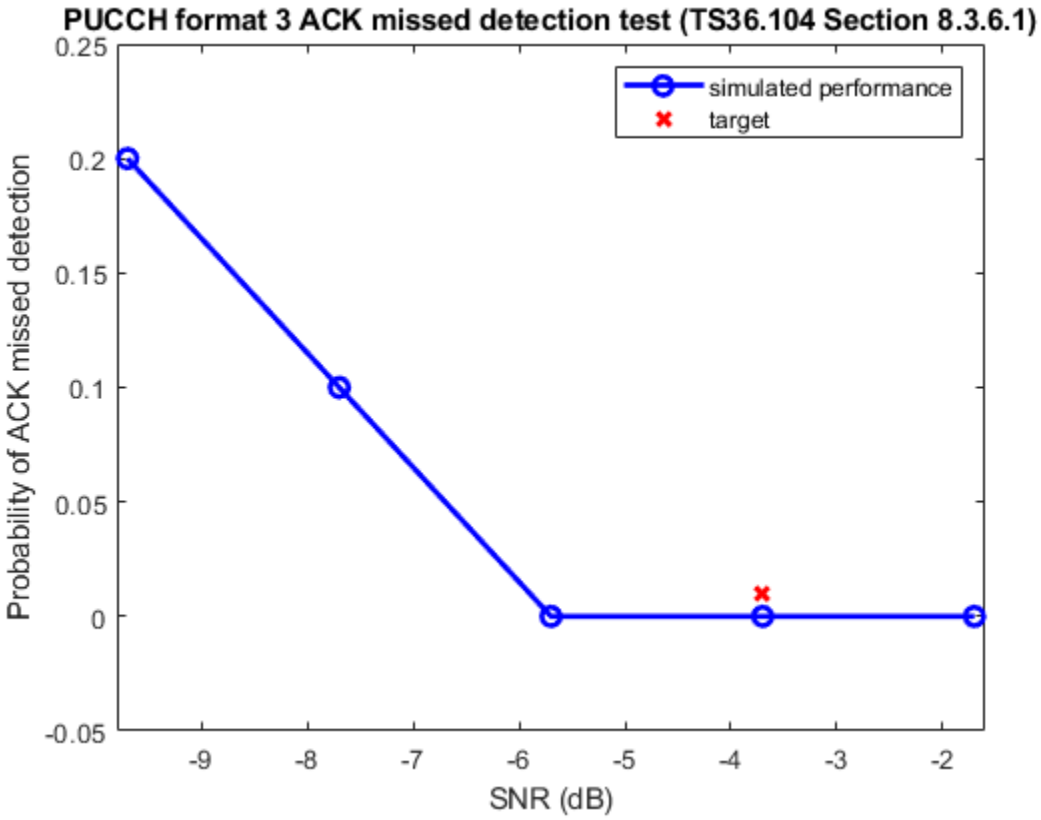
```
% Preallocate memory for missed detection probability vector
PMISS = zeros(size(SNRdB));
for nSNR = 1:length(SNRdB)

    % Detection failures counter
    missCount = 0;
    falseCount = 0;

    % Noise configuration
    SNR = 10^(SNRdB(nSNR)/20);              % Convert dB to linear
    % The noise added before SC-FDMA demodulation will be amplified by the
    % IFFT. The amplification is the square root of the size of the IFFT.
    % To achieve the desired SNR after demodulation the noise power is
    % normalized by this value. In addition, because real and imaginary
    % parts of the noise are created separately before being combined into
    % complex additive white Gaussian noise, the noise amplitude must be
    % scaled by 1/sqrt(2*ue.NTxAnts) so the generated noise power is 1.
    N = 1/(SNR*sqrt(double(info.Nfft)))/sqrt(2.0*ue.NTxAnts);
    % Set the type of random number generator and its seed to the default
    % value
    rng('default');

    % Loop for subframes
```

```
offsetused = 0;
for nsf = 1:numSubframes

    % Create resource grid
    ue.NSubframe = mod(nsf-1,10);       % Subframe number
    reGrid = lteULResourceGrid(ue);       % Resource grid

    % Generate PUCCH 3 and its DRS
    N_ACK = 4;                          % 4 ACK bits
    ACK = randi([0 1], N_ACK, 1);       % Generate N_ACK random bits
    coded = lteUCI3Encode(ACK);         % PUCCH 3 coding
    pucch3Sym = ltePUCCH3(ue, pucch, coded);     % PUCCH 3 modulation
    pucch3DRSSym = ltePUCCH3DRS(ue, pucch, ACK); % PUCCH 3 DRS creation

    % Generate indices for PUCCH 3 and its DRS
    pucch3Indices = ltePUCCH3Indices(ue, pucch);
    pucch3DRSIndices = ltePUCCH3DRSIndices(ue, pucch);

    % Map PUCCH 3 and its DRS to the resource grid
    reGrid(pucch3Indices) = pucch3Sym;
    reGrid(pucch3DRSIndices) = pucch3DRSSym;

    % SC-FDMA modulation
    txwave = lteSCFDMAModulate(ue, reGrid);

    % Channel state information: set the init time to the correct value
    % to guarantee continuity of the fading waveform after each
    % subframe
    channel.InitTime = (nsf-1)/1000;

    % Channel modeling
    % The additional 25 samples added to the end of the waveform
    % are to cover the range of delays expected from the channel
    % modeling (a combination of implementation delay and
    % channel delay spread)
    rxwave = lteFadingChannel(channel, [txwave; zeros(25, ue.NTxAnts)]);

    % Add noise at receiver
    noise = N*complex(randn(size(rxwave)), randn(size(rxwave)));
    rxwave = rxwave + noise;

    % Receiver

    % Synchronization
    % An offset within the range of delays expected from the channel
    % modeling (a combination of implementation delay and channel
    % delay spread) indicates success
    offset = lteULFrameOffsetPUCCH3(ue, pucch, rxwave);
    if (offset<25)
        offsetused = offset;
    end

    % SC-FDMA demodulation
    rxgrid = lteSCFDMADemodulate(ue, rxwave(1+offsetused:end, :));

    % Channel estimation
    [H, n0] = lteULChannelEstimatePUCCH3(ue, pucch, cec, rxgrid);
```

```
            % Extract REs corresponding to the PUCCH 3 from the given subframe
            % across all receive antennas and channel estimates
            [pucch3Rx, pucch3H] = lteExtractResources(pucch3Indices, rxgrid, H);

            % PUCCH 3 MMSE Equalization
            eqgrid = lteULResourceGrid(ue);
            eqgrid(pucch3Indices) = lteEqualizeMMSE(pucch3Rx, pucch3H, n0);

            % PUCCH 3 demodulation
            rxBits = ltePUCCH3Decode(ue, pucch, eqgrid(pucch3Indices));

            % PUCCH 3 decoding
            rxACK = lteUCI3Decode(rxBits, N_ACK);

            % Detect missed (empty rxACK) or incorrect Hybrid Automatic Repeat
            % Request (HARQ)-ACK
            % (compare against transmitted ACK)

            if (isempty(rxACK) || any(rxACK ~= ACK))
                missCount = missCount + 1;
            end

        end

        PMISS(nSNR) = missCount/numSubframes;

    end
```

Results

```
plot(SNRdB, PMISS, 'b-o', 'MarkerSize', 7, 'LineWidth', 2);
hold on;
plot(-3.7, 0.01, 'rx', 'MarkerSize', 7, 'LineWidth', 2);
xlabel('SNR (dB)');
ylabel('Probability of ACK missed detection');
title(['PUCCH format 3 ACK missed detection test' ...
       ' (TS36.104 Section 8.3.6.1)']);
axis([SNRdB(1)-0.1 SNRdB(end)+0.1 -0.05 0.25]);
legend('simulated performance', 'target');
```



Selected Bibliography

1 3GPP TS 36.104 "Base Station (BS) radio transmission and reception"

PRACH Detection Conformance Test

This example shows how the LTE Toolbox™ can be used to model a TS36.104 "PRACH Detection Requirements" conformance test. The probability of correct detection of the Physical Random Access Channel (PRACH) preamble is measured when the preamble signal is present.

Introduction

The Random Access Channel (RACH) is an uplink transmission used by the User Equipment (UE) to initiate synchronization with the eNodeB. TS36.104, Section 8.3.4.1 [1] defines the probability of Physical Random Access Channel (PRACH) detection must be greater than or equal to 99% at an SNR of -8.0 dB. There are several detection error cases:

- Detecting an incorrect preamble
- Not detecting a preamble
- Detecting the correct preamble but with the wrong timing estimation

For channel propagation conditions ETU70, a timing estimation error occurs if the estimation error of the timing offset of the strongest path is larger than 2.08 us. The strongest path has a delay of 310 ns for ETU70.

In this example, a PRACH waveform is configured and passed through an appropriate channel. At the receiver PRACH detection is performed and the PRACH detection probability is calculated. The example is executed for the parameters defined in TS36.104, Section 8.4.1 [1] and TS36.141, Table A.6-1 [2]; i.e. normal mode, 2 receive antennas, ETU70 channel, normal cyclic prefix, burst format 0, SNR -8.0dB.

Simulation Configuration

The test is carried out over 10 subframes at a number of SNRs. `foffset` is the frequency offset which will be modeled between transmitter and receiver and is specified for the test in TS36.104, Table 8.4.2.1-1 [1]. A large number of `numSubframes` should be used to produce meaningful results. `SNRdB` can be an array of values or a scalar.

```
numSubframes = 10;  % Number of subframes to simulate at each SNR
SNRdB = [-14.0 -12.0 -10.0 -8.0 -6.0];  % SNR points to simulate
foffset = 270.0;                        % Frequency offset in Hertz
```

UE Configuration

User Equipment (UE) settings are specified in the structure `ue`.

```
ue.NULRB = 6;                   % 6 Resource Blocks
ue.DuplexMode = 'FDD';          % Frequency Division Duplexing (FDD)
ue.CyclicPrefixUL = 'Normal';   % Normal cyclic prefix length
ue.NTxAnts = 1;                 % Number of transmission antennas
```

PRACH Configuration

The PRACH configuration is defined in the structure `prach`. This configuration is used to create a second structure `info` by calling `ltePRACHInfo`. This provides information about the PRACH generated given a specified configuration. Some of this information will be used later in the example.

```
prach.Format = 0;          % PRACH format: TS36.104, Table 8.4.2.1-1
prach.SeqIdx = 22;         % Logical sequence index: TS36.141, Table A.6-1
```

```
prach.CyclicShiftIdx = 1;  % Cyclic shift index: TS36.141, Table A.6-1
prach.HighSpeed = 0;       % Normal mode: TS36.104, Table 8.4.2.1-1
prach.FreqOffset = 0;      % Default frequency location
prach.PreambleIdx = 32;    % Preamble index: TS36.141, Table A.6-1

info = ltePRACHInfo(ue, prach);  % PRACH information
```

Propagation Channel Configuration

Configure the propagation channel model using a structure `chcfg` as per TS36.104, Table 8.4.2.1-1 [1].

```
chcfg.NRxAnts = 2;                       % Number of receive antenna
chcfg.DelayProfile = 'ETU';              % Delay profile
chcfg.DopplerFreq = 70.0;                % Doppler frequency
chcfg.MIMOCorrelation = 'Low';           % MIMO correlation
chcfg.Seed = 1;                          % Channel seed
chcfg.NTerms = 16;                       % Oscillators used in fading model
chcfg.ModelType = 'GMEDS';               % Rayleigh fading model type
chcfg.InitPhase = 'Random';              % Random initial phases
chcfg.NormalizePathGains = 'On';         % Normalize delay profile power
chcfg.NormalizeTxAnts = 'On';            % Normalize for transmit antennas
chcfg.SamplingRate = info.SamplingRate;  % Sampling rate
```

Loop for SNR Values

A loop is used to run the simulation for a set of SNR points, given by the vector `SNRdB`. The SNR vector configured here is a range of SNR points including a point at -8.0 dB, the SNR at which the test requirement for PRACH detection rate (99%) is to be achieved.

`ltePRACH` generates an output signal normalized to the same transmit power as for an uplink data transmissions within the LTE Toolbox. Therefore the same normalization must take place on the noise added to the PRACH. The noise added before SC-FDMA demodulation will be amplified by the IFFT. The amplification is the square root of the size of the IFFT (N_{FFT}), thus to ensure the power of the noise added is normalized after demodulation, to achieve the desired SNR the desired noise power is divided by N_{FFT}. In addition, as real and imaginary parts of the noise are created separately before being combined into complex additive white Gaussian noise, the noise amplitude must be scaled by $1/\sqrt{2}$ so the generated noise power is 1.

At each SNR test point the probability detection is calculated on a subframe by subframe basis using the following steps:

- *PRACH Transmission:* A PRACH waveform is generated using `ltePRACH` with a specified timing offset. As per TS 36.141 Section 8.4.1.4.2 [2], the preambles are sent with certain timing offsets defined in TS36.141, Figure 8.4.1.4.2-2 [2]. A timing offset base value is set to 50% of the number of cyclic shifts for PRACH generation. This offset is increased for each preamble, adding a step value of 0.1 us, until the end of the tested range, which is 0.9 us. This pattern then repeats.
- *Noisy Channel Modeling:* The waveform is passed through a fading channel and additive white Gaussian noise added. An additional 25 samples are added to the end of the waveform to cover the range of delays expected from the channel modeling (a combination of implementation delay and channel delay spread). This implementation delay is then removed to ensure the implementation delay is not interpreted as an actual timing offset in the preamble detector.
- *Application of Frequency Offset:* The frequency offset defined by the standard is also applied at this stage by performing a simple FM modulation of the received waveform.

- *PRACH Detection:* Perform PRACH detection using `ltePRACHDetect` for all cell preamble indices (0-63). The detected PRACH index and offset is returned and used to determine where a detection was successful according to the constraints discussed in the Introduction.

```
% Initialize the random number generator stream
rng('default');

% Initialize variables storing probability of detection at each SNR
pDetection = zeros(size(SNRdB));

for nSNR = 1:length(SNRdB)

    % Scale noise to ensure the desired SNR after SC-FDMA demodulation
    ulinfo = lteSCFDMAInfo(ue);
    SNR = 10^(SNRdB(nSNR)/20);
    N = 1/(SNR*sqrt(double(ulinfo.Nfft)))/sqrt(2.0);

    % Detected preamble count
    detectedCount = 0;

    % Loop for each subframe
    for nsf = 1:numSubframes

        % PRACH transmission
        ue.NSubframe = mod(nsf-1, 10);
        ue.NFrame = fix((nsf-1)/10);

        % Set PRACH timing offset in us as per TS36.141, Figure 8.4.1.4.2-2
        prach.TimingOffset = info.BaseOffset + ue.NSubframe/10.0;

        % Generate transmit wave
        txwave = ltePRACH(ue, prach);

        % Channel modeling
        chcfg.InitTime = (nsf-1)/1000;
        [rxwave, fadinginfo] = lteFadingChannel(chcfg, ...
                                [txwave; zeros(25, 1)]);

        % Add noise
        noise = N*complex(randn(size(rxwave)), randn(size(rxwave)));
        rxwave = rxwave + noise;

        % Remove the implementation delay of the channel modeling
        rxwave = rxwave((fadinginfo.ChannelFilterDelay + 1):end, :);

        % Apply frequency offset
        t = ((0:size(rxwave, 1)-1)/chcfg.SamplingRate).';
        rxwave = rxwave .* repmat(exp(1i*2*pi*foffset*t), ...
            1, size(rxwave, 2));

        % PRACH detection for all cell preamble indices
        [detected, offsets] = ltePRACHDetect(ue, prach, rxwave, (0:63).');

        % Test for preamble detection
        if (length(detected)==1)

            % Test for correct preamble detection
```

```
            if (detected==prach.PreambleIdx)

                % Calculate timing estimation error. The true offset is
                % PRACH offset plus channel delay
                trueOffset = prach.TimingOffset/1e6 + 310e-9;
                measuredOffset = offsets(1)/chcfg.SamplingRate;
                timingerror = abs(measuredOffset-trueOffset);

                % Test for acceptable timing error
                if (timingerror<=2.08e-6)
                    detectedCount = detectedCount + 1; % Detected preamble
                else
                    disp('Timing error');
                end
            else
                disp('Detected incorrect preamble');
            end
        else
            disp('Detected multiple or zero preambles');
        end

    end % of subframe loop

    % Compute final detection probability for this SNR
    pDetection(nSNR) = detectedCount/numSubframes;

end % of SNR loop
```

```
Detected multiple or zero preambles
Detected multiple or zero preambles
```

Analysis

At the end of the SNR loop, the calculated detection probabilities for each SNR value are plotted against the target probability.

```
targetSNR = -8.0;
PRACHDetectionResults(SNRdB, numSubframes, pDetection, targetSNR);
```

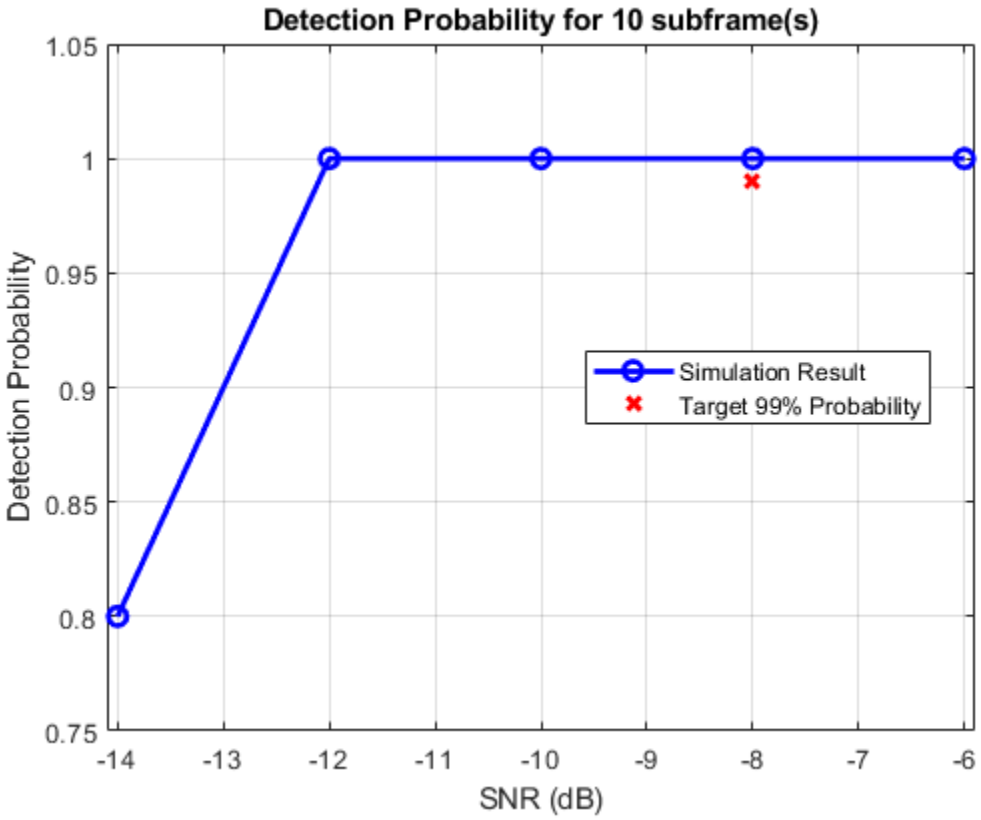
References

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)*.

Local Functions

```
function PRACHDetectionResults(SNRdB,numSubframes,pDetection,targetSNR)
    % Plot detection probability
    figure('NumberTitle','off','Name','PRACH Detection Probability');
    plot(SNRdB,pDetection,'b-o','LineWidth',2,'MarkerSize',7);
    title(['Detection Probability for ',num2str(numSubframes),' subframe(s)']);
    xlabel('SNR (dB)'); ylabel('Detection Probability');
    grid on; hold on;
```

```
        % Plot target probability
        plot(targetSNR,0.99,'rx','LineWidth',2,'MarkerSize',7);
        legend('Simulation Result', 'Target 99% Probability','Location','best');
        minP = 0;
        if(~isnan(min(pDetection)))
            minP = min([pDetection(:); 0.99]);
        end
        axis([SNRdB(1)-0.1, SNRdB(end)+0.1, minP-0.05, 1.05]);
    end
```



PRACH False Alarm Probability Conformance Test

This example shows how the LTE Toolbox™ can be used to model a TS36.104 Physical Random Access Channel (PRACH) false alarm probability conformance test. In this case the probability of erroneous detection of a PRACH preamble is to be measured when the input to the PRACH detector is only noise.

Introduction

Measuring the probability of erroneous detection is defined in TS36.104, Section 8.4.1 [1]. The target defined for any bandwidth, all frame structures and for any number of receive antennas is a false alarm probability less than or equal to 0.1%.

The input to the PRACH detector is made up exclusively of a Gaussian noise signal. Detection is attempted with all possible cell preamble indices, and if a preamble is detected within the noise, a false detection is accumulated.

UE Configuration

A configuration structure for the User Equipment (UE) is created and set to use 6 resource blocks in Frequency Division Duplexing (FDD).

```
ue.NULRB = 6;                   % Number of resource blocks
ue.DuplexMode = 'FDD';          % FDD duplexing mode
```

PRACH Configuration

Setup the PRACH configuration structure. The standard documents do not specify any values since no PRACH is transmitted.

```
prach.Format = 0;               % Preamble format
prach.SeqIdx = 2;               % Logical root sequence index
prach.CyclicShiftIdx = 1;       % Cyclic shift configuration index
prach.HighSpeed = 0;            % High speed flag
prach.FreqOffset = 0;           % Use default frequency resource index
prach.PreambleIdx = [];         % Empty since no preamble is transmitted
```

Establish PRACH Generator Output Length for This Configuration

`ltePRACHInfo` returns PRACH dimensionality information given UE-specific settings and PRACH channel transmission configuration. From this information the PRACH output length can be calculated using the PRACH duration in subframes, the duration of one subframe in seconds (1ms) and the PRACH modulator sampling rate.

```
info = ltePRACHInfo(ue, prach);
nSamples = info.SamplingRate*info.TotSubframes*0.001;
```

Loop for Detection in Each Subframe

Initialize the false detection count and enter a loop to process multiple detection trials. For each detection trial a Gaussian noise signal (of length `nSamples`) is created and detection is performed on it for every possible cell preamble index using `ltePRACHDetect`. In the ideal case, no preambles should have been detected. In the case where a preamble has been detected, a false alarm for this trial is recorded.

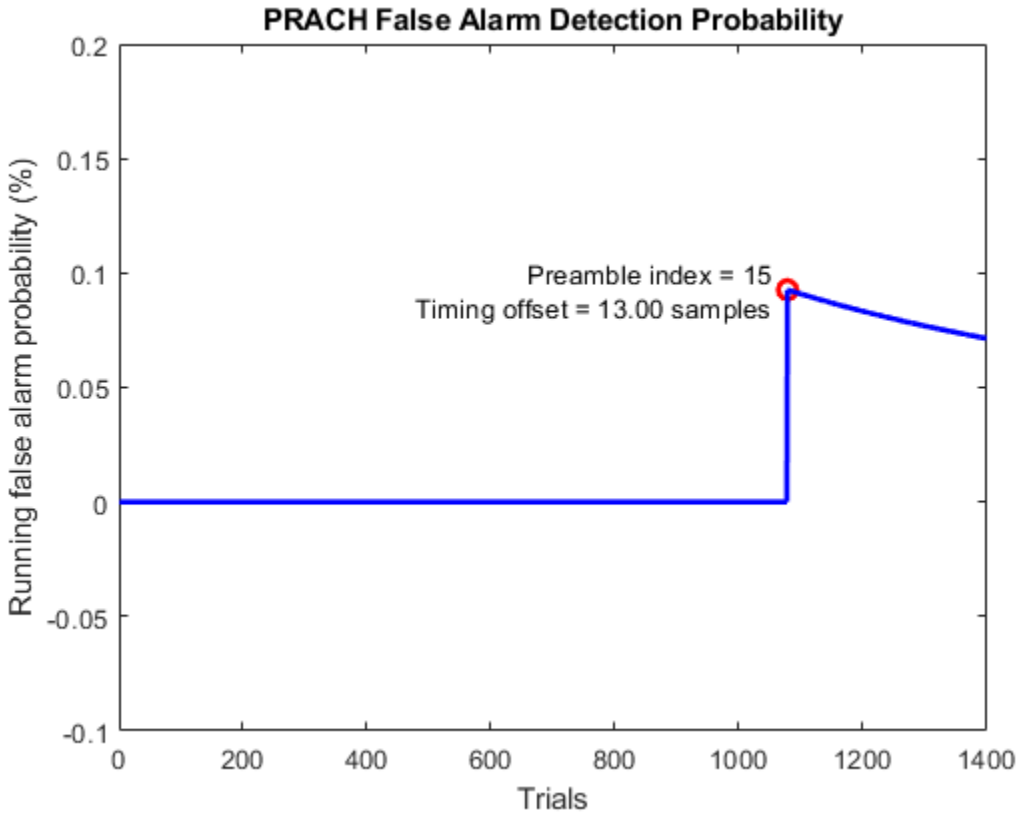
```
numTrials = 1400;
falseCount = 0;         % Initialize false detection counter
rng('default');         % Random number generator to default state

runningP=zeros(1, numTrials);
runningDetected(numTrials)=struct('Detected',[],'Offset',[]);
for nt = 1:numTrials

    % Create noise
    noise = complex(randn(nSamples, 1), randn(nSamples, 1));

    % Attempt detection for all cell preamble indices (0...63)
    [detected,offset] = ltePRACHDetect(ue, prach, noise, 0:63);

    % Record false alarm
    if (~isempty(detected))
        falseCount = falseCount+1;
    end

    % Calculate running false alarm probability
    runningP(nt) = falseCount/nt*100;

    % Store information about false alarm (if applicable)
    if (~isempty(detected))
        runningDetected(nt).Detected = detected;
        runningDetected(nt).Offset = offset;
    end

end
```

Compute Final False Alarm Probability

Compute the final false alarm probability and plot the running false alarm probability across the simulation.

```
P = falseCount / numTrials;
fprintf('\nFalse alarm probability = %0.4f%%\n',P*100);
hPlotFalseAlarmProbability(runningP,runningDetected);
```

```
False alarm probability = 0.0714%
```



Selected Bibliography

1 3GPP TS 36.104 "Base Station (BS) radio transmission and reception"

LTE Downlink Test Model (E-TM) Waveform Generation

This example shows how to generate a test model using LTE Toolbox™.

Overview

The LTE specifications define conformance test models for transmitter tests. These include transmit signal quality, output power dynamics, Error Vector Magnitude (EVM) for various modulation schemes, Base Station (BS) output power, Reference Symbol (RS) absolute accuracy, etc. This example demonstrates how these different test model waveforms can be generated using LTE Toolbox functions.

The following general parameters are used by all E-UTRA test models as defined in TS 36.141, Section 6.1.2 [1] :

- Single antenna port, 1 code word, 1 layer without any precoding
- Duration is 10 subframes (10 ms)
- Normal cyclic prefix
- Virtual resource blocks of localized type
- User Equipment (UE)-specific reference signals are not used

The following physical channels and signals will be generated:

- Reference Signals (CellRS)
- Primary Synchronization Signal (PSS)
- Secondary Synchronization Signal (SSS)
- Physical Broadcast Channel (PBCH)
- Physical Control Format Indicator Channel (PCFICH)
- Physical Hybrid-ARQ Indicator Channel (PHICH)
- Physical Downlink Control Channel (PDCCH)
- Physical Downlink Shared Channel (PDSCH)

Test models are selected according to required test cases. In our example, the considered test model, E-TM1.1, should be used for tests on:

- BS output power
- Unwanted emissions - Occupied bandwidth, Adjacent Channel Leakage power Ratio (ACLR), operating band unwanted emissions, transmitter spurious emissions
- Transmitter intermodulation
- Reference signals absolute accuracy

Test Model Selection

A number of test models are defined in TS 36.141 Section 6.1 [1]. This example will generate test model 1.1 as shown below.

```
tm = '1.1';     % Test model number
```

Allowed test model values in the toolbox are ('1.1', '1.2', '2', '2a', '3.1', '3.1a', '3.2', '3.3').

Bandwidth Selection

The test model generation function in the toolbox requires the bandwidth to be specified as shown below:

```
bw = '1.4MHz';      % Bandwidth
```

Test Model Generation

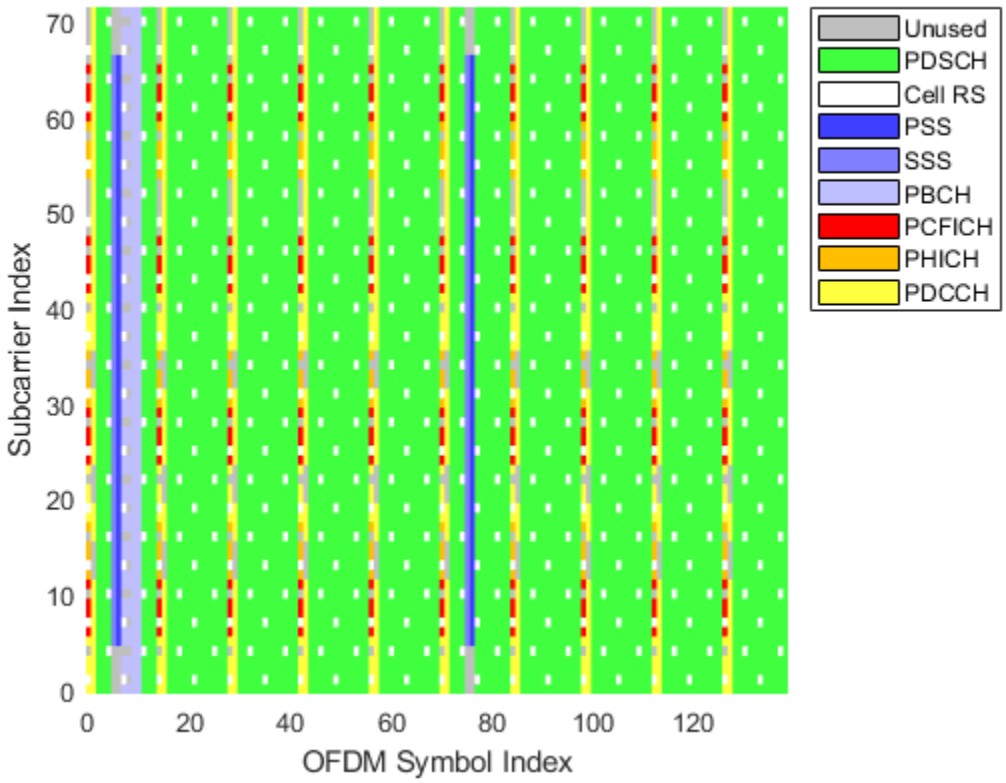
The channel model number and the bandwidth determine the physical channel and signal parameters as specified in TS 36.141. The generated waveform `timeDomainSig` is a time domain signal after performing OFDM modulation, cyclic prefix insertion and windowing. `txGrid` is a 2-dimensional array representing the resource grid spanning 10 subframes.

```
[timeDomainSig, txGrid, txInfo] = lteTestModelTool(tm,bw);
```

Plot Transmitted Resource Grid

Plot the resource grid `txGrid`, with a legend describing which resource elements are allocated to which physical channels and signals.

```
hPlotDLResourceGrid(txInfo,txGrid);
```

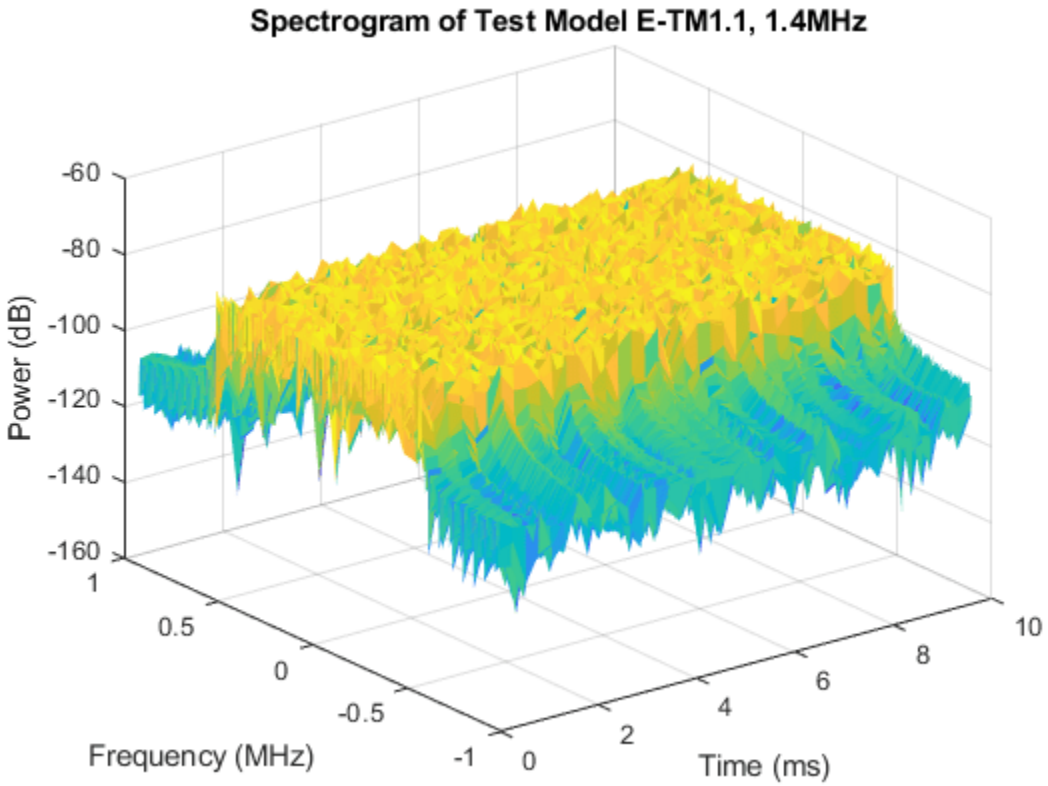


Plot Spectrogram

Plot a spectrogram of the time domain signal.

```
% Compute spectrogram
[y,f,t,p] = spectrogram(timeDomainSig, 512, 0, 512, txInfo.SamplingRate);
```

```
% Re-arrange frequency axis and spectrogram to put zero frequency in the
% middle of the axis i.e. represent as a complex baseband waveform
f = (f-txInfo.SamplingRate/2)/1e6;
p = fftshift(10*log10(abs(p)));

% Plot spectrogram
figure;
surf(t*1000,f,p,'EdgeColor','none');
xlabel('Time (ms)');
ylabel('Frequency (MHz)');
zlabel('Power (dB)');
title(sprintf('Spectrogram of Test Model E-TM%s, %s',tm, bw));
```



Further Exploration

For over-the-air transmission and analysis of the test model waveform, refer to the following example: “Waveform Generation and Transmission Using LTE Toolbox with Test and Measurement Equipment” on page 2-632.

Appendix

This example uses the following helper function:

- hPlotDLResourceGrid.m

Selected Bibliography

1 3GPP TS 36.141 "Base Station (BS) conformance testing"

LTE Uplink EVM and In-Band Emissions Measurements

This example shows how the LTE Toolbox™ can be used to perform Error Vector Magnitude (EVM) and in-band emissions measurements on an uplink signal as per TS 36.101 Annex F [1].

Introduction

TS 36.101 Annex F [1] defines two measurements for an uplink transmission, EVM and in-band emissions:

- EVM is used as a measure of the received signal constellation error. EVM is defined as the difference between the ideal received waveform and the measured waveform for allocated resource blocks. The average EVM is measured at two locations in time (low and high), where the low and high locations correspond to the alignment of the FFT window within the start and end of the cyclic prefix. LTE Toolbox requires the low and high locations to be specified as a fraction of the cyclic prefix length.

A User Equipment (UE) transmission is created using a Reference Measurement Channel (RMC) and random Physical Uplink Shared Channel (PUSCH) data and impaired by introducing additive noise to model transmitter EVM, and frequency and IQ offsets. The transmitted waveform is synchronized before the function computing EVM and in-band emissions.

Transmitter

To generate the RMC the function `lteRMCUL` creates a configuration structure for given UE settings specific to a given Fixed Reference Channel (FRC). This structure is used by `lteRMCULTool` to generate a UE transmission with random PUSCH data.

```
% Set the seeds of random number generators used to 0
rng(0);

% UE Configuration, TS36.101 FRC
frc = lteRMCUL('A3-1');
frc.PUSCH.RVSeq = 0;    % Redundancy version
frc.TotSubframes = 15;  % Total number of subframes to generate

% Create UE transmission with random PUSCH data
txWaveform = lteRMCULTool(frc,randi([0 1], frc.PUSCH.TrBlkSizes(1), 1));
```

Impairment Modeling

Impairments are added to the transmission to simulate a device under test:

- 1.2% transmit EVM modeled with additive noise.
- 33 Hz frequency offset.
- 0.01 - 0.005j IQ offset

```
% Model EVM with additive noise
scfdmaInfo = lteSCFDMAInfo(frc);
txEVMpc = 1.2; % Desired transmit EVM in percent
evmModel = txEVMpc/(100*sqrt(double(scfdmaInfo.Nfft)))* ...
    complex(randn(size(txWaveform)),randn(size(txWaveform)))/sqrt(2);
rxWaveform = txWaveform+evmModel;
```

```
% Add frequency offset impairment to received waveform
foffset = 33.0;         % Frequency offset in Hertz
t = (0:length(rxWaveform)-1).'/scfdmaInfo.SamplingRate;
rxWaveform = rxWaveform.* exp(1i*2*pi*foffset*t);

% Add IQ offset
iqoffset = complex(0.01, -0.005);
rxWaveform = rxWaveform+iqoffset;
```

Receiver

The received waveform is synchronized to allow for the measurements to be performed

```
% Apply frequency estimation and correction for the purposes of performing
% timing synchronization
foffset_est = lteFrequencyOffset(frc, rxWaveform);
rxWaveformFreqCorrected = ...
    lteFrequencyCorrect(frc, rxWaveform, foffset_est);

% Synchronize to received waveform
offset = lteULFrameOffset(frc, frc.PUSCH, rxWaveformFreqCorrected);
rxWaveform  = rxWaveform(1+offset:end,:);
```

Perform Measurements

The PUSCH EVM, PUSCH DRS EVM and in-band emissions are calculated by calling `hPUSCHEVM`.

The EVM results and absolute in-band emissions for each Δ_{RB} and slot number are displayed. Δ_{RB} is the starting frequency offset between the allocated resource block (RB) and the measured non-allocated RB, i.e. $\Delta_{RB} = 1$ for the first adjacent RB outside of the allocated bandwidth. A number of plots are also produced:

- EVM versus OFDM symbol
- EVM versus subcarrier
- EVM versus resource block
- EVM versus OFDM symbol and subcarrier (i.e. the EVM resource grid)

Note that the EVM measurement displayed at the command window is only calculated across allocated PUSCH resource blocks, in accordance with the LTE standard. The EVM plots are shown across all resource blocks (allocated or unallocated), allowing the in-band emissions to be viewed. In unallocated resource blocks, the EVM is calculated assuming that the received resource elements have an expected value of zero.

The EVM of each E-UTRA carrier for QPSK/BPSK, 16QAM, 64QAM and 256QAM modulation should not exceed the EVM level of 17.5%, 12.5%, 8% and 3.5% respectively as per TS 36.101 Table 6.5.2.1.1-1 [1].

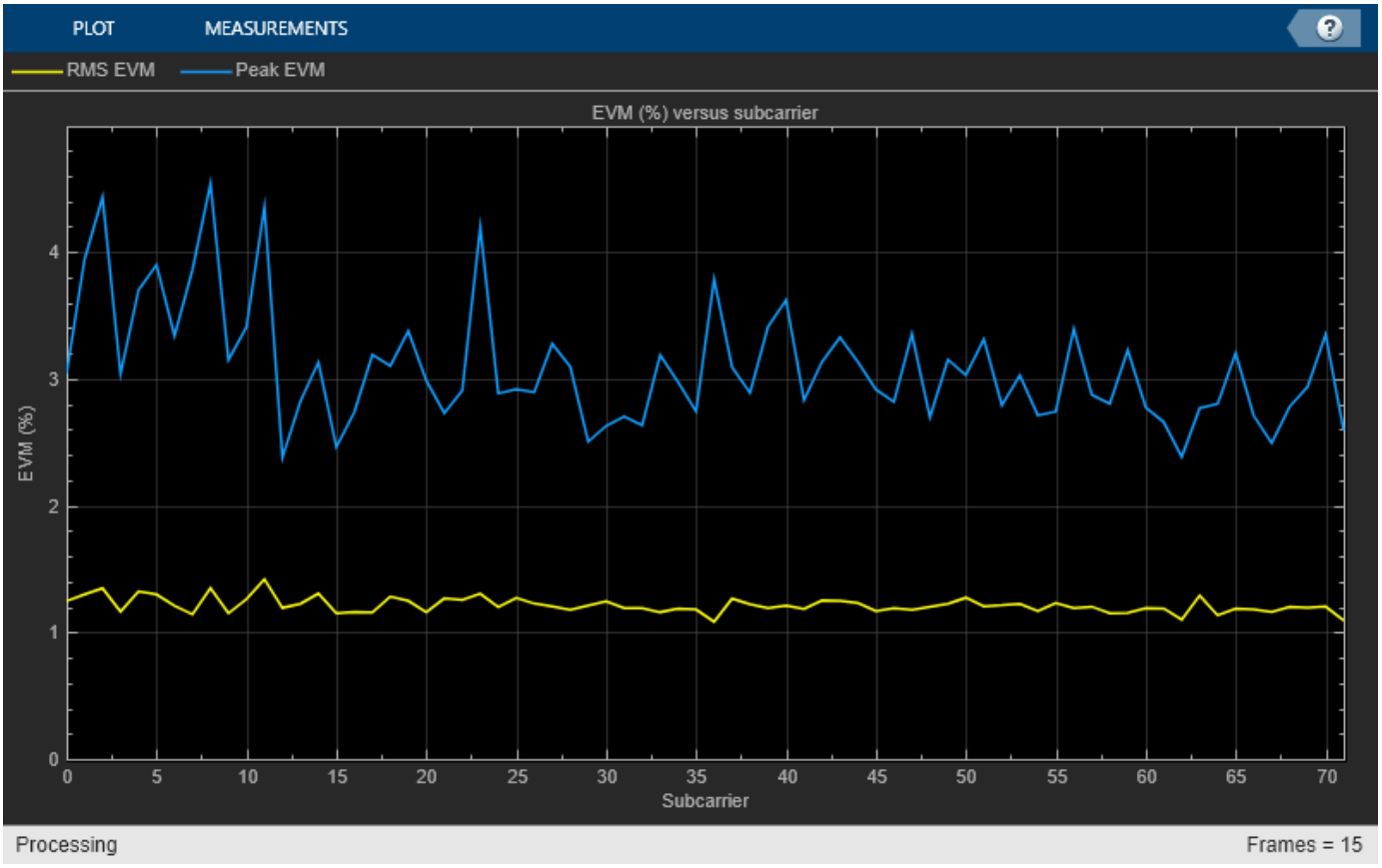
```
% Compute EVM and in-band emissions
[evmpusch, evmdrs, emissions, plots] = hPUSCHEVM(frc, rxWaveform);

% Plot the absolute in-band emissions
if (~isempty(emissions.DeltaRB))
    hPUSCHEVMEmissionsPlot(emissions);
end
```

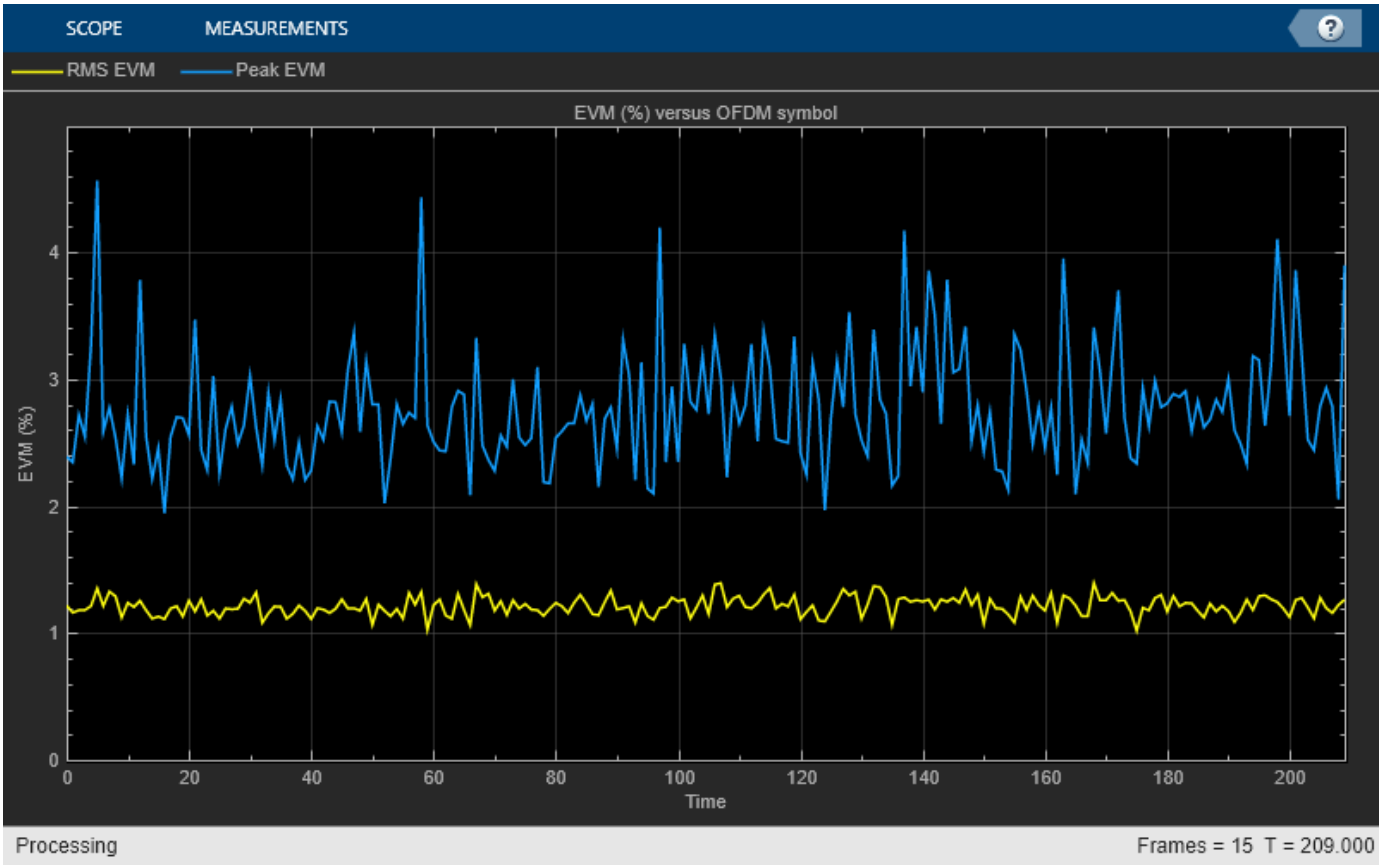
```
Low edge PUSCH EVM, slot 0: 1.231%
Low edge PUSCH EVM, slot 1: 1.231%
```

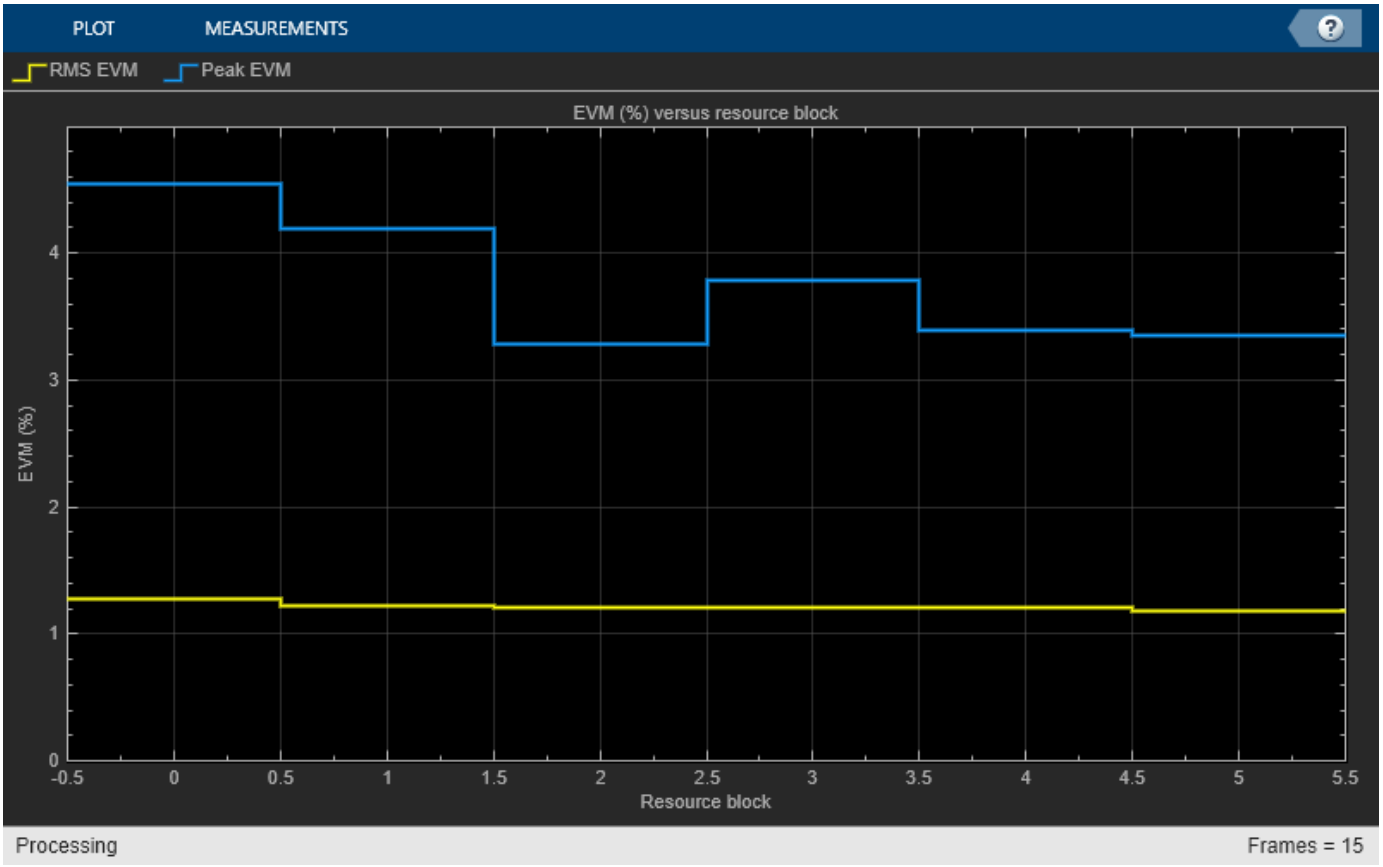
```
Low edge DRS EVM, slot 0: 1.298%
Low edge DRS EVM, slot 1: 1.266%
High edge PUSCH EVM, slot 0: 1.189%
High edge PUSCH EVM, slot 1: 1.221%
High edge DRS EVM, slot 0: 1.262%
High edge DRS EVM, slot 1: 1.297%
Low edge PUSCH EVM, slot 2: 1.156%
Low edge PUSCH EVM, slot 3: 1.260%
Low edge DRS EVM, slot 2: 1.309%
Low edge DRS EVM, slot 3: 1.089%
High edge PUSCH EVM, slot 2: 1.137%
High edge PUSCH EVM, slot 3: 1.249%
High edge DRS EVM, slot 2: 1.321%
High edge DRS EVM, slot 3: 1.022%
Low edge PUSCH EVM, slot 4: 1.240%
Low edge PUSCH EVM, slot 5: 1.223%
Low edge DRS EVM, slot 4: 1.516%
Low edge DRS EVM, slot 5: 1.035%
High edge PUSCH EVM, slot 4: 1.232%
High edge PUSCH EVM, slot 5: 1.234%
High edge DRS EVM, slot 4: 1.442%
High edge DRS EVM, slot 5: 1.114%
Low edge PUSCH EVM, slot 6: 1.308%
Low edge PUSCH EVM, slot 7: 1.284%
Low edge DRS EVM, slot 6: 1.335%
Low edge DRS EVM, slot 7: 1.233%
High edge PUSCH EVM, slot 6: 1.358%
High edge PUSCH EVM, slot 7: 1.284%
High edge DRS EVM, slot 6: 1.445%
High edge DRS EVM, slot 7: 1.185%
Low edge PUSCH EVM, slot 8: 1.336%
Low edge PUSCH EVM, slot 9: 1.289%
Low edge DRS EVM, slot 8: 0.943%
Low edge DRS EVM, slot 9: 1.303%
High edge PUSCH EVM, slot 8: 1.368%
High edge PUSCH EVM, slot 9: 1.282%
High edge DRS EVM, slot 8: 0.985%
High edge DRS EVM, slot 9: 1.255%
Low edge PUSCH EVM, slot 10: 1.211%
Low edge PUSCH EVM, slot 11: 1.199%
Low edge DRS EVM, slot 10: 1.244%
Low edge DRS EVM, slot 11: 1.126%
High edge PUSCH EVM, slot 10: 1.211%
High edge PUSCH EVM, slot 11: 1.203%
High edge DRS EVM, slot 10: 1.257%
High edge DRS EVM, slot 11: 1.068%
Low edge PUSCH EVM, slot 12: 1.275%
Low edge PUSCH EVM, slot 13: 1.079%
Low edge DRS EVM, slot 12: 1.275%
Low edge DRS EVM, slot 13: 1.197%
High edge PUSCH EVM, slot 12: 1.298%
High edge PUSCH EVM, slot 13: 1.070%
High edge DRS EVM, slot 12: 1.187%
High edge DRS EVM, slot 13: 1.139%
Low edge PUSCH EVM, slot 14: 1.133%
Low edge PUSCH EVM, slot 15: 1.253%
Low edge DRS EVM, slot 14: 1.287%
Low edge DRS EVM, slot 15: 1.014%
```

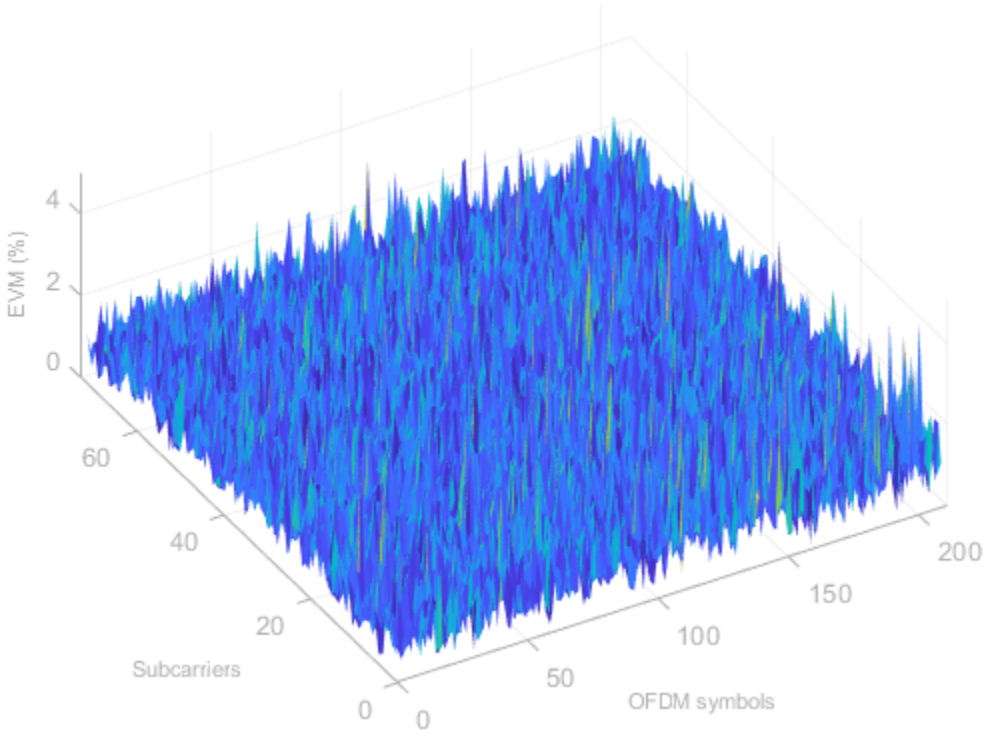
```
High edge PUSCH EVM, slot 14: 1.117%
High edge PUSCH EVM, slot 15: 1.262%
High edge DRS EVM, slot 14: 1.345%
High edge DRS EVM, slot 15: 0.980%
Low edge PUSCH EVM, slot 16: 1.327%
Low edge PUSCH EVM, slot 17: 1.244%
Low edge DRS EVM, slot 16: 1.351%
Low edge DRS EVM, slot 17: 1.268%
High edge PUSCH EVM, slot 16: 1.362%
High edge PUSCH EVM, slot 17: 1.193%
High edge DRS EVM, slot 16: 1.406%
High edge DRS EVM, slot 17: 1.322%
Low edge PUSCH EVM, slot 18: 1.370%
Low edge PUSCH EVM, slot 19: 1.328%
Low edge DRS EVM, slot 18: 1.229%
Low edge DRS EVM, slot 19: 1.354%
High edge PUSCH EVM, slot 18: 1.359%
High edge PUSCH EVM, slot 19: 1.341%
High edge DRS EVM, slot 18: 1.132%
High edge DRS EVM, slot 19: 1.430%
Averaged low edge PUSCH EVM, frame 0: 1.251%
Averaged high edge PUSCH EVM, frame 0: 1.251%
Averaged PUSCH EVM frame 0: 1.251%
Averaged DRS EVM frame 0: 1.238%
Low edge PUSCH EVM, slot 0: 1.402%
Low edge PUSCH EVM, slot 1: 1.343%
Low edge DRS EVM, slot 0: 1.272%
Low edge DRS EVM, slot 1: 1.157%
High edge PUSCH EVM, slot 0: 1.372%
High edge PUSCH EVM, slot 1: 1.358%
High edge DRS EVM, slot 0: 1.310%
High edge DRS EVM, slot 1: 1.122%
Low edge PUSCH EVM, slot 2: 1.292%
Low edge PUSCH EVM, slot 3: 1.235%
Low edge DRS EVM, slot 2: 1.273%
Low edge DRS EVM, slot 3: 1.502%
High edge PUSCH EVM, slot 2: 1.245%
High edge PUSCH EVM, slot 3: 1.199%
High edge DRS EVM, slot 2: 1.280%
High edge DRS EVM, slot 3: 1.527%
Low edge PUSCH EVM, slot 4: 1.410%
Low edge PUSCH EVM, slot 5: 1.114%
Low edge DRS EVM, slot 4: 1.464%
Low edge DRS EVM, slot 5: 1.195%
High edge PUSCH EVM, slot 4: 1.426%
High edge PUSCH EVM, slot 5: 1.141%
High edge DRS EVM, slot 4: 1.513%
High edge DRS EVM, slot 5: 1.159%
Low edge PUSCH EVM, slot 6: 1.262%
Low edge PUSCH EVM, slot 7: 1.288%
Low edge DRS EVM, slot 6: 1.376%
Low edge DRS EVM, slot 7: 0.913%
High edge PUSCH EVM, slot 6: 1.288%
High edge PUSCH EVM, slot 7: 1.305%
High edge DRS EVM, slot 6: 1.361%
High edge DRS EVM, slot 7: 0.900%
Low edge PUSCH EVM, slot 8: 1.445%
Low edge PUSCH EVM, slot 9: 1.256%
```

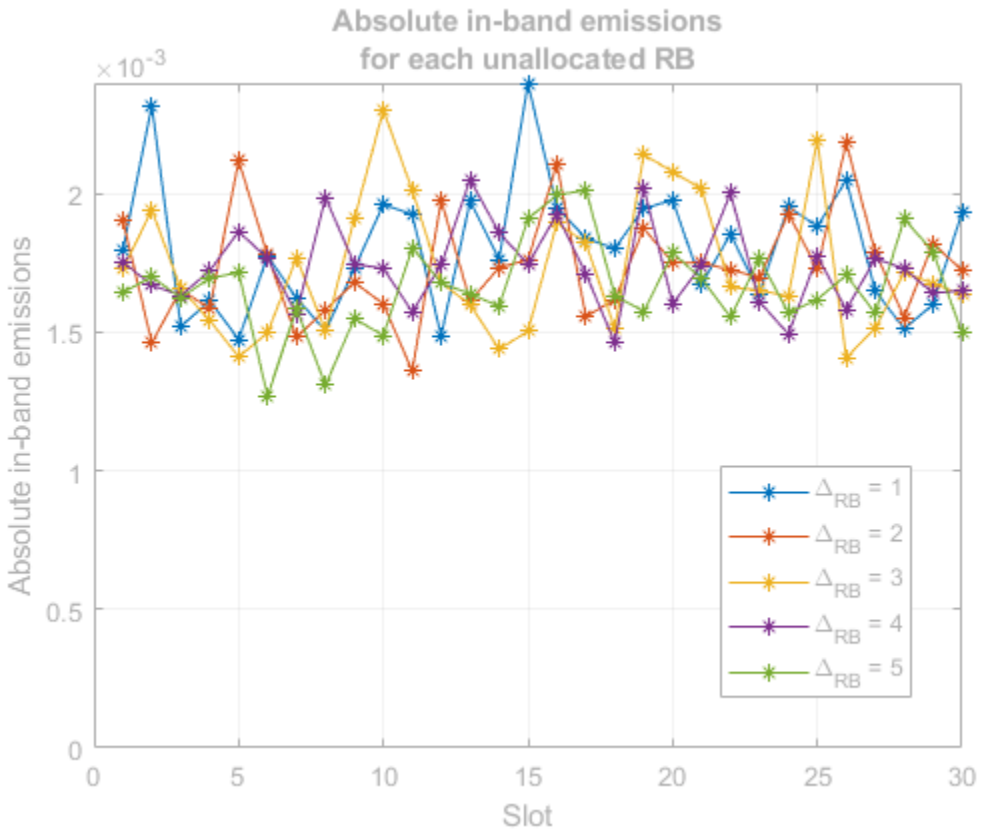
```
Low edge DRS EVM, slot 8: 1.360%
Low edge DRS EVM, slot 9: 1.308%
High edge PUSCH EVM, slot 8: 1.452%
High edge PUSCH EVM, slot 9: 1.298%
High edge DRS EVM, slot 8: 1.397%
High edge DRS EVM, slot 9: 1.296%
Averaged overall PUSCH EVM: 1.251%
Averaged overall DRS EVM: 1.238%
```




SCOPE
MEASUREMENTS
RMS EVM
Peak EVM
EVM (%) versus OFDM symbol
EVM (%)
0
1
2
3
4
0
20
40
60
80
100
120
140
160
180
200
Time
Processing
Frames = 15 T = 209.000


PLOT
MEASUREMENTS
RMS EVM
Peak EVM
EVM (%) versus resource block
EVM (%)
0
1
2
3
4
-0.5
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
5.5
Resource block
Processing
Frames = 15


EVM (%)
4
2
0
60
40
20
0
Subcarriers
0
50
100
150
200
OFDM symbols



Appendix

This example uses these helper functions.

