5G Toolbox™

Reference

MATLAB®

R2022b

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

5G Toolbox™ Reference

Revision History

September 2018	Online only	New for Version 1.0 (Release 2018b)
March 2019	Online only	Revised for Version 1.1 (Release 2019a)
September 2019	Online only	Revised for Version 1.2 (Release 2019b)
March 2020	Online only	Revised for Version 2.0 (Release 2020a)
September 2020	Online only	Revised for Version 2.1(Release 2020b)
March 2021	Online only	Revised for Version 2.2 (Release 2021a)
September 2021	Online only	Revised for Version 2.3 (Release 2021b)
March 2022	Online only	Revised for Version 2.4 (Release 2022a)
September 2022	Online only	Revised for Version 2.5 (Release 2022b)

Contents

Functions

addLinkTypeDecoder

Add custom link layer protocol decoder to PCAP file reader

Syntax

```
addLinkTypeDecoder(pcap,linkType,linkName,protocolDecoder)
addLinkTypeDecoder(pcap,linkType,linkName,protocolDecoder,protocolFields)
```

Description

You can use the `addLinkTypeDecoder` object function to plug in a custom link layer protocol decoder to the `pcapReader` object based on the link type of the packet capture (PCAP) global header. For example, to add a Bluetooth low energy (LE) link layer decoder for decoding Bluetooth LE link layer packets, specify the link type as 251. For information on link layer header types and the corresponding link type values, see Tcpdump/Libpcap Public Repository [1].

`addLinkTypeDecoder(pcap,linkType,linkName,protocolDecoder)` adds a custom link layer protocol decoder function handle, `protocolDecoder`, to the PCAP file reader, `pcap`, based on the link type of the PCAP global header, `linkType`. The `linkName` input specifies a name for the link type.

`addLinkTypeDecoder(pcap,linkType,linkName,protocolDecoder,protocolFields)` additionally specifies protocol fields to filter packets based on the specified fields of the protocol decoder output.

Examples

Add Bluetooth LE Link Layer Decoder to PCAP File Reader

Create a protocol decoder function handle for decoding Bluetooth LE link layer packets.

```
bleDecoderHandle = @decodeBLEPacket;
```

Create a PCAP file reader object, specifying the name of a PCAP file.

```
pcapReaderObj = pcapReader('blePackets.pcap');
```

Add the Bluetooth LE link layer decoder to the PCAP file reader.

```
addLinkTypeDecoder(pcapReaderObj,251,'ble',bleDecoderHandle, ...
    {'AccessAddress','hexadecimal'});
```

Display the Bluetooth LE link layer decoder.

```
pcapReaderObj.LinkTypeDecoders(end)
```

```
ans = struct with fields:
    ProtocolName: 'ble'
        LinkType: 251
         Decoder: @decodeBLEPacket
```

Read all of the Bluetooth LE link layer packets from the PCAP file to the MATLAB® workspace.

```
blePackets = readAll(pcapReaderObj)
```

```
blePackets=1×15 struct array with fields:
    SNo
    Timestamp
    LinkType
    Protocol
    PacketLength
    Packet
    RawBytes
```

Display the decoded Bluetooth LE link layer packet structure.

```
blePackets(1).Packet
```

```
ans = struct with fields:
    ble: [1x1 struct]
```

Reset the position of the PCAP file reader to the first packet of the PCAP file.

```
reset(pcapReaderObj);
```

Read the Bluetooth LE link layer packets that match the filter criteria.

```
blePackets = readAll(pcapReaderObj,'ble.AccessAddress == 8E89BED6')
```

```
blePackets=1×13 struct array with fields:
    SNo
    Timestamp
    LinkType
    Protocol
    PacketLength
    Packet
    RawBytes
```

Display the decoded Bluetooth LE link layer packet structure.

```
blePackets(1).Packet
```

```
ans = struct with fields:
    ble: [1x1 struct]
```

Input Arguments

pcap — PCAP file reader
`pcapReader` object

PCAP file reader, specified as a `pcapReader` object.

`linkType` — Link type in PCAP global header
nonnegative integer

Link type in the PCAP global header, specified as a nonnegative integer. If the link type in the PCAP file matches the specified link type value, the `read` or `readAll` object functions calls the decoder function handle.

Data Types: `double`

`linkName` — Link layer protocol decoder name
character vector | string scalar

Link layer protocol decoder name, specified as a character vector or a string scalar. To store the decoded packet of the link layer protocol decoder, the `read` or `readAll` object functions use this value to create a new field in the `Packet` field of the output decoded protocol packet structure.

Data Types: `char` | `string`

`protocolDecoder` — Link layer protocol decoder
function handle

Link layer protocol decoder that decodes the payload and returns the decoded packet with the processed length, specified as a function handle. This code shows the syntax of this argument.

```
[outputPacket,processedLength] = linkTypeDecoderFunction(payload);
protocolDecoder = @linkTypeDeocderFunction
```

The `linkTypeDecoderFunction` is the function that decodes the payload. The `outputPacket` output contains the decoded packet as a structure. The `processedLength` output is the number of decoded bytes. A negative value of the processed length indicates failed packet decoding. If packet decoding fails, the output decoded packet structure is empty.

Data Types: `function_handle`

`protocolFields` — Protocol fields and data types
two-column cell array

Protocol fields and data types, specified as a two-column cell array that indicates the protocol fields and their respective data types. Specify these fields and data types as character vectors or string scalars. The first column of the cell array specifies the field name. The second column of the cell array specifies the data type of the corresponding field name. This value specifies the fields (of the protocol decoder output structure) on which the `read` or `readAll` object functions can specify the `packetFilter` input. When you specify the `packetFilter` input of `read` or `readAll` object functions, this object function uses the `protocolFields` value for:

- Tab completion of `packetFilter` string
- Validating `packetFilter` string

For more information about how to use this value to filter packets, see the `packetFilter` input of the `read` or `readAll` object functions.

Data Types: `cell`

Version History

Introduced in R2021b

References

[1] Group, The Tcpdump. “Tcpdump/Libpcap Public Repository.” Accessed May 20, 2020. https://www.tcpdump.org.

See Also

Objects
`pcapReader`

Functions
`addUpperLayerDecoder` | `read` | `readAll` | `reset`

addUpperLayerDecoder

Add custom upper-layer protocol decoder to PCAP file reader

Syntax

```
addUpperLayerDecoder(pcap,protocolName,fieldName,fieldValue,
upperLayerProtocolName,upperLayerDecoder)
addUpperLayerDecoder(pcap,protocolName,fieldName,fieldValue,
upperLayerProtocolName,upperLayerDecoder,protocolFields)
```

Description

You can use the `addUpperLayerDecoder` object function to plug in a custom upper-layer protocol decoder to the `pcapReader` object for decoding the payload of an existing protocol decoder. For example, you can add an upper-layer protocol decoder, such as open radio access network (ORAN), to decode the payload in the output of an existing protocol decoder, enhanced common public radio interface (eCPRI).

`addUpperLayerDecoder(pcap,protocolName,fieldName,fieldValue, upperLayerProtocolName,upperLayerDecoder)` adds a custom upper-layer protocol decoder function handle, `upperLayerDecoder`, to the packet capture (PCAP) file reader, `pcap`, for decoding the payload in the output of an existing protocol decoder, `protocolName`. The `fieldName` input specifies the field in the output structure of the existing protocol decoder, and the `fieldValue` input specifies the value of this field name. The `upperLayerProtocolName` input specifies a name for the upper-layer protocol decoder.

`addUpperLayerDecoder(pcap,protocolName,fieldName,fieldValue, upperLayerProtocolName,upperLayerDecoder,protocolFields)` additionally specifies protocol fields to filter packets based on the specified fields of the protocol decoder output.

Input Arguments

pcap — PCAP file reader
`pcapReader` object

PCAP file reader, specified as a `pcapReader` object.

`protocolName` — Name of existing protocol decoder
character vector | string scalar

Name of the existing protocol decoder, specified as a character vector or a string scalar. The existing protocol decoder can be an Ethernet decoder, an eCPRI decoder, or any other newly added protocol decoder.

Data Types: `char` | `string`

`fieldName` — Field name in output structure of existing protocol decoder
character vector | string scalar

Field name in the output structure of the existing protocol decoder, specified as a character vector or a string scalar. For more information about the field names of Ethernet and eCPRI protocol decoders, see the `protocolPackets` output of the `read` or `readAll` object functions.

Data Types: `char` | `string`

`fieldValue` — Value of field name
nonnegative integer | character vector | string scalar

Value of the field name, specified as a nonnegative integer, character vector, or string scalar. If the `fieldName` input has this value, then this object function passes the payload to the upper-layer protocol decoder handle.

Data Types: `double` | `char` | `string`

`upperLayerProtocolName` — Name of upper-layer protocol decoder
character vector | string scalar

Name of the upper-layer protocol decoder, specified as a character vector or a string scalar. To store the decoded packet of the upper-layer protocol decoder, the `read` or `readAll` object functions use this value to create a new field in the `Packet` field of the output decoded protocol packet structure.

Data Types: `char` | `string`

`upperLayerDecoder` — Upper-layer protocol decoder
function handle

Upper-layer protocol decoder that decodes the payload of the existing protocol decoder and returns the decoded packet with the processed length, specified as a function handle. This code shows the syntax of this argument.

```
[outputPacket,processedLength] = upperLayerDecoderFunction(existingProtocolPayload);
upperLayerDecoder = @upperLayerDeocderFunction
```

The `upperLayerDecoderFunction` is the function that decodes the payload of the existing protocol decoder. The `outputPacket` output contains the decoded packet as a structure. The `processedLength` output is the number of decoded bytes. A negative value of the processed length indicates failed packet decoding. If packet decoding fails, the output decoded packet structure is empty.

Data Types: `function_handle`

`protocolFields` — Protocol fields and data types
two-column cell array

Protocol fields and data types, specified as a two-column cell array that indicates the protocol fields and their respective data types. Specify these fields and data types as character vectors or string scalars. The first column of the cell array specifies the field name. The second column of the cell array specifies the data type of the corresponding field name. The valid values for the data type are `numeric`, `string`, and `hexadecimal`. This value specifies the fields (of the protocol decoder output structure) on which the `read` or `readAll` object functions can specify the `packetFilter` input. When you specify the `packetFilter` input of `read` or `readAll` object functions, this object function uses the `protocolFields` value for:

- Tab completion of `packetFilter` string
- Validating `packetFilter` string

For more information about how to use this value to filter packets, see the `packetFilter` input of the `read` or `readAll` object functions.

Data Types: `cell`

Version History

Introduced in R2021b

References

[1] Group, The Tcpdump. “Tcpdump/Libpcap Public Repository.” Accessed May 20, 2020. https://www.tcpdump.org.

See Also

Objects
`pcapReader`

Functions
`addLinkTypeDecoder` | `read` | `readAll` | `reset`

displayChannel

Visualize and explore CDL channel model characteristics

Syntax

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

Description

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

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

Examples

Visualize CDL Channel Model Characteristics

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

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

```
cdl = nrCDLChannel;
cdl.DelayProfile = 'CDL-D';
```

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

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

Configure the vertical and horizontal element spacing and the vertical and horizontal panel spacing, in wavelength, as a vector of the form $[\lambda_v\ \lambda_h\ dg_v\ dg_h]$. Because panel spacing is measured from the center of the panels, to avoid panel overlapping, set dg_h to a value greater than 1 wavelength. To ensure uniform antenna element spacing across vertically and horizontally separated panels, configure panel spacings as $dg_v = \lambda_v \times M$ and $dg_h = \lambda_h \times N$, respectively.

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

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

```
cdl.TransmitArrayOrientation = [0 15 0]';
```

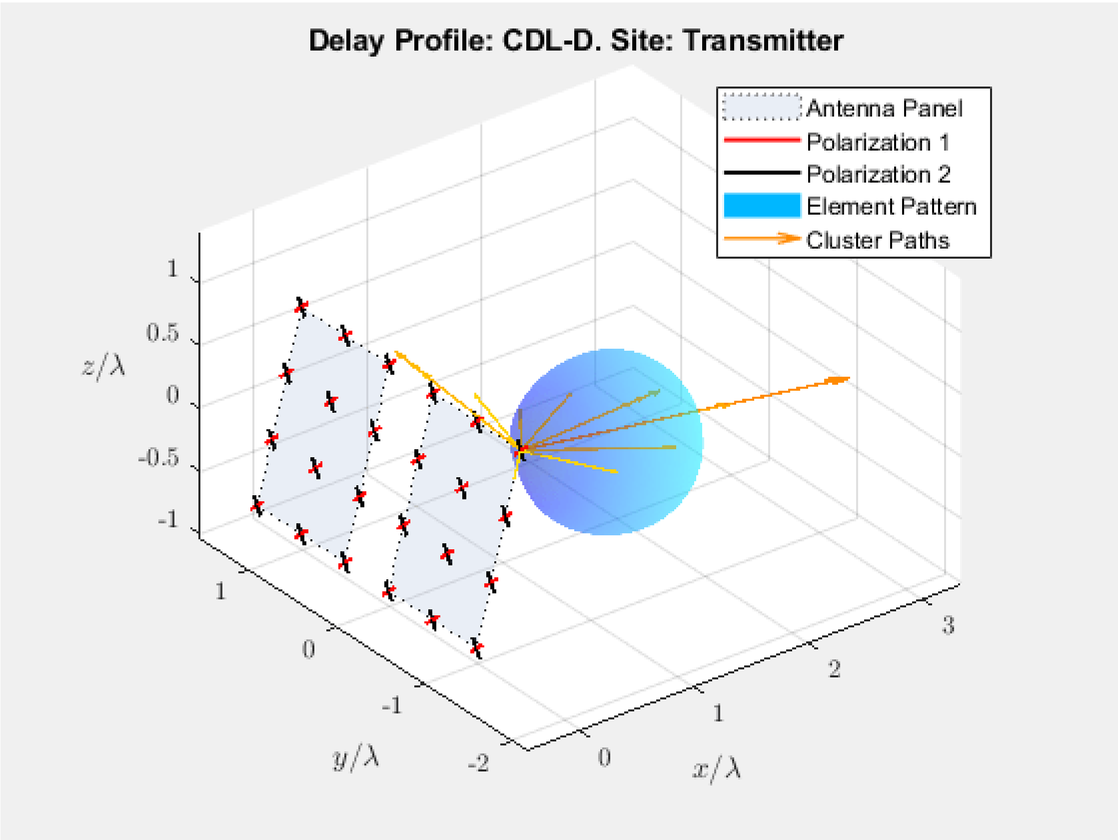
For an overview of all other transmit antenna array properties, see the "TransmitAntennaArray" on page 2-0 property of the `nrCDLChannel` System object.

Display the channel characteristics at the transmitter end.

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

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

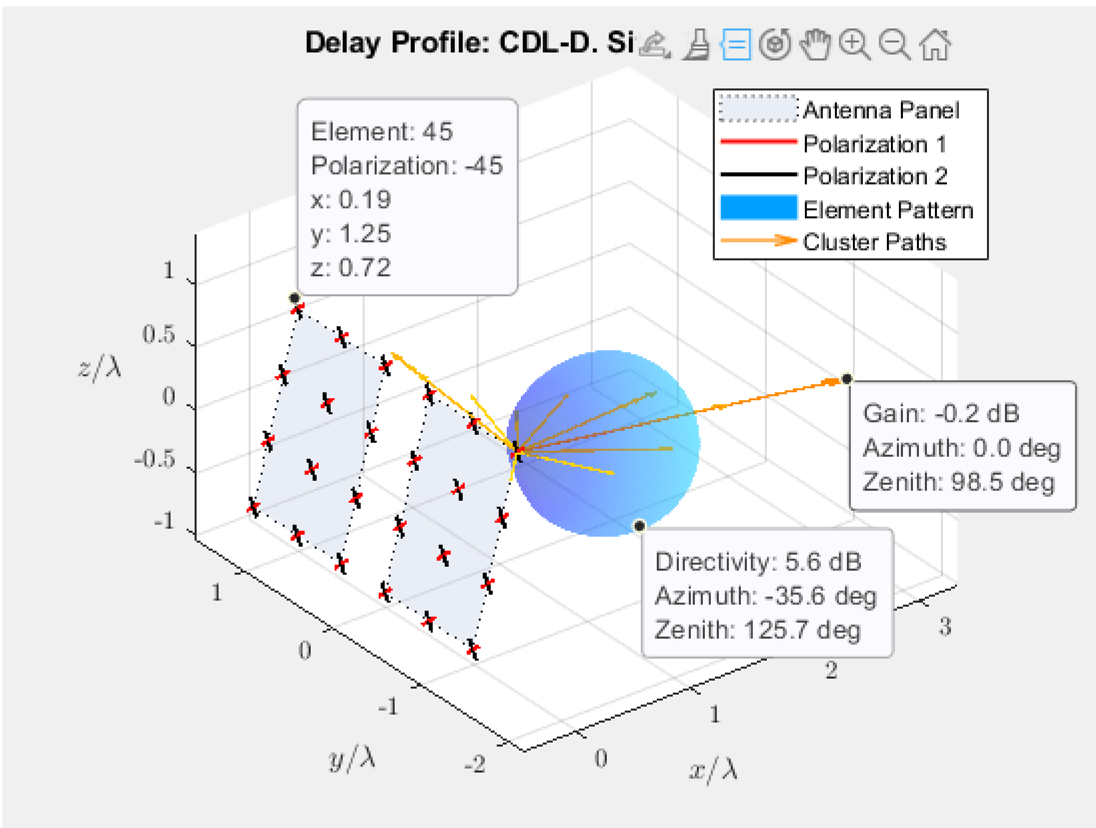
```
datacursormode on;
```



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

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

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

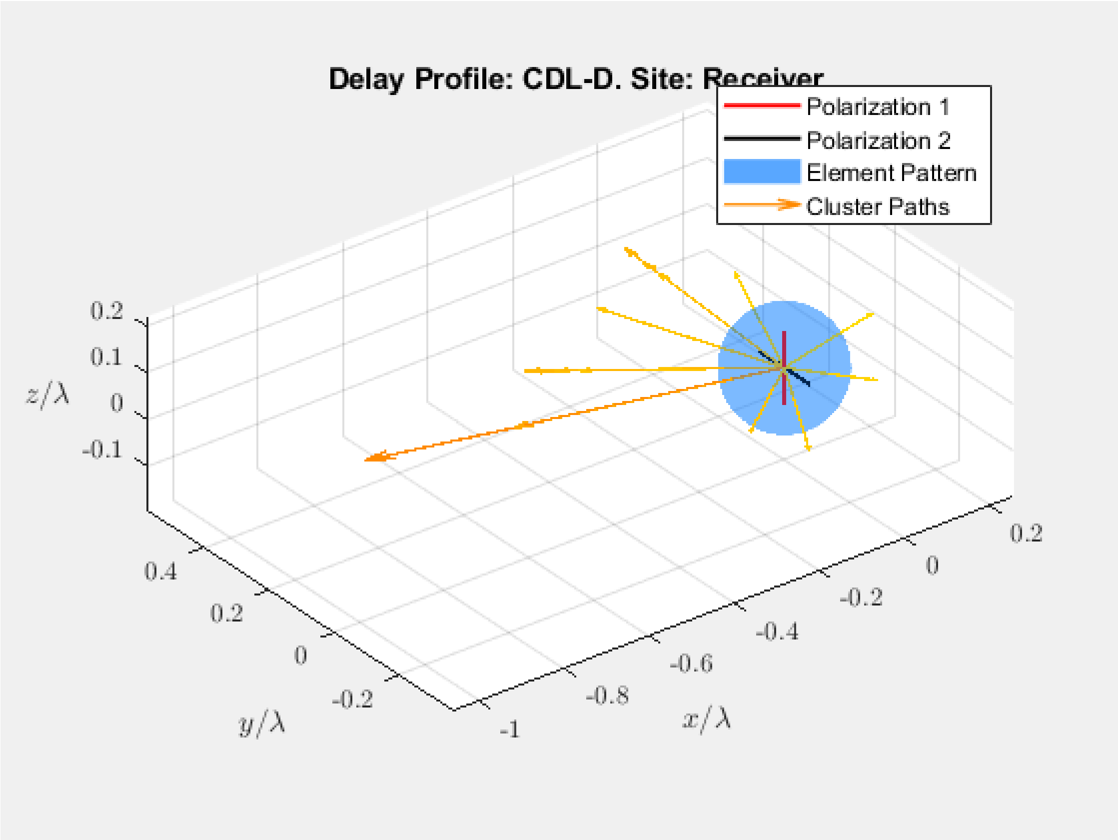


Visualize and explore channel characteristics at the receiver end. To customize the receive antenna array, use the “ReceiveAntennaArray” on page 2-0 property of the `nrCDLChannel` System object. Then, display the channel characteristics at the receiver end by calling the `displayChannel` function with the `'LinkEnd','Rx'` name-value pair argument.

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

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

```
datacursormode on;
```



Input Arguments

cdl — CDL channel model
`nrCDLChannel` System object™

CDL channel model, specified as an `nrCDLChannel` System object. The `nrCDLChannel` object implements the multi-input multi-output (MIMO) link-level fading channel specified in TR 38.901 Section 7.7.1.

Name-Value Pair Arguments

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

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

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

LinkEnd — Link-level channel end
`'Both'` (default) | `'Tx'` | `'Rx'`

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

- 'Both' — The function displays channel characteristics at both ends: the transmitter and receiver ends.
- 'Tx' — The function displays channel characteristics only at the transmitter end.
- 'Rx' — The function displays channel characteristics only at the receiver end.

Data Types: char | string

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

Polarization angle of antenna elements, specified as the comma-separated pair consisting of 'Polarization' and 'on' or 'off'. When set to 'on', the function displays the polarization angle of the antenna elements.

Data Types: char | string

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

Directivity radiation pattern of antenna, specified as the comma-separated pair consisting of 'ElementPattern' and 'on' or 'off'. When set to 'on', the function displays the directivity radiation pattern of the antenna elements.

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

Data Types: char | string

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

Direction and average gain of cluster paths, specified as the comma-separated pair consisting of 'ClusterPaths' and 'on' or 'off'. When set to 'on', the function displays the direction and average gain of the cluster paths.

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

Data Types: char | string

Output Arguments

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

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

Version History

Introduced in R2020b

References

[1] 3GPP TR 38.901. "Study on channel model for frequencies from 0.5 to 100 GHz." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Functions
`info`

Objects
`nrCDLChannel`

generate

Generate next FTP, On-Off, VoIP, or video conference application traffic packet

Syntax

```
[dt,packetSize] = generate(cfg)
[dt,packetSize] = generate(cfg,elapsedTime)
[___,packet] = generate(___)
```

Description

`[dt,packetSize] = generate(cfg)` generates the next FTP, On-Off, VoIP, or video application traffic pattern based on the specified configuration object, `cfg`. The object function returns the time remaining to generate the next packet, `dt`, and the size of the current packet, `packetSize`.

`[dt,packetSize] = generate(cfg,elapsedTime)` specifies the time elapsed, `elapsedTime`, since the previous call of this object function.

`[___,packet] = generate(___)` returns the FTP, On-Off, VoIP, or video application traffic packet. Specify an argument combination from any of the previous syntaxes.

Examples

Generate VoIP Application Traffic Pattern Without Jitter

Create a VoIP application traffic pattern object, disabling modeling of jitter.

```
cfgVoIP = networkTrafficVoIP('HasJitter',false);
```

Generate a VoIP application traffic pattern.

```
[dt,packetSize] = generate(cfgVoIP);
```

Input Arguments

`cfg` — Configuration object to generate FTP, On-Off, VoIP, or video application traffic pattern

`networkTrafficFTP` object | `networkTrafficOnOff` object | `networkTrafficVoIP` object | `networkTrafficVideoConference` object

Configuration object to generate FTP, On-Off, VoIP, or video application traffic pattern, specified as a `networkTrafficFTP`, `networkTrafficOnOff`, `networkTrafficVoIP`, or `networkTrafficVideoConference` object.

`elapsedTime` — Time elapsed since previous call of this object function

nonnegative scalar

Time elapsed since the previous call of this object function, specified as a nonnegative scalar. This argument must be expressed in milliseconds.

Data Types: `double`

Output Arguments

`dt` — Time remaining to generate next packet
nonnegative scalar

Time remaining to generate the next packet, returned as a nonnegative scalar. This argument must be expressed in milliseconds.

Data Types: `double`

`packetSize` — Size of current packet
positive scalar

Size of the current packet, returned as a positive scalar. The units of this argument are in bytes.

Data Types: `double`

`packet` — Application data packet
column vector of integers in the range [0, 255]

Application data packet, returned as a column vector of integers in the range [0, 255]. This value contains the application data specified by the `ApplicationData` property of the input `cfg`. If the `ApplicationData` property is not specified, `packet` is a column vector of 1s.

Dependencies

To enable this output argument, set the `GeneratePacket` property of the input `cfg` to `1` (`true`).

Data Types: `double`

Version History

Introduced in R2020b

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`networkTrafficFTP` | `networkTrafficOnOff` | `networkTrafficVoIP` | `networkTrafficVideoConference`

getPathFilters

Get path filter impulse response for link-level MIMO channel

Syntax

```
pathFilters = getPathFilters(channel)
```

Description

`pathFilters = getPathFilters(channel)` returns path filter impulse responses for the specified link-level multi-input multi-output (MIMO) channel, `channel`. Specify the channel as an `nrCDLChannel`, `nrTDLChannel`, or `nrHSTChannel` System object. Use `pathFilters` together with the `pathGains` output argument returned by the channel object to reconstruct a perfect channel estimate.

Note

- When `channel` is an `nrHSTChannel` System object with HST-SFN channel profile, the path filters are time-varying and depend on the state of the channel. In this case, the returned path filters always relate to the last call of the channel object. For all other channels, the path filters do not change with the channel object call.

Examples

Reconstruct Channel Impulse Response Using CDL Channel Path Filters

Reconstruct the channel impulse response and perform timing offset estimation using path filters of a Clustered Delay Line (CDL) channel model with delay profile CDL-D from TR 38.901 Section 7.7.1.

Define the channel configuration structure using an `nrCDLChannel` System object. Use delay profile CDL-D, a delay spread of 10 ns, and UE velocity of 15 km/h:

```
v = 15.0;                    % UE velocity in km/h
fc = 4e9;                    % carrier frequency in Hz
c = physconst('lightspeed'); % speed of light in m/s
fd = (v*1000/3600)/c*fc;     % UE max Doppler frequency in Hz

cdl = nrCDLChannel;
cdl.DelayProfile = 'CDL-D';
cdl.DelaySpread = 10e-9;
cdl.CarrierFrequency = fc;
cdl.MaximumDopplerShift = fd;
```

Configure the transmit array as [M N P Mg Ng] = [2 2 2 1 1], representing 1 panel (Mg=1, Ng=1) with a 2-by-2 antenna array (M=2, N=2) and P=2 polarization angles. Configure the receive antenna array as [M N P Mg Ng] = [1 1 2 1 1], representing a single pair of cross-polarized co-located antennas.

```
cdl.TransmitAntennaArray.Size = [2 2 2 1 1];
cdl.ReceiveAntennaArray.Size = [1 1 2 1 1];
```

Create a random waveform of 1 subframe duration with 8 antennas.

```
SR = 15.36e6;
T = SR * 1e-3;
cdl.SampleRate = SR;
cdlinfo = info(cdl);
Nt = cdlinfo.NumTransmitAntennas;

txWaveform = complex(randn(T,Nt),randn(T,Nt));
```

Transmit the input waveform through the channel.

```
[rxWaveform,pathGains] = cdl(txWaveform);
```

Obtain the path filters used in channel filtering.

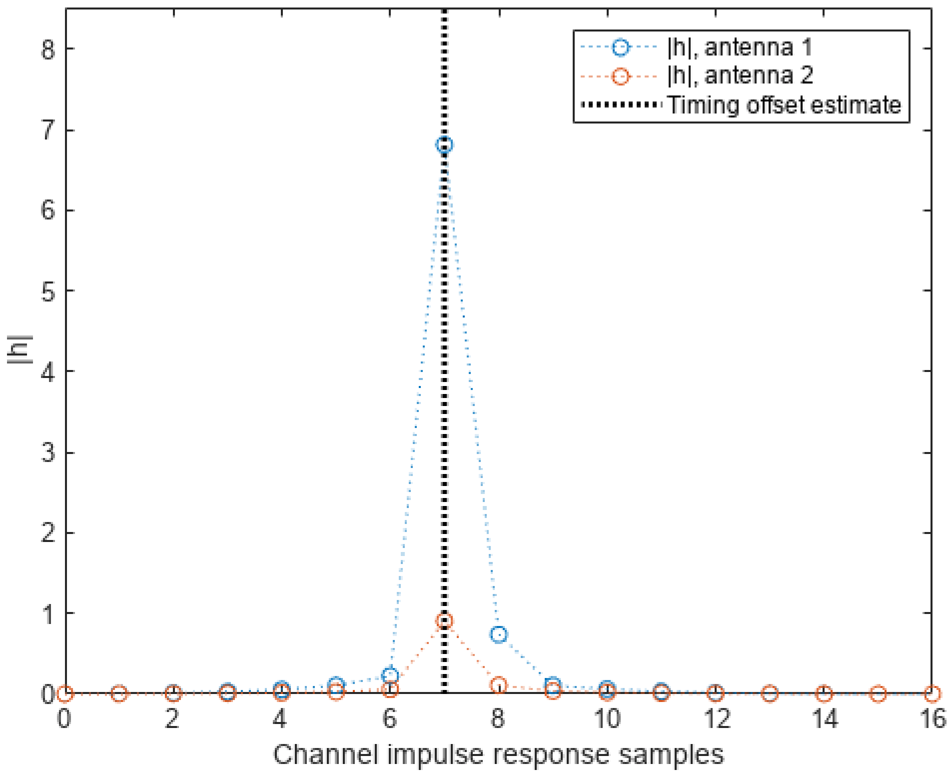
```
pathFilters = getPathFilters(cdl);
```

Perform timing offset estimation using `nrPerfectTimingEstimate`.

```
[offset,mag] = nrPerfectTimingEstimate(pathGains,pathFilters);
```

Plot the magnitude of the channel impulse response.

```
[Nh,Nr] = size(mag);
plot(0:(Nh-1),mag,'o:');
hold on;
plot([offset offset],[0 max(mag(:))*1.25],'k:','LineWidth',2);
axis([0 Nh-1 0 max(mag(:))*1.25]);
legends = "|h|, antenna " + num2cell(1:Nr);
legend([legends "Timing offset estimate"]);
ylabel('|h|');
xlabel('Channel impulse response samples');
```



Estimate HST-SFN Channel Delay

Create an HST-SFN multi-tap channel model with one receive antenna.

```
hst = nrHSTChannel( ...
    ChannelProfile='HST-SFN', ...
    NumReceiveAntennas=1);
```

Set the distance between the gNodeBs to 700 m. Set the minimum distance between the railway track and the gNodeBs to 150 m.

```
hst.Ds = 700;
hst.Dmin = 150;
```

Set the train velocity to 500 km/h. Set the maximum Doppler shift to 870 Hz.

```
hst.Velocity = 500;
hst.MaximumDopplerShift = 870;
```

Disable channel filtering.

```
hst.ChannelFiltering = false;
```

Set the sample rate and the number of channel samples to calculate 1 ms of path gains samples.

```
hst.SampleRate = 30.72e6;
hst.NumTimeSamples = hst.SampleRate*1e-3;
```

Set the initial time of the channel to configure the starting position of the train. The train position relative to a remote gNodeB determines the delay of each gNodeB signal.

```
hst.InitialTime = (hst.Ds/3)/(hst.Velocity/3.6);
```

Retrieve the path gains from the channel.

```
pathGains = hst();
```

Obtain the channel path filter responses relative to the previous channel call. The delay of each gNodeB signal changes over time.

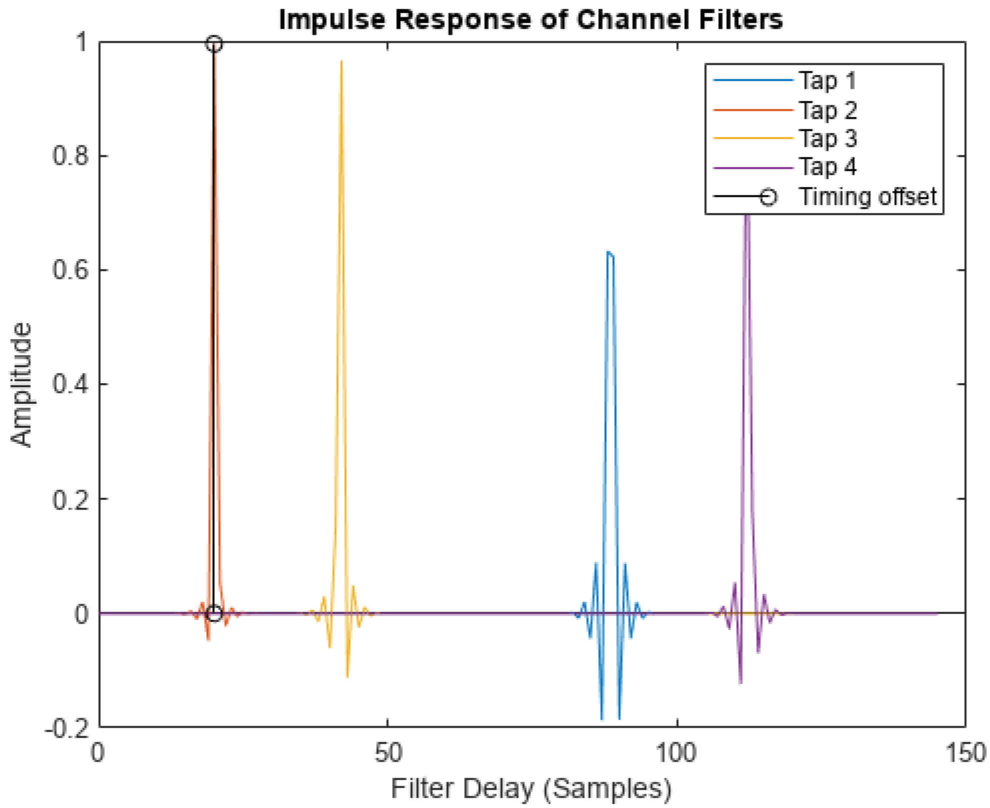
```
pathFilters = getPathFilters(hst);
```

Estimate the channel delay by obtaining the timing offset.

```
offset = nrPerfectTimingEstimate(pathGains,pathFilters);
```

Display the path filters and the estimated channel delay.

```
plot(0:size(pathFilters,1)-1,pathFilters);
hold on
stem(repmat(offset,1,hst.NumTaps),pathFilters(1+offset,:),'k')
legend(["Tap "+ (1:hst.NumTaps) "Timing offset"])
xlabel('Filter Delay (Samples)')
ylabel('Amplitude')
title('Impulse Response of Channel Filters')
```



Input Arguments

`channel` — MIMO channel model
`nrCDLChannel` | `nrTDLChannel` | `nrHSTChannel`

MIMO channel model, specified as an `nrCDLChannel`, `nrTDLChannel`, or `nrHSTChannel` System object.

Output Arguments

`pathFilters` — Path filter impulse response
N_h-by-N_p real matrix

Path filter impulse response, returned as an N_h-by-N_p real matrix, where:

- N_h is the number of impulse response samples.
- N_p is the number of paths.

Each column of the matrix contains the filter impulse response for each path of the delay profile.

Data Types: `double`

Version History

Introduced in R2018b

Path filter impulse response for HST channel models

The function accepts `nrHSTChannel` System object as input argument.

References

[1] 3GPP TR 38.901. "Study on channel model for frequencies from 0.5 to 100 GHz." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.101-4. "NR; User Equipment (UE) radio transmission and reception; Part 4: Performance requirements." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
`nrCDLChannel` | `nrTDLChannel` | `nrHSTChannel`

getTransportBlock

Get transport block from UL-SCH or DL-SCH encoder

Syntax

```
trblk = getTransportBlock(encUL)
trblk = getTransportBlock(encDL,trblkID)
trblk = getTransportBlock( ___ ,harqID)
```

Description

`trblk = getTransportBlock(encUL)` returns the transport block from the specified uplink shared channel (UL-SCH) encoder System object. The function assumes that a transport block was previously loaded into the specified UL-SCH encoder by using the `setTransportBlock` function.

`trblk = getTransportBlock(encDL,trblkID)` returns the transport block from the specified downlink shared channel (DL-SCH) encoder System object `encDL` for the specified transport block number `trblkID`. The function assumes that a transport block was previously loaded into the specified DL-SCH encoder by using the `setTransportBlock` function.

`trblk = getTransportBlock( ___ ,harqID)` returns the transport block loaded for the specified hybrid automatic repeat-request (HARQ) process number `harqID`. Specify `harqID` in addition to the input arguments in any of the previous syntaxes.

Examples

Retrieve Transport Block from UL-SCH Encoder with Multiple HARQ Processes

Generate a random sequence of binary values corresponding to one transport block of length 5120.

```
trBlkLen1 = 5120;
trBlk1 = randi([0 1],trBlkLen1,1,'int8');
```

Create and configure an UL-SCH encoder System object with multiple HARQ processes and the specified target code rate.

```
targetCodeRate = 567/1024;
encUL = nrULSCH('MultipleHARQProcesses',true);
encUL.TargetCodeRate = targetCodeRate;
```

Load the transport block into the UL-SCH encoder for HARQ process number 1.

```
setTransportBlock(encUL,trBlk1,1);
```

Call the encoder with QPSK modulation scheme, 1 transmission layer, an output length of 10,240 bits, redundancy version 0, and HARQ process number 1. The encoder applies the UL-SCH processing chain to the transport block loaded into the object using HARQ process number 1.

```
encUL('QPSK',1,10240,0,1);
```

Retrieve the transport block from the encoder for HARQ process number 1. Verify that the retrieved block is identical to the block originally loaded into the encoder for this HARQ process.

```
tmp = getTransportBlock(encUL,1);
isequal(tmp,trBlk1)
```

```
ans = logical
   1
```

Repeat the encoding operation for a new transport block of length 4400 and HARQ process number 2.

```
trBlkLen2 = 4400;
trBlk2 = randi([0 1],trBlkLen2,1,'int8');
setTransportBlock(encUL,trBlk2,2);
encUL('QPSK',1,8800,0,2);
```

Retrieve the first transport block again. Verify that the first transport block is still unchanged.

```
tmp = getTransportBlock(encUL,1);
isequal(tmp,trBlk1)
```

```
ans = logical
   1
```

Retrieve Transport Block from DL-SCH Encoder with Multiple HARQ Processes

Generate a random sequence of binary values corresponding to one transport block of length 5120.

```
trBlkLen = 5120;
trBlk = randi([0 1],trBlkLen,1,'int8');
```

Create and configure a DL-SCH encoder System object with multiple HARQ processes and the specified target code rate.

```
targetCodeRate = 567/1024;
encDL = nrDLSCH('MultipleHARQProcesses',true);
encDL.TargetCodeRate = targetCodeRate;
```

Load transport block `trBlk` for transport block number 0 into the DL-SCH encoder, specifying HARQ process number 2.

```
harqID = 2;
trBlkID = 0;
setTransportBlock(encDL,trBlk,trBlkID,harqID);
```

Call the encoder with QPSK modulation scheme, 3 transmission layers, an output length of 10,002 bits, and redundancy version 3. The encoder applies the DL-SCH processing chain to the transport block loaded into the object for HARQ process number 2.

```
mod = 'QPSK';
nLayers = 3;
outlen = 10002;
```

```
rv = 3;
codedTrBlock = encDL(mod,nLayers,outlen,rv,harqID);
```

Retrieve the transport block for transport block number 0 from the encoder, specifying HARQ process number 2. Verify that the retrieved block is identical to the block originally loaded into the encoder for this HARQ process.

```
tmp = getTransportBlock(encDL,trBlkID,harqID);
isequal(tmp,trBlk)
```

```
ans = logical
   1
```

Input Arguments

encUL — UL-SCH encoder
`nrULSCH` System object

UL-SCH encoder, specified as an `nrULSCH` System object. The objects implements the UL-SCH processing chain specified in TS 38.212 Section 6.2.

encDL — DL-SCH encoder
`nrDLSCH` System object

DL-SCH encoder, specified as an `nrDLSCH` System object. The object implements the DL-SCH processing chain specified in TS 38.212 Section 7.2.

`trblkID` — Transport block number in DL-SCH processing
`0` (default) | `1`

Transport block number in DL-SCH processing, specified as `0` or `1`.

Data Types: `double`

`harqID` — HARQ process number
integer from 0 to 15

HARQ process number, specified as an integer from 0 to 15.

Data Types: `double`

Output Arguments

`trblk` — Transport block
binary column vector

Transport block, returned as a binary column vector.

Data Types: `int8`

Version History

Introduced in R2019a

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`setTransportBlock`

Objects
`nrULSCH` | `nrDLSCH`

info

Characteristic information of link-level MIMO channel

Syntax

`channelInfo = info(channel)`

Description

`channelInfo = info(channel)` returns characteristic information of the link-level multi-input multi-output (MIMO) channel model, `channel`.

Examples

Get Characteristic Information About TDL Fading Channel

Create an `nrTDLChannel` System object.

```
tdl = nrTDLChannel;
```

To get characteristic information about the channel, call the info function on the object.

```
channelInfo = info(tdl)
```

```
channelInfo = struct with fields:
          ChannelFilterDelay: 7
         MaximumChannelDelay: 16
                  PathDelays: [0 1.1457e-08 1.2075e-08 1.7604e-08 ... ]
            AveragePathGains: [-13.4000 0 -2.2000 -4 -6 -8.2000 -9.9000 ... ]
             KFactorFirstTap: -Inf
         NumTransmitAntennas: 1
          NumReceiveAntennas: 2
    SpatialCorrelationMatrix: [2x2 double]
```

Input Arguments

`channel` — MIMO channel model
`nrCDLChannel` | `nrTDLChannel` | `nrHSTChannel`

MIMO channel model, specified as an `nrCDLChannel`, `nrTDLChannel`, or `nrHSTChannel` System object.

Output Arguments

`channelInfo` — Characteristic information of channel model
structure

Characteristic information of the channel model, `channel`, returned as a structure. The fields of the structure depend on the input `channel`.

- If `channel` is an `nrCDLChannel` System object, the `channelInfo` structure has these fields.

Parameter Field	Value	Description
PathDelays	Numeric row vector	Delays of discrete channel paths for each cluster in seconds, returned as a numeric row vector. These values include the effects of `DelaySpread` scaling and `KFactorScaling` (when enabled).
ClusterTypes	Cell array of character vectors	Type of each cluster in the delay profile, returned as a cell array of character vectors. Cluster types can be `'LOS'`, `'SubclusteredNLOS'`, or `'NLOS'`.
AveragePathGains	Numeric row vector	Average path gains of the discrete path or clusters in dB, returned as a numeric row vector. These values include the effect of `KFactorScaling` scaling (when enabled).
AnglesAoD	Numeric row vector	Azimuth of departure angles of the clusters in degrees, returned as a numeric row vector.
AnglesAoA	Numeric row vector	Azimuth of arrival angles of the clusters in degrees, returned as a numeric row vector.
AnglesZoD	Numeric row vector	Zenith of departure angles of the clusters in degrees, returned as a numeric row vector.
AnglesZoA	Numeric row vector	Zenith of arrival angles of the clusters in degrees, returned as a numeric row vector.
KFactorFirstCluster	Numeric scalar	K-factor of first cluster of delay profile in dB, returned as a numeric scalar. If the first cluster of the delay profile follows a Laplacian instead of a Rician distribution, `KFactorFirstCluster` is `-Inf`.
NumTransmitAntennas	Numeric scalar	Number of transmit antenna elements, returned as a numeric scalar.

Parameter Field	Value	Description
NumInputSignals	Numeric scalar	Number of input signals, returned as a numeric scalar. This value determines the number of columns in the input signal sent through the channel (see `signalIn`). The value of `NumInputSignals` and `NumTransmitAntennas` structure fields are equal, unless you specify the transmit antenna as a `phased.ReplicatedSubarray` or `phased.PartitionedArray` phased array object. In case of these two phased antenna arrays, `NumInputSignals` is equal to the number of antenna subarrays, so `NumInputSignals` ≤ `NumTransmitAntennas`.
NumReceiveAntennas	Numeric scalar	Number of receive antenna elements, returned as a numeric scalar.
NumOuputSignals	Numeric scalar	Number of output signals, returned as a numeric scalar. This value determines the number of columns in the output signal received from the channel (see `signalOut`). The value of `NumOutputSignals` and `NumReceiveAntennas` structure fields are equal, unless you specify the receive antenna as a `phased.ReplicatedSubarray` or `phased.PartitionedArray` phased array object. In case of these two phased antenna arrays, `NumOutputSignals` is equal to the number of subarrays, so `NumOutputSignals` ≤ `NumReceiveAntennas`.
ChannelFilterDelay	Numeric scalar	Channel filter delay in samples, returned as a numeric scalar.
MaximumChannelDelay	Numeric scalar	Maximum channel delay in samples, returned as a numeric scalar. This delay consists of the maximum path delay and the channel filter delay.

Note

- The step of splitting the strongest clusters into subclusters, described in TR 38.901 Section 7.5, requires sorting of the clusters by their average power. If the `NumStrongestClusters` property is nonzero (applies only when `DelayProfile` is set to `'Custom'`), the fields of the information structure are sorted by average power. That is, the `AveragePathGains`, `ClusterTypes`, `PathDelays`, `AnglesAoD`, `AnglesAoA`, `AnglesZoD`, and `AnglesZoA` fields are presented in descending order of the average gain.
- If the `HasLOSCluster` property is set to `true`, the NLOS (Laplacian) part of that cluster and the LOS cluster are not necessarily next to each other. However, the `KFactorFirstCluster` field still indicates the appropriate K-factor.

- If `channel` is an `nrTDLChannel` System object, the `channelInfo` structure has these fields.

Parameter Field	Value	Description
ChannelFilterDelay	Numeric scalar	Channel filter delay in samples, returned as a numeric scalar.
MaximumChannelDelay	Numeric scalar	Maximum channel delay in samples, returned as a numeric scalar. This delay consists of the maximum path delay and the channel filter delay.
AveragePathGains	Numeric row vector	Average path gains of the discrete paths in dB, returned as a numeric row vector. These values include the effect of `KFactorScaling` (when enabled).
PathDelays	Numeric row vector	Delays of discrete channel paths in seconds, returned as a numeric row vector. These values include the effects of `DelaySpread` scaling and `KFactorScaling` (when enabled).
KFactorFirstTap	Numeric scalar	K-factor of first tap of delay profile in dB, returned as a numeric scalar. If the first tap of the delay profile follows a Rayleigh instead of a Rician distribution, `KFactorFirstTap` is `-Inf`.
NumTransmitAntennas	Numeric scalar	Number of transmit antennas, returned as a numeric scalar.
NumReceiveAntennas	Numeric scalar	Number of receive antennas, returned as a numeric scalar.
SpacialCorrelationMatrix	Numeric matrix	Combined correlation matrix or 3-D array, returned as a numeric matrix.

- If `channel` is an `nrHSTChannel` System object, the `channelInfo` structure has these fields.

Parameter Field	Value	Description
PathDelays	N_S-by-N_P numeric matrix	Absolute propagation delays of the discrete channel paths at the input signal sample times in seconds, returned as an N_S-by-N_P numeric matrix, where: • N_S is the number of input samples. • N_P is the number of paths. If you call the `info` function before the first call of the `nrHSTChannel` System object, this field is returned as `[]`.
DopplerShifts	N_S-by-N_P numeric matrix	Doppler shifts of the discrete paths at the input signal sample times in Hz, returned as an N_S-by-N_P matrix. If you call the `info` function before the first call of the `nrHSTChannel` System object, this field is returned as `[]`.
PowerLevels	N_S-by-N_P numeric matrix	Path gains of the discrete paths at the input signal sample times in dB, returned as an N_S-by-N_P numeric matrix. If you call the `info` function before the first call of the `nrHSTChannel` System object, this field is returned as `[]`.
CarrierFrequency	Numeric scalar	Carrier frequency (f) in Hz, returned as a numeric scalar equal to $fd \times c/v$, where: • fd is the maximum Doppler shift. • v is the train velocity. • c is the speed of light. For static propagation conditions, when $fd = 0$ and $v = 0$, the carrier frequency is returned as `NaN`.

Parameter Field	Value	Description
NumTransmitAntennas	Positive integer	Number of transmit antennas, returned as a positive integer. • When the ChannelFiltering property is set to true, this field is equal to the second dimension of the input signal. • When the ChannelFiltering property is set to false, this field is equal to the NumTransmitAntennas property value.
NumReceiveAntennas	Positive integer	Number of receive antennas, returned as a positive integer.
ChannelFilterDelay	Numeric scalar	Channel filter delay in samples, returned as numeric scalar.
MaximumChannelDelay	Numeric scalar	Maximum channel delay in samples, returned as a numeric scalar. This delay consists of the channel filter delay and the maximum propagation delay relative to the minimum propagation delay, Dmin/*c*: MaximumChannelDelay = ChannelFilterDelay + (*MaxPropagationDelay* – *MinPropagationDelay*).

Data Types: struct

Version History

Introduced in R2018b

channelInfo output argument returns maximum channel delay

The channelInfo output argument returns the maximum channel delay. This delay consists of the maximum path delay and the channel filter delay.

Characteristic information for HST channel models

The function accepts nrHSTChannel System object as input argument.

References

[1] 3GPP TR 38.901. "Study on channel model for frequencies from 0.5 to 100 GHz." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.101-4. “NR; User Equipment (UE) radio transmission and reception; Part 4: Performance requirements.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
`nrCDLChannel` | `nrTDLChannel` | `nrHSTChannel`

nrBCH

Broadcast channel (BCH) encoding

Syntax

```
cdblk = nrBCH(trblk,sfn,hrf,lssb,idxoffset,ncellid)
```

Description

`cdblk = nrBCH(trblk,sfn,hrf,lssb,idxoffset,ncellid)` encodes BCH transport block `trblk`, as defined in TS 38.212, Section 7.1 [1], and returns the encoded BCH transport block. The function takes these additional input arguments:

- `sfn`, the system frame number
- `hrf`, the half frame bit in synchronization signal / physical broadcast channel (SS/PBCH) block transmissions
- `lssb`, the number of candidate SS/PBCH blocks in a half frame
- `idxoffset`, the subcarrier offset or the SS block index, depending on the input value of `lssb`
- `ncellid`, the physical layer cell identity number

Examples

Encode BCH Transport Block

Generate a random sequence of binary values corresponding to a BCH transport block of 24 bits.

```
trblk = randi([0 1],24,1,'int8');
```

Specify the physical layer cell identity number as 321, the system frame number as 10, and the second half frame.

```
nid = 321;
sfn = 10;
hrf = 1;
```

Specify the number of candidate SS/PBCH blocks as 8. When you specify the number of candidate SS/PBCH blocks as `4` or `8`, you can specify the subcarrier offset `kssb` as an input argument to the BCH encoder.

```
lssb = 8;
kssb = 18;
```

Encode the BCH transport block using the specified arguments.

```
cdblk = nrBCH(trblk,sfn,hrf,lssb,kssb,nid);
```

When you specify the number of candidate SS/PBCH blocks as 64, you can specify the SS block index `ssbIdx` as an input argument instead of the subcarrier offset `kssb`.

```
lssb = 64;
ssbIdx = 13;
```

Encode the BCH transport block with the updated input arguments.

```
cdblk2 = nrBCH(trblk,sfn,hrf,lssb,ssbIdx,nid);
```

Input Arguments

`trblk` — BCH transport block
24-by-1 binary column vector

BCH transport block, specified as a 24-by-1 binary column vector. The input `trblk` is the *BCCH-BCH-Message*, as defined in TS 38.331 Section 6.2.1 [2]. The *BCCH-BCH-Message* contains the master information block (MIB), as defined in TS 38.331 Section 6.2.2.

Data Types: `double` | `int8`

`sfn` — System frame number
nonnegative integer

System frame number, specified as a nonnegative integer.

Data Types: `double`

`hrf` — Half frame bit in SS/PBCH block transmissions
`0` | `1`

Half frame bit in SS/PBCH block transmissions, specified as `0` for the first half of a frame or `1` for the second half of a frame. For more information, see TS 38.214 Section 4.1 [3].

Data Types: `double`

`lssb` — Number of candidate SS/PBCH blocks
`4` | `8` | `64`

Number of candidate SS/PBCH blocks in a half frame, specified as `4`, `8`, or `64`.

Data Types: `double`

`idxoffset` — Subcarrier offset or SS block index
nonnegative integer

Subcarrier offset or SS block index, specified as a nonnegative integer.

- If `lssb` is `4` or `8`, `idxoffset` specifies the subcarrier offset, which must be an integer from 0 to 31.
- If `lssb` is `64`, `idxoffset` specifies the SS block index, which must be an integer from 0 to 63.

Data Types: `double`

`ncellid` — Physical layer cell identity number
integer

Physical layer cell identity number, specified as an integer from 0 to 1007.

Data Types: `double`

Output Arguments

`cdblk` — Encoded BCH transport block
864-by-1 binary column vector

Encoded BCH transport block, returned as an 864-by-1 binary column vector. `cdblk` inherits the data type of the input `trblk`.

Data Types: `double` | `int8`

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.331. “NR; Radio Resource Control (RRC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 38.214. “NR; Physical layer procedures for data.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrPBCHDecode` | `nrPBCH` | `nrBCHDecode`

nrBCHDecode

Broadcast channel (BCH) decoding

Syntax

```
scrblk = nrBCHDecode(softbits,L)
[scrblk,errFlag] = nrBCHDecode(softbits,L)
[scrblk,errFlag,trblk,lsbofsfn,hrf,msbidxoffset] = nrBCHDecode(softbits,L,
lssb,ncellid)
```

Description

`scrblk = nrBCHDecode(softbits,L)` decodes the log-likelihood ratios (LLRs) `softbits` in accordance with TS 38.212, Section 7.1 [1]. The function returns the decoded scrambled BCH transport block `scrblk`. The input argument `L` is the list length used for polar decoding.

`[scrblk,errFlag] = nrBCHDecode(softbits,L)` also returns an error flag, `errFlag`, to indicate whether `scrblk` contains an error after decoding.

`[scrblk,errFlag,trblk,lsbofsfn,hrf,msbidxoffset] = nrBCHDecode(softbits,L,lssb,ncellid)` also returns the decoded and unscrambled BCH transport block `trblk`. The additional input arguments are the number of candidate synchronization signal / physical broadcast channel (SS/PBCH) blocks, `lssb`, and the physical layer cell identity number, `ncellid`. The function also returns these information elements:

- `lsbofsfn`, the four least significant bits (LSBs) of the system frame number
- `hrf`, the half frame bit
- `msbidxoffset`, the most significant bits (MSBs) of the index offset

Examples

Decode Scrambled BCH Transport Block

Generate a random sequence of binary values corresponding to a BCH transport block of 24 bits.

```
trblk = randi([0 1],24,1,'int8');
```

Specify the physical layer cell identity number as 321, the system frame number as 10, and the second half frame.

```
nid = 321;
sfn = 10;
hrf = 1;
```

Specify the number of candidate SS/PBCH blocks as 8. When you specify the number of candidate SS/PBCH blocks as `4` or `8`, you can specify the subcarrier offset `kssb` as an input argument to the BCH encoder.

```
lssb = 8;
kssb = 18;
```

Encode the BCH transport block using the specified arguments.

```
bch = nrBCH(trblk,sfn,hrf,lssb,kssb,nid);
```

Decode the encoded transport block and recover information by using a polar decoding list length of 8 bits.

```
listLen = 8;
[~,errFlag,rxtrblk,rxSFN4lsb,rxHRF,rxKssb] = nrBCHDecode( ...
   double(1-2*bch),listLen,lssb,nid);
```

Verify that the decoding has no errors.

```
errFlag
```

```
errFlag = uint32
    0
```

```
isequal(trblk,rxtrblk)
```

```
ans = logical
   1
```

```
isequal(bit2int(rxSFN4lsb,4),mod(sfn,16))
```

```
ans = logical
   1
```

```
[isequal(hrf,rxHRF) isequal(int2bit(floor(kssb/16),1,false),rxKssb)]
```

```
ans = 1x2 logical array

   1   1
```

Input Arguments

`softbits` — Approximate log-likelihood ratio (LLR) soft bits

864-by-1 real-valued column vector

Approximate log-likelihood ratio (LLR) soft bits, specified as an 864-by-1 real-valued column vector.

Data Types: `single` | `double`

L — Polar decoding list length

power of 2

Polar decoding list length, specified as a power of 2.

Data Types: `double`

`lssb` — Number of candidate SS/PBCH blocks in a half frame

4 | 8 | 64

Number of candidate SS/PBCH blocks in a half frame, specified as `4`, `8`, or `64`.

Data Types: `double`

`ncellid` — Physical layer cell identity number
integer

Physical layer cell identity number, specified as an integer from 0 to 1007.

Data Types: `double`

Output Arguments

`scrblk` — Decoded scrambled BCH transport block
32-by-1 binary column vector

Decoded scrambled BCH transport block, returned as a 32-by-1 binary column vector.

Data Types: `int8`

`errFlag` — Error flag
`0` | `1`

Error flag to indicate whether `scrblk` contains an error, returned as `0` or `1`. If `errFlag` is `1`, then an error has occurred.

Data Types: `uint32`

`trblk` — Decoded and unscrambled BCH transport block
24-by-1 binary column vector

Decoded and unscrambled BCH transport block, returned as a 24-by-1 binary column vector. The output `trblk` is the *BCCH-BCH-Message*, as defined in TS 38.331 Section 6.2.1 [2]. The *BCCH-BCH-Message* contains the master information block (MIB), as defined in TS 38.331 Section 6.2.2.

Data Types: `logical`

`lsbofsfn` — LSBs of the system frame number
4-by-1 column vector

The four LSBs of the system frame number, returned as a 4-by-1 column vector.

Data Types: `logical`

`hrf` — Half frame bit in SS/PBCH block transmissions
`0` | `1`

Half frame bit in SS/PBCH block transmissions, returned as `0` indicating the first half of a frame or `1` indicating the second half of a frame. For more information, see TS 38.214 Section 4.1 [3].

Data Types: `logical`

`msbidxoffset` — MSBs of index offset
scalar | 3-by-1 column vector

MSBs of index offset, returned as a scalar or 3-by-1 column vector.

- If `lssb` is `4` or `8`, `msbidxoffset` is the decoded MSB of the subcarrier index, returned as a scalar.
- If `lssb` is `64`, the entries of `msbidxoffset` are the three decoded MSBs of the SSB index, returned as a 3-by-1 column vector.

Data Types: `logical`

Version History

Introduced in R2018b

Polar decoding metric update

Behavior changed in R2020a

In releases R2019b and before, polar decoding uses the exact form of the expression $\log(1 + e^x)$ for internal metric evaluation. Starting in release R2020a, because the exact form leads to numerical instability for high SNR ranges, polar decoding approximates $\log(1 + e^x)$ as 0 for $x < 0$ and as x for $x \geq 0$. This approximation affects the results of the `nrBCHDecode` function, resulting in a marginal degradation of the BLER performance in a link-level simulation.

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.331. “NR; Radio Resource Control (RRC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 38.214. “NR; Physical layer procedures for data.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

The input argument `L` must be a compile-time constant. Include `{coder.Constant(L)}` in the `-args` value of the `codegen` function. For more information, see `coder.Constant`.

See Also

Functions

`nrPBCHDecode` | `nrPBCH` | `nrBCH`

nrChannelEstimate

Practical channel estimation

Syntax

```
[h,nVar,info] = nrChannelEstimate(rxGrid,refInd,refSym)
[h,nVar,info] = nrChannelEstimate(rxGrid,refGrid)
[h,nVar,info] = nrChannelEstimate(carrier, ___ )
[h,nVar,info] = nrChannelEstimate( ___ ,Name,Value)
```

Description

`[h,nVar,info] = nrChannelEstimate(rxGrid,refInd,refSym)` performs practical channel estimation on the received resource grid `rxGrid` by using a reference resource grid containing reference symbols `refSym` at locations `refInd`. The function returns the channel estimate `h`, noise variance estimate `nVar`, and additional information `info` .

`[h,nVar,info] = nrChannelEstimate(rxGrid,refGrid)` specifies a predefined reference resource grid `refGrid`.

`[h,nVar,info] = nrChannelEstimate(carrier, ___ )` specifies carrier configuration parameters for a specific orthogonal frequency-division multiplexing (OFDM) numerology, in addition to the input arguments from any of the previous syntaxes. The function uses only the `CyclicPrefix` property of the `carrier` input.

`[h,nVar,info] = nrChannelEstimate( ___ ,Name,Value)` specifies options by using one or more name-value pair arguments in addition to the input arguments in any of the previous syntaxes.

Examples

Compare Practical and Perfect Channel Estimates

Generate physical broadcast channel (PBCH) demodulation reference signal (DM-RS) symbols for physical layer cell identity number 42. The time-dependent part of the DM-RS scrambling initialization is 0.

```
ncellid = 42;
ibar_SSB = 0;
dmrsSym = nrPBCHDMRS(ncellid,ibar_SSB);
```

Obtain resource element indices for the PBCH DM-RS.

```
dmrsInd = nrPBCHDMRSIndices(ncellid);
```

Create a resource grid containing the generated DM-RS symbols.

```
nrb = 20;
scs = 15;
carrier = nrCarrierConfig('NSizeGrid',nrb,'SubcarrierSpacing',scs);
nTxAnts = 1;
```

```
txGrid = nrResourceGrid(carrier,nTxAnts);
txGrid(dmrsInd) = dmrsSym;
```

Modulate the resource grid using the specified FFT length and cyclic prefix length.

```
ofdmInfo = nrOFDMInfo(carrier);
txWaveform = nrOFDMModulate(carrier,txGrid);
```

Create a TDL-C channel model with the specified properties.

```
channel = nrTDLChannel;
channel.NumReceiveAntennas = 1;
channel.SampleRate = ofdmInfo.SampleRate;
channel.DelayProfile = 'TDL-C';
channel.DelaySpread = 100e-9;
channel.MaximumDopplerShift = 20;
```

Obtain the maximum number of delayed samples from the channel path by using the largest delay and the implementation delay of the channel filter.

```
chInfo = info(channel);
maxChDelay = ceil(max(chInfo.PathDelays*channel.SampleRate)) + chInfo.ChannelFilterDelay;
```

To flush delayed samples from the channel, append zeros at the end of the transmitted waveform corresponding to the maximum number of delayed samples and the number of transmit antennas. Transmit the padded waveform through the TDL-C channel model.

```
[rxWaveform,pathGains] = channel([txWaveform; zeros(maxChDelay,nTxAnts)]);
```

Estimate timing offset for the transmission using the DM-RS symbols as reference symbols. The OFDM modulation of the reference symbols uses an initial slot number of 0.

```
initialSlot = 0;
offset = nrTimingEstimate(carrier,rxWaveform,txGrid);
```

Synchronize the received waveform according to the estimated timing offset.

```
rxWaveform = rxWaveform(1+offset:end,:);
```

Create a received resource grid containing the demodulated and synchronized received waveform.

```
cpFraction = 0.55;
rxGrid = nrOFDMDemodulate(carrier,rxWaveform,'CyclicPrefixFraction',cpFraction);
```

Obtain the practical channel estimate.

```
H = nrChannelEstimate(rxGrid,dmrsInd,dmrsSym);
```

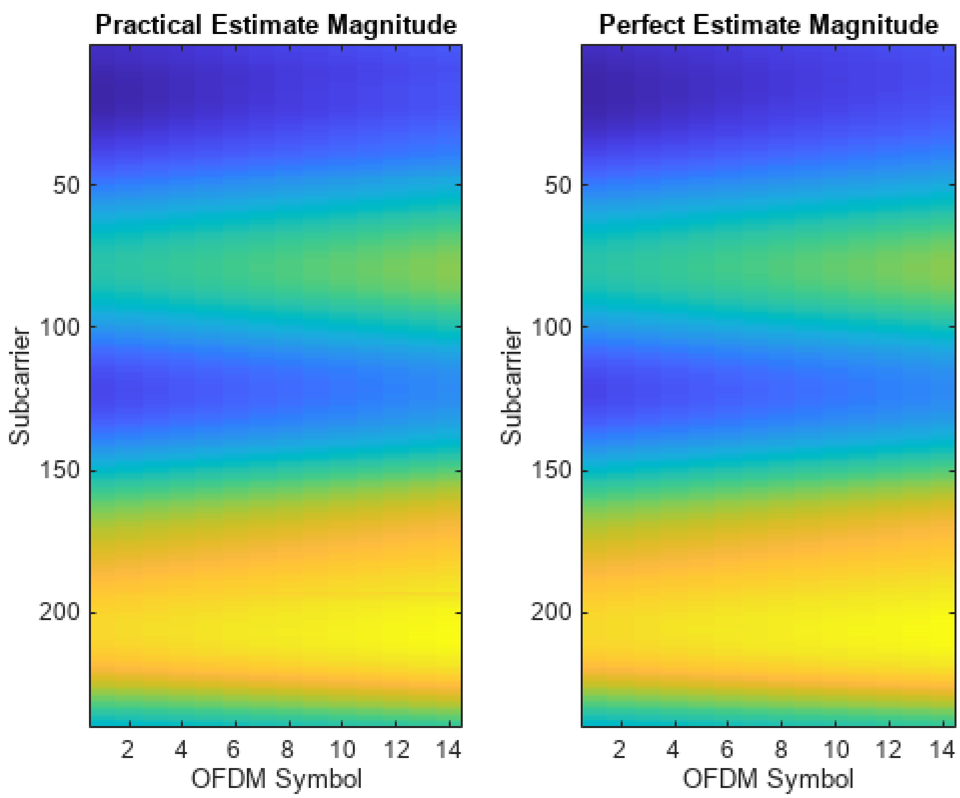
Obtain the perfect channel estimate.

```
pathFilters = getPathFilters(channel);
H_ideal = nrPerfectChannelEstimate(carrier,pathGains,pathFilters,offset);
```

Compare practical and perfect channel estimates.

```
figure;
subplot(1,2,1);
imagesc(abs(H));
xlabel('OFDM Symbol');
```

```
ylabel('Subcarrier');
title('Practical Estimate Magnitude');
subplot(1,2,2);
imagesc(abs(H_ideal));
xlabel('OFDM Symbol');
ylabel('Subcarrier');
title('Perfect Estimate Magnitude');
```



Input Arguments

rxGrid — Received resource grid
K-by-*L*-by-*R* complex array

Received resource grid, specified as a *K*-by-*L*-by-*R* complex array.

- *K* is the number of subcarriers equal to *NRB* × 12, where *NRB* is the number of resource blocks in the range from 1 to 275.
- *L* is the number of OFDM symbols in a slot or in a reference grid.
  - When you call `nrChannelEstimate` with reference symbols `refSym`, *L* is 12 for extended cyclic prefix and 14 for normal cyclic prefix. Set the cyclic prefix length by using the `'CyclicPrefix'` name-value pair argument.
  - When you call `nrChannelEstimate` with reference resource grid `refGrid`, *L* must equal *N*, the number of OFDM symbols in the reference grid.

- *R* is the number of receive antennas.

Data Types: `single` | `double`
Complex Number Support: Yes

`refInd` — Reference symbol indices
integer matrix

Reference symbol indices, specified as an integer matrix. The number of rows equals the number of resource elements. You can specify all indices in a single column or spread them across several columns. The number of elements in `refInd` and `refSym` must be the same but their dimensionality can differ. The function reshapes `refInd` and `refSym` into column vectors before mapping them into a reference grid: `refGrid(refInd(:)) = refSym(:)`.

The elements of `refInd` are 1-based linear indices addressing a *K*-by-*L*-by-*P* resource array.

- *K* is the number of subcarriers equal to *NRB* × 12, where *NRB* is the number of resource blocks in the range from 1 to 275. *K* must be equal to the first dimension of `rxGrid`.
- *L* is the number of OFDM symbols in a slot. *L* is 12 for extended cyclic prefix and 14 for normal cyclic prefix. Set the cyclic prefix length by using the `'CyclicPrefix'` name-value pair argument.
- *P* is the number of reference signal ports, inferred from the range of values in `refInd`.

Data Types: `double`

`refSym` — Reference symbols
complex matrix

Reference symbols, specified as a complex matrix. The number of rows equals the number of resource elements. You can specify all symbols in a single column or distribute them across several columns. The number of elements in `refInd` and `refSym` must be the same but their dimensionality can differ. The function reshapes `refInd` and `refSym` into column vectors before mapping them into a reference grid: `refGrid(refInd(:)) = refSym(:)`.

Data Types: `single` | `double`
Complex Number Support: Yes

`refGrid` — Predefined reference grid
K-by-*N*-by-*P* complex array

Predefined reference grid, specified as a *K*-by-*N*-by-*P* complex array. `refGrid` can span multiple slots.

- *K* is the number of subcarriers equal to *NRB* × 12, where *NRB* is the number of resource blocks in the range from 1 to 275.
- *N* is the number of OFDM symbols in the reference grid.
- *P* is the number of reference signal ports.

Data Types: `single` | `double`
Complex Number Support: Yes

`carrier` — Carrier configuration parameters
`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. The function uses only the `CyclicPrefix` property of this input.

Name-Value Pair Arguments

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

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

Example: `'CyclicPrefix','extended'` specifies extended cyclic prefix length.

`CyclicPrefix` — Cyclic prefix length

`'normal'` (default) | `'extended'`

Cyclic prefix length, specified as the comma-separated pair consisting of `'CyclicPrefix'` and one of these values:

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, the extended cyclic prefix length only applies to 60 kHz subcarrier spacing.

Note If you specify the `carrier` input, use the `CyclicPrefix` property of the `carrier` input to specify the cyclic prefix length. You cannot use this name-value pair argument together with the `carrier` input.

Data Types: `char` | `string`

`CDMLengths` — CDM arrangement for reference signals

[1 1] (default) | 1-by-2 array of nonnegative integers

Code domain multiplexing (CDM) arrangement for reference signals, specified as the comma-separated pair consisting of `'CDMLengths'` and a 1-by-2 array of nonnegative integers [*FD TD*]. Array elements *FD* and *TD* specify the length of CDM despreading in the frequency domain (FD-CDM) and time domain (TD-CDM), respectively. A value of 1 for an element specifies no CDM.

Example: `'CDMLengths',[2 1]` specifies FD-CDM2 and no TD-CDM.

Example: `'CDMLengths',[1 1]` specifies no orthogonal despreading.

Data Types: `double`

`AveragingWindow` — Pre-interpolation averaging window

`[0 0]` (default) | 1-by-2 array of nonnegative odd integers

Pre-interpolation averaging window, specified as the comma-separated pair consisting of `'AveragingWindow'` and a 1-by-2 array of nonnegative odd integers [*F T*]. Array elements *F* and *T* specify the number of adjacent reference symbols in the frequency domain and time domain, respectively, over which the function performs averaging before interpolation. If *F* or *T* is zero, the function determines the averaging value from the estimated signal-to-noise ratio (SNR) based on the noise variance estimate `nVar`.

Data Types: `double`

Output Arguments

`h` — Practical channel estimate

K-by-*L*-by-*R*-by-*P* complex array

Practical channel estimate, returned as a *K*-by-*L*-by-*R*-by-*P* complex array. *K*-by-*L*-by-*R* is the shape of the received resource grid `rxGrid`. *P* is the number of reference signal ports.

`h` inherits its data type from `rxGrid`.

Data Types: `double` | `single`

`nVar` — Noise variance estimate

nonnegative scalar

Noise variance estimate, returned as a nonnegative scalar. `nVar` is the measured variance of additive white Gaussian noise on the received reference symbols.

Data Types: `double`

`info` — Additional information

structure

Additional information, returned as a structure with the field `AveragingWindow`.

Parameter Field	Value	Description
`AveragingWindow`	1-by-2 array	Pre-interpolation averaging window, returned as a 1-by-2 array [*F* *T*]. Array elements *F* and *T* indicate the number of adjacent reference symbols in the frequency domain and time domain, respectively, over which the function performed averaging before interpolation.

Version History

Introduced in R2019b

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying extended cyclic prefix, include `{coder.Constant('CyclicPrefix'),coder.Constant('extended')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrTimingEstimate` | `nrPerfectChannelEstimate` | `nrPerfectTimingEstimate`

Objects
`nrCarrierConfig`

nrCodeBlockDesegmentLDPC

LDPC code block desegmentation and CRC decoding

Syntax

`[blk,err] = nrCodeBlockDesegmentLDPC(cbs,bgn,blklen)`

Description

`[blk,err] = nrCodeBlockDesegmentLDPC(cbs,bgn,blklen)` concatenates the input code block segments `cbs` into a single output data block `blk` of length `blklen`. The function validates the data dimensions of the input `cbs` based on the specified base graph number `bgn` and output block length `blklen`. The function removes any filler bits and type-24B cyclic redundancy check (CRC) bits present in the input `cbs`. The output `err` is the result of the type-24B CRC decoding (if applicable). This process is the inverse of the low-density parity-check (LDPC) code block segmentation specified in TS 38.212 Section 5.2.2 [1] and implemented in `nrCodeBlockSegmentLDPC`.

Examples

Back-to-Back LDPC Code Block Segmentation and Desegmentation

Perform code block segmentation of a random sequence of binary input data.

```
bgn = 1;
blklen = 10000;
cbs = nrCodeBlockSegmentLDPC(randi([0 1],blklen,1),bgn);
```

When the base graph number is 1, segmentation occurs whenever the input length is greater than 8448. The input data of length 10000 is split into two code block segments of length 5280. The code block segments have filler bits and CRC attached.

```
size(cbs)
```

```
ans = 1×2

        5280           2
```

Concatenate the code block segments.

```
[blk,err] = nrCodeBlockDesegmentLDPC(cbs,bgn,blklen);
```

The concatenated result is of the same size as the original input with CRC and filler bits removed.

```
blkSize = size(blk)
```

```
blkSize = 1×2

       10000           1
```

Verify if the CRC decoding was successful by checking the error vector.

```
err
```

```
err = 1x2 uint32 row vector

   0   0
```

Display Index Mapping of LDPC Code Block Desegmentation

Create a matrix representing two code block segments. Each element contains the linear index of that element within the matrix.

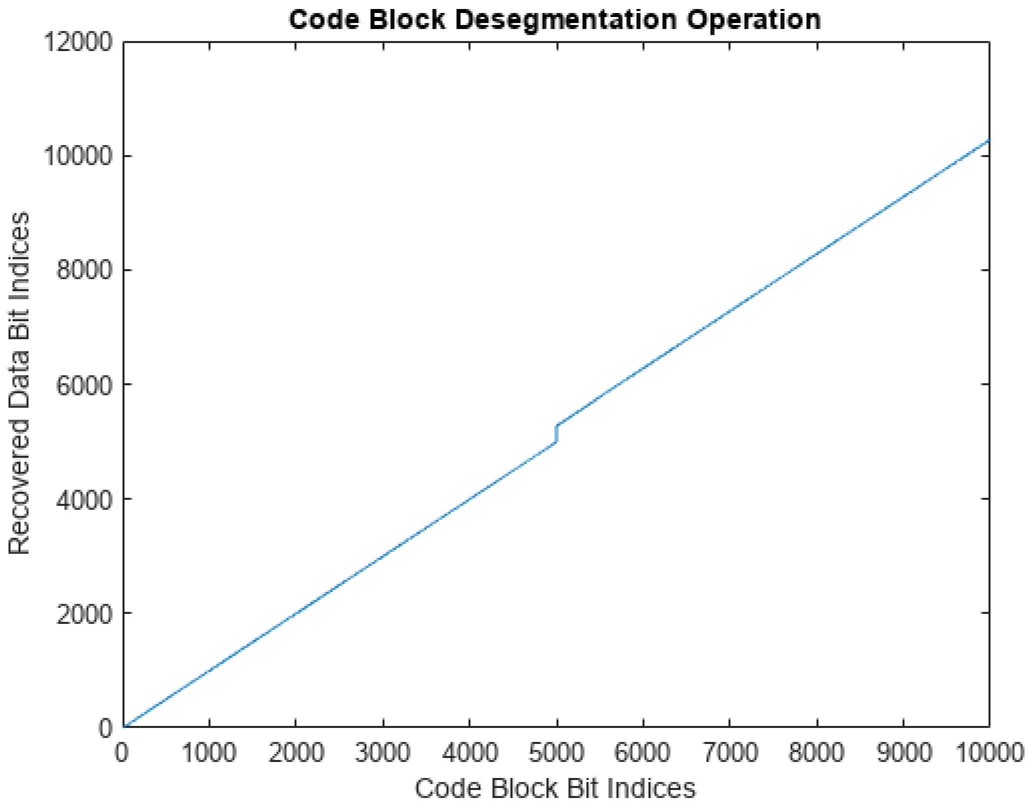
```
cbs = reshape([1:10560]',[],2);
```

Concatenate the code block segments using the specified base graph number and output block length.

```
bgn = 1;
blklen = 10000;
blk = nrCodeBlockDesegmentLDPC(cbs,bgn,blklen);
```

To see how the input maps onto the output, plot code block segment indices relative to the corresponding indices in the concatenated input. In each code block segment, the last 280 bits represent CRC and filler bits. These additional bits are removed from the recovered data.

```
plot(blk);
xlabel('Code Block Bit Indices');
ylabel('Recovered Data Bit Indices');
title('Code Block Desegmentation Operation');
```



Input Arguments

cbs — Code block segments
real matrix

Code block segments, specified as a real matrix. A matrix with only one column corresponds to one code block segment without CRC bits appended. If you specify a matrix with more than one column, each column in the matrix corresponds to a separate code block segment with type-24B CRC bits appended. In both cases, the code block segments can contain filler bits.