- hPUSCHEVM
- hPUSCHEVMEmissionsPlot

Selected Bibliography

1 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception"

LTE Downlink Adjacent Channel Leakage Power Ratio (ACLR) Measurement

This example shows how the Adjacent Channel Leakage Power Ratio (ACLR) can be measured within a downlink Reference Measurement Channel (RMC) signal using the LTE Toolbox™.

Introduction

This example performs Adjacent Channel Leakage power Ratio (ACLR) measurements according to TS36.104, Section 6.6.2 [1] for a downlink waveform. ACLR is used as a measure of the amount of power leaking into adjacent channels and is defined as the ratio of the filtered mean power centered on the assigned channel frequency to the filtered mean power centered on an adjacent channel frequency. Minimum ACLR conformance requirements are given for E-UTRA (LTE) carriers and UTRA (W-CDMA) carriers.

The example is structured as follows:

- A downlink waveform is generated using a Reference Measurement Channel (RMC) configuration
- Parameters for ACLR measurements are calculated including the required oversampling rate to ensure a signal capable of representing both the E-UTRA and UTRA 1st and 2nd adjacent carriers with at most 85% bandwidth occupancy
- The waveform is oversampled as required
- The E-UTRA ACLR is calculated using a square measurement filter
- The UTRA ACLR is calculated using a Root Raise Cosine (RRC) filter
- The ACLR measurements are displayed

Waveform Generation

A downlink RMC is used for measuring ACLR and created using `lteRMCDL` and `lteRMCDLTool`.

```
% Generate the downlink configuration structure for RMC R.6
cfg = lteRMCDL('R.6');

% Generate the waveform which is a T-by-P matrix where T is the number of
% time-domain samples and P is the number of receive antennas
[waveform, ~, info] = lteRMCDLTool(cfg, [1; 0; 0; 1]);

% Write the sampling rate and chip rate to the configuration structure to
% allow the calculation of ACLR parameters
cfg.SamplingRate = info.SamplingRate;
cfg.UTRAChipRate = 3.84;              % UTRA chip rate in MCPS
```

Calculate ACLR Parameters

The parameters required for ACLR measurement are calculated using the helper function hACLRParameters.m.

- *Determine Required Oversampling.* If the input waveform sampling rate (`cfg.SamplingRate`) is not sufficient to span the entire bandwidth (`aclr.BandwidthACLR`) of the adjacent channels (allowing for a maximum of 85% bandwidth occupancy), an upsampled version of the waveform must be used for ACLR calculations. `aclr.OSR` is the upsampling factor.

- *Determine UTRA Parameters;* the chip rates and bandwidths.

```
% Calculate ACLR measurement parameters
[aclr, nRC, R_C, BWUTRA] = hACLRParameters(cfg);
```

Perform Filtering of the Waveform to Improve ACLR

The waveform generated above has no filtering, so there are significant out of band spectral emissions owing to the implicit rectangular pulse shaping in the OFDM modulation (each OFDM subcarrier has a sinc shape in the frequency domain). In order to achieve a good ACLR performance, filtering must be applied to the waveform. A filter is designed with a transition band that starts at the edge of the occupied transmission bandwidth (`aclr.BandwidthConfig`) and stops at the edge of the overall channel bandwidth (`aclr.Bandwidth`). This filter involves no rate change, it just shapes the spectrum within the original bandwidth of the waveform. The filter is first designed, then applied to the waveform.

```
% Design filter
firFilter = dsp.LowpassFilter();
firFilter.SampleRate = info.SamplingRate;
firFilter.PassbandFrequency = aclr.BandwidthConfig/2;
firFilter.StopbandFrequency = aclr.Bandwidth/2;

% Apply filter
waveform = firFilter(waveform);
```

Calculate E-UTRA and UTRA ACLR

The ACLR for E-UTRA and UTRA is measured using two helper functions:

- hACLRMeasurementEUTRA.m measures the E-UTRA ACLR using a square window on adjacent channels. A DFT of the measurement signal is taken and the energy of the appropriate bins used to calculate the adjacent channel powers.
- hACLRMeasurementUTRA.m measures the UTRA ACLR using a RRC filter on adjacent channels with a roll-off factor of 0.22 and bandwidth equal to the chip rate.

```
% Apply required oversampling
resampled = resample(waveform,aclr.OSR,1);

% Calculate E-UTRA ACLR
aclr = hACLRMeasurementEUTRA(aclr, resampled);

% Calculate UTRA ACLR
aclr = hACLRMeasurementUTRA(aclr, resampled, nRC, R_C, BWUTRA);
```

Display Results

The ACLR results are returned in structure `aclr`. `aclr` contains the fields:

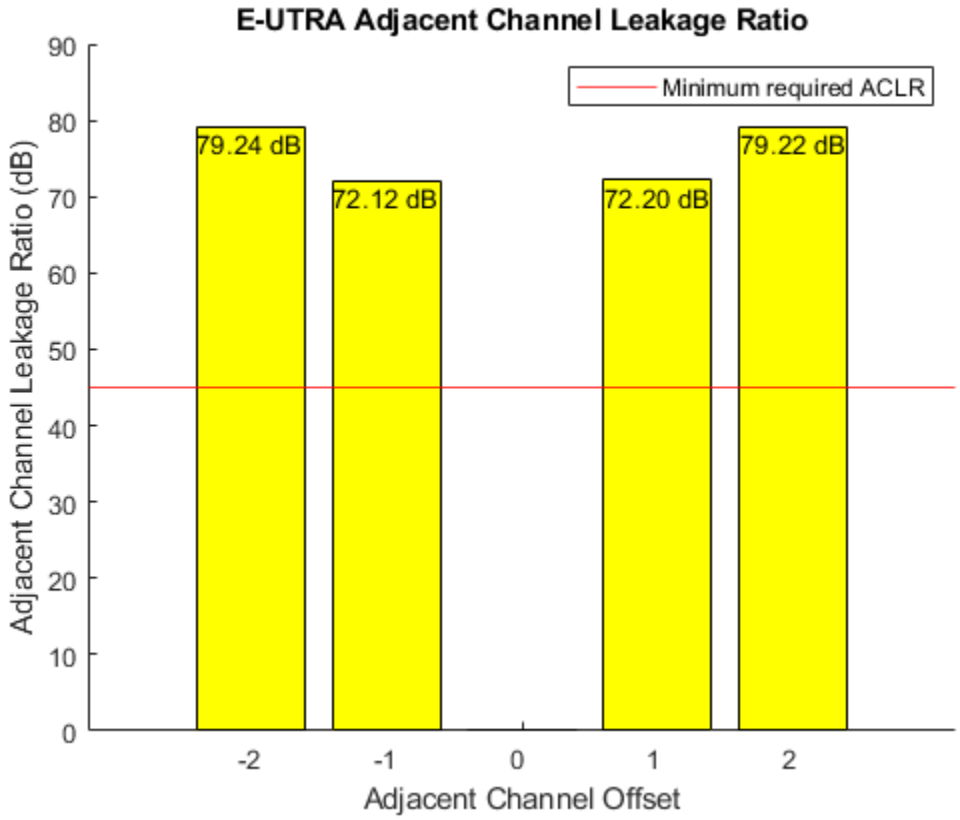
- `Bandwidth` : The channel bandwidth associated with `cfg.NDLRB` or `cfg.NULRB`, in Hertz. This is the overall bandwidth of the assigned channel.
- `BandwidthConfig` : The transmission bandwidth configuration associated with `cfg.NDLRB` or `cfg.NULRB`, in Hertz. This is the bandwidth within the channel bandwidth that contains active subcarriers.
- `BandwidthACLR` : The bandwidth required to represent both the E-UTRA and UTRA 1st and 2nd adjacent carriers; the sampling rate used internally for ACLR measurements will support this bandwidth with at most 85% bandwidth occupancy.

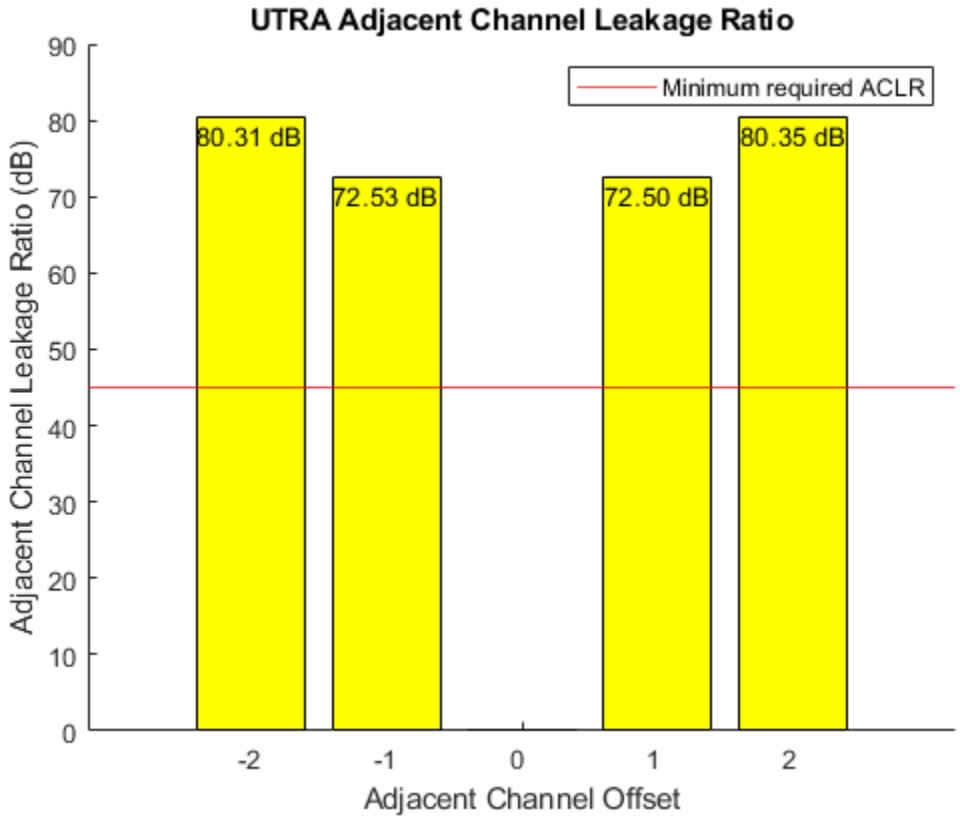
- `OSR` : The integer oversampling ratio of the input `waveform` required to create a signal capable of representing both the E-UTRA and UTRA 1st and 2nd adjacent carriers i.e. to represent `aclr.BandwidthACLR` with at most 85% bandwidth occupancy.
- `SamplingRate` : The sampling rate of the internal measurement signal from which the ACLR is calculated. If `OSR=1`, this signal is the input waveform; if `OSR>1`, this signal is the input waveform upsampled by `OSR`. Therefore: `aclr.SamplingRate = OSR*cfg.SamplingRate`.
- `EUTRAPowerdBm` : The power, in decibels relative to 1mW in 1ohm, of the input within the E-UTRA channel of interest i.e. in a square filter of bandwidth `aclr.BandwidthConfig` centered at 0Hz.
- `EUTRAdB` : A vector of E-UTRA ACLRs, in decibels relative to `aclr.EUTRAPowerdBm`, measured for adjacent channels [-2, -1, 1, 2].
- `EUTRACenterFreq`: A vector of E-UTRA center frequencies, in Hertz, for adjacent channels [-2, -1, 1, 2].
- `UTRAPowerdBm` : A vector of powers, in decibels relative to 1mW in 1ohm, of the input within the UTRA channel of interest; each element of the vector corresponds to each of the configured UTRA chip rates i.e. `UTRAPowerdBm(i)` gives the power of the input in an RRC filter designed for `R=cfg.UTRAChipRate(i)` Mchips/s, `alpha=0.22`, centered at 0 Hz.
- `UTRAdB` : A matrix of UTRA ACLRs, in decibels relative to `aclr.EUTRAPowerdBm`. The columns give the values for adjacent channels [-2, -1, 1, 2] and the rows give the values for each of the configured UTRA chip rates. Note that as required by the standard, these ACLRs are relative to `aclr.EUTRAPowerdBm`, not `aclr.UTRAPowerdBm`.
- `UTRACenterFreq` : A matrix of UTRA center frequencies, in Hertz. The columns give the values for adjacent channels [-2, -1, 1, 2] and the rows give the values for each of the configured UTRA chip rates.

hACLRResults.m displays the ACLR and plots the adjacent channel powers. According to TS 36.104 Table 6.6.2.1-1 [1], the minimum required ACLR for a Base Station in paired spectrum is 45 dB. As the ACLR results are higher than 45 dB, they fall within the requirements.

```
minACLR = 45;
hACLRResults(aclr,minACLR);
```

```
          Bandwidth: 5000000
    BandwidthConfig: 4500000
      BandwidthACLR: 25000000
                OSR: 4
       SamplingRate: 30720000
    EUTRACenterFreq: [-10000000 -5000000 5000000 10000000]
      EUTRAPowerdBm: -0.5918
            EUTRAdB: [79.2357 72.1187 72.2046 79.2157]
       UTRAPowerdBm: -1.3397
             UTRAdB: [80.3117 72.5323 72.5011 80.3540]
     UTRACenterFreq: [-10000000 -5000000 5000000 10000000]
```


E-UTRA Adjacent Channel Leakage Ratio
Minimum required ACLR
79.24 dB
72.12 dB
72.20 dB
79.22 dB
Adjacent Channel Leakage Ratio (dB)
Adjacent Channel Offset
90
80
70
60
50
40
30
20
10
0
-2
-1
0
1
2

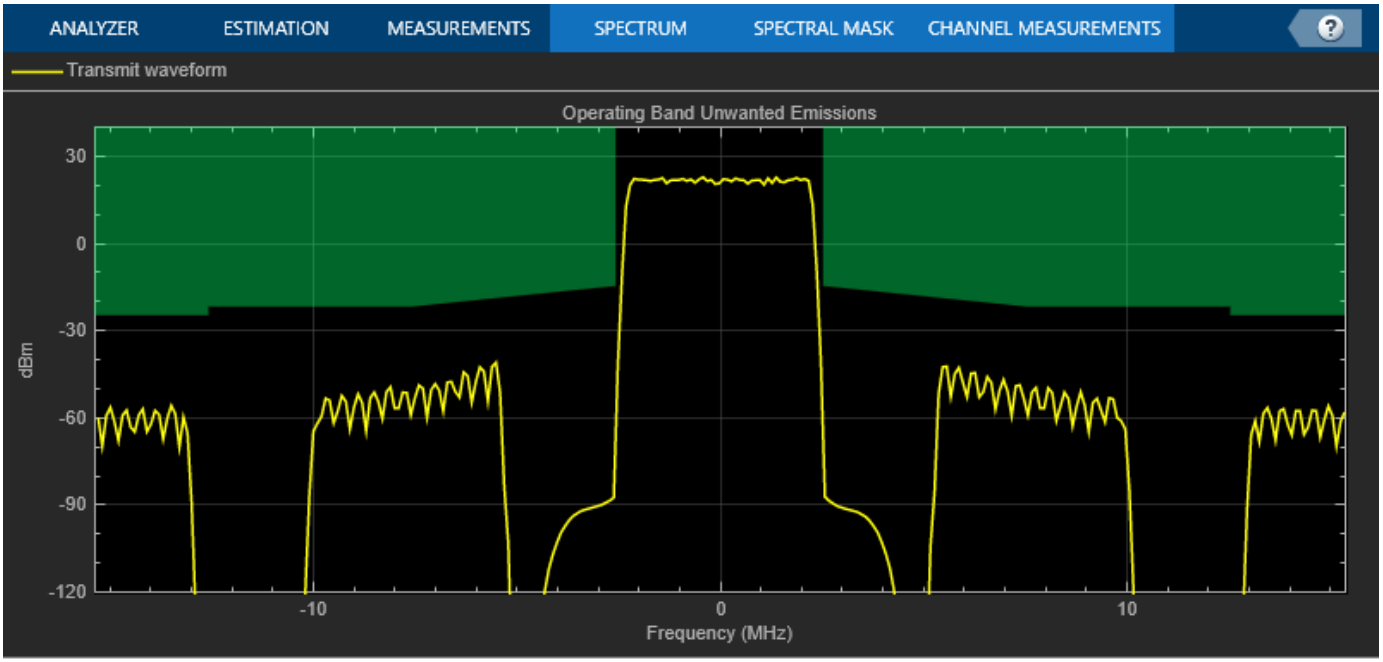


Operating Band Unwanted Emissions (Spectral Mask)

The waveform spectrum is displayed in conjunction with the transmit spectral mask defined in TS 36.104. This example assumes that the waveform corresponds to a Medium Range BS as described in TS 36.104 Table 6.2-1 and the power is set to 38.0dBm (~21.5dBm/100kHz). The corresponding spectral mask is provided in TS 36.104 Table 6.6.3.2C-5 "Medium Range BS operating band unwanted emission limits for 5, 10, 15 and 20MHz channel bandwidth, 31 < P_max,c <= 38dBm".

```
% Adjust waveform power to maximum rated output power
P_max = 38.0; % TS 36.104 Table 6.2-1
bsWaveform = resampled * 10^((P_max-aclr.EUTRAPowerdBm)/20);

% Create a spectrum analyzer, configure it for the waveform sampling rate
% and a resolution bandwidth of 100kHz, configure and display the spectral
% mask in TS 36.104 Table 6.6.3.2C-5, and perform spectrum analysis of the
% waveform
rbw = 100e3; % resolution bandwidth
vbw = 1e3;  % video bandwidth
spectrumScope = spectrumAnalyzer;
spectrumScope.Name = 'Operating Band Unwanted Emissions';
spectrumScope.Title = spectrumScope.Name;
spectrumScope.SampleRate = info.SamplingRate * aclr.OSR;
spectrumScope.RBWSource = 'property';
spectrumScope.RBW = rbw;
spectrumScope.AveragingMethod = 'vbw';
spectrumScope.VBWSource = 'property';
spectrumScope.VBW = vbw;
```

```
spectrumScope.ShowLegend = true;
spectrumScope.ChannelNames = {'Transmit waveform'};
spectrumScope.YLimits = [-120 40];

% "Frequency offset of measurement filter center frequency" in table
f_offset = [0.05; 5.05; 10.05; 10.05] * 1e6;

% "Minimum requirement" in table (dBm/100kHz)
mask_power = [(P_max-53); (P_max-60)*[1; 1]; min(P_max-60,-25)*[1; 1];];
% Add vertical mask segment at the band edge; no particular in-band power
% mask is required to be met for this test
f_offset = [repmat(f_offset(1),2,1); f_offset];
mask_power = [spectrumScope.YLimits(2)*[1; 1]; mask_power];

% Extend mask to analysis bandwidth edge, assumed to be closer to the edge
% frequency of the carrier of interest than f_offset_max (the offset to the
% frequency 10MHz outside the downlink operating band)
mask_freq = [f_offset + aclr.Bandwidth/2; spectrumScope.SampleRate/2];

% Add a mirrored version of the mask to cover negative frequencies, and
% enable the mask
mask_power = [flipud(mask_power); mask_power];
mask_freq = [-flipud(mask_freq); (mask_freq)];
spectrumScope.SpectralMask.EnabledMasks = 'Upper';
spectrumScope.SpectralMask.UpperMask = [mask_freq, mask_power];

% Perform spectrum analysis
spectrumScope(bsWaveform);
```



Spectral Mask

Success Rate (%)	Passed Tests	Total Tests	Current Status	Upper Mask St...	Lower Mask St...	Failing Channe...
100.0000	1.0	1.0	Passing	Passing	--	

Processing VBW = 1.00000 kHz RBW = 99.7403 kHz Sample Rate = 30.7200 MHz Frames = 1 T = 0.00999997

Perform Downsampling and Measure EVM

Finally the waveform is downsampled and resynchronized and an EVM measurement is performed. For more information on making EVM measurements, see “PDSCH Error Vector Magnitude (EVM) Measurement” on page 2-583. According to TS 36.104 Table 6.5.2-1 [1], the maximum EVM when the constellation is 64QAM is 8%. As the overall EVM, around 0.78%, is lower than 8%, this measurement falls within the requirements.

```
downsampled = resample(resampled,1,aclr.OSR);
offset = lteDLFrameOffset(cfg,downsampled,'TestEVM');
cec.PilotAverage = 'TestEVM';
evmsettings.EnablePlotting = 'Off';
evm = hPDSCHEVM(cfg,cec,downsampled(1+offset:end,:),evmsettings);
```

```
Low edge EVM, subframe 0: 0.741%
High edge EVM, subframe 0: 0.707%
Low edge EVM, subframe 1: 0.885%
High edge EVM, subframe 1: 0.789%
Low edge EVM, subframe 2: 0.785%
High edge EVM, subframe 2: 0.699%
Low edge EVM, subframe 3: 0.698%
High edge EVM, subframe 3: 0.667%
Low edge EVM, subframe 4: 0.817%
High edge EVM, subframe 4: 0.640%
Low edge EVM, subframe 6: 0.840%
High edge EVM, subframe 6: 0.747%
Low edge EVM, subframe 7: 0.732%
```

```
High edge EVM, subframe 7: 0.696%
Low edge EVM, subframe 8: 0.742%
High edge EVM, subframe 8: 0.745%
Averaged overall EVM: 0.783%
```

Further Exploration

You can modify parts of this example to calculate ACLR (as per TS36.101, Section 6.6.2.3 [2]) for the uplink by using `lteRMCUL` in the place of `lteRMCDL` to generate the waveform.

Selected Bibliography

1 3GPP TS 36.104 "Base Station (BS) radio transmission and reception"

2 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception"

PDSCH Error Vector Magnitude (EVM) Measurement

This example measures the EVM within a downlink reference measurement channel (RMC) signal and a downlink test model (E-TM) signal, according to the EVM measurement requirements specified in TS 36.104, Annex E [1].

Introduction

This example creates an RMC signal and applies noise to the transmission to model transmitter EVM. Frequency offset and IQ offset are also applied. The impaired signal is then processed according to the EVM measurement requirements specified in TS 36.104, Annex E [1]. This example measures peak and RMS EVM averaged across the two frames of the input signal.

The average EVM is measured at two locations in time (low and high), where the low and high locations correspond to the alignment of the FFT window within the start and end of the cyclic prefix. LTE Toolbox™ requires the low and high locations to be specified as a fraction of the cyclic prefix length.

Note that for multi-antenna RMCs, the EVM measurement assumes each receive signal antenna is directly connected to each transmit signal antenna, as shown in TS36.141 Annex I.1.1 [2]. In accordance with the EVM measurement requirements defined in TS 36.104 Annex E [1], the PDSCH decoding uses only zero forcing equalization. For an illustration of PDSCH reception that includes complete MIMO decoding, see the “Cell Search, MIB and SIB1 Recovery” on page 2-351 example.

Finally the EVM of a Test Model (E-TM) signal is measured, showing how to synchronize an E-TM signal that has been generated outside of MATLAB® or that has been played over-the-air after being generated inside MATLAB.

Transmitter

Setup the transmitter as per TS36.101 RMC [3].

```
% eNodeB Configuration
rng('default');         % Set the default random number generator
rmc = lteRMCDL('R.5');  % Configure RMC
rmc.PDSCH.RVSeq = 0;    % Redundancy version indicator
rmc.TotSubframes = 20;  % Total number of subframes to generate

% Create eNodeB transmission with random PDSCH data
txWaveform = lteRMCDLTool(rmc,randi([0 1],rmc.PDSCH.TrBlkSizes(1),1));
```

Impairment Modeling

Model the transmitter EVM and add frequency and IQ offsets.

```
% Model EVM with additive noise
ofdmInfo   = lteOFDMInfo(rmc);
txEVMpc    = 1.2;          % Transmit EVM in per cent
evmModel   = txEVMpc/(100*sqrt(double(ofdmInfo.Nfft)))* ...
    complex(randn(size(txWaveform)),randn(size(txWaveform)))/sqrt(2);
rxWaveform = txWaveform+evmModel;

% Add frequency offset impairment to received waveform
foffset    = 33.0;         % Frequency offset in Hertz
t = (0:length(rxWaveform)-1).'/ofdmInfo.SamplingRate;
```

```
rxWaveform = rxWaveform.*repmat(exp(1i*2*pi*foffset*t),1,rmc.CellRefP);

% Add IQ offset
iqoffset   = complex(0.01,-0.005);
rxWaveform = rxWaveform+iqoffset;
```

Receiver

The receiver synchronizes with the received signal and computes and displays the measured EVM.

```
% Apply frequency estimation and correction for the purposes of performing
% timing synchronization
foffset_est = lteFrequencyOffset(rmc,rxWaveform);
rxWaveformFreqCorrected = lteFrequencyCorrect(rmc,rxWaveform,foffset_est);

% Synchronize to received waveform
offset = lteDLFrameOffset(rmc,rxWaveformFreqCorrected,'TestEVM');
rxWaveform = rxWaveform(1+offset:end,:);

% Use 'TestEVM' pilot averaging
cec.PilotAverage = 'TestEVM';
```

Perform Measurements

The PDSCH EVM is calculated by calling `hPDSCHEVM`.

The average EVM for a downlink RMC is displayed. First the results for low and high edge EVM are calculated for each subframe within a frame and their averages are displayed at the command window. The max of these averages is the EVM per frame. The final EVM for the downlink RMC is the average of the EVM across all frames. A number of plots are also produced:

- EVM versus OFDM symbol
- EVM versus subcarrier
- EVM versus resource block
- EVM versus OFDM symbol and subcarrier (i.e. the EVM resource grid)

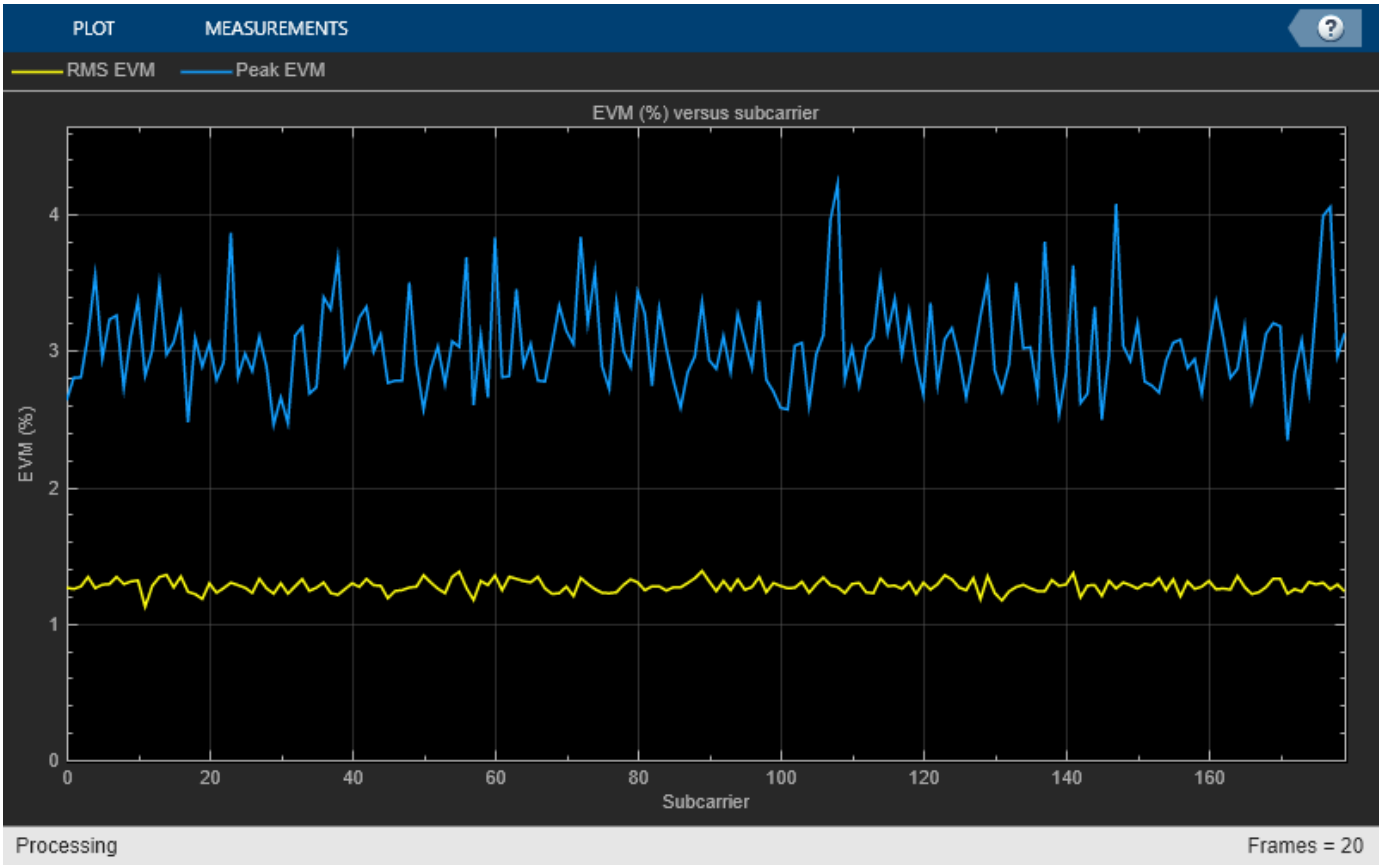
Note that the EVM measurement displayed at the command window is only calculated across allocated PDSCH resource blocks, in accordance with the LTE standard. The EVM plots are shown across all resource blocks (allocated or unallocated), allowing the quality of the signal to be measured more generally. In unallocated resource blocks, the EVM is calculated assuming that the received resource elements have an expected value of zero.

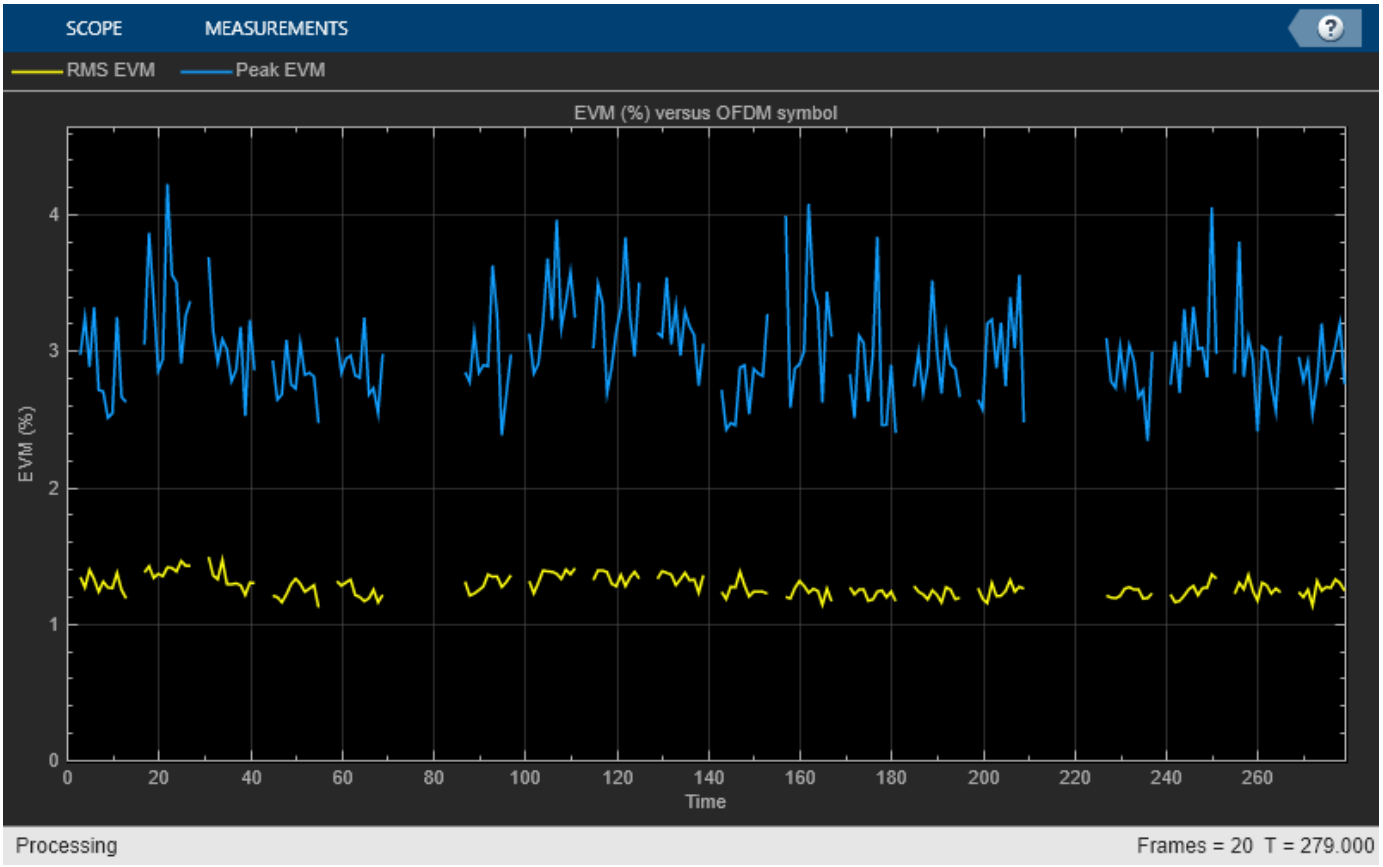
The EVM of each E-UTRA carrier for QPSK, 16QAM, 64QAM and 256QAM modulation schemes on PDSCH should be better than the required EVM of 17.5%, 12.5%, 8% and 3.5% respectively as per TS 36.104 Table 6.5.2-1 [1].

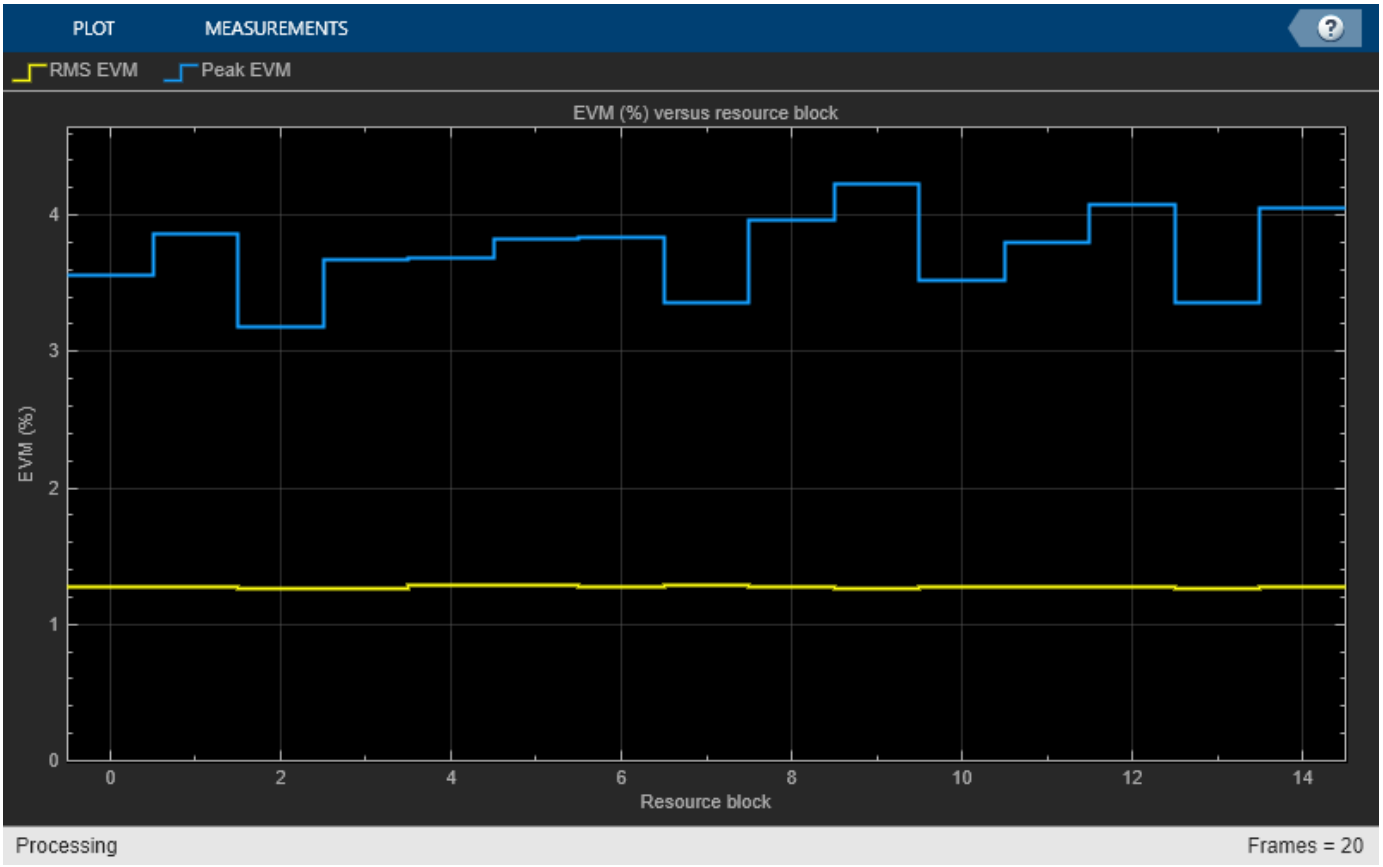
```
% Compute and display EVM measurements
[evmmeas, plots] = hPDSCHEVM(rmc,cec,rxWaveform);
```

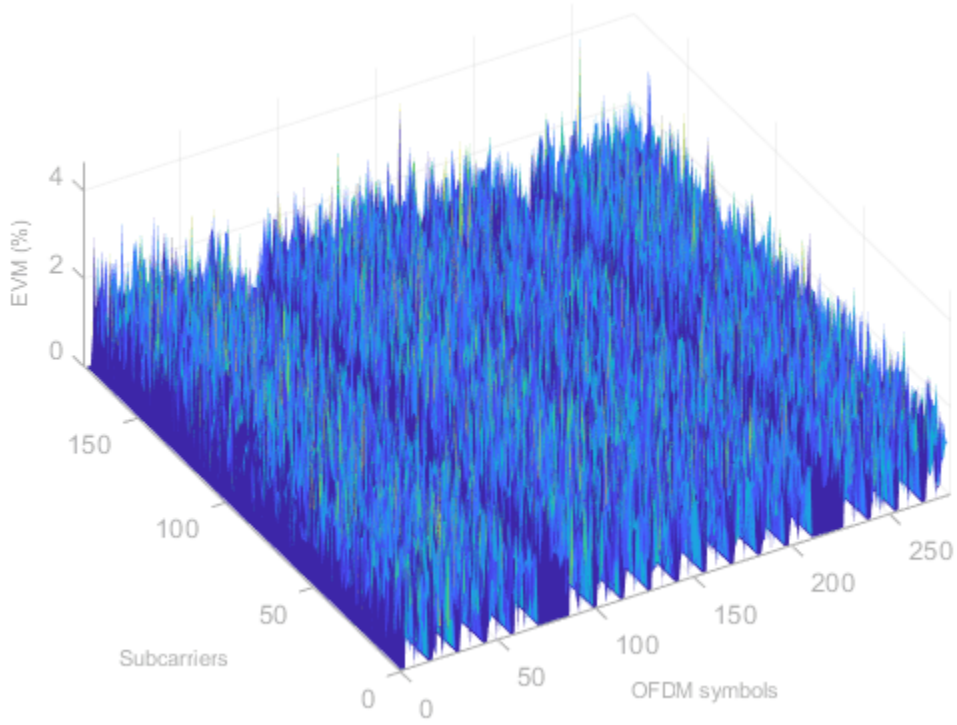
```
Low edge EVM, subframe 0: 1.287%
High edge EVM, subframe 0: 1.289%
Low edge EVM, subframe 1: 1.395%
High edge EVM, subframe 1: 1.390%
Low edge EVM, subframe 2: 1.330%
High edge EVM, subframe 2: 1.324%
Low edge EVM, subframe 3: 1.234%
```

```
High edge EVM, subframe 3: 1.239%
Low edge EVM, subframe 4: 1.235%
High edge EVM, subframe 4: 1.229%
Low edge EVM, subframe 6: 1.296%
High edge EVM, subframe 6: 1.294%
Low edge EVM, subframe 7: 1.350%
High edge EVM, subframe 7: 1.344%
Low edge EVM, subframe 8: 1.338%
High edge EVM, subframe 8: 1.336%
Low edge EVM, subframe 9: 1.331%
High edge EVM, subframe 9: 1.319%
Averaged low edge EVM, frame 0: 1.312%
Averaged high edge EVM, frame 0: 1.308%
Averaged EVM frame 0: 1.312%
Low edge EVM, subframe 0: 1.241%
High edge EVM, subframe 0: 1.243%
Low edge EVM, subframe 1: 1.230%
High edge EVM, subframe 1: 1.229%
Low edge EVM, subframe 2: 1.219%
High edge EVM, subframe 2: 1.220%
Low edge EVM, subframe 3: 1.216%
High edge EVM, subframe 3: 1.220%
Low edge EVM, subframe 4: 1.239%
High edge EVM, subframe 4: 1.239%
Low edge EVM, subframe 6: 1.219%
High edge EVM, subframe 6: 1.207%
Low edge EVM, subframe 7: 1.247%
High edge EVM, subframe 7: 1.246%
Low edge EVM, subframe 8: 1.257%
High edge EVM, subframe 8: 1.252%
Low edge EVM, subframe 9: 1.249%
High edge EVM, subframe 9: 1.246%
Averaged low edge EVM, frame 1: 1.235%
Averaged high edge EVM, frame 1: 1.234%
Averaged EVM frame 1: 1.235%
Averaged overall EVM: 1.274%
```


PLOT
MEASUREMENTS
RMS EVM
Peak EVM
EVM (%) versus subcarrier
EVM (%)
0
1
2
3
4
0
20
40
60
80
100
120
140
160
Subcarrier
Processing
Frames = 20


SCOPE
MEASUREMENTS
RMS EVM
Peak EVM
EVM (%) versus OFDM symbol
EVM (%)
0
1
2
3
4
0
20
40
60
80
100
120
140
160
180
200
220
240
260
Time
Processing
Frames = 20 T = 279.000


PLOT
MEASUREMENTS
RMS EVM
Peak EVM
EVM (%) versus resource block
EVM (%)
Resource block
0
1
2
3
4
0
2
4
6
8
10
12
14
Processing
Frames = 20



EVM Measurement on a Test Model Signal

Finally the EVM of a Test Model (E-TM) signal is measured, showing how to synchronize an E-TM signal that has been generated outside of MATLAB or that has been played over-the-air after being generated inside MATLAB. The following steps are performed:

- *Load captured waveform:* A waveform is generated inside `hGetTestModelWaveform` to simulate an over-the-air E-TM waveform captured at the returned sample rate. For more details on over-the-air transmission and analysis of Test Model waveforms, refer to the following example: “Waveform Generation and Transmission Using LTE Toolbox with Test and Measurement Equipment” on page 2-632.
- *Create local Test Model configuration:* Next a configuration structure representing the E-TM waveform content is created using the function `lteTestModel`. In order to create the configuration, the Test Model number and bandwidth must be known.
- *Resample to expected sampling rate:* The function `lteOFDMInfo` is called to get some information about the OFDM modulation/demodulation used within LTE Toolbox for the Test Model configuration `tmconfig`. The most important information here is `ofdmInfo.SamplingRate` which gives the sampling rate expected for OFDM demodulation of the waveform. The `resample` function is used to resample the captured waveform to this sampling rate.
- *Perform synchronization:* Frequency offset estimation and correction and timing synchronization are performed using the same steps as shown earlier in this example.
- *Measure EVM:* The EVM is measured by calling `hPDSCHEVM`. For E-TMs, the generated waveform contains one or more PDSCHs. The `hPDSCHEVM` function determines which PDSCH to analyze based on TS 36.141, section 6.1.1 [2].

```
% Load the captured Test Model waveform
[tmsignal,SR] = hGetTestModelWaveform();

% Create a local Test Model configuration, corresponding to the known E-TM
% number and bandwidth
tmconfig = lteTestModel('1.1','5MHz');
ofdmInfo = lteOFDMInfo(tmconfig);

% Resample the captured waveform to match the expected sampling rate used
% by LTE Toolbox for the Test Model bandwidth
tmsignal = resample(tmsignal,ofdmInfo.SamplingRate,SR);

% Apply frequency estimation and correction for the purposes of performing
% timing synchronization
foffset_est = lteFrequencyOffset(tmconfig,tmsignal);
tmsignalFreqCorrected = lteFrequencyCorrect(tmconfig,tmsignal,foffset_est);

% Synchronize the captured waveform
offset = lteDLFrameOffset(tmconfig,tmsignalFreqCorrected,'TestEVM');
tmsignal = tmsignal(1+offset:end,:);

% Compute EVM measurements, with plotting disabled
cec.PilotAverage = 'TestEVM';
alg.EnablePlotting = 'Off';
evm_tm = hPDSCHEVM(tmconfig,cec,tmsignal,alg);

Low edge EVM, subframe 0: 2.166%
High edge EVM, subframe 0: 1.922%
Low edge EVM, subframe 1: 2.010%
High edge EVM, subframe 1: 1.904%
Low edge EVM, subframe 2: 2.060%
High edge EVM, subframe 2: 1.915%
Low edge EVM, subframe 3: 1.988%
High edge EVM, subframe 3: 1.910%
Low edge EVM, subframe 4: 2.074%
High edge EVM, subframe 4: 1.920%
Low edge EVM, subframe 5: 2.010%
High edge EVM, subframe 5: 1.913%
Low edge EVM, subframe 6: 2.082%
High edge EVM, subframe 6: 1.912%
Low edge EVM, subframe 7: 2.047%
High edge EVM, subframe 7: 1.920%
Low edge EVM, subframe 8: 1.989%
High edge EVM, subframe 8: 1.905%
Low edge EVM, subframe 9: 2.022%
High edge EVM, subframe 9: 1.905%
Averaged low edge EVM, frame 0: 2.044%
Averaged high edge EVM, frame 0: 1.912%
Averaged EVM frame 0: 2.044%
Averaged overall EVM: 2.044%
```

Appendix

This example uses the following helper functions:

- hPDSCHEVM.m
- hGetTestModelWaveform.m

Selected Bibliography

1. 3GPP TS 36.104 "Base Station (BS) radio transmission and reception"
2. 3GPP TS 36.141 "Base Station (BS) conformance testing"
3. 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception"

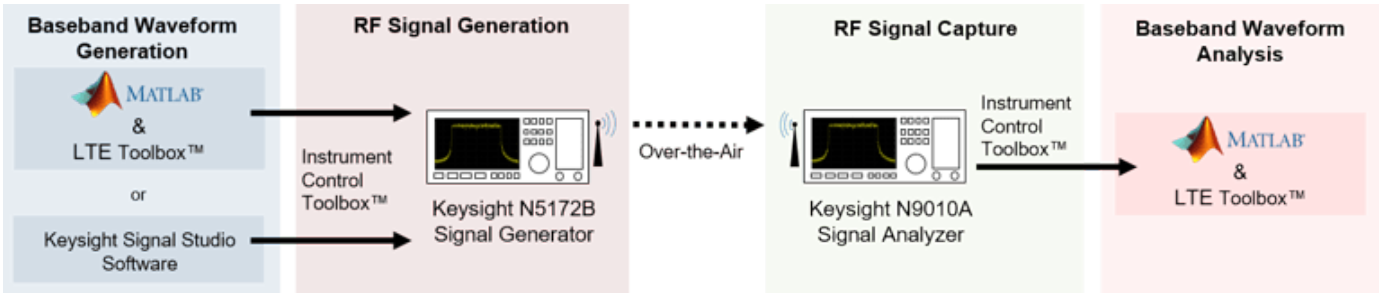
Waveform Acquisition and Analysis using LTE Toolbox with Test and Measurement Equipment

This example shows how an over-the-air LTE waveform can be captured and analyzed using the LTE Toolbox™, the Instrument Control Toolbox™ and RF signal analyzer hardware.

Introduction

The LTE Toolbox can be used to perform both standard compliant and custom decoding and analysis of baseband LTE signals. Using the LTE Toolbox with the Instrument Control Toolbox allows waveforms to be captured using test and measurement hardware and be taken into MATLAB® for visualization, analysis and decoding.

In this example, the Instrument Control Toolbox is used to capture an over-the-air LTE signal using a Keysight Technologies® N9010A signal analyzer and retrieve it into MATLAB for analysis. The over-the-air signal is generated using a Keysight Technologies N5172B signal generator.



In this example, the captured waveform is analyzed by performing two measurements using the LTE Toolbox:

- *Adjacent Channel Leakage Power Ratio*: ACLR is used as a measure of the amount of power leaking into adjacent channels and is defined as the ratio of the filtered mean power centered on the assigned channel frequency to the filtered mean power centered on an adjacent channel frequency. See “LTE Downlink Adjacent Channel Leakage Power Ratio (ACLR) Measurement” on page 2-575 for a more detailed explanation.
- *PDSCH Error Vector Magnitude*: EVM is a measure of the difference between the ideal symbols and the measured symbols after the equalization. See “PDSCH Error Vector Magnitude (EVM) Measurement” on page 2-583 for a more detailed explanation.

Generate the Over-the-Air Signal

The LTE Toolbox can be used to generate standard or custom baseband IQ waveforms. “Waveform Generation and Transmission Using LTE Toolbox with Test and Measurement Equipment” on page 2-632 demonstrates how to generate an over-the-air LTE waveform using the LTE Toolbox and a Keysight Technologies signal generator.

In this example, Keysight Technologies N7624B Signal Studio and an N5172B signal generator are used to generate a standard-compliant RF LTE downlink waveform at a 1GHz center frequency. Note 1GHz is selected as an example frequency and is not intended to be a recognized LTE channel.

A 40ms 5MHz FDD R.6 Reference Measurement Channel (RMC) waveform is generated and looped for capture. The HARQ retransmissions are turned off to simplify synchronization at the receiver and the OCNG is enabled to fill unused resource elements to keep signal power constant.

LTE Signal Parameters

To analyze the received waveform a number of system parameters must be known. As a standard RMC waveform is captured `lteRMCDL` is used to generate a configuration structure for RMC R.6. This provides the parameters required for analysis, such as the signal bandwidth, downlink control configuration and resource allocation. Alternatively, these parameters can be obtained through blind decoding as demonstrated in “Cell Search, MIB and SIB1 Recovery” on page 2-351.

```
% RMC configuration
rmc = lteRMCDL('R.6');
% Ensure that the HARQ retransmissions are turned off at the transmitter so
% that the Redundancy Version (RV) is the same in every subframe. This
% simplifies synchronization as the receiver does not need to take account
% of an RV pattern that spans multiple frames.
rmc.PDSCH.RVSeq = 0; % Single transmission of the transport block
% Enable OCNG fill
rmc.OCNGPDSCHEnable = 'On';
rmc.OCNGPDCCHEnable = 'On';

% Write the sampling rate and UTRA chip rate to the configuration structure
% to allow the calculation of ACLR parameters
info = lteOFDMInfo(rmc);
rmc.SamplingRate = info.SamplingRate;
% UTRA chip rate in MCPS
rmc.UTRAChipRate = 3.84;
```

Calculate ACLR Parameters

The parameters required for ACLR measurement are calculated using the helper function hACLRParameters.m.

- *Determine measurement bandwidth* - The measurement bandwidth range should cover the two E-UTRA adjacent channels of the same bandwidth as the signal and the two 5MHz UTRA channels as given by TS 36.104 Table 6.6.2.1-1
- *Determine UTRA Parameters* - The UTRA chip rates and bandwidths

```
% Calculate ACLR measurement parameters
[aclr, nRC, R_C, BWUTRA] = hACLRParameters(rmc);
```

Acquire the Baseband Signal in MATLAB from a Signal Analyzer

To analyze the over-the-air transmission in MATLAB, the Instrument Control Toolbox is used to configure the Keysight Technologies N9010A signal analyzer and capture baseband IQ data. The helper function hCaptureIQUsingN9010A.m retrieves the baseband IQ data and the capture sampling rate from the signal analyzer, ready for analysis in MATLAB. Note that 40 subframes are captured for analysis.

```
capSubframes = 40;      % Number of subframes to capture
centerFrequency = 1e9;  % 1GHz center frequency

% The frequency range should cover the two E-UTRA adjacent channels of the
% same bandwidth as the signal and the two 5MHz UTRA channels
startFreq = centerFrequency-aclr.BandwidthACLR/2;
```

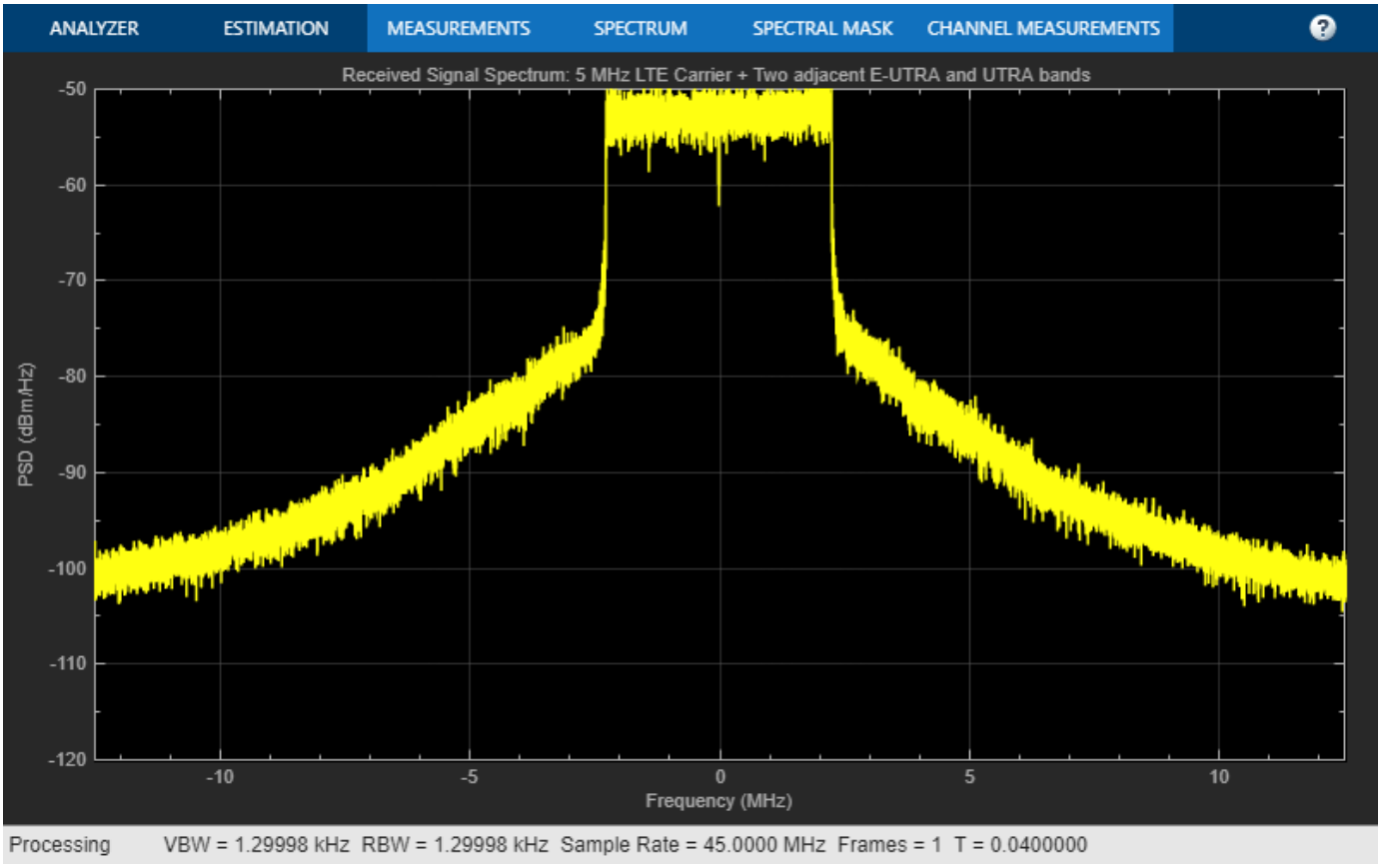
```
stopFreq = centerFrequency+aclr.BandwidthACLR/2;
externalTrigger = false;
capTime = capSubframes*1e-3; % 1 subframes is 1ms
resBW = 91e3;
videoBW = 91e3;
[captureWaveform,captureSampleRate] = hCaptureIQUsingN9010A( ...
    'A-N9010A-21026.dhcp.mathworks.com',capTime, ...
    centerFrequency,stopFreq-startFreq,externalTrigger,startFreq,stopFreq, ...
    resBW,videoBW);

rxWaveform = captureWaveform(1:end-1);
captureSampleRate = round(captureSampleRate);
```

Plot Received Signal Spectrum

Inspect the function hCaptureIQUsingN9010A.m for more details on input parameters and the commands needed to configure the Keysight Technologies N9010A signal analyzer and retrieve the data.

Plotting the frequency spectrum of the retrieved time domain baseband waveform using the DSP System Toolbox™ `spectrumAnalyzer` object shows the expected LTE 5 MHz occupied bandwidth and adjacent bands required for ACLR measurement, with impairments due to RF transmission and reception.

```
spectrumPlotRx = spectrumAnalyzer;
spectrumPlotRx.Method = 'welch';
spectrumPlotRx.SampleRate =  captureSampleRate;
spectrumPlotRx.SpectrumType = 'power-density';
spectrumPlotRx.SpectrumUnits =  'dBm';
spectrumPlotRx.RBWSource = 'property';
spectrumPlotRx.RBW = 1.3e3;
spectrumPlotRx.FrequencySpan = 'span-and-center-frequency';
spectrumPlotRx.Span = aclr.BandwidthACLR;
spectrumPlotRx.CenterFrequency = 0;
spectrumPlotRx.Window = 'rectangular';
spectrumPlotRx.YLimits = [-120 -50];
spectrumPlotRx.YLabel = 'PSD';
spectrumPlotRx.ShowLegend = false;
spectrumPlotRx.Title = 'Received Signal Spectrum: 5 MHz LTE Carrier + Two adjacent E-UTRA and UTR
spectrumPlotRx(rxWaveform);
```

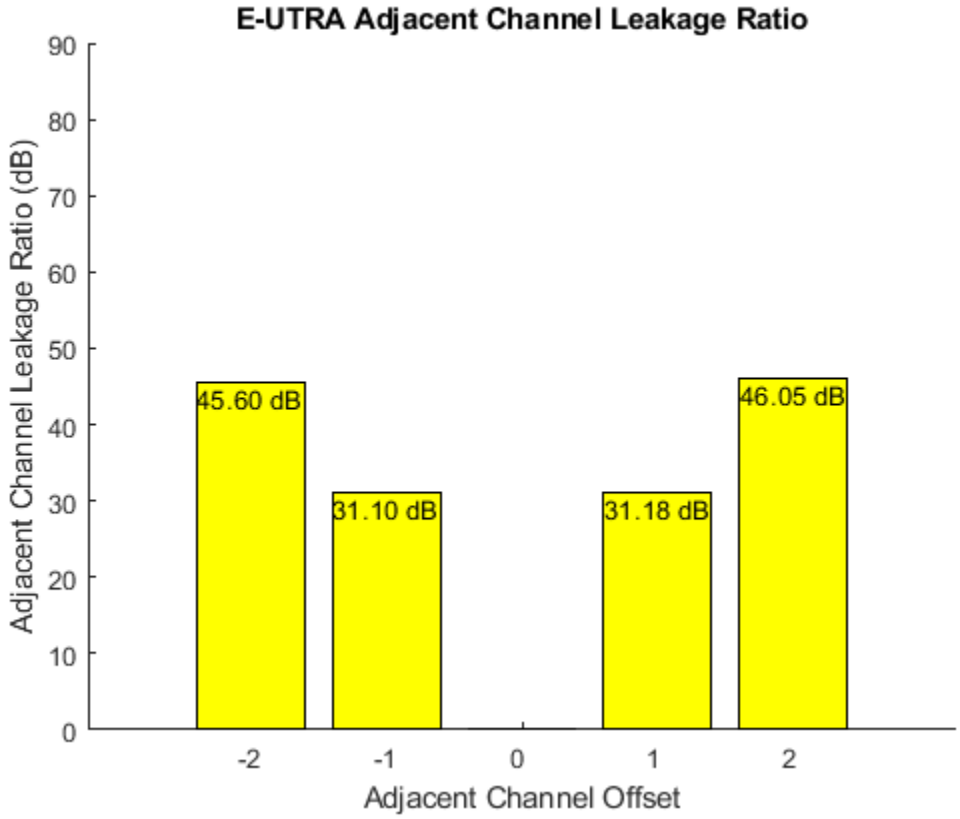


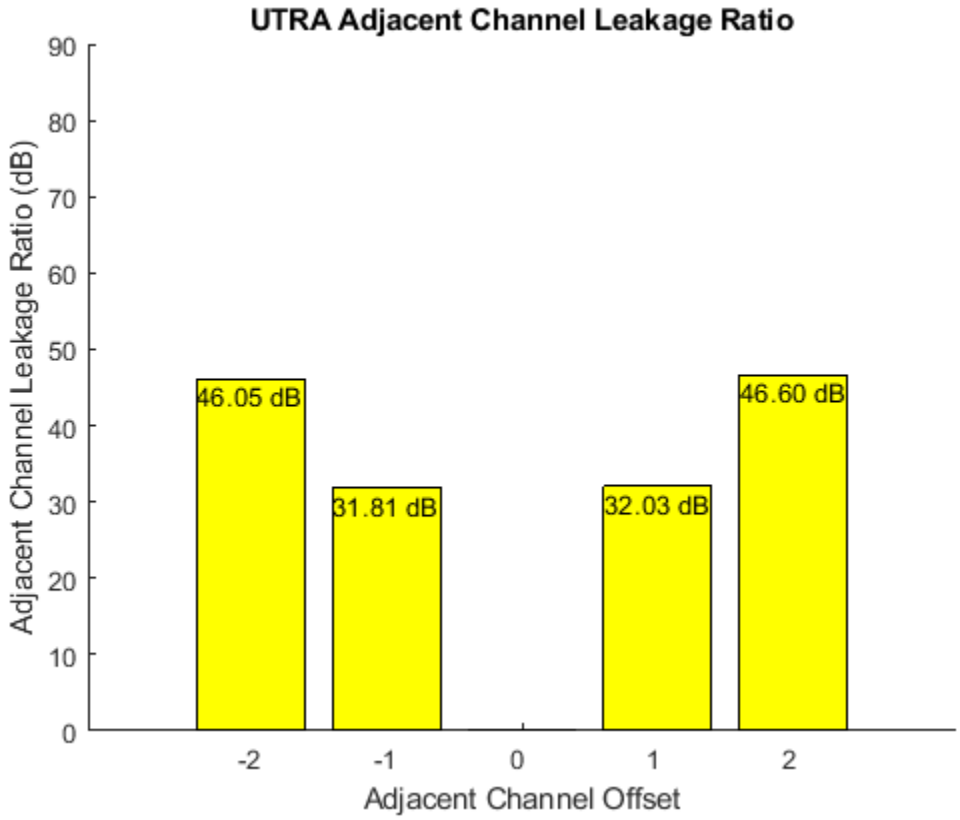
Adjacent Carrier Leakage Ratio Measurement

The E-UTRA and UTRA ACLR of the captured waveform are measured using the helper functions hACLRMeasurementEUTRA.m and hACLRMeasurementUTRA.m. The example “LTE Downlink Adjacent Channel Leakage Power Ratio (ACLR) Measurement” on page 2-575 describes E-UTRA and UTRA measurements in more detail. The filter used in the transmitter affects the ACLR performance, so by optimizing the transmit side filter, improvements can be made to the ACLR.

```
% Apply required resampling for ACLR calculation. The capture sampling rate
% must be greater than the ACLR sampling rate for correct measurement
if captureSampleRate < aclr.SamplingRate
    warning(['The capture sampling rate (%d) is less than the minimum sampling ' ...
        'rate required for ACLR measurement (%d), ACLR may be inaccurate!'],captureSampleRate,ac
end
resampled = resample(rxWaveform,aclr.SamplingRate,captureSampleRate);

% Measure E-UTRA and UTRA ACLR
aclr = hACLRMeasurementEUTRA(aclr,resampled);
aclr = hACLRMeasurementUTRA(aclr,resampled,nRC,R_C,BWUTRA);

% Plot ACLR results
fprintf('\nACLR Analysis:\n');
hACLRResults(aclr);


ACLR Analysis:
          Bandwidth: 5000000
```

```
BandwidthConfig: 4500000
  BandwidthACLR: 25000000
            OSR: 4
   SamplingRate: 30720000
EUTRACenterFreq: [-10000000 -5000000 5000000 10000000]
  EUTRAPowerdBm: 13.9473
        EUTRAdB: [45.6010 31.1040 31.1841 46.0544]
   UTRAPowerdBm: 13.3327
         UTRAdB: [46.0508 31.8147 32.0256 46.6009]
 UTRACenterFreq: [-10000000 -5000000 5000000 10000000]
```





Prepare the Captured LTE Signal for EVM Analysis

The waveform used above for ACLR measurement also contains adjacent bands which are not required for EVM measurement. So the waveform is resampled to the sample rate of the OFDM modulator which will be used to demodulate the received signal, and synchronized to the first frame boundary to allow for OFDM demodulation.

```
rxWaveform = resample(rxWaveform,rmc.SamplingRate,captureSampleRate);

% Synchronize to the first frame head
offset = lteDLFrameOffset(rmc,rxWaveform);
rxWaveform = rxWaveform(1+offset:end,:);

% Extract 2 frames (20ms) for analysis
nFramesAnalyse = 2;
nFramesWaveform = length(rxWaveform)/(info.SamplingRate*10e-3);
rxWaveform = rxWaveform( ...
    1:(info.SamplingRate*(min(nFramesAnalyse,nFramesWaveform)*10e-3)));
```

PDSCH Error Vector Magnitude Measurement

The average EVM of the received PDSCH symbols is measured using the helper function hPDSCHEVM.m. The example “PDSCH Error Vector Magnitude (EVM) Measurement” on page 2-583 demonstrates a standard compliant EVM measurement as per TS 36.104, Annex E [1]. Note that the helper function hPDSCHEVM.m can also measure the EVM of Test Model (E-TM) waveforms such as that generated in “Waveform Generation and Transmission Using LTE Toolbox with Test and Measurement Equipment” on page 2-632.

In this example, the channel estimator is configured to estimate a time and frequency varying channel as an over-the-air signal capture is analyzed. A conservative 9-by-9 pilot averaging window is used, in time and frequency, to reduce the impact of noise on pilot estimates during channel estimation.

```
cec.PilotAverage = 'UserDefined';
cec.FreqWindow = 9;
cec.TimeWindow = 9;
cec.InterpType = 'cubic';
cec.InterpWinSize = 3;
cec.InterpWindow = 'Causal';
```

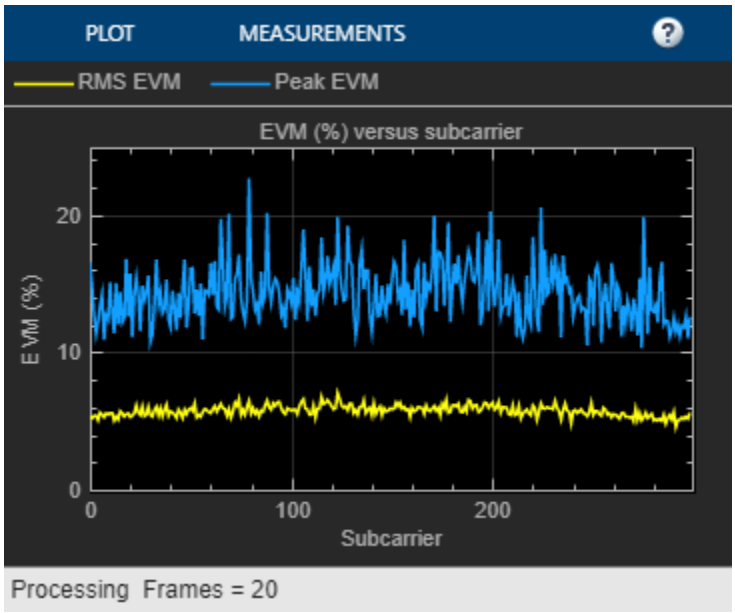
The average EVM for the received waveform is displayed at the command window. A number of plots are also produced:

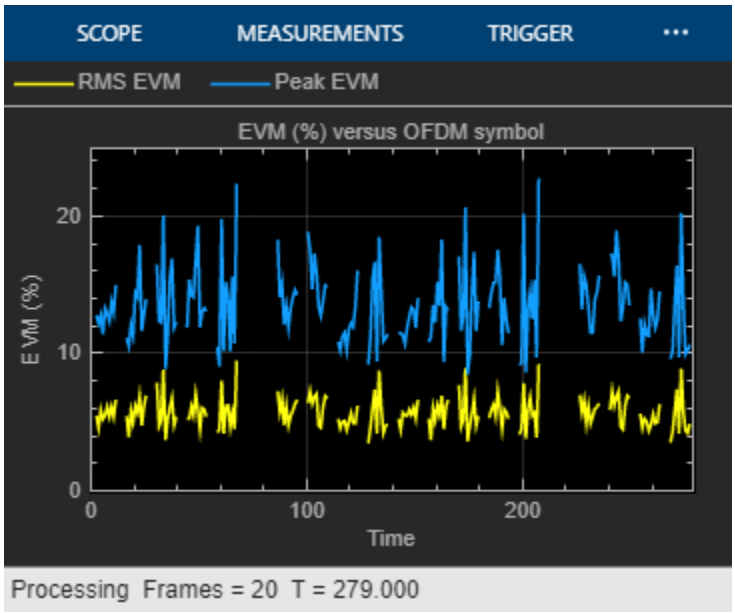
- EVM versus OFDM symbol
- EVM versus subcarrier
- EVM versus resource block
- EVM versus OFDM symbol and subcarrier (i.e. the EVM resource grid)

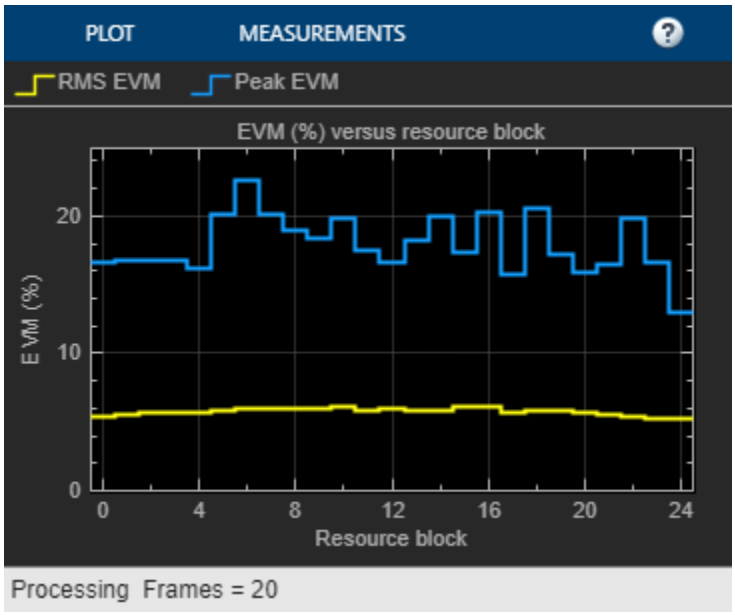
```
% Perform EVM measurement
fprintf('\nEVM Analysis:\n');
[evmMeas, evmPlots] = hPDSCHEVM(rmc,cec,rxWaveform);
```

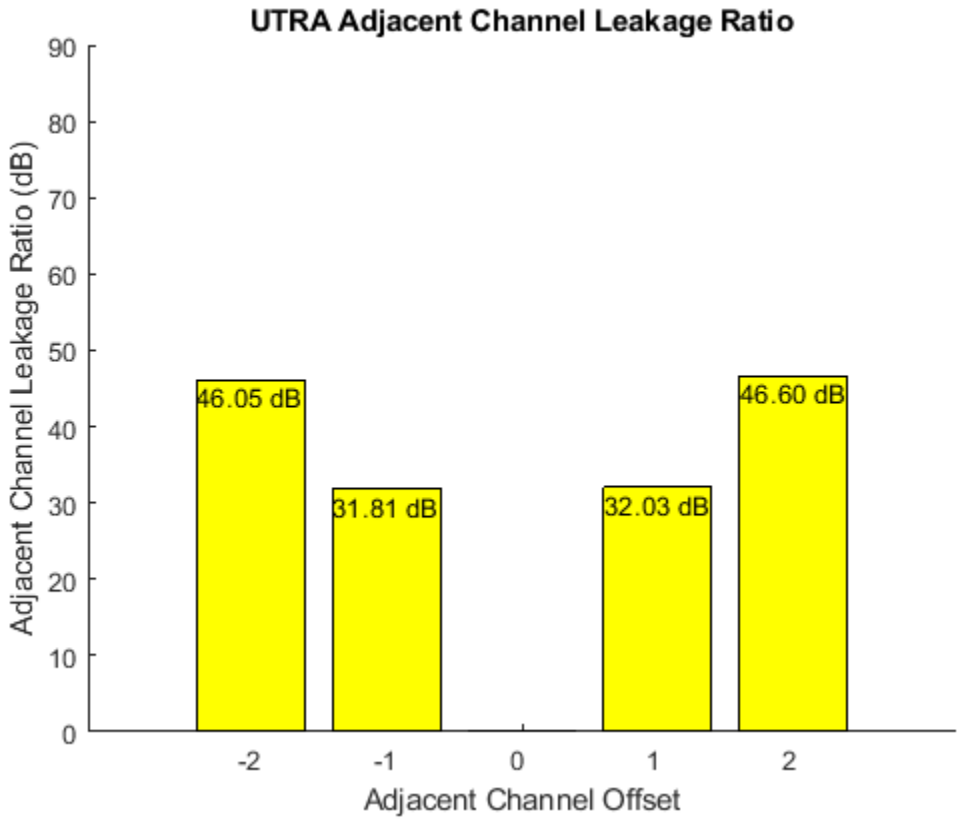
```
EVM Analysis:
Low edge EVM, subframe 0: 5.558%
High edge EVM, subframe 0: 5.551%
Low edge EVM, subframe 1: 5.751%
High edge EVM, subframe 1: 5.736%
Low edge EVM, subframe 2: 5.993%
High edge EVM, subframe 2: 5.991%
Low edge EVM, subframe 3: 5.797%
High edge EVM, subframe 3: 5.815%
Low edge EVM, subframe 4: 5.897%
High edge EVM, subframe 4: 5.912%
Low edge EVM, subframe 6: 5.928%
High edge EVM, subframe 6: 5.938%
Low edge EVM, subframe 7: 6.153%
High edge EVM, subframe 7: 6.148%
Low edge EVM, subframe 8: 5.133%
High edge EVM, subframe 8: 5.113%
Low edge EVM, subframe 9: 5.619%
High edge EVM, subframe 9: 5.619%
Averaged low edge EVM, frame 0: 5.769%
Averaged high edge EVM, frame 0: 5.769%
Averaged EVM frame 0: 5.769%
Low edge EVM, subframe 0: 5.517%
High edge EVM, subframe 0: 5.506%
Low edge EVM, subframe 1: 5.632%
High edge EVM, subframe 1: 5.615%
Low edge EVM, subframe 2: 5.883%
High edge EVM, subframe 2: 5.886%
Low edge EVM, subframe 3: 5.821%
High edge EVM, subframe 3: 5.806%
Low edge EVM, subframe 4: 5.892%
High edge EVM, subframe 4: 5.896%
```

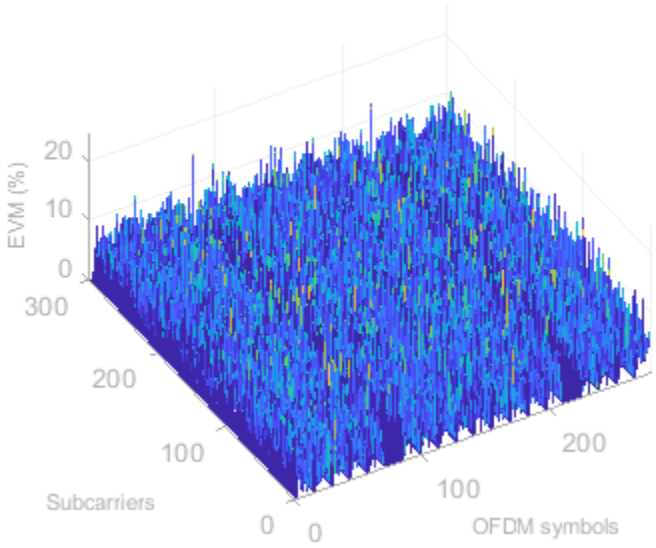
```
Low edge EVM, subframe 6: 5.894%
High edge EVM, subframe 6: 5.889%
Low edge EVM, subframe 7: 6.100%
High edge EVM, subframe 7: 6.108%
Low edge EVM, subframe 8: 5.183%
High edge EVM, subframe 8: 5.167%
Low edge EVM, subframe 9: 5.574%
High edge EVM, subframe 9: 5.594%
Averaged low edge EVM, frame 1: 5.731%
Averaged high edge EVM, frame 1: 5.728%
Averaged EVM frame 1: 5.731%
Averaged overall EVM: 5.750%
```






PLOT
MEASUREMENTS
RMS EVM
Peak EVM
EVM (%) versus resource block
EVM (%)
20
10
0
0
4
8
12
16
20
24
Resource block
Processing Frames = 20


UTRA Adjacent Channel Leakage Ratio
Adjacent Channel Leakage Ratio (dB)
90
80
70
60
50
40
30
20
10
0
46.05 dB
31.81 dB
32.03 dB
46.60 dB
-2
-1
0
1
2
Adjacent Channel Offset



Appendix

This example uses these helper functions:

- hCaptureIQUsingN9010A.m
- hPDSCHEVM.m
- hACLRMeasurementEUTRA.m
- hACLRMeasurementUTRA.m
- hACLRParameters.m
- hACLRResults.m

Selected Bibliography

1 3GPP TS 36.104 "Base Station (BS) radio transmission and reception"

Uplink Carrier Aggregation Waveform Generation, Demodulation, and Analysis

This example shows how multiple uplink carriers can be generated, aggregated, and demodulated using LTE Toolbox™. Error Vector Magnitude (EVM) and in-band emissions are measured for the demodulated carrier as per TS 36.101 Annex F [1].

Introduction

This example models an LTE uplink waveform with Carrier Aggregation (CA). The number of component carriers (CC) and their respective bandwidths can be specified as a parameter. An intra-band contiguous CA case is considered as per TS 36.101 V15.4.0.

To generate an aggregated uplink waveform a user equipment (UE) is configured for each component carrier. Carrier aggregation parameters are calculated and used to generate a modulated waveform for each CC configuration. All the component carrier modulated waveforms are resampled to a common sampling rate so that they can be combined to create an aggregated waveform. The aggregated waveform is demodulated and filtered to extract the component carrier of interest. Then EVM and in-band emissions are measured for that CC and finally a CRC check is made by performing PUSCH decoding.

Number of Component Carriers and Bandwidths

A vector `NULRB` specifies the number of resource blocks (RBs) for each component carrier. The length of this vector corresponds to the number of CCs. The elements of `NULRB` must be in the set {6, 15, 25, 50, 75, 100} RBs. TS 36.101 Table 5.6A.1-1 [1] lists the valid combinations of bandwidths for carrier aggregation.

```
% Configure the set of NULRB values to describe the carriers to be
% aggregated
NULRB = [50 75 100];

% Establish the number of component carriers
numCC = numel(NULRB);

CCBWs = zeros(1,numCC);

if numCC<2
    error('Please specify more than one CC bandwidth value.');
end
```

UE Configuration for each Component Carrier

A configuration structure is generated for each CC using `lteRMCUL`. To show the in-band emissions, partial RB allocation is being considered for each CC. The configuration structures for all CCs are stored in a cell array.

```
% Specify the number of subframes to generate
numSubframes = 10;

% Configure CCs
frc = cell(1,numCC);
for i = 1:numCC
    switch NULRB(i)
```

```
            case 6
                frc{i} = lteRMCUL('A3-1');
                frc{i}.PUSCH.PRBSet = (0:2).';
            case 15
                frc{i} = lteRMCUL('A1-2');
                frc{i}.PUSCH.PRBSet = (0:8).';
            case 25
                frc{i} = lteRMCUL('A1-3');
                frc{i}.PUSCH.PRBSet = (0:15).';
            case 50
                frc{i} = lteRMCUL('A3-5');
                frc{i}.PUSCH.PRBSet = (0:39).';
            case 75
                frc{i} = lteRMCUL('A3-6');
                frc{i}.PUSCH.PRBSet = (0:59).';
            case 100
                frc{i} = lteRMCUL('A3-7');
                frc{i}.PUSCH.PRBSet = (0:80).';
            otherwise
                fprintf('Not a valid number of resource blocks: %d\n',...
                    NULRB(i));
                return
        end
        CCBWs(i) = hNRBToBandwidth(NULRB(i));
        frc{i}.Bandwidth = CCBWs(i);
        frc{i}.TotSubframes = numSubframes;
        frc{i}.PUSCH.RVSeq = 0;
    end
```

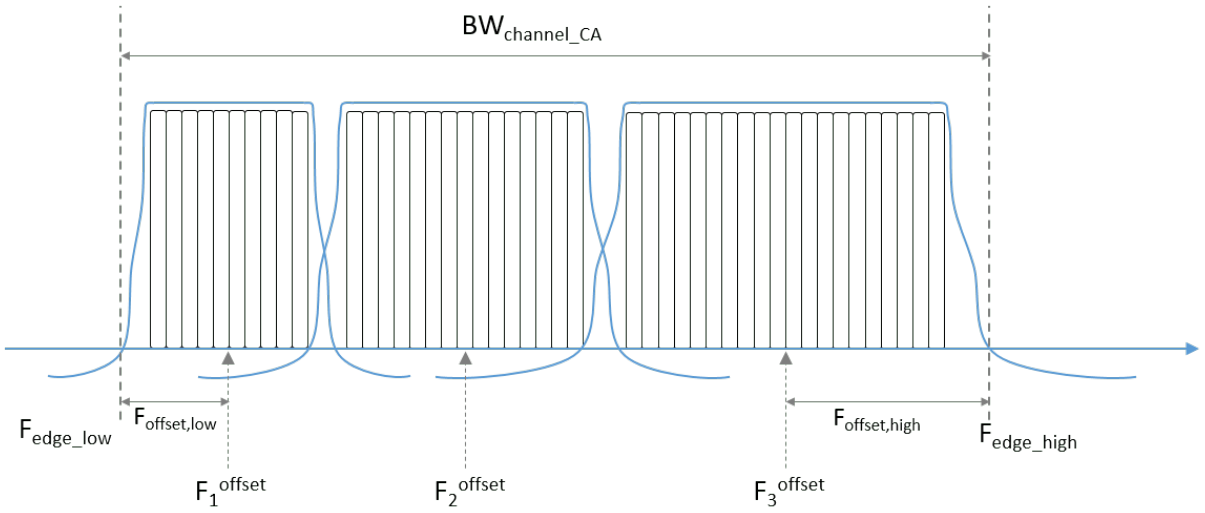
Channel Estimator Configuration

The channel estimator at the receiver end is parameterized using the structure `cec` defined below.

```
cec = struct;                           % Channel estimation config structure
cec.PilotAverage = 'UserDefined';       % Type of pilot symbol averaging
cec.FreqWindow = 15;                    % Frequency window size
cec.TimeWindow = 15;                    % Time window size
cec.InterpType = 'Cubic';               % 2D interpolation type
cec.InterpWindow = 'Centered';          % Interpolation window type
cec.InterpWinSize = 1;                  % Interpolation window size
cec.Reference = 'Antennas';             % Specifies point of reference for
                                        % channel estimation.
```

Carrier Aggregation Parameters Calculation

To perform carrier aggregation the frequency parameters described in TS 36.101, Sections 5.6 and 5.7 [1] are calculated. These parameters are summarized in the following figure.



This results in three variables:

- `F_c` is a vector containing the center frequency of each CC
- `ccSpacing` contains the spacing between CCs in MHz
- `BW_channel_CA` is the aggregated channel bandwidth of all CCs

In the code below we first calculate the value for all the CCs assuming the lower one is centered at baseband (0 Hz). Once the `BW_channel_CA` is calculated all the values are shifted so that the center of the aggregated bandwidth is located at baseband (0 Hz).

```
% Define Delta_f1 parameter in MHz for nominal guard band and frequency
% offset calculations. In the uplink Delta_f1 is zero
% (TS 36.101, Table 5.6A-1)
Delta_f1 = 0; % in MHz
maxBW = max(CCBWs);

% Initialize center frequency and CC spacing vectors
F_c = zeros(1,numCC); % Center frequencies
ccSpacing = zeros(1,numCC-1); % CC spacing

% Calculate nominal guard band, TS 36.101 5.6A-1
nominalGuardBand = 0.05*maxBW-0.5*Delta_f1;

% Initially assume lower carrier frequency is at baseband
F_c(1) = 0;

% Calculate CC spacing and center frequencies
for k = 2:numCC
    ccSpacing(k-1) = hCarrierAggregationChannelSpacing( ...
        frc{k-1}.Bandwidth, frc{k}.Bandwidth);
    F_c(k) = F_c(k-1) + ccSpacing(k-1);
end

% Calculate lower and higher frequency offsets, TS 36.101 5.6A
F_offset_low = (0.18*NULRB(1)+Delta_f1)/2 + nominalGuardBand;
```

```
F_offset_high = (0.18*NULRB(end)+Delta_f1)/2 + nominalGuardBand;

% Calculate lower and higher frequency edges, TS 36.101 5.6A
F_edge_low = F_c(1) - F_offset_low;
F_edge_high = F_c(end) + F_offset_high;

% Calculate aggregated channel bandwidth, TS 36.101 5.6A
BW_channel_CA = F_edge_high - F_edge_low;
fprintf('BW_channel_CA: %0.4f MHz\n',BW_channel_CA);

% Calculate shift to center baseband
shiftToCenter = -1*(BW_channel_CA/2 + F_edge_low);

% Shift the center frequencies so that the aggregated bandwidth is centered
% at baseband
F_c = F_c + shiftToCenter;
F_edge_low = F_c(1) - F_offset_low;
F_edge_high = F_c(end) + F_offset_high;

% Display frequency band edges
fprintf('F_edge_low:  %0.4f MHz\n',F_edge_low);
fprintf('F_edge_high: %0.4f MHz\n',F_edge_high);
fprintf('F_offset_low:  %0.4f MHz\n',F_offset_low);
fprintf('F_offset_high: %0.4f MHz\n',F_offset_high);

% Display carrier frequencies
fprintf('\n');
for i = 1:numCC
    fprintf('Component Carrier %d:\n',i);
    fprintf('Fc: %0.4f MHz\n', F_c(i));
end

BW_channel_CA: 44.6000 MHz
F_edge_low:  -22.3000 MHz
F_edge_high: 22.3000 MHz
F_offset_low:  5.5000 MHz
F_offset_high: 10.0000 MHz

Component Carrier 1:
Fc: -16.8000 MHz
Component Carrier 2:
Fc: -4.8000 MHz
Component Carrier 3:
Fc: 12.3000 MHz
```

Required Oversampling Rate Calculation

The required oversampling factors for each component carrier OSRs are calculated for a common sampling rate for the aggregated signal.

```
% Considering the bandwidth utilization as 85%
bwfraction = 0.85;

% Calculate sampling rates of the component carriers
CCSR = zeros(1,numCC);
for i = 1:numCC
    info = lteSCFDMAInfo(frc{i});
    CCSR(i) = info.SamplingRate;
```

```
end

% Calculate overall sampling rate for the aggregated signal

% Calculate the oversampling ratio (for the largest BW CC) to make sure the
% signal occupies 85% (bwfraction) of the aggregated channel bandwidth
OSR = (BW_channel_CA/bwfraction)/(max(CCSR)/1e6);
% To simplify the resampling operation, choose an oversampling ratio as the
% smallest power of 2 greater than or equal to OSR
OSR = 2^ceil(log2(OSR));
SR = OSR*max(CCSR);
fprintf('\nOutput sample rate: %0.4f Ms/s\n\n',SR/1e6);

% Calculate individual oversampling factors for the component carriers
OSRs = SR./CCSR;
```

```
Output sample rate: 61.4400 Ms/s
```

Waveform Generation and Carrier Aggregation

Generate the waveform for each CC using `lteRMCULTool`, then resample them to a common sampling rate. Finally, frequency modulate each CC to the appropriate center frequency and add them together to form the aggregated signal using Multiband Combiner

```
% Generate component carriers
tx = cell(1,numCC);
for i = 1:numCC
    tx{i} = lteRMCULTool(frc{i},randi([0 1],1000,1));
    tx{i} = resample(tx{i},OSRs(i),1)/OSRs(i);
end

% Aggregate all CC. comm.MultibandCombiner applies the frequency offsets
% and adds the resulting signals together.
sigAggregator = comm.MultibandCombiner(InputSampleRate = SR, ...
    FrequencyOffsets = F_c*1e6, ...
    OutputSampleRateSource = 'Property', ...
    OutputSampleRate = SR);

waveform = sigAggregator([tx{:}]);
```

Plot Carrier Aggregation Waveform Spectrum

Display the power spectrum of the carrier aggregated signal. In the spectrum the individual carrier bandwidths are visible. Note that the center of the aggregated bandwidth is located at baseband (0 Hz) as the signal is not modulated to RF in this example.

```
specPlot = spectrumAnalyzer('SampleRate',SR,'Title','Carrier Aggregated Signal Power Spectrum');
specPlot(waveform);
```

Demodulation and Filtering of CC of Interest

This section shows how to demodulate, filter, and downsample one of the CC of interest. The steps followed are:

- Demodulate the CC of interest and bring it to baseband (0 Hz).
- Filter out neighboring CCs and downsample. A suitable filter is designed to remove the unwanted neighboring CCs in the downsampling process. The filter design will have an impact on the quality and EVM of the recovered signal. Filters may need to be tweaked for different values of bandwidth and CC to demodulate.

We start by selecting the CC to demodulate and designing the appropriate downsampling filter with the calculated passband and stopband frequencies.

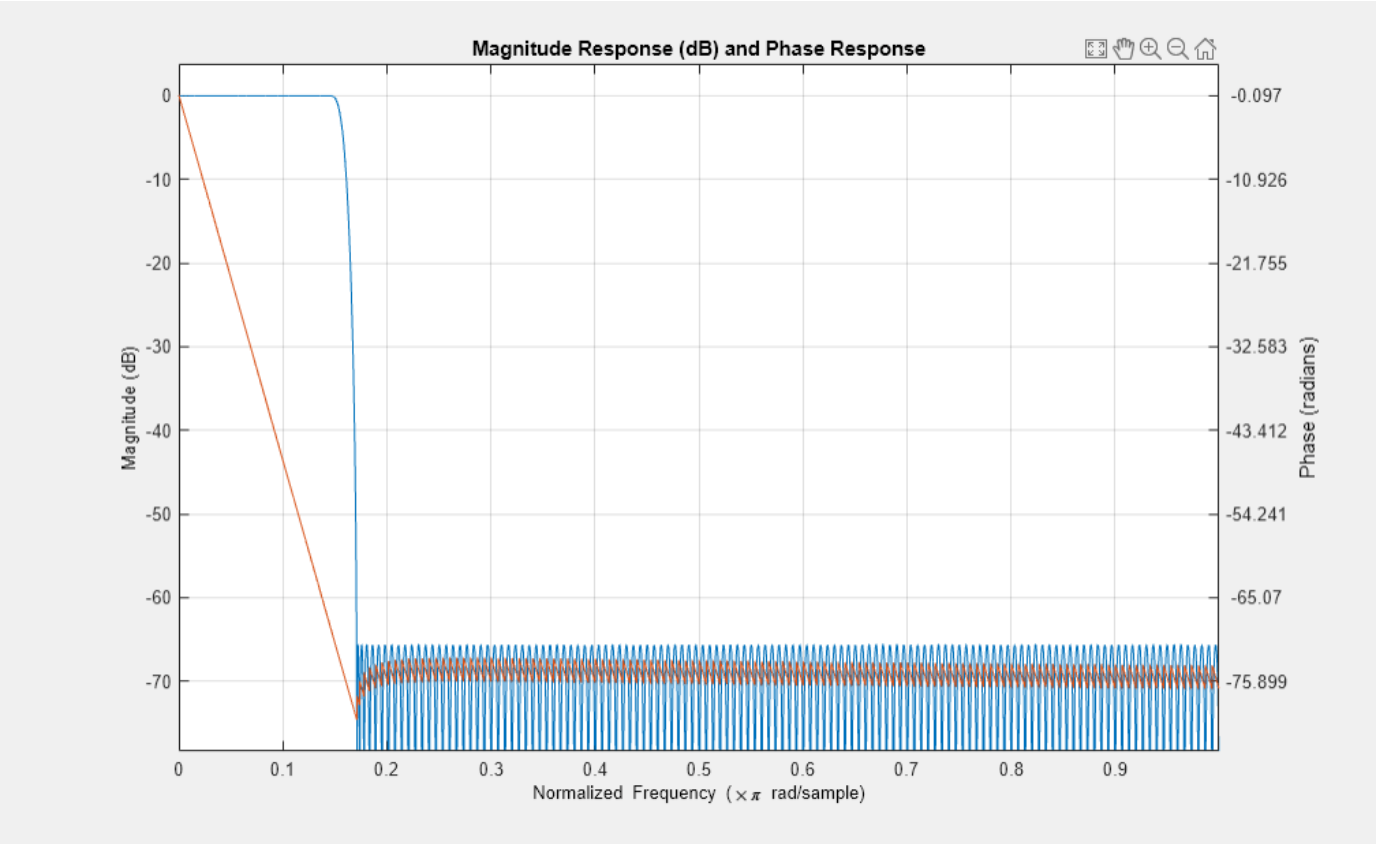
```
% Select CC to demodulate
CCofInterest = 1;
if CCofInterest > numCC || CCofInterest <= 0
    error('Cannot demodulate CC number %d, there are %d CCs\n',...
        CCofInterest, numCC) ;
end

% Define downsampling filter order
filterOrder = 301;

% Precalculate passband and stopband values for all CC
firPassbandVec = (0.18*NULRB+Delta_f1)/2 / (SR/1e6/2);
```

```
firStopbandVec = hCarrierAggregationStopband(ccSpacing,NULRB,SR);

% Define passband and stopband for carrier of interest
firPassband = firPassbandVec(CCofInterest);
firStopband = firStopbandVec(CCofInterest);

% Extract and decode CC of interest
fprintf(1,'Extracting CC number %d: \n', CCofInterest);

% Pad signal with zeros to take into account filter transient response
% length
waveform = [waveform; zeros(filterOrder+1,size(waveform,2))];

% Demodulate carrier of interest
demodulatedCC = ...
    hCarrierAggregationModulate(waveform,SR,-F_c(CCofInterest)*1e6);

% Downsampling is done in two stages if the filter is too narrow. This
% eases the filter design requirements. If this is the case an initial
% downsampling factor of 4 is applied. You may want to consider a different
% filter design in this initial stage if the quality of the resulting
% signal is not acceptable.
if (firStopband < 0.1)
    % Down-sample by 4
    SRC = 4;
    demodulatedCC = resample(demodulatedCC,1,SRC);
    % Update pass and stopband to take first downsampling into account
    firPassband = firPassband * SRC;
    firStopband = firStopband * SRC;
else
    % No pre-filter
    SRC = 1;
end

% Design lowpass filter to filter out CC of interest
frEdges = [0 firPassband firStopband 1];
firFilter = dsp.FIRFilter;
firFilter.Numerator = firpm(filterOrder,frEdges,[1 1 0 0]);
```

```
Extracting CC number 1:
```

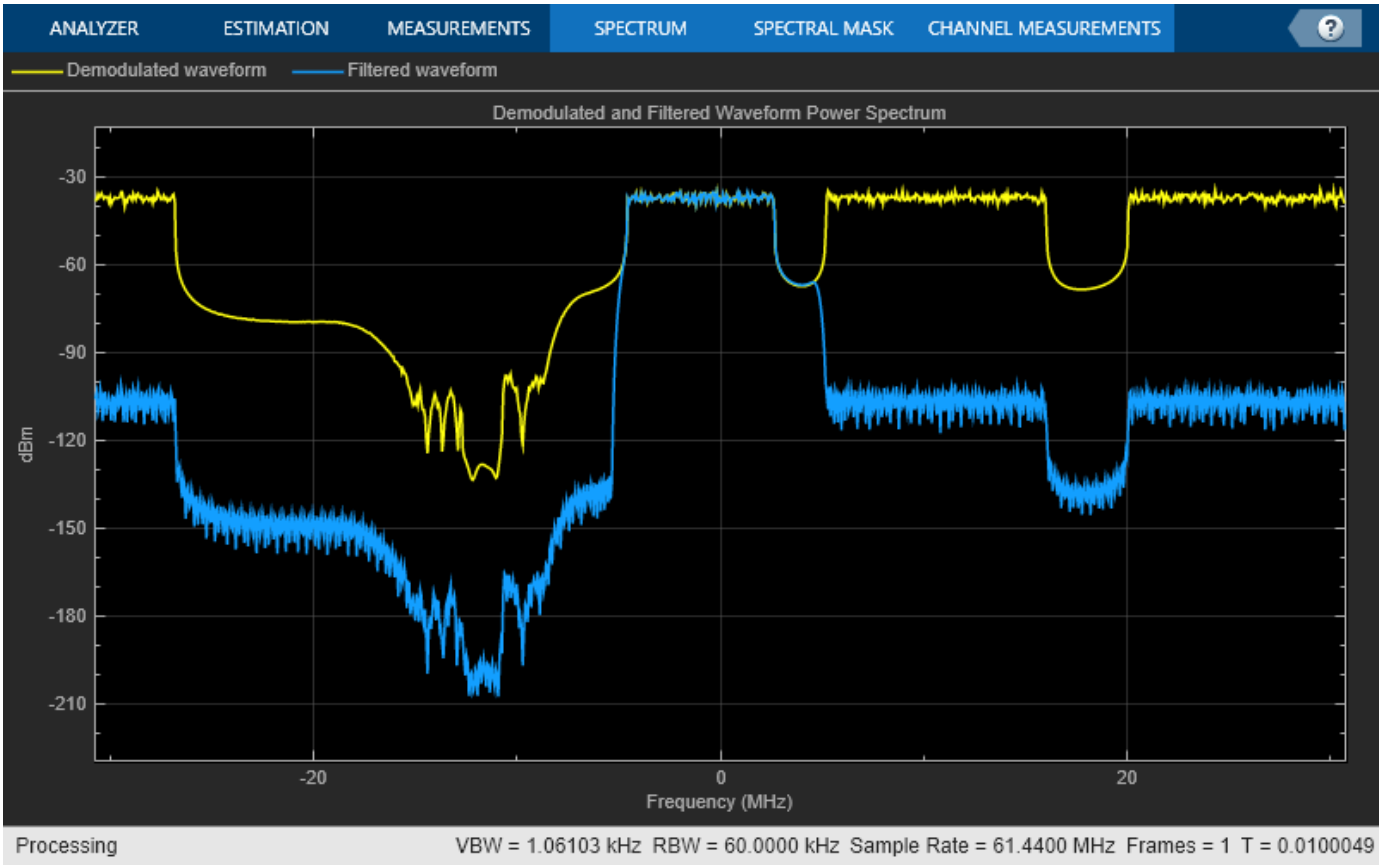
The response of the designed filter is displayed.

```
fvtool(firFilter,'Analysis','freq');
```



The demodulated waveform is then filtered to extract the CC of interest. The spectrum of the demodulated waveform before and after the filtering is plotted.

```
% Filter the signal to extract the component of interest
rxWaveform  = firFilter(demodulatedCC);

% Plot the demodulated and filtered signal spectra
filteredSpecPlot = spectrumAnalyzer('SampleRate',SR,'Title','Demodulated and Filtered Waveform Po
                                    'ChannelNames',{'Demodulated waveform' 'Filtered waveform'},
filteredSpecPlot([demodulatedCC, rxWaveform]);
```



At this point the filtered waveform can be downsampled to its baseband rate.

```
rxWaveform = downsample(rxWaveform,OSRs(CCofInterest)/SRC);
```

Synchronization

Synchronization is applied to the resulting signal before EVM and in-band emissions measurement.

```
% Get the parameters for CC of interest
FRC = frc{CCofInterest};
PUSCH = FRC.PUSCH;

% Synchronize received waveform
offset = lteULFrameOffset(FRC,PUSCH,rxWaveform);
rxWaveform = rxWaveform(1+offset:end, :);
```

EVM, In-Band Emissions Measurements, and PUSCH Decoding

The code below provides PUSCH EVM, PUSCH DRS EVM, and in-band emissions calculation using `hPUSCHEVM`.

The EVM is the difference between the reconstructed ideal symbols and measured received symbols over one slot measurement interval in the time domain. The process of reconstructing the ideal symbols includes:

- Single-carrier frequency division multiple access (SC-FDMA) demodulation to obtain the received resource grid

- Channel estimation
- PUSCH equalization
- Symbol demodulation
- Decoding followed by reencoding of the received bits, rescrambling and remodulation

The average EVM is measured at two locations in time (low and high, which are denoted as $\overline{EVM_l}$ and $\overline{EVM_h}$), where the low and high locations correspond to the alignment of the FFT window within the start and end of the cyclic prefix. LTE Toolbox requires the low and high locations to be specified as a fraction of the cyclic prefix length.

The in-band emissions are a measure of the interference falling into the non-allocated RBs. In-band emissions are calculated for each Δ_{RB} and slot number, where Δ_{RB} is the starting frequency offset between the allocated RB and the measured non-allocated RB (e.g., $\Delta_{RB} = 1$ for the first adjacent RB outside of the allocated bandwidth).

From the above measurements the following plots are also produced:

- EVM versus OFDM symbol
- EVM versus subcarrier
- EVM versus resource block
- EVM versus OFDM symbol and subcarrier (i.e. the EVM resource grid)
- Uplink in-band emissions for non-allocated RBs

Note that the EVM measurements displayed at the command window are only calculated across allocated PUSCH resource blocks, in accordance with the LTE standard. The EVM plots are shown across all resource blocks (allocated or unallocated), allowing the in-band emissions to be viewed. In unallocated resource blocks, the EVM is calculated assuming that the received resource elements have an expected value of zero.

The PUSCH of the recovered signal is then decoded and the resulting CRC is checked for errors.

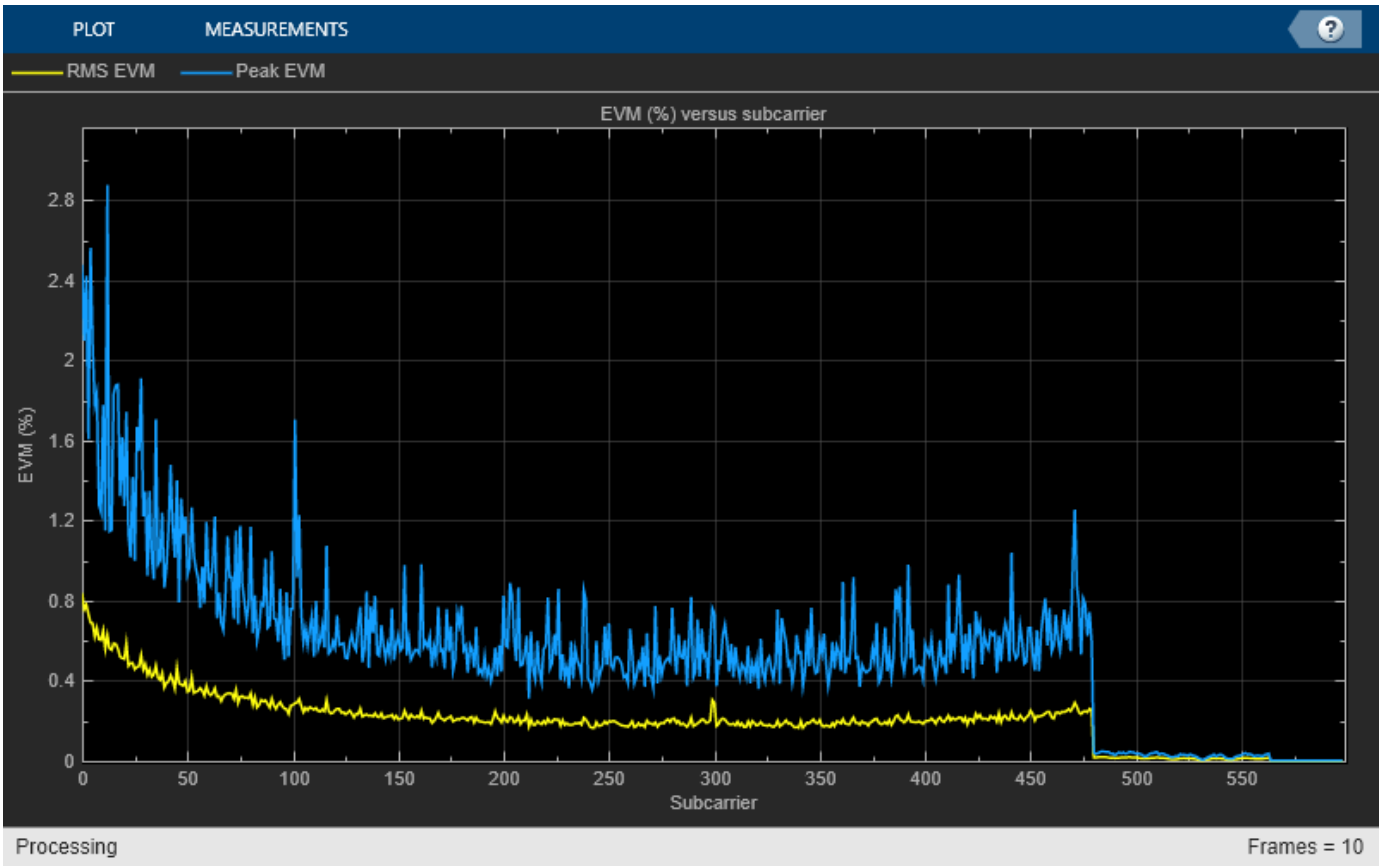
```
% Configure channel estimation structure for EVM measurement
cecEVM = cec;
cecEVM.PilotAverage = 'TestEVM';
[evmpusch, evmdrs, emissions, plots] = hPUSCHEVM(frc{CCofInterest},cecEVM,rxWaveform);

% Plot the absolute in-band emissions
if (~isempty(emissions.DeltaRB))
    hPUSCHEVMEmissionsPlot(emissions);
end

% Perform SC-FDMA demodulation
rxGrid = lteSCFDMADemodulate(FRC,rxWaveform);

% Get the number of received subframes and OFDM symbols per subframe
dims = lteSCFDMAInfo(FRC);
samplesPerSubframe = dims.SamplingRate/1000;
nRxSubframes = floor(size(rxWaveform, 1)/samplesPerSubframe);
FRC.TotSubframes = 1;
resGridSize = lteResourceGridSize(FRC);
L = resGridSize(2);
```

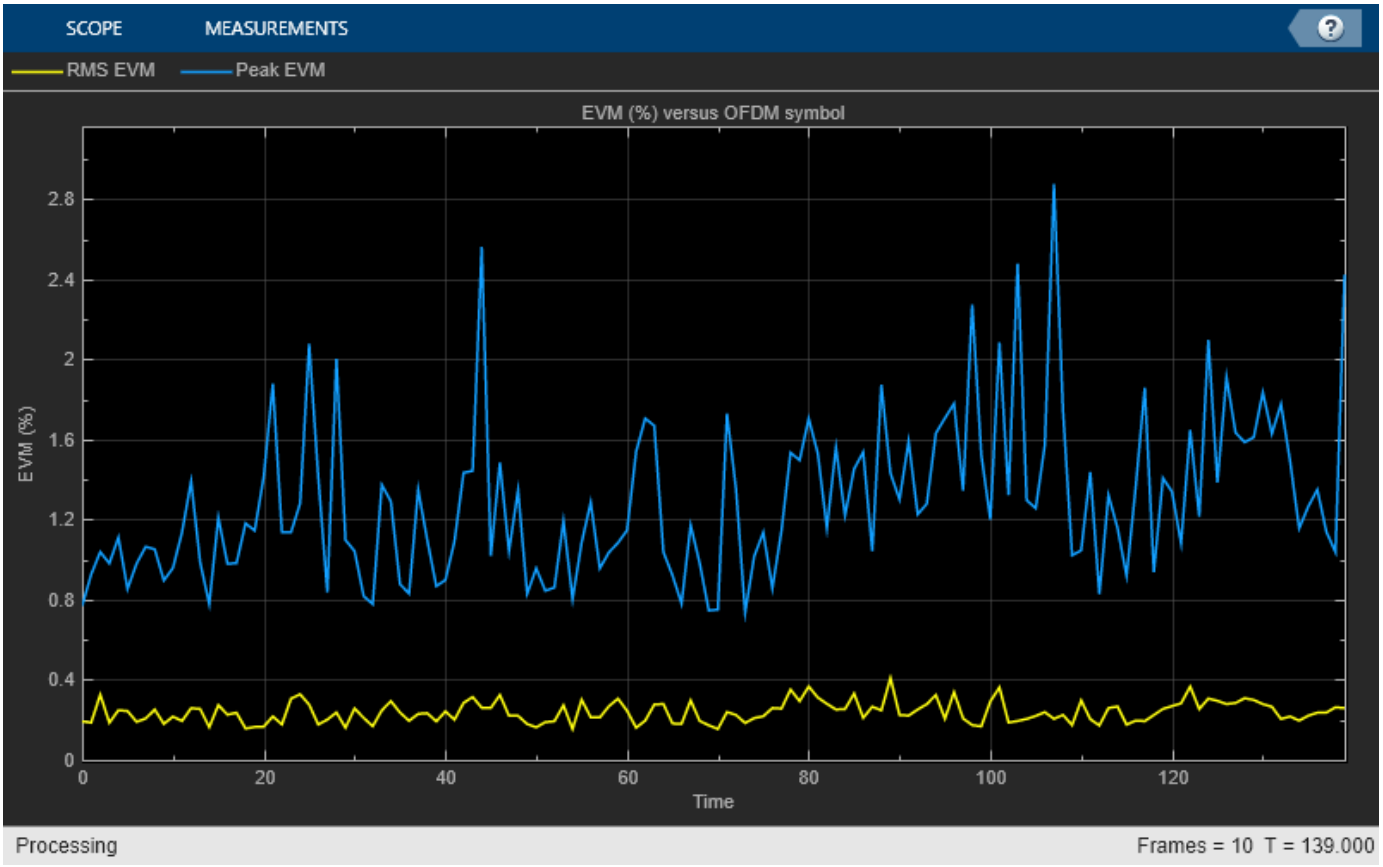
```
disp('Decode transmitted subframes and check CRC.');

for n=0:nRxSubframes-1

    % Extract subframe
    rxSubframe = rxGrid(:,(1:L)+(n*L),:);

    % Get the transport block size for current subframe
    FRC.NSubframe = n;
    trBlkSize = PUSCH.TrBlkSizes(n+1);

    % Perform channel estimation
    [estChannelGrid, noiseEst] = lteULChannelEstimate( ...
        FRC, PUSCH, cec, rxSubframe);

    % Generate PUSCH indices and extract symbols from received and
    % channel estimate grids in preparation for PUSCH decoding
    ind = ltePUSCHIndices(FRC, PUSCH);
    [rxSym,hestSym] = lteExtractResources(ind, rxSubframe, estChannelGrid);

    % Update PUSCH to carry complete information of the UL-SCH coding
    PUSCH = lteULSCHInfo(FRC, PUSCH, trBlkSize, 'chsconcat');

    % Perform equalization, transform deprecoding, layer demapping,
    % demodulation, and descrambling on the received data using the channel
    % estimate
    [rxEncodedBits, puschsymbols] = ltePUSCHDecode(FRC, PUSCH, ...
        rxSym, hestSym, noiseEst);

    % Decode Uplink Shared Channel (UL-SCH)
    decState = [];
    [decbits,crc] = ...
        lteULSCHDecode(FRC, PUSCH, trBlkSize ,rxEncodedBits, decState);
    if crc
        disp(['Subframe ' num2str(n) ': CRC failed']);
    else
        disp(['Subframe ' num2str(n) ': CRC passed']);
    end
end
```

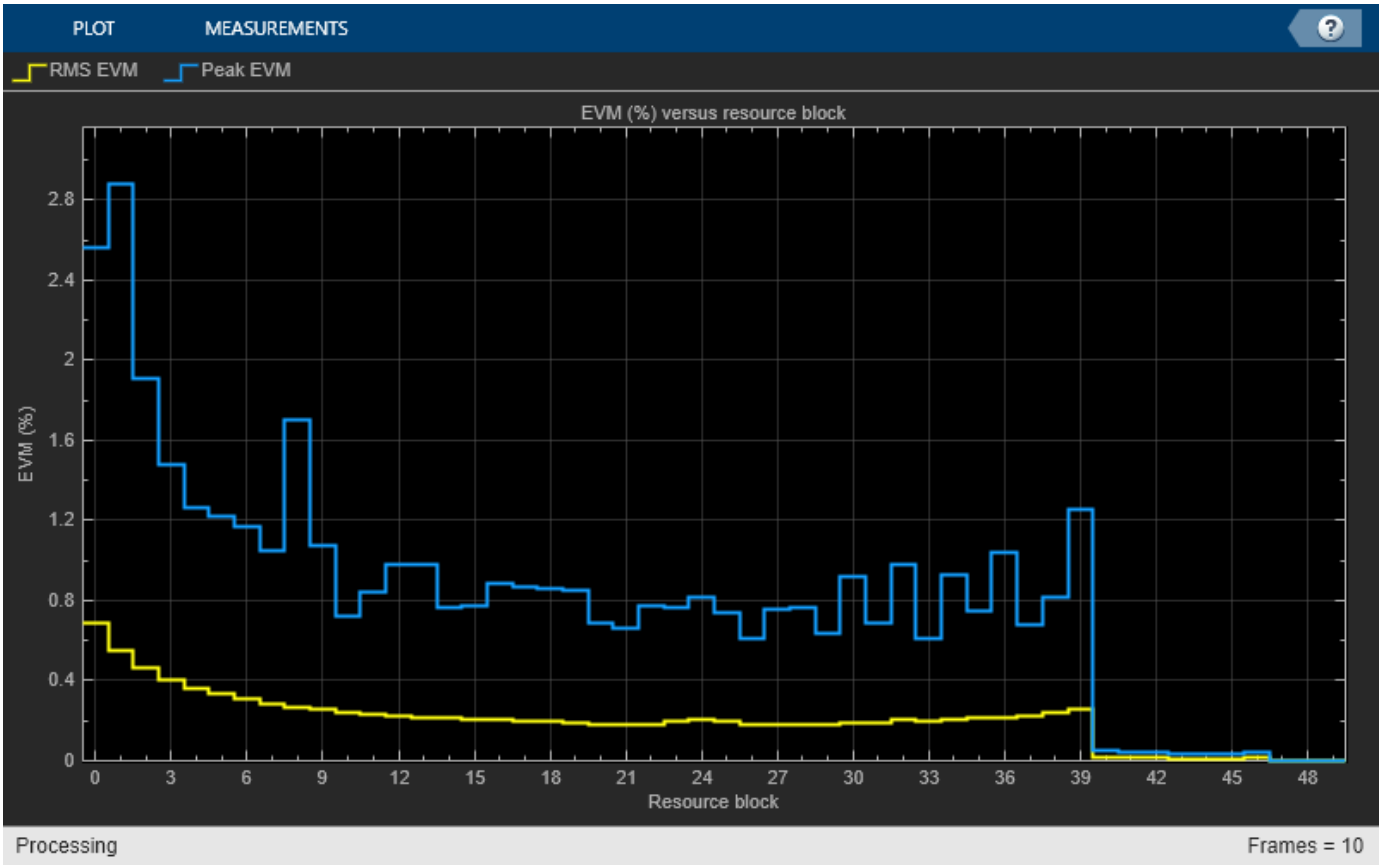
```
Low edge PUSCH EVM, slot 0: 0.165%
Low edge PUSCH EVM, slot 1: 0.244%
Low edge DRS EVM, slot 0: 0.210%
Low edge DRS EVM, slot 1: 0.199%
High edge PUSCH EVM, slot 0: 0.264%
High edge PUSCH EVM, slot 1: 0.253%
High edge DRS EVM, slot 0: 0.208%
High edge DRS EVM, slot 1: 0.242%
Low edge PUSCH EVM, slot 2: 0.213%
Low edge PUSCH EVM, slot 3: 0.175%
Low edge DRS EVM, slot 2: 0.186%
Low edge DRS EVM, slot 3: 0.307%
High edge PUSCH EVM, slot 2: 0.220%
High edge PUSCH EVM, slot 3: 0.258%
High edge DRS EVM, slot 2: 0.264%
High edge DRS EVM, slot 3: 0.367%
Low edge PUSCH EVM, slot 4: 0.225%
Low edge PUSCH EVM, slot 5: 0.175%
```