Data Types: `double` | `int8`

bgn — Base graph number
1 | 2

Base graph number, specified as `1` or `2`.

Data Types: `double`

blklen — Output block length
nonnegative integer

Output block length, specified as a nonnegative integer. If `blklen` is `0`, then both `blk` and `err` are empty. The function uses `blklen` to validate the data dimensions of the input `cbs` and to calculate the number of filler bits to remove.

Data Types: `double`

Output Arguments

`blk` — Concatenated data block
empty vector | real column vector

Concatenated data block, returned as an empty vector (when `blklen` is `0`) or a real column vector. The function removes any filler bits and type-24B CRC bits present in the input `cbs`. The output `blk` inherits its data type from the input `cbs`.

Data Types: `double` | `int8`

`err` — CRC error
empty vector | vector of nonnegative integers

CRC error, returned as one of these values:

- Empty vector — The function returns this value when `blklen` is `0` or if `cbs` has only one column (CRC decoding does not take place).
- Vector of nonnegative integers — If `cbs` has more than one column, `err` contains the CRC error bits obtained from decoding the type-24B CRC bits in each code block segment. The length of `err` is equal to the number of code block segments (number of columns in the input `cbs`).

Data Types: `uint32`

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.212. "NR; Multiplexing and channel coding." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrCodeBlockSegmentLDPC` | `nrCRCDecode` | `nrLDPCDecode` | `nrRateRecoverLDPC`

nrCodeBlockSegmentLDPC

LDPC code block segmentation and CRC attachment

Syntax

`cbs = nrCodeBlockSegmentLDPC(blk,bgn)`

Description

`cbs = nrCodeBlockSegmentLDPC(blk,bgn)` splits the input data block `blk` into code block segments based on the base graph number `bgn`, as specified in TS 38.212 Section 5.2.2 [1]. The function appends cyclic redundancy check (CRC) and filler bits to each code block segment in `cbs` (if applicable). `nrCodeBlockSegmentLDPC` provides input to low-density parity-check (LDPC) coders in transport channels, including downlink and uplink shared channels, and paging channels.

Examples

LDPC Code Block Segmentation

Create a random sequence of binary input data.

```
in = randi([0,1],4000,1);
```

Perform LDPC code block segmentation.

```
cbs1 = nrCodeBlockSegmentLDPC(in,1);
cbs2 = nrCodeBlockSegmentLDPC(in,2);
```

When the base graph number is 1, the segmentation results in one code block segment. When the base graph number is 2, the segmentation results in two code block segments. Segmentation occurs only if the input length is greater than the maximum code block size. The maximum code block size is 8448 when the base graph number is 1 and 3840 when the base graph number is 2.

```
size(cbs1)
```

```
ans = 1×2

        4224           1
```

```
size(cbs2)
```

```
ans = 1×2

        2080           2
```

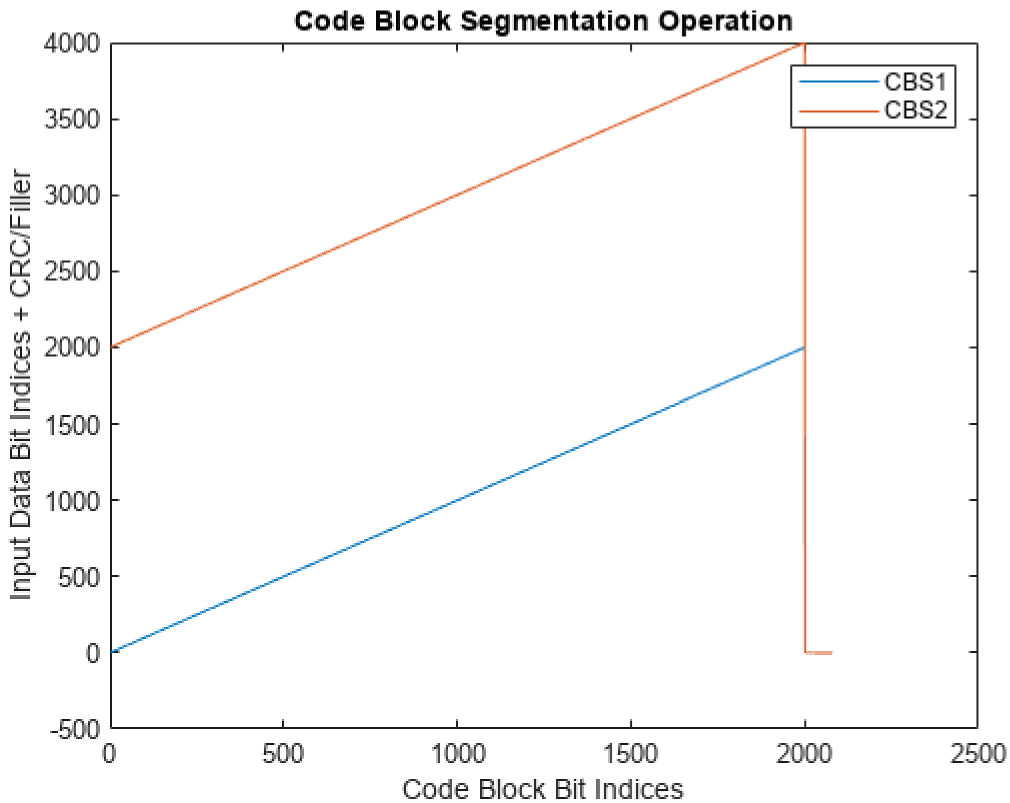
Display Index Mapping of LDPC Code Block Segmentation

Create a ramp data input and perform code block segmentation.

```
cbs = nrCodeBlockSegmentLDPC([1:4000]',2);
```

The input of length 4000 is split into two code block segments of equal size with 24B CRC and filler bits attached. To see how the input maps onto the output, plot the input data indices relative to the corresponding code block segment indices.

```
plot(cbs)
legend('CBS1','CBS2')
xlabel('Code Block Bit Indices');
ylabel('Input Data Bit Indices + CRC/Filler');
title('Code Block Segmentation Operation')
```



Input Arguments

blk — Input data block
column vector of real numbers

Input data block, specified as a column vector of real numbers.

Data Types: `double` | `int8` | `logical`

bgn — Base graph number
1 | 2

Base graph number, specified as 1 or 2.

Data Types: double

Output Arguments

cbs — Code block segments
integer or real matrix

Code block segments, returned as an integer or real matrix. Each column corresponds to a separate code block segment. The number of code block segments depends on the maximum code block size of the LDPC coder, *Kcb*, and the length of the input blk, *B*. If bgn is set to 1, *Kcb* = 8448. If bgn is set to 2, *Kcb* = 3840. If $B \leq Kcb$, then the function does not perform segmentation and does not append CRC to the resulting code block. If $B > Kcb$, the segmentation results in several smaller code blocks with a type-24B CRC bits appended.

The function appends filler bits to each code block (with or without CRC) if necessary. The filler bits ensure that the code block segments entering the LDPC coder have a valid length and are a multiple of the LDPC lifting size. To accommodate the filler bits represented by –1, the data type of cbs is cast to int8 when the input blk is logical. Otherwise, cbs inherits the data type of the input blk.

Data Types: double | int8

Version History
Introduced in R2018b

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
nrCodeBlockDesegmentLDPC | nrLDPCEncode | nrRateMatchLDPC | nrRateMatchLDPC | nrLDPCEncode

nrCRCDecode

Decode and remove cyclic redundancy check (CRC)

Syntax

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

Description

`[blk,err] = nrCRCDecode(blkcrc,poly)` checks the input data `blkcrc` for a CRC error. The function assumes that the input data comprises the CRC parity bits associated with the polynomial `poly`. The function returns `blk`, which is the data part of the input `blkcrc`. The function also returns `err`, which is the logical difference (XOR) between the CRC comprised in the input and the CRC recalculated across the data part of the input. If `err` is not equal to 0, either an error has occurred or the input CRC has been masked. For details on the associated polynomials, see TS 38.212 Section 5.1 [1].

`[blk,err] = nrCRCDecode(blkcrc,poly,mask)` XOR-masks the CRC difference with `mask` before returning it in `err`. The `mask` value is applied to the CRC difference with the most significant bit (MSB) first to the least significant bit (LSB) last.

Examples

Check Data Block for CRC Error

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

Define a mask corresponding to the radio network temporary identifier (RNTI) equal to 12. Append RNTI-masked CRC parity bits to an all-ones matrix of one data block.

```
rnti = 12;
blkCrc = nrCRCEncode(ones(100,1),'24C',rnti);
```

When you perform CRC decoding without a mask, `err1` is equal to the RNTI because the CRC was masked during coding. The logical difference between the original CRC and the recalculated CRC is the CRC mask.

```
[blk,err1] = nrCRCDecode(blkCrc,'24C');
err1
```

```
err1 = uint32
    12
```

When you perform CRC decoding using the RNTI value as a mask, `err` is equal to 0.

```
[blk,err2] = nrCRCDecode(blkCrc,'24C',err1);
err2
```

```
err2 = uint32
    0
```

Input Arguments

`blkcrc` — CRC encoded data
matrix of real numbers

CRC encoded data, specified as a matrix of real numbers. Each column of the matrix is considered as a separate CRC encoded data block.

Data Types: `double` | `int8` | `logical`

`poly` — CRC polynomial
`'6'` | `'11'` | `'16'` | `'24A'` | `'24B'` | `'24C'`

CRC polynomial, specified as `'6'`, `'11'`, `'16'`, `'24A'`, `'24B'`, or `'24C'`. For details on the associated polynomials, see TS 38.212 Section 5.1.

Data Types: `char` | `string`

`mask` — XOR mask
`0` (default) | nonnegative integer

XOR mask, specified as a nonnegative integer. The mask is typically a radio network temporary identifier (RNTI).

Data Types: `double`

Output Arguments

`blk` — CRC decoded data
matrix of real numbers

CRC decoded data, returned as a matrix of real numbers. `blk` is the data-only part of the input `blkcrc`.

Data Types: `double` | `int8` | `logical`

`err` — Logical CRC difference
integer

Logical CRC difference, returned as an integer. `err` is the logical difference between the CRC comprised in the input `blkcrc` and the CRC recalculated across the data part of the input. If a mask is specified, the function XOR-masks `err` with `mask` before returning it.

Data Types: `uint32`

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrCRCEncode` | `nrPolarDecode` | `nrLDPCDecode` | `nrCodeBlockDesegmentLDPC` | `nrRateRecoverPolar` | `nrRateRecoverLDPC` | `nrBCHDecode` | `nrDCIDecode`

nrCRCEncode

Calculate and append cyclic redundancy check (CRC)

Syntax

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

Description

`blkcrc = nrCRCEncode(blk,poly)` calculates the CRC defined by the polynomial `poly` for the input data `blk`. The function returns the CRC encoded data, which is a copy of the input data with the CRC parity bits appended. For details on the associated polynomials, see TS 38.212 Section 5.1 [1].

`blkcrc = nrCRCEncode(blk,poly,mask)` applies a logical difference (XOR) mask on the appended CRC bits with the integral value of `mask`. The appended CRC bits in `blkcrc` are XOR-masked with the most significant bit (MSB) first to the least significant bit (LSB) last. The masked CRC is of the form (p_0 xor m_0), (p_1 xor m_1),..., (p_{L-1} xor m_{L-1}), where L is the number of parity bits, and p_0 and m_0 are the MSBs in the binary representation of CRC and `mask`, respectively. If the mask value is greater than 2^L - 1, the L LSBs are considered for the mask.

Examples

Calculate and Append CRC

Calculate and append CRC parity bits to an all-zeros matrix of two data blocks. The result is an all-zeros matrix of size 124-by-2.

```
blkcrc = nrCRCEncode(zeros(100,2),'24C');
any(blkcrc(:,1:2));
```

Calculate and Append Masked CRC

Calculate and append masked CRC parity bits to an all-zeros matrix of two data blocks. The appended CRC bits are XOR-masked with the specified `mask`, from the MSB first to the LSB last. The result is an all-zeros matrix apart from the elements in the last position.

```
mask = 1;
blkcrc = nrCRCEncode(zeros(100,2),'24C',mask);
blkcrc(end-5:end,1:2)
```

```
ans = 6×2

     0     0
     0     0
     0     0
     0     0
```

```
0     0
1     1
```

Input Arguments

blk — Input data
matrix of real numbers

Input data, specified as a matrix of real numbers. Each column of the matrix is treated as a separate data block.

Data Types: double | int8 | logical

poly — CRC polynomial
'6' | '11' | '16' | '24A' | '24B' | '24C'

CRC polynomial, specified as '6', '11', '16', '24A', '24B', or '24C'. For details on the associated polynomials, see TS 38.212 Section 5.1.

Data Types: char | string

mask — XOR mask
0 (default) | nonnegative integer

XOR mask, specified as a nonnegative integer. The mask is typically a radio network temporary identifier (RNTI).

Data Types: double

Output Arguments

blkcrc — CRC encoded data
matrix of real numbers

CRC encoded data, returned as a matrix of real numbers. blkcrc is a copy of the input blk with the CRC parity bits appended. Each column corresponds to a separate CRC encoded code block. blkcrc inherits its data type from the input blk.

Data Types: double | int8 | logical

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.212. "NR; Multiplexing and channel coding." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
nrCRCDecode | nrPolarEncode | nrRateMatchLDPC | nrLDPCEncode | nrDCIEncode | nrBCH | nrRateMatchPolar | nrCodeBlockSegmentLDPC

nrCSIRS

Generate CSI-RS symbols

Syntax

```
[sym,info] = nrCSIRS(carrier,csirs)
[sym,info] = nrCSIRS(carrier,csirs,Name,Value)
```

Description

`[sym,info] = nrCSIRS(carrier,csirs)` returns channel state information reference signal (CSI-RS) symbols `sym`, as defined in TS 38.211 Section 7.4.1.5 [1]. The input `carrier` specifies carrier configuration parameters for a specific OFDM numerology. The input `csirs` specifies CSI-RS resource configuration parameters for one or more zero-power (ZP) or non-zero-power (NZP) CSI-RS resources. When configuring both ZP and NZP resources, the returned symbols are in the order ZP followed by NZP, irrespective of the resource order specified by `csirs`. The function also returns the structure `info`, which contains additional information about the CSI-RS locations.

`[sym,info] = nrCSIRS(carrier,csirs,Name,Value)` specifies output formatting options by using one or more name-value pair arguments. Unspecified options take default values.

Examples

Generate CSI-RS Symbols and Indices for 10 MHz Carrier

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier.

```
carrier = nrCarrierConfig;
```

Create a CSI-RS configuration object with default properties.

```
csirs = nrCSIRSConfig;
```

Generate CSI-RS symbols of `single` data type.

```
[sym,info_sym] = nrCSIRS(carrier,csirs,'OutputDataType','single');
```

Generate resource element indices for CSI-RS.

```
[ind,info_ind] = nrCSIRSIndices(carrier,csirs);
```

Generate ZP and NZP-CSI-RS Symbols and Indices

Create a carrier configuration object, specifying the slot number as `10`.

```
carrier = nrCarrierConfig('NSlot',10);
```

Create a CSI-RS resource configuration object for two periodic resources. Specify one NZP resource and one ZP resource with row numbers 3 and 5, symbol locations 13 and 9, and subcarrier locations 6 and 4, respectively. For both resources, set the periodicity to 5, offset to 1, and density to 'one'.

```
csirs = nrCSIRSConfig;
csirs.CSIRSType = {'nzp','zp'};
csirs.CSIRSPeriod = {[5 1],[5 1]};
csirs.RowNumber = [3 5];
csirs.Density = {'one','one'};
csirs.SymbolLocations = {13,9};
csirs.SubcarrierLocations = {6,4};
```

Generate CSI-RS symbols and indices for the specified carrier, CSI-RS resource configuration, and output formatting name-value pair arguments. Verify the format of the symbols and indices.

```
[sym,info_sym] = nrCSIRS(carrier,csirs,...
                'OutputResourceFormat','cell')
```

```
sym=1×2 cell array
    {0x1 double}    {0x1 double}


info_sym = struct with fields:
      ResourceOrder: [2 1]
           KBarLBar: {{1x1 cell}  {1x2 cell}}
    CDMGroupIndices: {[0]  [0 1]}
             KPrime: {[0 1]  [0 1]}
             LPrime: {[0]  [0]}
```

```
[ind,info_ind] = nrCSIRSIndices(carrier,csirs,...
                'IndexStyle','subscript','OutputResourceFormat','cell')
```

```
ind=1×2 cell array
    {0x3 uint32}    {0x3 uint32}


info_ind = struct with fields:
      ResourceOrder: [2 1]
           KBarLBar: {{1x1 cell}  {1x2 cell}}
    CDMGroupIndices: {[0]  [0 1]}
             KPrime: {[0 1]  [0 1]}
             LPrime: {[0]  [0]}
```

Verify that the generated outputs are in the order of ZP-CSI-RS resources followed by NZP-CSI-RS resources in terms of the specified `csirs.CSIRSType` indices.

```
info_sym.ResourceOrder
```

```
ans = 1×2

     2     1
```

```
info_ind.ResourceOrder
```

```
ans = 1×2
```

```
     2     1
```

Generate and Map CSI-RS Symbols Used for Tracking

Create a carrier configuration object with default properties.

```
carrier = nrCarrierConfig;
```

Create a CSI-RS resource configuration object with CSI-RS parameters set for tracking. Specify four periodic NZP-CSI-RS resources in two consecutive slots. Specify for each slot to contain two periodic NZP-CSI-RS resources with periodicity set to `20`. Set the offset for the first two resources to `0`. Set the offset for the next two resources to `1`. Set the row number to `1` and density to `'three'` for all resources.

```
csirs = nrCSIRSConfig;
csirs.CSIRSType = {'nzp','nzp','nzp','nzp'};
csirs.CSIRSPeriod = {[20 0],[20 0],[20 1],[20 1]};
csirs.RowNumber = [1 1 1 1];
csirs.Density = {'three','three','three','three'};
csirs.SymbolLocations = {6,10,6,10};
csirs.SubcarrierLocations = {0,0,0,0};
```

Generate CSI-RS symbols and indices for the default slot number of the carrier configuration object (slot number 0).

```
ind0 = nrCSIRSIndices(carrier,csirs);
sym0 = nrCSIRS(carrier,csirs);
```

Map the symbols to a carrier grid of one slot duration.

```
gridSize = [12*carrier.NSizeGrid carrier.SymbolsPerSlot max(csirs.NumCSIRSPorts)];
slotgrid0 = complex(zeros(gridSize));
slotgrid0(ind0) = sym0;
```

Change the absolute slot number in the carrier configuration from `0` to `1`.

```
carrier.NSlot = 1;
```

Generate CSI-RS symbols and indices for slot number 1.

```
ind1 = nrCSIRSIndices(carrier,csirs);
sym1 = nrCSIRS(carrier,csirs);
```

Map the symbols to another carrier grid of one slot duration.

```
slotgrid1 = complex(zeros(gridSize));
slotgrid1(ind1) = sym1;
```

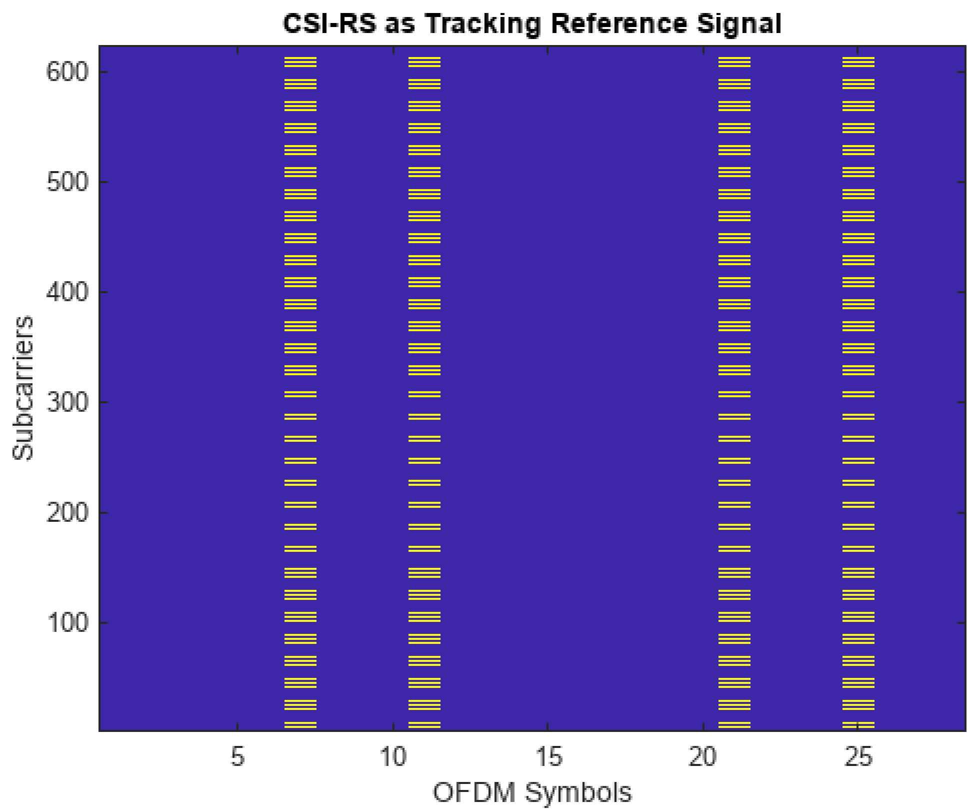
Concatenate the two slots to form the final grid.

```
grid = [slotgrid0 slotgrid1];
```

Plot the grid.

```
imagesc(abs(grid(:,:,1)));
axis xy;
```

```
title('CSI-RS as Tracking Reference Signal');
xlabel('OFDM Symbols');
ylabel('Subcarriers');
```



Input Arguments

carrier — Carrier configuration parameters

nrCarrierConfig object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object.

csirs — CSI-RS resource configuration parameters

nrCSIRSConfig object

CSI-RS resource configuration parameters, specified as an `nrCSIRSConfig` object.

Name-Value Pair Arguments

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

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

Example: `'OutputDataType','single'` specifies `single` data type for the output symbols.

OutputDataType — Data type of generated CSI-RS symbols
'double' (default) | 'single'

Data type of generated CSI-RS symbols, specified as the comma-separated pair consisting of 'OutputDataType' and 'double' or 'single'.

Data Types: char | string

OutputResourceFormat — Output format of CSI-RS symbols
'concatenated' (default) | 'cell'

Output format of CSI-RS symbols, specified as the comma-separated pair consisting of 'OutputResourceFormat' and one of these values:

- 'concatenated' — The output sym is a single column vector containing all CSI-RS symbols concatenated.
- 'cell' — The output sym is a cell array where each cell corresponds to a single CSI-RS resource.

Data Types: char | string

Output Arguments

sym — CSI-RS symbols
complex column vector | cell array of complex column vectors

CSI-RS symbols, returned as a complex column vector or a cell array of complex column vectors.

Data Types: single | double

info — CSI-RS locations information
structure

CSI-RS locations information, returned as a structure containing these fields:

Fields	Description
ResourceOrder	Order of CSI-RS resources in terms of CSIRSType indices. CSIRSType is a property of the input csirs configuration object, specifying all CSI-RS resources for which the function generates the output.
KBarLBar	Frequency-domain and time-domain locations of the lowest resource elements corresponding to all code division multiplexing (CDM) groups
CDMGroupIndices	CDM group indices
KPrime	Frequency-domain indexing within a CDM group
LPrime	Time-domain indexing within a CDM group

Each field, apart from ResourceOrder, returns the information in the resource order specified by the CSIRSType property of the input csirs configuration object. These fields represent the frequency-domain and time-domain locations of the CSI-RS within a slot for each resource, as defined in TS 38.211 Table 7.4.1.5.3-1.

Version History

Introduced in R2019b

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying `'single'` as the output data type, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrCSIRSIndices` | `nrCSIRSMeasurements`

Objects
`nrCSIRSConfig` | `nrCarrierConfig`

nrCSIRSIndices

Generate CSI-RS resource element indices

Syntax

```
[ind,info] = nrCSIRSIndices(carrier,csirs)
[ind,info] = nrCSIRSIndices(carrier,csirs,Name,Value)
```

Description

`[ind,info] = nrCSIRSIndices(carrier,csirs)` returns resource element indices `ind` for the channel state information reference signal (CSI-RS), as defined in TS 38.211 Section 7.4.1.5.3 [1]. The input `carrier` specifies carrier configuration parameters for a specific OFDM numerology. The input `csirs` specifies CSI-RS resource configuration parameters for one or more zero-power (ZP) or non-zero-power (NZP) CSI-RS resources. When configuring both ZP and NZP resources, the returned indices are in the order ZP followed by NZP, irrespective of the resource order specified by `csirs`. The function also returns the structure `info`, which contains additional information about the CSI-RS locations.

`[ind,info] = nrCSIRSIndices(carrier,csirs,Name,Value)` specifies output formatting options by using one or more name-value pair arguments. Unspecified options take default values.

Examples

Generate CSI-RS Symbols and Indices for 10 MHz Carrier

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier.

```
carrier = nrCarrierConfig;
```

Create a CSI-RS configuration object with default properties.

```
csirs = nrCSIRSConfig;
```

Generate CSI-RS symbols of `single` data type.

```
[sym,info_sym] = nrCSIRS(carrier,csirs,'OutputDataType','single');
```

Generate resource element indices for CSI-RS.

```
[ind,info_ind] = nrCSIRSIndices(carrier,csirs);
```

Generate ZP and NZP-CSI-RS Symbols and Indices

Create a carrier configuration object, specifying the slot number as `10`.

```
carrier = nrCarrierConfig('NSlot',10);
```

Create a CSI-RS resource configuration object for two periodic resources. Specify one NZP resource and one ZP resource with row numbers 3 and 5, symbol locations 13 and 9, and subcarrier locations 6 and 4, respectively. For both resources, set the periodicity to 5, offset to 1, and density to `'one'`.

```
csirs = nrCSIRSConfig;
csirs.CSIRSType = {'nzp','zp'};
csirs.CSIRSPeriod = {[5 1],[5 1]};
csirs.RowNumber = [3 5];
csirs.Density = {'one','one'};
csirs.SymbolLocations = {13,9};
csirs.SubcarrierLocations = {6,4};
```

Generate CSI-RS symbols and indices for the specified carrier, CSI-RS resource configuration, and output formatting name-value pair arguments. Verify the format of the symbols and indices.

```
[sym,info_sym] = nrCSIRS(carrier,csirs,...
                'OutputResourceFormat','cell')
```

```
sym=1×2 cell array
    {0x1 double}    {0x1 double}


info_sym = struct with fields:
      ResourceOrder: [2 1]
           KBarLBar: {{1x1 cell}  {1x2 cell}}
    CDMGroupIndices: {[0]  [0 1]}
             KPrime: {[0 1]  [0 1]}
             LPrime: {[0]  [0]}
```

```
[ind,info_ind] = nrCSIRSIndices(carrier,csirs,...
                'IndexStyle','subscript','OutputResourceFormat','cell')
```

```
ind=1×2 cell array
    {0x3 uint32}    {0x3 uint32}


info_ind = struct with fields:
      ResourceOrder: [2 1]
           KBarLBar: {{1x1 cell}  {1x2 cell}}
    CDMGroupIndices: {[0]  [0 1]}
             KPrime: {[0 1]  [0 1]}
             LPrime: {[0]  [0]}
```

Verify that the generated outputs are in the order of ZP-CSI-RS resources followed by NZP-CSI-RS resources in terms of the specified `csirs.CSIRSType` indices.

```
info_sym.ResourceOrder
```

```
ans = 1×2

     2     1
```

```
info_ind.ResourceOrder
```

```
ans = 1×2
```

2 1

Generate and Map CSI-RS Symbols Used for Tracking

Create a carrier configuration object with default properties.

```
carrier = nrCarrierConfig;
```

Create a CSI-RS resource configuration object with CSI-RS parameters set for tracking. Specify four periodic NZP-CSI-RS resources in two consecutive slots. Specify for each slot to contain two periodic NZP-CSI-RS resources with periodicity set to `20`. Set the offset for the first two resources to `0`. Set the offset for the next two resources to `1`. Set the row number to `1` and density to `'three'` for all resources.

```
csirs = nrCSIRSConfig;
csirs.CSIRSType = {'nzp','nzp','nzp','nzp'};
csirs.CSIRSPeriod = {[20 0],[20 0],[20 1],[20 1]};
csirs.RowNumber = [1 1 1 1];
csirs.Density = {'three','three','three','three'};
csirs.SymbolLocations = {6,10,6,10};
csirs.SubcarrierLocations = {0,0,0,0};
```

Generate CSI-RS symbols and indices for the default slot number of the carrier configuration object (slot number 0).

```
ind0 = nrCSIRSIndices(carrier,csirs);
sym0 = nrCSIRS(carrier,csirs);
```

Map the symbols to a carrier grid of one slot duration.

```
gridSize = [12*carrier.NSizeGrid carrier.SymbolsPerSlot max(csirs.NumCSIRSPorts)];
slotgrid0 = complex(zeros(gridSize));
slotgrid0(ind0) = sym0;
```

Change the absolute slot number in the carrier configuration from `0` to `1`.

```
carrier.NSlot = 1;
```

Generate CSI-RS symbols and indices for slot number 1.

```
ind1 = nrCSIRSIndices(carrier,csirs);
sym1 = nrCSIRS(carrier,csirs);
```

Map the symbols to another carrier grid of one slot duration.

```
slotgrid1 = complex(zeros(gridSize));
slotgrid1(ind1) = sym1;
```

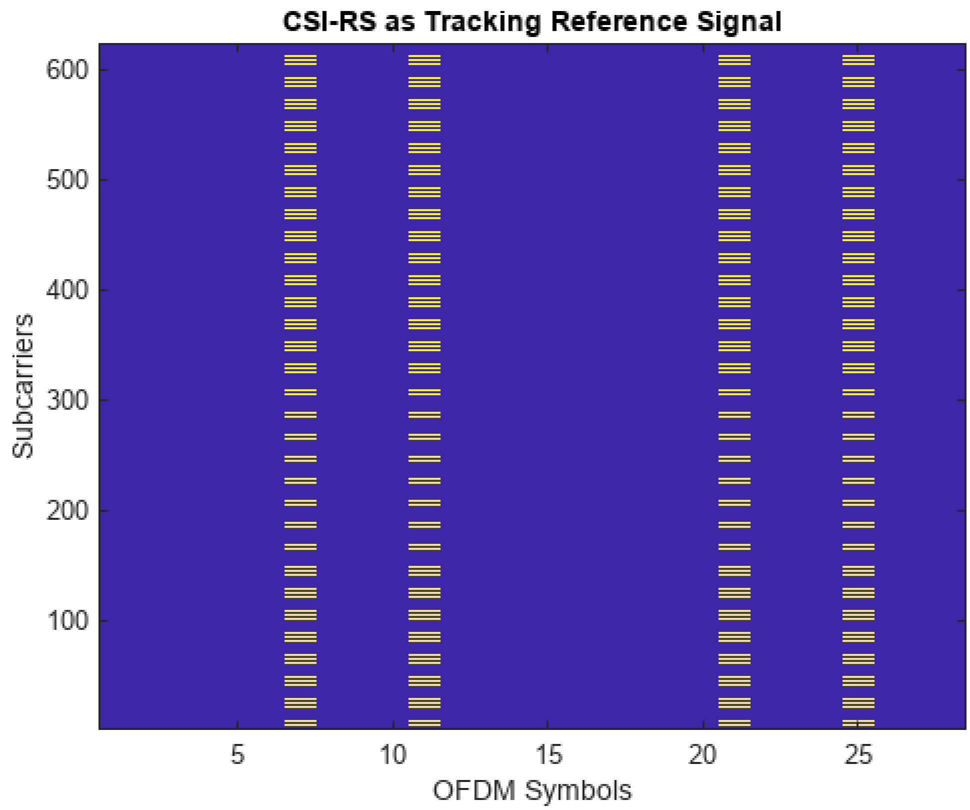
Concatenate the two slots to form the final grid.

```
grid = [slotgrid0 slotgrid1];
```

Plot the grid.

```
imagesc(abs(grid(:,:,1)));
axis xy;
```

```
title('CSI-RS as Tracking Reference Signal');
xlabel('OFDM Symbols');
ylabel('Subcarriers');
```



Input Arguments

carrier — Carrier configuration parameters
nrCarrierConfig object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object.

csirs — CSI-RS resource configuration parameters
nrCSIRSConfig object

CSI-RS resource configuration parameters, specified as an `nrCSIRSConfig` object.

Name-Value Pair Arguments

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

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

Example: 'IndexStyle','subscript','IndexBase','0based' specifies the indexing style and indexing base of the output.

IndexStyle — RE indexing form
'index' (default) | 'subscript'

RE indexing form, specified as one of these values:

- 'index' — The indices are in linear index form.
- 'subscript' — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: char | string

IndexBase — RE indexing base
'1based' (default) | '0based'

RE indexing base, specified as one of these values:

- '1based' — The index counting starts from 1.
- '0based' — The index counting starts from 0.

Data Types: char | string

OutputResourceFormat — Output format of CSI-RS resource element indices
'concatenated' (default) | 'cell'

Output format of CSI-RS resource element indices, specified as the comma-separated pair consisting of 'OutputResourceFormat' and one of these values:

- 'concatenated' — The output ind is a single column vector containing all CSI-RS resource element indices concatenated.
- 'cell' — The output ind is a cell array where each cell corresponds to a single CSI-RS resource.

Data Types: char | string

Output Arguments

ind — CSI-RS resource element indices
column vector | *M*-by-3 matrix | cell array

CSI-RS resource element indices, returned as one of these values:

- Column vector — The function returns this type of value when 'OutputResourceFormat' is set to 'concatenated' and 'IndexStyle' is set to 'index'.
- *M*-by-3 matrix — The function returns this type of value when 'OutputResourceFormat' is set to 'concatenated' and 'IndexStyle' is set to 'subscript'. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and number of antennas, respectively.
- Cell array — The function returns this type of value when 'OutputResourceFormat' is set to 'cell'. If 'IndexStyle' is set to 'index', each cell is a column vector. If 'IndexStyle' is set to 'subscript', each cell is an *M*-by-3 matrix.

Depending on the value of 'IndexBase', the function returns either 1-based or 0-based indices.

Data Types: `uint32`

`info` — CSI-RS locations information
structure

CSI-RS locations information, returned as a structure containing these fields:

Fields	Description
`ResourceOrder`	Order of CSI-RS resources in terms of `CSIRSType` indices. `CSIRSType` is a property of the input `csirs` configuration object, specifying all CSI-RS resources for which the function generates the output.
`KBarLBar`	Frequency-domain and time-domain locations of the lowest resource elements corresponding to all code division multiplexing (CDM) groups
`CDMGroupIndices`	CDM group indices
`KPrime`	Frequency-domain indexing within a CDM group
`LPrime`	Time-domain indexing within a CDM group

Each field, apart from `ResourceOrder`, returns the information in the resource order specified by the `CSIRSType` property of the input `csirs` configuration object. These fields represent the frequency-domain and time-domain locations of the CSI-RS within a slot for each resource, as defined in TS 38.211 Table 7.4.1.5.3-1.

Version History

Introduced in R2019b

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrCSIRS` | `nrCSIRSMeasurements`

Objects
`nrCSIRSConfig` | `nrCarrierConfig`

nrCSIRSMeasurements

CSI-RS-based physical layer measurements

Syntax

```
meas = nrCSIRSMeasurements(carrier,csirs,grid)
```

Description

`meas = nrCSIRSMeasurements(carrier,csirs,grid)` returns physical layer measurements `meas` for a channel state information reference signal (CSI-RS), as defined in TS 38.215 Sections 5.1.2 and 5.1.4 [1]. The returned structure `meas` contains the reference signal received power (RSRP), received signal strength indicator (RSSI), and reference signal received quality (RSRQ). The input `carrier` specifies carrier configuration parameters for a specific OFDM numerology. The input `csirs` specifies CSI-RS resource configuration parameters for one or more zero-power (ZP) or non-zero-power (NZP) CSI-RS resources. The input `grid` specifies a carrier resource array.

Examples

Perform CSI-RS Measurements

Create a carrier configuration object with default properties.

```
carrier = nrCarrierConfig;
```

Create a CSI-RS configuration object. Set the row number to 1 and the frequency density to 3.

```
csirs = nrCSIRSConfig;
csirs.RowNumber = 1;
csirs.Density = 'three';
```

Generate CSI-RS symbols and indices for the specified configurations. Map the symbols to the slot grid.

```
ind = nrCSIRSIndices(carrier,csirs);
sym = nrCSIRS(carrier,csirs);
```

Initialize the carrier resource grid for one slot.

```
ports = max(csirs.NumCSIRSPorts);
txGrid = nrResourceGrid(carrier,ports);
txGrid(ind) = sym;
```

Perform OFDM modulation.

```
txWaveform = nrOFDMModulate(carrier,txGrid);
```

Apply two different levels of power scaling to the transmitted waveform.

```
EsdBm1 = -50;
rxWaveform1 = txWaveform * sqrt(10^((EsdBm1-30)/10));
```

```
EsdBm2 = -10;
rxWaveform2 = txWaveform * sqrt(10^((EsdBm2-30)/10));
```

Perform OFDM demodulation on the two received waveforms.

```
rxGrid1 = nrOFDMDemodulate(carrier,rxWaveform1);
rxGrid2 = nrOFDMDemodulate(carrier,rxWaveform2);
```

Perform CSI-RS measurements on the two received resource grids.

```
meas1 = nrCSIRSMeasurements(carrier,csirs,rxGrid1)
```

```
meas1 = struct with fields:
    RSRPPerAntenna: -50
    RSSIPerAntenna: -28.0688
    RSRQPerAntenna: -4.7712
```

```
meas2 = nrCSIRSMeasurements(carrier,csirs,rxGrid2)
```

```
meas2 = struct with fields:
    RSRPPerAntenna: -10.0000
    RSSIPerAntenna: 11.9312
    RSRQPerAntenna: -4.7712
```

Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function uses only these properties of the `nrCarrierConfig` object.