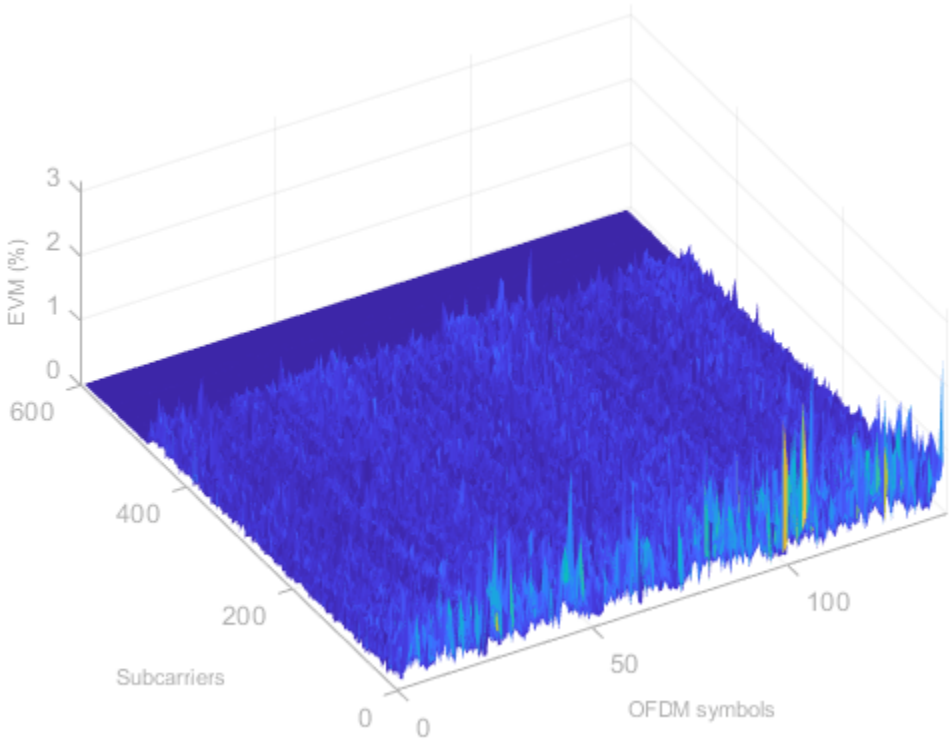
```
Low edge DRS EVM, slot 4: 0.180%
Low edge DRS EVM, slot 5: 0.236%
High edge PUSCH EVM, slot 4: 0.260%
High edge PUSCH EVM, slot 5: 0.243%
High edge DRS EVM, slot 4: 0.234%
High edge DRS EVM, slot 5: 0.261%
Low edge PUSCH EVM, slot 6: 0.228%
Low edge PUSCH EVM, slot 7: 0.194%
Low edge DRS EVM, slot 6: 0.294%
Low edge DRS EVM, slot 7: 0.149%
High edge PUSCH EVM, slot 6: 0.306%
High edge PUSCH EVM, slot 7: 0.243%
High edge DRS EVM, slot 6: 0.290%
High edge DRS EVM, slot 7: 0.216%
Low edge PUSCH EVM, slot 8: 0.205%
Low edge PUSCH EVM, slot 9: 0.203%
Low edge DRS EVM, slot 8: 0.269%
Low edge DRS EVM, slot 9: 0.164%
High edge PUSCH EVM, slot 8: 0.244%
High edge PUSCH EVM, slot 9: 0.268%
High edge DRS EVM, slot 8: 0.342%
High edge DRS EVM, slot 9: 0.201%
Low edge PUSCH EVM, slot 10: 0.184%
Low edge PUSCH EVM, slot 11: 0.249%
Low edge DRS EVM, slot 10: 0.211%
Low edge DRS EVM, slot 11: 0.203%
High edge PUSCH EVM, slot 10: 0.246%
High edge PUSCH EVM, slot 11: 0.327%
High edge DRS EVM, slot 10: 0.206%
High edge DRS EVM, slot 11: 0.410%
Low edge PUSCH EVM, slot 12: 0.221%
Low edge PUSCH EVM, slot 13: 0.273%
Low edge DRS EVM, slot 12: 0.170%
Low edge DRS EVM, slot 13: 0.255%
High edge PUSCH EVM, slot 12: 0.322%
High edge PUSCH EVM, slot 13: 0.285%
High edge DRS EVM, slot 12: 0.297%
High edge DRS EVM, slot 13: 0.362%
Low edge PUSCH EVM, slot 14: 0.215%
Low edge PUSCH EVM, slot 15: 0.214%
Low edge DRS EVM, slot 14: 0.274%
Low edge DRS EVM, slot 15: 0.183%
High edge PUSCH EVM, slot 14: 0.232%
High edge PUSCH EVM, slot 15: 0.253%
High edge DRS EVM, slot 14: 0.405%
High edge DRS EVM, slot 15: 0.251%
Low edge PUSCH EVM, slot 16: 0.224%
Low edge PUSCH EVM, slot 17: 0.240%
Low edge DRS EVM, slot 16: 0.278%
Low edge DRS EVM, slot 17: 0.260%
High edge PUSCH EVM, slot 16: 0.248%
High edge PUSCH EVM, slot 17: 0.310%
High edge DRS EVM, slot 16: 0.197%
High edge DRS EVM, slot 17: 0.409%
Low edge PUSCH EVM, slot 18: 0.290%
Low edge PUSCH EVM, slot 19: 0.214%
Low edge DRS EVM, slot 18: 0.279%
Low edge DRS EVM, slot 19: 0.190%
```

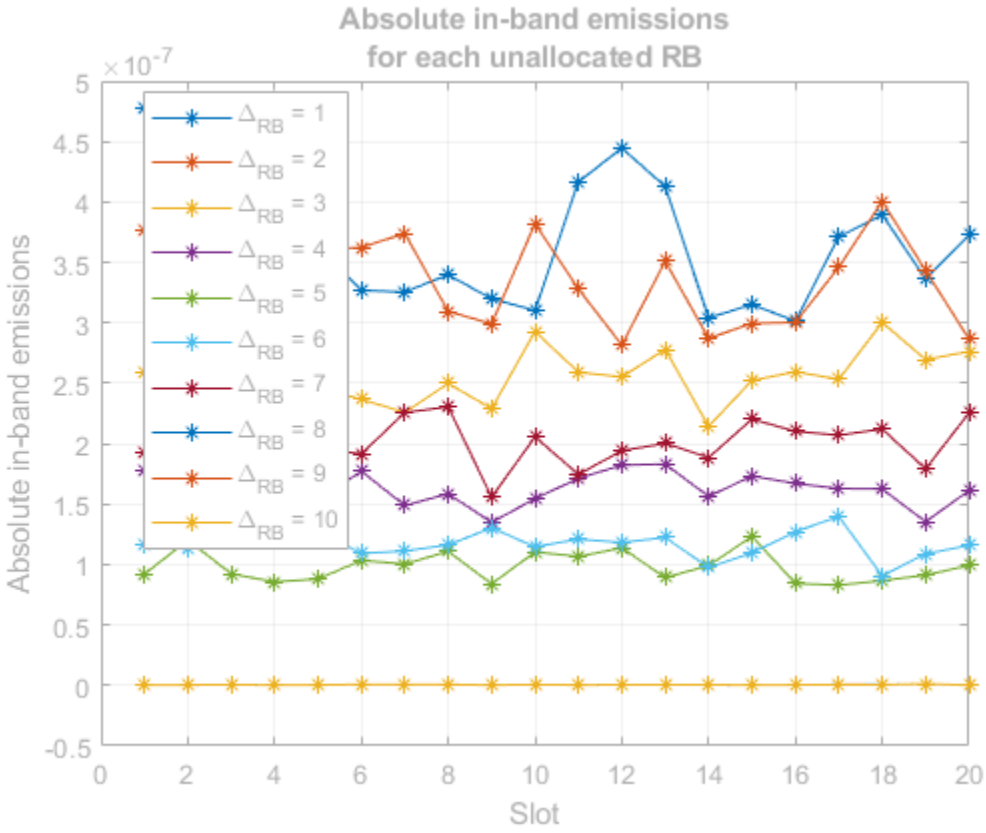
```
High edge PUSCH EVM, slot 18: 0.304%
High edge PUSCH EVM, slot 19: 0.261%
High edge DRS EVM, slot 18: 0.335%
High edge DRS EVM, slot 19: 0.264%
Averaged low edge PUSCH EVM, frame 0: 0.220%
Averaged high edge PUSCH EVM, frame 0: 0.269%
Averaged PUSCH EVM frame 0: 0.269%
Averaged DRS EVM frame 0: 0.297%
Averaged overall PUSCH EVM: 0.269%
Averaged overall DRS EVM: 0.297%
Decode transmitted subframes and check CRC.
Subframe 0: CRC passed
Subframe 1: CRC passed
Subframe 2: CRC passed
Subframe 3: CRC passed
Subframe 4: CRC passed
Subframe 5: CRC passed
Subframe 6: CRC passed
Subframe 7: CRC passed
Subframe 8: CRC passed
Subframe 9: CRC passed
```




SCOPE
MEASUREMENTS
RMS EVM
Peak EVM
EVM (%) versus OFDM symbol
EVM (%)
2.8
2.4
2
1.6
1.2
0.8
0.4
0
0
20
40
60
80
100
120
Time
Processing
Frames = 10 T = 139.000


PLOT
MEASUREMENTS
RMS EVM
Peak EVM
EVM (%) versus resource block
EVM (%)
2.8
2.4
2
1.6
1.2
0.8
0.4
0
0
3
6
9
12
15
18
21
24
27
30
33
36
39
42
45
48
Resource block
Processing
Frames = 10


EVM (%)
3
2
1
0
600
400
200
0
Subcarriers
0
50
100
OFDM symbols



Selected Bibliography

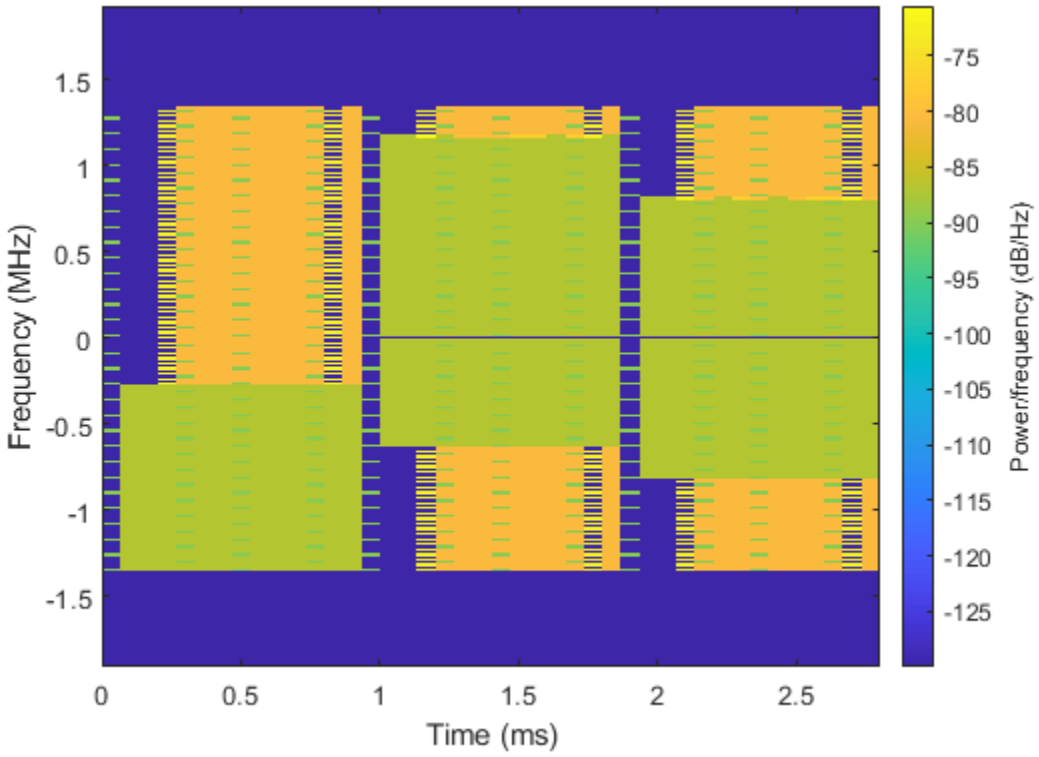
1 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception"

Dynamic Spectrum Sharing for 5G NR and LTE Coexistence

This example shows how to generate a waveform containing LTE and 5G NR waveforms for dynamic spectrum sharing (DSS) by using LTE Toolbox™ and 5G Toolbox™.

Introduction

Dynamic spectrum sharing allows 5G and LTE to share resources dynamically in the same spectrum. Therefore, 5G can use the existing LTE spectrum without the need for re-farming existing bands. 5G and LTE coexistence also maximizes system efficiency because the LTE and 5G base stations, eNodeB and gNodeB, respectively, can use all available resources at all times. This figure shows an example in which the gNodeB scheduler allocates 5G resources dynamically every millisecond, filling all the resources unused by LTE. Specifically, the figure shows the spectrogram of the combined LTE and 5G waveform, in which the LTE waveform is in shades of green and the 5G waveform is in shades of yellow.



The main requirement of DSS is backward compatibility. That is, LTE devices must function as normal regardless of whether DSS is implemented. Specifically, 5G waveforms in DSS must not interfere with LTE always-on signals and channels, such as the cell reference signal (CRS), primary synchronization signal (PSS), secondary synchronization signal (SSS), and physical broadcast channel (PBCH). The 5G and LTE scheduling of resources can change dynamically every subframe, which is the lowest common time unit between the two technologies.

This example shows how to generate a waveform containing:

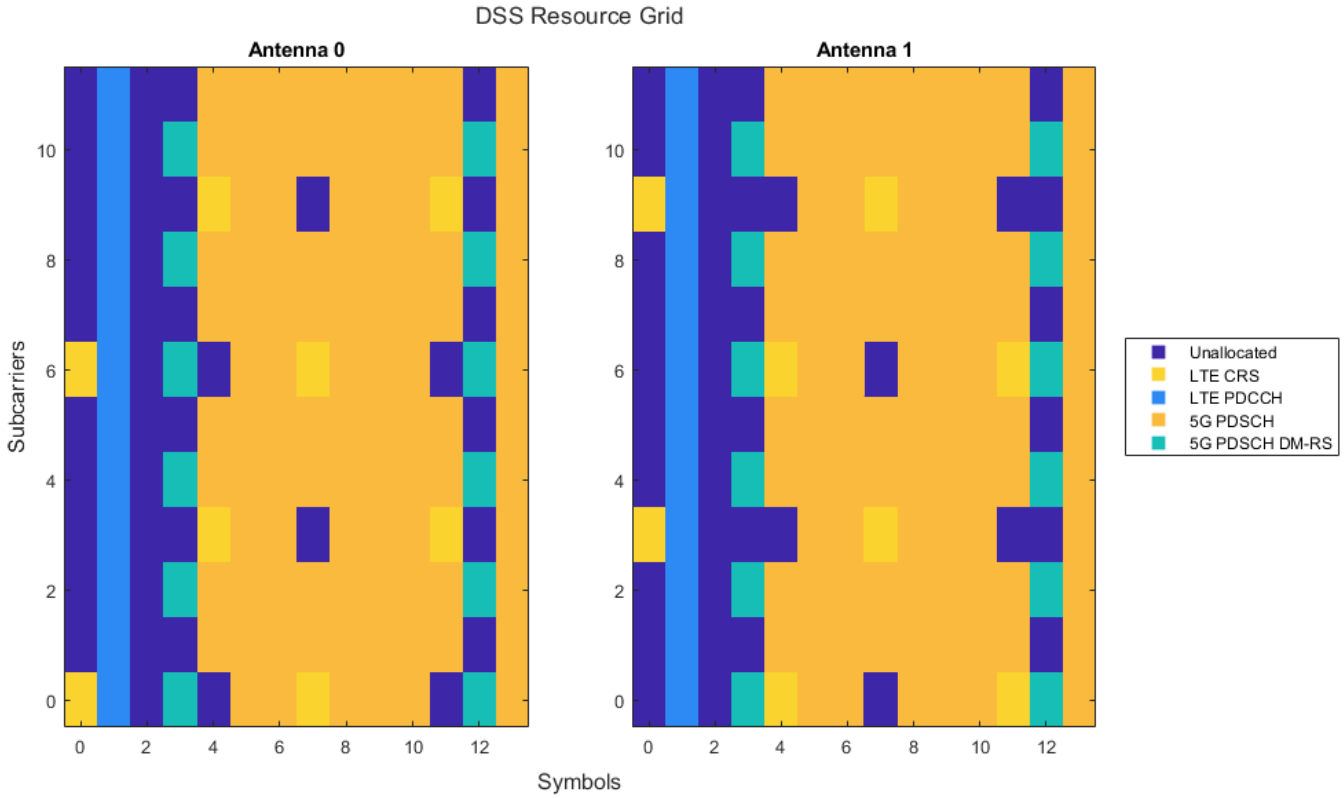
- 5G physical downlink shared channel (PDSCH)

- LTE CRS
- LTE PSS
- LTE SSS
- LTE PBCH
- LTE physical downlink control channel (PDCCH)
- LTE physical control format indicator channel (PCFICH)
- LTE physical hybrid ARQ indicator channel (PHICH)

The 5G PDSCH occupies all resources unused by the LTE waveform. The example implements these techniques to enable DSS.

- Perform rate-matching of the 5G PDSCH around the resource elements (REs) used by the LTE CRS. The higher layer parameter `RateMatchPatternLTE-CRS`, defined in TS 38.331 Section 6.3.2, contains the information needed by the gNodeB to rate-match the PDSCH around the CRS.
- Reserve 5G PDSCH physical resource blocks (PRBs) that are occupied by the LTE PBCH, PSS, and SSS.
- Use Rel-16 alternative locations for the 5G PDSCH demodulation reference signal (DM-RS), as described in TS 38.211 Section 7.4.1.1.2, for a 5G PDSCH configuration with: mapping type A, the first DM-RS allocated in symbol 3, and an additional DM-RS.
- Allocate the 5G PDSCH starting from symbol 3 of each slot to ensure that the 5G PDSCH never collides with LTE control information channels.

This figure shows an example resource grid for a single PRB and single subframe containing the LTE CRS, LTE PDCCH, and 5G PDSCH with its DM-RS. The LTE configuration consists of two CRS ports and the PDCCH scheduled in symbol 1. The 5G PDSCH configuration consists of two layers, mapping type A, allocated symbols from 3 to 13, a DM-RS in symbol 3, and an additional DM-RS. The figure shows the 5G PDSCH rate-matching around the REs allocated for the LTE CRS and the additional DM-RS allocated in symbol 12, instead of 11, to avoid collisions with the LTE CRS in antenna 1.



Waveform Configuration

Configure the LTE and 5G waveforms. You can change the length of the generated waveform, the bandwidth of 5G and LTE carriers in terms of number of PRBs, the number of ports for the LTE CRS, the shifting value for assigning the LTE cell identity, and the LTE carrier offset to NR point A. Set the LTE carrier offset to NR point A in units of 15 kHz subcarriers, as discussed in TS 38.214 Section 5.1.4.2.

```
numSubframes = 10;          % Number of subframes
gnbNumPRB = 52;             % Size of 5G carrier in number of PRBs (1...275)
enbNumPRB = 25;             % Size of LTE carrier in number of PRBs
enbNumCRSPorts = 2;         % Number of LTE CRS ports
enbVShift = 0;              % LTE shifting value v-Shift
enbCarrierFrequency = 0;    % LTE carrier offset to NR point A, in units of 15
```

Initialize the random number generator to its default value for reproducibility.

```
rng("default");
```

LTE Configuration

Configure the cell-wide settings for LTE.

```
enb = getLTEConfig(enbNumPRB,enbNumCRSPorts,enbVShift,enbCarrierFrequency,numSubframes);
```

5G Configuration

Configure the cell-wide settings for 5G considering a subcarrier spacing of 15 kHz. This value of the subcarrier spacing ensures that the 5G waveform has the same time-frequency OFDM structure as the LTE waveform.

```
gnb = nrCarrierConfig;
gnb.NSizeGrid = gnbNumPRB;
disp(gnb)
```

```
  nrCarrierConfig with properties:

             NCellID: 1
    SubcarrierSpacing: 15
         CyclicPrefix: 'normal'
            NSizeGrid: 52
          NStartGrid: 0
               NSlot: 0
              NFrame: 0

   Read-only properties:
       SymbolsPerSlot: 14
     SlotsPerSubframe: 1
        SlotsPerFrame: 10
```

5G PDSCH Configuration

Configure the 5G PDSCH to occupy all the resources available and unused by the LTE transmission. In particular, symbol 3 is the first symbol allocated for the PDSCH. This choice is intentionally conservative because the PCFICH, PHICH, and PDCCH in LTE can occupy up to the first three symbols in a subframe.

```
pdsch = nrPDSCHConfig;
pdsch.PRBSet = 0:gnb.NSizeGrid-1;      % Full-band allocation
pdsch.SymbolAllocation = [3 11];       % Full-slot allocation, starting from symbol 3
pdsch.MappingType = "A";               % PDSCH mapping type A
pdsch.DMRS.DMRSTypeAPosition = 3;      % First DM-RS in symbol 3 for PDSCH mapping type A
pdsch.DMRS.DMRSAdditionalPosition = 1; % Additional DM-RS position
```

For a 5G PDSCH mapping type A configured with a DM-RS on symbol 3 and an additional DM-RS, the second DM-RS collides with the LTE CRS on some REs. To avoid collision between the 5G PDSCH DM-RS and the LTE CRS, TS 38.211 Section 7.4.1.1.2 defines an alternative DM-RS location, moving the allocation of the second DM-RS from symbol 11 to symbol 12. The option of using an alternative DM-RS location is subject to UE capability, as specified in TS 38.306 Section 4.2.7.5. The example sets the alternative DM-RS location by using the `CustomSymbolSet` property of the `nrPDSCHDMRSConfig` (5G Toolbox) object.

```
if (pdsch.MappingType == "A") && ...
        (pdsch.DMRS.DMRSAdditionalPosition == 1) && ...
        (pdsch.DMRS.DMRSTypeAPosition == 3)
    pdsch.DMRS.CustomSymbolSet = [3 12];
end
```

Compute the indices occupied by the LTE CRS in the 5G numerology and assign them to the `ReservedRE` property of the `nrPDSCHConfig` (5G Toolbox) object. This property specifies the RE indices that are unavailable for the PDSCH.

```
pdsch.ReservedRE = getReservedRE(enbNumPRB,enbNumCRSPorts,enbVShift,enbCarrierFrequency,gnb);
```

Compute the PRBs occupied by the LTE PBCH and synchronization signals (PSS and SSS) in the 5G numerology and assign them to the `ReservedPRB` property of the `nrPDSCHConfig` object. This property specifies the PRB and symbol patterns that are unavailable for the PDSCH.

```
pdsch.ReservedPRB = getReservedPRB(enb,gnb);
```

Define the power scaling for each channel and signal in dB. The example chooses these values solely to ease the visualization of each channel and signal in the spectrogram view.

```
pdschPower = 15;     % PDSCH power scaling in dB
pdschDMRSPower = 18; % PDSCH DM-RS power scaling in dB
```

LTE Waveform Generation

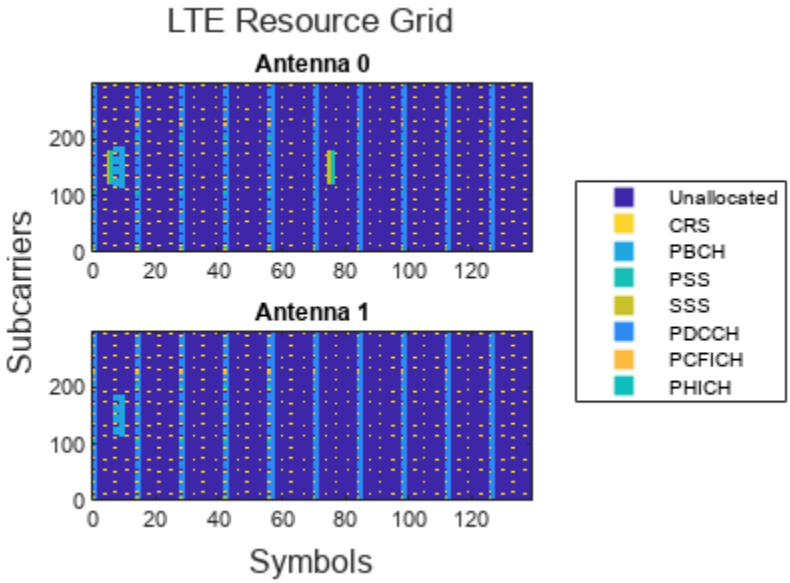
Generate the LTE waveform, resource grid, and information output. The LTE waveform contains:

- Always-on signals CRS, PSS, and SSS.
- Always-on channel PBCH.
- PCFICH, PHICH, and PDCCH.

```
[enbWave,enbGrid] = lteRMCDLTool(enb,[1;0;0;1]);
enbInfo = lteOFDMInfo(enb);
```

Plot the LTE resource grid.

```
enbChannelNames = ["CRS","PBCH","PSS","SSS","PDCCH","PCFICH","PHICH"];
plotResourceGrid(enb,enbGrid,numSubframes,enbChannelNames);
```



5G Waveform Generation

Generate the 5G waveform containing the PDSCH and its DM-RS.

Define the total number of slots.

```
numSlots = numSubframes*gnb.SlotsPerSubframe;
```

Initialize an empty resource grid.

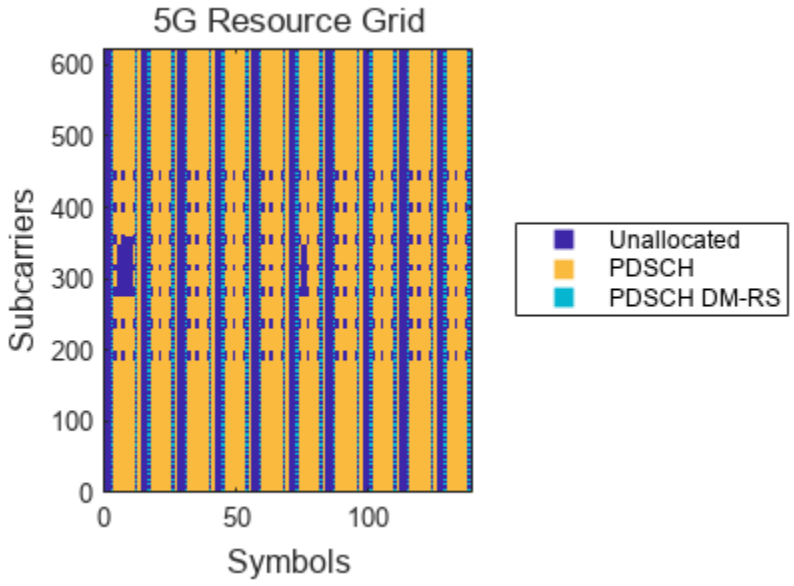
```
gnbGrid = [];
```

Loop over the total number of slots to generate the 5G resource grid.

```
for nslot = 0:numSlots-1

    % Update the carrier slot number and create an empty resource grid for
    % the current slot
    gnb.NSlot = nslot;
    gridSlot = nrResourceGrid(gnb,pdsch.NumLayers);

    % Get the PDSCH and PDSCH DM-RS indices and symbols
    [pdschIndices,pdschIndicesInfo] = nrPDSCHIndices(gnb,pdsch);
    cw = randi([0 1],pdschIndicesInfo.G,1);
    pdschSymbols = nrPDSCH(gnb,pdsch,cw) * db2mag(pdschPower);
    pdschDMRSIndices = nrPDSCHDMRSIndices(gnb,pdsch);
    pdschDMRSSymbols = nrPDSCHDMRS(gnb,pdsch) * db2mag(pdschDMRSPower);

    % Place the symbols in the grid
    gridSlot(pdschIndices) = pdschSymbols;
    gridSlot(pdschDMRSIndices) = pdschDMRSSymbols;

    % Append the resource grid for the current slot to the overall resource
    % grid
    gnbGrid = cat(2,gnbGrid,gridSlot);

end
```

Plot the 5G resource grid.

```
gnbChannelNames = ["PDSCH","PDSCH DM-RS"];
plotResourceGrid(struct("gnb",gnb,"pdsch",pdsch),gnbGrid,numSubframes,gnbChannelNames);
```



Generate the 5G waveform by performing OFDM modulation on the resource grid.

```
[gnbWave,gnbInfo] = nrOFDMModulate(gnb,gnbGrid,Windowing=0);
```

Generate Combined DSS Waveform

Shift the LTE waveform according to the value of `enbCarrierFrequency`.

```
enbfotx = comm.PhaseFrequencyOffset;
enbfotx.SampleRate = enbInfo.SamplingRate;
enbfotx.FrequencyOffset = enbCarrierFrequency*15e3; % Hz
enbWave = enbfotx(enbWave);
```

If LTE and 5G sample rates are different, resample the waveform with the lowest sample rate to the highest sample rate.

```
enbSR = enbInfo.SamplingRate;
gnbSR = gnbInfo.SampleRate;
if gnbSR >= enbSR
    enbWave = resample(enbWave,gnbSR,enbSR);
    ofdmInfo = gnbInfo;
else
    gnbWave = resample(gnbWave,enbSR,gnbSR);
    ofdmInfo = enbInfo;
end
```

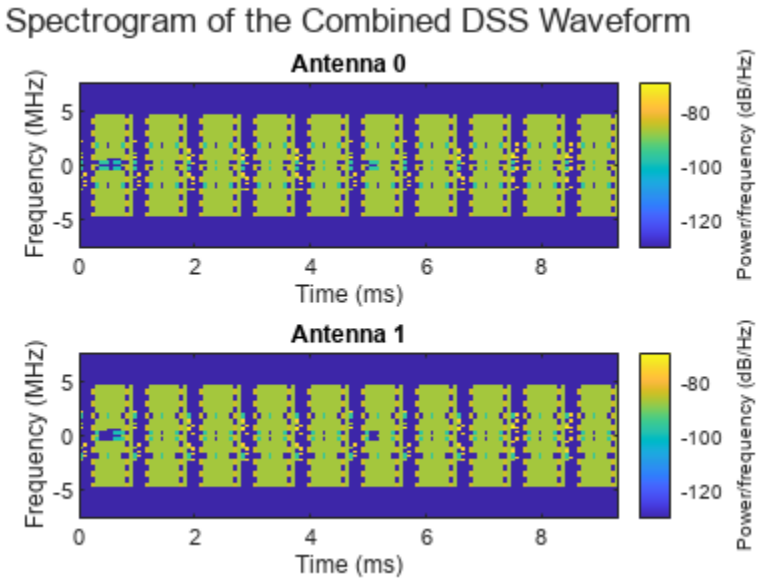
Combine the two waveforms.

```
wave = enbWave + gnbWave;
```

Spectrogram of Combined DSS Waveform

Plot the spectrogram of the combined waveform. You can see the LTE and 5G waveforms coexisting in the same spectrum.

```
plotDSSSpectrogram(wave,ofdmInfo,enbSR,gnbSR,numSubframes);
```



References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.214. “NR; Physical layer procedures for data.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 38.306. “NR; User Equipment (UE) radio access capabilities.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[4] 3GPP TS 38.331. "NR; Radio Resource Control (RRC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Local Functions

```
function enb = getLTEConfig(numPRB,numCRSPorts,vShift,carrierFrequency,numSubframes)
    % Generate the cell-wide configuration for LTE, given the number of
    % PRBs, the number of CRS ports, the shifting value v-Shift for the LTE
    % CRS, and the carrier frequency offset from point A.

    % Define the LTE configuration
    enb = struct();
    enb.NDLRB = numPRB;
    enb.CellRefP = numCRSPorts;
    enb.NCellID = vShift;
    enb.CyclicPrefix = "Normal";
    enb.CFI = 2;
    enb.Ng = "Sixth";
    enb.PHICHDuration = "Normal";
    enb.NFrame = 0;
    enb.NSubframe = 0;
    enb.TotSubframes = numSubframes;
    enb.DuplexMode = "FDD";
    if numCRSPorts>1
        enb.PDSCH.TxScheme = "SpatialMux";
    end
    enb.CarrierFrequency = carrierFrequency;

    % Define the power scaling of LTE signals and channels.
    % These values are chosen solely to ease the visualization of each
    % channel and signal in the spectrogram view.
    enb.PDSCH.PDCCHPower = 24;
    enb.PCFICHPower = 12;
    enb.PHICHPower = 30;
    % To make sure that the LTE waveform contains PDCCH but not PDSCH, set
    % the PDSCH power resource element allocation so that the content of
    % the PDSCH does not affect the LTE waveform
    enb.PDSCH.Rho = -inf;
end

function ind = getReservedRE(numPRB,numCRSPorts,vShift,carrierFrequency,gnb)
    % Compute the REs used by the LTE CRS that the 5G PDSCH needs to
    % rate-match around.

    % Get the LTE configuration for the rate-matching around the CRS
    enb = struct();
    enb.NDLRB = numPRB;
```

```
    enb.CellRefP = numCRSPorts;
    enb.NCellID = vShift;
    enb.CarrierFrequency = carrierFrequency;
    enb.CyclicPrefix = "Normal";
    enb.DuplexMode = "FDD";

    % Get the RE indices for the CRS
    ind = getLTEREIndices(enb,gnb,"CRS");
end

function prbs = getReservedPRB(enb,gnb)
    % Compute the physical resource blocks used by the LTE PBCH, PSS, and
    % SSS that the 5G PDSCH needs to rate-match around.

    % Initialize empty nrPDSCHReservedConfig objects to store the reserved
    % PRBs. The example defines two objects because the periodicity of the
    % PBCH is different from that of the PSS and SSS.
    prbs = {nrPDSCHReservedConfig,nrPDSCHReservedConfig};

    % Compute the 5G BWP grid size
    gnbSize = size(nrResourceGrid(gnb));

    % Get the RE indices for the PBCH and convert them to subscript
    pbchInd = getLTEREIndices(enb,gnb,"PBCH");
    [pbchREInd,pbchSymbInd] = ind2sub(gnbSize,pbchInd);

    % Store the reserved PRBs for the LTE PBCH. Set the period considering
    % that the PBCH is transmitted in subframe 0 of each frame.
    prbInd = unique(floor(pbchREInd/12)); % 0-based PRBs associated to the RE indices
    prbs{1}.PRBSet = prbInd;
    prbs{1}.SymbolSet = unique(pbchSymbInd-1); % 0-based
    prbs{1}.Period = 10*gnb.SlotsPerSubframe;

    % Get the RE indices for the PSS and SSS and convert them to subscript
    ssInd = getLTEREIndices(enb,gnb,["PSS","SSS"]);
    [ssREInd,ssSymbInd] = ind2sub(gnbSize,ssInd);

    % Store the reserved PRBs for the LTE synchronization signals PSS and
    % SSS. Set the period considering that the PSS and SSS are transmitted
    % in subframes 0 and 5 of each frame.
    prbInd = unique(floor(ssREInd/12)); % 0-based PRBs associated to the RE indices
    prbs{2}.PRBSet = prbInd;
    prbs{2}.SymbolSet = unique(ssSymbInd-1); % 0-based
    prbs{2}.Period = 5*gnb.SlotsPerSubframe;
end

function ind = getLTEREIndices(enb,gnb,signalName)
    % Compute the REs used by LTE always-on signals (CRS, PBCH, PSS, and
    % SSS) that the 5G PDSCH needs to rate-match around.

    % enb frequency indices 'enbf' in terms of 15 kHz subcarrier spacing
    enbSize = lteResourceGridSize(enb);
    enbK = enbSize(1);
    enbf = [-enbK/2:-1 1:enbK/2].' + enb.CarrierFrequency;

    % gnb frequency indices 'gnbf' in terms of 15 kHz subcarrier spacing
    gnbSize = size(nrResourceGrid(gnb));
    gnbK = gnbSize(1);
```

```
        gnbf = (-gnbK/2:(gnbK/2 - 1)).';
        gnbf = gnbf(gnb.NStartGrid*12 + (1:gnbK));

        % Compute the CRS indices
        crsIndices = [];
        if any(signalName=="CRS")
            crsIndices = lteCellRSIndices(enb,"sub");
        end
        % Compute the PBCH indices
        pbchIndices = [];
        if any(signalName=="PBCH")
            pbchIndices = ltePBCHIndices(enb,"sub");
        end
        % Compute the PSS indices
        pssIndices = [];
        if any(signalName=="PSS")
            pssIndices = ltePSSIndices(enb,0,"sub");
        end
        % Compute the SSS indices
        sssIndices = [];
        if any(signalName=="SSS")
            sssIndices = lteSSSIndices(enb,0,"sub");
        end
        % Create the list of all indices
        indTot = [crsIndices; pbchIndices; pssIndices; sssIndices];
        enbk = indTot(:,1);
        enbl = indTot(:,2);

        enbf = enbf(enbk);

        gnbk = arrayfun(@(x)find(gnbf==x),enbf,UniformOutput=false);
        z = cellfun(@isempty,gnbk);
        gnbk = cat(1,zeros(0,1),gnbk{:});

        gnbl = enbl;
        gnbl(z) = [];

        ind = sub2ind(gnbSize,gnbk,gnbl);
        ind = ind - 1; % 1-based to 0-based
    end

    function plotResourceGrid(in,grid,numSubframes,chNames)
        % Plot the resource grid for each antenna.

        if isfield(in,"CellRefP") % LTE
            isLTE = 1;
            nsf = numSubframes;
            enb = in;
            numPorts = enb.CellRefP;
        else % 5G
            isLTE = 0;
            gnb = in.gnb;
            pdsch = in.pdsch;
            nsf = numSubframes* gnb.SlotsPerSubframe;
            numPorts = pdsch.NumLayers;
        end
        chNames = ["Unallocated" chNames];
```

```
% Define channel power levels
chplevel.Unallocated = 0;   % Unallocated REs
chplevel.CRS = 0.9;         % LTE CRS
chplevel.PBCH = 0.39;       % LTE PBCH
chplevel.PSS = 0.51;        % LTE PSS
chplevel.SSS = 0.75;        % LTE SSS
chplevel.PDCCH = 0.3;       % LTE PDCCH
chplevel.PCFICH = 0.83;     % LTE PCFICH
chplevel.PHICH = 0.5;       % LTE PHICH
chplevel.PDSCH = 0.83;      % 5G PDSCH
chplevel.PDSCH_DMRS = 0.45; % 5G PDSCH DM-RS

% Generate the resource grid for plotting
rbGrid = [];
gridSize = size(grid);
gridSize(2) = gridSize(2)/nsf; % Length of a single subframe or slot
for ns = 0:nsf-1 % Loop over each LTE subframe or 5G slot
    % Initialize an empty grid for the current subframe or slot
    rbGridS = zeros(gridSize);

    % Populate the grid
    if isLTE
        enb.NSubframe = mod(ns,10);

        rbGridS(lteCellRSIndices(enb)) = chplevel.CRS;
        rbGridS(ltePSSIndices(enb)) = chplevel.PSS;
        rbGridS(lteSSSIndices(enb)) = chplevel.SSS;
        rbGridS(ltePBCHIndices(enb)) = chplevel.PBCH;
        rbGridS(ltePCFICHIndices(enb)) = chplevel.PCFICH;
        rbGridS(ltePHICHIndices(enb)) = chplevel.PHICH;
        rbGridS(ltePDCCHIndices(enb)) = chplevel.PDCCH;

    else % 5G
        gnb.NSlot = mod(ns,10);

        rbGridS(nrPDSCHIndices(gnb,pdsch)) = chplevel.PDSCH;
        rbGridS(nrPDSCHDMRSIndices(gnb,pdsch)) = chplevel.PDSCH_DMRS;
    end

    % Append the resource grid for the current subframe or slot to the
    % overall resource grid
    rbGrid = cat(2,rbGrid,rbGridS);
end

% Define the colormap and the scaling needed for the plot
cmap = parula(256);
chpval = struct2cell(chplevel);
cscaling = length(cmap);
fnames = strrep(fieldnames(chplevel),"_"," ");
fnames = strrep(fnames,"DMRS","DM-RS");
chpval = chpval(contains(fnames,chNames));
clevels = floor(cscaling*[chpval{:}]);

% Define the plot title
if isLTE
    figTitle = "LTE Resource Grid";
else
    figTitle = "5G Resource Grid";
```

```
        end
        plotTitle = "Antenna ";

        % Define the axes limits for the plot to be 0-based
        minX = 0;
        maxX = size(rbGrid,2)-1;
        minY = 0;
        maxY = size(rbGrid,1)-1;

        % Plot the resource grid for each port or layer
        fh = figure;
        t = tiledlayout(fh,"flow",TileSpacing="Compact");
        for idx = 1:numPorts
            nexttile;
            ax = gca;
            img = image(ax,cscaling*abs(rbGrid(:,:,idx)));
            img.XData = minX:maxX; % 0-based symbol number
            img.YData = minY:maxY; % 0-based subcarrier number
            axis(ax,[minX-0.5 maxX+0.5 minY-0.5 maxY+0.5]);
            axis(ax,"xy");
            if numPorts>1
                title(ax,plotTitle+num2str(idx-1)); % 0-based antenna number
            end
            colormap(ax,cmap);
        end
        title(t,figTitle);
        xlabel(t,"Symbols");
        ylabel(t,"Subcarriers");

        % Create the legend
        N = length(clevels);
        p = struct(Marker="s",LineStyle="none",MarkerSize=8,LineWidth=1.25);
        L = line(ax,NaN(N),NaN(N),p); % Generate markers
        % Index the color map and associate the selected colors with the markers
        c = mat2cell([cmap(1,:);cmap(clevels(2:end),:)],ones(1,N),3);
        set(L,{"color"},c); %#ok<STRSCALR>
        set(L,{"MarkerFaceColor"},c); %#ok<STRSCALR>
        lg = legend(ax,chNames{:});
        lg.Layout.Tile = "East";
    end

    function plotDSSSpectrogram(wave,ofdmInfo,enbSR,gnbSR,numSubframes)
        % Plot the spectrogram of the combined waveform.

        % To minimize the artifacts shown in the spectrogram and due to out of
        % band leakage of the resampled waveform, adjust the waveform to
        % consider the filter response.
        cpLengths = double(ofdmInfo.CyclicPrefixLengths);
        OSR = max(enbSR/gnbSR,gnbSR/enbSR);
        n = min(10,floor(max(cpLengths)/(2*OSR)));
        order = 2*n*OSR; % Order of the filter used during resampling
        wave = circshift(wave,order/2);

        fh = figure;
        t = tiledlayout(fh,"flow",TileSpacing="Compact");
        % Compute the cyclic prefixes to remove from the waveform for each
        % subframe.
        nfft = double(ofdmInfo.Nfft);
```

```
    if isfield(ofdmInfo,'SampleRate')
        sampleRate = ofdmInfo.SampleRate;
    else
        sampleRate = ofdmInfo.SamplingRate;
    end
    symbolLengths = nfft+cpLengths;
    cpidxs = arrayfun(@(x,y)(x+(1:y)),cumsum([0 symbolLengths(1:end-1)]),cpLengths,UniformOutput=
    cpidxs = [cpidxs{:}];
    sfWaveLength = sampleRate/1000;

    % Loop over each antenna
    for idx = 1:size(wave,2)
        nexttile;
        txWaveNoCP = [];
        for nsf = 1:numSubframes
            waveNoCP = wave(1+(nsf-1)*sfWaveLength:nsf*sfWaveLength,idx);
            waveNoCP(cpidxs) = []; % Remove the cyclic prefix in this subframe
            txWaveNoCP = cat(1,txWaveNoCP, waveNoCP);
        end
        spectrogram(txWaveNoCP,ones(nfft,1),0,nfft,"centered",sampleRate,"yaxis",MinThreshold=-1
        title(['Antenna ' num2str(idx-1)]); % 0-based antenna number
    end
    title(t,"Spectrogram of the Combined DSS Waveform");
end
```

See Also

Functions
`lteRMCDLTool`

Objects
`nrPDSCHConfig`

Related Examples

- “5G NR Downlink Vector Waveform Generation” (5G Toolbox)
- “5G NR Uplink Vector Waveform Generation” (5G Toolbox)
- “5G NR-TM and FRC Waveform Generation” (5G Toolbox)

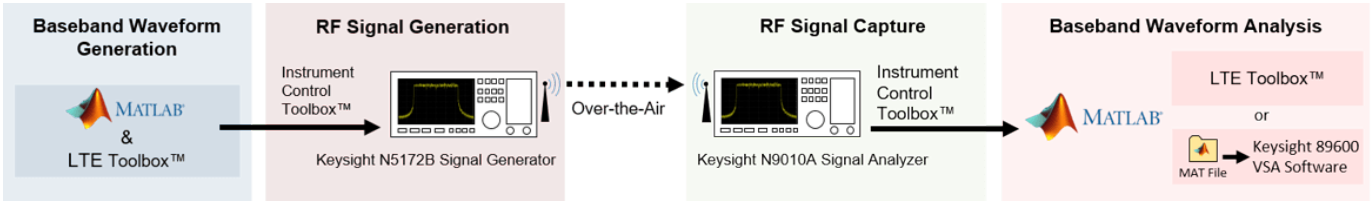
Waveform Generation and Transmission Using LTE Toolbox with Test and Measurement Equipment

This example shows how to generate and analyze an over-the-air LTE waveform by using the LTE Toolbox™, the Instrument Control Toolbox™ and a Keysight Technologies® RF signal generator and analyzer.

Introduction

The LTE Toolbox can be used to generate standard baseband IQ downlink test model (E-TM) waveforms and uplink and downlink reference measurement channel (RMC) waveforms. Using the LTE Toolbox with the “Instrument Control Toolbox” allows LTE waveforms created in MATLAB® to be used with test and measurement hardware. Waveforms created by the LTE Toolbox can be modulated for transmission using a signal generator. Waveforms captured using a signal analyzer can be analyzed using MATLAB and LTE Toolbox functions.

In this example the Instrument Control Toolbox is used to interface with an RF signal generator and analyzer. An E-TM waveform, synthesized in MATLAB using the LTE Toolbox, is downloaded to a Keysight Technologies N5172B signal generator for over-the-air transmission. The over-the-air signal is captured using a Keysight Technologies N9010A signal analyzer and retrieved into MATLAB for analysis.



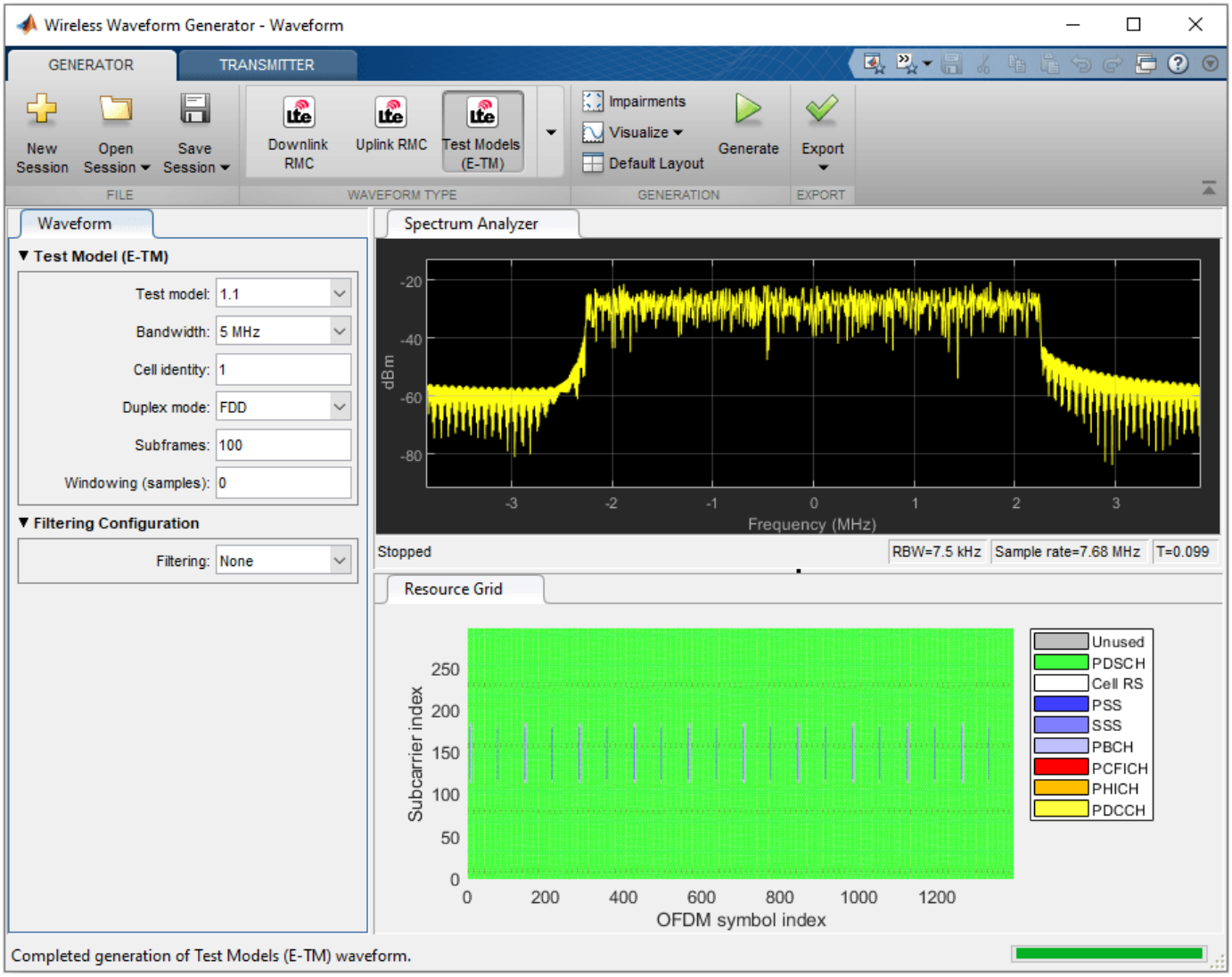
The captured waveform can be analyzed using the LTE Toolbox as demonstrated in the following examples:

- “Waveform Acquisition and Analysis using LTE Toolbox with Test and Measurement Equipment” on page 2-592
- “PDSCH Error Vector Magnitude (EVM) Measurement” on page 2-583
- “LTE Downlink Adjacent Channel Leakage Power Ratio (ACLR) Measurement” on page 2-575

This example illustrates how external test and measurement equipment can be used to analyze the received waveform; in this case Keysight Technologies Vector Signal Analysis (VSA) software is also used.

Generate a Baseband Waveform Using the LTE Toolbox

The LTE Toolbox provides GUIs and functions that generate test model waveforms as per [1]. `lteTestModelTool` can be used to configure and create the signal using a GUI.



Alternatively, the functions `lteTestModel` and `lteTestModelTool` allow the programmatic configuration and generation of LTE test models and baseband IQ waveforms.

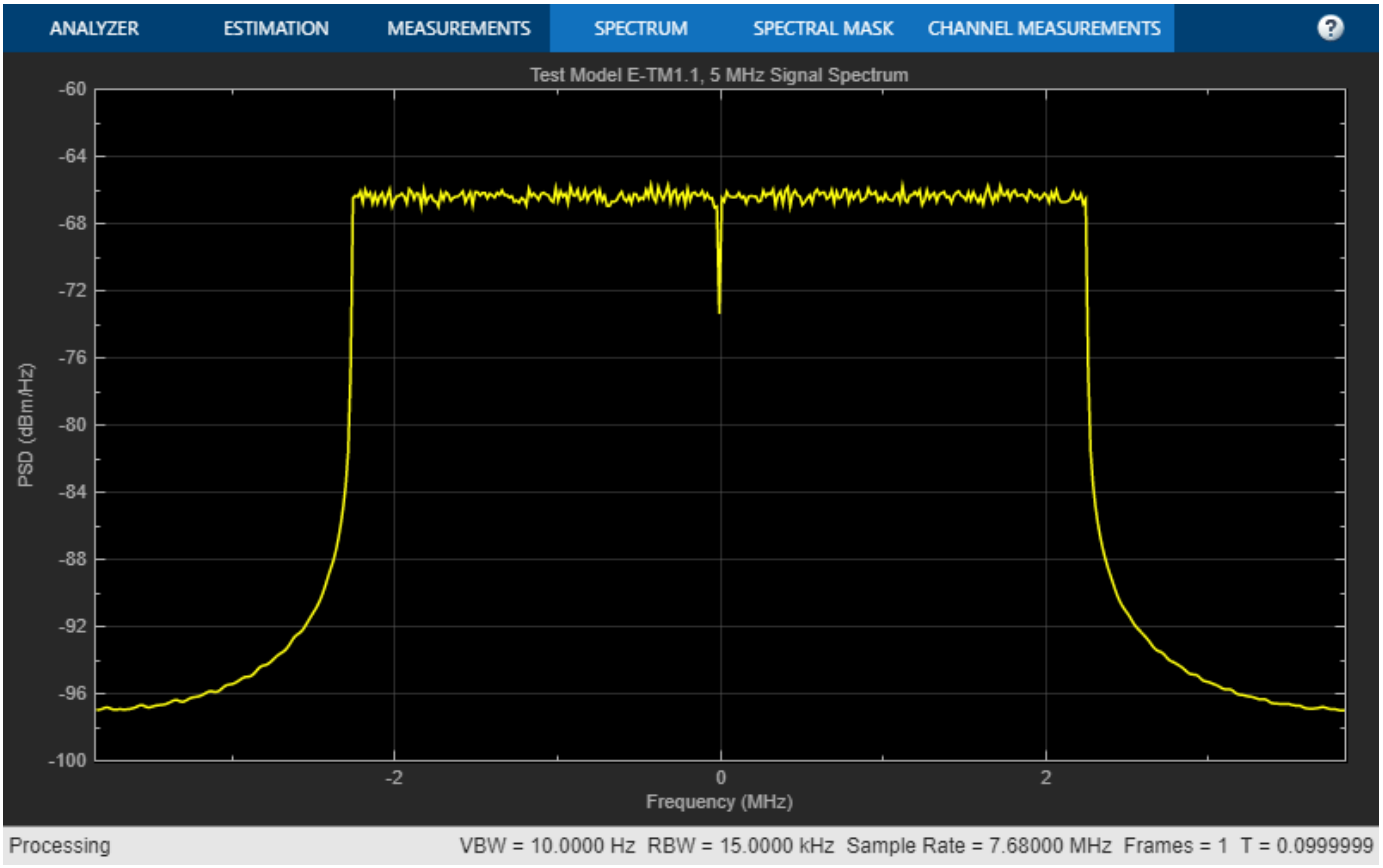
```
config = lteTestModel('1.1', '5MHz');  % Test Model 1.1, 5MHz bandwidth
config.TotSubframes = 100;             % Generate 100 subframes
[waveform, tmgrid, config] = lteTestModelTool(config);
```

For more details on the LTE test model signal, refer to the accompanying “LTE Downlink Test Model (E-TM) Waveform Generation” on page 2-562 example.

The frequency spectrum of the generated time-domain waveform, `waveform`, can be viewed using the “DSP System Toolbox” `spectrumAnalyzer` object. As expected, the 5MHz signal bandwidth is clearly visible at baseband.

```
% Calculate the spectral content in the LTE signal
spectrumPlotTx = spectrumAnalyzer;
spectrumPlotTx.SampleRate = config.SamplingRate;
spectrumPlotTx.SpectrumType = 'power-density';
spectrumPlotTx.SpectrumUnits =  'dBm';
```

```
spectrumPlotTx.RBWSource = 'Property';
spectrumPlotTx.RBW = 15e3;
spectrumPlotTx.FrequencySpan = 'span-and-center-frequency';
spectrumPlotTx.Span = 7.68e6;
spectrumPlotTx.CenterFrequency = 0;
spectrumPlotTx.Window = 'rectangular';
spectrumPlotTx.YLimits = [-100 -60];
spectrumPlotTx.YLabel = 'PSD';
spectrumPlotTx.Title = 'Test Model E-TM1.1, 5 MHz Signal Spectrum';
spectrumPlotTx.ShowLegend = false;
spectrumPlotTx(waveform);
```



Generate an Over-the-Air Signal Using an RF Signal Generator

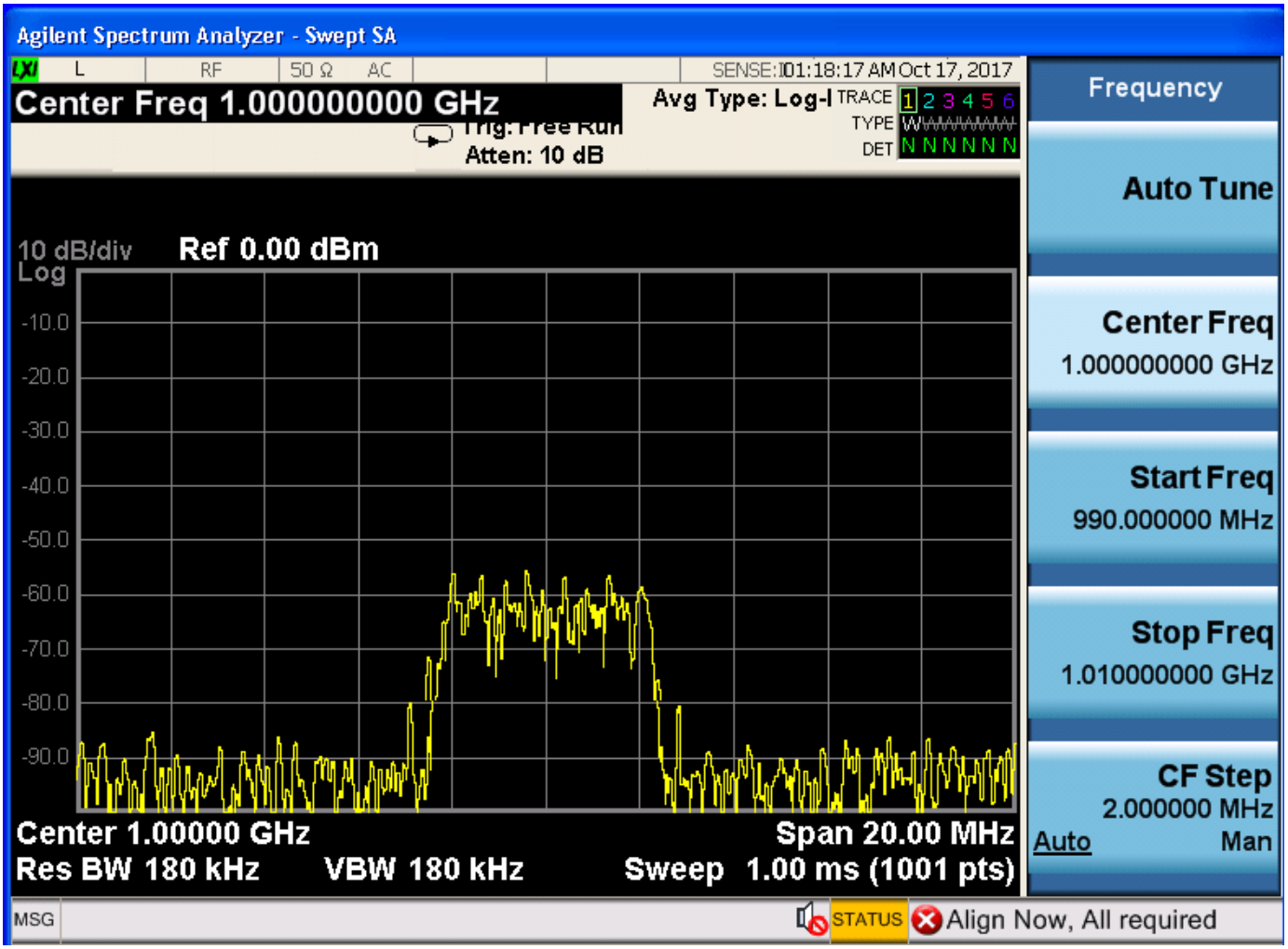
The Instrument Control Toolbox is used to download and play the test model waveform created by the LTE Toolbox, `waveform`, using the Keysight Technologies N5172B signal generator. This creates an RF LTE signal with a center frequency of 1GHz. Note 1GHz was selected as an example frequency and is not intended to be a recognized LTE channel.

```
% Download the baseband IQ waveform to the instrument. Generate the RF
% signal at a center frequency of 1GHz and output power of 0dBm.
power = 0;       % Output power
loopCount = Inf; % Number of times to loop

% Configure the signal generator, download the waveform and loop
rf = rfsiggen('TCPIP0::A-N5172B-50283.dhcp.mathworks.com::inst0::INSTR');
```

```
download(rf,waveform.',config.SamplingRate);
start(rf,1e9,power,loopCount);
```

Inspect the RF signal generator object `rfsiggen` (Instrument Control Toolbox) for more details on the commands used to download and play the waveform.

The frequency spectrum of the RF signal transmitted by the signal generator can be viewed using a spectrum analyzer tuned to the 1GHz center frequency. The screen capture below, from a Keysight Technologies N9010A signal analyzer, clearly shows the 5MHz signal bandwidth.



Acquire the Baseband Signal in MATLAB from a Signal Analyzer

To analyze the over-the-air transmission in MATLAB, the Instrument Control Toolbox is used to configure the Keysight Technologies N9010A signal analyzer and capture baseband IQ data. The helper function hCaptureIQUsingN9010A.m retrieves the baseband IQ data, `IQData`, and the sample rate, `sampleRate`, from the signal analyzer, ready for analysis in MATLAB.

```
[IQData, sampleRate] = hCaptureIQUsingN9010A( ...
    'A-N9010A-21026.dhcp.mathworks.com', config.TotSubframes*1e-3, ...
    1e9, 5e6, false, 990e6, 1010e6, 200e3, 200e3);
```

Inspect the function hCaptureIQUsingN9010A.m for more details on input parameters and the commands needed to configure the Keysight Technologies N9010A signal analyzer and retrieve the data.

```
% When finished transmitting and receiving, stop the waveform output
stop(rf);
disconnect(rf);
clear rf;
```

Once the captured baseband IQ data, `IQData`, is acquired from the signal analyzer into MATLAB using Instrument Control Toolbox, custom visualization, analysis and decoding of the acquired baseband signal can be immediately performed on the data within MATLAB using the LTE Toolbox, Communications Toolbox™ and DSP System Toolbox. The data can also be stored in a MAT file for post-data analysis in MATLAB. In this example, the baseband IQ data is stored in a MAT file along with system parameters for use with Keysight Technologies VSA software.

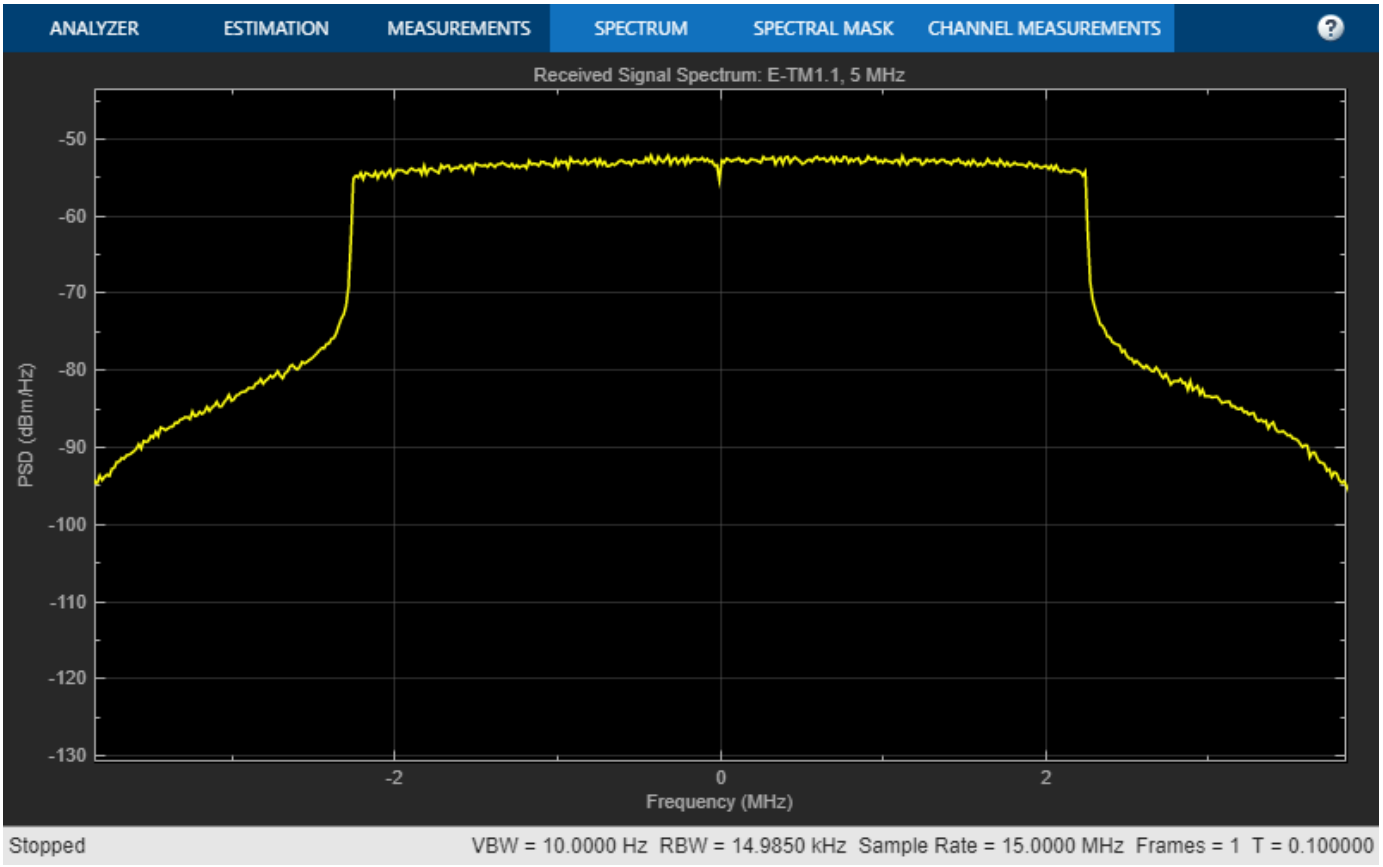
Note the returned sampling rate from the instrumentation hardware is different than the configured sampling rate. The returned sample rate is valid for measurements using Keysight Technologies VSA software, but actual signal decoding would require a resampling of the acquired data.

```
% MAT file interface parameters
FreqValidMax = 1.010e9;
FreqValidMin = 9.90e8;
InputCenter = 1e9;
XDelta = 1/sampleRate;
Y = IQData;

% Common set with fixed values
IQ = 0;
InputRefImped = 50;
InputZoom = 1;
XDomain = 2;
XStart = 0;
XUnit = 'Sec';
YUnit = 'V';

% Save the variables for subsequent up-loading into VSA.
save('DownlinkTestModel1p1FDD5MHz_Rx.mat', 'FreqValidMax', 'FreqValidMin',...
     'IQ', 'InputCenter', 'InputRefImped', 'InputZoom', 'XDelta', ...
     'XDomain', 'XStart', 'XUnit', 'Y', 'YUnit');
```

Plotting the frequency spectrum of the retrieved time-domain baseband waveform, `Y`, using the `spectrumAnalyzer` object shows the expected 5MHz occupied bandwidth, with impairments due to RF transmission and reception.

```
spectrumPlotRx = spectrumAnalyzer;
spectrumPlotRx.SampleRate =  1/XDelta;
spectrumPlotRx.SpectrumType = 'power-density';
spectrumPlotRx.SpectrumUnits =  'dBm';
spectrumPlotRx.RBWSource = 'property';
spectrumPlotRx.RBW = 15e3;
spectrumPlotRx.FrequencySpan = 'span-and-center-frequency';
spectrumPlotRx.Span = 7.68e6;
spectrumPlotRx.CenterFrequency = 0;
spectrumPlotRx.Window = 'rectangular';
spectrumPlotRx.YLabel = 'PSD';
spectrumPlotRx.Title = 'Received Signal Spectrum: E-TM1.1, 5 MHz';
```

```
spectrumPlotRx.ShowLegend = false;
spectrumPlotRx(Y);
release(spectrumPlotRx)
```

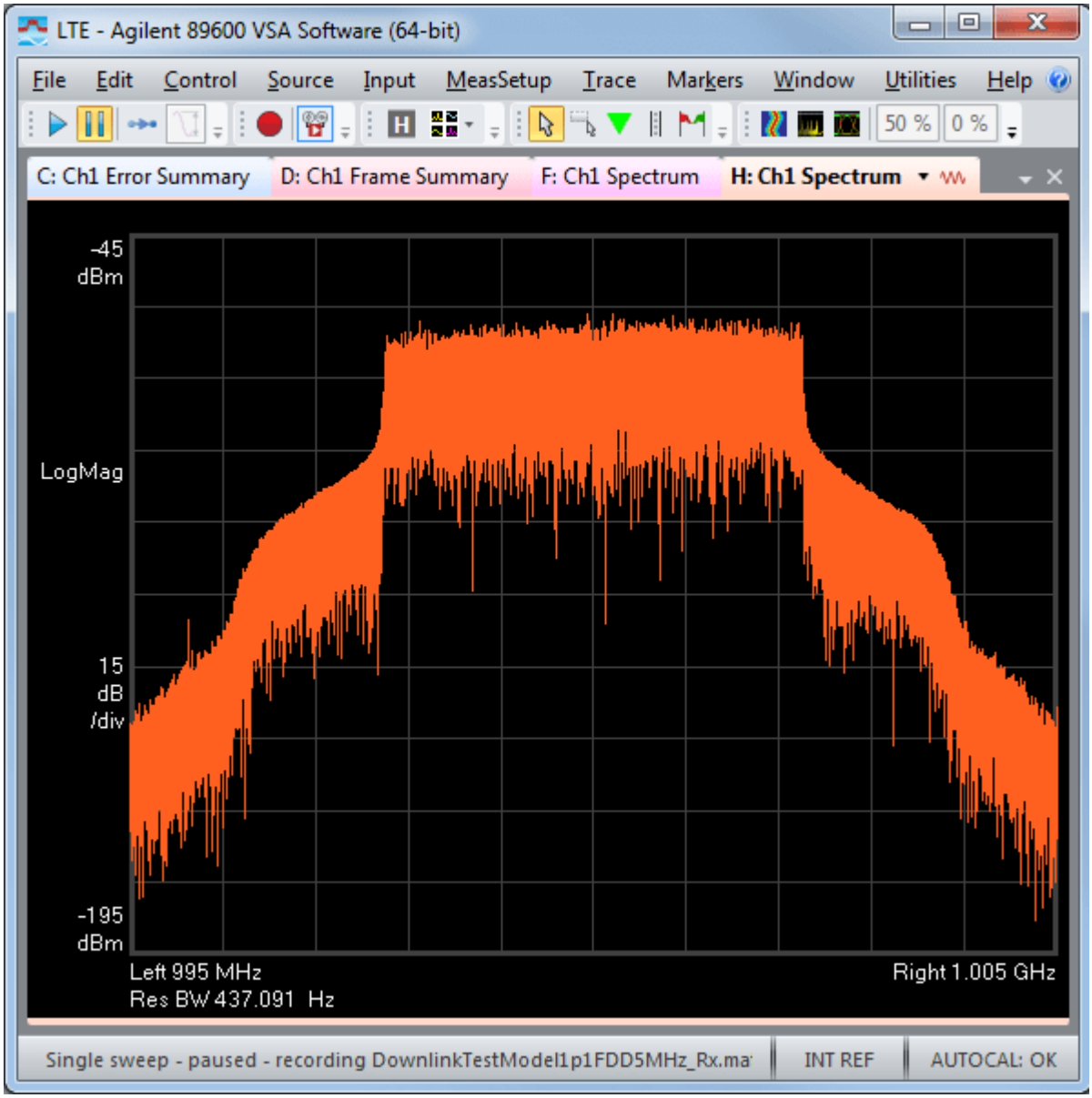


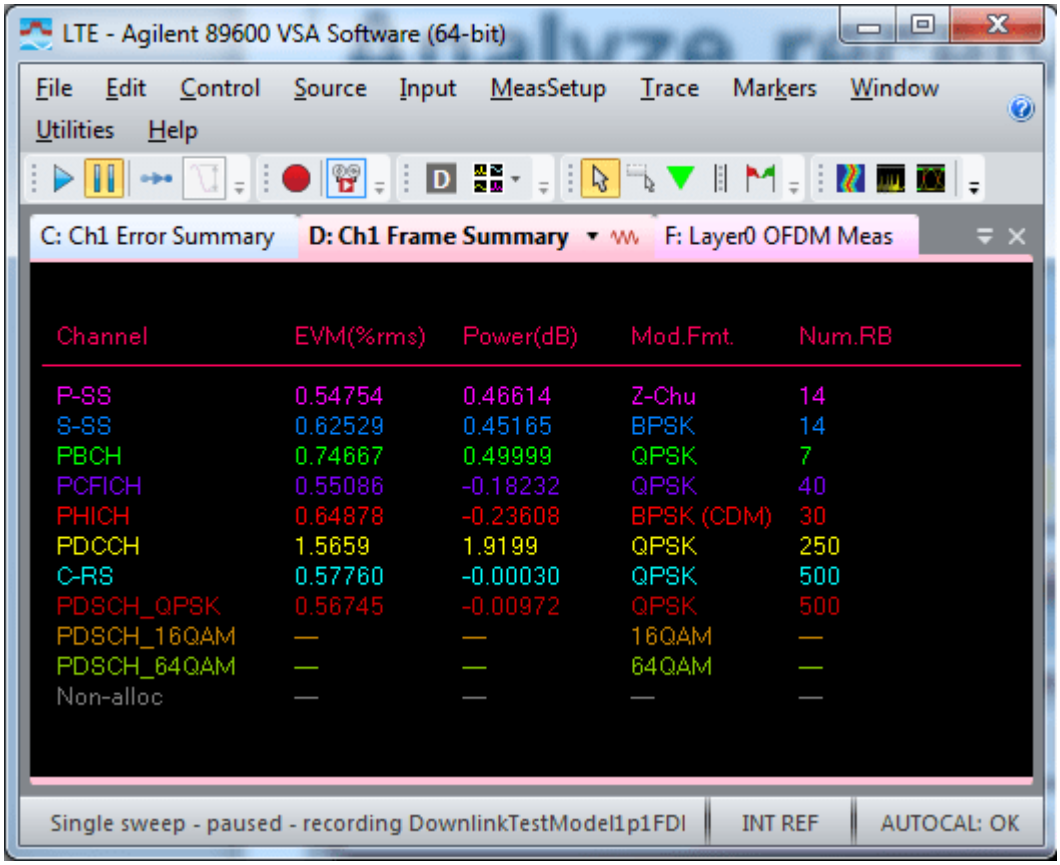
Analyze the Received LTE Signal

The captured waveform can be analyzed using the LTE Toolbox as demonstrated in the following examples:

- “Waveform Acquisition and Analysis using LTE Toolbox with Test and Measurement Equipment” on page 2-592
- “PDSCH Error Vector Magnitude (EVM) Measurement” on page 2-583
- “LTE Downlink Adjacent Channel Leakage Power Ratio (ACLR) Measurement” on page 2-575

The MAT file containing the retrieved data is loaded into Keysight Technologies VSA software and LTE-specific measurements are made. The screenshots of the interactive displays from the VSA software confirms the waveform characteristics of the E-TM1.1 test model in terms of the occupied bandwidth, number of resource blocks used per channel and signal, their corresponding power levels and low EVM values.


LTE - Agilent 89600 VSA Software (64-bit)
File Edit Control Source Input MeasSetup Trace Markers Window Utilities Help
50 % 0 %
C: Ch1 Error Summary
D: Ch1 Frame Summary
F: Ch1 Spectrum
H: Ch1 Spectrum
-45 dBm
LogMag
15 dB /div
-195 dBm
Left 995 MHz
Res BW 437.091 Hz
Right 1.005 GHz
Single sweep - paused - recording DownlinkTestModel1p1FDD5MHz_Rx.ma
INT REF
AUTOCAL: OK



Appendix

This example uses this helper function.

- hCaptureIQUsingN9010A.m

Selected Bibliography

1 3GPP TS 36.141 "Base Station (BS) conformance testing"

Reference Signal Measurements (RSRP, RSSI, RSRQ) for Cell Reselection

In the LTE system, a UE must detect and monitor the presence of multiple cells and perform cell reselection to ensure that it is "camped" on the most suitable cell. A UE "camped" on a particular cell will monitor the System Information and Paging of that cell, but it must continue to monitor the quality and strength of other cells to determine if cell reselection is required.

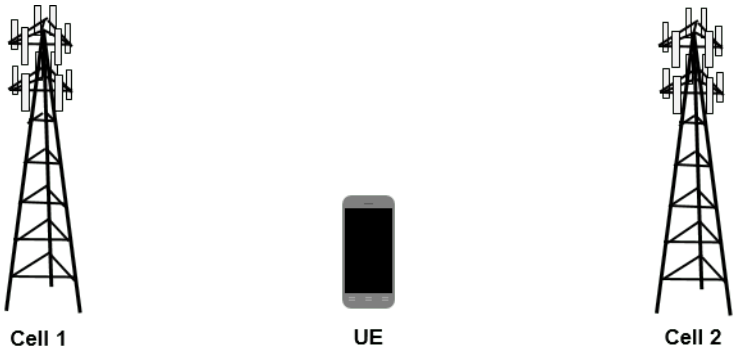
In this example, the cell reselection test environment described in TS 36.133 Annex A.4.2.2.1 [1] is configured. Cell search is then performed to determine the detected cells. Reference Signal (RS) measurements are made:

- Reference Signal Received Power (RSRP)
- Received Signal Strength Indicator (RSSI)
- Reference Signal Received Quality (RSRQ)

Finally the RSRP is used as the criterion for cell reselection.

Introduction

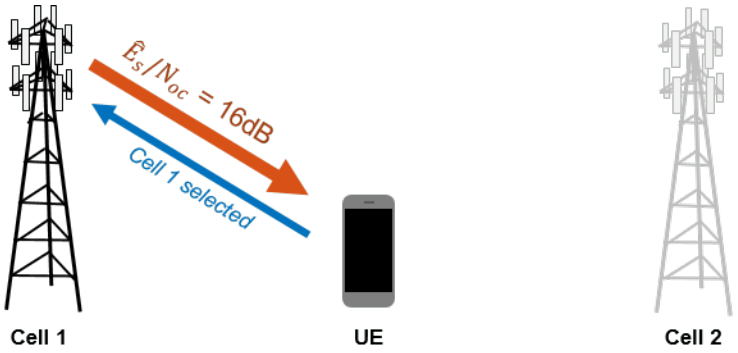
The purpose of the test in TS 36.133 Annex A.4.2.2.1 [1] is to verify that the requirements for TDD to TDD intra-frequency cell reselection are met. The test environment consists of a single TDD carrier, a single UE and two cells (Cell 1 and Cell 2) as shown in the diagram below:



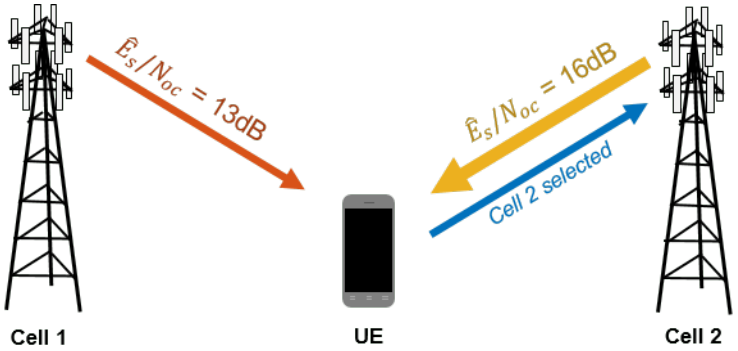
The test then defines three successive time periods T1...T3, during which Cell 1 and Cell 2 have different power levels. In each of the time periods, the UE must select the correct cell based on the cell reselection criteria defined in TS 36.304 Section 5.2 [2]. Those criteria involve the measured RSRP and RSRQ, minimum required RSRP and RSRQ levels, and various offsets. In this example, a simplified reselection procedure is used, where the cell with the highest RSRP is selected.

The power levels of each cell and the expected behavior of the UE in each time period are as follows:

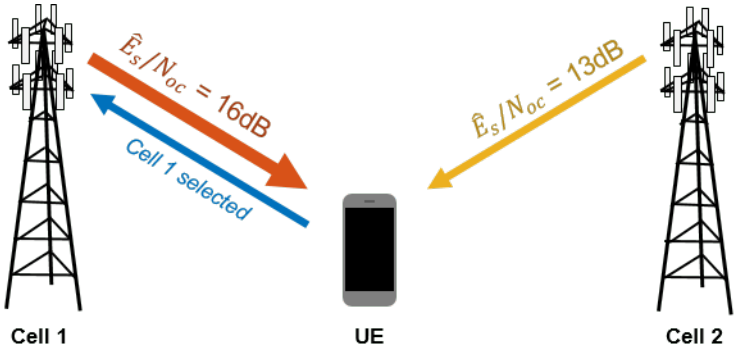
In time period T1, only Cell 1 is active and Cell 2 is powered off. The UE should select Cell 1:



In time period T2, Cell 2 is powered on and the power of Cell 1 is decreased. The UE should select Cell 2:



In time period T3, the power of Cell 2 is decreased and the power of Cell 1 is increased. The UE should select Cell 1:



This example will show how to use the LTE Toolbox™ to configure the test environment in TS 36.133 Annex A.4.2.2.1 [1], perform cell search to detect the cells which are present, and finally make RSRP measurements in order to perform cell reselection.

Signal Power Setup

The signal powers (in dB) for both cells in each time period are set up according to TS 36.133 Table A.4.2.2.1-2 [1]. The signal powers are vectors, where each element gives the signal power for each of the three time periods of the test.

```
SINRdB1 = [  16 13 16];     % Es/Noc for Cell 1
SINRdB2 = [-Inf 16 13];     % Es/Noc for Cell 2
```

Noise Power Setup

The noise power (in dBm) is set up according to TS 36.133 Table A.4.2.2.1-2 [1]. The linear noise power is then calculated and will be used later to configure the AWGN added in the test.

```
NocdBm = -98;               % dBm/15kHz average power spectral density
NocdBW = NocdBm-30;         % Noc in dBW/15kHz
Noc = 10^(NocdBW/10);       % linear Noc
```

Cell 1 Configuration

The function `lteRMCDL`, which creates a Reference Measurement Channel (RMC) configuration, is used to create a complete eNodeB configuration for Cell 1. The RMC used is RMC R.7, which has the required bandwidth of 10MHz as specified in TS 36.133 Table A.4.2.2.1-1 [1]. The TDD uplink-downlink configuration and special subframe configuration for the test are also specified in that table. Orthogonal Channel Noise Generation (OCNG) is enabled and an arbitrarily-chosen cell identity is set.

```
cell1 = lteRMCDL('R.7','TDD');
cell1.TDDConfig = 1;
cell1.SSC = 6;
cell1.OCNGPDCCHEnable = 'On';
cell1.OCNGPDSCHEnable = 'On';
cell1.NCellID = 101;
```

Cell 2 Configuration

The configuration of Cell 2 is identical to that of Cell 1 except a different cell identity is used.

```
cell2 = cell1;
cell2.NCellID = 313;
```

Cell Search Configuration

A structure `searchalg` is created, which will be used to configure the operation of the `lteCellSearch` function used to detect cells. When detecting multiple cells, this function ranks the cells according to the peak magnitude of the correlations used to detect PSS and SSS, rather than RSRP. Therefore `MaxCellCount`, the number of cells to detect, is set to 3 as the strongest two cells in terms of RSRP (expected to be Cell 1 and Cell 2) are not necessarily the strongest two cells detected by the `lteCellSearch` function. The SSS detection method is set to `'PostFFT'`, where SSS detection is performed in the frequency domain, with the OFDM demodulation synchronized using the timing estimate from PSS detection.

```
searchalg.MaxCellCount = 3;
searchalg.SSSDetection = 'PostFFT';
```

Simulation Loop for Test Time Periods

The simulation is run in a loop for the three time periods T1...T3 defined in the test. The processing steps for each time period are as follows:

- Cell 1 is transmitted at the specified power level for the time period
- Cell 2 is transmitted at the specified power level for the time period, and the timing offset between cells specified in TS 36.133 Table A.4.2.2.1-1 [1] is applied

- An AWGN waveform is created at the specified power level for the test, and the Cell 1, Cell 2 and AWGN waveforms are added together to model the received waveform at the UE
- Cell search is performed on the received waveform
- For each detected cell, the received waveform is synchronized, OFDM demodulated and the RSRP is measured (using the function hRSMeasurements) and the detected cell identities are listed in order of decreasing RSRP
- The selected cell (Cell 1 or Cell 2) is determined by selecting the cell identity which has the highest measured RSRP

A number of values are recorded at the MATLAB® Command Window for each time period:

- For Cell 1 and Cell 2: the cell identity, SINR ($\hat{Es}$ / Iot), SNR ($\hat{Es}$ / Noc) and ideal RSRP (measured from the transmitted waveforms)
- For each detected cell at the receiver: the cell identity and the measured RSRP (measured from the received waveform)
- The selected cell (and its cell identity)

Note that a number of other physical layer parameters such as the cyclic prefix length and duplex mode are assumed to be known and are assumed to be equal for each eNodeB. See the “Cell Search, MIB and SIB1 Recovery” on page 2-351 example for more information on detecting these parameters.

```
nTimePeriods = 3;

txRSRPs = -inf(nTimePeriods,2);
rxRSRPs = -inf(nTimePeriods,searchalg.MaxCellCount);
detectedCells = zeros(nTimePeriods,1);

rng('default');
separator = repmat('-',1,44);

% For each time period:
for T = 1:nTimePeriods

    fprintf('\n%s\n Time period T%d\n%s\n\n',separator,T,separator);
    fprintf('       tx:    Cell 1     Cell 2\n');
    fprintf('  NCellID: %7d    %7d\n',cell1.NCellID,cell2.NCellID);

    % Cell 1 transmission.
    SINR1 = 10^(SINRdB1(min(T,end))/10);     % linear Es/Noc
    Es1 = SINR1*Noc;                         % linear Es per RE
    [txcell1,~,info] = lteRMCDLTool(cell1,randi([0 1],1000,1));
    txcell1 = txcell1 * sqrt(Es1);
    rxwaveform = txcell1;

    % Cell 2 transmission.
    SINR2 = 10^(SINRdB2(min(T,end))/10);     % linear Es/Noc
    Es2 = SINR2*Noc;                         % linear Es per RE
    txcell2 = lteRMCDLTool(cell2,randi([0 1],1000,1));
    txcell2 = txcell2 * sqrt(Es2);
    delta_t = round(info.SamplingRate*3e-6); % Time offset between cells
    rxwaveform = rxwaveform + circshift(txcell2,delta_t);

    % Display ideal signal to noise/interference ratios based on test
    % parameters.
    EsToIot1 = 10*log10(Es1) - 10*log10(Es2 + Noc);
```

```
EsToNoc1 = 10*log10(Es1) - 10*log10(Noc);
EsToIot2 = 10*log10(Es2) - 10*log10(Es1 + Noc);
EsToNoc2 = 10*log10(Es2) - 10*log10(Noc);
fprintf('   Es/Iot: %7.2fdB  %7.2fdB\n',EsToIot1,EsToIot2);
fprintf('   Es/Noc: %7.2fdB  %7.2fdB\n',EsToNoc1,EsToNoc2);

% Perform Reference Signal (RS) measurements on the transmitted
% signals.
rxgridcell1 = lteOFDMDemodulate(cell1,txcell1);
rsmeas1 = hRSMeasurements(cell1,rxgridcell1);
txRSRPs(T,1) = rsmeas1.RSRPdBm;
rxgridcell2 = lteOFDMDemodulate(cell2,txcell2);
rsmeas2 = hRSMeasurements(cell2,rxgridcell2);
txRSRPs(T,2) = rsmeas2.RSRPdBm;
fprintf('     RSRP: %7.2fdBm %7.2fdBm\n',txRSRPs(T,1),txRSRPs(T,2));

% Add noise.
No = sqrt(Noc/(2*double(info.Nfft)));
noise = No*complex(randn(size(rxwaveform)),randn(size(rxwaveform)));
rxwaveform = rxwaveform + noise;

% Cell search.
% NDLRB is only required so that lteCellSearch can infer the sampling
% rate of 'rxwavefom'
enb.NDLRB = cell1.NDLRB;
% assumed parameters
enb.DuplexMode = cell1.DuplexMode;
enb.CyclicPrefix = cell1.CyclicPrefix;
% perform cell search
[cellIDs,offsets] = lteCellSearch(enb,rxwaveform,searchalg);

% Compute RSRPs for each detected cell.
% The TDD uplink-downlink configuration and special subframe
% configuration are assumed to be known. The assumption of CellRefP=1
% here means that the RS measurements will only be calculated for
% cell-specific reference signal port 0. NSubframe is set to zero
% because the timing offsets returned by lteCellSearch are relative to
% the start of a frame.
enb.TDDConfig = cell1.TDDConfig;
enb.SSC = cell1.SSC;
enb.CellRefP = 1;
enb.NSubframe = 0;
nDetected = length(cellIDs);
for n = 1:nDetected
    enb.NCellID = cellIDs(n);
    rxgrid = lteOFDMDemodulate(enb,rxwaveform(1+offsets(n):end,:));
    rsmeas = hRSMeasurements(enb,rxgrid);
    rxRSRPs(T,n) = rsmeas.RSRPdBm;
end
[~,idx] = sort(rxRSRPs(T,1:nDetected),'descend');
fprintf('\n       rx:\n');
for n = 1:nDetected
    fprintf('  NCellID: %3d  RSRP: %7.2fdBm\n',cellIDs(idx(n)),rxRSRPs(T,idx(n)));
end

% Select the cell with the highest RSRP.
enb.NCellID = cellIDs(idx(1));
detectedCells(T) = find(enb.NCellID==[cell1.NCellID cell2.NCellID]);
```

```
    fprintf('\n Selected: Cell %d (NCellID=%d)\n',detectedCells(T),enb.NCellID);

end
```

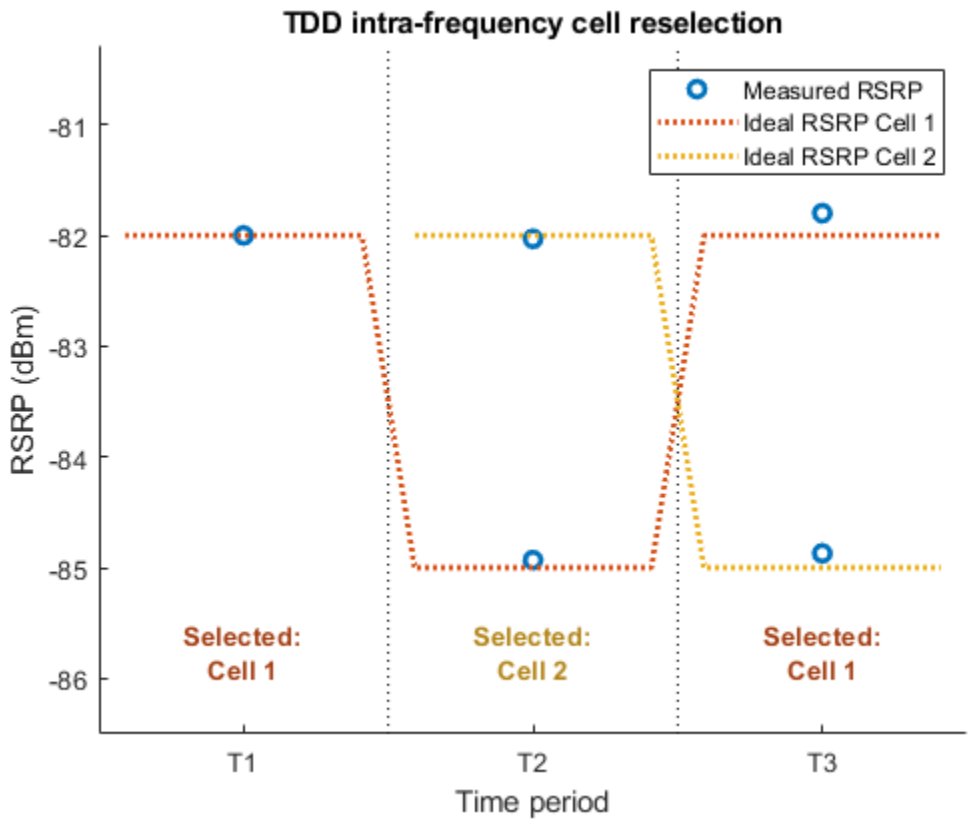
```
--------------------------------------------
 Time period T1
--------------------------------------------

       tx:    Cell 1     Cell 2
  NCellID:      101        313
    Es/Iot:   16.00dB     -InfdB
    Es/Noc:   16.00dB     -InfdB
      RSRP:  -82.00dBm    -InfdBm

       rx:
  NCellID: 101  RSRP:  -82.00dBm
  NCellID: 278  RSRP: -108.41dBm
  NCellID: 437  RSRP: -109.49dBm

 Selected: Cell 1 (NCellID=101)

--------------------------------------------
 Time period T2
--------------------------------------------

       tx:    Cell 1     Cell 2
  NCellID:      101        313
    Es/Iot:   -3.11dB     2.79dB
    Es/Noc:   13.00dB    16.00dB
      RSRP:  -85.00dBm  -82.00dBm

       rx:
  NCellID: 313  RSRP:  -82.03dBm
  NCellID: 101  RSRP:  -84.93dBm
  NCellID: 325  RSRP: -108.91dBm

 Selected: Cell 2 (NCellID=313)

--------------------------------------------
 Time period T3
--------------------------------------------

       tx:    Cell 1     Cell 2
  NCellID:      101        313
    Es/Iot:    2.79dB    -3.11dB
    Es/Noc:   16.00dB    13.00dB
      RSRP:  -82.00dBm  -85.00dBm

       rx:
  NCellID: 101  RSRP:  -81.80dBm
  NCellID: 313  RSRP:  -84.87dBm
  NCellID: 437  RSRP: -108.33dBm

 Selected: Cell 1 (NCellID=101)
```

Simulation Results

Finally, the results obtained through simulation are plotted. The figure produced below illustrates the three time periods, showing for each time period:

- The ideal RSRPs (measured at the transmitter) for Cell 1 and Cell 2
- The measured RSRPs (measured at the receiver) for each detected cell
- The selected cell (Cell 1 or Cell 2) based on the cell identity of the cell which has the highest measured RSRP

```
hRSMeasurementsExamplePlot(txRSRPs,rxRSRPs,detectedCells);
```



It can be seen that the measured RSRPs are close to the expected ideal values, and that in each time period the UE selects the expected cell as described in the introduction.

Note that there are actually nine measured RSRP points (`MaxCellCount`=3 for each of the three time periods), but the plot axes are adjusted to focus on the RSRP region of interest (around -85dBm to -82dBm). The other measured RSRPs are around -110dBm, a function of the AWGN noise power (`NocdBm`=-98dBm) and the number of Cell-Specific Reference Signal resource elements integrated during the RS measurements.

Appendix

This example uses these helper functions.

- hRSMeasurements.m
- hRSMeasurementsExamplePlot.m

Selected Bibliography

1. 3GPP TS 36.133 "Requirements for support of radio resource management"
2. 3GPP TS 36.304 "User Equipment (UE) procedures in idle mode"

UMTS Downlink Waveform Generation

This example shows how to generate an HSDPA FRC H-Set using LTE Toolbox™.

Introduction

The LTE Toolbox can be used to generate standard compliant W-CDMA/HSPA/HSPA+ uplink and downlink complex baseband waveforms including pre-defined configurations for standard defined measurement channels. For the downlink this includes the Reference Measurement Channels (RMC), Fixed Reference Channel (FRC) H-Sets and Test Models (TM) defined in TS25.101 [1].

This example demonstrates how the two downlink related functions, `umtsDownlinkReferenceChannels` and `umtsDownlinkWaveformGenerator`, combine to support this feature. We show how they can generate an FRC H-Set waveform for HSDPA UE testing using one of the pre-defined configurations provided. We also present explicit MATLAB® code which lists all downlink generator parameters set up for this particular measurement channel. The FRC H-Sets are defined in TS25.101, Section A.7.1 [1]. This code also provides a useful template for full waveform customization.

The `umtsDownlinkWaveformGenerator` function can generate custom W-CDMA/HSPA/HSPA+ waveforms using the physical layer channels listed below. Arbitrary Coded Composite Transport Channels (CCTrCH) can be configured too. The output waveforms are loopable for continuous playback in simulation or via test equipment.

Physical channels supported:

- *Dedicated Physical Channel (DPCH)*
- *Primary Common Pilot Channel (P-CPICH)*
- *Secondary Common Pilot Channel (S-CPICH)*
- *Primary Common Control Physical Channel (P-CCPCH)*
- *Secondary Common Control Physical Channel (S-CCPCH)*
- *Primary Synchronization Channel (P-SCH)*
- *Secondary Synchronization Channel (S-SCH)*
- *Paging Indicator Channel (PICH)*
- *High Speed Physical Downlink Shared Channel (HS-PDSCH)*
- *Shared Control Channel for HS-DSCH (HS-SCCH)*
- *Orthogonal Channel Noise Simulator channels (OCNS)*

Transport channels supported:

- *Dedicated Channel (DCH)*
- *Broadcast Channel (BCH)*
- *Forward Access Channel (FACH)*
- *Paging Channel (PCH)*
- *High Speed Downlink Shared Channel (HS-DSCH)*

The physical channel processing is defined in TS25.211 and TS25.213 [2][4]. The processing for the transport channels is defined in TS25.212 [3].

The waveforms generated can be used for a number of applications:

- *Golden reference for transmitter implementations*
- *Receiver testing and algorithm development*
- *Testing RF hardware and software*
- *Interference testing*

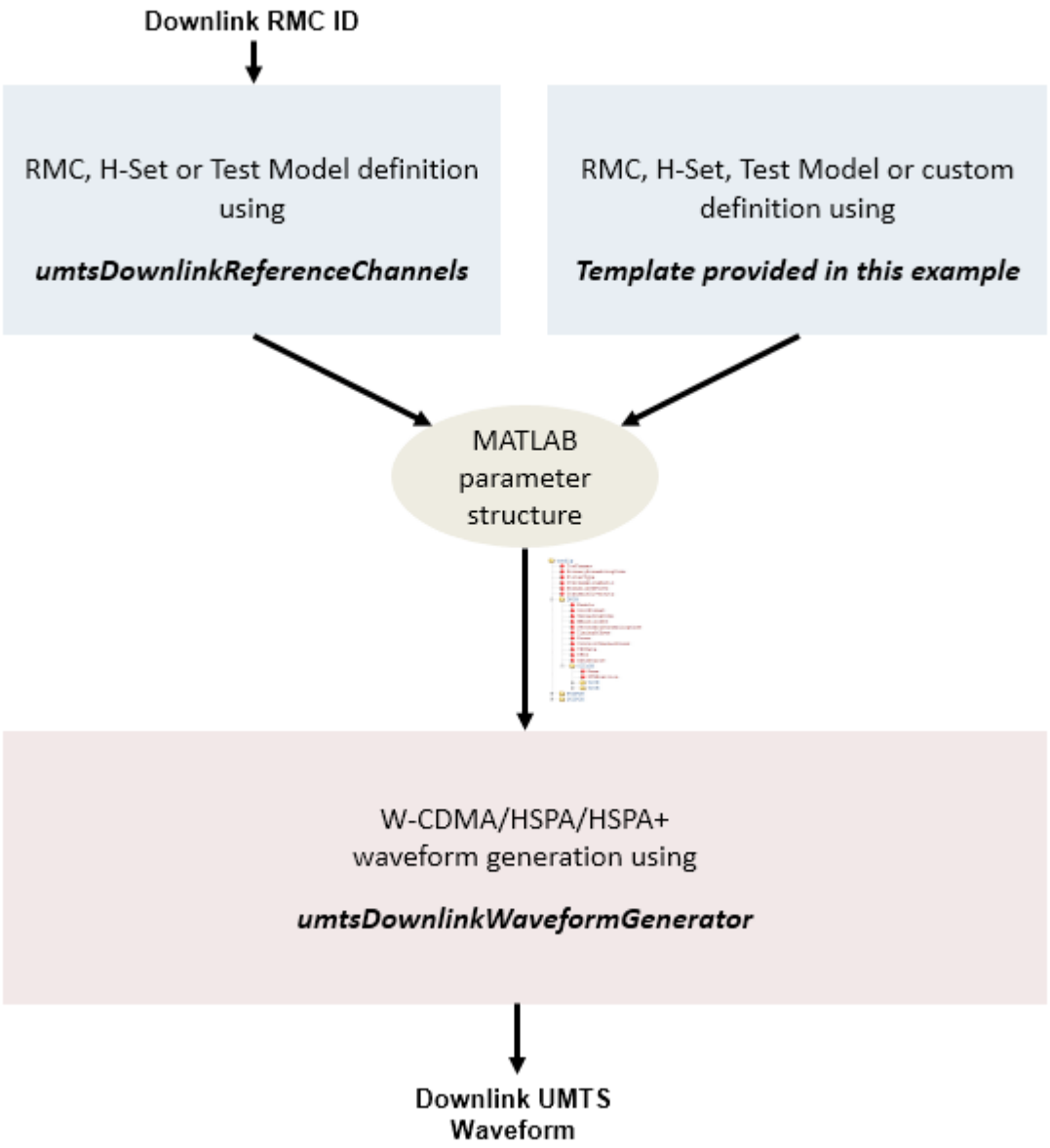
See “Waveform Generation and Transmission Using LTE Toolbox with Test and Measurement Equipment” on page 2-632 for a more detailed explanation of how to interface the waveforms with external hardware.

W-CDMA/HSPA/HSPA+ Waveform Generation and Parameterization Functions

The waveform generator function `umtsDownlinkWaveformGenerator` requires a single hierarchical MATLAB structure which specifies the set of all parameters for the physical and transport channels present in the output waveform.

The toolbox includes a function `umtsDownlinkReferenceChannels`, which can return a fully populated parameter structure for all the pre-configured Reference Measurement Channels (RMC), Fixed Reference Channel (FRC) H-Sets and Test Models (TM).

By combining the two functions these standard defined measurement waveforms can be generated easily. The pre-configured parameters returned from `umtsDownlinkReferenceChannels` can also be used as a starting point for parameter customization, for example changing the output filtering, channel power levels or even the reference CCTrCH configuration, prior to calling the generator function. If full waveform parameter control is required then this example includes MATLAB code below which lists all possible downlink parameters. The following diagram shows these steps.



H-Set1 (QPSK) Generation Using a Pre-configured Parameter Structure

The `umtsDownlinkReferenceChannels` function requires the H-Set number and modulation to be specified as shown below. Allowed H-Set values are ('H-Set1', 'H-Set2', 'H-Set3', 'H-Set4', 'H-Set5', 'H-Set6', 'H-Set7', 'H-Set8', 'H-Set10', 'H-Set12') and the choices for modulation schemes are 'QPSK', '16QAM' and '64QAM'. The output structure `preconfigParams` is the pre-built configuration for FRC H-Set1 and this can then be used to generate the standard defined H-Set waveform by calling the `umtsDownlinkWaveformGenerator` function.

```
hset = 'H-Set1';     % H-Set number
modulation = 'QPSK'; % Modulation scheme
preconfigParams = umtsDownlinkReferenceChannels(hset,modulation); % Get H-Set parameters
frcWaveform = umtsDownlinkWaveformGenerator(preconfigParams);     % Generate H-Set waveform
```

H-Set1 (QPSK) Generation Using Full Parameter List

In this section, we will build the H-Set1 (QPSK) configuration structure from scratch and show that this is identical to the structure defined using the `umtsDownlinkReferenceChannels` function as shown above. The `downlinkParams` structure defined below has the full list of the parameters supported by the `umtsDownlinkWaveformGenerator` function and so also can be used as a template to create custom waveforms when a large set of parameter values need to be changed from the structure output by `umtsDownlinkReferenceChannels`.

```
% H-Set parameter structure definition from scratch
% General settings
downlinkParams.TotFrames = 1;                       % Number of 10ms frames to be generated
downlinkParams.PrimaryScramblingCode = 0;           % Primary scrambling code
downlinkParams.FilterType = 'RRC';                  % Enable the RRC filter
downlinkParams.OversamplingRatio = 4;               % Oversampling set to 4
downlinkParams.NormalizedPower = 'Off';             % Power normalization disabled

% Define Downlink Dedicated Physical Channel  (DPCH)
downlinkParams.DPCH.Enable = 'On';                  % Enable DPCH
downlinkParams.DPCH.SlotFormat = 11;                % DPCH slot format
downlinkParams.DPCH.SpreadingCode = 6;              % DPCH spreading code
downlinkParams.DPCH.NMulticodes = 1;                % Number of DPCH
downlinkParams.DPCH.SecondaryScramblingCode = 1;    % Secondary scrambling code
downlinkParams.DPCH.TimingOffset = 0;               % Timing Offset
downlinkParams.DPCH.Power = 0;                      % Power in dB
downlinkParams.DPCH.TPCData = 0;                    % TPC value
downlinkParams.DPCH.TFCI = 0;                       % TFCI value
downlinkParams.DPCH.DataSource = 'CCTrCH';          % DPCH data source is CCTrCH
% DPCH carries the Coded Composite Transport Channel (CCTrCH) containing
% one or more transport channels. Since DPCH source is specified as CCTrCH,
% define the CCTrCH containing DTCH and DCCH transport channels
% Build DTCH definition
TrCH(1).Name = 'DTCH';                              % Name of the transport channel
TrCH(1).CRC = '16';                                 % CRC type
TrCH(1).TTI = 20;                                   % TTI in ms
TrCH(1).CodingType = 'conv3';                       % The coding type and rate
TrCH(1).RMA = 256;                                  % Rate matching attribute
TrCH(1).DataSource = 'PN9-ITU';                     % Tr channel data source
TrCH(1).ActiveDynamicPart = 1;                      % Index to active dynamic part
TrCH(1).DynamicPart(1) = struct('BlockSize',244,'BlockSetSize',244); % 1x244 blocks
% Build DCCH definition
TrCH(2).Name = 'DCCH';                              % Name of the transport channel
TrCH(2).CRC = '12';                                 % CRC type
TrCH(2).TTI = 40;                                   % TTI in ms
TrCH(2).CodingType = 'conv3';                       % The coding type and rate
TrCH(2).RMA = 256;                                  % Rate matching attribute
TrCH(2).DataSource = 'PN9-ITU';                     % Tr channel data source
TrCH(2).ActiveDynamicPart = 1;                      % Index to active dynamic part
TrCH(2).DynamicPart(1) = struct('BlockSize',100,'BlockSetSize',100);   % 1x100 blocks
% Finalize CCTrCH structure array using the TrCH structures defined above
downlinkParams.DPCH.CCTrCH.Name = 'DCH';            % Name of the CCTrCH
downlinkParams.DPCH.CCTrCH.DTXPosition = 'fixed';  % DTX position
downlinkParams.DPCH.CCTrCH.TrCH = TrCH;             % Assign DTCH/DCCH to CCTrCH

% Define P-CCPCH
downlinkParams.PCCPCH.Enable = 'On';                % Enable P-CCPCH
downlinkParams.PCCPCH.Power = 0;                    % Set power to be 0dB
```

```
downlinkParams.PCCPCH.DataSource = 'CCTrCH';       % P-CCPCH data source is CCTrCH
% P-CCPCH CCTrCH carries the BCH transport channel. Since P-CCPCH source is
% CCTrCH, define CCTrCH containing BCH
clear TrCH;
TrCH(1).Name = 'BCH';                              % Name of the Tr channel
TrCH(1).CRC = '16';                                % CRC type
TrCH(1).TTI = 20;                                  % TTI in ms
TrCH(1).CodingType = 'conv2';                      % The coding type and rate
TrCH(1).RMA = 256;                                 % Rate matching attribute
TrCH(1).DataSource = 'PN9-ITU';                    % Tr channel data source
TrCH(1).ActiveDynamicPart = 1;                     % Index to active dynamic part
TrCH(1).DynamicPart(1) = struct('BlockSize',246,'BlockSetSize',246); % 1x246 block
% Finalize CCTrCH structure array using the TrCH structure defined above
downlinkParams.PCCPCH.CCTrCH.Name = 'BCH';         % Name of the CCTrCH
downlinkParams.PCCPCH.CCTrCH.DTXPosition = 'fixed';% DTX position
downlinkParams.PCCPCH.CCTrCH.TrCH = TrCH;          % Assign BCH to CCTrCH

% Define S-CCPCH, but this channel is not required for H-Set1 generation
downlinkParams.SCCPCH.Enable = 'Off';              % Disable S-CCPCH
downlinkParams.SCCPCH.SlotFormat = 7;              % Slot format number
downlinkParams.SCCPCH.SpreadingCode = 3;           % S-CCPCH spreading code
downlinkParams.SCCPCH.SecondaryScramblingCode = 3; % Secondary scrambling code
downlinkParams.SCCPCH.TimingOffset = 0;            % Timing Offset
downlinkParams.SCCPCH.Power = 0;                   % Power in dB
downlinkParams.SCCPCH.TFCI = 0;                    % TFCI value
downlinkParams.SCCPCH.DataSource = 'CCTrCH';       % S-CCPCH data source is CCTrCH
% S-CCPCH CCTrCH can carry PCH and FACH transport channels. Since S-CCPCH
% source is CCTrCH, define CCTrCH containing PCH and FACH
% Build PCH definition
TrCH(1).Name = 'PCH';                              % Name of the Tr channel
TrCH(1).CRC = '16';                                % CRC type
TrCH(1).TTI = 10;                                  % TTI in ms
TrCH(1).CodingType = 'conv2';                      % The coding type
TrCH(1).RMA = 256;                                 % Rate matching attribute
TrCH(1).DataSource = 'PN9-ITU';                    % Tr channel data source
TrCH(1).ActiveDynamicPart = 1;                     % Index to active dynamic part
TrCH(1).DynamicPart(1) = struct('BlockSize',64,'BlockSetSize',64);   % 1x64 block
% Build FACH definition
TrCH(2).Name = 'FACH';                             % Name of the Tr channel
TrCH(2).CRC = '16';                                % CRC type
TrCH(2).TTI = 10;                                  % TTI in ms
TrCH(2).CodingType = 'turbo';                      % The coding type
TrCH(2).RMA = 256;                                 % Rate matching attribute
TrCH(2).DataSource = 'PN9-ITU';                    % Tr channel data source
TrCH(2).ActiveDynamicPart = 1;                     % Index to active dynamic part
TrCH(2).DynamicPart(1) = struct('BlockSize',360,'BlockSetSize',360);   % 1x360 block
% Finalize CCTrCH using the above
downlinkParams.SCCPCH.CCTrCH.Name = '';            % Name of the CCTrCH
downlinkParams.SCCPCH.CCTrCH.DTXPosition = 'fixed';% DTX position
downlinkParams.SCCPCH.CCTrCH.TrCH = TrCH;          % Assign PCH/FACH to CCTrCH

% Define P-CPICH
downlinkParams.PCPICH.Enable = 'On';               % Enable P-CPICH
downlinkParams.PCPICH.Power = 0;                   % Power in dB

% Define S-CPICH
downlinkParams.SCPICH.Enable = 'Off';              % Disable S-CPICH
downlinkParams.SCPICH.SpreadingCode = 4;           % S-CPICH spreading code
```

```
downlinkParams.SCPICH.SecondaryScramblingCode = 4; % Secondary scrambling code
downlinkParams.SCPICH.Power = 0;                   % Power in dB

% Define P-SCH
downlinkParams.PSCH.Enable = 'On';                 % Enable P-SCH
downlinkParams.PSCH.Power = 0;                     % Power in dB

% Define S-SCH
downlinkParams.SSCH.Enable = 'On';                 % Enable S-SCH
downlinkParams.SSCH.Power = 0;                     % Power in dB

% Define PICH
downlinkParams.PICH.Enable = 'On';                 % Enable PICH
downlinkParams.PICH.SpreadingCode = 16;            % PICH spreading code
downlinkParams.PICH.TimingOffset = 0;              % Timing offset
downlinkParams.PICH.Power = 0;                     % Power in dB
downlinkParams.PICH.DataSource = 'PagingData';     % PICH data source
downlinkParams.PICH.Np = 144;                      % Number of paging indicators

% Define HSDPA
downlinkParams.HSDPA.Enable = 'On';                % Enable HSDPA channels
downlinkParams.HSDPA.CodeGroup = 5;                % Number of HS-PDSCHs
downlinkParams.HSDPA.CodeOffset = 1;               % Code offset to first HS-PDSCH
downlinkParams.HSDPA.Modulation = 'QPSK';          % Modulation scheme
downlinkParams.HSDPA.VirtualBufferCapacity = 9600; % Buffer capacity
downlinkParams.HSDPA.InterTTIDistance = 3;         % Inter TTI interval
downlinkParams.HSDPA.NHARQProcesses = 2;           % Number of HARQ processes
downlinkParams.HSDPA.XrvSequence = [0 2 5 6];      % The XRV sequence
downlinkParams.HSDPA.UEId = 0;                     % UE Identity
downlinkParams.HSDPA.TransportBlockSizeId = 41;    % The transport block size id
downlinkParams.HSDPA.HSSCCHSpreadingCode = 9;      % Shared channel spreading code
downlinkParams.HSDPA.SecondaryScramblingCode = 6;  % Secondary scrambling code
downlinkParams.HSDPA.HSPDSCHPower = 0;             % HS-PDSCH power in dB
downlinkParams.HSDPA.HSSCCHPower = 0;              % HS-SCCH power in dB
downlinkParams.HSDPA.DataSource = 'HSDSCH';        % Data source is HS-DSCH
% HS-DSCH transport channel definition
downlinkParams.HSDPA.HSDSCH.BlockSize = 3202;      % The transport block size
downlinkParams.HSDPA.HSDSCH.DataSource = 'PN9-ITU';% HS-DSCH data source

% Define OCNS channels as defined in TS25.101 Table C.13
downlinkParams.OCNS.Enable = 'On';                 % Enable OCNS channels
downlinkParams.OCNS.Power = 0;                     % OCNS power scaling in dB
downlinkParams.OCNS.OCNSType = 'H-Set_6DPCH';      % OCNS definition

% The structure defined above can be used to generate the waveform:
frcWaveform2 = umtsDownlinkWaveformGenerator(downlinkParams);

% For completeness we can see that the H-Set definition structures obtained
% by the above two parameterization approaches are identical
if(isequal(preconfigParams,downlinkParams))
    disp(['H-Set1 configuration structures generated with and without using' ...
    ' umtsDownlinkReferenceChannels function are the same.']);
end
```

```
H-Set1 configuration structures generated with and without using umtsDownlinkReferenceChannels fu
```

Waveform Comparison

Compare the waveforms generated using both approaches described above and see that the generated waveforms are identical

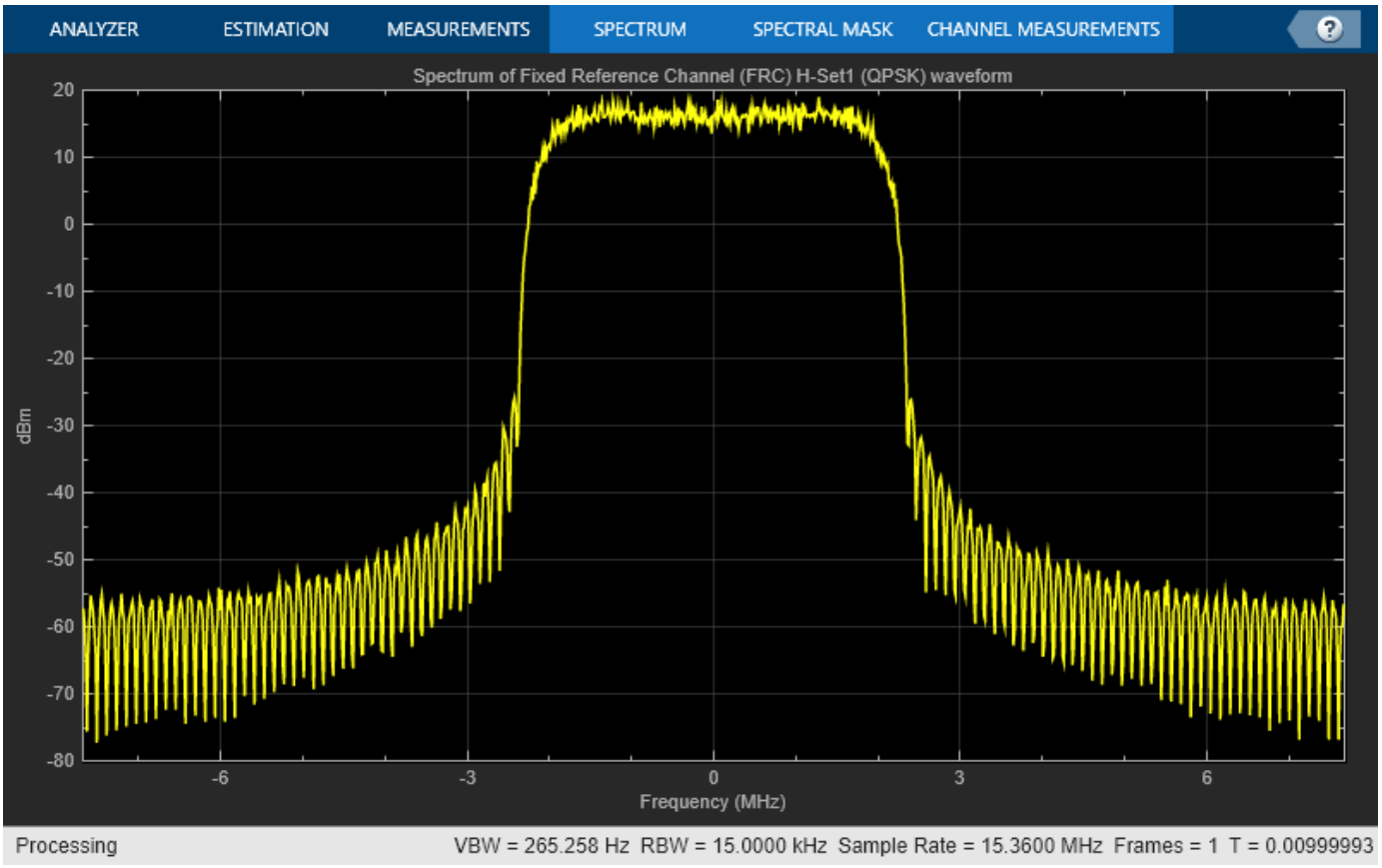
```
if(isequal(frcWaveform,frcWaveform2))
    disp(['H-Set1 waveforms generated with and without using' ...
    ' umtsDownlinkReferenceChannels function are the same.']);
end
```

```
H-Set1 waveforms generated with and without using umtsDownlinkReferenceChannels function are the
```

Plot Spectrum

Plot the spectrum of the time domain signal `frcWaveform`.

```
chiprate = 3.84e6;   % Chip rate of the baseband waveform
spectrumPlot = spectrumAnalyzer(SampleRate=chiprate*downlinkParams.OversamplingRatio);
spectrumPlot.Title = sprintf('Spectrum of Fixed Reference Channel (FRC) %s (%s) waveform', hset,
spectrumPlot(frcWaveform);
```



Conclusion

This example shows how to generate standard defined and custom W-CDMA/HSPA/HSPA+ waveforms using LTE Toolbox functions. The example also provides a full parameter template for full user customization of the pre-defined waveform configurations.