- `SubcarrierSpacing`
- `CyclicPrefix`
- `NSizeGrid`
- `NStartGrid`
- `NSlot`
- `NFrame`

`csirs` — CSI-RS resource configuration parameters

`nrCSIRSConfig` object

CSI-RS resource configuration parameters, specified as an `nrCSIRSConfig` object. This function uses only these properties of the `nrCSIRSConfig` object.

- `CSIRSType`
- `CSIRSPeriod`
- `RowNumber`
- `Density`
- `SymbolLocations`

- `SubcarrierLocations`
- `NumRB`
- `RBOffset`
- `NID`

`grid` — Carrier resource array
complex-valued array

Carrier resource array, specified as a complex-valued array of size *K*-by-*N*-by-*R*.

- *K* is the number of subcarriers.
- *N* is the number of OFDM symbols.
- *R* is the number of receive antennas.

Data Types: `single` | `double`
Complex Number Support: Yes

Output Arguments

`meas` — CSI-RS-based physical layer measurements
structure

CSI-RS-based physical layer measurements, returned as a structure containing these fields.

Field	Value	Description
`RSRPPerAntenna`	Real-valued matrix	Matrix of RSRP values in dBm relative to 1 milliwatt in 1 ohm
`RSSIPerAntenna`	Real-valued matrix	Matrix of RSSI values in dBm relative to 1 milliwatt in 1 ohm
`RSRQPerAntenna`	Real-valued matrix	Matrix of RSRQ values in dB

The rows of these matrices correspond to receive antennas. The columns correspond to the CSI-RS resources specified in the input `csirs`.

Data Types: `struct`

Version History

Introduced in R2022b

References

[1] 3GPP TS 38.215. "NR; Physical layer measurements." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrCSIRS` | `nrCSIRSIndices` | `nrSSBMeasurements`

Objects
`nrCarrierConfig` | `nrCSIRSConfig`

nrDCIDecode

Decode downlink control information (DCI)

Syntax

```
dcibits = nrDCIDecode(softbits,K,L)
[dcibits,mask] = nrDCIDecode(softbits,K,L)
[dcibits,mask] = nrDCIDecode(softbits,K,L,rnti)
```

Description

`dcibits = nrDCIDecode(softbits,K,L)` decodes the input `softbits` and returns the decoded DCI bits of length `K`. The function implements the inverse of the features specified in TS 38.212 Sections 7.3.4, 7.3.3, and 7.3.2 [1], such as rate recovery, polar decoding, and cyclic redundancy check (CRC) decoding. `L` specifies the list length used for polar decoding.

`[dcibits,mask] = nrDCIDecode(softbits,K,L)` also looks for a cyclic redundancy check (CRC) error in the DCI decoding. If `mask` is not equal to 0, either an error has occurred or the input CRC has been masked. When there are no CRC errors, `mask` is the actual value used for masking the CRC bits.

`[dcibits,mask] = nrDCIDecode(softbits,K,L,rnti)` specifies a radio network temporary identifier (RNTI). You can use this syntax when the value of `rnti` masks the CRC parity bits at the transmit end. When you specify `rnti` and there are no CRC errors, `mask` equals to 0.

Examples

Decode DCI Codeword

Create a random sequence of binary values corresponding to a DCI message of 32 bits. Encode the message based on the specified RNTI and rate-matched DCI codeword length. The RNTI masks the CRC parity bits.

```
K = 32;
rnti = 100;
E = 240;
dciBits = randi([0 1],K,1);
dcicw = nrDCIEncode(dciBits,rnti,E);
```

Decode the soft bits representing the DCI codeword `dcicw` by specifying the RNTI used for the CRC masking. Set the length of the polar decoding list to `8`.

```
L = 8;
[recBits,mask] = nrDCIDecode(1-2*dcicw,K,L,rnti)
```

```
recBits = 32x1 int8 column vector

   1
   1
   0
```

```
   1
   1
   0
   0
   1
   1
   1
      ⋮

mask = uint32
    0
```

Verify that the transmitted and received message bits are identical.

```
isequal(recBits,dciBits)
```

```
ans = logical
   1
```

Verify that the decoding is without error. As the decoding specified the RNTI used for masking, a mask value of 0 indicates no error.

```
mask
```

```
mask = uint32
    0
```

Input Arguments

`softbits` — Coded block of soft bits
column vector of real numbers

Coded block of soft bits, specified as a column vector of real numbers.

Data Types: `double` | `single`

`K` — Length of decoded output in bits
integer

Length of decoded output in bits, specified as an integer from 12 to 140.

Data Types: `double`

`L` — Length of polar decoding list
power of two

Length of polar decoding list, specified as a power of two.

Data Types: `double`

`rnti` — RNTI of UE
integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

Output Arguments

`dcibits` — Decoded DCI message bits
K-by-1 column vector of binary values

Decoded DCI message bits, returned as a K-by-1 column vector of binary values. The message bits were transmitted on a single physical downlink control channel (PDCCH).

Data Types: `int8`

`mask` — Result of CRC decoding
nonnegative integer

Result of CRC decoding, returned as a nonnegative integer less than or equal to 2^{16}–1. If `mask` is not equal to 0, either an error has occurred or the CRC has been masked. When there are no errors, `mask` is the actual value used for masking the CRC bits.

Data Types: `uint32`

Version History

Introduced in R2018b

Polar decoding metric update
Behavior changed in R2020a

In releases R2019b and before, polar decoding uses the exact form of the expression $\log(1 + e^x)$ for internal metric evaluation. Starting in release R2020a, because the exact form leads to numerical instability for high SNR ranges, polar decoding approximates $\log(1 + e^x)$ as 0 for $x < 0$ and as x for $x \geq 0$. This approximation affects the results of the `nrDCIDecode` function, resulting in a marginal degradation of the BLER performance in a link-level simulation.

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The input argument `L` must be a compile-time constant. Include `{coder.Constant(L)}` in the `-args` value of the `codegen` function. For more information, see `coder.Constant`.

See Also

Functions
`nrDCIEncode` | `nrPDCCH` | `nrPDCCHDecode`

nrDCIEncode

Encode downlink control information (DCI)

Syntax

`dcicw = nrDCIEncode(dcibits,rnti,E)`

Description

`dcicw = nrDCIEncode(dcibits,rnti,E)` encodes the input DCI bits and returns the rate-matched DCI codeword of length `E`. The function implements the features described in TS 38.212 Section 7.3.2, 7.3.3, and 7.3.4 [1], such as cyclic redundancy check (CRC) attachment, polar encoding, and rate matching. The CRC parity bits are masked with `rnti`, the radio network temporary identifier (RNTI) of the user equipment (UE).

Examples

Encode DCI Message Bits

Create a random sequence of binary values corresponding to a DCI message of 32 bits.

```
dcibits = randi([0 1],32,1);
```

Encode the message for the specified RNTI and rate-matched output length.

```
rnti = 100;
E = 240;
dcicw = nrDCIEncode(dcibits,rnti,E)
```

```
dcicw = 240×1

     0
     1
     0
     1
     0
     0
     0
     0
     0
     0
      ⋮
```

Input Arguments

`dcibits` — DCI message bits
column vector of binary values

DCI message bits, specified as a column vector of binary values. `dcibits` is the input to the DCI processing to be transmitted on a single physical downlink control channel (PDCCH).

Data Types: `double` | `int8`

`rnti` — RNTI of UE
integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

`E` — Length of rate-matched DCI codeword in bits
positive integer

Length of rate-matched DCI codeword in bits, specified as a positive integer. `E` must be in the range $K + 24 < E \leq 8192$, where K is the length of `dcibits`.

Data Types: `double`

Output Arguments

`dcicw` — Rate-matched DCI codeword
`E`-by-1 column vector of binary values

Rate-matched DCI codeword, returned as an `E`-by-1 column vector of binary values. `dcicw` inherits its data type from the input `dcibits`.

Data Types: `double` | `int8`

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.212. "NR; Multiplexing and channel coding." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrDCIDecode` | `nrPDCCH` | `nrPDCCHDecode`

nrDLSCHInfo

Get downlink shared channel (DL-SCH) information

Syntax

`info = nrDLSCHInfo(tBlkLen,targetCodeRate)`

Description

`info = nrDLSCHInfo(tBlkLen,targetCodeRate)` returns a structure containing DL-SCH information for an input transport block size `tBlkLen` and target code rate `targetCodeRate`. The DL-SCH information includes the cyclic redundancy check (CRC) attachment, code block segmentation (CBS), and channel coding.

Examples

Get DL-SCH Information

Show DL-SCH information before rate matching for an input transport block of length 8456 and target code rate 517/1024. The displayed DL-SCH information shows:

- The transport block has 312 <NULL> filler bits per code block.
- The number of bits per code block, after CBS, is 4576.
- The number of bits per code block, after low-density parity-check (LDPC) coding, is 13,728.

```
tBlkLen = 8456;
targetCodeRate = 517/1024;
nrDLSCHInfo(tBlkLen,targetCodeRate)
```

```
ans = struct with fields:
    CRC: '24A'
      L: 24
    BGN: 1
      C: 2
    Lcb: 24
      F: 312
     Zc: 208
      K: 4576
      N: 13728
```

Input Arguments

`tBlkLen` — Transport block size
nonnegative integer

Transport block size, specified as a nonnegative integer.

Data Types: `double`

targetCodeRate — Target code rate
real number

Target code rate, specified as a real number in the range (0, 1).

Data Types: double

Output Arguments

info — DL-SCH information
structure

DL-SCH information, returned as a structure containing these fields.

Fields	Values	Description
CRC	'16', '24A'	CRC polynomial selection
L	0, 16, 24	Number of CRC bits
BGN	1, 2	LDPC base graph selection
C	Positive integer	Number of code blocks
Lcb	0, 24	Number of parity bits per code block
F	Nonnegative integer	Number of <NULL> filler bits per code block
Zc	Positive integer	Lifting size selection
K	Nonnegative integer	Number of bits per code block after CBS
N	Nonnegative integer	Number of bits per code block after LDPC coding

Version History

Introduced in R2018b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
nrDLSCH | nrDLSCHDecoder

Functions
nrPDSCH | nrPDSCHDecode

nrEqualizeMMSE

Minimum mean-squared error (MMSE) equalization

Syntax

`[eqSym,csi] = nrEqualizeMMSE(rxSym,hest,nVar)`

Description

`[eqSym,csi] = nrEqualizeMMSE(rxSym,hest,nVar)` applies MMSE equalization to the extracted resource elements of a physical channel `rxSym` and returns the equalized symbols in `eqSym`. The equalization process uses the estimated channel information `hest` and the estimate of the received noise variance `nVar`. The function also returns the soft channel state information `csi`.

Examples

Perform MMSE Equalization for PBCH

Perform MMSE equalization on extracted resource elements of the physical broadcast channel (PBCH).

Create symbols and indices for a PBCH transmission.

```
ncellid = 146;
v = 0;
E = 864;
cw = randi([0 1],E,1);
pbchTxSym = nrPBCH(cw,ncellid,v);
pbchInd = nrPBCHIndices(ncellid);
```

Generate an empty resource array for one transmitting antenna. Populate the array with the PBCH symbols by using the generated PBCH indices.

```
carrier = nrCarrierConfig('NSizeGrid',20);
P = 1;
txGrid = nrResourceGrid(carrier,P);
txGrid(pbchInd) = pbchTxSym;
```

Perform OFDM modulation.

```
txWaveform = nrOFDMModulate(carrier,txGrid);
```

Create channel matrix and apply channel to transmitted waveform.

```
R = 4;
H = dftmtx(max([P R]));
H = H(1:P,1:R);
H = H / norm(H);
rxWaveform = txWaveform * H;
```

Create channel estimate.

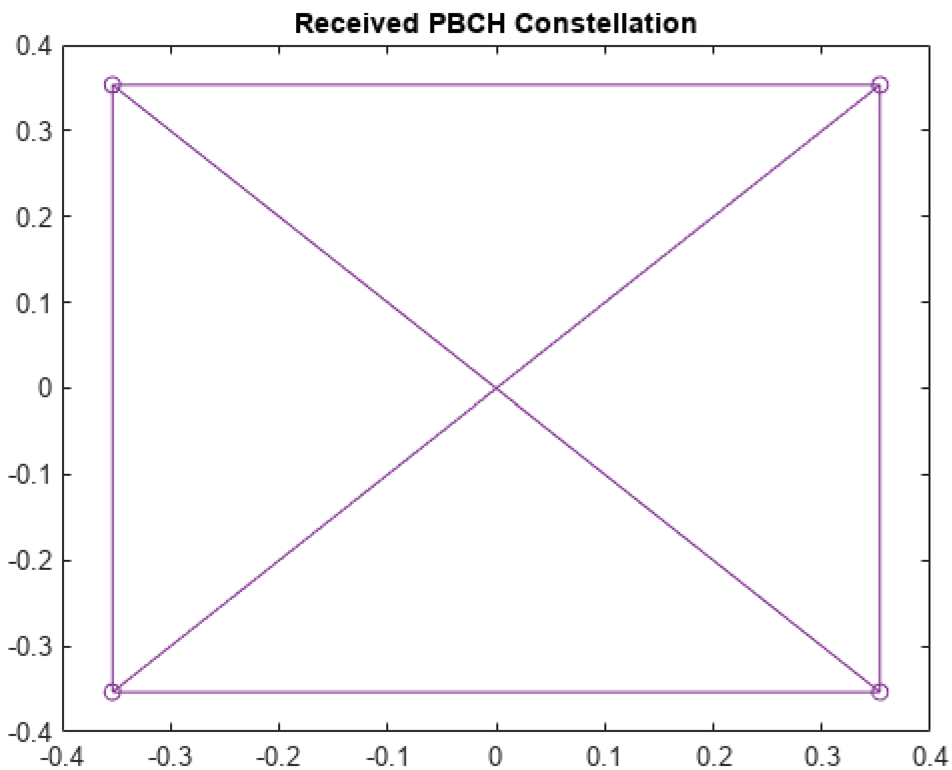
```
hEstGrid = repmat(permute(H.',[3 4 1 2]),[240 4]);
nEst = 0.1;
```

Perform OFDM demodulation.

```
rxGrid = nrOFDMDemodulate(carrier,rxWaveform);
```

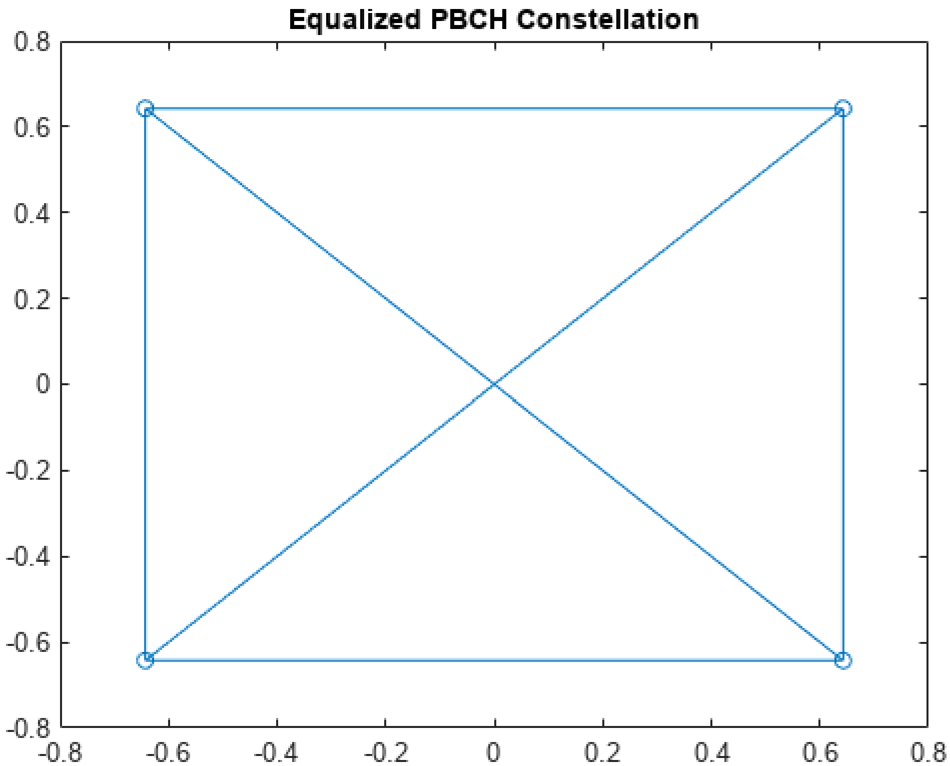
To prepare for PBCH decoding, use `nrExtractResources` to extract symbols from received and channel estimate grids. Plot the received PBCH constellation.

```
[pbchRxSym,pbchHestSym] = nrExtractResources(pbchInd,rxGrid,hEstGrid);
figure;
plot(pbchRxSym,'o:');
title('Received PBCH Constellation');
```



Decode the PBCH with the extracted resource elements. Plot the equalized PBCH constellation.

```
[pbchEqSym,csi] = nrEqualizeMMSE(pbchRxSym,pbchHestSym,nEst);
pbchBits = nrPBCHDecode(pbchEqSym,ncellid,v);
figure;
plot(pbchEqSym,'o:');
title('Equalized PBCH Constellation');
```



Input Arguments

`rxSym` — Extracted resource elements

2-D numeric matrix

Extracted resource elements of a physical channel, specified as an *NRE*-by-*R* numeric matrix. *NRE* is the number of resource elements extracted from each *K*-by-*L* plane of the received grid. *K* is the number of subcarriers and *L* is the number of OFDM symbols. *R* is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

`hest` — Estimated channel information

3-D numeric array

Estimated channel information, specified as an *NRE*-by-*R*-by-*P* numeric array. *NRE* is the number of resource elements extracted from each *K*-by-*L* plane of the received grid. *K* is the number of subcarriers and *L* is the number of OFDM symbols. *R* is the number of receive antennas. *P* is the number of layers.

Data Types: `double`
Complex Number Support: Yes

`nVar` — Estimated noise variance

real nonnegative scalar

Estimated noise variance, specified as a real nonnegative scalar.

Data Types: `double`

Output Arguments

`eqSym` — Equalized symbols
2-D numeric matrix

Equalized symbols, returned as an *NRE*-by-*P* numeric matrix. *NRE* is the number of resource elements extracted from each *K*-by-*L* plane of the received grid. *K* is the number of subcarriers and *L* is the number of OFDM symbols. *P* is the number of layers.

Data Types: `double`
Complex Number Support: Yes

`csi` — Soft channel state information
2-D numeric matrix

Soft channel state information, returned as an *NRE*-by-*P* numeric matrix. *NRE* is the number of resource elements extracted from each *K*-by-*L* plane of the received grid. *K* is the number of subcarriers and *L* is the number of OFDM symbols. *P* is the number of layers.

Data Types: `double`
Complex Number Support: Yes

Version History
Introduced in R2018b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrExtractResources` | `nrPerfectChannelEstimate` | `nrPerfectTimingEstimate`

nrExtractResources

Extract resource elements from resource array

Syntax

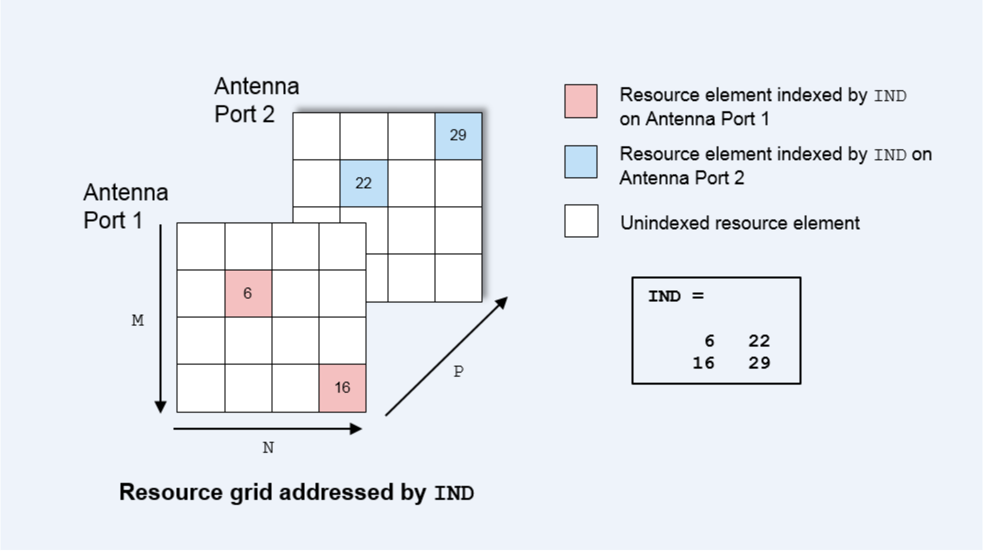
```
re = nrExtractResources(ind,grid)
[re,reind] = nrExtractResources(ind,grid)
[re1,...,reN,reind1,...,reindN] = nrExtractResources(
ind,grid1,grid2,....,gridN)
[ ___ ] = nrExtractResources( ___ ,Name,Value)
```

Description

`re = nrExtractResources(ind,grid)` returns the resource elements from the resource array `grid` using resource element indices `ind`. The function can extract resource elements even if `grid` has a dimensionality which is different than the dimensionality of the indices `ind`. In this syntax, the specified indices are 1-based using linear indexing form.

Typically, channel or signal specific functions generate resource element indices to map the channel or signal symbols to a resource grid. The indices address resource elements in an *M*-by-*N*-by-*P* array. *M* is the number of subcarriers, *N* is the number of OFDM symbols, and *P* is the number of antenna ports.

For example, the following diagram highlights resource elements of a 4-by-4-by-2 resource array. The resource element indices are in 1-based linear indexing form. The number of the antenna ports is two (*P* = 2).



`[re,reind] = nrExtractResources(ind,grid)` also returns `reind`, the indices of the extracted resource elements `re` within the resource array `grid`. The array `reind` is the same size as the extracted resource elements `re`.

`[re1,...,reN,reind1,...,reindN] = nrExtractResources(ind,grid1,grid2,....,gridN)` extracts resource elements from multiple resource arrays using the resource element indices `ind`.

`[ ___ ] = nrExtractResources( ___ ,Name,Value)` specifies optional name-value pair arguments in addition to any of the input argument sets in previous syntaxes. Use these name-value pair arguments to specify the format of the input indices and the extraction method. Unspecified arguments take default values.

Examples

Extract PBCH Symbols and Channel Estimates for Decoding

Extract physical broadcast channel (PBCH) symbols from a received grid and associated channel estimates in preparation for decoding a beamformed PBCH.

PBCH Coding and Beamforming

Create a random sequence of binary values corresponding to a BCH codeword. The length of the codeword is 864, as specified in TS 38.212 Section 7.1.5. Using the codeword, create symbols and indices for a PBCH transmission. Specify the physical layer cell identity number.

```
E = 864;
cw = randi([0 1],E,1);
ncellid = 17;
v = 0;
pbchTxSym = nrPBCH(cw,ncellid,v);
pbchInd = nrPBCHIndices(ncellid);
```

Use `nrExtractResources` to create indices for the two transmit antennas of a beamformed PBCH. Use these indices to map the beamformed PBCH into the transmitter resource array.

```
carrier = nrCarrierConfig('NSizeGrid',20);
P = 2;
txGrid = nrResourceGrid(carrier,P);
F = [1 1i];
[~,bfInd] = nrExtractResources(pbchInd,txGrid);
txGrid(bfInd) = pbchTxSym*F;
```

OFDM modulate the PBCH symbols mapped into the transmitter resource array.

```
txWaveform = nrOFDMModulate(carrier,txGrid);
```

PBCH Transmission and Decoding

Create and apply a channel matrix to the waveform. Receive the transmitted waveforms.

```
R = 3;
H = dftmtx(max([P R]));
H = H(1:P,1:R);
H = H/norm(H);
rxWaveform = txWaveform*H;
```

Create channel estimates including beamforming.

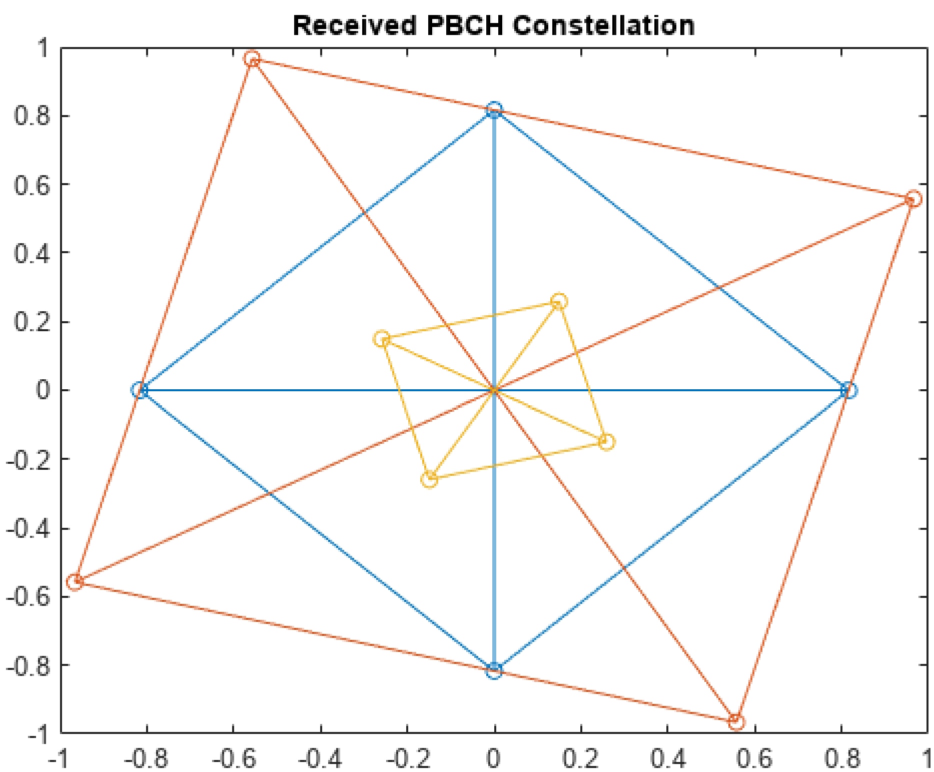
```
hEstGrid = repmat(permute(H.'*F.',[3 4 1 2]),[240 4]);
nEst = 0;
```

Demodulate the received waveform using orthogonal frequency division multiplexing (OFDM).

```
rxGrid = nrOFDMDemodulate(carrier,rxWaveform);
```

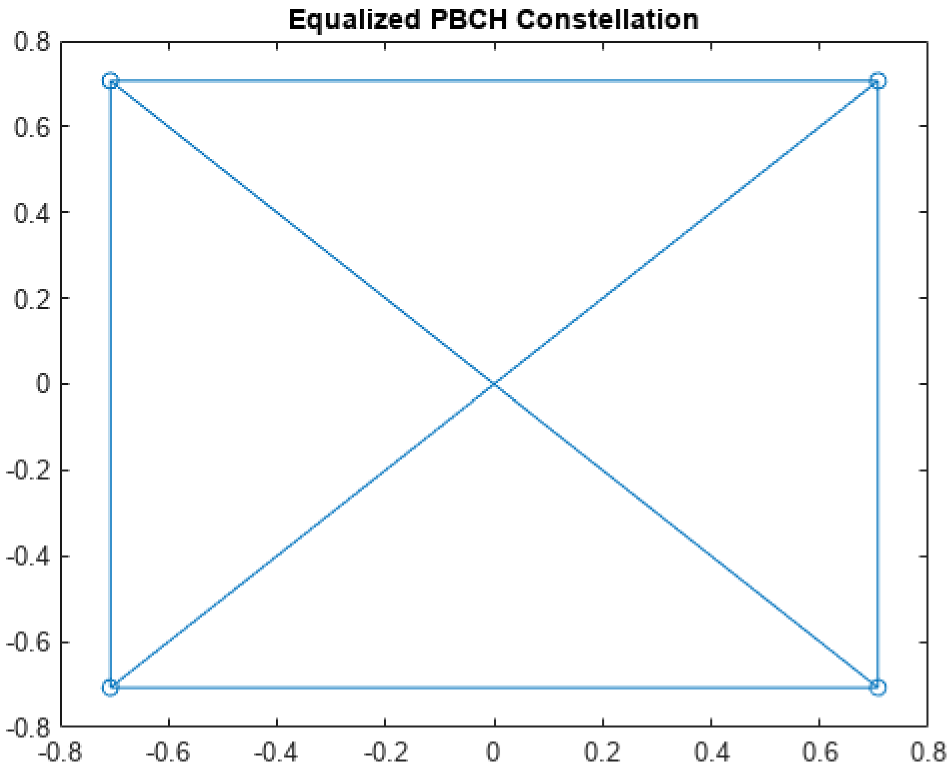
In preparation for PBCH decoding, extract symbols from the received grid and the channel estimate grid.

```
[pbchRxSym,pbchHestSym] = nrExtractResources(pbchInd,rxGrid,hEstGrid);
figure;
plot(pbchRxSym,'o:');
title('Received PBCH Constellation');
```



Equalize the symbols by performing MMSE equalization on the extracted resources. Plot the results.

```
pbchEqSym = nrEqualizeMMSE(pbchRxSym,pbchHestSym,nEst);
figure;
plot(pbchEqSym,'o:');
title('Equalized PBCH Constellation');
```



Retrieve soft bits by performing PBCH decoding on the equalized symbols.

```
pbchBits = nrPBCHDecode(pbchEqSym,ncellid,v)
```

```
pbchBits = 864×1
10^10 ×

   -2.0000
   -2.0000
    2.0000
   -2.0000
   -2.0000
    2.0000
    2.0000
   -2.0000
   -2.0000
   -2.0000
      ⋮
```

Input Arguments

ind — Resource element indices

matrix

Resource element indices, specified as a matrix.

- If `'IndexStyle'` is `'index'`, each column of the matrix contains linear indices for the corresponding antenna.
- If `'IndexStyle'` is `'subscript'`, `ind` is a three-column matrix. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and antennas, respectively.

The function assumes that the indices are 1-based, unless you specify otherwise with the `'IndexBase'` argument.

Data Types: `double`

`grid` — Resource array
3-D numeric array (default) | 4-D numeric array

Resource array from which to extract resource elements, specified as one of these values:

- 3-D numeric array of size *M*-by-*N*-by-*R* that corresponds to a received grid — *M* is the number of subcarriers, *N* is the number of OFDM symbols, and *R* is the number of receive antennas. The grid is created after OFDM demodulation.
- A 4-D numeric array of size *M*-by-*N*-by-*R*-by-*P* that corresponds to a channel estimation grid — *P* is the number of antenna ports. The grid is created after channel estimation.

Data Types: `double`

Name-Value Pair Arguments

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

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

Example:
`nrExtractResources(ind,grid,'ExtractionMethod','direct','IndexBase','0based')` specifies direct extraction method with 0-based indexing.

`IndexStyle` — RE indexing form
`'index'` (default) | `'subscript'`

RE indexing form, specified as one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

`IndexBase` — RE indexing base
`'1based'` (default) | `'0based'`

RE indexing base, specified as one of these values:

- `'1based'` — The index counting starts from 1.
- `'0based'` — The index counting starts from 0.

Data Types: `char` | `string`

`ExtractionMethod` — Resource element extraction method
`'allplanes'` (default) | `'direct'`

Resource element extraction method, specified as the comma-separated pair consisting of `'ExtractionMethod'` and `'allplanes'` or `'direct'`.

- `'allplanes'` — The function extracts resource elements from each *M*-by-*N* plane within `grid`. The function uses indices that address unique subcarrier and symbol locations over all planes of the indexed resource array. See “All-Planes Extraction Method (Default)” on page 1-94.
- `'direct'` — The function extracts resource elements from each *M*-by-*N* plane (for a 3-D `grid`) or *M*-by-*N*-by-*R* array (for a 4-D `grid`). The function uses indices that address the corresponding plane of the indexed resource array directly. See “Direct Extraction Method” on page 1-96.

Data Types: `string` | `char`

Output Arguments

`re` — Extracted resource elements
column vector | numeric array

Extracted resource elements, returned as a column vector, or a numeric array.

When `'ExtractionMethod'` is set to `'allplanes'`, the size of `re` is N_{RE}-by-R-by-P, where:

- N_{RE} is the number of resource elements extracted from each *M*-by-*N* plane of `grid`.
- *R* number of receive antennas.
- *P* is the number of planes.

When `'ExtractionMethod'` is set to `'direct'`, the size of `re` depends on the number of indices addressing each plane of the indexed resource grid.

- If the number of indices addressing each plane is the same, then `re` is of size N_{RE}-by-R-by-P.
- If the number of indices addressing each plane is different, then `re` is a column vector containing all extracted resource elements.

For more details on the resource extraction methods, see “Algorithms” on page 1-94.

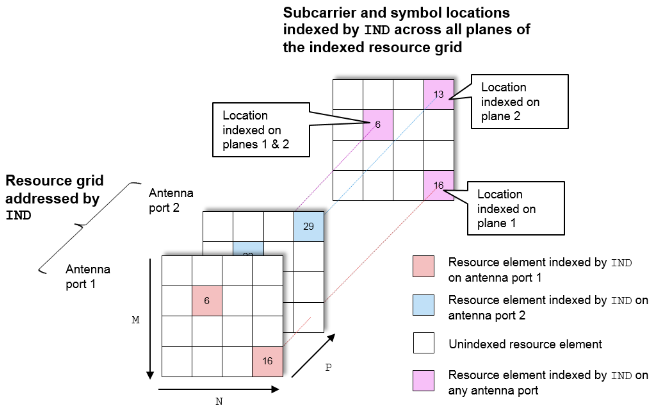
`reind` — Indices of extracted resource elements
numeric array

Indices of extracted resource elements within `grid`, returned as numeric array. `reind` is the same size as the extracted resource elements array `re`. The `reind` output inherits the indexing style and index base from `ind`.

Algorithms

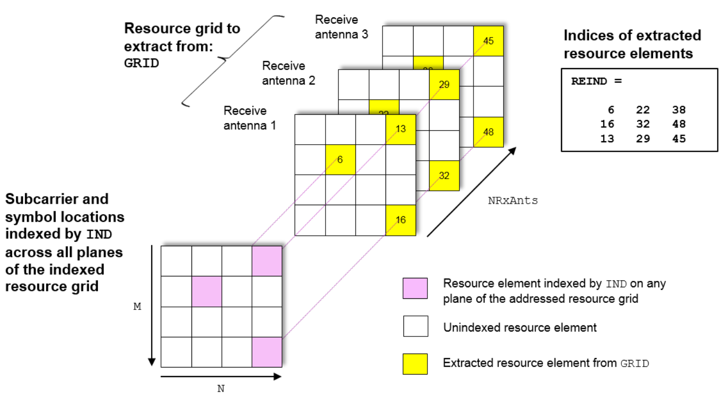
All-Planes Extraction Method (Default)

To use this method, set `'ExtractionMethod'` to `'allplanes'`. This method extracts resource elements from each *M*-by-*N* plane within `grid`. The indices address unique subcarrier and symbol locations over all the planes of the indexed resource array. The diagram highlights the indices used to extract resource elements from a resource grid with $P = 2$.



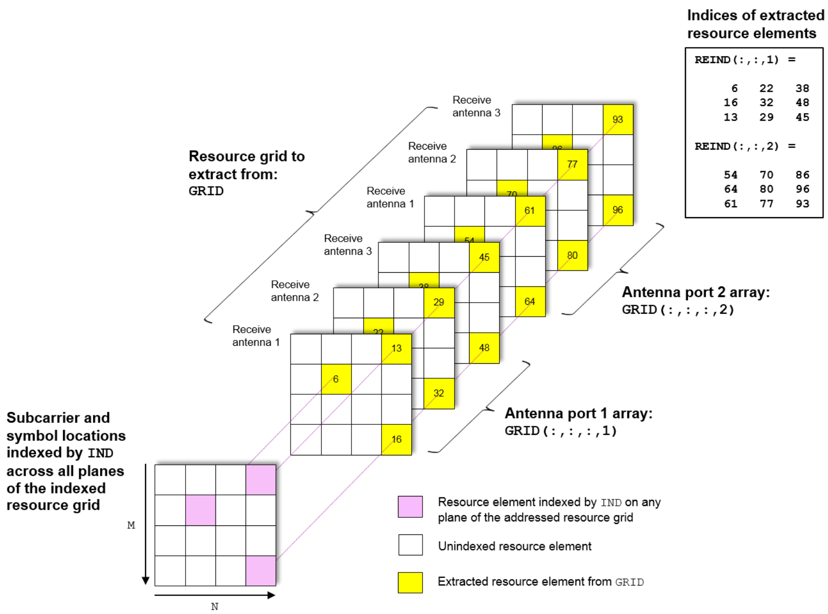
Extraction Process for a 3-D Received Grid

The following diagrams illustrate the resource element extraction from a 3-D received grid, where the number of receive antennas R = 3. Resource elements are extracted from the grid at the symbol and subcarrier locations.



Extraction Process for a 4-D Channel Estimate Grid

The following diagram shows the extraction process for a 4-D channel estimate grid. The number of receive antennas R = 3 and the number of antenna ports P = 2. The 4-D resource grid consists of P M-by-N-by-R arrays, each associated with an antenna port. Resource elements are extracted from all planes within these arrays.



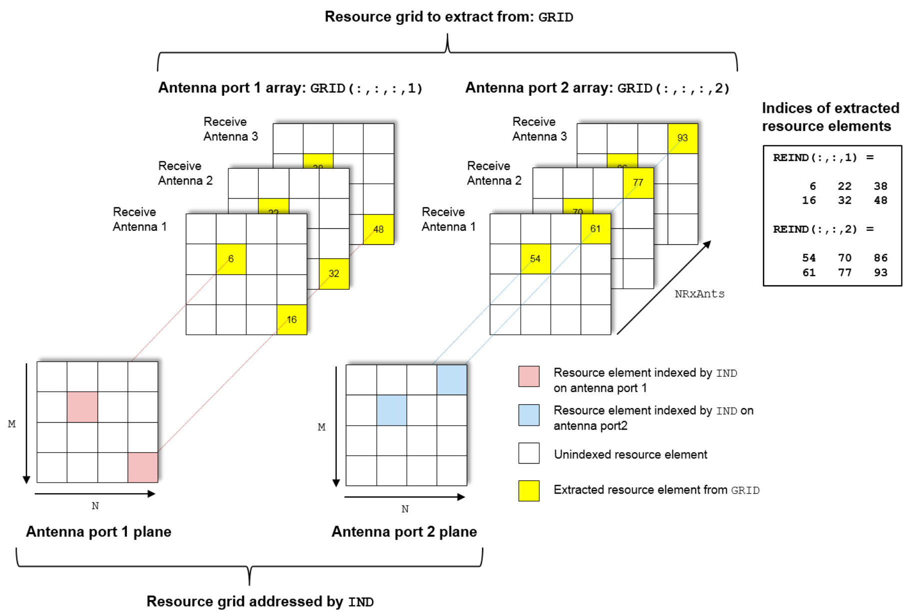
Direct Extraction Method

To use this method, set `'ExtractionMethod'` to `'direct'`. This method extracts resource elements from `grid` assuming that the third and fourth dimensions of the `grid` represent the same property as the planes of the indexed resource array such as antenna ports, layers, transmit antennas. Therefore the function extracts only the resource elements relevant to each plane of the indexed resource grid.

- For a 3-D `grid`, the direct method extracts elements from each *M*-by-*N* plane of `grid` using indices addressing the same plane of the indexed resource array. This method is the same as the standard MATLAB® operation `re = grid(ind)`. Therefore, `reind = ind`.
- For a 4-D `grid`, the direct method extracts elements from each *M*-by-*N*-by-*R* array of `grid` by using indices addressing the same plane of the indexed resource array. The function assumes that the property represented by the planes of the indexed resource array is the same as the fourth dimension of `grid`.

Extraction Process for a 4-D Channel Estimate Grid

The following diagram shows the extraction process for a 4-D channel estimate grid. The number of receive antennas $R = 3$ and the number of antenna ports $P = 2$. The 4-D resource grid consists of P number of M-by-N-by-R arrays, each associated with an antenna port. The indices corresponding to each individual antenna port in the indexed resource array are used to extract resource elements from each of these arrays.



Version History

Introduced in R2018b

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrEqualizeMMSE` | `nrPSSIndices` | `nrSSSIndices` | `nrPBCHIndices` | `nrPBCHDMRSIndices`

nrLayerDemap

Layer demapping onto scrambled and modulated codewords

Syntax

```
out = nrLayerDemap(in)
```

Description

`out = nrLayerDemap(in)` returns one or two codewords obtained from layer demapping the received layered symbols specified by `in`. The function determines the number of codewords based on the number of layers, as specified in TS 38.211 Table 7.3.1.3-1 [1].

Examples

Layer Mapping and Demapping of Single Codeword

Map a single codeword of length 20 to four transmission layers.

```
codeword = ones(20,1);
nLayers = 4;
layeredOut = nrLayerMap(codeword,nLayers);
```

Recover the original codeword using layer demapping.

```
out = nrLayerDemap(layeredOut);
```

Check for errors.

```
isequal(codeword,out{1})
```

```
ans = logical
   1
```

Input Arguments

in — Layered modulation symbols
complex matrix

Layered modulation symbols, specified as a complex matrix of size *M*-by-*nLayers*. *M* is the number of modulation symbols in a transmission layer. *nLayers* is the number of transmission layers in the range 1 to 8.

Data Types: `double`

Output Arguments

out — Modulation symbols in codewords
cell array of one or two complex column vectors

Modulation symbols in codewords, returned as a cell array of one or two complex column vectors. This output inherits the data type of the input `in`. One vector corresponds to one codeword. The number of codewords is based on the number of layers. The function determines the number of codewords using TS 38.211 Table 7.3.1.3-1.

Data Types: `cell`

Version History
Introduced in R2018b

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrLayerMap` | `nrSymbolDemodulate` | `nrPDSCHDecode`

nrLayerMap

Layer mapping of modulated and scrambled codewords

Syntax

`out = nrLayerMap(in,nLayers)`

Description

`out = nrLayerMap(in,nLayers)` performs layer mapping of one or two codewords, specified by `in`, based on the number of transmission layers `nLayers`. The transmission layers in the output are formed by multiplexing the modulation symbols from either one or two codewords. The function implements the transpose of the overall layer mapping specified in TS 38.211 Section 6.3.1.3 and Section 7.3.1.3 [1]. In other words, the symbols in a layer lie in columns rather than rows.

Examples

Map One Codeword to Four Transmission Layers

Perform layer mapping of one codeword of length 40, using 4 transmission layers.

```
out = nrLayerMap(ones(40,1),4);
sizeOut = size(out)
```

```
sizeOut = 1×2

    10     4
```

Map Two Codewords to Five Transmission Layers

Perform layer mapping of two codewords of length 20 and 30 respectively, using 5 transmission layers.

```
out = nrLayerMap({ones(20,1),ones(30,1)},5);
sizeOut = size(out)
```

```
sizeOut = 1×2

    10     5
```

Input Arguments

in — Modulation symbols in codewords
complex column vector | cell array of one or two complex column vectors

Modulation symbols in codewords, specified as one of these values:

- Complex column vector — Use this value to specify one codeword.
- Cell array of one or two complex column vectors — Use this value to specify one or two codewords.

Data Types: `double`

`nLayers` — Number of transmission layers
integer from 1 to 8

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

Data Types: `double`

Output Arguments

`out` — Layered modulation symbols
complex matrix

Layered modulation symbols, returned as a complex matrix of size *M*-by-`nLayers`. *M* is the number of modulation symbols (rows) in a transmission layer (column). The output `out` inherits the data type of the input `in`.

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrLayerDemap` | `nrSymbolModulate` | `nrPDSCH`

nrLDPCDecode

Low-density parity-check (LDPC) decoding

Syntax

```
[out,actNumIter,finalParityChecks] = nrLDPCDecode(in,bgn,maxNumIter)
[out,actNumIter,finalParityChecks] = nrLDPCDecode( ___ ,Name,Value)
```

Description

`[out,actNumIter,finalParityChecks] = nrLDPCDecode(in,bgn,maxNumIter)` returns the LDPC-decoded output matrix `out` for the input data matrix `in`, base graph number `bgn`, and maximum number of decoding iterations `maxNumIter`. The function also returns the actual number of iterations `actNumIter` and the final parity checks per codeword `finalParityChecks`.

The decoder uses the sum-product message-passing algorithm. The data bits must be LDPC-encoded as defined in TS 38.212 Section 5.3.2 [1].

`[out,actNumIter,finalParityChecks] = nrLDPCDecode( ___ ,Name,Value)` specifies optional name-value pair arguments, in addition to the input arguments in the previous syntax.

Examples

Decode LDPC Codeword

Create transmit data consisting of two code block segments of length 2560 and 36 filler bits at the end.

```
C = 2;
K = 2560;
F = 36;
txcbs = ones(K-F,C);
fillers = -1*ones(F,C);
txcbs = [txcbs;fillers];
```

Generate LDPC codeword for the transmit data. Use base graph number two.

```
bgn = 2;
txcodedcbs = nrLDPCEncode(txcbs,bgn);
```

Convert transmit data to soft bits. Fillers in the transmit data do not have log likelihood ratio (LLR) soft bits.

```
rxcodedcbs = double(1-2*txcodedcbs);
FillerIndices = find(txcodedcbs(:,1) == -1);
rxcodedcbs(FillerIndices,:) = 0;
```

Decode the encoded codeword with a maximum of 25 iterations.

```
[rxcbs,actualniters] = nrLDPCDecode(rxcodedcbs,bgn,25);
```

Replace filler bits with zero in transmit data and compare the results of encoding and decoding.

```
txcbs(end-F+1:end,:) = 0;
isequal(rxcbs,txcbs)
```

```
ans = logical
   1
```

```
actualniters
```

```
actualniters = 1×2

     1     1
```

Input Arguments

in — Rate recovered soft bits for input code block segments
real matrix

Rate recovered soft bits for input code block segments, specified as a real matrix. The number of columns in `in` is equal to the number of scheduled code block segments. The number of rows in `in` is equal to the length of the codeword, with some systematic bits punctured.

Data Types: `double` | `single`

bgn — Base graph number
`1` | `2`

Base graph number, specified as `1` or `2`. The value selects one of the two base graphs defined in TS 38.212 Section 5.3.2 [1].

Data Types: `double`

`maxNumIter` — Maximum number of decoding iterations
positive integer scalar

Maximum number of decoding iterations, specified as a positive integer scalar. The decoding is terminated when all parity checks are satisfied, or after `maxNumIter` number of iterations.

Data Types: `double`

Name-Value Pair Arguments

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

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

Example: `[out,actNumIter,finalParityChecks] = nrLDPCDecode(in,bgn,maxNumIter,'DecisionType','hard')`

`OutputFormat` — Output format
`'info'` (default) | `'whole'`

Output format, specified as the comma-separated pair consisting of `'OutputFormat'` and one of these values:

- `'info'` — The number of rows in `out` is equal to the length of the information bits.
- `'whole'` — The number of rows in `out` is equal to the length of the codeword.

Data Types: `char` | `string`

`DecisionType` — Decision method used for decoding
`'hard'` (default) | `'soft'`

Decision method used for decoding, specified as the comma-separated pair consisting of `'DecisionType'` and one of these values:

- `'hard'` — The data type of `out` is `int8`.
- `'soft'` — The output `out` contains log-likelihood ratios of the same data type as `in`.

Data Types: `char` | `string`

`Algorithm` — LDPC decoding algorithm
`'Belief propagation'` (default) | `'Layered belief propagation'` | `'Normalized min-sum'` | `'Offset min-sum'`

LDPC decoding algorithm, specified as the comma-separated pair consisting of `'Algorithm'` and one of these values:

- `'Belief propagation'` — Use this option to specify the belief-passing or message-passing algorithm. For more information, see "Belief Propagation Decoding" on page 1-106.
- `'Layered belief propagation'` — Use this option to specify the layered belief-passing algorithm, which is suitable for quasi-cyclic parity-check matrices (PCMs). For more information, see "Layered Belief Propagation Decoding" on page 1-107.
- `'Normalized min-sum'` — Use this option to specify the layered belief propagation algorithm with normalized min-sum approximation. For more information, see "Normalized Min-Sum Decoding" on page 1-108.
- `'Offset min-sum'` — Use this option to specify the layered belief propagation algorithm with offset min-sum approximation. For more information, see "Offset Min-Sum Decoding" on page 1-108.

Note When you specify the value of this name-value pair argument as `'Normalized min-sum'` or `'Offset min-sum'`, the function clips the input (log-likelihood ratio) LLR values to the [-1e10 1e10] range before decoding.

Data Types: `char` | `string`

`ScalingFactor` — Scaling factor for normalized min-sum decoding
`0.75` (default) | real scalar in the range (0, 1]

Scaling factor for normalized min-sum decoding, specified as the comma-separated pair consisting of `'ScalingFactor'` and a real scalar in the range (0, 1].

Dependencies

To enable this name-value pair argument, set the `'Algorithm'` name-value pair argument to `'Normalized min-sum'`.

Data Types: `double`

`Offset` — Offset for offset min-sum decoding
`0.5` (default) | nonnegative finite real scalar

Offset for offset min-sum decoding, specified as the comma-separated pair consisting of `'Offset'` and a nonnegative finite real scalar.

Dependencies

To enable this name-value pair argument, set the `'Algorithm'` name-value pair argument to `'Offset min-sum'`.

Data Types: `double`

`Termination` — Decoding termination criteria
`'early'` (default) | `'max'`

Decoding termination criteria, specified as the comma-separated pair consisting of `'Termination'` and one of these values:

- `'early'` — The decoding terminates when all parity checks are satisfied or after `maxNumIter` number of iterations.
- `'max'` — The decoding terminates after `maxNumIter` number of iterations.

Data Types: `char` | `string`

Output Arguments

`out` — Decoded LDPC codeword
real matrix

Decoded LDPC codeword or information bits, returned as a real matrix. The number of columns in `out` is equal to the number of scheduled code block segments. The number of rows in `out` depends on the name-value pair argument `'OutputFormat'`. The data type of `out` depends on the name-value pair argument `'DecisionType'`.

Data Types: `single` | `double` | `int8`

`actNumIter` — Actual number of iterations
row vector of positive integers

Actual number of iterations, returned as a row vector of positive integers. The length of `actNumIter` is equal to the number of columns in `in`. The ith element in `actNumIter` corresponds to the actual number of iterations executed for the ith column of `in`.

Data Types: `double`

`finalParityChecks` — Final parity checks
matrix

Final parity checks, returned as a matrix. The number of rows in `finalParityChecks` is equal to the number of parity-check bits in an LDPC codeword. The `i`th column in `finalParityChecks` corresponds to the final parity checks for the `i`th codeword.

Data Types: `double`

Algorithms

The `nrLDPCDecode` function supports these four LDPC decoding algorithms.

Belief Propagation Decoding

The implementation of the belief propagation algorithm is based on the decoding algorithm presented in [2]. For transmitted LDPC-encoded codeword, c, where $c = (c_0, c_1, \ldots, c_{n-1})$, the input to the LDPC decoder is the log-likelihood ratio (LLR) value $L(c_i) = \log\left(\frac{\Pr(c_i = 0 \mid \text{channel output for } c_i)}{\Pr(c_i = 1 \mid \text{channel output for } c_i)}\right)$.

In each iteration, the key components of the algorithm are updated based on these equations:

$$L(r_{ji}) = 2\,\text{atanh}\left(\prod_{i' \in V_j \backslash i} \tanh\left(\frac{1}{2}L(q_{i'j})\right)\right),$$

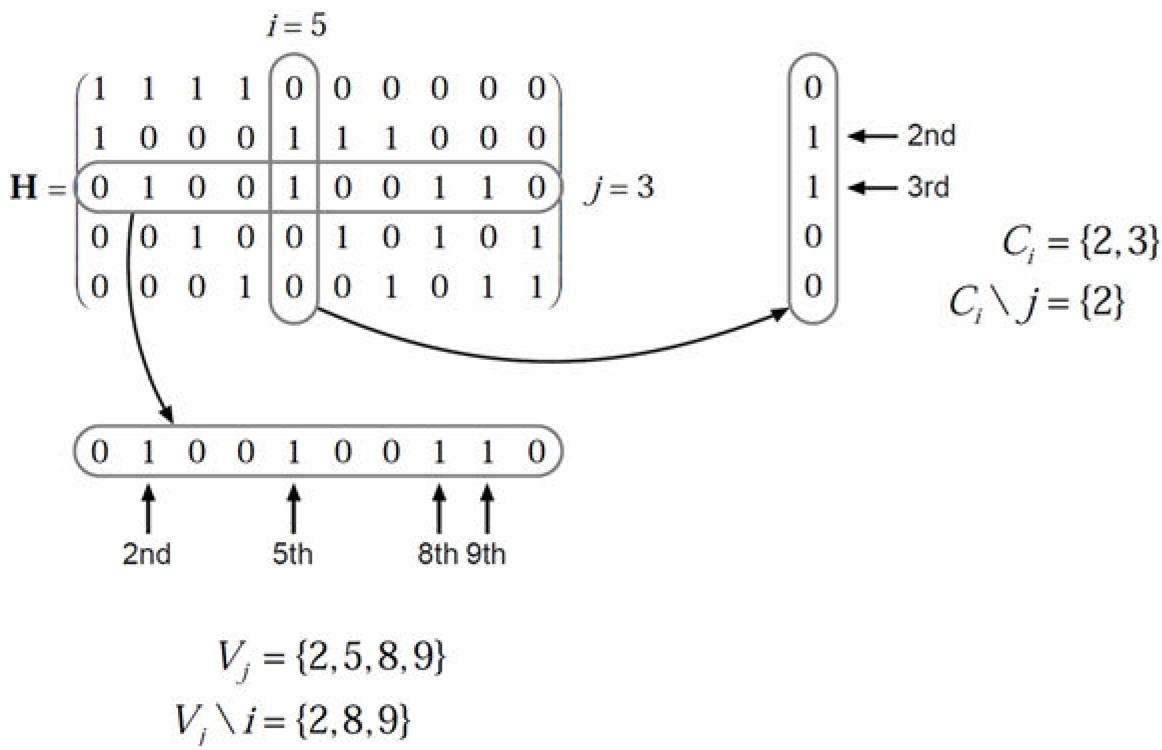
$L(q_{ij}) = L(c_i) + \sum_{j' \in C_i \backslash j} L(r_{j'i})$, initialized as $L(q_{ij}) = L(c_i)$ before the first iteration, and

$$L(Q_i) = L(c_i) + \sum_{j' \in C_i} L(r_{j'i}).$$

At the end of each iteration, $L(Q_i)$ is an updated estimate of the LLR value for the transmitted bit c_i. The value $L(Q_i)$ is the soft-decision output for c_i. If $L(Q_i) < 0$, the hard-decision output for c_i is 1. Otherwise, the output is 0.

Index sets $C_i \backslash j$ and $V_j \backslash i$ are based on the parity-check matrix (PCM). Index sets C_i and V_j correspond to all nonzero elements in column i and row j of the PCM, respectively.

This figure highlights the computation of these index sets in a given PCM for $i = 5$ and $j = 3$.



To avoid infinite numbers in the algorithm equations, atanh(1) and atanh(-1) are set to 19.07 and –19.07, respectively. Due to finite precision, MATLAB returns 1 for tanh(19.07) and –1 for tanh(–19.07).

When the name-value pair argument `'Termination'` is set to `'max'`, the decoding terminates after `maxNumIter` number of iterations. When `'Termination'` is set to `'early'`, the decoding terminates when all parity checks are satisfied ($\mathbf{Hc}^{T} = 0$) or after `maxNumIter` number of iterations.

Layered Belief Propagation Decoding

The implementation of the layered belief propagation algorithm is based on the decoding algorithm presented in [3], Section II.A. The decoding loop iterates over subsets of rows (layers) of the PCM. For each row, m, in a layer and each bit index, j, the implementation updates the key components of the algorithm based on these equations:

(1) $L(q_{mj}) = L(q_j) - R_{mj}$,

(2) $A_{mj} = \sum_{\substack{n \in N(m) \\ n \neq j}} \psi(L(q_{mn}))$,

(3) $s_{mj} = \prod_{\substack{n \in N(m) \\ n \neq j}} \mathrm{sign}(L(q_{mn}))$,

(4) $R_{mj} = -s_{mj}\psi(A_{mj})$, and

(5) $L(q_j) = L(q_{mj}) + R_{mj}$.

For each layer, the decoding equation (5) works on the combined input obtained from the current LLR inputs $L(q_{mj})$ and the previous layer updates R_{mj}.

Because only a subset of the nodes is updated in a layer, the layered belief propagation algorithm is faster compared to the belief propagation algorithm. To achieve the same error rate as attained with belief propagation decoding, use half the number of decoding iterations when using the layered belief propagation algorithm.

Normalized Min-Sum Decoding

The implementation of the normalized min-sum decoding algorithm follows the layered belief propagation algorithm with equation (2) replaced by

$$A_{mj} = \min_{\substack{n \in N(m) \\ n \neq j}} (|L(q_{mn})| \cdot \alpha),$$

where α is in the range (0, 1] and is the scaling factor specified by `ScalingFactor`. This equation is an adaptation of equation (4) presented in [4].

Offset Min-Sum Decoding

The implementation of the offset min-sum decoding algorithm follows the layered belief propagation algorithm with equation (2) replaced by

$$A_{mj} = \max\left(\min_{\substack{n \in N(m) \\ n \neq j}} (|L(q_{mn})| - \beta),\ 0\right),$$

where $\beta \geq 0$ and is the offset specified by `Offset`. This equation is an adaptation of equation (5) presented in [4].

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] Gallager, Robert G. *Low-Density Parity-Check Codes*, Cambridge, MA, MIT Press, 1963.

[3] Hocevar, D.E. "A reduced complexity decoder architecture via layered decoding of LDPC codes." In *IEEE Workshop on Signal Processing Systems, 2004. SIPS 2004*. doi: 10.1109/SIPS.2004.1363033

[4] Chen, Jinghu, R.M. Tanner, C. Jones, and Yan Li. "Improved min-sum decoding algorithms for irregular LDPC codes." In *Proceedings. International Symposium on Information Theory, 2005. ISIT 2005*. doi: 10.1109/ISIT.2005.1523374

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying soft decision type, include `{coder.Constant('DecisionType'),coder.Constant('soft')}` in the `-args` value of the `codegen` function. For more information, see `coder.Constant`.

See Also

Functions

`nrLDPCEncode` | `nrRateRecoverLDPC` | `nrDLSCHInfo` | `nrCodeBlockDesegmentLDPC` | `nrCRCDecode`

nrLDPCEncode

Low-density parity-check (LDPC) encoding

Syntax

`out = nrLDPCEncode(in,bgn)`

Description

`out = nrLDPCEncode(in,bgn)` returns the LDPC-encoded output matrix for the input data matrix `in` and base graph number `bgn`, as specified in TS 38.212 Section 5.3.2 [1]. If applicable, the function replaces each filler bit represented by `-1` in the input by `0`. After encoding, the function replaces each filler bit again by `-1`. The encoding includes puncturing of some of the systematic information bits.

Examples

Generate LDPC Codeword

Create input data for encoding consisting of two code block segments of length 2560 and 36 filler bits at the end.

```
C = 2;
K = 2560;
F = 36;
cbs = ones(K-F,C);
fillers = -1*ones(F,C);
cbs = [cbs;fillers];
```

Generate LDPC codeword for the two code block segments. Use base graph number two.

```
bgn = 2;
codedcbs = nrLDPCEncode(cbs,bgn);
size(codedcbs)
```

```
ans = 1×2

       12800           2
```

Input Arguments

in — Code block segments before encoding
matrix | column vector

Code block segments before encoding, specified as a matrix or a column vector. The number of columns in `in` is equal to the number of scheduled code block segments in the transport block. The number of rows in `in` is equal to the length of the code block segment, including the filler bits, if any.

Note Filler bits are represented by `-1` and are treated as `0` when performing encoding.

Data Types: `double` | `int8`

bgn — Base graph number
1 | 2

Base graph number, specified as `1` or `2`. The values correspond to the two base graphs defined in TS 38.212 Section 5.3.2 [1]

Data Types: `double`

Output Arguments

`out` — Encoded LDPC codeword
matrix

Encoded LDPC codeword output, returned as a matrix. The number of columns in `out` is equal to the number of scheduled code block segments in the transport block. The number of rows in `out` is equal to the length of the codeword. Each codeword punctures some of the systematic bits and can contain filler bits.

Data Types: `double` | `int8`

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.212. "NR; Multiplexing and channel coding." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrCodeBlockSegmentLDPC` | `nrCRCEncode` | `nrDLSCHInfo` | `nrLDPCDecode` | `nrRateMatchLDPC`

nrLowPAPRS

Generate low peak-to-average power ratio (low-PAPR) sequence

Syntax

```
seq = nrLowPAPRS(u,v,alpha,m)
seq = nrLowPAPRS(u,cinit,m)
seq = nrLowPAPRS( ___ ,'OutputDataType',datatype)
```

Description

`seq = nrLowPAPRS(u,v,alpha,m)` generates type 1 low peak to average power ratio (low-PAPR) sequences, `seq` of length `m`, as defined in TS 38.211, Section 5.2.2 [1]. `u` specifies the sequence group numbers. `v` specifies the base sequence numbers (0 or 1) within the sequence group. The function applies phase rotations to the base sequence corresponding to the cyclic shifts specified by `alpha`. When you specify `u`, `v`, and `alpha` input arguments in vector form, the function generates multiple low-PAPR type 1 sequences.

Type 1 low-PAPR sequences are used for the generation of the demodulation reference signal (DM-RS) for the physical uplink shared channel (PUSCH), sounding reference signals (SRS), physical uplink control channel (PUCCH) formats 0 and 1 modulation symbols, and DM-RS for the PUCCH format 1, 3 and 4.

`seq = nrLowPAPRS(u,cinit,m)` generates type 2 low-PAPR sequences, `seq`, as defined in TS 38.211, Section 5.2.3 [1]. `cinit` specifies the sequence initializations. When you specify `u` and `cinit` input arguments in vector form, the function generates multiple type 2 low-PAPR sequences.

Type 2 low-PAPR sequences are used for the generation of the DM-RS for the PUSCH and PUCCH formats 3 and 4, when configured with Release 16 DM-RS transform precoding.

`seq = nrLowPAPRS( ___ ,'OutputDataType',datatype)` specifies the data type of the low-PAPR sequence in addition to the input arguments in any of the previous syntaxes.

Examples

Generate Low-PAPR Sequence

Generate a low-PAPR sequence of length 36 for sequence group number 9, base sequence number 0, and the specified cyclic shift.

```
u = 9;
v = 0;
alpha = 2*pi;
m = 36;
seq = nrLowPAPRS(u,v,alpha,m)
```

```
seq = 36×1 complex

   1.0000 + 0.0000i
```

```
  -0.4404 - 0.8978i
   0.9795 + 0.2013i
   0.9190 + 0.3944i
   0.1514 - 0.9885i
   0.5290 + 0.8486i
   0.1514 + 0.9885i
   0.9795 - 0.2013i
  -0.7588 + 0.6514i
  -0.9949 + 0.1012i
      ⋮
```

Generate Multiple Low-PAPR Sequences

Generate low-PAPR sequences of `single` data type and length 36 for sequence group number 9, base sequence number 0, and the specified cyclic shifts. Specifying more than one cyclic shifts as a vector results in the generation of multiple low-PAPR sequences.

```
u = 9;
v = 0;
alpha = [pi/2,pi];
m = 36;
seq = nrLowPAPRS(u,v,alpha,m,'OutputDataType','single')
```

```
seq = 36x2 single matrix

   1.0000 + 0.0000i   1.0000 + 0.0000i
   0.8978 - 0.4404i   0.4404 + 0.8978i
  -0.9795 - 0.2013i   0.9795 + 0.2013i
   0.3944 - 0.9190i  -0.9190 - 0.3944i
   0.1514 - 0.9885i   0.1514 - 0.9885i
  -0.8486 + 0.5290i  -0.5290 - 0.8486i
  -0.1514 - 0.9885i   0.1514 + 0.9885i
  -0.2013 - 0.9795i  -0.9795 + 0.2013i
  -0.7588 + 0.6514i  -0.7588 + 0.6514i
  -0.1012 - 0.9949i   0.9949 - 0.1012i
      ⋮
```

Generate Type 2 Low-PAPR Sequence

Generate a type 2 low-PAPR sequence of length 36 for sequence group number 11. Specify the sequence initializations as [18, 209].

```
u = 11;
cinit = [18, 209];
m = 36;
seq = nrLowPAPRS(u,cinit,m)
```

```
seq = 36×2 complex

  -0.4714 - 0.4714i   0.0000 + 0.9428i
  -1.8745 + 0.9602i   0.0489 + 0.1049i
```

```
  1.4169 - 1.1518i   0.8459 - 0.1127i
 -0.2357 - 0.2357i   1.2016 - 0.0231i
 -1.0127 - 1.0239i  -0.1278 - 0.0469i
 -0.6247 + 0.2351i  -0.4183 + 0.2929i
 -0.8797 - 0.0632i  -1.0291 + 0.9659i
 -0.6351 + 0.2740i   0.9863 - 0.7222i
 -0.7476 + 0.4365i   0.2328 + 0.5454i
  0.0000 + 0.0000i  -0.4714 - 0.4714i
     ⋮
```

Input Arguments

u — Sequence group numbers
vector of integers from 0 to 29

Sequence group numbers, specified as a vector of integers from 0 to 29. When the sequence length `m` is less than 30 for type 2 low-PAPR sequence, the `u` argument is used to get the base sequence.

Data Types: `double`

v — Base sequence numbers
vector of binary values 0 and 1

Base sequence numbers within a sequence group, specified as a vector of binary values 0 and 1. When the low-PAPR sequence length `m` is less than 72, the sequence group has only one base sequence. In this case, only base sequence number `0` applies. When the low-PAPR sequence length `m` is greater than or equal to 72, the sequence group has two base sequences. In this case, both base sequence numbers `0` and `1` apply.

Data Types: `double`

`alpha` — Cyclic shifts
vector of nonnegative values

Cyclic shifts, specified as a vector of nonnegative values. The function applies different phase rotations to the base sequence corresponding to the specified cyclic shifts.

Data Types: `double`

`cinit` — Sequence initializations
vector of nonnegative integers

Pseudorandom binary sequence (PRBS) initializations, specified as a vector of nonnegative integers. When the sequence length `m` is greater than or equal to 30, the `cinit` argument is used to get the base sequence for type 2 low-PAPR sequence.

Data Types: `double`

m — Low-PAPR sequence length
nonnegative integer

Low-PAPR sequence length, specified as a nonnegative integer. When the value of `m` is less than 30, the sequence length must be 6, 12, 18, or 24. When `m` is `0`, `seq` is an empty vector.

Data Types: `double`

datatype — Data type of output symbols
'double' (default) | 'single'

Data type of the output symbols, specified as 'double' or 'single'.

Data Types: char | string

Output Arguments

seq — Low-PAPR sequence
M-by-*NumAlpha*-by-*NumU*-by-*NumV* matrix | *M*-by-*NumCinit*-by-*NumU* matrix

Low-PAPR sequence, returned as one of these values.

- *M*-by-*NumAlpha*-by-*NumU*-by-*NumV* matrix — The function returns this type of value for type 1 low-PAPR sequence. *M* is the sequence length length provided by input m. *NumAlpha* is the number of cyclic shifts provided in the input alpha. *NumU* is the number of sequence groups provided in the input u. *NumV* is the number of base sequence numbers within the sequence group provided in the input v.
- *M*-by-*NumCinit*-by-*NumU* matrix — The function returns this type of value for type 2 low-PAPR sequence. *NumCinit* is the number of PRBS initializations provided in the input cinit.

When input m is 0, the output argument seq is an empty vector.

Data Types: single | double
Complex Number Support: Yes

Version History

Introduced in R2019a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation (Release 16)." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The datatype input argument must be compile-time constant. For example, when specifying 'single' as the output data type, include {coder.Constant('OutputDataType'),coder.Constant('single')} in the -args value of the codegen function. For more information, see the coder.Constant class.

See Also

Functions
nrPUCCH0 | nrPUCCH1 | nrPRBS | nrSRS | nrPUCCHDMRS | nrPUSCHDMRS | nrPUCCH

nrMACBSR

Generate BSR MAC CE

Syntax

```
[lcid,bsr] = nrMACBSR(lcgBufferSize)
[lcid,bsr] = nrMACBSR(lcgBufferSize,lcgPriority,paddingBytes)
```

Description

`[lcid,bsr] = nrMACBSR(lcgBufferSize)` generates a regular or periodic buffer status report (BSR) medium access control (MAC) control element (CE). The input `lcgBufferSize` specifies the buffer size values for logical channel groups (LCGs). The function generates the BSR for the nonzero values specified in `lcgBufferSize`. The function also returns `lcid`, which indicates the logical channel ID corresponding to the generated BSR.

`[lcid,bsr] = nrMACBSR(lcgBufferSize,lcgPriority,paddingBytes)` generates a padding BSR MAC CE. `lcgPriority` specifies the priority of the LCGs. `paddingBytes` specifies the number of available padding bytes for the function to use to generate the padded BSR MAC CE.

Examples

Generate Regular or Periodic BSR MAC CE

Generate a regular or periodic BSR MAC CE for a single LCG.

```
lcgbuffersize = [0 0 0 9000];        % LCG 3 has data for transmission
[lcid,bsr] = nrMACBSR(lcgbuffersize)
```

```
lcid = 61

bsr = 118
```

LCID 61 represents a short BSR format because only one LCG has data to transmit.

Now generate regular or periodic BSR MAC CE when all eight LCGs have data to transmit.

LCID 62 represents a long BSR format because more than one LCG has data to transmit.

```
lcgbuffersize = [20000 700 624 3030 125 1020 3500 2100];
[lcid,bsr] = nrMACBSR(lcgbuffersize)
```

```
lcid = 62

bsr = 9×1

   255
   121
    68
    66
    91
```

```
    41
    74
    94
    86
```

Generate Padding BSR MAC CE

Generate a padding BSR MAC CE for multiple LCGs.

Specify the LCG buffer size, LCG priority, and padding bytes.

```
lcgBufferSize = [1200 3450 7000 4500 5250 6000 2100 9000];
lcgPriority = [4 15 2 10 7 3 2 9];
paddingBytes = 5;
```

Generate a padding BSR MAC CE.

```
[lcid,bsr] = nrMACBSR(lcgBufferSize,lcgPriority,paddingBytes)
```

```
lcid = 60
```

```
bsr = 3×1

   255
   105
    86
```

Input Arguments

`lcgBufferSize` — Buffer size of LCGs
vector

Buffer size of the LCGs, specified as a vector with a maximum of eight elements.

In this vector, specify the LCGs in increasing order of their corresponding LCG IDs. The LCG IDs are indexed from 0 to 7.

The size of the vector is limited by the highest LCG ID having nonzero data available for transmission.

Data Types: `double`

`lcgPriority` — Priority of LCGs
vector with elements in the range [1, 16]

Priority of the LCGs, specified as a vector with elements in the range [1, 16]. In this vector, list the priority in increasing order of LCG ID.

A lower priority value indicates an LCG with higher priority, and a higher priority value indicates an LCG with lower priority. For a long truncated BSR format, the LCGs with equal priority are reported in the order of their increasing LCG ID.

An LCG priority value is derived from the highest priority logical channel mapped to it.

Data Types: `double`

`paddingBytes` — Number of available padding bytes
integer greater than 1

Number of available padding bytes, specified as an integer greater than 1. You must specify these bytes to generate a padding BSR MAC CE.

The function might not use all of the padding bytes, depending on the value of input `lcgBufferSize`.

The `paddingbytes` value limits the number of LCG buffer size values that the function can report in a long truncated BSR format.

Data Types: `double`

Output Arguments

`bsr` — BSR MAC CE
nonnegative integer | column vector of octets in decimal format

BSR MAC CE, returned as one of these options.

- Nonnegative integer — For short and short truncated BSR formats.
- Column vector of octets in decimal format — For long and long truncated BSR formats. The length of the vector is determined by the number of reported LCGs.

BSR is of one of these three types. For more details, see 3GPP TS 38.321 Section 5.4.5 [1].

- Regular — When new data arrives in the uplink (UL) buffer and has higher priority than the data that is already waiting in the buffer of the user equipment (UE); or when none of the other LCGs have data to transmit.
- Periodic — When sent with a predefined periodicity which is set by the network, at the expiry of the periodic BSR timer.
- Padding — When the number of padding bits in a data message is larger than the size of BSR. In this case, padding bit space can be used to send the BSR.

Each of these three types of BSR can have one of these four formats.

- Short — When the UE triggers a regular or periodic BSR, and only a single LCG has data to transmit, then the BSR is reported in the short format.
- Short truncated — When the UE triggers a padding BSR and multiple LCGs have data to transmit, and the number of padding bits is equal to the length of short BSR and sub header, then the BSR is reported in the short truncated format.
- Long — When the UE triggers a regular or periodic BSR and multiple LCGs have data to transmit, then the BSR is reported in the long format.
- Long truncated — When the UE triggers a padding BSR and multiple LCGs have data to transmit, and the padding bit length of the MAC protocol data unit (PDU) is insufficient to transmit the long BSR format, then the UE uses the long truncated format to report the BSR.

The short truncated and long truncated formats are supported only with padding BSR type. The short and long formats support all three types of BSR CE (regular, periodic, or padding).

For more details, see 3GPP TS 38.321 Section 6.1.3.1 [1].

Data Types: double

lcid — Logical channel ID
59 | 60 | 61 | 62

Logical channel ID, returned as one of these values, which corresponds to the generated BSR format.

- 59 — Short truncated BSR format
- 60 — Long truncated BSR format
- 61 — Short BSR format
- 62 — Long BSR format

Data Types: double

Version History

Introduced in R2021b

References

[1] 3GPP TS 38.321. "NR; Medium Access Control (MAC) protocol specification." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
nrMACBSRDecode | nrPUSCH

Objects
nrULSCH | nrPUSCHConfig

nrMACBSRDecode

Decode BSR MAC CE

Syntax

```
[lcgID,lcgBufferRange] = nrMACBSRDecode(lcid,bsr)
```

Description

`[lcgID,lcgBufferRange] = nrMACBSRDecode(lcid,bsr)` decodes the buffer status report (BSR) medium access control (MAC) control element (CE). The input, `lcid`, specifies the logical channel ID (LCID) corresponding to the specified input `bsr`. The function returns the decoded logical channel group (LCG) IDs, `lcgID`, reported in the BSR, as well as the buffer size range, `lcgBufferRange`, for the values specified in the BSR.