Selected Bibliography

1. 3GPP TS 25.101 "User Equipment (UE) radio transmission and reception (FDD)"
2. 3GPP TS 25.211 "Physical channels and mapping of transport channels onto physical channels (FDD)"
3. 3GPP TS 25.212 "Multiplexing and channel coding (FDD)"
4. 3GPP TS 25.213 "Spreading and modulation (FDD)"

UMTS Uplink Waveform Generation

This example shows how to generate an HSUPA FRC using LTE Toolbox™.

Introduction

The LTE Toolbox can be used to generate standard compliant W-CDMA/HSPA/HSPA+ uplink and downlink complex baseband waveforms including pre-defined configurations for standard defined measurement channels. For the uplink this includes the Reference Measurement Channels (RMC) and Fixed Reference Channel (FRC) defined in TS25.141 [1].

This example demonstrates how the two uplink related functions, `umtsUplinkReferenceChannels` and `umtsUplinkWaveformGenerator`, combine to support this feature. We show how they can generate an FRC waveform for HSUPA testing using one of the pre-defined configurations provided. We also present explicit MATLAB® code which lists all uplink generator parameters set up for this particular measurement channel. The FRCs are defined in TS25.141, Annex A.10 [1]. This code also provides a useful template for full waveform customization.

The `umtsUplinkWaveformGenerator` function can generate custom W-CDMA/HSPA/HSPA+ waveforms using the physical layer channels listed below. Arbitrary Coded Composite Transport Channels (CCTrCH) can be configured too. The output waveforms are loopable for continuous playback in simulation or via test equipment.

Physical channels supported:

- Dedicated Physical Data Channel (DPDCH)
- Dedicated Physical Control Channel (DPCCH)
- E-DCH Dedicated Physical Data Channel (E-DPDCH)
- E-DCH Dedicated Physical Control Channel (E-DPCCH)
- Dedicated Control Channel associated with HS-DSCH transmission (HS-DPCCH)

Transport channels supported:

- Dedicated Channel (DCH)
- Enhanced Dedicated Channel (E-DCH)

The physical channel processing is defined in TS25.211 and TS25.213 [2][4]. The processing for transport channels is defined in TS25.212 [3].

The waveforms generated can be used for a number of applications:

- *Golden reference for transmitter implementations*
- *Receiver testing and algorithm development*
- *Testing RF hardware and software*

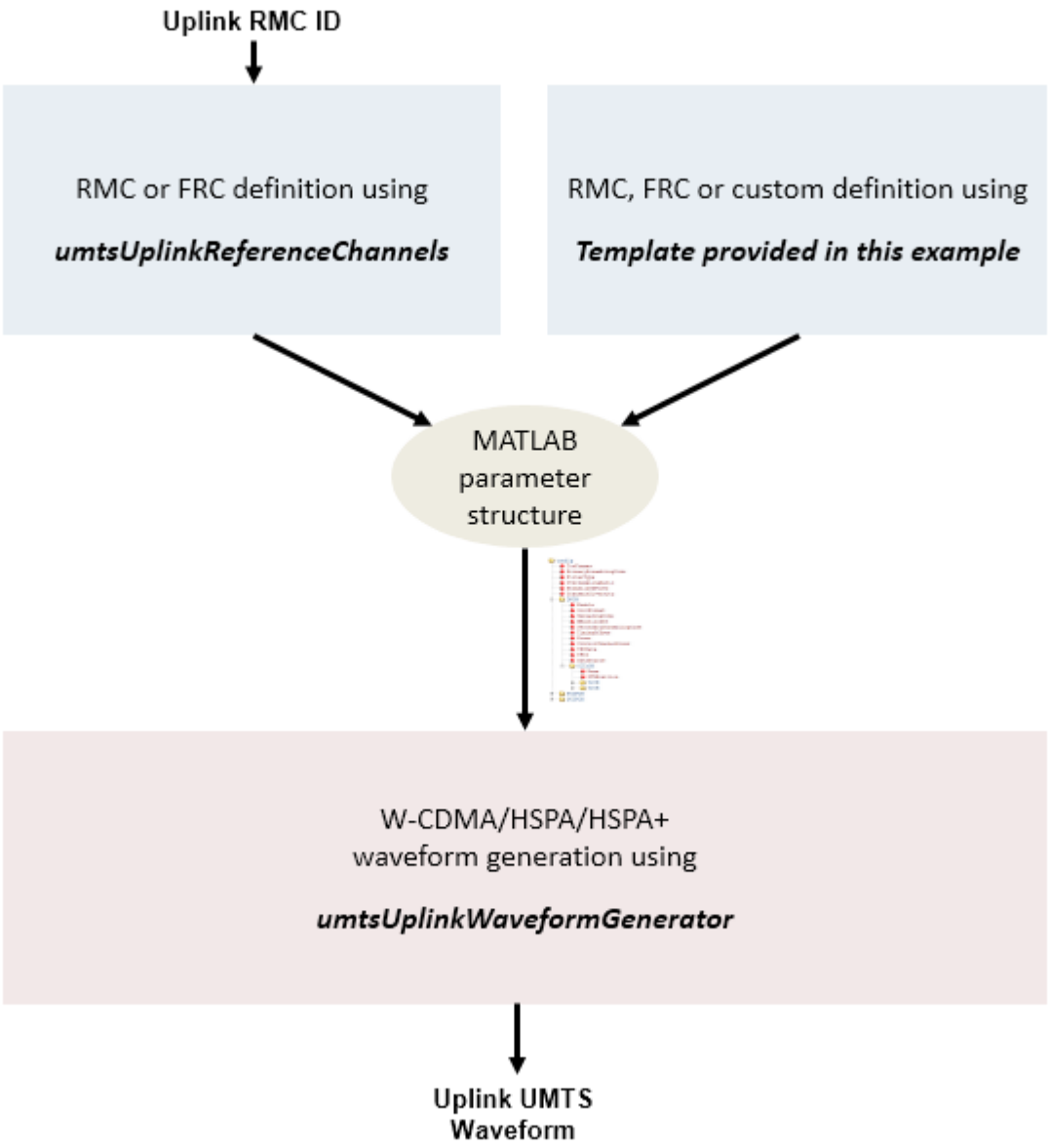
See “Waveform Generation and Transmission Using LTE Toolbox with Test and Measurement Equipment” on page 2-632 for a more detailed explanation of how to interface the waveforms with external hardware.

W-CDMA/HSPA/HSPA+ Waveform Generation and Parameterization Functions

The waveform generator function `umtsUplinkWaveformGenerator` requires a single hierarchical MATLAB structure which specifies the set of all parameters for the physical and transport channels present in the output waveform.

The toolbox includes a function `umtsUplinkReferenceChannels` which can return a fully populated parameter structure for all the pre-configured Reference Measurement Channels (RMC) and Fixed Reference Channels (FRC).

By combining the two functions these standard defined measurement waveforms can be generated easily. The pre-configured parameters returned from `umtsUplinkReferenceChannels` can also be used as a starting point for parameter customization, for example changing the output filtering, channel power levels or even the reference CCTrCH configuration, prior to calling the generator function. If full waveform parameter control is required then this example includes MATLAB code below which lists all possible uplink parameters. The following diagram shows the steps.



FRC1 Generation Using a Pre-configured Parameter Structure

The `umtsUplinkReferenceChannels` function requires the FRC number to be specified as shown below. Allowed FRC values are 'FRC1', 'FRC2', 'FRC3', 'FRC4', 'FRC5', 'FRC6', 'FRC7', and 'FRC8'. The output structure `preconfigParams` is the pre-built configuration for FRC1 and this can then be used to generate the standard defined FRC waveform by calling the `umtsUplinkWaveformGenerator` function.

```
frc = 'FRC1';     % FRC number
preconfigParams = umtsUplinkReferenceChannels(frc);        % Get FRC parameters
frcWaveform = umtsUplinkWaveformGenerator(preconfigParams);% Generate FRC waveform
```

FRC Definition Using Full Parameter List

In this section, we will build the FRC1 configuration structure from scratch and show that this is identical to the structure defined using the `umtsUplinkReferenceChannels` function as shown above. The `uplinkParams` structure defined below has the full list of the parameters supported by the `umtsUplinkWaveformGenerator` function and so can also be used as a template to create custom waveforms when a large set of parameter values need to be changed from the structure output by `umtsUplinkReferenceChannels`.

```
% FRC definition from scratch
% General settings
uplinkParams.TotFrames = 1;                     % Number of frames to be generated
uplinkParams.ScramblingCode = 1;                % Scrambling code
uplinkParams.FilterType = 'RRC';                % Enable the RRC filter
uplinkParams.OversamplingRatio = 4;             % Oversampling set to 4
uplinkParams.NormalizedPower = 'Off';           % No power normalization

% Define Uplink Dedicated Physical Data Channel (DPDCH)
uplinkParams.DPDCH.Enable = 'On';               % Enable DPDCH
uplinkParams.DPDCH.SlotFormat = 2;              % DPDCH slot format
uplinkParams.DPDCH.CodeCombination = 64;        % DPDCH spreading factor
uplinkParams.DPDCH.Power = 0;                   % Power in dB
uplinkParams.DPDCH.DataSource = 'CCTrCH';       % DPDCH data source is CCTrCH
% DPDCH carries the Coded Composite Transport Channel (CCTrCH) containing
% one or more transport channels. Since DPDCH source is specified as
% CCTrCH, define the CCTrCH containing DTCH and DCCH transport channels
% Build DTCH definition
TrCH(1).Name = 'DTCH';                          % Name of the transport channel
TrCH(1).CRC = '16';                             % CRC type
TrCH(1).CodingType = 'conv3';                   % The coding type and rate
TrCH(1).RMA = 256;                              % Rate matching attribute
TrCH(1).TTI = 20;                               % TTI in ms
TrCH(1).DataSource = 'PN9-ITU';                 % Tr channel data source
TrCH(1).ActiveDynamicPart = 1;                  % Index to active dynamic part
TrCH(1).DynamicPart(1) = struct('BlockSize',244,'BlockSetSize',244); % 1x244 blocks
% Build DCCH definition
TrCH(2).Name = 'DCCH';                          % Name of the transport channel
TrCH(2).CRC = '12';                             % CRC type
TrCH(2).CodingType = 'conv3';                   % The coding type and rate
TrCH(2).RMA = 256;                              % Rate matching attribute
TrCH(2).TTI = 40;                               % TTI in ms
TrCH(2).DataSource = 'PN9-ITU';                 % Tr channel data source
TrCH(2).ActiveDynamicPart = 1;                  % Index to active dynamic part
TrCH(2).DynamicPart(1) = struct('BlockSize',100,'BlockSetSize',100);   % 1x100 blocks
% Finalize CCTrCH structure array using the TrCH structures defined above
uplinkParams.DPDCH.CCTrCH.Name = 'DCH';         % Name of the CCTrCH
uplinkParams.DPDCH.CCTrCH.TrCH = TrCH;          % Assign DTCH/DCCH to CCTrCH

% Define DPCCH
uplinkParams.DPCCH.Enable = 'On';               % Enable DPCCH
uplinkParams.DPCCH.SlotFormat = 0;              % Slot format number
uplinkParams.DPCCH.Power = -5.46;               % Power in dB
uplinkParams.DPCCH.TPCData = 1;                 % TPC value
uplinkParams.DPCCH.TFCI = 0;                    % TFCI value
uplinkParams.DPCCH.FBIData = 0;                 % FBI value

% Define HSUPA channels
```

```
uplinkParams.HSUPA.Enable = 'On';                % Enable HSUPA channels
uplinkParams.HSUPA.CodeCombination = [4 4];      % E-DPDCH spreading factors
uplinkParams.HSUPA.EDPDCHPower = -5.46+12.04;    % Power in dB
uplinkParams.HSUPA.EDPCCHPower = -5.46+6.02;     % Power in dB
uplinkParams.HSUPA.RSNSequence = 0;              % RSN value
uplinkParams.HSUPA.ETFCI = 0;                    % E-TFCI value
uplinkParams.HSUPA.HappyBit = 0;                 % Happy Bit value
uplinkParams.HSUPA.DataSource = 'EDCH';          % Data source is E-DCH
uplinkParams.HSUPA.EDCH.BlockSize = 2706;        % E-DCH transport block size
uplinkParams.HSUPA.EDCH.TTI = 2;                 % E-DCH TTI in ms
uplinkParams.HSUPA.EDCH.Modulation = 'BPSK';     % Modulation scheme
uplinkParams.HSUPA.EDCH.DataSource = 'PN9-ITU'; % E-DCH Data source

% Define HS-DPCCH, but disable for FRC1 generation
uplinkParams.HSDPCCH.Enable = 'Off';             % Disable HS-DPCCH
uplinkParams.HSDPCCH.Power = 0;                  % Power in dB
uplinkParams.HSDPCCH.CQI = 0;                    % CQI value
uplinkParams.HSDPCCH.HARQACK = 1;                % HARQ ACK bit value
uplinkParams.HSDPCCH.UEMIMO = 0;                 % UE not in MIMO mode

% The structure defined above can be used to generate the waveform:
frcWaveform2 = umtsUplinkWaveformGenerator(uplinkParams);

% For completeness we can see that the FRC definition structures obtained
% by the above two parameterization approaches are identical
if(isequal(uplinkParams,preconfigParams))
    disp(['FRC1 definitions generated with and without using ' ...
    'umtsUplinkReferenceChannels function are the same.']);
end
```

```
FRC1 definitions generated with and without using umtsUplinkReferenceChannels function are the sa
```

Waveform Comparison

Compare the waveforms generated using both approaches described above and see that the generated waveforms are identical

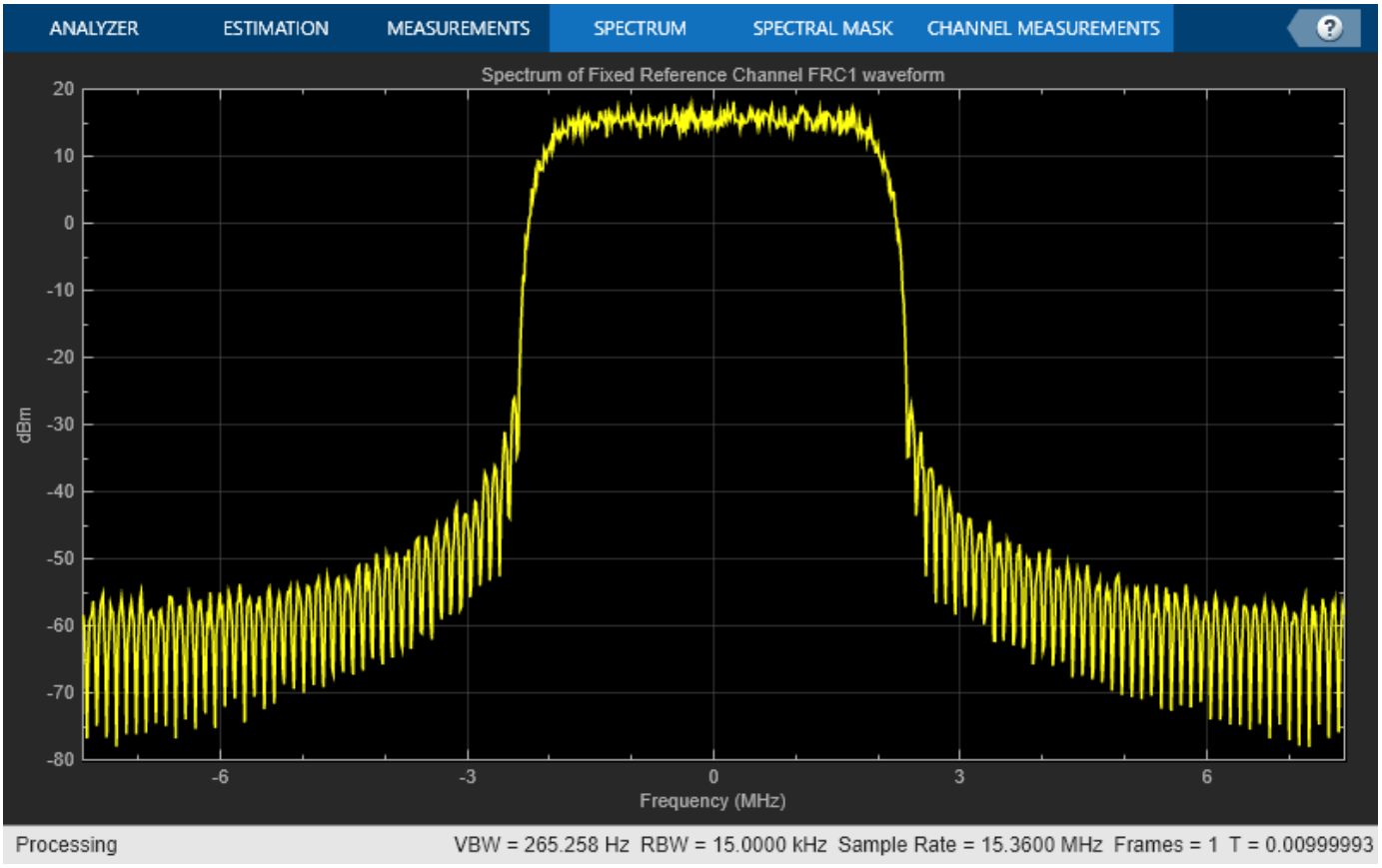
```
if(isequal(frcWaveform,frcWaveform2))
    disp(['FRC1 waveforms generated with and without using' ...
    ' umtsUplinkReferenceChannels function are the same.']);
end
```

```
FRC1 waveforms generated with and without using umtsUplinkReferenceChannels function are the same
```

Plot Spectrum

Plot the spectrum of the time domain signal `frcWaveform`.

```
chiprate = 3.84e6;   % Chip rate of the baseband waveform
spectrumPlot = spectrumAnalyzer(SampleRate=chiprate*uplinkParams.OversamplingRatio);
spectrumPlot.Title = sprintf('Spectrum of Fixed Reference Channel %s waveform', frc);
spectrumPlot(frcWaveform);
```



Selected Bibliography

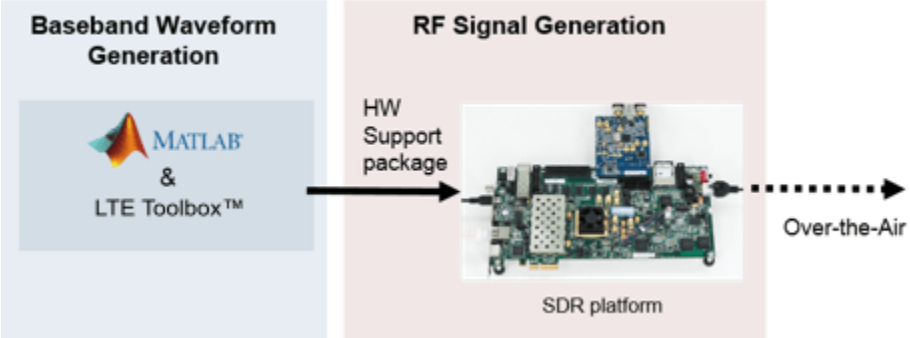
1. 3GPP TS 25.141 "Base Station (BS) conformance testing (FDD)"
2. 3GPP TS 25.211 "Physical channels and mapping of transport channels onto physical channels (FDD)"
3. 3GPP TS 25.212 "Multiplexing and channel coding (FDD)"
4. 3GPP TS 25.213 "Spreading and modulation (FDD)"

LTE Transmitter Using Software Defined Radio

This example shows how to generate a reference measurement channel (RMC) downlink (DL) LTE waveform suitable for over-the-air transmission. This example also shows how to use a software-defined radio (SDR) to transmit the generated waveform using single or multiple antennas.

Introduction

This example generates eight frames of a baseband RMC DL waveform. Using SDR hardware such as the Xilinx® Zynq®-Based Radio, this baseband waveform can be modulated for RF transmission. The SDR transmits the waveform by looping transmission of the eight frames for a specified time period.



This example supports these SDRs.

- ADALM-Pluto from the Communications Toolbox Support Package for Analog Devices® ADALM-Pluto Radio
- USRP™ E310/E312 from the Communications Toolbox Support Package for USRP™ Embedded Series Radio
- AD936x/FMCOMMS5 from the Communications Toolbox Support Package for Xilinx® Zynq®-Based Radio
- USRP™ N300/N310/N320/N321/B200/B210/X300/X310 from the Communications Toolbox Support Package for USRP™ Radio

Example Setup

Before running the example, ensure that you have installed the appropriate support package for the SDR that you intend to use and that you have configured the hardware.

The `TransmitOnSDR` field of the `txsim` structure determines whether the example transmits the generated waveform using an SDR.

```
txsim.TransmitOnSDR = ;
```

If you select the `TransmitOnSDR` field of the `txsim` structure, configure the variables required for SDR transmission.

```
if txsim.TransmitOnSDR
    txsim.SDRDeviceName = AD936x;       % SDR that is used for waveform transmis
```

```
    txsim.RunTime = 20;                % Time period to loop waveform in second
    txsim.RadioCenterFrequency = 2450000000; % Center frequency in Hz
    txsim.RadioIdentifier = 192.168.3.2;      % Value used to identify radio, for exam
end
```

Configure the other fields in the `txsim` structure for LTE downlink waveform generation.

```
txsim.RC = R.4;            % Base RMC configuration, 1.4 MHz bandwidth
txsim.NCellID = 17;        % Physical layer cell identity
txsim.NFrame = 700;        % Initial frame number
txsim.TotFrames = 8;       % Number of frames to generate
txsim.NumAntennas = 1; % Number of transmit antennas
```

Transmitter Design

Follow these steps to understand how the LTE transmitter functions.

1. Generate a baseband LTE signal.
2. Prepare the baseband signal for transmission using the SDR hardware.
3. Send the baseband data to the SDR hardware for upsampling and transmission at the desired center frequency.

Generate Baseband LTE Signal

The `lteRMCDLTool` function provides the default configuration parameters defined in 3GPP TS 36.101 Annex A.3, which are required to generate an RMC.

Customize the parameters within the configuration structure `rmc`.

```
rmc = lteRMCDL(txsim.RC);
rmc.NCellID = txsim.NCellID;
rmc.NFrame = txsim.NFrame;
rmc.TotSubframes = txsim.TotFrames*10; % 10 subframes per frame
rmc.CellRefP = txsim.NumAntennas;      % Configure number of cell-specific reference signal ante
rmc.OCNGPDSCHEnable = "On";            % Adds noise to unallocated PDSCH resource elements
```

If using two or more antennas, enable transmit diversity.

```
if rmc.CellRefP >= 2
    rmc.PDSCH.TxScheme = "TxDiversity";
    rmc.OCNGPDSCH.TxScheme = "TxDiversity";
else
    rmc.PDSCH.TxScheme = "Port0";
    rmc.OCNGPDSCH.TxScheme = "Port0";
end
rmc.PDSCH.NLayers = txsim.NumAntennas;
```

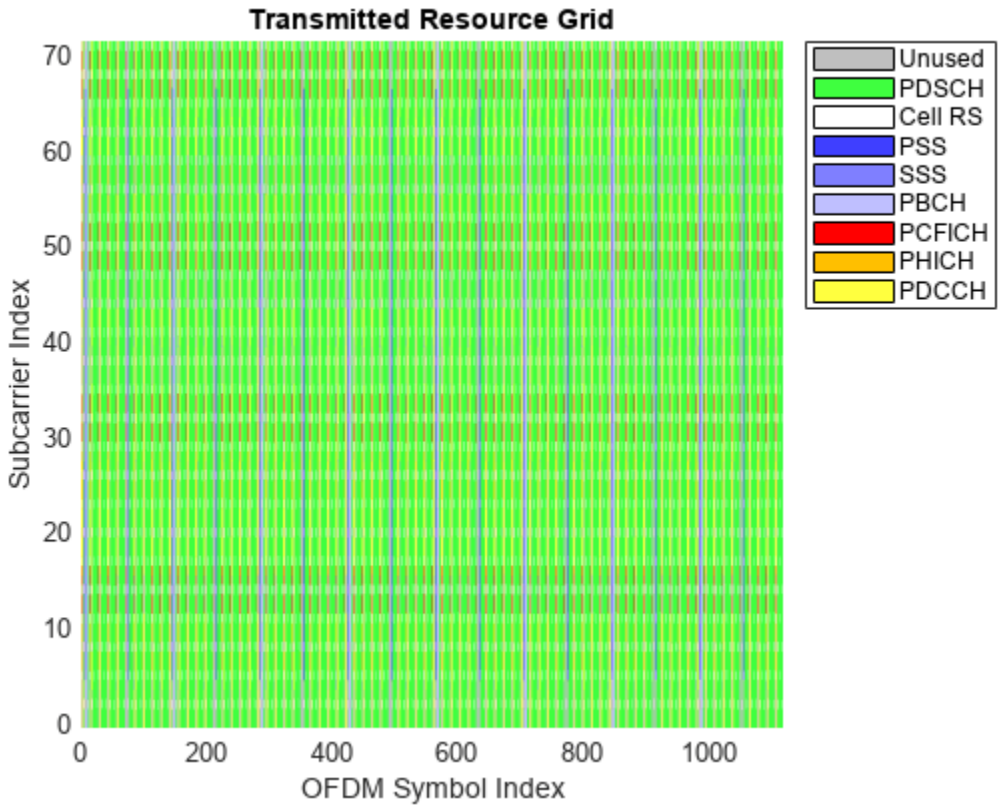
Create the baseband waveform (`eNodeBOutput`), a fully populated resource grid (`txGrid`), and the full configuration of the RMC using the `lteRMCDLTool` function.

```
trData = [1;0;0;1]; % Transport data
[eNodeBOutput,txGrid,rmc] = lteRMCDLTool(rmc,trData);
txsim.SamplingRate = rmc.SamplingRate;
```

Display the resource grid populated with the highlighted channels and the power spectral density of the LTE baseband signal. You can see a 1.4 MHz signal bandwidth at baseband in the spectrum plot.

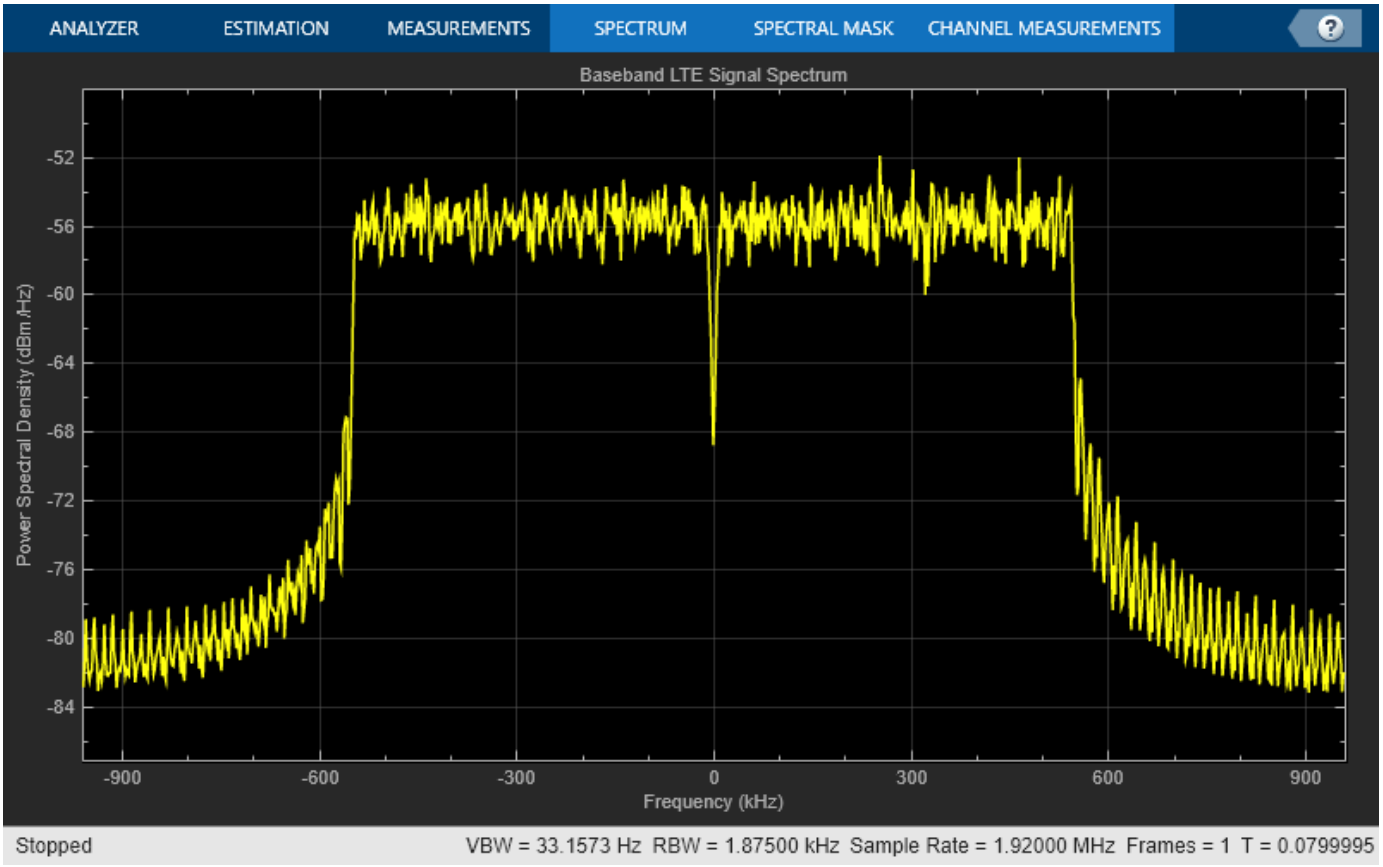
If using multiple transmit antennas, set `displayAntennaForGrid` to the antenna port you would like to display.

```
displayAntennaForGrid = 1;
ax = axes;
hPlotDLResourceGrid(rmc,txGrid(:,:,displayAntennaForGrid),ax,displayAntennaForGrid);
ax.Children(1).EdgeColor = "none";
title("Transmitted Resource Grid");
```



Display the power spectral density.

```
spectrumScope = spectrumAnalyzer( ...
    SampleRate=txsim.SamplingRate, ...
    SpectrumType="power-density", ...
    Title="Baseband LTE Signal Spectrum", ...
    YLabel="Power Spectral Density");
spectrumScope(eNodeBOutput);
release(spectrumScope);
```



Prepare for Transmission

The transmitter plays the LTE signal in a loop. The example splits the baseband signal into LTE frames of data, and the SDR Transmitter object (`sdrTransmitter`) transmits a full LTE frame. The example reshapes the baseband LTE signal into an *M*- by -*N* array, where *M* is the number of samples per LTE frame and *N* is the number of frames generated.

```
if txsim.TransmitOnSDR
    % Scale the signal for better power output and convert to int16, which
    % is the native format for the SDR hardware.
    powerScaleFactor = 0.7;
    eNodeBOutput = eNodeBOutput.*(1./max(abs(eNodeBOutput))*powerScaleFactor);
    eNodeBOutput = int16(eNodeBOutput*2^15);

    % LTE frames are 10 ms long
    samplesPerFrame = 10e-3*txsim.SamplingRate;
    numFrames = length(eNodeBOutput)/samplesPerFrame;

    % Ensure you are using an integer number of frames
    if mod(numFrames,1)
        warning("Non integer number of frames. Trimming transmission ...");
        numFrames = floor(numFrames);
    end

    % Reshape the baseband LTE data into frames for simpler transmission
    fprintf("Splitting transmission into %i frames\n",numFrames)
```

```
    txFrame = reshape(eNodeBOutput,samplesPerFrame,numFrames,txsim.NumAntennas);

    if matches(txsim.SDRDeviceName, ["AD936x", "FMCOMMS5", "Pluto", "E3xx"])
        sdrTransmitter = sdrtx( ...
            txsim.SDRDeviceName, ...
            CenterFrequency=txsim.RadioCenterFrequency, ...
            BasebandSampleRate=txsim.SamplingRate);
        if matches(txsim.SDRDeviceName, ["AD936x", "FMCOMMS5", "E3xx"])
            sdrTransmitter.ShowAdvancedProperties = true;
            sdrTransmitter.BypassUserLogic = true;
            sdrTransmitter.IPAddress = txsim.RadioIdentifier;
        else
            sdrTransmitter.RadioID = txsim.RadioIdentifier;
        end
    else
        % For the USRP SDRs
        sdrTransmitter = comm.SDRuTransmitter(...
            Platform=txsim.SDRDeviceName,...
            CenterFrequency=txsim.RadioCenterFrequency);
        [sdrTransmitter.MasterClockRate, sdrTransmitter.InterpolationFactor] = ...
            hGetUSRPRateInformation(txsim.SDRDeviceName,txsim.SamplingRate);
        if matches(txsim.SDRDeviceName, ["B200", "B210"])
            % Change the serial number as needed for USRP B200/B210
            sdrTransmitter.SerialNum = txsim.RadioIdentifier;
        else
            sdrTransmitter.IPAddress = txsim.RadioIdentifier;
        end
        sdrTransmitter.EnableBurstMode = true;
        sdrTransmitter.NumFramesInBurst = numFrames;
    end
    sdrTransmitter.ChannelMapping = 1:txsim.NumAntennas;
end
```

Transmission Using SDR Hardware

The example uses a `try` block to transfer the baseband data to the SDR hardware. Using the `try, catch` block means that if an error occurs during the transmission, the hardware releases the resources used by the SDR System object™. The `sdrTransmitter` System object transmits a full frame of LTE data.

```
if txsim.TransmitOnSDR
    fprintf("Starting transmission at Fs = %g MHz\n",txsim.SamplingRate/1e6)
    currentTime = 0;
    try
        while currentTime<txsim.RunTime
            for n = 1:numFrames
                bufferUnderflow = sdrTransmitter(squeeze(txFrame(:,n,:)));
                if bufferUnderflow
                    warning("Dropped samples.")
                end
            end
            currentTime = currentTime+numFrames*10e-3; % One frame is 10 ms
        end
    catch ME
        release(sdrTransmitter);
        rethrow(ME)
    end
    fprintf("Transmission finished\n")
```

```
    release(sdrTransmitter);
end
```

Further Exploration

- You can use the companion example “LTE Receiver Using Software Defined Radio” on page 2-668 to decode the broadcast channel of the waveform generated by this example. Try changing the cell identity and initial system frame number and observe the detected cell identity and frame number at the receiver.
- If using a supported multi-channel SDR, try increasing the number of antennas

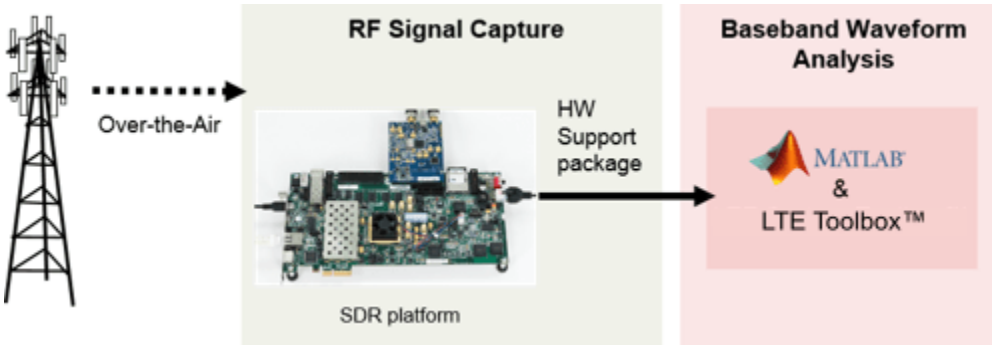
SDR Troubleshooting

- ADALM-PLUTO Radio “Common Problems and Fixes” (Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio)
- USRP Embedded Series Radio “Common Problems and Fixes” (Communications Toolbox Support Package for USRP Embedded Series Radio)
- Xilinx Zynq-Based Radio “Common Problems and Fixes” (Communications Toolbox Support Package for Xilinx Zynq-Based Radio)
- USRP Radio “Common Problems and Fixes” (Communications Toolbox Support Package for USRP Radio)

LTE Receiver Using Software Defined Radio

This example shows how to recover the master information block (MIB) and basic system information from an over-the-air LTE downlink (DL) waveform. This example also shows how to receive an LTE waveform using a software-defined radio (SDR) with single or multiple antennas.

Introduction



In LTE, the broadcast channel (BCH) carries the MIB. The MIB provides basic cell-wide settings, including the system bandwidth and frame number. This example decodes the MIB from a burst of captured frames, and then decodes the control format indicator (CFI) for each subframe, which informs the user equipment (UE) of the control region size. The example then continues to decode the physical downlink control channel (PDCCH). A UE requires these steps to be associated with a cell. For an example showing the procedure for cell search and system information acquisition, see “Cell Search, MIB and SIB1 Recovery” on page 2-351.

An LTE transmission contains the BCH in the middle six resource blocks (RBs). Therefore, the captured waveform bandwidth needs to be only 1.92 MHz to decode the MIB, regardless of the cell bandwidth. Even though the MIB exists only in subframe 0 of a frame, this example demodulates each entire frame for visualization and analysis.

This example supports these SDRs for LTE waveform capture.

- ADALM-Pluto from the Communications Toolbox Support Package for Analog Devices® ADALM-Pluto Radio
- USRP™ E310/E312 from the Communications Toolbox Support Package for USRP™ Embedded Series Radio
- AD936x/FMCOMMS5 from the Communications Toolbox Support Package for Xilinx® Zynq®-Based Radio
- USRP™ N300/N310/N320/N321/B200/B210/X300/X310 from the Communications Toolbox Support Package for USRP™ Radio

Alternatively, if an SDR is not available for waveform capture, the example supports importing a file with a precaptured waveform.

Example Setup

Before running the example, ensure that you have installed the appropriate support package for the SDR that you intend to use and that you have set up the hardware.

The `ReceiveOnSDR` field of the `rxsim` structure determines whether the example receives a waveform off the air or imports a waveform from a MAT file.

```
rxsim.ReceiveOnSDR = ;
```

If you use an SDR for reception of LTE waveforms and the SDR detects no LTE waveforms, you can generate and transmit an LTE waveform by using the “LTE Transmitter Using Software Defined Radio” on page 2-662.

The parameters defined in the `rxsim` structure control the receiver. The sample rate of the receiver is 1.92 MHz, which is the standard sample rate for capturing an LTE bandwidth of 6 RBs. 6 RBs equates to a signal bandwidth of 1.4 MHz. To capture more frames, increase the `rxsim.NumCaptures` parameter value. By default, the example captures five LTE frames with each execution.

Specify the file name of a precaptured waveofrm in the `fileName` variable. Confirm that the MAT file contains a `numCaptures` variable, a `radioSampleRate` variable, and a `capturedData` variable. Each column of `captureData` represents one capture antenna and the third dimension of `captureData` represents the number of captures.

```
fileName = "capturedLTERCDLWaveform.mat";
```

If you select the `ReceiveOnSDR` field of the `rxsim` structure, configure the variables required for SDR reception. If you use an SDR device capable of multi-channel reception, try increasing `rxsim.NumAntennas` to a value supported by your SDR.

```
if rxsim.ReceiveOnSDR
    rxsim.SDRDeviceName = AD936x;          % SDR that is used for waveform receptio
    rxsim.RadioIdentifier = 192.168.3.2;   % Value used to identify radio, for exam
    rxsim.RadioSampleRate = 1.92e6;        % Configured for 1.92e6 Hz capture bandw
    rxsim.RadioCenterFrequency = 2450000000; % Center frequency in Hz
    rxsim.FramesPerCapture = 8;            % Number of contiguous LTE frames to cap
    rxsim.NumCaptures = 1;                 % Number of captures for the SDR to perf
    rxsim.NumAntennas = 1;                 % Number of receive antennas

    % Derived parameter
    captureTime = (rxsim.FramesPerCapture + 1)* 10e-3; % Increase capture frame by 1 to account

else
    rx = load(fileName);

    rxsim.NumCaptures = rx.numCaptures;
    rxsim.RadioSampleRate = rx.radioSampleRate;

end
```

Receiver Design: System Architecture

Follow these steps to understand how the LTE receiver functions.

1. If using an SDR, capture a suitable number of frames of an LTE signal.
2. Determine and correct the frequency offset of the received signal.
3. Determine the cell identity by performing a blind cell search.
4. Synchronize the captured signal to the start of an LTE frame.
5. Extract an LTE resource grid by OFDM demodulating the received signal.
6. Perform a channel estimation for the received signal.
7. Determine the cell-wide settings by decoding the MIB for each captured frame.
8. Decode the CFI and PDCCH for each subframe within the captured signal.

This example plots the power spectral density of the captured waveform and shows visualizations of the received LTE resource grid, estimated channel, and equalized PBCH symbols for each frame.

Configure SDR Hardware

This example communicates with the radio hardware using the object pertaining to the selected radio. For example, the AD936x radio uses the `comm.SDRRxAD936x` (Communications Toolbox Support Package for Xilinx Zynq-Based Radio) object.

```
if rxsim.ReceiveOnSDR
    if matches(rxsim.SDRDeviceName, ["AD936x", "FMCOMMS5", "Pluto", "E3xx"])
        sdrReceiver = sdrrx( ...
            rxsim.SDRDeviceName, ...
            CenterFrequency=rxsim.RadioCenterFrequency, ...
            BasebandSampleRate=rxsim.RadioSampleRate);
        if matches(rxsim.SDRDeviceName, ["AD936x", "FMCOMMS5", "E3xx"])
            sdrReceiver.ShowAdvancedProperties = true;
            sdrReceiver.BypassUserLogic = true;
            sdrReceiver.IPAddress = rxsim.RadioIdentifier;
        else
            sdrReceiver.RadioID = rxsim.RadioIdentifier;
        end
    else
        % For the USRP SDRs
        sdrReceiver = comm.SDRuReceiver(...
            Platform=rxsim.SDRDeviceName,...
            CenterFrequency=rxsim.RadioCenterFrequency);
        [sdrReceiver.MasterClockRate, sdrReceiver.DecimationFactor] = ...
            hGetUSRPRateInformation(rxsim.SDRDeviceName,rxsim.RadioSampleRate);
        if matches(rxsim.SDRDeviceName, ["B200", "B210"])
            % Change the serial number as needed for USRP B200/B210
            sdrReceiver.SerialNum = rxsim.RadioIdentifier;
        else
            sdrReceiver.IPAddress = rxsim.RadioIdentifier;
        end
        sdrReceiver.EnableBurstMode = true;
        sdrReceiver.SamplesPerFrame = 10e-3*rxsim.RadioSampleRate;
        sdrReceiver.NumFramesInBurst = rxsim.FramesPerCapture + 1; % Increase capture frame by 1
    end
    sdrReceiver.OutputDataType = "double";
    numSamplesToCapture = ceil(captureTime*rxsim.RadioSampleRate);
    sdrReceiver.ChannelMapping = 1:rxsim.NumAntennas;
end
```

Set up the spectrum analyzer to display the received waveform.

```
spectrumScope = spectrumAnalyzer( ...
    SampleRate=rxsim.RadioSampleRate, ...
    SpectrumType="power-density", ...
    Title="Baseband LTE Signal Spectrum", ...
    YLabel="Power Spectral Density");
```

LTE Setup

The `enb` structure contains the parameters for decoding the MIB. The example assumes frequency division duplexing (FDD), a normal cyclic prefix length, and four cell-specific reference ports (`CellRefP`) for the MIB decoding as these values are unknown. The MIB provides the actual value of `CellRefP`.

```
enb.DuplexMode = "FDD";
enb.CyclicPrefix = "Normal";
enb.CellRefP = 4;
```

The sample rate of the signal controls the captured bandwidth and the number of RBs in the waveform. The example uses a lookup table to determine the number of RBs.

```
% Bandwidth: {1.4 MHz, 3 MHz, 5 MHz, 10 MHz, 20 MHz}
SampleRateLUT = [1.92 3.84 7.68 15.36 30.72]*1e6;
NDLRBLUT = [6 15 25 50 100];
enb.NDLRB = NDLRBLUT(SampleRateLUT==rxsim.RadioSampleRate);
if rxsim.ReceiveOnSDR
    fprintf("\nSDR hardware sampling rate configured to capture %d LTE RBs.\n",enb.NDLRB);
end
```

The `cec` structure controls the channel estimation parameters. To minimize the effect of noise on pilot estimates, use a conservative 9-by-9 pilot averaging window.

```
cec.FreqWindow = 9;               % Frequency averaging window in resource elements (REs)
cec.TimeWindow = 9;               % Time averaging window in REs
cec.InterpType = "Cubic";         % Cubic interpolation
cec.PilotAverage = "UserDefined"; % Pilot averaging method
cec.InterpWindow = "Centred";     % Interpolation windowing method
cec.InterpWinSize = 3;            % Interpolate up to 3 subframes simultaneously
```

Signal Capture and Processing

To capture and decode bursts of LTE frames, the example uses a `while` loop. For each captured frame, the example decodes the MIB, and if decoding is successful, it decodes the CFI and the PDCCH for each subframe. For each successfully decoded subframe, the example displays the channel estimate and equalized PDCCH symbols.

Set up the constellation diagram viewer for equalized PDCCH symbols and the figure handle for the channel estimate plots.

```
constellation = comm.ConstellationDiagram("Title","Equalized PDCCH Symbols") ;
channelEstimatePlot = figure("Visible","Off");
```

Store the default decoding parameters.

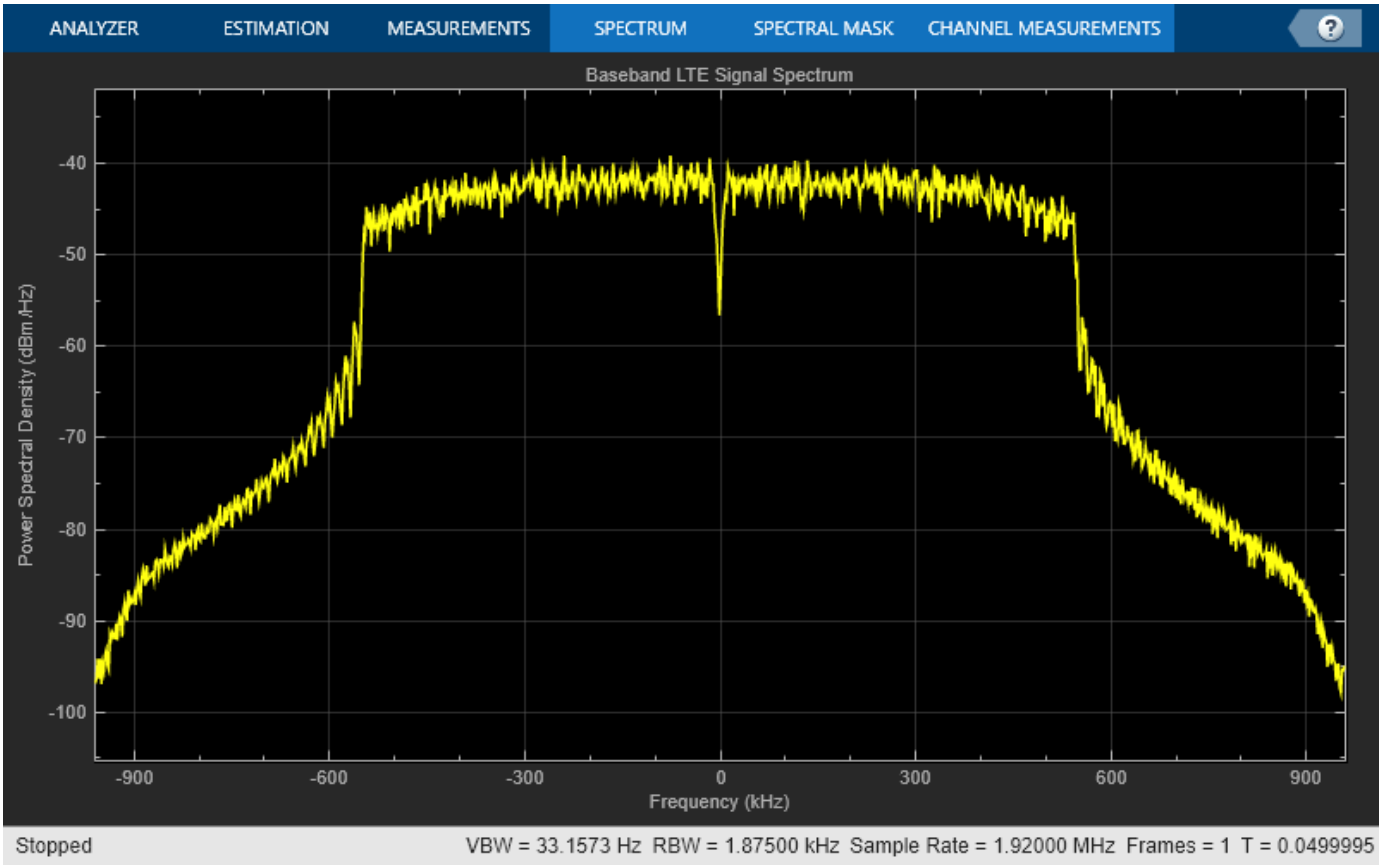
```
enbDefault = enb;
samplesPerFrame = 10e-3*rxsim.RadioSampleRate; % LTE frame period is 10 ms
```

Perform LTE receiver processing.

```
for i = 1:rxsim.NumCaptures
    % Set default LTE parameters
    enb = enbDefault;

    % rxWaveform holds |rxsim.FramesPerCapture| number of consecutive
    % frames worth of contiguous baseband LTE samples.
    if rxsim.ReceiveOnSDR
        % SDR Capture
        fprintf("\nStarting a new RF capture.\n")
        rxWaveform = captureWaveform(sdrReceiver,numSamplesToCapture);
    else
        rxWaveform = rx.capturedData(:,:,i);
    end

    % Show power spectral density of captured burst
    spectrumScope(rxWaveform);
    release(spectrumScope);

    % Perform frequency offset correction
    frequencyOffset = lteFrequencyOffset(enb,rxWaveform);
    rxWaveform = lteFrequencyCorrect(enb,rxWaveform,frequencyOffset);
    fprintf("Corrected a frequency offset of %g Hz.\n",frequencyOffset)

    % Perform the blind cell search to obtain cell identity and timing
    % offset Use "PostFFT" secondary synchronization signal (SSS) detection
    % method to improve speed
    cellSearch.SSSDetection = "PostFFT";
    cellSearch.MaxCellCount = 1;
    [NCellID,frameOffset] = lteCellSearch(enb,rxWaveform,cellSearch);
    fprintf("Detected a cell identity of %i.\n", NCellID);
    enb.NCellID = NCellID; % From lteCellSearch

    % Sync the captured samples to the start of an LTE frame, and trim off
    % any samples that are part of an incomplete frame.
    rxWaveform = rxWaveform(frameOffset+1:end,:);
    tailSamples = mod(length(rxWaveform),samplesPerFrame);
    rxWaveform = rxWaveform(1:end-tailSamples,:);
    enb.NSubframe = 0;

    % OFDM demodulation
    rxGrid = lteOFDMDemodulate(enb,rxWaveform);

    % Perform channel estimation
    [hest,nest] = lteDLChannelEstimate(enb,cec,rxGrid);

    sfDims = lteResourceGridSize(enb);
    Lsf = sfDims(2); % OFDM symbols per subframe
    LFrame = 10*Lsf; % OFDM symbols per frame
    numFullFrames = length(rxWaveform)/samplesPerFrame;

    % For each frame, decode the MIB and CFI
    for frame = 0:(numFullFrames-1)
        fprintf("\nPerforming MIB decode for frame %i of %i in burst...\n", ...
            frame+1,numFullFrames)

        % Extract subframe 0 from each frame of the received resource grid
        % and channel estimate.
        enb.NSubframe = 0;
```

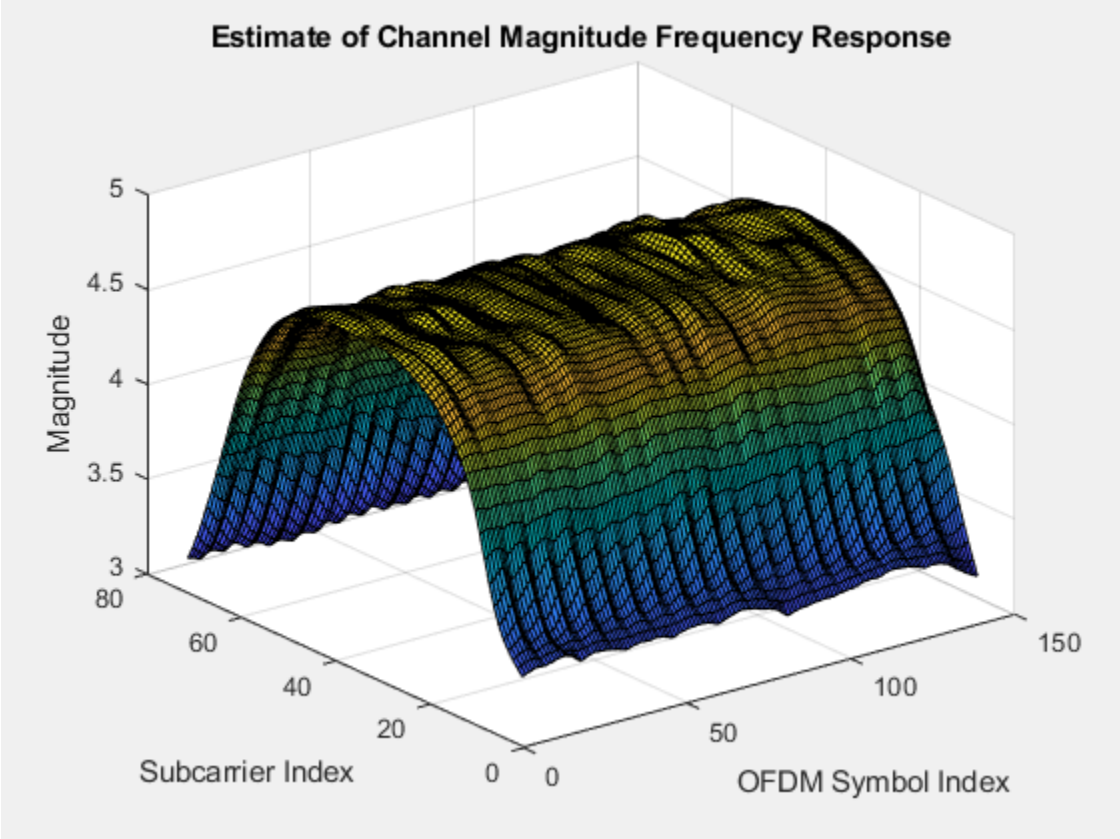
```
rxsf = rxGrid(:,frame*LFrame+(1:Lsf),:);
hestsf = hest(:,frame*LFrame+(1:Lsf),:,:);

% PBCH demodulation. Extract REs corresponding to the PBCH from the
% received grid and channel estimate grid for demodulation. Assume
% 4 cell-specific reference signals for PBCH decode as initially
% the value is unknown.
enb.CellRefP = 4;
pbchIndices = ltePBCHIndices(enb);
[pbchRx,pbchHest] = lteExtractResources(pbchIndices,rxsf,hestsf);
[~,~,nfmod4,mib,CellRefP] = ltePBCHDecode(enb,pbchRx,pbchHest,nest);

% If PBCH decoding is not successful, go to next iteration of for-
% loop
if ~CellRefP
    fprintf("  No PBCH detected for frame = %d.\n",frame);
    continue;
end

% With successful PBCH decoding, decode the MIB and obtain system
% information including system bandwidth
enb = lteMIB(mib,enb);
enb.CellRefP = CellRefP; % From ltePBCHDecode
% Incorporate the nfmod4 value output from the function
% ltePBCHDecode, as the NFrame value established from the MIB is
% the system frame number modulo 4.
enb.NFrame = enb.NFrame+nfmod4;
fprintf("  Successful MIB Decode.\n")
fprintf("  Frame number: %d.\n",enb.NFrame);

% The eNodeB transmission bandwidth can be greater than the
% captured bandwidth, so limit the bandwidth for processing
enb.NDLRB = min(enbDefault.NDLRB,enb.NDLRB);

% Process subframes within frame
for sf = 0:9
    % Extract subframe
    enb.NSubframe = sf;
    rxsf = rxGrid(:,frame*LFrame+sf*Lsf+(1:Lsf));

    % Perform channel estimation with the correct number of
    % CellRefP
    [hestsf,nestsf] = lteDLChannelEstimate(enb,cec,rxsf);

    % Physical CFI channel (PCFICH) demodulation Extract REs
    % corresponding to the PCFICH from the received grid and
    % channel estimate for demodulation.
    pcfichIndices = ltePCFICHIndices(enb);
    [pcfichRx,pcfichHest] = lteExtractResources(pcfichIndices,rxsf,hestsf);
    [cfiBits,recsym] = ltePCFICHDecode(enb,pcfichRx,pcfichHest,nestsf);

    % CFI decoding
    enb.CFI = lteCFIDecode(cfiBits);
    fprintf("    Subframe %d, decoded CFI value: %d.\n",sf,enb.CFI);

    % PDCCH demodulation. Extract REs corresponding to the PDCCH
    % from the received grid and channel estimate for demodulation.
    pdcchIndices = ltePDCCHIndices(enb);
```

```
                [pdcchRx,pdcchHest] = lteExtractResources(pdcchIndices,rxsf,hestsf);
                [pdcchBits,pdcchEq] = ltePDCCHDecode(enb,pdcchRx,pdcchHest,nestsf);

                release(constellation);
                constellation(pdcchEq);
            end

            % Plot channel estimate between CellRefP 0 and the receive antenna
            focalFrameIdx = frame*LFrame+(1:LFrame);
            figure(channelEstimatePlot);
            surf(abs(hest(:,focalFrameIdx,1,1)));
            xlabel("OFDM Symbol Index");
            ylabel("Subcarrier Index");
            zlabel("Magnitude");
            title("Estimate of Channel Magnitude Frequency Response");
        end
    end
```

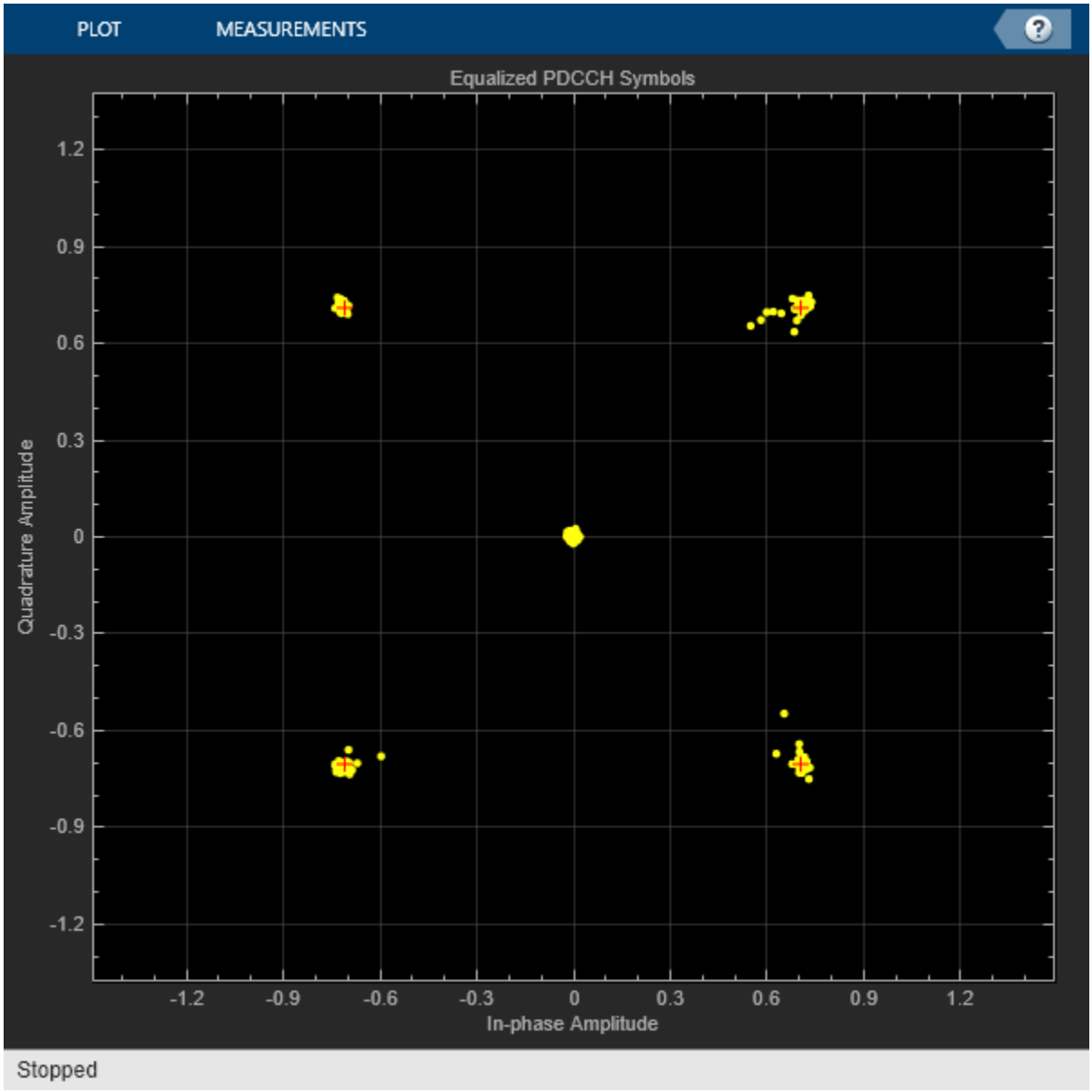



```
Corrected a frequency offset of -2.48294 Hz.

Detected a cell identity of 17.

Performing MIB decode for frame 1 of 4 in burst...

  Successful MIB Decode.

  Frame number: 705.
```

```
    Subframe 0, decoded CFI value: 3.
    Subframe 1, decoded CFI value: 3.
    Subframe 2, decoded CFI value: 3.
    Subframe 3, decoded CFI value: 3.
    Subframe 4, decoded CFI value: 3.
    Subframe 5, decoded CFI value: 3.
    Subframe 6, decoded CFI value: 3.
    Subframe 7, decoded CFI value: 3.
    Subframe 8, decoded CFI value: 3.
    Subframe 9, decoded CFI value: 3.

Performing MIB decode for frame 2 of 4 in burst...

  Successful MIB Decode.

  Frame number: 706.

    Subframe 0, decoded CFI value: 3.
    Subframe 1, decoded CFI value: 3.
    Subframe 2, decoded CFI value: 3.
    Subframe 3, decoded CFI value: 3.
    Subframe 4, decoded CFI value: 3.
    Subframe 5, decoded CFI value: 3.
    Subframe 6, decoded CFI value: 3.
    Subframe 7, decoded CFI value: 3.
    Subframe 8, decoded CFI value: 3.
    Subframe 9, decoded CFI value: 3.

Performing MIB decode for frame 3 of 4 in burst...

  Successful MIB Decode.

  Frame number: 707.

    Subframe 0, decoded CFI value: 3.
    Subframe 1, decoded CFI value: 3.
    Subframe 2, decoded CFI value: 3.
    Subframe 3, decoded CFI value: 3.
    Subframe 4, decoded CFI value: 3.
    Subframe 5, decoded CFI value: 3.
    Subframe 6, decoded CFI value: 3.
    Subframe 7, decoded CFI value: 3.
    Subframe 8, decoded CFI value: 3.
    Subframe 9, decoded CFI value: 3.

Performing MIB decode for frame 4 of 4 in burst...

  Successful MIB Decode.

  Frame number: 700.

    Subframe 0, decoded CFI value: 3.
    Subframe 1, decoded CFI value: 3.
    Subframe 2, decoded CFI value: 3.
    Subframe 3, decoded CFI value: 3.
    Subframe 4, decoded CFI value: 3.
    Subframe 5, decoded CFI value: 3.
    Subframe 6, decoded CFI value: 3.
    Subframe 7, decoded CFI value: 3.
    Subframe 8, decoded CFI value: 3.
    Subframe 9, decoded CFI value: 3.
```




```
if rxsim.ReceiveOnSDR
    release(sdrReceiver);
end
release(constellation);  % Release constellation diagram object
```



Further Exploration

- Transmit a standard compliant LTE waveform by using the “LTE Transmitter Using Software Defined Radio” on page 2-662. You can then decode the waveform using this example. In the companion example, try changing the cell identity and initial frame number, and observe the detected cell identity and frame number in this example. If using a supported multi-channel SDR, try increasing the number of antennas in the companion example to visualize the benefit of using multi-channel transmission/reception.
- This example only decodes basic system information. For an example of how to robustly decode more physical channels, see the “Cell Search, MIB and SIB1 Recovery” on page 2-351 example.

SDR Troubleshooting

- ADALM-PLUTO Radio “Common Problems and Fixes” (Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio)
- USRP Embedded Series Radio “Common Problems and Fixes” (Communications Toolbox Support Package for USRP Embedded Series Radio)
- Xilinx Zynq-Based Radio “Common Problems and Fixes” (Communications Toolbox Support Package for Xilinx Zynq-Based Radio)
- USRP Radio “Common Problems and Fixes” (Communications Toolbox Support Package for USRP Radio)

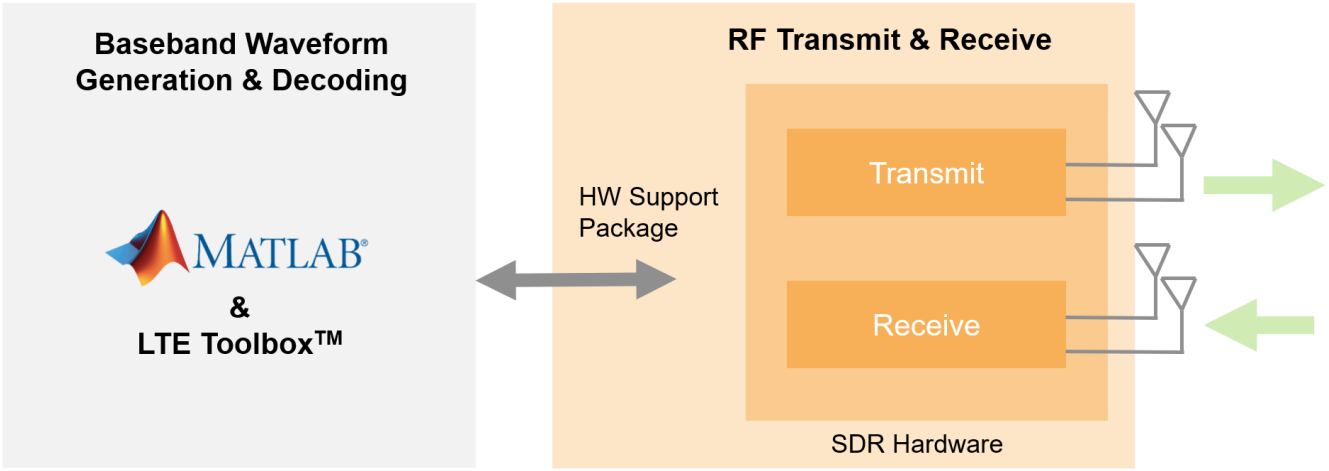
Local Functions

```
function waveform = captureWaveform(sdrReceiver,numSamplesToCapture)
% CAPTUREWAVEFORM returns a column vector of complex values given an
% SDRRECEIVER object and a scalar NUMSAMPLESTOCAPTURE value.
% For a comm.SDRuReceiver object, use the burst capture technique to
% acquire the waveform
    if isa(sdrReceiver,'comm.SDRuReceiver')
        waveform = complex(zeros(numSamplesToCapture,length(sdrReceiver.ChannelMapping)));
        samplesPerFrame = sdrReceiver.SamplesPerFrame;
        for i = 1:sdrReceiver.NumFramesInBurst
            waveform(samplesPerFrame*(i-1)+(1:samplesPerFrame),:) = sdrReceiver();
        end
    else
        waveform = capture(sdrReceiver,numSamplesToCapture);
    end
end
```

Image Transmission and Reception Using LTE Waveform and SDR

This example shows how to transmit and receive an image using an LTE waveform and a software-defined radio (SDR). The example generates a single or multi-antenna LTE waveform for simultaneous transmission and reception on a single SDR platform. During waveform generation, the example encodes and packs an image into a radio frame. Then, on reception, the example decodes the image from the received LTE waveform.

Introduction



The LTE Toolbox™ generates standard-compliant baseband IQ downlink and uplink reference measurement channel (RMC) waveforms and downlink test model (E-TM) waveforms. Using an SDR such as the Xilinx® Zynq®-Based Radio, you can modulate these waveforms for RF transmission and reception.

This example imports an image file and packs it into multiple radio frames of a baseband RMC waveform generated by using the LTE Toolbox. The example creates a continuous RF LTE waveform by using the repeated transmission functionality of a supported SDR. The repeated transmission functionality transfers the baseband RMC waveform to the hardware memory on the SDR and transmits the waveform continuously over the air without gaps. If you use an SDR device capable of multiple channel transmission and reception, the example generates and transmits a multi-antenna LTE waveform using LTE Transmit Diversity.

The example captures the resultant waveform by using the same SDR. If you have the appropriate hardware, the example uses multi-channel reception in the receiver.

This example supports these SDRs.

- ADALM-Pluto from the Communications Toolbox Support Package for Analog Devices® ADALM-Pluto Radio
- USRP™ E310/E312 from the Communications Toolbox Support Package for USRP™ Embedded Series Radio

- AD936x/FMCOMMS5 from the Communications Toolbox Support Package for Xilinx® Zynq®-Based Radio

Alternatively, if an SDR is not available, the example supports simulating an additive white gaussian noise (AWGN) channel or passing the waveform directly to the decode stage with no impairments.

Example Setup

Before running the example, set the `channel` variable to one of these options:

- `OverTheAir`: Use an SDR to transmit and receive the WLAN waveform
- `GaussianNoise`: Pass the transmission waveform through an AWGN channel (default)
- `NoImpairments`: Pass the transmission waveform through with no impairments

```
channel = GaussianNoise;
```

If you set `channel` to `OverTheAir`, configure the transmit gain and center frequency, and set `deviceName` to the desired SDR:

- Set to `Pluto` to use the ADALM-Pluto Radio (default)
- Set to `E3xx` to use the USRP Embedded Series Radio
- Set to `AD936x` or `FMCOMMS5` to use the Xilinx Zynq-Based Radio

If you set channel to `GaussianNoise`, set the signal-to-noise ratio (SNR) for the receive waveform.

```
if channel == "OverTheAir"
    deviceName = Pluto;
    centerFrequency = 2200000000;
    txGain = -10 ;
elseif channel == "GaussianNoise"
    % Specify SNR of received signal for a simulated channel
    SNR = 20;
end
```

Configure these parameters for LTE waveform generation.

```
txsim.RC = R.7;          % Base RMC configuration, 10 MHz bandwidth
txsim.NCellID = 88;      % Cell identity
txsim.NFrame = 700;      % Initial frame number
txsim.NumAntennas = 1; % Number of transmit and receive antennas
```

Configure all the scopes and figures for the example.

Set up a handle for the image plot.

```
if ~exist("imFig", "var") || ~ishandle(imFig)
    imFig = figure;
```

```
    imFig.NumberTitle = "off";
    imFig.Name = "Image Plot";
    imFig.Visible = "off";
else
    clf(imFig); % Clear figure
    imFig.Visible = "off";
end
```

Set up a handle for the channel estimate plots.

```
if ~exist("hhest", "var") || ~ishandle(hhest)
    hhest = figure("Visible","Off");
    hhest.NumberTitle = "off";
    hhest.Name = "Channel Estimate";
else
    clf(hhest); % Clear figure
    hhest.Visible = "off";
end
```

Set up a spectrum scope to view the received waveform later in the example.

```
spectrumScope = spectrumAnalyzer( ...
    SpectrumType="power-density", ...
    Title="Received Baseband LTE Signal Spectrum", ...
    YLabel="Power spectral density", ...
    ShowLegend=true);
```

Set up the constellation diagram viewer for the equalized physical downlink shared channel (PDSCH) symbols.

```
constellation = comm.ConstellationDiagram(Title="Equalized PDSCH Symbols", ...
    ShowReferenceConstellation=false);
```

Determine and set the positions for all figures in this example.

```
[positionsValid,positions] = hPlotPositions;
if positionsValid
    imFig.Position = positions(5,:);
    hhest.Position = positions(1,:);
    spectrumScope.Position = positions(6,:);
    constellation.Position = positions(3,:);
end
```

Transmitter Design: System Architecture

The general structure of the LTE transmitter consists of these steps:

1. Import and convert an image to a binary data stream.
2. Generate a baseband LTE signal using LTE Toolbox, packing the binary data stream into the transport blocks of the downlink shared channel (DL-SCH).
3. If using an SDR, prepare the baseband signal for transmission using the SDR hardware and send the baseband data to the SDR for continuous transmission at the desired center frequency.

Prepare Image File

The example reads data from the image file, scales it for transmission, and converts it to a binary data stream. The size of the transmitted image directly impacts the number of LTE radio frames

required for the transmission of the image data. A scaling factor of `0.5` requires five LTE radio frame transmissions. Increasing the scaling factor results in more frame transmissions. Reducing the scaling factor reduces the number of frames.

Specify the image file name and its scaling factor.

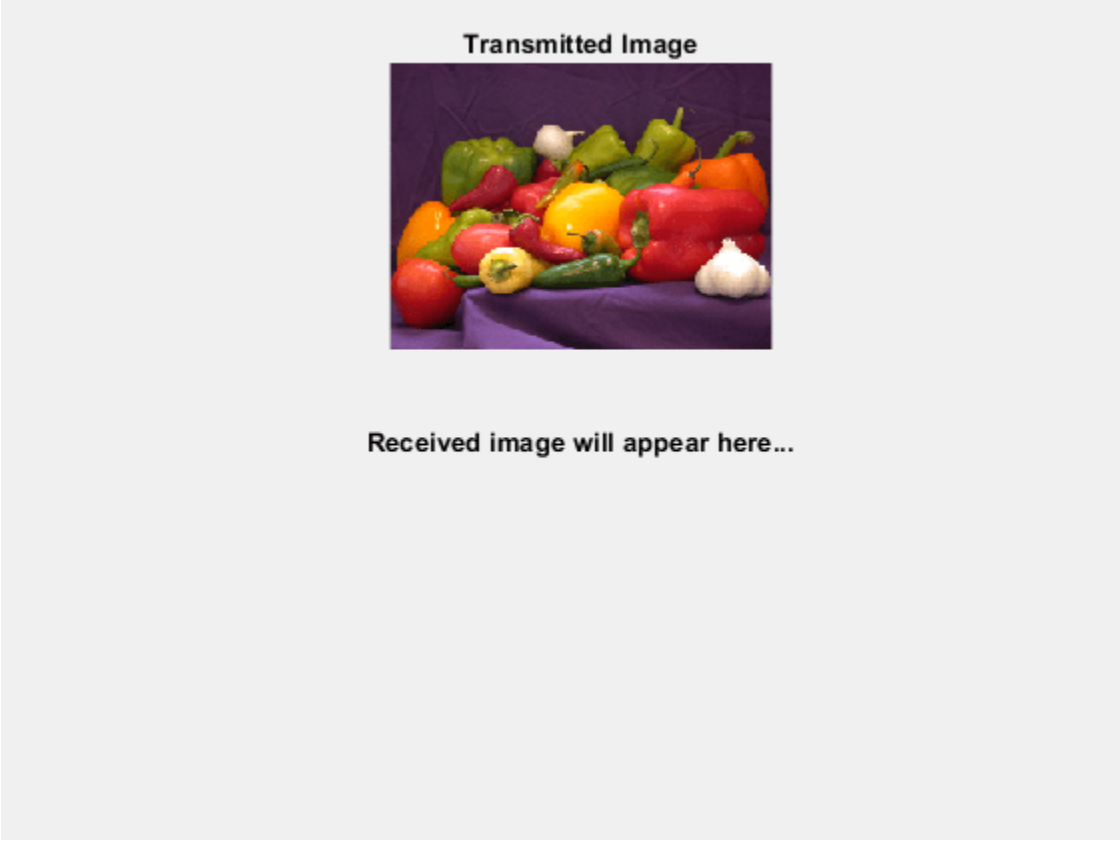
```
fileTx = 'peppers.png';                                                 % Image file name
scale = 0.5;                                                            % Image scaling factor
```

Import the specified image file and covert it to a binary stream.

```
fData = imread(fileTx);                                                 % Read image data from fi
origSize = size(fData);                                                 % Original input image si
scaledSize = max(floor(scale.*origSize(1:2)),1);                        % Calculate new image siz
heightIx = min(round(((1:scaledSize(1))-0.5)./scale+0.5),origSize(1));
widthIx = min(round(((1:scaledSize(2))-0.5)./scale+0.5),origSize(2));
fData = fData(heightIx,widthIx,:);                                      % Resize image
imsize = size(fData);                                                   % Store new image size
binData = dec2bin(fData(:),8);                                          % Convert to 8 bit unsign
trData = reshape((binData-'0').',1,[]).';                               % Create binary stream
```

Using the previously created handle for the image plot, display the transmit image. Later in the example, if the LTE waveform is successfully decoded, the received image displays beneath the transmitted image.

```
% Plot transmit image
figure(imFig);
imFig.Visible = "on";
subplot(211);
imshow(fData);
title("Transmitted Image");
subplot(212);
title("Received image will appear here...");
set(gca,"Visible","off"); % Hide axes
```




```
set(findall(gca, "type", "text"), "visible", "on"); % Unhide title
```

Generate Baseband LTE Signal

Use the `lteRMCDL` function to generate the default configuration parameters for an RMC as defined in TS36.101 Annex A.3 [1 on page 2-694]. The parameters within the configuration structure `rmc` allow customization as required.

```
rmc = lteRMCDL(txsim.RC);
```

Calculate the required number of LTE frames based on the size of the image data.

```
trBlkSize = rmc.PDSCH.TrBlkSizes;
txsim.TotFrames = ceil(numel(trData)/sum(trBlkSize(:)));
```

Customize RMC parameters

```
rmc.NCellID = txsim.NCellID;
rmc.NFrame = txsim.NFrame;
rmc.TotSubframes = txsim.TotFrames*10; % 10 subframes per frame
rmc.CellRefP = txsim.NumAntennas;      % Configure number of cell reference ports
rmc.PDSCH.RVSeq = 0;
```

Fill subframe 5 with dummy data.

```
rmc.OCNGPDSCHEnable = "On";
rmc.OCNGPDCCHEnable = "On";
```

If transmitting over two or more channels, enable transmit diversity.

```
if rmc.CellRefP >= 2
    rmc.PDSCH.TxScheme = "TxDiversity";
    rmc.OCNGPDSCH.TxScheme = "TxDiversity";
else
    rmc.PDSCH.TxScheme = "Port0";
    rmc.OCNGPDSCH.TxScheme = "Port0";
end
rmc.PDSCH.NLayers = txsim.NumAntennas;
fprintf("\nGenerating LTE transmit waveform:\n"+ ...
    "  Packing image data into %d frame(s).\n\n", txsim.TotFrames);
```

```
Generating LTE transmit waveform:
  Packing image data into 5 frame(s).
```

Use the `lteRMCDLTool` to generate a baseband waveform, `eNodeBOutput`, a fully populated resource grid, `txGrid`, and the full configuration of the RMC. The tool takes the binary stream, `trData`, created from the input image file and packs it into multiple transport blocks in the PDSCH.

```
% Pack the image data into a single LTE frame
[eNodeBOutput,txGrid,rmc] = lteRMCDLTool(rmc,trData);
```

Transmit LTE Waveform

If using an SDR, create an `sdrTransmitter` object using the `sdrtx` function. Set the center frequency, sample rate, gain, and channel configuration to the corresponding properties of the `sdrTransmitter` object. Then, use the `transmitRepeat` function to transfer the baseband LTE transmission to the SDR platform for continuous transmission.

```
if channel == "OverTheAir"

    % Transmitter properties
    sdrTransmitter = sdrtx(deviceName);
    sdrTransmitter.BasebandSampleRate = rmc.SamplingRate; % 15.36 MHz for default RMC (R.7) with
    sdrTransmitter.CenterFrequency = centerFrequency;
    sdrTransmitter.Gain = txGain;
    sdrTransmitter.ChannelMapping = 1:txsim.NumAntennas;

    % Pass the SDR I/O directly to host skipping FPGA on Zynq Radio or USRP
    % Embedded Series Radio
    if deviceName ~= "Pluto"
        sdrTransmitter.ShowAdvancedProperties = true;
        sdrTransmitter.BypassUserLogic = true;
    end

    fprintf("\nGenerating LTE transmit waveform:\n")

    % Scale the normalized signal to avoid saturation of RF stages
    powerScaleFactor = 0.8;
    eNodeBOutput = eNodeBOutput.*(1./max(abs(eNodeBOutput))*powerScaleFactor);

    % Transmit RF waveform
    transmitRepeat(sdrTransmitter,eNodeBOutput);
end
```

Receiver Design: System Architecture

The general structure of the LTE receiver consists of these steps:

1. If using an SDR, capture a suitable number of frames of the transmitted LTE signal. Otherwise, apply AWGN to the `eNodeBOutput` or apply no impairments.
2. Determine and correct the frequency offset of the received signal.
3. Synchronize the captured signal to the start of an LTE frame.
4. OFDM demodulate the received signal to get an LTE resource grid.
5. Perform a channel estimation for the received signal.
6. Decode the PDSCH and DL-SCH to obtain the transmitted data from the transport blocks of each radio frame.
7. Recombine received transport block data to form the received image.

This example plots the power spectral density of the captured waveform, and shows visualizations of the estimated channel, equalized PDSCH symbols, and received image.

Set Up SDR Receiver

Capture one more LTE frame than transmitted to allow for timing offset wraparound in receiver processing.

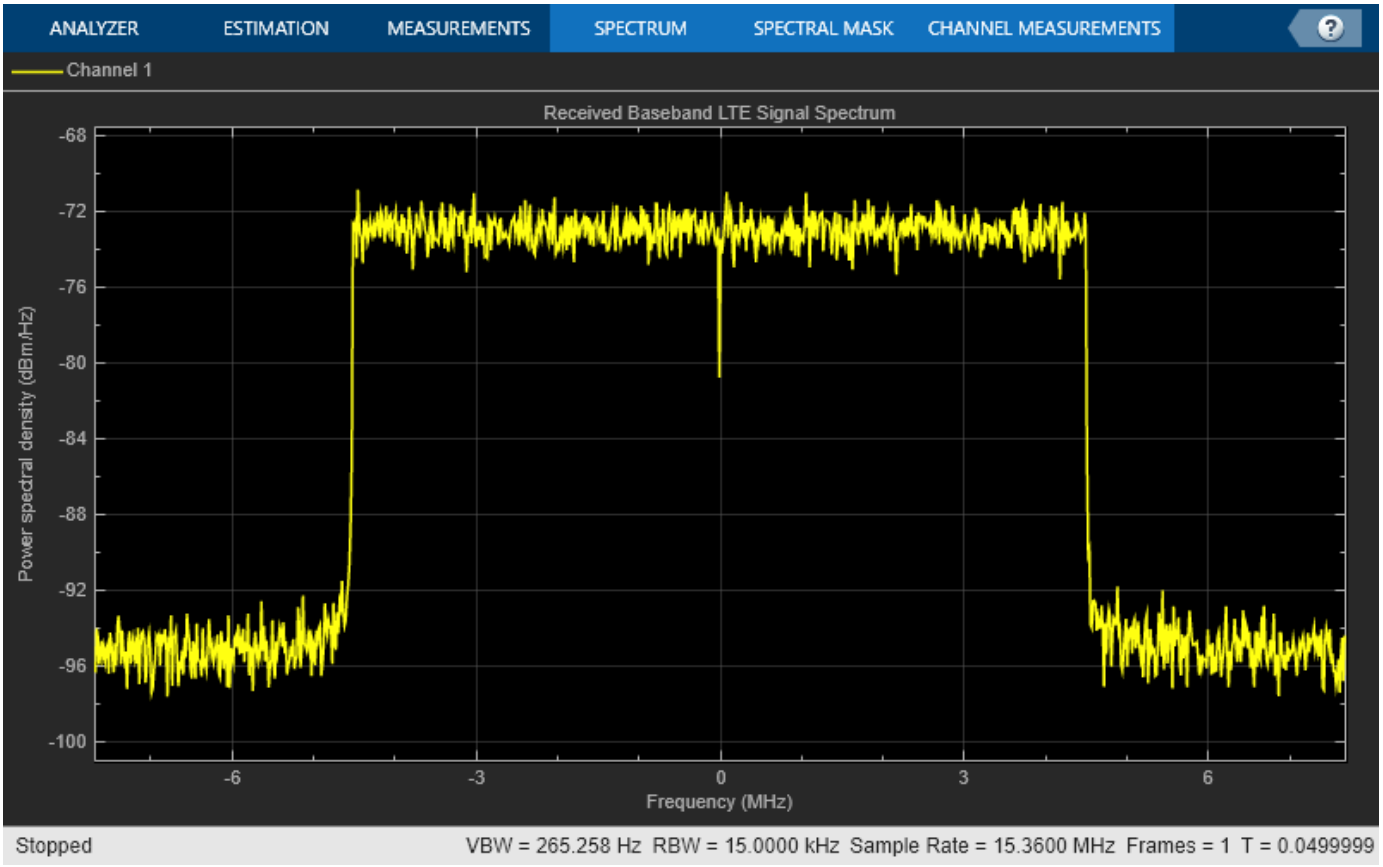
```
framesPerCapture = txsim.TotFrames+1;   % Number of LTE frames to capture.
captureTime = framesPerCapture * 10e-3; % Capture time in seconds
```

Create an SDR receiver System Object with the specified properties for the device used for the image transmission. The sample rate of the receiver is 15.36MHz, which is the standard sample rate for capturing an LTE bandwidth of 50 resource blocks (RBs). 50 RBs is equivalent to a signal bandwidth of 10 MHz.

```
if channel == "OverTheAir"

    sdrReceiver = sdrrx(deviceName);
    sdrReceiver.BasebandSampleRate = sdrTransmitter.BasebandSampleRate;
    sdrReceiver.CenterFrequency = sdrTransmitter.CenterFrequency;
    sdrReceiver.OutputDataType = "double";
    sdrReceiver.ChannelMapping = 1:txsim.NumAntennas;

    if deviceName ~= "Pluto"
        sdrReceiver.ShowAdvancedProperties = true;
        sdrReceiver.BypassUserLogic = true;
    end

    fprintf("\nStarting a new RF capture.\n")

    rxWaveform = capture(sdrReceiver,captureTime,"Seconds");
elseif channel == "GaussianNoise"
    rxWaveform = awgn(eNodeBOutput,SNR,"measured");
else % No Impairments
    rxWaveform = eNodeBOutput;
end
```

Show the power spectral density of the captured waveform.

```
spectrumScope.SampleRate = rmc.SamplingRate;
spectrumScope(rxWaveform);
release(spectrumScope)
```



Set Up LTE Receiver

The example simplifies the LTE signal reception by assuming that the transmitted PDSCH properties are known.

Assume FDD duplexing mode and a normal cyclic prefix length, as well as four cell-specific reference ports (CellRefP) for the master information block (MIB) decode. The MIB provides the number of actual CellRefP.

```
enb.PDSCH = rmc.PDSCH;
enb.DuplexMode = "FDD";
enb.CyclicPrefix = "Normal";
enb.CellRefP = 4;
```

The sampling rate of the signal controls the captured bandwidth. Obtain the number of RBs captured from a lookup table using the chosen sampling rate.

```
% Bandwidth:    {1.4  3    5    10    20   } MHz
SampleRateLUT = [1.92 3.84 7.68 15.36 30.72]*1e6;
NDLRBLUT = [6 15 25 50 100];
enb.NDLRB = NDLRBLUT(SampleRateLUT==rmc.SamplingRate);
if isempty(enb.NDLRB)
    error("Sampling rate not supported. Supported rates are %s.",...
        "1.92 MHz, 3.84 MHz, 7.68 MHz, 15.36 MHz, 30.72 MHz");
end
```

Configure channel estimation to be performed by using cell-specific reference signals. A 9-by-9 averaging window minimizes the effect of noise.

```
% Channel estimation configuration structure
cec.PilotAverage = "UserDefined";  % Type of pilot symbol averaging
cec.FreqWindow = 9;                % Frequency window size in REs
cec.TimeWindow = 9;                % Time window size in REs
cec.InterpType = "Cubic";          % 2D interpolation type
cec.InterpWindow = "Centered";     % Interpolation window type
cec.InterpWinSize = 3;             % Interpolation window size
```

Process Captured Signal

As the LTE waveform is continuously transmitted over the air in a loop, the first frame captured by the receiver is not guaranteed to be the first transmitted frame. This means that the frames may be decoded out of sequence. To enable the received frames to be recombined in the correct order, their frame numbers must be determined. The MIB contains information on the current system frame number, and therefore must be decoded. After the frame number has been determined, the example decodes the PDSCH and DL-SCH and displays the equalized PDSCH symbols. No data is transmitted in subframe 5; therefore, the captured data for subframe is ignored for the decoding.

When the LTE frames have been successfully decoded, the example displays the detected frame number on a frame-by-frame basis and the equalized PDSCH symbol constellation for each subframe. The example also displays an estimate of the channel magnitude frequency response between cell reference point 0 and the receive antenna for each frame.

Perform a frequency offset correction for known cell ID.

```
frequencyOffset = lteFrequencyOffset(enb,rxWaveform);
rxWaveform = lteFrequencyCorrect(enb,rxWaveform,frequencyOffset);
fprintf("\nCorrected a frequency offset of %i Hz.\n",frequencyOffset)
```

```
Corrected a frequency offset of -2.447074e+00 Hz.
```

Perform the blind cell search to obtain cell identity and timing offset. Use `"PostFFT"` SSS detection method to improve detection speed.

```
cellSearch.SSSDetection = "PostFFT"; cellSearch.MaxCellCount = 1;
[NCellID,frameOffset] = lteCellSearch(enb,rxWaveform,cellSearch);
enb.NCellID = NCellID;
fprintf("Detected a cell identity of %i.\n", NCellID);
```

```
Detected a cell identity of 88.
```

Sync the captured samples to the start of an LTE frame, and trim off any samples that are part of an incomplete frame.

```
rxWaveform = rxWaveform(frameOffset+1:end,:);
samplesPerFrame = 10e-3*rmc.SamplingRate; % LTE frames period is 10 ms
tailSamples = mod(length(rxWaveform),samplesPerFrame);
rxWaveform = rxWaveform(1:end-tailSamples,:);
enb.NSubframe = 0;
fprintf("Corrected a timing offset of %i samples.\n",frameOffset)
```

```
Corrected a timing offset of 0 samples.
```

OFDM demodulate the waveform and perform channel estimation for 4 cell-specific reference ports as the cell-specific reference ports are unknown for the eNodeB.

```
rxGrid = lteOFDMDemodulate(enb,rxWaveform);
[hest,nest] = lteDLChannelEstimate(enb,cec,rxGrid);
sfDims = lteResourceGridSize(enb);
Lsf = sfDims(2); % OFDM symbols per subframe
LFrame = 10*Lsf; % OFDM symbols per frame
numFullFrames = length(rxWaveform)/samplesPerFrame;
```

Initialize these variables for processing the received frames.

```
rxDataFrame = zeros(sum(enb.PDSCH.TrBlkSizes(:)),numFullFrames);
recFrames = zeros(numFullFrames,1);
rxSymbols = []; txSymbols = [];
```

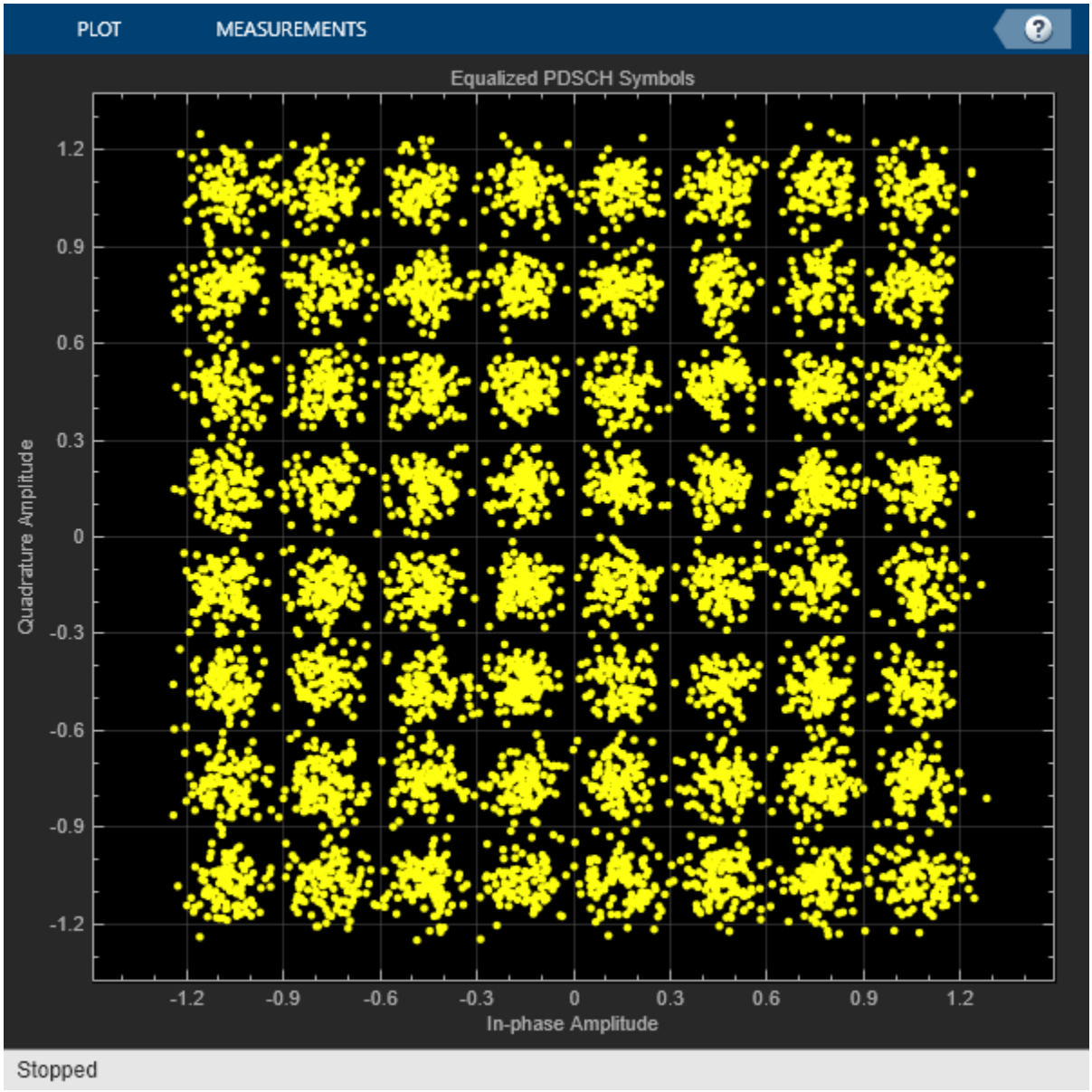
For each frame, decode the MIB, PDSCH, and DL-SCH.

```
for frame = 0:(numFullFrames-1)
    fprintf("\nPerforming DL-SCH Decode for frame %i of %i in burst:\n", ...
        frame+1,numFullFrames)

    % Extract subframe #0 from each frame of the received resource grid
    % and channel estimate.
    enb.NSubframe = 0;
    rxsf = rxGrid(:,frame*LFrame+(1:Lsf),:);
    hestsf = hest(:,frame*LFrame+(1:Lsf),:,:);

    % PBCH demodulation. Extract resource elements (REs)
    % corresponding to the PBCH from the received grid and channel
    % estimate grid for demodulation.
    enb.CellRefP = 4;
    pbchIndices = ltePBCHIndices(enb);
    [pbchRx,pbchHest] = lteExtractResources(pbchIndices,rxsf,hestsf);
    [~,~,nfmod4,mib,CellRefP] = ltePBCHDecode(enb,pbchRx,pbchHest,nest);

    % If PBCH decoding successful CellRefP~=0 then update info
    if ~CellRefP
        fprintf("  No PBCH detected for frame.\n");
        continue;
    end
    enb.CellRefP = CellRefP;

    % Decode the MIB to get current frame number
    enb = lteMIB(mib,enb);

    % Incorporate the nfmod4 value output from the function
    % ltePBCHDecode, as the NFrame value established from the MIB
    % is the system frame number modulo 4.
    enb.NFrame = enb.NFrame+nfmod4;
    fprintf("  Successful MIB Decode.\n")
    fprintf("  Frame number: %d.\n",enb.NFrame);

    % Store received frame number
    recFrames(frame+1) = enb.NFrame;

    % Process subframes within frame (ignoring subframe 5)
    for sf = 0:9
        if sf~=5 % Ignore subframe 5
            % Extract subframe
            enb.NSubframe = sf;
```

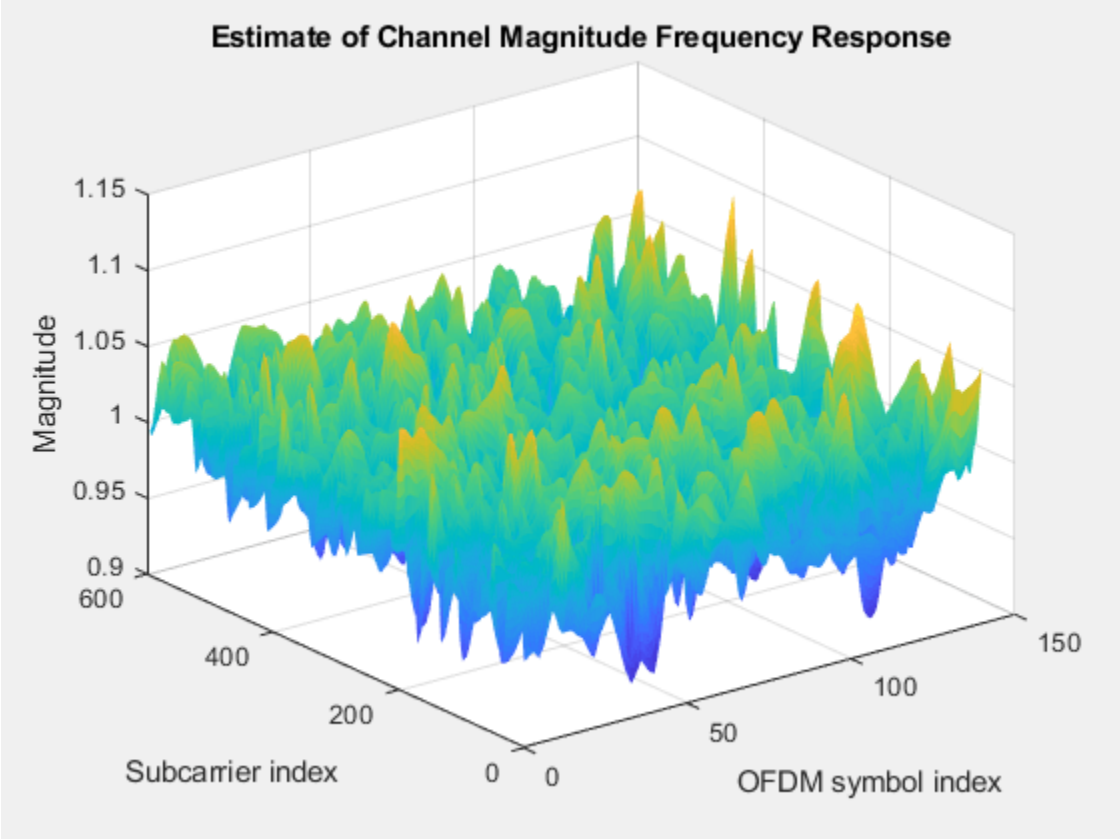
```
            rxsf = rxGrid(:,frame*LFrame+sf*Lsf+(1:Lsf),:);

            % Perform channel estimation with the correct number of CellRefP
            [hestsf,nestsf] = lteDLChannelEstimate(enb,cec,rxsf);

            % Physical control format indicator channel (PCFICH)
            % demodulation. Extract REs corresponding to the PCFICH from
            % the received grid and channel estimate for demodulation.
            pcfichIndices = ltePCFICHIndices(enb);
            [pcfichRx,pcfichHest] = lteExtractResources(pcfichIndices,rxsf,hestsf);
            [cfiBits,recsym] = ltePCFICHDecode(enb,pcfichRx,pcfichHest,nestsf);

            % Control format indicator (CFI) decoding
            enb.CFI = lteCFIDecode(cfiBits);

            % Get PDSCH indices
            [pdschIndices,pdschIndicesInfo] = ltePDSCHIndices(enb, enb.PDSCH, enb.PDSCH.PRBSet);
            [pdschRx, pdschHest] = lteExtractResources(pdschIndices, rxsf, hestsf);

            % Perform deprecoding, layer demapping, demodulation and
            % descrambling on the received data using the estimate of
            % the channel
            [rxEncodedBits, rxEncodedSymb] = ltePDSCHDecode(enb,enb.PDSCH,pdschRx,...
                pdschHest,nestsf);

            % Append decoded symbol to stream
            rxSymbols = [rxSymbols; rxEncodedSymb{:}]; %#ok<AGROW>

            % Transport block sizes
            outLen = enb.PDSCH.TrBlkSizes(enb.NSubframe+1);

            % Decode DownLink Shared Channel (DL-SCH)
            [decbits{sf+1}, blkcrc(sf+1)] = lteDLSCHDecode(enb,enb.PDSCH,...
                outLen, rxEncodedBits);  %#ok<SAGROW>

            % Recode transmitted PDSCH symbols for EVM calculation
            % Encode transmitted DLSCH
            txRecode = lteDLSCH(enb,enb.PDSCH,pdschIndicesInfo.G,decbits{sf+1});
            % Modulate transmitted PDSCH
            txRemod = ltePDSCH(enb, enb.PDSCH, txRecode);
            % Decode transmitted PDSCH
            [~,refSymbols] = ltePDSCHDecode(enb, enb.PDSCH, txRemod);
            % Add encoded symbol to stream
            txSymbols = [txSymbols; refSymbols{:}]; %#ok<AGROW>

            constellation(rxEncodedSymb{:}); % Plot current constellation
            release(constellation); % Release previous constellation plot
        end
    end

    % Reassemble decoded bits
    fprintf("  Retrieving decoded transport block data.\n");
    rxdata = [];
    for i = 1:length(decbits)
        if i~=6 % Ignore subframe 5
            rxdata = [rxdata; decbits{i}{:}]; %#ok<AGROW>
        end
    end
```

```
    % Store data from receive frame
    rxDataFrame(:,frame+1) = rxdata;

    % Plot channel estimate between CellRefP 0 and the receive antennae
    focalFrameIdx = frame*LFrame+(1:LFrame);
    figure(hhest);
    hhest.Visible = "On";
    surf(abs(hest(:,focalFrameIdx,1,1)));
    shading flat;
    xlabel("OFDM symbol index");
    ylabel("Subcarrier index");
    zlabel("Magnitude");
    title("Estimate of Channel Magnitude Frequency Response");
end
```

```
Performing DL-SCH Decode for frame 1 of 5 in burst:

  Successful MIB Decode.

  Frame number: 700.

  Retrieving decoded transport block data.

Performing DL-SCH Decode for frame 2 of 5 in burst:

  Successful MIB Decode.

  Frame number: 701.

  Retrieving decoded transport block data.

Performing DL-SCH Decode for frame 3 of 5 in burst:

  Successful MIB Decode.

  Frame number: 702.

  Retrieving decoded transport block data.

Performing DL-SCH Decode for frame 4 of 5 in burst:

  Successful MIB Decode.

  Frame number: 703.

  Retrieving decoded transport block data.

Performing DL-SCH Decode for frame 5 of 5 in burst:

  Successful MIB Decode.

  Frame number: 704.
```




```
Retrieving decoded transport block data.
```




```
if channel == "OverTheAir"
    release(sdrTransmitter);
    release(sdrReceiver);
end
```

Result Qualification and Display

To determine the quality of the received data, the example calculates the bit error rate (BER) between the transmitted and received data. The received data reforms into an image.

Determine the index of first transmitted frame (lowest received frame number).

```
[~,frameIdx] = min(recFrames);
```

Initialize these variables for image reconstruction.

```
decodedRxDataStream = zeros(length(rxDataFrame(:)),1);
frameLen = size(rxDataFrame,1);
```

Recombine the received data blocks (in correct order) into a continuous stream.

```
for n=1:numFullFrames
    currFrame = mod(frameIdx-1,numFullFrames)+1;                                % Get current fr
    decodedRxDataStream((n-1)*frameLen+1:n*frameLen) = rxDataFrame(:,currFrame);
    frameIdx = frameIdx+1;                                                      % Increment fram
end
```

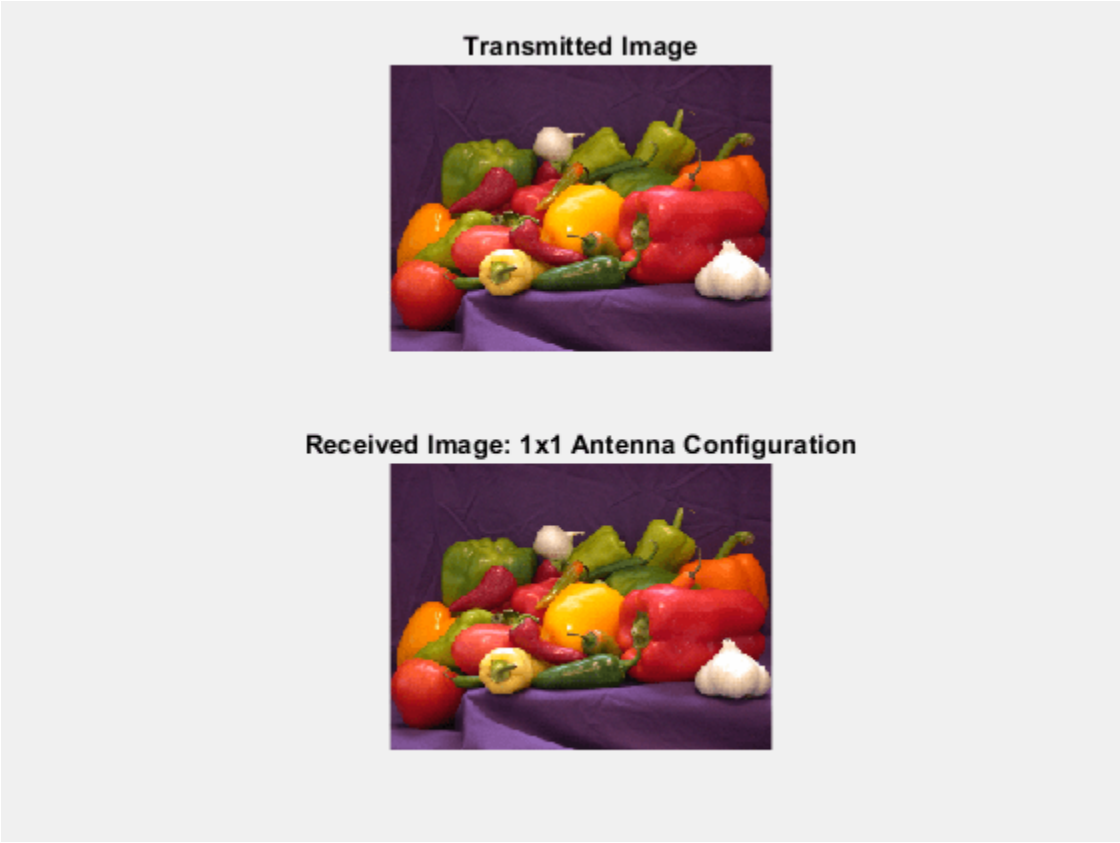
Use the EVM and error rate System Objects, to calculate the EVM and bit error rate of the decoded data.

```
if ~isempty(rxSymbols)
    evmCalculator = comm.EVM();
    evmCalculator.MaximumEVMOutputPort = true;
    [evm.RMS,evm.Peak] = evmCalculator(txSymbols, rxSymbols);
    fprintf('  EVM peak = %0.3f%%\n',evm.Peak);
    fprintf('  EVM RMS  = %0.3f%%\n',evm.RMS);

    % Perform bit error rate (BER) calculation
    bitErrorRate = comm.ErrorRate;
    err = bitErrorRate(decodedRxDataStream(1:length(trData)), trData);
    fprintf('  Bit Error Rate (BER) = %0.5f.\n',err(1));
    fprintf('  Number of bit errors = %d.\n',err(2));
    fprintf('  Number of transmitted bits = %d.\n',length(trData));
else
    fprintf('  No transport blocks decoded.\n');
end
```

```
  EVM peak = 33.306%

  EVM RMS  = 8.429%

  Bit Error Rate (BER) = 0.00000.

  Number of bit errors = 0.

  Number of transmitted bits = 1179648.
```

Recreate the image from the received data.

```
str = reshape(sprintf('%d',decodedRxDataStream(1:length(trData))), 8, []).';
decdata = uint8(bin2dec(str));
receivedImage = reshape(decdata,imsize);

if exist("imFig", "var") && ishandle(imFig) % If TX figure is open
    figure(imFig); subplot(212);
else
    figure; subplot(212);
end

imshow(receivedImage);
title(sprintf("Received Image: %dx%d Antenna Configuration",txsim.NumAntennas, txsim.NumAntennas
```



Further Exploration

- If using an SDR, try reducing the transmitter gain to impair the quality of the received image and to increase the number of bit errors. Otherwise, reduce the signal-to-noise ratio (`SNR` on page 2-680).
- If using a supported multi-channel SDR, try increasing the number of antennas (`txsim.NumAntennas` on page 2-680) to visualize the benefit of using multi-channel transmission.

SDR Troubleshooting

- ADALM-PLUTO Radio “Common Problems and Fixes” (Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio)
- USRP Embedded Series Radio “Common Problems and Fixes” (Communications Toolbox Support Package for USRP Embedded Series Radio)
- Xilinx Zynq-Based Radio “Common Problems and Fixes” (Communications Toolbox Support Package for Xilinx Zynq-Based Radio)

Selected Bibliography

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).

Real Time LTE MIB Recovery Using Analog Devices AD9361/ AD9364

This example shows how to implement an LTE MIB recovery system on the Xilinx® Zynq® radio platform partitioned across the ARM® and the programmable logic fabric.

Example Summary

This example is part of the Communications Toolbox™ Support Package for Xilinx Zynq-Based Radio. This section provides a summary and links to the example.

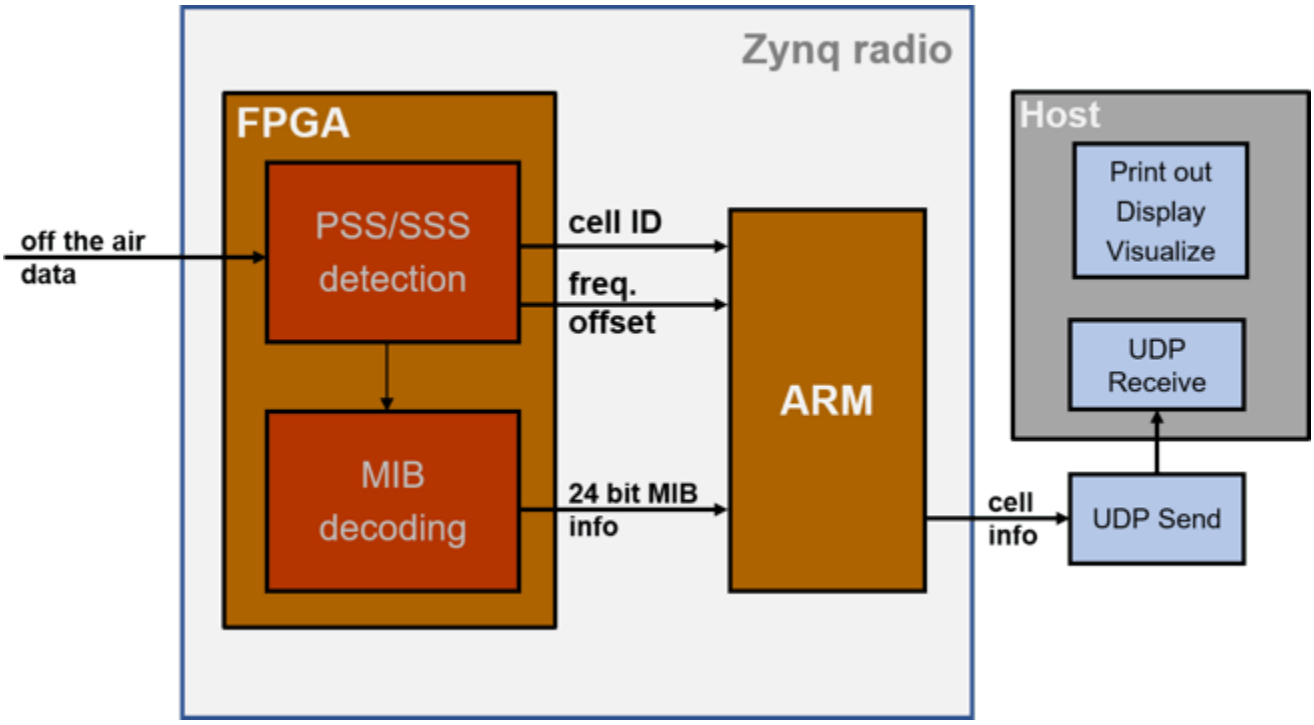
The example implements an LTE MIB decoding algorithm on the Xilinx Zynq radio platform. Using a Simulink model, the example partitions the algorithm across the ARM processor and the field programmable gate array (FPGA), and the resulting implementation executes in real time.

Other examples use the LTE Toolbox™ to verify the developed algorithm. These steps show the verification process of the algorithm.

- The “Cell Search, MIB and SIB1 Recovery” on page 2-351 example introduces a behavioral model of the algorithm in floating point. This example studies the algorithm's performance and behavior.
- The “LTE HDL Cell Search” (Wireless HDL Toolbox) example provides a fixed point model optimized for HDL code generation.
- The current example shows the hardware-software partitioning of the optimized fixes point model.

Hardware-Software Partitioning

In this examples, Simulnk performs the hardware-software partitioning. The figure below shows the MIB recovery algorithm implemented entirely on the FPGA as it involves high rate signal processing. The ARM parses the MIB data and sends some useful information back to the host over a UDP link for display. The ARM also controls the start/reset of the FPGA intellectual property (IP) and which input data to use: off-the-air or test data stored on the FPGA.



Support Packages

Use the Add-On Explorer to install the required support packages.

More information about other supported SDR platforms can be found here.

Full Example

The full example description and source code can be found in the Communications Toolbox Support Package for Xilinx Zynq-Based Radio documentation under the name “LTE MIB Recovery and Cell Scanner Using Analog Devices AD9361/AD9364” (Communications Toolbox Support Package for Xilinx Zynq-Based Radio).

LTE Physical Layer Examples

Create Synchronization Signals

This example shows how to construct synchronization signals using LTE Toolbox™. In this example, you create the primary and secondary synchronization signals and map them to a resource grid.

Set up the cell-wide settings. Create a structure and specify the cell-wide settings as its fields.

```
enb.NDLRB = 9;
enb.CyclicPrefix = 'Normal';
enb.CellRefP = 1;
enb.NCellID = 1;
enb.NSubframe = 0;
enb.DuplexMode = 'FDD';
```

Many of the functions used in this example require a subset of the preceding settings specified.

Generate the PSS symbols bu calling the `ltePSS` function with the cell-wide settings specified by `enb`.

```
pss = ltePSS(enb);
```

When a PSS signal is not located in `enb.NSubframe`, the function does not generate PSS symbols and returns an empty vector.

Next, generate the PSS indices. These indices map the PSS complex symbols to the subframe resource grid. Use the `ltePSSIndices` function for the specified cell-wide settings and antenna number. In this case, since only one antenna port is used, specify `antenna` as `0`.

```
antenna = 0;
pssIndices = ltePSSIndices(enb, antenna);
```

In this example, you generate subframe 0. Since subframe 0 contains a PSS signal, the function generates PSS indices. If `enb.NSubframe` is a subframe that does not contain a PSS signal, the function would return an empty vector.

Generate the SSS symbols by calling the `lteSSS` function with the cell-wide settings specified by `enb`.

```
sss = lteSSS(enb);
```

When an SSS signal is not located in `enb.NSubframe`, the function does not generate SSS symbols. It returns an empty vector.

Next, generate the SSS indices. These indices map the SSS complex symbols to the subframe resource grid. Call the `lteSSSIndices` function, providing the cell-wide settings `enb` and the antenna port number `antenna`.

```
antenna = 0;
sssIndices = lteSSSIndices(enb, antenna);
```

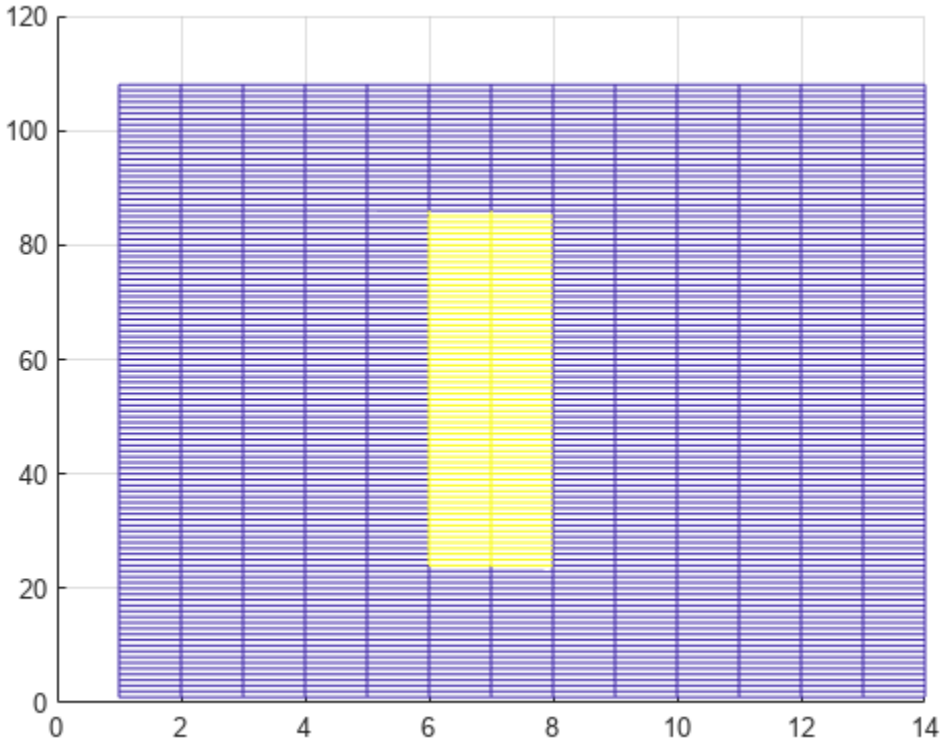
In this example, you generate subframe 0. Since subframe 0 contains an SSS signal, the function generates SSS indices. If `enb.NSubframe` is a subframe that does not contain an SSS signal, the function returns empty indices.