Examples

Generate and Decode BSR MAC CE

Specify the LCG buffer size, the LCG priority, and the number of padding bytes available for generating a padding BSR.

```
lcgBufferSize = [234 63 943 1002 4325 994 666 48]; % In bytes
lcgPriority = [4 3 2 6 7 2 8 1];
paddingBytes = 2;                                  % In bytes
```

Generate a padding BSR MAC CE for multiple LCGs by using the `nrMACBSR` function. This function generates a short truncated BSR with the buffer information from LCG index 7.

```
[lcid,bsr] = nrMACBSR(lcgBufferSize,lcgPriority,paddingBytes)
```

```
lcid = 59
```

```
bsr = 230
```

Decode the LCG ID and the buffer size range of the BSR MAC CE by using the generated LCID and BSR values. The LCG ID output specifies the buffer information reported in LCG index 7.

```
[lcgID,lcgBufferRange] = nrMACBSRDecode(lcid,bsr)
```

```
lcgID = 7
```

```
lcgBufferRange = 1×2

    39    53
```

Decode Short BSR MAC CE

Specify the LCID and BSR MAC CE of a short BSR.

```
lcid = 61;
bsr = 67;
```

Decode the LCG ID and the buffer size range.

```
[lcgID,lcgBufferRange] = nrMACBSRDecode(lcid,bsr)
```

```
lcgID = 2

lcgBufferRange = 1×2

    15    20
```

Decode Long and Long Truncated BSR MAC CE

Specify the LCID and BSR MAC CE of a long BSR.

```
lcid = 62;
bsr = [36; 75; 253];
```

Decode the LCG ID and the buffer size range of a long BSR. The function returns the indices of the LCG whose buffer status was reported and contains data for transmission. The LCG buffer range specifies the corresponding buffer size range of the LCG IDs. In a long BSR, the function maps the LCG ID and LCG buffer ranges on a one-to-one basis.

```
[lcgID,lcgBufferRange] = nrMACBSRDecode(lcid,bsr)
```

```
lcgID = 2×1

     2
     5

lcgBufferRange = 2×2

        1052        1119
    76380420    81338368
```

Now, specify the LCID and BSR MAC CE of a long truncated BSR.

```
lcid = 60;
bsr = [145; 51; 26];
```

Decode the LCG ID and the buffer size range of a long truncated BSR. The function returns the indices of the LCG containing data for transmission. The LCG buffer range specifies the buffer size range of the LCG IDs whose buffer status was reported. In a long truncated BSR, the function does not map the LCG ID and LCG buffer ranges on a one-to-one basis.

```
[lcgID,lcgBufferRange] = nrMACBSRDecode(lcid,bsr)
```

```
lcgID = 3×1

     0
     4
     7


lcgBufferRange = 2×2

   234   248
    50    52
```

Now, for a long truncated BSR, specify the BSR containing one octet. This octet specifies the LCG bitmap.

```
lcid = 60;
bsr = 17;
```

Decode the LCG ID and the buffer size range.

```
[lcgID,lcgBufferRange] = nrMACBSRDecode(lcid,bsr)
```

```
lcgID = 2×1

     0
     4


lcgBufferRange =

  0x2 empty double matrix
```

Input Arguments

`lcid` — Logical channel ID of BSR

integer in the range [59, 62]

Logical channel ID of the BSR, specified as an integer in the range [59, 62]. The integers in this range correspond to BSR MAC formats.

`lcid` Value	BSR Format
59	Short truncated BSR
60	Long truncated BSR
61	Short BSR
62	Long BSR

Data Types: `double`

`bsr` — BSR MAC CE

nonnegative integer in the range [0, 255] | column vector of octets in the range [0, 255]

BSR MAC CE, specified as one of these values.

`bsr` Value	BSR Format Supported
Nonnegative integer in the range [0, 255]	Use this value for short and short truncated BSR formats.
Column vector of octets in the range [0, 255]	Use this value for long and long truncated BSR formats. • For long BSR format, set the length of the column vector equal to the number of reported LCGs + 1. The maximum length of the column vector is 9. • For long truncated BSR format, set the length of the column vector to less than or equal to the number of reported LCGs. The maximum length of the column vector is 8.

The function uses this value to calculate the number of reported LCGs.

Data Types: `double`

Output Arguments

`lcgID` — Logical channel group ID
integer | column vector

Logical channel group ID, returned as an integer or a column vector with length equal to the number of LCGs reported.

- For a short BSR, short truncated BSR, and long BSR, this output returns the index of the logical channels whose buffer status was reported by `bsr`.
- For a long truncated BSR, this output returns the index of the logical channels that contains data for transmission.

Data Types: `double`

`lcgBufferRange` — Buffer size range of LCGs
row vector | matrix

Buffer size range of the LCGs, returned as a row vector or a matrix. The first and second column of this output specify the lower and upper range of the buffer size, respectively.

BSR Format	`lcgBufferRange` Value Description
Short or short truncated BSR	• Return a row vector of length 2. • The function maps this output to the corresponding `lcgID` output. • For a buffer size index of 31, the upper range of this output is set to the largest value of the 32-bit signed integer type for the purpose of calculation.
Long BSR	• Return a matrix of size *N*-by-2, where *N* denotes the number of LCGs containing data for transmission. • The function maps each row of this output to the corresponding row in the `lcgID` output. • For a buffer size index of 254, the upper range is set to the largest value of the 32-bit signed integer type for the purpose of calculation.

BSR Format	lcgBufferRange Value Description
Long truncated BSR	• Return a matrix of size M-by-2, where M ($M<N$) denotes the number of buffer size values reported in BSR. • The function does not map this output and the lcgID output on a one-to-one basis. During long truncated BSR generation, the LCG buffer information of one or more lcgID is lost in the truncation. • For a buffer size index of 254, the upper range is set to the largest value of the 32-bit signed integer type for the purpose of calculation.

For more information about the buffer size range of the LCGs, see Section 6.1.3.1 of 3GPP TS 38.321 [1].

Data Types: double

Version History

Introduced in R2022a

References

[1] 3GPP TS 38.321. “NR; Medium Access Control (MAC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

nrMACBSR | nrPUSCH | nrMACSubPDU

Objects

nrULSCH | nrPUSCHConfig

nrMACPDUDecode

Decode MAC PDU

Syntax

```
[msgIndexList,payloadList] = nrMACPDUDecode(macPDU,linkDir)
[msgIndexList,payloadList] = nrMACPDUDecode(macPDU,linkDir,softErrorFlag)
```

Description

`[msgIndexList,payloadList] = nrMACPDUDecode(macPDU,linkDir)` decodes the medium access control (MAC) protocol data unit (PDU) `macPDU` as defined in 3GPP TS 38.321 section 6.1.2 release 16 [1]. `linkDir` specifies the transmission direction of the MAC PDU. The function returns the message indices `msgIndexList`, where each message index represents either the logical channel ID (LCID) or extended logical channel ID (eLCID) field of a MAC subPDU. The function also returns the payloads of the corresponding MAC subPDUs `payloadList`.

`[msgIndexList,payloadList] = nrMACPDUDecode(macPDU,linkDir,softErrorFlag)` also specifies how to handle errors in decoding.

Examples

Decode MAC PDU in Uplink Direction

Decode an uplink MAC PDU that contains five MAC subPDUs in the uplink direction with LCID and eLCID values 20, 2487, 55, 54, and 63 (2487 represents an eLCID and the rest are LCIDs).

The five subPDUs carry these payloads, in order:

- Two service data units (SDUs), one of 344 bytes and the other of 166 bytes
- Fixed-size control element (CE) of 0 bytes
- Variable-size CE of 4 bytes
- Padding of 0 bytes

Generate the MAC PDU that contains the five MAC subPDUs.

```
macPDU = [84; 1; 88; ones(344,1); ...
          33; 8; 119; 166; ones(166,1); ...
          55; ...
          54; 4; ones(4,1); ...
          63];
```

Set the transmission direction of the link.

```
linkDir = 1; % Uplink
```

Decode the MAC PDU.

```
[msgIndexList,payloadList] = nrMACPDUDecode(macPDU,linkDir)
```

```
msgIndexList = 5×1

          20
        2487
          55
          54
          63


payloadList=5×1 cell array
    {344x1 double}
    {166x1 double}
    {  0x0 double}
    {  4x1 double}
    {  0x0 double}
```

Decode MAC PDU in Downlink Direction

Decode a downlink MAC PDU that contains a padding MAC subPDU in the downlink direction (with an LCID value of 63). The MAC subPDU carries a padding payload of 7 bytes.

```
macPDU = [63; 0; 0; 0; 0; 0; 0; 0];
linkDir = 0;                  % Downlink direction
```

Decode the MAC PDU.

```
[msgIndexList,payloadList] = nrMACPDUDecode(macPDU,linkDir)
```

```
msgIndexList = 63

payloadList = 1x1 cell array
    {7x1 double}
```

Input Arguments

macPDU — MAC PDU

vector of octets in decimal format

MAC PDU, specified as a vector of octets in decimal format.

Data Types: `double`

`linkDir` — Transmission direction of MAC PDU

`0` | `1`

Transmission direction of the MAC PDU, specified as one of these values.

- `0` — Use this value to set the transmission direction to downlink.
- `1` — Use this value to set the transmission direction to uplink.

Data Types: `double`

softErrorFlag — Soft error flag
false or 0 (default) | true or 1

Soft error flag, specified as one of these numeric or logical values. This value determines the error handling in decoding.

- 0 (false) — When the function encounters a corrupted MAC PDU, the execution stops with a corresponding error message.
- 1 (true) — When the function encounters a corrupted MAC PDU, the execution does not stop, and the function returns empty values.

Data Types: logical

Output Arguments

msgIndexList — Message indices
vector of integers

Message indices, returned as a vector of integers. Each message index represents the LCID or eLCID field of a MAC subPDU.

The index values returned in msgIndexList correspond to these tables specified in 3GPP TS 38.321.

- For a downlink shared channel (DL-SCH) subPDU, the msgIndex corresponds to the index value specified in 3GPP TS 38.321 table 6.2.1-1, 6.2.1-1a, or 6.2.1-1b [1].
- For an uplink shared channel (UL-SCH) subPDU, the msgIndex corresponds to the index value specified in 3GPP TS 38.321 table 6.2.1-2, 6.2.1-2a, or 6.2.1-2b [1].

Data Types: double

payloadList — Payload of subPDUs
cell array

Payload of subPDUs, returned as a cell array. Each payload in the payload list is either MAC SDU, MAC CE, or padding payload.

Data Types: cell

Version History

Introduced in R2022b

References

[1] 3GPP TS 38.321. "NR; Medium Access Control (MAC) protocol specification." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also
`nrMACSubPDU`

nrMACSubPDU

Generate NR MAC subPDU

Syntax

```
macSubPDU = nrMACSubPDU(linkDir,msgIndex,payload)
macSubPDU = nrMACSubPDU(paddingLength)
```

Description

`macSubPDU = nrMACSubPDU(linkDir,msgIndex,payload)` generates a medium access control (MAC) sub protocol data unit (subPDU), as defined in 3GPP TS 38.321 Section 6.1.2 [1].

This syntax is applicable for MAC subPDUs with MAC service data unit (SDU) or MAC control element (CE) as payload.

`macSubPDU = nrMACSubPDU(paddingLength)` generates a padding MAC subPDU for the uplink or downlink direction.

For this syntax, the logical channel ID (LCID) value is set to `63` in the subheader.

Examples

Generate MAC subPDU for SDU Payload

Generate a MAC subPDU for service data unit (SDU) payload of 6 bytes in uplink direction. Specify the LCID index value as 25.

```
linkDir = 1;
msgIndex = 25;
payload = [64;21;202;238;10;169]; % 6 bytes
```

Generate the MAC subPDU.

```
macSubPDU = nrMACSubPDU(linkDir,msgIndex,payload)
```

```
macSubPDU = 8×1

    25
     6
    64
    21
   202
   238
    10
   169
```

Generate MAC subPDU for CE Payload

Generate a MAC subPDU for control element (CE) payload of 3 bytes in downlink direction. Specify the eLCID index value as 310.

```
linkDir = 0;
msgIndex = 310;
payload = [22;138;244]; % 3 bytes
```

Generate the MAC subPDU.

```
macSubPDU = nrMACSubPDU(linkDir,msgIndex,payload)
```

```
macSubPDU = 6×1

    34
   246
     3
    22
   138
   244
```

Generate MAC subPDU with Padding Payload

Specify the required MAC padding size.

```
paddingLength = 4; % In bytes
```

Generate the MAC subPDU.

```
macSubPDU = nrMACSubPDU(paddingLength)
```

```
macSubPDU = 4×1

    63
     0
     0
     0
```

Generate MAC subPDU with MAC BSR payload

Generate a MAC subPDU in uplink direction. Use the `nrMACBSR` function to generate a MAC buffer status report (BSR) payload.

```
linkDir = 1;
lcgBufferSize = [0 2000 3000 4000];
[msgIndex,bsr] = nrMACBSR(lcgBufferSize)
```

```
msgIndex = 62

bsr = 4×1
```

```
    14
    85
    91
    96
```

Generate a MAC subPDU.

```
payload = bsr;
macSubPDU = nrMACSubPDU(linkDir, msgIndex, payload)
```

```
macSubPDU = 6×1

    62
     4
    14
    85
    91
    96
```

Input Arguments

`linkDir` — Transmission direction
0 | 1

Transmission direction of the MAC subPDU, specified as one of these values.

- `0` — Use this value to set the transmission direction to downlink.
- `1` — Use this value to set the transmission direction to uplink.

Data Types: `double`

`msgIndex` — Message index
integer in the range [0, 62], [64, 319], or [320, 65855]

Message index, specified as one of these values.

- Integer in the range [0, 62] — Use this option to represent the logical channel ID (LCID) field of the subheader in a subPDU.
- Integer in the range [64, 319] — Use this option to represent the one-octet extended logical channel ID (eLCID) field of the subheader in a subPDU. For this case, the LCID value is set to `34`.
- Integer in the range [320, 65855] — Use this option to represent the two-octet eLCID field of the subheader in a subPDU. For this case, the LCID value is set to `33`.

To set `msgIndex`, refer to these tables specified in 3GPP TS 38.321.

- For a downlink shared channel (DL-SCH) subPDU, the `msgIndex` corresponds to the index value specified in 3GPP TS 38.321 Table 6.2.1-1, 6.2.1-1a, or 6.2.1-1b [1].
- For an uplink shared channel (UL-SCH) subPDU, the `msgIndex` corresponds to the index value specified in 3GPP TS 38.321 Table 6.2.1-2, 6.2.1-2a, or 6.2.1-2b [1].

Data Types: `double`

`payload` — MAC payload
`[ ]` | vector of octets in decimal format

MAC payload, specified a vector of octets in decimal format. The payload could be MAC SDU or MAC CE.

To indicate a MAC CE with an empty payload, set `payload` to `[ ]`.

Data Types: `double`

`paddingLength` — Required MAC padding size
positive integer

Required MAC padding size in bytes, specified as a positive integer.

Data Types: `double`

Output Arguments

`macSubPDU` — Generated MAC subPDU
column vector of octets in decimal format

Generated MAC subPDU, returned as a column vector of octets in decimal format.

Data Types: `double`

Version History

Introduced in R2022a

References

[1] 3GPP TS 38.321. "NR; Medium Access Control (MAC) protocol specification." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`nrMACBSR` | `nrMACBSRDecode`

nrOFDMDemodulate

Demodulate OFDM waveform

Syntax

```
grid = nrOFDMDemodulate(carrier,waveform)
grid = nrOFDMDemodulate(waveform,nrb,scs,initialNSlot)
grid = nrOFDMDemodulate( ___ ,Name,Value)
```

Description

`grid = nrOFDMDemodulate(carrier,waveform)` recovers a carrier resource array by demodulating `waveform`, an OFDM modulated waveform, for carrier configuration parameters `carrier`.

`grid = nrOFDMDemodulate(waveform,nrb,scs,initialNSlot)` demodulates `waveform` for `nrb`, the specified number of resource blocks, subcarrier spacing `scs`, and initial slot number `initialNSlot`.

`grid = nrOFDMDemodulate( ___ ,Name,Value)` specifies options by using one or more name-value pair arguments in addition to the input arguments in any of the previous syntaxes.

Examples

Demodulate OFDM Waveform

Recover a transmitted carrier resource array by demodulating an OFDM waveform.

Set carrier configuration parameters, specifying 106 resource blocks (RBs) in the carrier resource array.

```
carrier = nrCarrierConfig('NSizeGrid',106);
```

Generate physical downlink shared channel (PDSCH) demodulation reference signal (DM-RS) symbols and indices.

```
p = 2;
pdsch = nrPDSCHConfig('NumLayers',p);
sym = nrPDSCHDMRS(carrier,pdsch);
ind = nrPDSCHDMRSIndices(carrier,pdsch);
```

Create a carrier resource array containing the PDSCH DM-RS symbols.

```
txGrid = nrResourceGrid(carrier,p);
txGrid(ind) = sym;
```

Generate OFDM modulated waveform.

```
[txWaveform,~] = nrOFDMModulate(carrier,txGrid);
```

Pass the waveform through a simple 2-by-1 channel.

```
H = [0.6; 0.4];
waveform = txWaveform*H;
```

Recover the carrier resource array by demodulating the received OFDM waveform.

```
grid = nrOFDMDemodulate(carrier,waveform);
```

Demodulate OFDM Waveform with Extended Cyclic Prefix

Recover a resource array that contains PDSCH DM-RS symbols by demodulating an OFDM waveform.

Set carrier configuration parameters, specifying a subcarrier spacing of 60 kHz.

```
scs = 60;
carrier = nrCarrierConfig('SubcarrierSpacing',scs);
```

Generate PDSCH DM-RS symbols and indices.

```
p = 2;
pdsch = nrPDSCHConfig('NumLayers',p);
sym = nrPDSCHDMRS(carrier,pdsch);
ind = nrPDSCHDMRSIndices(carrier,pdsch);
```

Create a carrier resource array containing the PDSCH DM-RS symbols.

```
txGrid = nrResourceGrid(carrier,p);
txGrid(ind) = sym;
```

Generate an OFDM modulated waveform, specifying the subcarrier spacing, initial slot number, and cyclic prefix length.

```
initialNSlot = carrier.NSlot;
cpl = 'extended';
[txWaveform,info] = nrOFDMModulate(txGrid,scs,initialNSlot,'CyclicPrefix',cpl);
```

Pass the waveform through a simple 2-by-1 channel.

```
H = [0.9; 0.95];
waveform = txWaveform*H;
```

Recover the carrier resource array by demodulating the received OFDM waveform.

```
nrb = carrier.NSizeGrid;
grid = nrOFDMDemodulate(waveform,nrb,scs,initialNSlot,'CyclicPrefix',cpl);
```

Demodulate OFDM Waveform with Specified Sample Rate

Recover a transmitted resource array that contains sounding reference signals (SRSs) and spans an entire frame by demodulating an OFDM waveform.

Set carrier configuration parameters, specifying a subcarrier spacing of 30 kHz and 24 resource blocks in the carrier resource array.

```
carrier = nrCarrierConfig('SubcarrierSpacing',30,'NSizeGrid',24);
```

Configure SRS parameters, setting the slot periodicity and offset.

```
srs = nrSRSConfig('SRSPeriod',[4 0]);
```

Get OFDM information for the specified carrier configuration.

```
info = nrOFDMInfo(carrier);
```

Produce the frame resource array by creating and concatenating slot resource arrays.

```
frameGrid = [];
for nslot = 0:(info.SlotsPerFrame - 1)
    carrier.NSlot = nslot;
    slotGrid = nrResourceGrid(carrier);
    ind = nrSRSIndices(carrier,srs);
    sym = nrSRS(carrier,srs);
    slotGrid(ind) = sym;
    frameGrid = [frameGrid slotGrid];
end
```

Generate the OFDM modulated waveform.

```
[txWaveform,~] = nrOFDMModulate(carrier,frameGrid);
```

Pass the waveform through a simple channel.

```
H = 0.86;
waveform = txWaveform*H;
```

Recover the carrier resource array by demodulating the received OFDM waveform, specifying the sample rate.

```
sr = info.SampleRate;
grid = nrOFDMDemodulate(carrier,waveform,'SampleRate',sr);
```

Input Arguments

carrier — Carrier configuration parameters
nrCarrierConfig object

Carrier configuration parameters for a specific OFDM numerology, specified as an nrCarrierConfig object. Only these object properties are relevant for this function.

NSizeGrid — Number of RBs in carrier resource grid
52 (default) | integer from 1 to 275

Number of RBs in the carrier resource grid, specified as an integer from 1 to 275. The default value of 52 corresponds to the maximum number of RBs of a 10 MHz carrier with 15 kHz SCS.

Data Types: double

SubcarrierSpacing — Subcarrier spacing in kHz
15 (default) | 30 | 60 | 120 | 240

Subcarrier spacing in kHz, for all channels and reference signals of the carrier, specified as `15`, `30`, `60`, `120`, or `240`.

Data Types: `double`

`NSlot` — Slot number

`0` (default) | nonnegative integer

Slot number, specified as a nonnegative integer. You can set `NSlot` to a value larger than the number of slots per frame. For example, you can set this value using transmission loop counters in a MATLAB simulation. In this case, you may have to ensure that the property value is modulo the number of slots per frame in a calling code.

Data Types: `double`

`CyclicPrefix` — Cyclic prefix length

`'normal'` (default) | `'extended'`

Cyclic prefix length, specified as one of these options.

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies for only 60 kHz subcarrier spacing.

Data Types: `char` | `string`

`waveform` — OFDM modulated waveform

complex-valued matrix

OFDM modulated waveform, specified as a complex-valued matrix of size *T*-by-*R*.

- *T* is the number of time-domain samples in the waveform.
- *R* is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

`nrb` — Number of resource blocks

integer from 1 to 275

Number of resource blocks, specified as an integer from 1 to 275.

Data Types: `double`

`scs` — Subcarrier spacing in kHz

`15` | `30` | `60` | `120` | `240`

Subcarrier spacing in kHz, specified as `15`, `30`, `60`, `120`, or `240`.

Data Types: `double`

`initialNSlot` — Initial slot number

nonnegative integer

Initial slot number, in 0-based form, specified as a nonnegative integer. The function selects the appropriate cyclic prefix lengths for OFDM demodulation by using the value of `initialNSlot` mod *S*, where *S* is the number of slots per subframe.

Data Types: `double`

Name-Value Pair Arguments

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

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

Example: `'CyclicPrefixFraction',0.75` specifies the start location for demodulation relative to the cyclic prefix length.

`CyclicPrefix` — Cyclic prefix length
`'normal'` (default) | `'extended'`

Cyclic prefix length, specified as the comma-separated pair consisting of `'CyclicPrefix'` and one of these values:

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, the extended cyclic prefix length only applies to 60 kHz subcarrier spacing.

Note If you specify the `carrier` input, use the `CyclicPrefix` property of the `carrier` input to specify the cyclic prefix length. You cannot use this name-value pair argument together with the `carrier` input.

Data Types: `char` | `string`

`Nfft` — Number of FFT points
integer greater than 127 (default depends on other input values) | `[]`

Number of fast Fourier transform (FFT) points, specified as the comma-separated pair consisting of `'Nfft'` and a nonnegative integer greater than 127 or `[]`. The value you specify must result in integer-valued cyclic prefix lengths and a maximum occupancy of 100%. The occupancy is defined as the value of $(12 \times N_{RB})$/`Nfft`, where N_{RB} is the number of resource blocks.

If you do not specify this input, or if you specify `'Nfft',[]`, the function sets an integer value greater than 127 as a default value for this input. The actual default value depends on other input values.

- If you do not specify the `SampleRate` input, or if you specify `'SampleRate',[]`, the function sets `Nfft` satisfying these conditions.
  - `Nfft` is an integer power of 2.
  - `Nfft` results in a maximum occupancy of 85%.

- If you specify the `SampleRate` input, the function sets `Nfft` satisfying these conditions.
  - `Nfft` results in integer-valued cyclic prefix lengths.
  - `Nfft` maximises the value of `gcd` (`Nfft` × *SCS*, `SampleRate`), where *SCS* is specified by the `carrier.SubcarrierSpacing` property or the `scs` input.

For more information, see "Configure OFDM Sample Rate and FFT Size".

Data Types: `double`

`SampleRate` — Waveform sample rate
positive scalar (default depends on other input values) | `[]`

Waveform sample rate, specified as the comma-separated pair consisting of `'SampleRate'` and either a positive scalar or `[]`.

If you do not specify this input, or if you specify `'SampleRate',[]`, then the function sets this input to the value of $N_{fft} \times SCS$.

- N_{fft} is the value of the `'Nfft'` input.
- *SCS* is the subcarrier spacing. Depending on the function syntax you use, *SCS* is specified by the `carrier.SubcarrierSpacing` property or the `scs` input.

For more information, see "Configure OFDM Sample Rate and FFT Size".

Data Types: `double`

`CarrierFrequency` — Carrier frequency in Hz
`0` (default) | real number

Carrier frequency in Hz, specified as the comma-separated pair consisting of `'CarrierFrequency'` and a real number. This input corresponds to f_0, defined in TS 38.211 Section 5.4.

Data Types: `double`

`CyclicPrefixFraction` — FFT window position within cyclic prefix
`0.5` (default) | scalar in the interval [0, 1]

Fast Fourier transform (FFT) window position within the cyclic prefix, specified as the comma-separated pair consisting of `'CyclicPrefixFraction'` and a scalar in the interval [0, 1].

The value that you specify indicates the start location for OFDM demodulation relative to the beginning of the cyclic prefix.

Data Types: `double`

Output Arguments

`grid` — Carrier resource array
complex-valued array

Carrier resource array, returned as a complex-valued array of size *K*-by-*L*-by-*R*.

- *K* is the number of subcarriers.
- *L* is the number of OFDM symbols.

- *R* is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

Version History

Introduced in R2020b

References

[1] 3GPP TS 38.101-1. "NR; User Equipment (UE) radio transmission and reception; Part 1: Range 1 Standalone." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.101-2. "NR; User Equipment (UE) radio transmission and reception; Part 2: Range 2 Standalone." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 38.104. "NR; Base Station (BS) radio transmission and reception." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[4] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- Names and values in name-value pair arguments must be compile-time constants. For example, when specifying extended cyclic prefix, include `{coder.Constant('CyclicPrefix'),coder.Constant('extended')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.
- The input arguments `nrb` and `scs` must be compile-time constants. For example, include `{coder.Constant(nrb)}` and `{coder.Constant(scs)}` in the `-args` value of the `codegen` function.
- The `'SampleRate'` name-value pair argument cannot be used together with the `carrier` input.

See Also

Functions

`nrOFDMInfo` | `nrOFDMModulate` | `nrResourceGrid`

Objects

`nrCarrierConfig`

Topics

"Configure OFDM Sample Rate and FFT Size"

“Resampling Filter Design in OFDM Functions”

nrOFDMInfo

Get OFDM information

Syntax

```
info = nrOFDMInfo(carrier)
info = nrOFDMInfo(nrb,scs)
info = nrOFDMInfo( ___ ,Name,Value)
```

Description

`info = nrOFDMInfo(carrier)` provides dimensional information relevant to orthogonal frequency-division multiplexing (OFDM) modulation for the specified carrier configuration parameters.

`info = nrOFDMInfo(nrb,scs)` provides OFDM information for the specified number of resource blocks, `nrb`, and subcarrier spacing `scs`.

`info = nrOFDMInfo( ___ ,Name,Value)` specifies options by using one or more name-value pair arguments in addition to the input arguments in any of the previous syntaxes.

Examples

Generate OFDM Modulated Waveform

Generate a waveform by performing OFDM modulation of a resource array that contains sounding reference signals (SRSs). The resource array spans an entire frame.

Set carrier configuration parameters, specifying a subcarrier spacing of 30 kHz and 24 resource blocks (RBs) in the carrier resource array.

```
carrier = nrCarrierConfig('SubcarrierSpacing',30,'NSizeGrid',24);
```

Configure SRS parameters, setting the slot periodicity to 2 and the offset to zero.

```
srs = nrSRSConfig('SRSPeriod',[2 0]);
```

Get OFDM information for the specified carrier configuration.

```
info = nrOFDMInfo(carrier);
```

Produce the frame resource array by creating and concatenating individual slot resource arrays.

```
grid = [];
for nslot = 0:(info.SlotsPerFrame - 1)
    carrier.NSlot = nslot;
    slotGrid = nrResourceGrid(carrier);
    ind = nrSRSIndices(carrier,srs);
    sym = nrSRS(carrier,srs);
    slotGrid(ind) = sym;
```

```
    grid = [grid slotGrid];
end
```

Perform OFDM modulation on the resource array for the specified carrier configuration.

```
[waveform,info] = nrOFDMModulate(carrier,grid);
```

Get OFDM Information for Extended Cyclic Prefix

Set carrier configuration parameters, specifying a subcarrier spacing of 60 kHz and extended cyclic prefix.

```
scs = 60;
cpl = 'Extended';
```

Set the number of resource blocks to `150`.

Generate and display OFDM information.

```
nrb = 150;
info = nrOFDMInfo(nrb,scs,'CyclicPrefix',cpl)
```

```
info = struct with fields:
                   Nfft: 4096
             SampleRate: 245760000
    CyclicPrefixLengths: [1024 1024 1024 1024 1024 1024 1024 1024 1024 ... ]
          SymbolLengths: [5120 5120 5120 5120 5120 5120 5120 5120 5120 ... ]
              Windowing: 116
           SymbolPhases: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ... ]
         SymbolsPerSlot: 12
       SlotsPerSubframe: 4
          SlotsPerFrame: 40
```

Get OFDM Information for Specified Sample Rate

Set carrier configuration parameters, specifying 106 RBs in the carrier resource array.

```
carrier = nrCarrierConfig('NSizeGrid',106);
```

Generate and display OFDM information for the specified sample rate.

```
sr = 1e8;
info = nrOFDMInfo(carrier,'SampleRate',sr)
```

```
info = struct with fields:
                   Nfft: 3200
             SampleRate: 100000000
    CyclicPrefixLengths: [250 225 225 225 225 225 225 250 225 225 225 ... ]
          SymbolLengths: [3450 3425 3425 3425 3425 3425 3425 3450 3425 ... ]
              Windowing: 112
           SymbolPhases: [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
         SymbolsPerSlot: 14
```

```
    SlotsPerSubframe: 1
       SlotsPerFrame: 10
```

Input Arguments

carrier — Carrier configuration parameters
nrCarrierConfig object

Carrier configuration parameters for a specific OFDM numerology, specified as an nrCarrierConfig object. Only these object properties are relevant for this function.

- NSizeGrid
- SubcarrierSpacing
- CyclicPrefix

nrb — Number of resource blocks
integer from 1 to 275

Number of resource blocks, specified as an integer from 1 to 275.

Data Types: double

scs — Subcarrier spacing in kHz
15 | 30 | 60 | 120 | 240

Subcarrier spacing in kHz, specified as 15, 30, 60, 120, or 240.

Data Types: double

Name-Value Pair Arguments

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

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

Example: 'SampleRate','1e9' specifies a sample rate of 1×10^9 Hz.

CyclicPrefix — Cyclic prefix length
'normal' (default) | 'extended'

Cyclic prefix length, specified as the comma-separated pair consisting of 'CyclicPrefix' and one of these values:

- 'normal' — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- 'extended' — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, the extended cyclic prefix length only applies to 60 kHz subcarrier spacing.

Note If you specify the `carrier` input, use the `CyclicPrefix` property of the `carrier` input to specify the cyclic prefix length. You cannot use this name-value pair argument together with the `carrier` input.

Data Types: `char` | `string`

`Nfft` — Number of FFT points
integer greater than 127 (default depends on other input values) | `[]`

Number of fast Fourier transform (FFT) points, specified as the comma-separated pair consisting of `'Nfft'` and a nonnegative integer greater than 127 or `[]`. The value you specify must result in integer-valued cyclic prefix lengths and a maximum occupancy of 100%. The occupancy is defined as the value of (12 × N_{RB})/`Nfft`, where N_{RB} is the number of resource blocks.

If you do not specify this input, or if you specify `'Nfft',[]`, the function sets an integer value greater than 127 as a default value for this input. The actual default value depends on other input values.

- If you do not specify the `SampleRate` input, or if you specify `'SampleRate',[]`, the function sets `Nfft` satisfying these conditions.
  - `Nfft` is an integer power of 2.
  - `Nfft` results in a maximum occupancy of 85%.
- If you specify the `SampleRate` input, the function sets `Nfft` satisfying these conditions.
  - `Nfft` results in integer-valued cyclic prefix lengths.
  - `Nfft` maximises the value of `gcd` (`Nfft` × *SCS*, `SampleRate`), where *SCS* is specified by the `carrier.SubcarrierSpacing` property or the `scs` input.

For more information, see "Configure OFDM Sample Rate and FFT Size".

Data Types: `double`

`SampleRate` — Waveform sample rate
positive scalar (default depends on other input values) | `[]`

Waveform sample rate, specified as the comma-separated pair consisting of `'SampleRate'` and either a positive scalar or `[]`.

If you do not specify this input, or if you specify `'SampleRate',[]`, then the function sets this input to the value of N_{fft} × *SCS*.

- N_{fft} is the value of the `'Nfft'` input.
- *SCS* is the subcarrier spacing. Depending on the function syntax you use, *SCS* is specified by the `carrier.SubcarrierSpacing` property or the `scs` input.

For more information, see "Configure OFDM Sample Rate and FFT Size".

Data Types: `double`

`Windowing` — Number of time-domain samples for OFDM symbol windowing and overlapping
nonnegative integer (default depends on other input values) | `[]`

Number of time-domain samples over which the function applies raised cosine windowing and overlapping of OFDM symbols, specified as the comma-separated pair consisting of `'Windowing'` and either a nonnegative integer or `[]`.

If you do not specify this input, or if you specify `'Windowing',[]`, the function sets this input to the maximum value E that does not impact error vector magnitude (EVM) tests, as specified in TS 38.101-1 Annexes F.5.3 and F.5.4, TS 38.101-2 Annexes F.5.3 and F.5.4, and TS 38.104 Annexes B.5.2 and C.5.2. E is equal to value of `floor`((N_{CP} − W) × `info.Nfft`/$N_{FFT, nominal}$), where N_{CP}, W, and $N_{FFT, nominal}$ are the values in the table columns labeled "Cyclic prefix length", "EVM window length", and "FFT size", respectively.

Data Types: `double`

`CarrierFrequency` — Carrier frequency in Hz
`0` (default) | real number

Carrier frequency in Hz, specified as the comma-separated pair consisting of `'CarrierFrequency'` and a real number. This input corresponds to f_0, defined in TS 38.211 Section 5.4.

Data Types: `double`

Output Arguments

`info` — OFDM information
structure

OFDM information, returned as a structure containing these fields.

Fields	Values	Description
`Nfft`	Positive integer	Number of FFT points
`SampleRate`	Positive scalar	Waveform sample rate
`CyclicPrefixLengths`	1-by-N vector of positive integers, where N is the number of OFDM symbols in a subframe.	Cyclic prefix lengths of each OFDM symbol, in samples
`SymbolLengths`	1-by-N vector of positive integers	OFDM symbol lengths, in samples
`Windowing`	Positive integer	Number of time-domain samples over which the function applies raised cosine windowing and overlapping of OFDM symbols

Fields	Values	Description
`SymbolPhases`	1-by-*N* vector of numbers in the interval [-π, π]	Phase compensation of each OFDM symbol, in radians The `nrOFDMModulate` function applies this compensation during modulation to account for phase terms per OFDM symbol, as specified in TS 38.211 Section 5.4 [4]. The `nrOFDMDemodulate` function inverts this phase compensation during demodulation.
`SymbolsPerSlot`	Positive integer	Number of OFDM symbols in a slot
`SlotsPerSubframe`	Positive integer	Number of slots in a 1 ms subframe
`SlotsPerFrame`	Positive integer	Number of slots in a 10 ms frame

For more information on these fields, see the “Use Custom OFDM Sample Rate and Custom FFT Size” example.

Data Types: `struct`

Version History

Introduced in R2020b

References

[1] 3GPP TS 38.101-1. “NR; User Equipment (UE) radio transmission and reception; Part 1: Range 1 Standalone.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.101-2. “NR; User Equipment (UE) radio transmission and reception; Part 2: Range 2 Standalone.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 38.104. “NR; Base Station (BS) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[4] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- Names and values in name-value pair arguments must be compile-time constants. For example, when specifying extended cyclic prefix, include `{coder.Constant('CyclicPrefix'),coder.Constant('extended')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.
- The input arguments `nrb` and `scs` must be compile-time constants. For example, include `{coder.Constant(nrb)}` and `{coder.Constant(scs)}` in the `-args` value of the `codegen` function.

See Also

Functions
`nrOFDMDemodulate` | `nrOFDMModulate` | `nrResourceGrid`

Objects
`nrCarrierConfig`

Topics
"Configure OFDM Sample Rate and FFT Size"
"Resampling Filter Design in OFDM Functions"

nrOFDMModulate

Generate OFDM modulated waveform

Syntax

```
[waveform,info] = nrOFDMModulate(carrier,grid)
[waveform,info] = nrOFDMModulate(grid,scs,initialNSlot)
[waveform,info] = nrOFDMModulate( ___ ,Name,Value)
```

Description

`[waveform,info] = nrOFDMModulate(carrier,grid)` generates `waveform`, a time-domain waveform, by performing orthogonal frequency-division multiplexing (OFDM) modulation of carrier resource array `grid` for carrier configuration parameters `carrier`. The function also returns `info`, a structure containing OFDM information.

`[waveform,info] = nrOFDMModulate(grid,scs,initialNSlot)` modulates the carrier resource array with subcarrier spacing `scs` and initial slot number `initialNSlot`.

`[waveform,info] = nrOFDMModulate( ___ ,Name,Value)` specifies options by using one or more name-value pair arguments in addition to the input arguments in any of the previous syntaxes.

Examples

Generate OFDM Modulated Waveform

Generate a waveform by performing OFDM modulation of a resource array that contains sounding reference signals (SRSs). The resource array spans an entire frame.

Set carrier configuration parameters, specifying a subcarrier spacing of 30 kHz and 24 resource blocks (RBs) in the carrier resource array.

```
carrier = nrCarrierConfig('SubcarrierSpacing',30,'NSizeGrid',24);
```

Configure SRS parameters, setting the slot periodicity to 2 and the offset to zero.

```
srs = nrSRSConfig('SRSPeriod',[2 0]);
```

Get OFDM information for the specified carrier configuration.

```
info = nrOFDMInfo(carrier);
```

Produce the frame resource array by creating and concatenating individual slot resource arrays.

```
grid = [];
for nslot = 0:(info.SlotsPerFrame - 1)
    carrier.NSlot = nslot;
    slotGrid = nrResourceGrid(carrier);
    ind = nrSRSIndices(carrier,srs);
    sym = nrSRS(carrier,srs);
```

```
    slotGrid(ind) = sym;
    grid = [grid slotGrid];
end
```

Perform OFDM modulation on the resource array for the specified carrier configuration.

```
[waveform,info] = nrOFDMModulate(carrier,grid);
```

Generate OFDM Modulated Waveform for Extended Cyclic Prefix

Generate a waveform by performing OFDM modulation of a resource array that contains physical downlink shared channel (PDSCH) demodulation reference signal (DM-RS) symbols.