Generate the subframe resource grid by calling the `lteDLResourceGrid` function. You create an empty resource grid for one subframe.

```
subframe = lteDLResourceGrid(enb);
```

Finally, map the PSS and SSS symbols directly to the resource grid using the generated indices. Show the synchronization symbols mapped in RE grid.

```
subframe(pssIndices) = pss;
subframe(sssIndices) = sss;
mesh(abs(subframe))
view(2)
```



See Also

`ltePSS` | `ltePSSIndices` | `lteSSS` | `lteSSSIndices` | `lteDLResourceGrid` | `zadoffChuSeq`

More About

- “Synchronization Signals (PSS and SSS)” on page 1-9

Model CFI and PCFICH

This example shows how to generate a PCFICH with LTE Toolbox™. First, create a CFI based on the eNodeB configuration and code it. Then, generate a PCFICH using the coded CFI and map it to a resource grid.

Set up the cell-wide settings. Create a structure and specify the cell-wide settings as its fields.

```
enb.NDLRB = 9;
enb.CyclicPrefix = 'Normal';
enb.PHICHDuration = 'Normal';
enb.CFI = 3;
enb.CellRefP = 4;
enb.NCellID = 1;
enb.NSubframe = 0;
```

Many of the functions used in this example require a subset of the preceding settings specified.

Create an empty resource grid for one subframe by calling the `lteDLResourceGrid` function.

```
subframe = lteDLResourceGrid(enb);
```

The resulting subframe is a 3 dimensional matrix. The number of rows represents the number of subcarriers available, 12×(`enb.NDLRB`), since there are 12 subcarriers per resource block. The number of columns corresponds to the number of OFDM symbols in a subframe, 7×2, since there are 7 OFDM symbols per slot for normal cyclic prefix and there are 2 slots in a subframe. The third dimension of the matrix corresponds to the number of transmit antenna ports used. There are four specified in the example, so `enb.CellRefP` is 4.

Use the `lteCFI` function to code the CFI channel. The result, `cfiCodedBits`, is a 32-bit-long set of coded bits.

```
cfiCodedBits = lteCFI(enb);
```

As described earlier, the number of OFDM symbols used to transmit the control information in a subframe is defined by the CFI value. The `eNodeB` configuration structure assigns the CFI a value of 3. Thus, 4 OFDM symbols are used for the control region because the number of resource blocks used is less than 11, since `enb.NDLRB` is 9.

Generate the PCFICH complex symbols by using the `ltePCFICH` function. This function scrambles the CFI coded bits, QPSK modulates the symbols, maps symbols to layers, and precodes to form the PCFICH complex symbols.

```
pcfichSymbols = ltePCFICH(enb,cfiCodedBits);
```

The resulting matrix, `pcfichSymbols`, has 4 columns. Each column contains the PCFICH complex symbols that map to each of the antenna ports.

Generate the PCFICH mapping indices by calling the `ltePCFICHIndices` function. These indices map the PCFICH complex symbols to the subframe resource grid.

```
pcfichIndices = ltePCFICHIndices(enb,'1based');
```

The resulting matrix, `pcfichIndices`, has 4 columns. Each column contains the indices in linear form for each antenna port. These indices are one-based, since MATLAB® uses one-based indices. However, you can also generate 0-based indices.

Map the PCFICH complex symbols to the subframe resource grid using the appropriate mapping indices. The linear indexing style used makes the mapping process straightforward.

```
subframe(pcfichIndices) = pcfichSymbols;
```

The resulting matrix, `subframe`, contains the complex symbols in `pcfichSymbols` in the locations specified by `pcfichIndices`.

To view the resource usage, call the `ltePCFICHInfo` function. This function returns the number of resource elements, `NRE`, and the number of resource element groups, `NREG`, used by the PCFICH within the structure, `info`.

```
info = ltePCFICHInfo;
```

The resulting structure, `info`, contains the fields `NRE`, the number of resource elements, and `NREG`, the number of resource element groups, used by the PHICH.

See Also

`lteCFI` | `ltePCFICH` | `ltePCFICHInfo` | `ltePCFICHIndices` | `lteDLResourceGrid` | `ltePCFICHPRBS` | `lteSymbolModulate` | `lteSymbolDemodulate` | `lteLayerMap` | `lteLayerDemap` | `lteDLPrecode` | `lteDLDeprecode`

More About

- “Control Format Indicator (CFI) Channel” on page 1-23

Model HARQ Indicator and PHICH

This example shows how to implement the HARQ Indicator (HI) and physical HI channel (PHICH). You create the processing chain of coding hybrid indicator values, create the PHICH, and map it to a resource grid.

Set up the cell-wide settings. Create a structure and specify the cell-wide settings as its fields.

```
enb.NDLRB = 9;
enb.CyclicPrefix = 'Normal';
enb.PHICHDuration = 'Normal';
enb.Ng = 'Sixth';
enb.CellRefP = 4;
enb.NCellID = 1;
enb.NSubframe = 0;
enb.DuplexMode = 'FDD';
```

Many of the functions used in this example require a subset of the preceding settings specified.

To generate PHICH resource information, use the `ltePHICHInfo` function.

```
phichInfo = ltePHICHInfo(enb);
```

The function returns `phichInfo`, a structure containing the relevant data required to define PHICH sets. The elements and values of the structure are:

Structure Element	Description	Value
NREG	Number of resource element groups used to map the PHICHs.	3
NRE	Number of resource elements used to map the PHICHs.	12
NPHICH	Maximum number of PHICH that can be used.	8
NGroups	Maximum number of PHICH groups that can be used.	1
NMappingUnits	Number of PHICH mapping units used to map the maximum number of PHICH groups.	1
NSequences	Maximum number of orthogonal sequences that can be used within each group.	8
PHICHDuration	Number of OFDM symbols used to map the PHICH.	1

Generate the HARQ indicator (HI) set. An HI set consists of a HARQ indicator value, 1 for ACK and 0 for NACK, and a PHICH index pair that contains the PHICH group index, $n_{\mathrm{PHICH}}^{\mathrm{group}}$, and the orthogonal sequence index, $n_{\mathrm{PHICH}}^{\mathrm{seq}}$, for the PHICH containing HI. The values of $n_{\mathrm{PHICH}}^{\mathrm{group}}$ and $n_{\mathrm{PHICH}}^{\mathrm{seq}}$ can be determined using the PHICH resource dimension information returned by the `ltePHICHInfo` function. The number of groups determines acceptable values of the PHICH group index and the number of sequences determines acceptable values of sequence indexes.

```
HISet = [[0 0 1];[0 1 0];[0 4 0];[0 7 1]];
```

In this example, you create one PHICH group containing four PHICHs with the indices:

PHICH Group Index	PHICH Sequence Index	HARQ Indicator Value
0	0	1 – *ACK*
0	1	0 – *NACK*
0	4	0 – *NACK*
0	7	1 – *ACK*

In the LTE Toolbox™, a HI set matrix is used to define the HI and PHICH index pair for each HI within the subframe. The HI set matrix defines a single PHICH in terms of $n_{\mathrm{PHICH}}^{\mathrm{group}}$, $n_{\mathrm{PHICH}}^{\mathrm{seq}}$, and the HARQ indicator.

Generate the PHICH complex symbols from the cell-wide settings configuration and HARQ indicator matrix. To perform the required channel coding, modulation, scrambling, layer mapping, precoding, and combining of the PHICH groups, call the `ltePHICH` function.

```
phichSymbols = ltePHICH(enb,HISet);
disp(phichSymbols)
```

```
   0.0000 + 0.0000i   0.0000 + 0.0000i  -2.0000 + 0.0000i   0.0000 + 0.0000i
   2.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   1.0000 - 1.0000i   0.0000 + 0.0000i   1.0000 - 1.0000i   0.0000 + 0.0000i
  -1.0000 - 1.0000i   0.0000 + 0.0000i   1.0000 + 1.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   2.0000 + 0.0000i
   0.0000 + 0.0000i  -2.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -1.0000 + 1.0000i   0.0000 + 0.0000i   1.0000 - 1.0000i
   0.0000 + 0.0000i  -1.0000 - 1.0000i   0.0000 + 0.0000i  -1.0000 - 1.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i  -2.0000 + 0.0000i   0.0000 + 0.0000i
   2.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
  -1.0000 + 1.0000i   0.0000 + 0.0000i   1.0000 - 1.0000i   0.0000 + 0.0000i
  -1.0000 - 1.0000i   0.0000 + 0.0000i  -1.0000 - 1.0000i   0.0000 + 0.0000i
```

The resulting vector, `phichSymbols`, has 12 rows and four columns. Each column contains the complex symbols to be mapped to the resource grids for each of the four antenna ports.

To generate the PHICH mapping indices, use the `ltePHICHIndices` function. These mapping indices are required to map the complex PHICH symbols to the subframe resource grid.

```
phichIndices = ltePHICHIndices(enb);
disp(phichIndices)
```

```
    13   1525   3037   4549
    15   1527   3039   4551
    16   1528   3040   4552
    18   1530   3042   4554
    43   1555   3067   4579
    45   1557   3069   4581
    46   1558   3070   4582
    48   1560   3072   4584
    79   1591   3103   4615
    81   1593   3105   4617
    82   1594   3106   4618
    84   1596   3108   4620
```

This function returns a matrix with four columns, one for each antenna port. The rows contain the indices in linear form for mapping the PHICH symbols to the subframe resource grid.

To generate a subframe resource grid, use the `lteDLResourceGrid` function. This function creates an empty resource grid for one subframe.

```
subframe = lteDLResourceGrid(enb);
disp(size(subframe))
```

```
   108    14     4
```

Map the complex PHICH symbols to the resource grid by assigning `phichSymbols` to the `phichIndices` locations in `subframe`.

```
subframe(phichIndices) = phichSymbols;
```

See Also

`ltePHICH` | `ltePHICHInfo` | `ltePHICHIndices` | `ltePHICHPRBS` | `lteDLResourceGrid` | `lteLayerMap` | `lteLayerDemap` | `lteDLPrecode` | `lteDLDeprecode` | `lteCRCEncode` | `lteCRCDecode` | `lteSymbolModulate` | `lteSymbolDemodulate`

More About

- “HARQ Indicator (HI) Channel” on page 1-29

Model DCI and PDCCH

This example shows how to model the control region used in an LTE downlink subframe and its channel structure. It demonstrates how you create a DCI message, encode it, create the PDCCH, and map it to a resource grid.

Specify the cell-wide settings as fields in the structure `enb`. Many of the functions used in this example require a subset of these settings.

```
enb.NDLRB = 9;
enb.CyclicPrefix = 'Normal';
enb.PHICHDuration = 'Normal';
enb.CFI = 3;
enb.Ng = 'Sixth';
enb.CellRefP = 1;
enb.NCellID = 1;
enb.NSubframe = 0;
enb.DuplexMode = 'FDD';
```

Set up the DCI message structure.

```
dci.NDLRB = enb.NDLRB;
dci.DCIFormat = 'Format1A';
dci.Allocation.RIV = 26;
dci.DuplexMode = 'FDD';
dci.NTxAnts = 1;
```

The DCI message contains these parameters:

- `NDLRB` — number of downlink resource blocks (RBs)
- `DCIFormat` — DCI format, selected from those discussed in “DCI Message Formats” on page 1-39
- `Allocation.RIV` — resource indication value (RIV)
- `DuplexMode` — transmission frame structure type, `'FDD'` for frame structure type 1 or `'TDD'` for frame structure type 2
- `NTxAnts` — number of transmit antennas

The RIV indicates the contiguous RB allocations for a UE. The UE uses the RIV to determine the first virtual RB and the length of contiguous allocation of RBs. In this example, an RIV setting of 26 corresponds to full bandwidth assignment.

Generate a DCI message by calling the `lteDCI` function. You can map this generated message to the PDCCH.

```
[dciMessage,dciMessageBits] = lteDCI(enb,dci);
```

The `lteDCI` function returns a structure, `dciMessage`, and a vector containing the DCI message bits, `dciMessageBits`. Both outputs contain the same information, but are best suited for different purposes. The output structure is more readable, while the serialized DCI message is in a more suitable format to send to the channel coding stage.

Set up the PDCCH configuration structure. The channel coding stages require these parameters:

- number of downlink resource blocks (RBs)

- UE-specific mask (16-bit C-RNTI value)
- PDCCH format

```
pdcch.RNTI = 100;
pdcch.PDCCHFormat = 0;
```

Channel encode the DCI message bits. This process consists of the addition of a CRC attachment, convolutional coding, and rate matching according to the PDCCH format capacity.

```
codedDciBits = lteDCIEncode(pdcch,dciMessageBits);
```

The resulting vector, `codedDciBits`, has 72 elements.

Generate PDCCH bits. The encoded DCI messages are then assigned to CCEs, as discussed in “Matching PDCCHs to CCE Positions” on page 1-44. The capacity of the control region depends on the bandwidth, the CFI, the number of antenna ports and the HICH groups. You can calculate the total number of resources available for PDCCH by using the `ltePDCCHInfo` function.

```
pdcchInfo = ltePDCCHInfo(enb);
```

This function returns a structure, `pdcchInfo`, which contains the resources available to the PDCCH in different units (one per field): bits, CCEs, REs and REGs.

The total number of bits available in the PDCCH region can be found in the field `pdcchInfo.MTot`. This allows a vector to be built with the appropriate number of elements.

```
pdcchBits = -1*ones(1,pdcchInfo.MTot);
```

Not all the available bits in the PDCCH region are necessarily used. Therefore, following the convention in the LTE Toolbox™ product, set unused bits to −1. Since all elements have been initialized in `pdcchBits` to −1, this indicates that initially all the bits are unused. Now elements of `codedDciBits` can be mapped to the appropriate locations in `pdcchBits`.

Calculate indices of candidate bits by calling the `ltePDCCHSpace` function. Only a subset of all the bits in `pdcchBits` may be used, which are called the candidate bits.

```
candidates = ltePDCCHSpace(enb,pdcch,{'bits','1based'});
```

This function returns a two-column matrix. Each row contains an available candidate location for the cell-wide settings provided by `enb` and the PDCCH configuration structure `pdcch`. The first and second columns contain the indices of the first and last locations of each candidate. In this example, the indices are 1-based and refer to bits. Hence, they can be used to access locations in `pdcchBits`.

Use the first available candidate to map the coded DCI bits.

```
pdcchBits (candidates(1,1):candidates(1,2)) = codedDciBits;
```

The vector `pdcchBits` has 736 elements. The 72 bits of `codedDciBits` are mapped to the chosen candidate in `pdcchBits`. Therefore, out of 736 elements, 72 will take 0 and 1 values, while the rest remain set to −1. The `ltePDCCH` function, which is used to generate complex-modulated symbols, interprets these locations as unused and will only consider those containing 1s and 0s.

To generate the PDCCH symbols, use the `ltePDCCH` function. You can generate the PDCCH complex symbols from the set of bits used in `pdcchBits`, values not set to −1. The function performs the required scrambling, QPSK modulation, layer mapping, and precoding operations. Since there are

two bits per QPSK symbol, 368 symbols are generated. The occupied bits result in 36 non-zero QPSK symbols.

```
pdcchSymbols = ltePDCCH(enb, pdcchBits);
size(pdcchSymbols)
```

```
ans = 1×2

   368     1
```

```
size(find(pdcchSymbols))
```

```
ans = 1×2

    36     1
```

To generate PDCCH mapping indices, use the `ltePDCCHIndices` function. You can use these indices to map the complex values in `pdcchSymbols` to the subframe resource grid.

```
pdcchIndices = ltePDCCHIndices(enb,{'1based'});
size(pdcchIndices)
```

```
ans = 1×2

   368     1
```

This function returns a column vector. The rows contain the 1-based indices in linear form for mapping the PDCCH symbols to the subframe resource grid.

Map the PDCCH to the resource grid. You can easily map the complex PDCCH symbols to the resource grid for each antenna port.

- Create an empty resource grid with the `lteDLResourceGrid` function.
- Map the `pdcchSymbols` to the `pdcchIndices` index locations of the `subframe` resource grid.

```
subframe = lteDLResourceGrid(enb);
subframe(pdcchIndices) = pdcchSymbols;
```

See Also

`lteDCI` | `lteDCIEncode` | `ltePDCCH` | `ltePDCCHIndices` | `ltePDCCHSpace` | `ltePDCCHInfo` | `ltePDCCHPRBS` | `ltePDCCHInterleave` | `ltePDCCHDeinterleave` | `ltePDCCHDecode` | `lteLayerMap` | `lteLayerDemap` | `lteDLPrecode` | `lteDLDeprecode` | `lteCRCEncode` | `lteCRCDecode` | `lteConvolutionalEncode` | `lteConvolutionalDecode` | `lteRateMatchConvolutional` | `lteRateRecoverConvolutional` | `lteSymbolModulate` | `lteSymbolDemodulate`

More About

- “Downlink Control Channel” on page 1-39

Model PUCCH Format 1

This example shows how to model the control region used in an LTE uplink subframe and its channel structure. It demonstrates how you create the physical uplink control channel (PUCCH) format 1 structures and map the generated symbols to a resource grid.

Specify the user-equipment (UE) settings in the structure `ue`.

```
ue1.NCellID = 10;
ue1.CyclicPrefixUL = 'Normal';
ue1.NSubframe = 0;
ue1.Hopping = 'Off';
ue1.NULRB = 9;
ue1.Shortened = 0;
```

Many of the functions used in this example require a subset of the preceding settings specified.

Configure the PUCCH. In addition to the UE settings specified in `ue`, you must define parameters related to the physical channel to generate the PUCCH Format 1.

```
pucch1.ResourceIdx = 0;
pucch1.DeltaShift = 1;
pucch1.CyclicShifts = 6;
```

Generate the PUCCH Format 1 symbols by calling the `ltePUCCH1` function, providing the UE and PUCCH configuration structures as input arguments.

```
hi = [0 0];
pucch1symbols = ltePUCCH1(ue1,pucch1,hi);
```

The variable `hi` specifies the HARQ Indicator bits.

Generate the PUCCH Format 1 indices by calling the `ltePUCCH1Indices` function. You can use these generated indices to map the PUCCH complex symbols to the subframe resource grid. This function requires the same input argument structures as the `ltePUCCH1` function.

```
pucch1indices = ltePUCCH1Indices(ue1,pucch1);
```

Generate the PUCCH Format 1 demodulation reference signals (DRS) by calling the `ltePUCCH1DRS` function. This function requires the same input argument structures as the `ltePUCCH1` and `ltePUCCH1Indices` functions.

```
drs1 = ltePUCCH1DRS(ue1,pucch1);
```

Generate the PUCCH Format 1 DRS indices by calling the `ltePUCCH1DRSIndices` function. These indices map the DRS to the subframe resource grid.

```
drs1indices = ltePUCCH1DRSIndices(ue1,pucch1);
```

Generate the subframe resource grid by calling the `lteULResourceGrid` function. This function creates an empty resource grid for one subframe.

```
subframe = lteULResourceGrid(ue1);
```

Map the PUCCH Format 1 symbols and DRS to the resource grid using the generated indices.

```
subframe(pucch1indices) = pucch1symbols;
subframe(drs1indices) = drs1;
```

See Also

`ltePUCCH1` | `ltePUCCH1Indices` | `ltePUCCH1DRS` | `ltePUCCH1DRSIndices` | `lteULResourceGrid`

More About

- “Uplink Control Channel Format 1” on page 1-56

Model PUCCH Format 2

This example shows how to model the control region used in an LTE uplink subframe and its channel structure. It demonstrates how you create the physical uplink control channel (PUCCH) format 2 structures and map the generated symbols to a resource grid.

Specify user equipment (UE) settings in a structure, `ue`. Many of the functions used in this example require a subset of these settings.

```
ue.NCellID = 10;
ue.CyclicPrefixUL = 'Normal';
ue.NSubframe = 0;
ue.Hopping = 'Off';
ue.NULRB = 9;
ue.RNTI = 77;
```

Configure the PUCCH Format 2. In addition to the UE settings specified in `ue`, you must define parameters related to the physical channel to generate the PUCCH Format 2.

```
pucch2.ResourceIdx = 36;
pucch2.ResourceSize = 3;
pucch2.CyclicShifts = 6;
```

Generate the UCI message from the CQI bits.

```
cqi = [0 1 1 0 0 1];
codedCQI = lteUCIEncode(cqi);
```

Generate the PUCCH Format 2 symbols by calling the `ltePUCCH2` function, providing the UE settings, PUCCH configuration, and UCI message as input arguments.

```
pucch2Sym = ltePUCCH2(ue,pucch2,codedCQI);
```

Generate the PUCCH Format 2 indices by calling the `ltePUCCH2Indices` function. You can use these generated indices to map the PUCCH complex symbols to the subframe resource grid. This function requires the same input argument structures as the `ltePUCCH2` function.

```
pucch2Indices = ltePUCCH2Indices(ue,pucch2);
```

Generate the PUCCH Format 2 demodulation reference signals (DRS) by calling the `ltePUCCH2DRS` function. This function requires the same input argument structures as the `ltePUCCH2` and `ltePUCCH2Indices` functions. Since no HARQ bits are transmitted, specify an empty vector as the third input argument of the function.

```
pucch2DRSSym = ltePUCCH2DRS(ue,pucch2,[]);
```

Generate the PUCCH Format 2 DRS indices by calling the `ltePUCCH2DRSIndices` function. You can use these indices to map the DRS to the subframe resource grid.

```
pucch2DRSIndices = ltePUCCH2DRSIndices(ue,pucch2);
```

Generate the subframe resource grid by calling the `lteULResourceGrid` function. This function creates an empty resource grid for one subframe.

```
subframe = lteULResourceGrid(ue);
```

Map the PUCCH Format 2 symbols and DRS to the resource grid using the generated indices.

```
subframe(pucch2Indices) = pucch2Sym;
subframe(pucch2DRSIndices) = pucch2DRSSym;
```

See Also

`ltePUCCH2` | `ltePUCCH2Indices` | `ltePUCCH2DRS` | `ltePUCCH2DRSIndices` | `lteULResourceGrid`

More About

- “Uplink Control Channel Format 2” on page 1-63

Model DL-SCH and PDSCH

This example shows how to construct the physical downlink shared channel (PDSCH). It also demonstrates how to generate a transport block, perform downlink shared channel (DL-SCH) coding to create a codeword, perform physical channel coding to create the physical channel, and map the complex symbols to the resource grid.

Specify cell-wide settings as fields in the structure `enb`. Many of the functions used in this example require a subset of these settings.

```
enb.NDLRB = 9;
enb.CyclicPrefix = 'Normal';
enb.PHICHDuration = 'Normal';
enb.CFI = 3;
enb.Ng = 'Sixth';
enb.CellRefP = 4;
enb.NCellID = 1;
enb.NSubframe = 0;
enb.DuplexMode = 'FDD';
```

Configure the PDSCH. In addition to the cell-wide settings specified in `enb`, you must define other parameters related to the modulation and channel transmission configuration, `pdsch`, such as the radio network temporary identifier (RNTI), to generate the PDSCH.

```
pdsch.NTxAnts = 4;
pdsch.NLayers = 4;
pdsch.TxScheme = 'TxDiversity';
pdsch.Modulation = {'QPSK'};
pdsch.RV = 0;
pdsch.RNTI = 1;
```

In this example, you use a single codeword to form the PDSCH symbols. However, in LTE, up to two codewords can be combined to form the PDSCH. Each codeword can be modulated with a different scheme. Use a cell array to indicate the modulation scheme for each codeword.

Determine how the PDSCH is mapped to resource elements by allocating the physical resource blocks (PRBs). A column vector containing the indices of PDSCH allocated PRBs is required. In this example, assume full allocation; all resource blocks are allocated to the PDSCH. Specify this full subframe resource allocation using a column vector.

```
prbs = (0:enb.NDLRB-1).';
```

The allocation specified in `prbs` is zero-based. In this case, assume that both slots in the subframe share the same resource allocation. To have different allocations for each slot, specify a two-column matrix where each column refers to each slot in the subframe.

Generate the PDSCH indices. To do so, call the `ltePDSCHIndices` function for the cell-wide settings `enb`, the channel transmission configuration `pdsch`, and the physical resource block allocation `prbs`.

```
[pdschIndices,pdschIndInfo] = ltePDSCHIndices(enb,pdsch,prbs,{'1based'});
```

The first output, `pdschIndices`, specifies the PDSCH indices. The second output, `pdschIndInfo`, provides additional information related to the PDSCH capacity.

Determine DL-SCH payload and coded transport block size. These items are required for creating the PDSCH payload due to the rate matching portion of the DL-SCH transport block coding. There are the following two methods of determining coded transport block size and the DL-SCH payload size.

- Using the PDSCH indices information output, as shown in this example
- Using the reference measurement channel (RMC) transport block sizes as a guide

The coded transport block size is one of the fields of the PDSCH indices information output, `pdschIndInfo`.

```
codedTrBlkSize = pdschIndInfo.G;
```

In this example, `codedTrBlkSize` is 480. Alternatively, you could read the coded transport block size for a given modulation scheme, PRB allocation, and number of antennas from the RMC tables in TS 36.101, Annex A.3.3 and A.3.4. Once you know the coded transport block size, calculate the DL-SCH payload using the rules in TS 36.101, Annex A.2.1.2, titled, "Determination of payload size", with target code rate, R, equal to 1/3, and the number of bits per subframe given by `codedTrBlkSize`. Determine the payload size, A, such that the resulting coding rate is as close as possible to the desired coding rate, R, for a given coded transport block size, N_{ch}, as shown in the following equation.

$$\min |R - (A + 24)/N_{\mathrm{ch}}|$$

In this example, the payload size for 6 RBs calculated using the preceding equation is A=152. This is the value at which the error between the desired code rate and actual code rate is minimized.

The payload size, A, must be one of a specific set for a specific number of resource blocks given in TS 36.213, Table 7.1.7.2.1 1 or 7.1.7.2.2 1 (in Section 7.1.7.2). These tables are represented by the `lteTBS` function. In this example, the payload size for 6 RBs that minimizes the error between the desired code rate and the actual code rate is A = 152. This value was selected from table 7.1.7.2.2 1. Therefore, the payload size, `transportBlkSize`, is 152.

Alternative Method: Use RMCs to Determine Transport Block Sizes

Alternatively, you could determine suitable payload size and coded transport block size from the tables in TS 36.101, Annex A.3.3 and A.3.4, titled "Reference Measurement Channels for PDSCH performance requirements." Despite the advantage of simply being able to read values from the tables, the channel bandwidths and PDSCH allocations are restricted to the RMCs available. For example, you can use Table A.3.3.2.2-1, titled "Fixed Reference Channel four antenna ports."

To generate a PDSCH for a 1.4MHz channel bandwidth, four-antenna transmission with QPSK modulation, and a coding rate of , use the highlighted rows titled "Information Bit Payload" to find the DL-SCH payload size for each subframe, and "Binary Channel Bits," to find the coded transport block size for each subframe.

Parameter	Unit	Value				
Reference Channel		*R*.12 *FDD*	*R*.13 *FDD*	*R*.14 *FDD*	*R*.14 − 1 *FDD*	
Channel bandwidth	*MHz*	1.4	10	10	10	
Allocated resource blocks		6	50	50	6	
Allocated subframes per Radio Frame		9	9	9	8	
Modulation		*QPSK*	*QPSK*	16*QAM*	16*QAM*	
Target Coding Rate		1/3	1/3	1/2	1/2	
Information Bit Payload						
For Sub − Frames 1, 2, 3, 4, 6, 7, 8, and 9	*Bits*	408	4392	12960	1544	
For Sub − Frame 5	*Bits*	*n*/*a*	*n*/*a*	*n*/*a*	*n*/*a*	
For Sub − Frame 0	*Bits*	152	3264	11448	*n*/*a*	
Number of Code Blocks						
For Sub − Frames 1, 2, 3, 4, 6, 7, 8, and 9		1	1	3	1	
For Sub − Frame 5		*n*/*a*	*n*/*a*	*n*/*a*	*n*/*a*	
For Sub − Frame 0		1	1	2	*n*/*a*	
Binary Channel Bits per Sub − Frame						
For Sub − Frames 1, 2, 3, 4, 6, 7, 8, and 9	*Bits*	1248	12800	25600	3072	
For Sub − Frame 5	*Bits*	*n*/*a*	*n*/*a*	*n*/*a*	*n*/*a*	
For Sub − Frame 0	*Bits*	480	12032	24064	*n*/*a*	
Max. Throughput averaged over 1 frame	*Mbps*	0.342	3.876	11.513	1.235	
UE Category		≥ 1	≥ 1	≥ 2	≥ 1	

Define a transport block of information bits, using the payload size, `transportBlkSize`, calculated in the last step.

```
dlschTransportBlk = round(rand(1,152));
```

Create the PDSCH payload. To encode the transport block bits into a single codeword, call the `lteDLSCH` function for the cell-wide settings and channel transmission configuration. This process includes CRC calculation, code block segmentation and CRC insertion, turbo coding, rate matching, and code block concatenation.

```
codeword = lteDLSCH(enb,pdsch,codedTrBlkSize,dlschTransportBlk);
```

Generate the PDSCH complex symbols by calling the `ltePDSCH` function for the specified cell-wide settings, channel transmission configuration, and codeword. This function applies scrambling, modulation, layer mapping, and precoding operations to the coded transport block.

```
pdschSymbols = ltePDSCH(enb,pdsch,codeword);
```

The resulting matrix, `pdschSymbols`, has four columns. Each column contains the complex symbols to map to each antenna port.

Generate the subframe resource grid by calling the `lteDLResourceGrid` function. This function creates an empty resource grid for one subframe.

```
subframe = lteDLResourceGrid(enb);
```

Map the PDSCH symbols to the resource grid using the generated indices.

```
subframe(pdschIndices) = pdschSymbols;
```

See Also

`lteDLSCH` | `ltePDSCH` | `ltePDSCHIndices` | `lteDLResourceGrid` | `lteDLSCHInfo` | `lteDLSCHDecode` | `lteLayerMap` | `lteLayerDemap` | `lteDLPrecode` | `lteDLDeprecode` | `lteTurboEncode` | `lteTurboDecode` | `ltePDSCHPRBS` | `lteCRCEncode` | `lteCRCDecode` | `lteCodeBlockSegment` | `lteCodeBlockDesegment` | `lteRateMatchTurbo` | `lteRateRecoverTurbo`

More About

- “Downlink Shared Channel” on page 1-71

Model UL-SCH and PUSCH

This example shows how to construct the physical uplink shared channel (PUSCH). It demonstrates how to generate a transport block, perform uplink shared channel (UL-SCH) coding to create a codeword, perform physical channel coding to create the physical channel, and map the complex symbols to the resource grid.

Specify user-equipment (UE) settings in the structure `ue`. Many of the functions used in this example require a subset of these fields.

```
ue.NULRB = 9;
ue.NSubframe = 0;
ue.NCellID = 10;
ue.RNTI = 1;
ue.CyclicPrefixUL = 'Normal';
ue.Hopping = 'Off';
ue.SeqGroup = 0;
ue.CyclicShift = 0;
ue.Shortened = 0;
```

Configure the PUSCH. In addition to the UE settings specified in `ue`, you must define parameters related to the physical channel to generate the PUSCH.

```
pusch.PRBSet = (0:5).';
pusch.Modulation = 'QPSK';
pusch.RV = 0;
pusch.DynCyclicShift = 0;
```

Generate the subframe resource grid by calling the `lteULResourceGrid` function. This function creates an empty resource grid for one subframe.

```
subframe = lteULResourceGrid(ue);
```

Generate the UL-SCH message by calling the `lteULSCH` function, providing the transport block data `trblk`, UE-specific structure `ue`, and channel-specific structure `pusch` as input arguments.

```
trblk = round(rand(1,504));
cw = lteULSCH(ue,pusch,trblk);
```

Generate the PUSCH symbols by calling the `ltePUSCH` function, providing the UE settings, PUSCH configuration, and codeword as input arguments.

```
puschSymbols = ltePUSCH(ue,pusch,cw);
```

Generate the PUSCH indices by calling the `ltePUSCHIndices` function. You can use these generated indices to map the PUSCH complex symbols to the subframe resource grid. This function requires the same input argument structures as the `ltePUSCH` function.

```
puschIndices = ltePUSCHIndices(ue,pusch);
```

Generate the PUSCH DRS symbols by calling the `ltePUSCHDRS` function. This function requires the same input argument structures as the `ltePUSCH` function.

```
drsSymbols = ltePUSCHDRS(ue,pusch);
```

Generate the PUSCH DRS indices by calling the `ltePUSCHDRSIndices` function. You can use these indices to map the DRS to the subframe resource grid.

```
drsIndices = ltePUSCHDRSIndices(ue,pusch);
```

Map the PUSCH symbols and DRS to the resource grid using the generated indices.

```
subframe(puschIndices) = puschSymbols;
subframe(drsIndices) = drsSymbols;
```

See Also

`lteULSCH` | `lteULSCHInfo` | `ltePUSCH` | `ltePUSCHIndices` | `ltePUSCHDRS` | `ltePUSCHDRSIndices` | `lteULResourceGrid`

More About

- “Uplink Shared Channel” on page 1-90

Simulate Propagation Channels

This example shows how to simulate propagation channels. It demonstrates how to generate cell-specific reference signals, map them onto a resource grid, perform OFDM modulation, and pass the result through a fading channel.

Specify the cell-wide settings as fields in the structure `enb`. Many of the functions used in this example require a subset of these fields.

```
enb.NDLRB = 9;
enb.CyclicPrefix = 'Normal';
enb.PHICHDuration = 'Normal';
enb.CFI = 3;
enb.Ng = 'Sixth';
enb.CellRefP = 1;
enb.NCellID = 10;
enb.NSubframe = 0;
enb.DuplexMode = 'FDD';
antennaPort = 0;
```

Resource Grid and Transmission Waveform

Generate a subframe resource grid. To create the resource grid, call the `lteDLResourceGrid` function. This function creates an empty resource grid for one subframe.

```
subframe = lteDLResourceGrid(enb);
```

Generate cell-specific reference symbols (CellRS) and map them onto the resource elements (REs) of a resource grid using linear indices.

```
cellRSsymbols = lteCellRS(enb,antennaPort);
cellRSindices = lteCellRSIndices(enb,antennaPort,{'1based'});
subframe(cellRSindices) = cellRSsymbols;
```

Perform OFDM modulation of the complex symbols in a subframe, `subframe`, using cell-wide settings structure `enb`.

```
[txWaveform,info] = lteOFDMModulate(enb,subframe);
```

The first output argument, `txWaveform`, contains the transmitted OFDM modulated symbols. The second output argument, `info`, is a structure that contains details about the modulation process. The field `info.SamplingRate` provides the sampling rate, R_{sampling}, of the time domain waveform:

$$R_{\text{sampling}} = \frac{30.72\ \text{MHz}}{2048 \times N_{\text{FFT}}},$$

where N_{FFT} is the size of the OFDM inverse Fourier transform (IFT).

Propagation Channel

Construct the LTE multipath fading channel. First, set up the channel parameters by creating a structure, `channel`.

```
channel.Seed = 1;
channel.NRxAnts = 1;
```

```
channel.DelayProfile = 'EVA';
channel.DopplerFreq = 5;
channel.MIMOCorrelation = 'Low';
channel.SamplingRate = info.SamplingRate;
channel.InitTime = 0;
```

The sampling rate in the channel model, `channel.SamplingRate`, must be set to the `info` field of the `SamplingRate` returned by the `lteOFDMModulate` function.

Pass data through the LTE fading channel by calling the `lteFadingChannel` function. This function generates an LTE multipath fading channel, as specified in TS 36.101 (See reference [1]). The first input argument, `txWaveform`, is an array of LTE transmitted samples. Each row contains the waveform samples for each of the transmit antennas. These waveforms are filtered with the delay profiles as specified in the parameter structure, `channel`.

```
rxWaveform = lteFadingChannel(channel,txWaveform);
```

Received Waveform

The output argument, `rxWaveform`, is the channel output signal matrix. Each row corresponds to the waveform at each of the receive antennas. Since you have defined one receive antenna, the number of rows in the `rxWaveform` matrix is one.

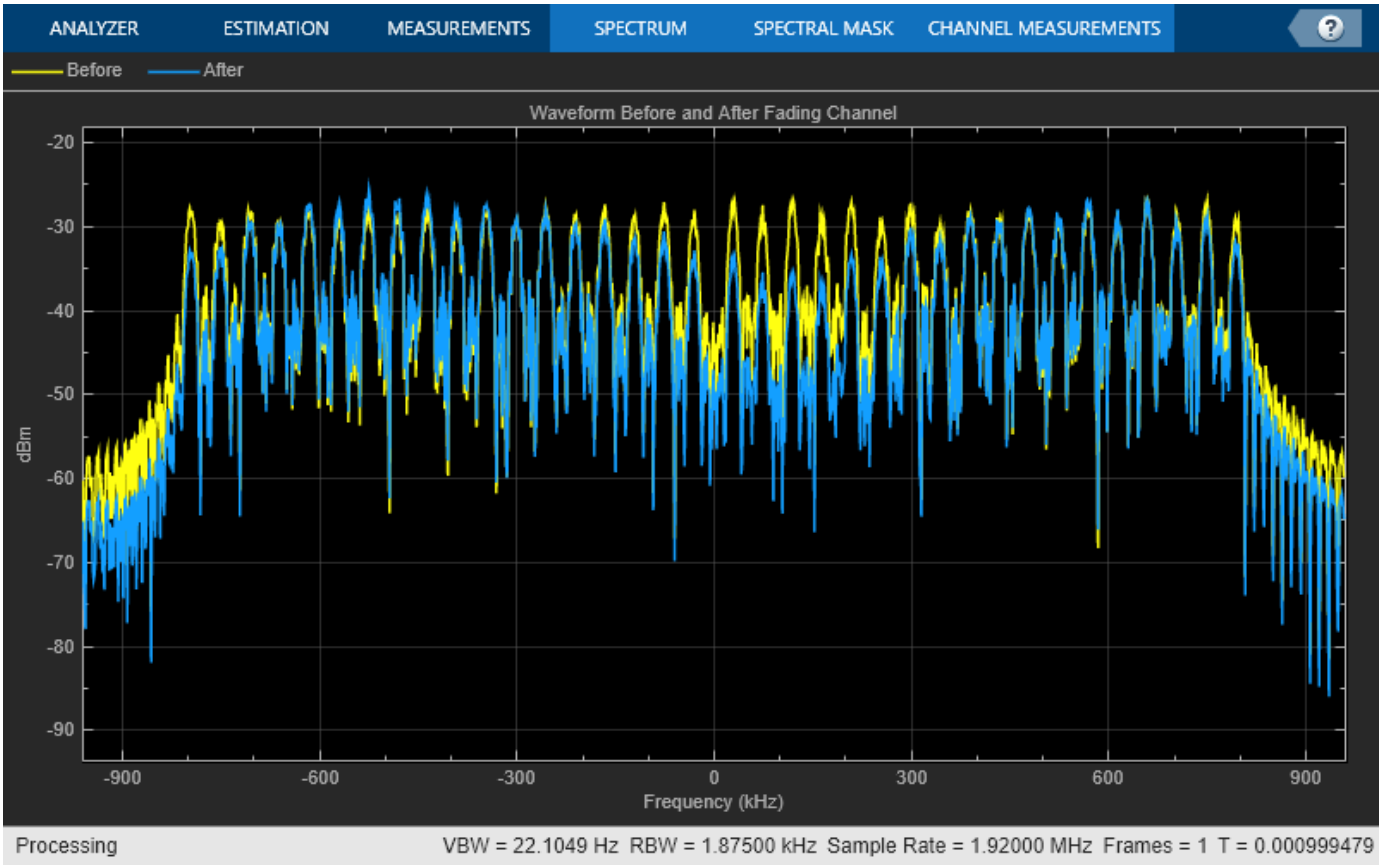
```
size(rxWaveform)
```

```
ans = 1×2

        1920           1
```

Plot Signal Before and After Fading Channel

Display a spectrum analyzer with before-channel and after-channel waveforms, using Welch's spectrum estimation method.

```
title = 'Waveform Before and After Fading Channel';
saScope = spectrumAnalyzer(SampleRate=info.SamplingRate, Method='welch', ShowLegend=true,...
     Title = title,ChannelNames={'Before','After'});
saScope([txWaveform,rxWaveform]);
```



References

1 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception".

See Also

`lteFadingChannel` | `lteHSTChannel` | `lteMovingChannel`

Related Examples

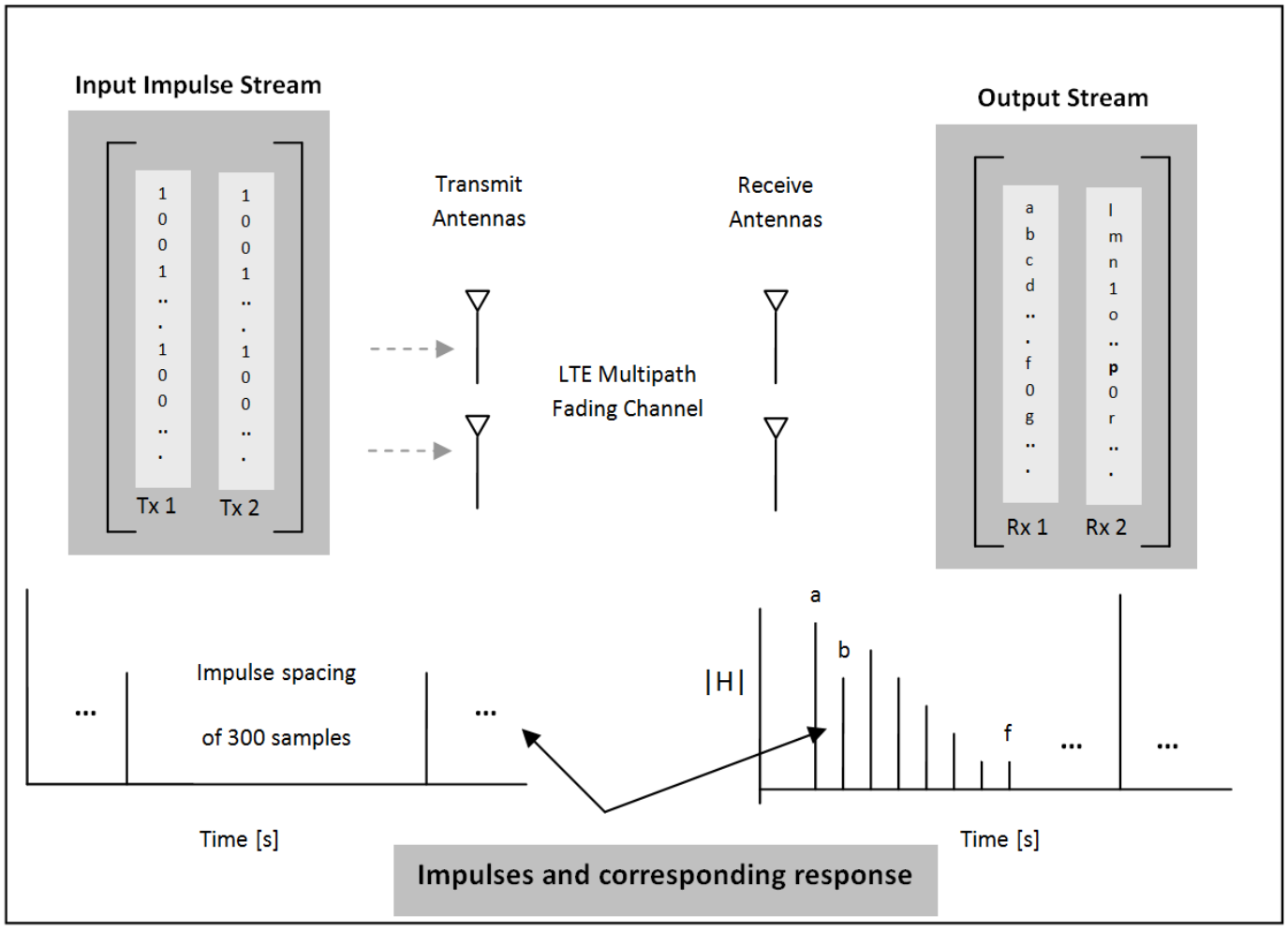
- “Find Channel Impulse Response” on page 3-25

More About

- “Propagation Channel Models” on page 1-106

Find Channel Impulse Response

This example shows how to find the channel impulse response of a 2-by-2 MIMO system. The input is a matrix of impulses where each impulse is separated by 300 samples. Each column in the matrix, the size of which is the number of transmit antennas, is the input waveform to the channel model function and is therefore a series of impulses. This series of impulses allows the changing impulse response of the channel to be viewed over time. For clear visualization, the impulse spacing should be greater than maximum delay spread of the channel. The input waveform is passed through the LTE multipath fading channel model. The output matrix has complex samples corresponding to each receive antenna.



Pre-configure the LTE multipath fading channel. To do so, set up a simple structure and specify the fading channel parameters.

```
channel.Seed = 1;
channel.NRxAnts = 2;
channel.DelayProfile = 'EVA';
channel.DopplerFreq = 300;
channel.CarrierFreq = 2e9;
channel.MIMOCorrelation = 'Low';
```

```
channel.SamplingRate = 1/10e-9;
channel.InitTime = 0;
channel.InitPhase = 'Random';
channel.ModelType = 'GMEDS';
channel.NTerms = 16;
channel.NormalizeTxAnts = 'On';
channel.NormalizePathGains = 'On';
```

Create two identical input streams of data. These input streams are passed through two transmit antennas, as shown in the preceding figure.

```
nAntIn = 2;
impulseSpacing = 300;
noImpResponse = 150;
nInputSamples = impulseSpacing * noImpResponse;
in = zeros(nInputSamples, nAntIn);
onesLocations = 1:impulseSpacing:nInputSamples;
in(onesLocations,1) = 1;
```

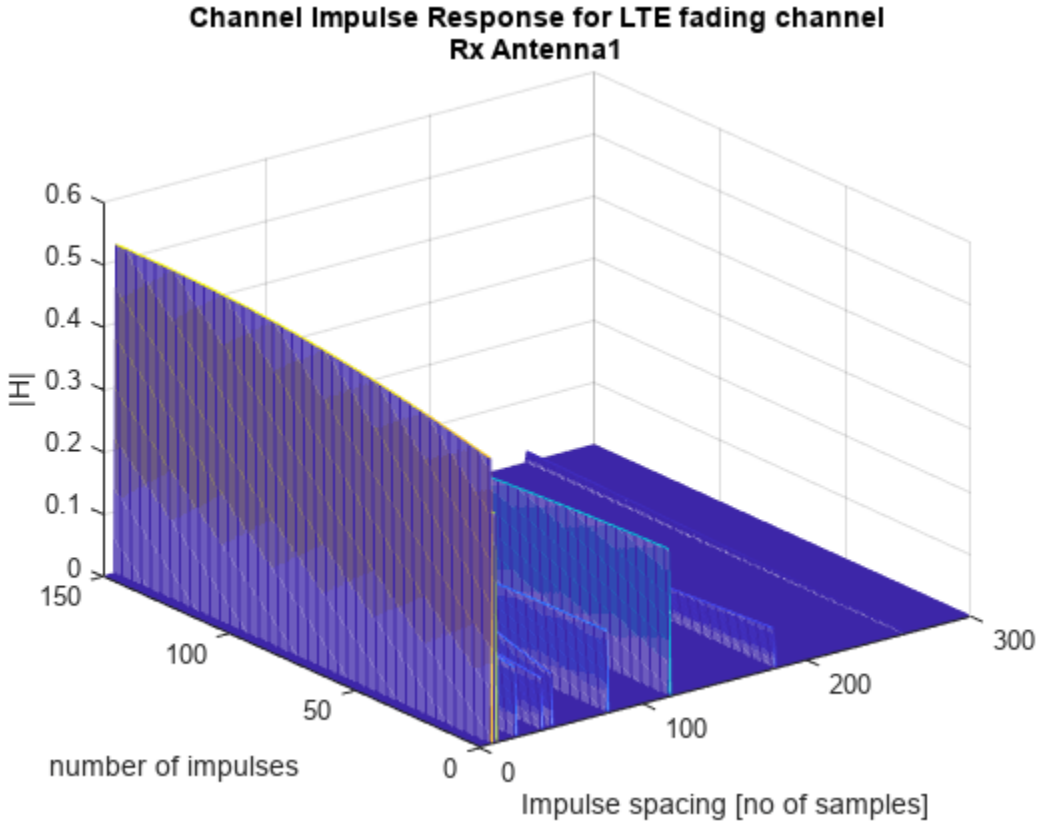
The variable `nAntIn` is the number of transmit antennas. The variable `impulseSpacing` is greater than the maximum channel delay spread. The variable `noImpResponse` is the number of impulse responses to calculate.

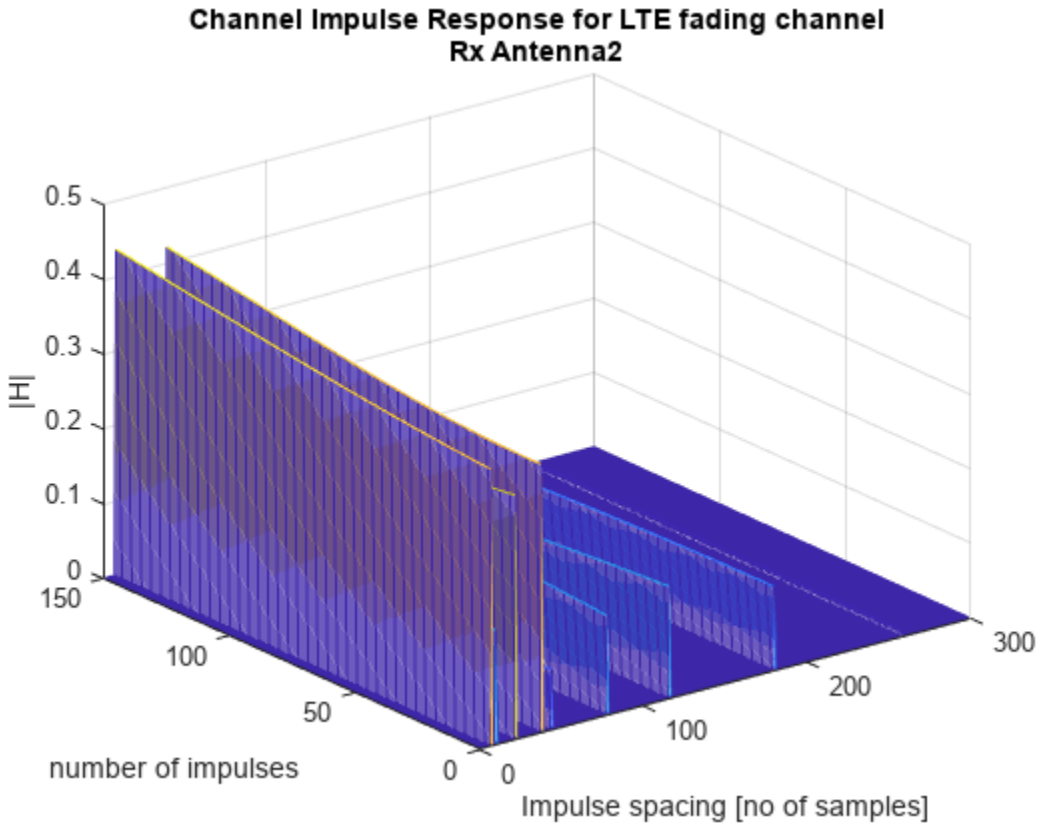
Filter with the LTE fading channel. To do so, call the `lteFadingChannel` function. This function generates an LTE multi-path fading channel, as specified in TS 36.101 [1]. The first input argument, `in`, is an array of LTE transmitted samples. Each row contains the waveform samples for each of the transmit antennas. These waveforms are filtered with the delay profiles as specified in the parameter structure, `channel`.

```
out = lteFadingChannel(channel,in);
```

Finally, plot the receive waveform, showing the channel impulse response for both receive antennas.

```
for antNo = 1:channel.NRxAnts
    figure
    mesh(squeeze(abs(reshape(out(:,antNo), ...
        impulseSpacing,noImpResponse).')))
    titleStr = ['Rx Antenna' num2str(antNo)];
    title({'Channel Impulse Response for LTE fading channel',titleStr})
    ylabel('number of impulses')
    xlabel('Impulse spacing [no of samples]')
    zlabel('|H|')
end
```


Channel Impulse Response for LTE fading channel
Rx Antenna1
|H|
0.6
0.5
0.4
0.3
0.2
0.1
0
150
100
50
0
number of impulses
0
100
200
300
Impulse spacing [no of samples]



References

1 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception".

See Also

`lteFadingChannel` | `lteHSTChannel` | `lteMovingChannel`

Related Examples

- “Simulate Propagation Channels” on page 3-22

More About

- “Propagation Channel Models” on page 1-106

UMTS Concepts

UMTS Parameterization Overview

In this section...

Many parameters must be defined to generate UMTS waveforms. To organize parameters for initialization, review, and use, the LTE Toolbox product groups relevant UMTS parameters into structures.

To generate downlink or uplink UMTS waveforms, you must define the waveform properties and the channels you want to include in the waveforms. Separate downlink and uplink configuration structures consolidate the parameters required to initialize and generate a waveform for the target link direction.

Downlink Reference Channel and Waveform Generation Parameter Structures

Defining a downlink waveform requires initialization of a handful of top-level parameters and substructures associated with permissible channels. The top-level parameters include `TotFrames`, `PrimaryScramblingCode`, `FilterType`, `OversamplingRatio`, and `NormalizedPower`. The channel substructures can include combinations of these channels: `DPCH`, `PCCPCH`, `SCCPCH`, `PCPICH`, `SCPICH`, `PSCH`, `SSCH`, `PICH`, `HSDPA`, and `OCNS`. You only include and initialize the individual channel substructures needed to generate desired waveform.

The `umtsDownlinkReferenceChannels` function initializes the configuration structure based on an input character vector argument aligning with one of the reference channels defined in these 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].

You can generate waveforms using `umtsDownlinkWaveformGenerator` with an input configuration structure that you:

- Initialize by calling `umtsDownlinkReferenceChannels`, specifying one of the predefined reference channels as an input
- Initialize to one of the predefined reference channels as above, and adjust settings manually
- Manually initialize, complying with the structure defined as input to `umtsDownlinkWaveformGenerator`

This image shows the downlink reference channel configuration structure fields. The left side of the figure expands the substructures to reveal fields in the top half of the structure. The right side of the figure expands the substructures to reveal fields in the bottom half of the structure.

Note The single output data stream from the TrCH multiplexing, including the downlink DTX indication bits, is denoted as the coded composite transport channel (CCTrCH). A CCTrCH can be

mapped to one or several physical channels. Each physical channel substructure (`DPCH`, `PCCPH`, and `SCCPCH`) can contain one `CCTrCH` substructure which in turn contains one or more `TrCH` substructures. Each `CCTrCH` substructure is individually initialized and fully parameterizable. The general TrCH coding and multiplexing and the CCTrCH processing are defined in TS 25.212, Section 4.2 [3]. TS 25.302, Section 6 [4] specifies C/I, the power control and duplex mode restrictions when mapping multiple CCTrCH on a physical channel.

The config structure with DPCH, CCTrCH, PCCPCH, and SCCPCH substructures expanded to show fields.

The config structure with PCPICH, SCPICH, PSCH, SSCH, PICH, HSDPA, HSDSCH, and OCNS substructures expanded to show fields.

Note Each instance of the `CCTrCH` substructure contains the same fields, so the contents is only expanded in the first appearance the figure.

Uplink Reference Channel and Waveform Generation Parameter Structures

Defining an uplink waveform requires initialization of a handful of top-level parameters and substructures associated with permissible channels. The top-level parameters include `TotFrames`, `ScramblingCode`, `FilterType`, `OversamplingRatio`, and `NormalizedPower`. The channel substructures can include combinations of these channels `DPDCH`, `DPCCH`, `HSUPA`, and `HSDPCCH`. Each channel substructure includes the fields necessary to specify the indicated channel. You only include and initialize the individual channel substructures needed to generate desired waveform.

The `umtsUplinkReferenceChannels` function outputs a configuration structure based on an input character vector argument, which maps to one of the reference channels defined in these 3GPP standards:

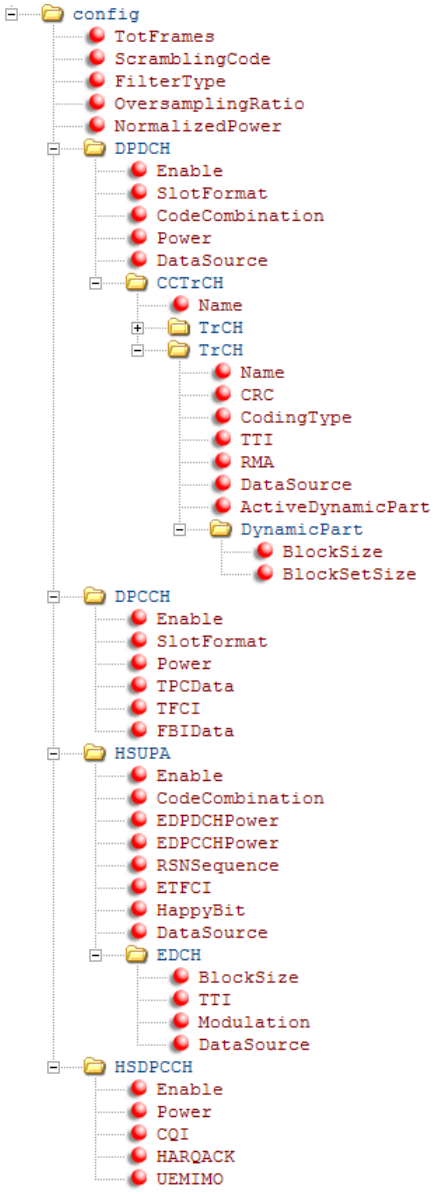
- Uplink RMC configurations, as defined in TS25.101, Annex A2 [1].
- Uplink E-DPDCH FRC configurations, as defined in TS 25.141, Annex A10, [2].

You can generate waveform using `umtsUplinkWaveformGenerator` with an input configuration structure that you:

- Initialize by calling `umtsUplinkReferenceChannels`, specifying one of the predefined reference channels as an input
- Initialize to one of the predefined reference channels as above, and adjust settings manually
- Manually initialize complying with the structure defined as input to `umtsUplinkWaveformGenerator`

This image shows the uplink reference channel configuration structure.

Note The single data stream output from the TrCH multiplexing is denoted as the coded composite transport channel (CCTrCH). A CCTrCH can be mapped to one or more physical channels. The `DPDCH` substructure can contain one or more `CCTrCH` substructures. Each `CCTrCH` substructure is individually initialized and fully parameterizable. The general TrCH coding and multiplexing and the CCTrCH processing are defined in TS 25.212, Section 4.2 [3]. TS25.302, Section 6 of [4] specifies C/I, the power control and duplex mode restrictions when mapping multiple CCTrCH on the physical channel.



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 25.212. “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.

[4] 3GPP TS 25.302. “Universal Mobile Telecommunications System (UMTS); Services provided by the physical layer.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. URL: https://www.3gpp.org.

See Also

`umtsDownlinkWaveformGenerator` | `umtsUplinkReferenceChannels` | `umtsUplinkWaveformGenerator`