Set carrier configuration parameters, specifying a subcarrier spacing of 60 kHz.

```
scs = 60;
carrier = nrCarrierConfig('SubcarrierSpacing',scs);
```

Generate PDSCH DM-RS symbols and indices.

```
p = 2;
pdsch = nrPDSCHConfig('NumLayers',p);
sym = nrPDSCHDMRS(carrier,pdsch);
ind = nrPDSCHDMRSIndices(carrier,pdsch);
```

Create a carrier resource array containing the PDSCH DM-RS symbols.

```
grid = nrResourceGrid(carrier,p);
grid(ind) = sym;
```

Generate an OFDM modulated waveform, specifying the subcarrier spacing, initial slot number, and cyclic prefix type. Display the OFDM information.

```
initialNSlot = carrier.NSlot;
cpl = 'extended';
[waveform,info] = nrOFDMModulate(grid,scs,initialNSlot,'CyclicPrefix',cpl);
disp(info)
```

```
                   Nfft: 1024
             SampleRate: 61440000
    CyclicPrefixLengths: [256 256 256 256 256 256 256 256 256 256 256 ... ]
          SymbolLengths: [1280 1280 1280 1280 1280 1280 1280 1280 1280 ... ]
              Windowing: 36
           SymbolPhases: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ... ]
         SymbolsPerSlot: 12
       SlotsPerSubframe: 4
          SlotsPerFrame: 40
```

Generate OFDM Modulated Waveform for Specified Sample Rate

Generate a waveform by performing OFDM modulation of a resource array that contains PDSCH DM-RS symbols.

Set carrier configuration parameters, specifying 106 RBs in the carrier resource array.

```
carrier = nrCarrierConfig('NSizeGrid',106);
```

Configure PDSCH and generate the corresponding symbols and indices.

```
p = 4;
pdsch = nrPDSCHConfig('NumLayers',p);
sym = nrPDSCHDMRS(carrier,pdsch);
ind = nrPDSCHDMRSIndices(carrier,pdsch);
```

Create a carrier resource array and map the PDSCH symbols.

```
grid = nrResourceGrid(carrier,p,'OutputDataType','single');
grid(ind) = sym;
```

Generate OFDM modulated waveform, specifying the sample rate.

```
sr = 1e8;
[waveform,info] = nrOFDMModulate(carrier,grid,'SampleRate',sr);
```

Input Arguments

carrier — Carrier configuration parameters
`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. Only these object properties are relevant for this function.

NSizeGrid — Number of RBs in carrier resource grid
`52` (default) | integer from 1 to 275

Number of RBs in the carrier resource grid, specified as an integer from 1 to 275. The default value of `52` corresponds to the maximum number of RBs of a 10 MHz carrier with 15 kHz SCS.

Data Types: `double`

SubcarrierSpacing — Subcarrier spacing in kHz
`15` (default) | `30` | `60` | `120` | `240`

Subcarrier spacing in kHz, for all channels and reference signals of the carrier, specified as `15`, `30`, `60`, `120`, or `240`.

Data Types: `double`

NSlot — Slot number
`0` (default) | nonnegative integer

Slot number, specified as a nonnegative integer. You can set `NSlot` to a value larger than the number of slots per frame. For example, you can set this value using transmission loop counters in a MATLAB simulation. In this case, you may have to ensure that the property value is modulo the number of slots per frame in a calling code.

Data Types: `double`

CyclicPrefix — Cyclic prefix length
`'normal'` (default) | `'extended'`

Cyclic prefix length, specified as one of these options.

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies for only 60 kHz subcarrier spacing.

Data Types: `char` | `string`

`grid` — Carrier resource array
complex-valued array

Carrier resource array, specified as a complex-valued array of size *K*-by-*N*-by-*P*.

- *K* is the number of subcarriers.
- *N* is the number of OFDM symbols.
- *P* is the number of transmit antennas.

Data Types: `single` | `double`
Complex Number Support: Yes

`scs` — Subcarrier spacing in kHz
`15` | `30` | `60` | `120` | `240`

Subcarrier spacing in kHz, specified as `15`, `30`, `60`, `120`, or `240`.

Data Types: `double`

`initialNSlot` — Initial slot number
nonnegative integer

Initial slot number, in 0-based form, specified as a nonnegative integer. The function selects the appropriate cyclic prefix lengths for OFDM modulation by using the value of `initialNSlot` mod *S*, where *S* is the number of slots per subframe.

Data Types: `double`

Name-Value Pair Arguments

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

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

Example: `'CyclicPrefix','extended'` specifies extended cyclic prefix length.

`CyclicPrefix` — Cyclic prefix length
`'normal'` (default) | `'extended'`

Cyclic prefix length, specified as the comma-separated pair consisting of `'CyclicPrefix'` and one of these values:

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.

- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, the extended cyclic prefix length only applies to 60 kHz subcarrier spacing.

Note If you specify the `carrier` input, use the `CyclicPrefix` property of the `carrier` input to specify the cyclic prefix length. You cannot use this name-value pair argument together with the `carrier` input.

Data Types: `char` | `string`

`Nfft` — Number of FFT points
integer greater than 127 (default depends on other input values) | `[]`

Number of fast Fourier transform (FFT) points, specified as the comma-separated pair consisting of `'Nfft'` and a nonnegative integer greater than 127 or `[]`. The value you specify must result in integer-valued cyclic prefix lengths and a maximum occupancy of 100%. The occupancy is defined as the value of ($12 \times N_{RB}$)/`Nfft`, where N_{RB} is the number of resource blocks.

If you do not specify this input, or if you specify `'Nfft',[]`, the function sets an integer value greater than 127 as a default value for this input. The actual default value depends on other input values.

- If you do not specify the `SampleRate` input, or if you specify `'SampleRate',[]`, the function sets `Nfft` satisfying these conditions.
  - `Nfft` is an integer power of 2.
  - `Nfft` results in a maximum occupancy of 85%.
- If you specify the `SampleRate` input, the function sets `Nfft` satisfying these conditions.
  - `Nfft` results in integer-valued cyclic prefix lengths.
  - `Nfft` maximises the value of `gcd` (`Nfft` × *SCS*, `SampleRate`), where *SCS* is specified by the `carrier.SubcarrierSpacing` property or the `scs` input.

For more information, see “Configure OFDM Sample Rate and FFT Size”.

Data Types: `double`

`SampleRate` — Waveform sample rate
positive scalar (default depends on other input values) | `[]`

Waveform sample rate, specified as the comma-separated pair consisting of `'SampleRate'` and either a positive scalar or `[]`.

If you do not specify this input, or if you specify `'SampleRate',[]`, then the function sets this input to the value of $N_{fft} \times SCS$.

- N_{fft} is the value of the `'Nfft'` input.
- *SCS* is the subcarrier spacing. Depending on the function syntax you use, *SCS* is specified by the `carrier.SubcarrierSpacing` property or the `scs` input.

For more information, see “Configure OFDM Sample Rate and FFT Size”.

Data Types: `double`

`Windowing` — Number of time-domain samples for OFDM symbol windowing and overlapping
nonnegative integer (default depends on other input values) | `[]`

Number of time-domain samples over which the function applies raised cosine windowing and overlapping of OFDM symbols, specified as the comma-separated pair consisting of `'Windowing'` and either a nonnegative integer or `[]`.

If you do not specify this input, or if you specify `'Windowing',[]`, the function sets this input to the maximum value E that does not impact error vector magnitude (EVM) tests, as specified in TS 38.101-1 Annexes F.5.3 and F.5.4, TS 38.101-2 Annexes F.5.3 and F.5.4, and TS 38.104 Annexes B.5.2 and C.5.2. E is equal to value of `floor`$((N_{CP} - W) \times$ `info.Nfft`$/N_{FFT,\ nominal})$, where N_{CP}, W, and $N_{FFT,\ nominal}$ are the values in the table columns labeled "Cyclic prefix length", "EVM window length", and "FFT size", respectively.

Data Types: `double`

`CarrierFrequency` — Carrier frequency in Hz
`0` (default) | real number

Carrier frequency in Hz, specified as the comma-separated pair consisting of `'CarrierFrequency'` and a real number. This input corresponds to f_0, defined in TS 38.211 Section 5.4.

Data Types: `double`

Output Arguments

`waveform` — OFDM modulated waveform
complex-valued matrix

OFDM modulated waveform, returned as a complex-valued matrix of size T-by-P.

- T is the number of time-domain samples in the waveform.
- P is the number of transmit antennas.

Data Types: `single` | `double`
Complex Number Support: Yes

`info` — OFDM information
structure

OFDM information, returned as a structure containing these fields.

Fields	Values	Description
`Nfft`	Positive integer	Number of FFT points
`SampleRate`	Positive scalar	Waveform sample rate
`CyclicPrefixLengths`	1-by-N vector of positive integers, where N is the number of OFDM symbols in a subframe.	Cyclic prefix lengths of each OFDM symbol, in samples
`SymbolLengths`	1-by-N vector of positive integers	OFDM symbol lengths, in samples

Fields	Values	Description
`Windowing`	Positive integer	Number of time-domain samples over which the function applies raised cosine windowing and overlapping of OFDM symbols
`SymbolPhases`	1-by-*N* vector of scalars in the interval [-π, π]	Phase compensation of each OFDM symbol, in radians The function applies this compensation during modulation to account for phase terms per OFDM symbol, as specified in TS 38.211 Section 5.4 [4]. The `nrOFDMDemodulate` function inverts this phase compensation during demodulation.
`SymbolsPerSlot`	Positive integer	Number of OFDM symbols in a slot
`SlotsPerSubframe`	Positive integer	Number of slots in a 1 ms subframe
`SlotsPerFrame`	Positive integer	Number of slots in a 10 ms frame

Data Types: `struct`

Version History

Introduced in R2020b

References

[1] 3GPP TS 38.101-1. “NR; User Equipment (UE) radio transmission and reception; Part 1: Range 1 Standalone.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.101-2. “NR; User Equipment (UE) radio transmission and reception; Part 2: Range 2 Standalone.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 38.104. “NR; Base Station (BS) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[4] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- Names and values in name-value pair arguments must be compile-time constants. For example, when specifying extended cyclic prefix, include `{coder.Constant('CyclicPrefix'),coder.Constant('extended')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.
- When calling the function with input arguments `grid` and `scs`, the first dimension of `grid` and the `scs` input must be compile-time constants. For example, include `{coder.typeof(grid, [624 Inf Inf],[0 1 1])}` and `{coder.Constant(scs)}` in the `-args` value of the `codegen` function.
- The `'SampleRate'` name-value pair argument cannot be used together with the `carrier` input.

See Also

Functions
`nrOFDMDemodulate` | `nrOFDMInfo` | `nrResourceGrid`

Objects
`nrCarrierConfig`

Topics
"Configure OFDM Sample Rate and FFT Size"
"Resampling Filter Design in OFDM Functions"

nrORANBlockCompress

O-RAN fronthaul block compression

Syntax

```
[cGrid,cParam] = nrORANBlockCompress(grid,method,cIQWidth,IQWidth)
[cGrid,cParam] = nrORANBlockCompress(grid,method,cIQWidth)
```

Description

`[cGrid,cParam] = nrORANBlockCompress(grid,method,cIQWidth,IQWidth)` performs open radio access network (O-RAN) fronthaul block compression on the user plane (U-Plane) IQ data, `grid`. The input `method` specifies the compression method. The function implements the block floating point (BFP), block scaling, and µ-law compression methods, as defined in O-RAN.WG4.CUS TS Annexes A.1.1, A.2.1, and A.3.1, respectively [1]. The function returns the compressed grid, `cGrid`, and the compression parameter, `cParam`. Inputs `IQWidth` and `cIQWidth` specify the bit width of the IQ samples before and after compression, respectively.

`[cGrid,cParam] = nrORANBlockCompress(grid,method,cIQWidth)` performs only BFP or µ-law compression.

Examples

Perform O-RAN Fronthaul Block Compression and Decompression

Generate an NR downlink resource grid to perform O-RAN fronthaul block compression.

```
cfg = nrDLCarrierConfig;
[waveform,info] = nrWaveformGenerator(cfg);
grid = info.ResourceGrids.ResourceGridBWP;
```

Scale the bit width of the IQ samples in the resource grid to 16 bits.

```
IQWidth = 16;
peak = max(abs([real(grid(:)); imag(grid(:))]));
scaleFactor = peak / (2^(IQWidth-1)-1);
scaledGrid = round(grid / scaleFactor);
```

Specify the bit width for the compressed IQ samples.

```
cIQWidth = 9;
```

Specify the compression method.

```
method = 'BFP';
```

Perform O-RAN fronthaul block compression.

```
[cGrid,cParam] = nrORANBlockCompress(scaledGrid,method,cIQWidth);
```

Perform O-RAN fronthaul block decompression.

```
dGrid = nrORANBlockDecompress(cGrid,cParam,method);
```

Descale the decompressed grid using the scaling factor from before the compression.

```
descaledGrid = dGrid * scaleFactor;
```

Input Arguments

grid — Resource grid of U-Plane IQ data
K-by-*L*-by-*P* complex array

Resource grid of U-Plane IQ data, specified as a *K*-by-*L*-by-*P* complex array. *K* is the number of subcarriers, a multiple of 12, where 12 corresponds to the resource elements (REs) in a physical resource block (PRB). *L* is the number of OFDM symbols. *P* is the number of antennas. The input `IQWidth` specifies the bit width of the IQ samples in the grid.

Data Types: `double` | `single`
Complex Number Support: Yes

method — O-RAN fronthaul compression method
`'BFP'` | `'blockScaling'` | `'muLaw'`

O-RAN fronthaul compression method, specified as one of these values.

- `'BFP'` — Specify this value for BFP compression, as defined in O-RAN.WG4.CUS TS Annex A.1.1.
- `'blockScaling'` — Specify this value for block scaling compression, as defined in O-RAN.WG4.CUS TS Annex A.2.1.
- `'muLaw'` — Specify this value for µ-law compression, as defined in O-RAN.WG4.CUS TS Annex A.3.1.

Data Types: `string` | `char`

cIQWidth — Bit width of compressed IQ samples
integer from 1 to 16 | *K*/12-by-*L*-by-*P* integer array

Bit width of the compressed IQ samples (in the output `cGrid`), including the sign bit, specified as one of these values.

- Integer from 1 to 16 — This value specifies the bit width for all compressed IQ samples.
- *K*/12-by-*L*-by-*P* integer array with values from 1 to 16 — The vector elements specify the bit width of the compressed IQ samples within a PRB.

Data Types: `double`

IQWidth — Bit width of IQ samples before compression
integer from 1 to 32

Bit width of the IQ samples before compression (in the input `grid`), specified as an integer from 1 to 32. For µ-law compression, this input must be 16. For BFP compression, the function ignores this input.

Data Types: `double`

Output Arguments

`cGrid` — Compressed resource grid of U-Plane IQ data
K-by-*L*-by-*P* complex array

Compressed resource grid of U-Plane IQ data, returned as a *K*-by-*L*-by-*P* complex array. The input `cIQWidth` specifies the bit width of the IQ samples in the compressed grid.

Data Types: `double` | `single`
Complex Number Support: Yes

`cParam` — Compression parameter
K/12-by-*L*-by-*P* integer array

Compression parameter, returned as a *K*/12-by-*L*-by-*P* integer array.

- For BFP compression, `cParam` is the common exponent applied per compressed PRB, with integer values from 0 to `IQWidth` – 1.
- For block scaling compression, `cParam` is the common scale factor applied per compressed PRB, with integer values from 1 to 128.
- For µ-law compression, `cParam` is the common shift applied per compressed PRB, with integer values from 0 to 7.

Data Types: `double`

Version History

Introduced in R2022b

References

[1] O-RAN.WG4.CUS TS. “O-RAN Fronthaul Working Group; Control, User and Synchronization Plane Specification” *Open Radio Access Network Alliance Technical Specification*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrORANBlockDecompress`

nrORANBlockDecompress

O-RAN fronthaul block decompression

Syntax

```
dGrid = nrORANBlockDecompress(cGrid,cParam,method,cIQWidth,IQWidth)
dGrid = nrORANBlockDecompress(cGrid,cParam,method,cIQWidth)
dGrid = nrORANBlockDecompress(cGrid,cParam,method)
```

Description

`dGrid = nrORANBlockDecompress(cGrid,cParam,method,cIQWidth,IQWidth)` performs open radio access network (O-RAN) fronthaul block decompression on the compressed user plane (U-Plane) IQ data, `cGrid`. The input `method` specifies the decompression method. The function implements the block floating point (BFP), block scaling, and μ-law decompression methods, as defined in O-RAN.WG4.CUS TS Annexes A.1.2, A.2.2, and A.3.2, respectively [1]. The function returns the decompressed grid, `dGrid`. Inputs `IQWidth` and `cIQWidth` specify the bit width of the IQ samples before and after compression, respectively. `cParam` specifies the compression parameter applied in the compressed data, `cGrid`.

`dGrid = nrORANBlockDecompress(cGrid,cParam,method,cIQWidth)` performs only BFP or μ-law decompression.

`dGrid = nrORANBlockDecompress(cGrid,cParam,method)` performs only BFP decompression.

Examples

Perform O-RAN Fronthaul Block Compression and Decompression

Generate an NR downlink resource grid to perform O-RAN fronthaul block compression.

```
cfg = nrDLCarrierConfig;
[waveform,info] = nrWaveformGenerator(cfg);
grid = info.ResourceGrids.ResourceGridBWP;
```

Scale the bit width of the IQ samples in the resource grid to 16 bits.

```
IQWidth = 16;
peak = max(abs([real(grid(:)); imag(grid(:))]));
scaleFactor = peak / (2^(IQWidth-1)-1);
scaledGrid = round(grid / scaleFactor);
```

Specify the bit width for the compressed IQ samples.

```
cIQWidth = 9;
```

Specify the compression method.

```
method = 'BFP';
```

Perform O-RAN fronthaul block compression.

```
[cGrid,cParam] = nrORANBlockCompress(scaledGrid,method,cIQWidth);
```

Perform O-RAN fronthaul block decompression.

```
dGrid = nrORANBlockDecompress(cGrid,cParam,method);
```

Descale the decompressed grid using the scaling factor from before the compression.

```
descaledGrid = dGrid * scaleFactor;
```

Input Arguments

`cGrid` — Compressed resource grid of U-Plane IQ data

K-by-*L*-by-*P* complex array

Compressed resource grid of U-Plane IQ data, specified as a *K*-by-*L*-by-*P* complex array. *K* is the number of subcarriers, a multiple of 12, where 12 corresponds to the resource elements (REs) in a physical resource block (PRB). *L* is the number of OFDM symbols. *P* is the number of antennas. The input `cIQWidth` specifies the bit width of the IQ samples in the compressed grid.

Data Types: `double` | `single`
Complex Number Support: Yes

`cParam` — Compression parameter

K/12-by-*L*-by-*P* integer array

Compression parameter, specified as a *K*/12-by-*L*-by-*P* integer array.

- For BFP compression, `cParam` is the common exponent applied per compressed PRB, with integer values from 0 to `IQWidth` – 1.
- For block scaling compression, `cParam` is the common scale factor applied per compressed PRB, with integer values from 1 to 128.
- For µ-law compression, `cParam` is the common shift applied per compressed PRB, with integer values from 0 to 7.

Data Types: `double`

`method` — O-RAN fronthaul decompression method

`'BFP'` | `'blockScaling'` | `'muLaw'`

O-RAN fronthaul decompression method, specified as one of these values.

- `'BFP'` — Specify this value for BFP decompression, as defined in O-RAN.WG4.CUS TS Annex A.1.2.
- `'blockScaling'` — Specify this value for block scaling decompression, as defined in O-RAN.WG4.CUS TS Annex A.2.2.
- `'muLaw'` — Specify this value for µ-law decompression, as defined in O-RAN.WG4.CUS TS Annex A.3.2.

Data Types: `string` | `char`

`cIQWidth` — Bit width of compressed IQ samples

integer from 1 to 16 | *K*/12-by-*L*-by-*P* integer array

Bit width of the compressed IQ samples (in the input `cGrid`), including the sign bit, specified as one of these values.

- Integer from 1 to 16 — This value specifies the bit width for all compressed IQ samples.
- *K*/12-by-*L*-by-*P* integer array with values from 1 to 16 — The vector elements specify the bit width of the compressed IQ samples within a PRB.

For BFP decompression, the function ignores this input.

Data Types: `double`

`IQWidth` — Bit width of IQ samples before compression
integer from 1 to 32

Bit width of the IQ samples before compression, specified as an integer from 1 to 32. This value determines the bit width of the decompressed IQ samples in the output `dGrid`. For µ-law compression, this input must be 16. For BFP decompression, the function ignores this input.

Data Types: `double`

Output Arguments

`dGrid` — Decompressed resource grid of U-Plane IQ data
K-by-*L*-by-*P* complex array

Decompressed resource grid of U-Plane IQ data, returned as a *K*-by-*L*-by-*P* complex array. The input `IQWidth` determines the bit width of the IQ samples in the decompressed grid.

Data Types: `double` | `single`
Complex Number Support: Yes

Version History

Introduced in R2022b

References

[1] O-RAN.WG4.CUS TS. "O-RAN Fronthaul Working Group; Control, User and Synchronization Plane Specification" *Open Radio Access Network Alliance Technical Specification*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrORANBlockCompress`

nrPathLoss

Path loss and shadow fading between BS and UE

Syntax

```
[pathloss,shadowfading] = nrPathLoss(pathlossconf,freq,los,bs,ue)
```

Description

`[pathloss,shadowfading] = nrPathLoss(pathlossconf,freq,los,bs,ue)` returns the path loss, `pathloss`, between the base station (BS) and the user equipment (UE) for frequency `freq`. The function also returns the associated shadow fading standard deviation, `shadowfading`, as defined in TR 38.901 Section 7.4.1 [1]. `pathlossconf` specifies the scenario characteristics and the path loss model. `bs` and `ue` specify the Cartesian coordinates of the BS and UE, respectively. `los` specifies the presence or lack of line of sight (LOS) between the BS and UE. The function supports multiple BSs and multiple UEs.

Examples

Calculate Path Loss Between Single BS and UE

Create a path loss configuration object. Set the characteristics such that they correspond to a rural macrocell scenario with an average building height of 7 m and street width of 25 m.

```
pathlossconf = nrPathLossConfig;
pathlossconf.Scenario = "RMa";
pathlossconf.BuildingHeight = 7;
pathlossconf.StreetWidth = 25;
```

Specify the carrier frequency, LOS condition, and BS and UE coordinates.

```
freq = 3.5e9;
los = true;
bs = [0;0;30];
ue = [1e3;1e3;1.5];
```

Calculate the path loss between the BS and UE.

```
pathloss = nrPathLoss(pathlossconf,freq,los,bs,ue)
```

```
pathloss = 110.1615
```

Calculate Path Loss Between Multiple BSs and UEs

Create a path loss configuration object. Set the characteristics such that they correspond to a rural macrocell scenario with an average building height of 7 m and street width of 25 m.

```
pathlossconf = nrPathLossConfig;
pathlossconf.Scenario = "RMa";
```

```
pathlossconf.BuildingHeight = 7;
pathlossconf.StreetWidth = 25;
```

Specify the carrier frequency in Hz.

```
freq = 3.5e9;
```

Specify the coordinates of two BSs and ten UEs. The BSs are 1 km apart. The UEs are randomly placed inside a 2 km-by-2 km square region at elevations between 1 m and 2 m.

```
bs = [-500 500; 0 0; 30 50];
nbs = size(bs,2);
nue = 10;
ue = zeros(3,nue);
ue(1:2,:) = 2e3*(rand(2,nue)-0.5);
ue(3,:) = 1 + rand(1,nue);
```

Specify the LOS condition between each BS and UE pair.

```
los = randi([0 1],nbs,nue);
```

Calculate the path loss between each BS and UE pair.

```
pathloss = nrPathLoss(pathlossconf,freq,los,bs,ue)
```

```
pathloss = 2×10

  109.9708  131.3512  133.6539   85.0968  112.2626  131.3747  110.1649  137.4889  130.5121  110.2
  123.4839  134.9305  104.4709  125.2657  126.9743  133.3612   97.4411  121.9781  106.9546  105.4
```

Input Arguments

pathlossconf — Path loss configuration
nrPathLossConfig object

Path loss configuration, specified as an `nrPathLossConfig` object. This input specifies the scenario characteristics and the path loss model.

freq — Carrier frequency
positive numeric scalar

Carrier frequency, specified as a positive numeric scalar.

Data Types: `double`

los — LOS between BS and UE
`true` or `1` | `false` or `0` | logical matrix

LOS between the BS and UE, specified as one of these options.

- `1` (`true`) — Specifies the presence of LOS between the BS and UE
- `0` (`false`) — Specifies the lack of LOS between the BS and UE (NLOS)
- Logical matrix of size N_{BS}-by-N_{UE} — Specifies the existence or lack of LOS between each BS and UE pair. N_{BS} is the number of BSs. N_{UE} is the number of UEs.

Data Types: `double` | `logical`

bs — Cartesian coordinates of BS
numeric matrix

Cartesian coordinates of the BS, specified as a numeric matrix of size 3-by-N_{BS}. N_{BS} is the number of BSs. The first two rows of the matrix specify the 2-D position of the BS. The third row of the matrix specifies the height of the BS.

Data Types: `double`

ue — Cartesian coordinates of UE
numeric matrix

Cartesian coordinates of the UE, specified as a numeric matrix of size 3-by-N_{UE}. N_{UE} is the number of UEs. The first two rows of the matrix specify the 2-D position of the UE. The third row of the matrix specifies the height of the UE.

Data Types: `double`

Output Arguments

`pathloss` — Path loss between BS and UE
numeric matrix

Path loss between the BS and UE, returned as a numeric matrix of size N_{BS}-by-N_{UE}. N_{BS} is the number of BSs. N_{UE} is the number of UEs.

Note Because the input arguments are not restricted to the value ranges defined in TR 38.901, the function warns when the resulting path loss is not a positive number.

Data Types: `double`

`shadowfading` — Shadow fading standard deviation
numeric matrix

Shadow fading standard deviation in dB, returned as a numeric matrix of size N_{BS}-by-N_{UE}. N_{BS} is the the number of BSs. N_{UE} is the number of UEs.

Data Types: `double`

Version History

Introduced in R2021b

References

[1] 3GPP TR 38.901. “Study on channel model for frequencies from 0.5 to 100 GHz.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
`nrPathLossConfig`

Topics
"Include Path Loss in NR Link-Level Simulations"

nrPBCH

Generate PBCH modulation symbols

Syntax

```
sym = nrPBCH(cw,ncellid,v)
sym = nrPBCH(cw,ncellid,v,'OutputDataType',datatype)
```

Description

`sym = nrPBCH(cw,ncellid,v)` returns the physical broadcast channel (PBCH) modulation symbols for the physical layer cell identity number `ncellid`. The function implements TS 38.211 Section 7.3.3 [1]. The input `cw` is the BCH codeword, as described in TS 38.212 Section 7.1.5 [2]. The input `v` specifies the scrambling sequence phase.

`sym = nrPBCH(cw,ncellid,v,'OutputDataType',datatype)` specifies the data type of the PBCH symbol.

Examples

Generate Physical Broadcast Channel Symbols

Consider the first Synchronization Signal / Physical Broadcast Channel (SS/PBCH) block in a burst and assume that the number of SS/PBCH blocks per half-frame is 4.

```
ssbindex = 0;
v = mod(ssbindex,4);
```

Generate a random sequence of binary values that represent encoded BCH bits. The length of the random sequence corresponds to the PBCH bit capacity as specified in TS 38.212 Section 7.1.5.

```
E = 864;
cw = randi([0 1],E,1);
```

Generate the sequence of 432 PBCH quadrature phase shift keying (QPSK) modulation symbols.

```
ncellid = 17;
sym = nrPBCH(cw,ncellid,v);
```

Input Arguments

`cw` — BCH codeword
column vector of binary values

BCH codeword, specified as a column vector of binary values. The size of the vector is E = 864, as specified in TS 38.212 Section 7.1.5.

Data Types: `double` | `int8` | `logical`

ncellid — Physical layer cell identity number
integer

Physical layer cell identity number, specified as an integer from 0 to 1007.

Data Types: double

v — Scrambling sequence phase
integer from 0 to 7

Scrambling sequence phase, specified as an integer from 0 to 7. v is derived in a synchronization signal (SS) burst configuration, from the least significant bits (LSBs) of the SS/PBCH block index.

- If the number of SS/PBCH blocks per half-frame is 4, then v is the two LSBs of the SS/PBCH block index (0 to 3).
- If the number of SS/PBCH blocks per half-frame is 8 or 64, then v is the three LSBs of the SS/PBCH block index (0 to 7).

Data Types: double

datatype — Data type of output symbols
'double' (default) | 'single'

Data type of the output symbols, specified as 'double' or 'single'.

Data Types: char | string

Output Arguments

sym — PBCH modulation symbols
complex column vector

PBCH modulation symbols, returned as a complex column vector.

Data Types: single | double

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.212. "NR; Multiplexing and channel coding." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The datatype input argument must be compile-time constant. For example, when specifying 'single' as the output data type, include {coder.Constant('OutputDataType'),coder.Constant('single')} in the -args value of the codegen function. For more information, see the coder.Constant class.

See Also

Functions
nrPBCHIndices | nrPBCHDecode | nrPBCHDMRS | nrPRBS | nrPBCHPRBS | nrPBCHDMRSIndices | nrPSS | nrSSS

nrPBCHDecode

Decode PBCH modulation symbols

Syntax

```
cw = nrPBCHDecode(sym,ncellid,v)
cw = nrPBCHDecode(sym,ncellid,v,nVar)
```

Description

`cw = nrPBCHDecode(sym,ncellid,v)` returns a vector of soft bits `cw` resulting from performing the inverse of the physical broadcast channel (PBCH) processing defined in TS 38.211 Section 7.3.3 [1]. `sym` specifies the received PBCH symbols, `ncellid` is the physical layer cell identity number, and `v` specifies the scrambling sequence phase.

`cw = nrPBCHDecode(sym,ncellid,v,nVar)` specifies the noise variance scaling factor of the soft bits in the PBCH demodulation.

Examples

Demodulate Physical Broadcast Channel Symbols

Consider the first Synchronization Signal / Physical Broadcast Channel (SS/PBCH) block in a burst and assume that the number of SS/PBCH blocks per half-frame is 4.

```
ssbindex = 0;
v = mod(ssbindex,4);
```

Generate a random sequence of binary values that represent encoded BCH bits. The length of the random sequence corresponds to the PBCH bit capacity as specified in TS 38.212 Section 7.1.5.

```
E = 864;
cw = randi([0 1],E,1);
```

Generate the sequence of 432 PBCH quadrature phase shift keying (QPSK) modulation symbols.

```
ncellid = 17;
sym = nrPBCH(cw,ncellid,v);
```

Create bit estimates by demodulating the PBCH symbols. Compare the result with the original input by casting the bit estimates to logical values.

```
rxcw = nrPBCHDecode(sym,ncellid,v);
isequal(cw,rxcw<0)
```

```
ans = logical
   1
```

Input Arguments

`sym` — Received PBCH modulation symbols
complex column vector

Received PBCH modulation symbols, specified as a complex column vector.

Data Types: `single` | `double`
Complex Number Support: Yes

`ncellid` — Physical layer cell identity number
integer

Physical layer cell identity number, specified as an integer from 0 to 1007.

Data Types: `double`

`v` — Scrambling sequence phase
integer from 0 to 7

Scrambling sequence phase, specified as an integer from 0 to 7. `v` is derived in a synchronization signal (SS) burst configuration, from the least significant bits (LSBs) of the SS/PBCH block index.

- If the number of SS/PBCH blocks per half-frame is 4, then `v` is the two LSBs of the SS/PBCH block index (0 to 3).
- If the number of SS/PBCH blocks per half-frame is 8 or 64, then `v` is the three LSBs of the SS/PBCH block index (0 to 7).

Data Types: `double`

`nVar` — Noise variance
`1e-10` (default) | nonnegative numeric scalar

Noise variance, specified as a nonnegative numeric scalar. The soft bits are scaled with the variance of additive white Gaussian noise (AWGN). The default value corresponds to an SNR of 100 dB, assuming unit signal power.

Note The default value assumes the decoder and coder are connected back-to-back, where the noise variance is zero. To avoid `-Inf` or `+Inf` values in the output, the function uses `1e-10` as the default value for noise variance. To get appropriate results when the signal is transmitted through a noisy channel, adjust the noise variance accordingly.

Data Types: `double`

Output Arguments

`cw` — Approximate LLR soft bits
column vector of binary values

Approximate log likelihood ratio (LLR) soft bits, returned as a column vector of binary values. The length of `cw` is twice the length of the input `sym`.

Data Types: `double` | `single`

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.212. "NR; Multiplexing and channel coding." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrPBCHIndices` | `nrPBCH` | `nrPBCHPRBS` | `nrPBCHDMRS` | `nrPBCHDMRSIndices` | `nrPRBS` | `nrPSS` | `nrSSS`

nrPBCHDMRS

Generate PBCH DM-RS symbols

Syntax

```
sym = nrPBCHDMRS(ncellid,ibar_SSB)
sym = nrPBCHDMRS(ncellid,ibar_SSB,'OutputDataType',datatype)
```

Description

`sym = nrPBCHDMRS(ncellid,ibar_SSB)` returns the physical broadcast channel (PBCH) demodulation reference signal (DM-RS) symbols for the physical layer cell, identified by `ncellid`. The `ibar_SSB` input specifies the time-dependent part of the DM-RS scrambling initialization. The function implements TS 38.211 Section 7.4.1.4.1 [1].

`sym = nrPBCHDMRS(ncellid,ibar_SSB,'OutputDataType',datatype)` specifies the data type of the DM-RS symbol.

Examples

Generate PBCH DM-RS Symbols

Generate the sequence of 144 PBCH DM-RS symbols associated with the third SS block (`i_SSB` = 2) in the second half frame (`n_hf` = 1) of a frame.

```
ncellid = 17;
i_SSB = 2;
n_hf = 1;
ibar_SSB = i_SSB + (4*n_hf);

dmrs = nrPBCHDMRS(ncellid,ibar_SSB);
```

Input Arguments

`ncellid` — Physical layer cell identity number
integer

Physical layer cell identity number, specified as an integer from 0 to 1007.

Data Types: `double`

`ibar_SSB` — Time-dependent part of DM-RS scrambling initialization
integer from 0 to 7 (default)

Time-dependent part of the DM-RS scrambling initialization, specified as an integer from 0 to 7. `ibar_SSB` is derived in a synchronization signal (SS) burst configuration, from the least significant bits (LSBs) of the SS/PBCH block index and the half-frame number.

- If the number of SS/PBCH blocks per half-frame is 4, ibar_SSB = $i_{SSB} + 4 \times n_{hf}$, where i_{SSB} is the two LSBs of the SS/PBCH block index (0 to 3). n_{hf} is the half-frame number within the frame (0,1).
- If the number of SS/PBCH blocks per half-frame is 8 or 64, ibar_SSB is the three LSBs of the SS/PBCH block index (0 to 7).

Data Types: double

datatype — Data type of output symbols
'double' (default) | 'single'

Data type of the output symbols, specified as 'double' or 'single'.

Data Types: char | string

Output Arguments

sym — PBCH DM-RS symbols
complex column vector

PBCH DM-RS symbols, returned as a complex column vector.

Data Types: single | double

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The datatype input argument must be compile-time constant. For example, when specifying 'single' as the output data type, include {coder.Constant('OutputDataType'),coder.Constant('single')} in the -args value of the codegen function. For more information, see the coder.Constant class.

See Also

Functions
nrPBCH | nrPSS | nrSSS | nrSSBMeasurements | nrPBCHDMRSIndices | nrPRBS | nrSymbolModulate

nrPBCHDMRSIndices

Generate PBCH DM-RS resource element indices

Syntax

```
ind = nrPBCHDMRSIndices(ncellid)
ind = nrPBCHDMRSIndices(ncellid,Name,Value)
```

Description

`ind = nrPBCHDMRSIndices(ncellid)` returns the resource element indices for the physical broadcast channel (PBCH) demodulation reference signal (DM-RS). The function implements TS 38.211 Section 7.4.3.1 [1]. The corresponding physical layer cell is identified by `ncellid`. The returned indices are 1-based using linear indexing form. This indexing form can directly index the elements of a 240-by-4 matrix corresponding to the Synchronization Signal / Physical Broadcast Channel (SS/PBCH) block. The order of the indices indicates how the PBCH DM-RS modulation symbols are mapped.

`ind = nrPBCHDMRSIndices(ncellid,Name,Value)` specifies additional index formatting options by using one or more name-value pair arguments. Unspecified options take default values.

Examples

Get PBCH DM-RS Resource Element Indices

Generate the 144 resource element indices associated with the PBCH DM-RS symbols within a single SS/PBCH block for a given cell identity.

```
ncellid = 17;
indices = nrPBCHDMRSIndices(ncellid)
```

```
indices = 144x1 uint32 column vector

   242
   246
   250
   254
   258
   262
   266
   270
   274
   278
     ⋮
```

Input Arguments

`ncellid` — Physical layer cell identity number
integer

Physical layer cell identity number, specified as an integer from 0 to 1007.

Data Types: `double`

Name-Value Pair Arguments

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

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

Example: `'IndexStyle','subscript','IndexBase','0based'` specifies nondefault resource element index formatting properties.

`IndexStyle` — RE indexing form
`'index'` (default) | `'subscript'`

RE indexing form, specified as one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

`IndexBase` — RE indexing base
`'1based'` (default) | `'0based'`

RE indexing base, specified as one of these values:

- `'1based'` — The index counting starts from 1.
- `'0based'` — The index counting starts from 0.

Data Types: `char` | `string`

Output Arguments

`ind` — PBCH DM-RS resource element indices
column vector | *M*-by-3 matrix

PBCH DM-RS resource element indices, returned as one of the following.

- Column vector — When `'IndexStyle'` is `'index'`.
- *M*-by-3 matrix — When `'IndexStyle'` is `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers and OFDM symbols in a SS/PBCH block, and the number of antennas, respectively.

Depending on `'IndexBase'`, the indices are either 1-based or 0-based.

Data Types: `uint32`

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrPBCHDMRS` | `nrPBCHIndices` | `nrPSSIndices` | `nrSSBMeasurements` | `nrSSSIndices`

nrPBCHIndices

Generate PBCH resource element indices

Syntax

```
[ind,info] = nrPBCHIndices(ncellid)
[ind,info] = nrPBCHIndices(ncellid,Name,Value)
```

Description

`[ind,info] = nrPBCHIndices(ncellid)` returns the resource element indices `ind` for the physical broadcast channel (PBCH) and related index information `info`. The function implements TS 38.211 Section 7.4.3.1 [1]. The corresponding physical layer cell identity number is `ncellid`. The returned indices are 1-based using linear indexing form. This indexing form can directly index the elements of a 240-by-4 matrix corresponding to the Synchronization Signal / Physical Broadcast Channel (SS/PBCH) block. The order of the indices indicates how the PBCH modulation symbols are mapped.

`[ind,info] = nrPBCHIndices(ncellid,Name,Value)` specifies additional index formatting options by using one or more name-value pair arguments. Unspecified options take default values.

Examples

Get PBCH Resource Element Indices

Generate the 432 resource element indices associated with the PBCH symbols within a single SS/PBCH block for a given cell identity.

```
ncellid = 17;
indices = nrPBCHIndices(ncellid);
```

Input Arguments

`ncellid` — Physical layer cell identity number
integer

Physical layer cell identity number, specified as an integer from 0 to 1007.

Data Types: `double`

Name-Value Pair Arguments

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

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

Example: 'IndexStyle','subscript','IndexBase','0based' specifies nondefault resource element index formatting properties.

IndexStyle — RE indexing form
'index' (default) | 'subscript'

RE indexing form, specified as one of these values:

- 'index' — The indices are in linear index form.
- 'subscript' — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: char | string

IndexBase — RE indexing base
'1based' (default) | '0based'

RE indexing base, specified as one of these values:

- '1based' — The index counting starts from 1.
- '0based' — The index counting starts from 0.

Data Types: char | string

Output Arguments

ind — PBCH resource element indices
column vector | *M*-by-3 matrix

PBCH resource element indices, returned as one of the following.

- column vector — When 'IndexStyle' is 'index'.
- *M*-by-3 matrix — When 'IndexStyle' is 'subscript'. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers and OFDM symbols in a SS/PBCH block, and the number of antennas, respectively.

Depending on 'IndexBase', the indices are either 1-based or 0-based.

Data Types: uint32

info — Characteristic information about PBCH indices
structure

Characteristic information about PBCH indices, returned as a structure with the following fields.

Parameter Field	Value	Description
G	864	Number of coded and rate matched PBCH data bits.
Gd	432	Number of coded and rate matched PBCH data symbols. Gd is equal to the number of rows in the PBCH indices.

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrPBCH` | `nrPBCHDMRSIndices` | `nrPSSIndices` | `nrSSSIndices`

nrPBCHPRBS

Generate PBCH scrambling sequence

Syntax

```
[seq,cinit] = nrPBCHPRBS(ncellid,v,n)
[seq,cinit] = nrPBCHPRBS(ncellid,v,n,Name,Value)
```

Description

`[seq,cinit] = nrPBCHPRBS(ncellid,v,n)` returns the first `n` elements of the physical broadcast channel (PBCH) scrambling sequence. The pseudorandom binary sequence (PRBS) generator is initialized with the physical layer cell identity number `ncellid` and scrambling sequence phase `v`. The function implements TS 38.211 Section 7.3.3.1 [1]. The function also returns the initialization value `cinit` for the PRBS generator.

`[seq,cinit] = nrPBCHPRBS(ncellid,v,n,Name,Value)` specifies additional output formatting options by using one or more name-value pair arguments. Unspecified options take their default values.

Examples

Generate PBCH Scrambling Sequence

Generate the first 864 outputs of the PBCH scrambling sequence initialized with the specified physical layer cell identity number. The specified length of 864 corresponds to the PBCH bit capacity as specified in TS 38.212 Section 7.1.5. Consider the 43rd Synchronization Signal / Physical Broadcast Channel (SS/PBCH) block in a burst. Assume that the number of SS/PBCH blocks per half-frame is 64.

```
ncellid = 17;
ssbindex = 42;
v = mod(ssbindex,8); % assume L_max = 64
E = 864;
seq = nrPBCHPRBS(ncellid,v,E);
```

Input Arguments

`ncellid` — Physical layer cell identity number
integer

Physical layer cell identity number, specified as an integer from 0 to 1007.

Data Types: `double`

`v` — Scrambling sequence phase
integer from 0 to 7

Scrambling sequence phase, specified as an integer from 0 to 7. `v` is derived in a synchronization signal (SS) burst configuration, from the least significant bits (LSBs) of the SS/PBCH block index.

- If the number of SS/PBCH blocks per half-frame is 4, then `v` is the two LSBs of the SS/PBCH block index (0 to 3).
- If the number of SS/PBCH blocks per half-frame is 8 or 64, then `v` is the three LSBs of the SS/PBCH block index (0 to 7).

Data Types: `double`

n — Number of elements in output sequence
nonnegative integer

Number of elements in output sequence, specified as a nonnegative integer.

Data Types: `double`

Name-Value Pair Arguments

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

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

Example: `'MappingType','signed'` specifies nondefault output sequence formatting.

`MappingType` — Output sequence formatting
`'binary'` (default) | `'signed'`

Output sequence formatting, specified as the comma-separated pair consisting of `'MappingType'` and one of these values:

- `'binary'` — This value maps `true` to `1` and `false` to `0`. The data type of the output sequence is `logical`.
- `'signed'` — This value maps `true` to `–1` and `false` to `1`. The data type of the output sequence is `double`. To specify `single` data type, use the `'OutputDataType'` name-value pair.

Data Types: `char` | `string`

`OutputDataType` — Data type of output sequence
`'double'` (default) | `'single'`

Data type of output sequence, specified as the comma-separated pair consisting of `'OutputDataType'` and `'double'` or `'single'`. This name-value pair applies only when `'MappingType'` is set to `'signed'`.

Data Types: `char` | `string`

Output Arguments

seq — PBCH scrambling sequence
logical column vector | numeric column vector

PBCH scrambling sequence, returned as a logical or numeric column vector. The output `seq` contains the first `n` elements of the PBCH scrambling sequence. If you set `'MappingType'` to `'signed'`, the data type of `seq` is either `double` or `single`. If you set `'MappingType'` to `'binary'`, the data type of `seq` is `logical`.

Data Types: `double` | `single` | `logical`

`cinit` — Initialization value for PRBS generator
nonnegative integer from 0 to 1007

Initialization value for PRBS generator, returned as a nonnegative integer from 0 to 1007. `cinit` is the same value as `ncellid`.

Data Types: `double`

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying `'single'` as the output data type, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrPBCH` | `nrPBCHDecode` | `nrPBCHIndices` | `nrPRBS`

nrPDCCH

Generate PDCCH modulation symbols

Syntax

```
sym = nrPDCCH(dcicw,nid,nrnti)
sym = nrPDCCH( ___ ,'OutputDataType',datatype)
```

Description

`sym = nrPDCCH(dcicw,nid,nrnti)` returns the physical downlink control channel (PDCCH) modulation symbols, as defined in TS 38.211 Section 7.3.2 [1]. `dcicw` is the encoded downlink control information (DCI) codeword, as specified in TS 38.212 Section 7.3 [2]. The generation process consists of scrambling the input DCI codeword with scrambling identity `nid`, and QPSK symbol modulation. `nrnti` specifies the user equipment (UE).

`sym = nrPDCCH( ___ ,'OutputDataType',datatype)` specifies the PDCCH symbol data type in addition to the input arguments in the previous syntax.

Examples

Generate PDCCH Modulation Symbols Using DMRS Scrambling Identity

Specify a random sequence of binary values corresponding to a DCI codeword of 560 bits. Generate modulation symbols by scrambling with the PDCCH demodulation reference signal (DMRS) scrambling identity.

```
dcicw = randi([0 1],560,1);
nid = 2^11;                    % pdcch-DMRS-ScramblingID
nrnti = 123;                   % C-RNTI
sym = nrPDCCH(dcicw,nid,nrnti)
```

```
sym = 280×1 complex

   0.7071 + 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 - 0.7071i
  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
  -0.7071 - 0.7071i
      ⋮
```

Generate PDCCH Modulation Symbols Using NcellID for Scrambling

Specify a random sequence of binary values corresponding to a DCI codeword of 560 bits. Generate PDCCH modulation symbols by setting the scrambling identity to the physical layer cell identity (NcellID).

```
dcicw = randi([0 1],560,1);
nid = 123;                   % NcellID (0 to 1007)
nrnti = 0;
sym = nrPDCCH(dcicw,nid,nrnti)
```

```
sym = 280×1 complex

  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
   0.7071 + 0.7071i
  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 - 0.7071i
   0.7071 + 0.7071i
      ⋮
```

Input Arguments

dcicw — Encoded DCI codeword
column vector of binary values

Encoded DCI codeword, specified as a column vector of binary values.

Data Types: `double` | `int8` | `logical`

nid — Scrambling identity
integer from 0 to 65,535

Scrambling identity, specified as an integer from 0 to 65,535. Specify with `nid` the physical layer cell identity number, ranging from 0 to 1007, or higher layer parameter *pdcch-DMRS-ScramblingID*, ranging from 0 to 65,535. For more information on these values, see TS 38.211 Section 7.3.2.3 and 7.4.1.3.

Data Types: `double`

nrnti — UE identifier
integer from 0 to 65,519

UE identifier, specified as an integer from 0 to 65,519.

- If `nid` is the PDCCH DMRS scrambling identity, `nrnti` is the cell radio network temporary identifier (C-RNTI) in a UE-specific search space.
- If `nid` is the physical layer cell identity, `nrnti` is `0`.

For more information, TS 38.211 Section 7.3.2.3 and 7.4.1.3.

Data Types: `double`

`datatype` — Data type of output symbols
`'double'` (default) | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

`sym` — PDCCH modulation symbols
complex column vector

PDCCH modulation symbols, returned as a complex column vector.

Data Types: `single` | `double`
Complex Number Support: Yes

Version History

Introduced in R2018b

Updates to upper limit of UE identifier
Behavior changed in R2020b

Starting in R2020b, the upper limit of UE identifier `nrnti` is 65,519 instead of 65,535.

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.212. "NR; Multiplexing and channel coding." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The `datatype` input argument must be compile-time constant. For example, when specifying `'single'` as the output data type, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPDCCHDecode` | `nrPDCCHPRBS` | `nrDCIDecode` | `nrDCIEncode` | `nrPDCCHResources`

nrPDCCHDecode

Decode PDCCH modulation symbols

Syntax

```
dcicw = nrPDCCHDecode(sym,nid,nrnti)
dcicw = nrPDCCHDecode(sym,nid,nrnti,nVar)
```

Description

`dcicw = nrPDCCHDecode(sym,nid,nrnti)` returns the soft bits resulting from the inverse operation of the physical downlink control channel (PDCCH) processing specified in TS 38.211 Section 7.3.2 [1]. The decoding consists of the QPSK demodulation of `sym`, and descrambling with the scrambling identity `nid`. The argument `nrnti` specifies the user equipment (UE).

`dcicw = nrPDCCHDecode(sym,nid,nrnti,nVar)` specifies the noise variance scaling factor of the soft bits in the PDCCH demodulation.

Examples

Decode PDCCH Modulation Symbols

Specify a random sequence of binary values corresponding to a DCI codeword of 560 bits. Generate PDCCH modulation symbols by scrambling with the PDCCH demodulation reference signal (DMRS) scrambling identity. Specify the user equipment by using the cell radio network temporary identifier.

```
dcicw = randi([0 1],560,1);
nid = 2^11;                  % pdcch-DMRS-ScramblingID
nrnti = 123;                 % C-RNTI
sym = nrPDCCH(dcicw,nid,nrnti)
```

```
sym = 280×1 complex

   0.7071 + 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 - 0.7071i
  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
  -0.7071 - 0.7071i
      ⋮
```

Demodulate and compare the soft bits with the input codeword.

```
nVar = 0;
rxdcicw = nrPDCCHDecode(sym,nid,nrnti,nVar);
isequal(dcicw,rxdcicw<0)
```

```
ans = logical
   1
```

Input Arguments

`sym` — Received PDCCH modulation symbols

complex column vector

Received PDCCH modulation symbols, specified as a complex column vector.

Data Types: `single` | `double`

`nid` — Scrambling identity

integer from 0 to 65,535

Scrambling identity, specified as an integer from 0 to 65,535. Specify with `nid` the physical layer cell identity number, ranging from 0 to 1007, or higher layer parameter *pdcch-DMRS-ScramblingID*, ranging from 0 to 65,535. For more information on these values, see TS 38.211 Section 7.3.2.3 and 7.4.1.3.

Data Types: `double`

`nrnti` — UE identifier

integer from 0 to 65,519

UE identifier, specified as an integer from 0 to 65,519.

- If `nid` is the PDCCH DMRS scrambling identity, `nrnti` is the cell radio network temporary identifier (C-RNTI) in a UE-specific search space.
- If `nid` is the physical layer cell identity, `nrnti` is `0`.

For more information, TS 38.211 Section 7.3.2.3 and 7.4.1.3.

Data Types: `double`

`nVar` — Noise variance

`1e-10` (default) | nonnegative numeric scalar

Noise variance, specified as a nonnegative numeric scalar. The soft bits are scaled with the variance of additive white Gaussian noise (AWGN). The default value corresponds to an SNR of 100 dB, assuming unit signal power.

Note The default value assumes the decoder and coder are connected back-to-back, where the noise variance is zero. To avoid `-Inf` or `+Inf` values in the output, the function uses `1e-10` as the default value for noise variance. To get appropriate results when the signal is transmitted through a noisy channel, adjust the noise variance accordingly.

Data Types: `double`

Output Arguments

dcicw — Approximate LLR soft bits
column vector of real numbers

Approximate log-likelihood ratio (LLR) soft bits, returned as a column vector of real numbers. `dcicw` inherits the data type of `sym`.

Data Types: `double` | `single`

Version History

Introduced in R2018b

Updates to upper limit of UE identifier
Behavior changed in R2020b

Starting in R2020b, the upper limit of UE identifier `nrnti` is 65,519 instead of 65,535.

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrPDCCH` | `nrPDCCHPRBS` | `nrDCIDecode` | `nrDCIEncode` | `nrPDCCHSpace` | `nrPDCCHResources`

nrPDCCHPRBS

Generate PDCCH scrambling sequence

Syntax

```
[seq,cinit] = nrPDCCHPRBS(nid,nrnti,n)
[seq,cinit] = nrPDCCHPRBS(nid,nrnti,n,Name,Value)
```

Description

`[seq,cinit] = nrPDCCHPRBS(nid,nrnti,n)` returns the first `n` elements of the physical downlink control channel (PDCCH) scrambling sequence. The function also returns the initialization value `cinit` of the pseudorandom binary sequence (PRBS) generator. The initialization value depends on the scrambling identity number `nid` and the user equipment (UE) identifier `nrnti`. The function implements TS 38.211 Section 7.3.2.3 [1].

`[seq,cinit] = nrPDCCHPRBS(nid,nrnti,n,Name,Value)` specifies additional output formatting options by using one or more name-value pair arguments. Unspecified options take their default values.

Examples

Generate PDCCH Scrambling Sequence Using DMRS Scrambling Identity

Generate the first 100 elements of the PDCCH scrambling sequence. The PDCCH demodulation reference signal (DMRS) scrambling identity and the cell radio network temporary identifier determine the initialization value.

```
n = 100;
nid = 10;                 % pdcch-DMRS-ScramblingID
nrnti = 20;               % C-RNTI
seq = nrPDCCHPRBS(nid,nrnti,n);
```

Generate PDCCH Scrambling Sequence Using *NcellID*

Generate the first 120 elements of the PDCCH scrambling sequence initialized with the physical layer cell identity number (*NcellID*).

```
n = 120;
nid = 123;                    % NcellID (0 to 1007)
nrnti = 0;
seq = nrPDCCHPRBS(nid,nrnti,n);
```

Input Arguments

nid — Scrambling identity
integer from 0 to 65,535

Scrambling identity, specified as an integer from 0 to 65,535. Specify with `nid` the physical layer cell identity number, ranging from 0 to 1007, or higher layer parameter *pdcch-DMRS-ScramblingID*, ranging from 0 to 65,535. For more information on these values, see TS 38.211 Section 7.3.2.3 and 7.4.1.3.

Data Types: `double`

`nrnti` — UE identifier
integer from 0 to 65,519

UE identifier, specified as an integer from 0 to 65,519.

- If `nid` is the PDCCH DMRS scrambling identity, `nrnti` is the cell radio network temporary identifier (C-RNTI) in a UE-specific search space.
- If `nid` is the physical layer cell identity, `nrnti` is `0`.

For more information, TS 38.211 Section 7.3.2.3 and 7.4.1.3.

Data Types: `double`

`n` — Number of elements in output sequence
nonnegative integer

Number of elements in output sequence, specified as a nonnegative integer.

Data Types: `double`

Name-Value Pair Arguments

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

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

Example: `'MappingType','signed'` specifies nondefault output sequence formatting.

`MappingType` — Output sequence formatting
`'binary'` (default) | `'signed'`

Output sequence formatting, specified as the comma-separated pair consisting of `'MappingType'` and one of these values:

- `'binary'` — This value maps `true` to `1` and `false` to `0`. The data type of the output sequence is `logical`.
- `'signed'` — This value maps `true` to `–1` and `false` to `1`. The data type of the output sequence is `double`. To specify `single` data type, use the `'OutputDataType'` name-value pair.

Data Types: `char` | `string`

`OutputDataType` — Data type of output sequence
`'double'` (default) | `'single'`

Data type of output sequence, specified as the comma-separated pair consisting of `'OutputDataType'` and `'double'` or `'single'`. This name-value pair applies only when `'MappingType'` is set to `'signed'`.

Data Types: char | string

Output Arguments

seq — PDCCH scrambling sequence
logical column vector | numeric column vector

PDCCH scrambling sequence, returned as a logical or numeric column vector. seq contains the first n elements of the PDCCH scrambling sequence. If you set 'MappingType' to 'signed', the output data type is either double or single. If you set 'MappingType' to 'binary', the output data type is logical.

Data Types: double | single | logical

cinit — Initialization value for PRBS generator
nonnegative integer

Initialization value for the PRBS generator, returned as a nonnegative integer.

Data Types: double

Version History

Introduced in R2018b

Updates to upper limit of UE identifier
Behavior changed in R2020b

Starting in R2020b, the upper limit of UE identifier nrnti is 65,519 instead of 65,535.

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying 'single' as the output data type, include {coder.Constant('OutputDataType'),coder.Constant('single')} in the -args value of the codegen function. For more information, see the coder.Constant class.

See Also

Functions
nrPDCCH | nrPDCCHDecode | nrPRBS

nrPDCCHResources

Generate PDCCH and PDCCH DM-RS resources

Syntax

```
ind = nrPDCCHResources(carrier,pdcch)
[ind,dmrsSym,dmrsInd] = nrPDCCHResources(carrier,pdcch)
[ind,dmrsSym,dmrsInd] = nrPDCCHResources(carrier,pdcch,Name,Value)
```

Description

`ind = nrPDCCHResources(carrier,pdcch)` returns physical downlink control channel (PDCCH) resource element indices `ind`, as defined in TS 38.211 Section 7.3.2 [1]. The input `carrier` specifies carrier configuration parameters for a specific OFDM numerology. The input `pdcch` specifies PDCCH configuration parameters.

`[ind,dmrsSym,dmrsInd] = nrPDCCHResources(carrier,pdcch)` also returns PDCCH demodulation reference signal (DM-RS) symbols `dmrsSym` and PDCCH DM-RS resource element indices `dmrsInd`, as defined in TS 38.211 Section 7.4.1.3.

`[ind,dmrsSym,dmrsInd] = nrPDCCHResources(carrier,pdcch,Name,Value)` specifies output formatting options using one or more name-value pair arguments.

Examples

Generate and Map PDCCH Symbols to Carrier Grid

Configure the carrier and the PDCCH with default configuration parameters.

```
carrier = nrCarrierConfig;
pdcch = nrPDCCHConfig;
```

Generate PDCCH symbols for a random DCI codeword by using PDCCH configuration parameters for scrambling and identifying the UE.

```
dciCW = randi([0 1],864,1);
sym = nrPDCCH(dciCW,pdcch.DMRSScramblingID,pdcch.RNTI);
```

Generate PDCCH resource element indices by using the specified carrier and PDCCH objects.

```
ind = nrPDCCHResources(carrier,pdcch);
```

Create a grid for mapping PDCCH symbols to the grid.

```
cgrid = zeros(12*carrier.NSizeGrid,carrier.SymbolsPerSlot);
```

Map PDCCH symbols to the grid.

```
cgrid(ind) = sym;
```

Generate PDCCH DM-RS Symbols and Indices

Configure the carrier with default configuration parameters.

```
carrier = nrCarrierConfig;
```

Configure the CORESET with 6 frequency resources, a duration of 3 OFDM symbols, and a REG bundle size of 3.

```
crst = nrCORESETConfig;
crst.FrequencyResources = ones(1,6);
crst.Duration = 3;
crst.REGBundleSize = 3;
```

Configure the PDCCH with the specified bandwidth part and CORESET.

```
pdcch = nrPDCCHConfig;
pdcch.NStartBWP = 6;
pdcch.NSizeBWP = 36;
pdcch.CORESET = crst;
pdcch.AggregationLevel = 16;
```

Generate PDCCH DM-RS symbols and indices for the specified carrier and PDCCH.

```
[~,dmrs,dmrsInd] = nrPDCCHResources(carrier,pdcch);
```

Generate PDCCH and DM-RS Indices Relative to BWP Grid

Configure a carrier grid of 60 resource blocks (RBs), where the starting RB index relative to the common resource block 0 (CRB 0) is 3.

```
carrier = nrCarrierConfig;
carrier.NStartGrid = 3;
carrier.NSizeGrid = 60;
```

Configure noninterleaved CORESET with 6 frequency resources and a duration of 3 OFDM symbols.

```
crst = nrCORESETConfig;
crst.FrequencyResources = ones(1,6);
crst.Duration = 3;
crst.CCEREGMapping = 'noninterleaved';
```

Configure the PDCCH with the specified bandwidth part and CORESET.

```
pdcch = nrPDCCHConfig;
pdcch.NStartBWP = 5;
pdcch.NSizeBWP = 48;
pdcch.CORESET = crst;
pdcch.AggregationLevel = 16;
```

Generate PDCCH resource element indices and DM-RS symbol indices using 1-based, subscript indexing form relative to the BWP grid.

```
[ind,~,dmrsInd] = nrPDCCHResources(carrier,pdcch,...
    'IndexOrientation','bwp','IndexStyle','subscript');
```

Input Arguments

carrier — Carrier configuration parameters
nrCarrierConfig object

Carrier configuration parameters for a specific OFDM numerology, specified as an nrCarrierConfig object.

pdcch — PDCCH configuration parameters
nrPDCCHConfig object

PDCCH configuration parameters, specified as an nrPDCCHConfig object.

Name-Value Pair Arguments

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

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

Example: 'IndexStyle','subscript','IndexBase','0based' specifies the indexing form and indexing base of the output.

IndexStyle — RE indexing form
'index' (default) | 'subscript'

RE indexing form, specified as one of these values:

- 'index' — The indices are in linear index form.
- 'subscript' — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: char | string

IndexBase — RE indexing base
'1based' (default) | '0based'

RE indexing base, specified as one of these values:

- '1based' — The index counting starts from 1.
- '0based' — The index counting starts from 0.

Data Types: char | string

IndexOrientation — Indexing orientation of PDCCH and DM-RS resource elements
'carrier' (default) | 'bwp'

Indexing orientation of PDCCH and DM-RS resource elements, specified as the comma-separated pair consisting of 'IndexOrientation' and one of these values:

- 'carrier' — Indices are referenced with respect to the carrier grid.

- `'bwp'` — Indices are referenced with respect to the bandwidth part.

Data Types: `char` | `string`

`OutputDataType` — Data type of PDCCH DM-RS symbols
`'double'` (default) | `'single'`

Data type of PDCCH DM-RS symbols, specified as the comma-separated pair consisting of `'OutputDataType'` and one of these values:

- `'double'` — Output symbols are of `double` data type.
- `'single'` — Output symbols are of `single` data type.

Data Types: `char` | `string`

Output Arguments

`ind` — PDCCH resource element indices
M-by-1 vector (default) | *M*-by-3 matrix

PDCCH resource element indices, returned as one of these values:

- *M*-by-1 vector — When '`IndexStyle`' is set to `'index'`.
- *M*-by-3 matrix — When '`IndexStyle`' is set to `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and number of antennas, respectively.

M depends on the aggregation level of the PDCCH and is equal to `pdcch.AggregationLevel` × 6 × 12 × 3 / 4.

Depending on the value of '`IndexBase`', the indices are either 1-based or 0-based.

Data Types: `uint32`

`dmrsSym` — DM-RS symbols
N-by-1 complex vector

DM-RS symbols, returned as an *N*-by-1 complex vector. *N* depends on the aggregation level of the PDCCH and is equal to `pdcch.AggregationLevel` × 6 × 12 × 1 / 4.

Data Types: `single` | `double`

`dmrsInd` — DM-RS resource element indices
N-by-1 vector (default) | *N*-by-3 matrix

DM-RS resource element indices, returned as one of these values:

- *N*-by-1 vector — When '`IndexStyle`' is set to `'index'`.
- *N*-by-3 matrix — When '`IndexStyle`' is set to `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and number of antennas, respectively.

N depends on the aggregation level of the PDCCH and is equal to `pdcch.AggregationLevel` × 6 × 12 × 1 / 4.

Depending on the value of 'IndexBase', the indices are either 1-based or 0-based.

Data Types: uint32

Version History

Introduced in R2020a

DM-RS reference point update for CORESET ID 0
Behavior changed in R2020b

Starting in R2020b, the reference point for the DM-RS sequence-to-subcarrier resource mapping for CORESET ID 0 is the lowest physical resource block of the CORESET instead of CRB 0. This update affects the resources that you generate with this function for CORESET ID 0. For all other CORESET ID values, the reference point remains CRB0.

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.213. “NR; Physical layer procedures for control.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPDCCH` | `nrPDCCHSpace`

Objects
`nrCORESETConfig` | `nrPDCCHConfig` | `nrSearchSpaceConfig` | `nrCarrierConfig`

Topics
“Downlink Control Processing and Procedures”

nrPDCCHSpace

Generate PDCCH resources for all candidates and aggregation levels

Syntax

```
allInd = nrPDCCHSpace(carrier,pdcch)
[allInd,allDMRSSym,allDMRSInd] = nrPDCCHSpace(carrier,pdcch)
[allInd,allDMRSSym,allDMRSInd] = nrPDCCHSpace(carrier,pdcch,Name,Value)
```

Description

`allInd = nrPDCCHSpace(carrier,pdcch)` returns physical downlink control channel (PDCCH) resource element indices `allInd` for all candidates at each aggregation level, as defined in TS 38.211 Section 7.3.2 [1]. The input `carrier` specifies carrier configuration parameters for a specific OFDM numerology. The input `pdcch` specifies PDCCH configuration parameters.

`[allInd,allDMRSSym,allDMRSInd] = nrPDCCHSpace(carrier,pdcch)` also returns PDCCH demodulation reference signal (DM-RS) symbols `allDMRSSym` and PDCCH DM-RS resource element indices `allDMRSInd` for all candidates at each aggregation level, as defined in TS 38.211 Section 7.4.1.3.

`[allInd,allDMRSSym,allDMRSInd] = nrPDCCHSpace(carrier,pdcch,Name,Value)` specifies output formatting options using one or more name-value pair arguments.

Examples

Generate PDCCH Indices for All Candidates and Aggregation Levels

Configure the carrier and the PDCCH with default configuration parameters.

```
carrier = nrCarrierConfig;
pdcch = nrPDCCHConfig;
```

Generate all PDCCH resource element indices for all candidates and aggregation levels.

```
allInd = nrPDCCHSpace(carrier,pdcch)
```

```
allInd=5×1 cell array
    { 54x8 uint32}
    {108x8 uint32}
    {216x4 uint32}
    {432x2 uint32}
    {864x1 uint32}
```

Verify that the number of generated candidates for the PDCCH indices at each aggregation level matches the number of candidates specified by the default search space set.

```
numCandidates = [...
    size(allInd{1},2) ...
```

```
    size(allInd{2},2) ...
    size(allInd{3},2) ...
    size(allInd{4},2) ...
    size(allInd{5},2)];
isequaln(pdcch.SearchSpace.NumCandidates,numCandidates)
```

```
ans = logical
   1
```

Generate PDCCH DM-RS Symbols for All Candidates and Aggregation Levels

Configure the carrier with default configuration parameters.

```
carrier = nrCarrierConfig;
```

Configure the CORESET with 6 frequency resources, a duration of 3 OFDM symbols, and a REG bundle size of 3.

```
crst = nrCORESETConfig;
crst.FrequencyResources = ones(1,6);
crst.Duration = 3;
crst.REGBundleSize = 3;
```

Configure the search space set for the PDCCH with the specified number of candidates at each aggregation level.

```
cfgSS = nrSearchSpaceConfig;
cfgSS.NumCandidates = [5 5 3 2 1];
```

Configure the PDCCH with the specified bandwidth part, CORESET, and search space set.

```
pdcch = nrPDCCHConfig;
pdcch.NStartBWP = 6;
pdcch.NSizeBWP = 36;
pdcch.CORESET = crst;
pdcch.SearchSpace = cfgSS;
```

Generate PDCCH DM-RS symbols for all candidates and aggregation levels.

```
[~,allDMRS] = nrPDCCHSpace(carrier,pdcch)
```

```
allDMRS=5×1 cell array
    { 18x5 double}
    { 36x5 double}
    { 72x3 double}
    {144x2 double}
    {288x1 double}
```

Verify that the number of generated candidates for the PDCCH DM-RS symbols at each aggregation level matches the number of candidates specified by the search space set.

```
numCandidates = [...
    size(allDMRS{1},2) ...
    size(allDMRS{2},2) ...
```

```
    size(allDMRS{3},2) ...
    size(allDMRS{4},2) ...
    size(allDMRS{5},2)];
isequaln(cfgSS.NumCandidates,numCandidates)
```

```
ans = logical
   1
```

Generate PDCCH DM-RS Indices for All Candidates and Aggregation Levels

Configure a carrier grid of 60 resource blocks (RBs), where the starting RB index relative to the common resource block 0 (CRB 0) is 3.

```
carrier = nrCarrierConfig;
carrier.NStartGrid = 3;
carrier.NSizeGrid = 60;
```

Configure noninterleaved CORESET with 6 frequency resources and a duration of 3 OFDM symbols.

```
crst = nrCORESETConfig;
crst.FrequencyResources = ones(1,6);
crst.Duration = 3;
crst.CCEREGMapping = 'noninterleaved';
```

Configure the search space set for the PDCCH with the specified number of candidates at each aggregation level.

```
cfgSS = nrSearchSpaceConfig;
cfgSS.NumCandidates = [5 5 3 2 1];
```

Configure the PDCCH with the specified bandwidth part, CORESET, and search space set.

```
pdcch = nrPDCCHConfig;
pdcch.NStartBWP = 5;
pdcch.NSizeBWP = 48;
pdcch.CORESET = crst;
pdcch.SearchSpace = cfgSS;
```

Generate PDCCH DM-RS resource element indices for all candidates and aggregation levels using 1-based, subscript indexing form relative to the BWP grid.

```
[~,~,allDMRSInd] = nrPDCCHSpace(carrier,pdcch, ...
    'IndexOrientation','bwp','IndexStyle','subscript')
```

```
allDMRSInd=5×1 cell array
    { 18x3x5 uint32}
    { 36x3x5 uint32}
    { 72x3x3 uint32}
    {144x3x2 uint32}
    {288x3   uint32}
```

Verify that the number of generated candidates for PDCCH DM-RS indices at each aggregation level matches the number of candidates specified by the search space set.

```
numCandidates = [...
    size(allDMRSInd{1},3) ...
    size(allDMRSInd{2},3) ...
    size(allDMRSInd{3},3) ...
    size(allDMRSInd{4},3) ...
    size(allDMRSInd{5},3)];
isequaln(cfgSS.NumCandidates,numCandidates)
```

```
ans = logical
   1
```

Input Arguments

carrier — Carrier configuration parameters
nrCarrierConfig object

Carrier configuration parameters for a specific OFDM numerology, specified as an nrCarrierConfig object.

pdcch — PDCCH configuration parameters
nrPDCCHConfig object

PDCCH configuration parameters, specified as an nrPDCCHConfig object.

Name-Value Pair Arguments

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

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

Example: 'IndexStyle','subscript','IndexBase','0based' specifies the indexing form and indexing base of the output.

IndexStyle — RE indexing form
'index' (default) | 'subscript'

RE indexing form, specified as one of these values:

- 'index' — The indices are in linear index form.
- 'subscript' — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: char | string

IndexBase — RE indexing base
'1based' (default) | '0based'

RE indexing base, specified as one of these values:

- '1based' — The index counting starts from 1.
- '0based' — The index counting starts from 0.

Data Types: char | string

`IndexOrientation` — Indexing orientation of PDCCH and DM-RS resource elements
`'carrier'` (default) | `'bwp'`

Indexing orientation of PDCCH and DM-RS resource elements, specified as the comma-separated pair consisting of `'IndexOrientation'` and one of these values:

- `'carrier'` — Indices are referenced with respect to the carrier grid.
- `'bwp'` — Indices are referenced with respect to the bandwidth part.

Data Types: `char` | `string`

`OutputDataType` — Data type of PDCCH DM-RS symbols
`'double'` (default) | `'single'`

Data type of PDCCH DM-RS symbols, specified as the comma-separated pair consisting of `'OutputDataType'` and one of these values:

- `'double'` — Output symbols are of `double` data type.
- `'single'` — Output symbols are of `single` data type.

Data Types: `char` | `string`

Output Arguments

`allInd` — PDCCH resource element indices for all candidates at each aggregation level
5-by-1 cell array

PDCCH resource element indices for all candidates at each aggregation level, returned as a 5-by-1 cell array. The five cells correspond to aggregation levels 1, 2, 4, 8, and 16, respectively. Each cell contains a matrix corresponding to all candidates at the appropriate aggregation level. The dimensionality of all matrices is either two or three, depending on the '`IndexStyle`' name-value pair argument. The last dimension of each matrix corresponds to the number of candidates specified by the `pdcch.SearchSpace.NumCandidates` property for the appropriate aggregation level.

Data Types: `uint32`

`allDMRSSym` — PDCCH DM-RS symbols for all candidates at each aggregation level
5-by-1 cell array

PDCCH DM-RS symbols for all candidates at each aggregation level, returned as a 5-by-1 cell array. The five cells correspond to aggregation levels 1, 2, 4, 8, and 16, respectively. Each cell contains a 2-D matrix corresponding to all candidates at the appropriate aggregation level. The number of matrix columns in each cell corresponds to the number of candidates specified by the `pdcch.SearchSpace.NumCandidates` property for the appropriate aggregation level.

Data Types: `single` | `double`

`allDMRSInd` — PDCCH DM-RS resource element indices for all candidates at each aggregation level
5-by-1 cell array

PDCCH DM-RS resource element indices for all candidates at each aggregation level, returned as a 5-by-1 cell array. The five cells correspond to aggregation levels 1, 2, 4, 8, and 16, respectively. Each cell contains a matrix corresponding to all candidates at the appropriate aggregation level. The dimensionality of all matrices is either two or three, depending on the 'IndexStyle' name-value pair argument. The last dimension of each matrix corresponds to the number of candidates specified by the pdcch.SearchSpace.NumCandidates property for the appropriate aggregation level.

Data Types: uint32

Version History

Introduced in R2020a

DM-RS reference point update for CORESET ID 0

Behavior changed in R2020b

Starting in R2020b, the reference point for the DM-RS sequence-to-subcarrier resource mapping for CORESET ID 0 is the lowest physical resource block of the CORESET instead of CRB 0. This update affects the resources that you generate with this function for CORESET ID 0. For all other CORESET ID values, the reference point remains CRB0.

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.213. “NR; Physical layer procedures for control.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying linear indexing form, include {coder.Constant('IndexStyle'),coder.Constant('index')} in the -args value of the codegen function. For more information, see the coder.Constant class.

See Also

Functions

nrPDCCH | nrPDCCHDecode | nrPDCCHResources

Objects

nrCORESETConfig | nrPDCCHConfig | nrSearchSpaceConfig | nrCarrierConfig

Topics

“Downlink Control Processing and Procedures”

nrPDSCH

Generate PDSCH modulation symbols

Syntax

```
sym = nrPDSCH(cws,mod,nlayers,nid,rnti)
sym = nrPDSCH(carrier,pdsch,cws)
sym = nrPDSCH( ___ ,'OutputDataType',datatype)
```

Description

`sym = nrPDSCH(cws,mod,nlayers,nid,rnti)` returns `sym` containing physical downlink shared channel (PDSCH) modulation symbols, as defined in TS 38.211 Sections 7.3.1.1–3 [1]. The process consists of scrambling with scrambling identity `nid`, performing symbol modulation with modulation scheme `mod`, and layer mapping. `cws` represents one or two downlink shared channel (DL-SCH) codewords, as described in TS 38.212 Section 7.2.6. `nlayers` specifies the number of transmission layers. `rnti` is the radio network temporary identifier (RNTI) of the user equipment (UE).

`sym = nrPDSCH(carrier,pdsch,cws)` returns PDSCH modulation symbols for the specified carrier configuration, `carrier`, and PDSCH configuration, `pdsch`. The input `cws` specifies a downlink shared channel (DL-SCH) codeword.

`sym = nrPDSCH( ___ ,'OutputDataType',datatype)` specifies the PDSCH symbol data type, in addition to the input arguments in any of the previous syntaxes.

Examples

Generate PDSCH Symbols for Single Codeword

Specify a random sequence of binary values corresponding to a codeword of 8000 bits using 256-QAM modulation. Generate PDSCH modulation symbols for the specified physical layer cell identity number, RNTI, and number of transmission layers.

```
modulation = '256QAM';
nlayers = 4;
ncellid = 42;
rnti = 6143;
data = randi([0 1],8000,1);
sym = nrPDSCH(data,modulation,nlayers,ncellid,rnti)
```

```
sym = 250×4 complex

  -0.2301 + 0.5369i  -0.3835 + 0.9971i   0.3835 + 1.1504i  -0.2301 + 0.9971i
   0.8437 - 0.0767i  -0.9971 + 0.6903i  -0.6903 - 0.6903i   0.6903 - 0.6903i
   0.2301 - 1.1504i  -0.9971 + 0.0767i   0.6903 - 1.1504i   1.1504 + 0.6903i
  -0.3835 - 1.1504i  -0.0767 - 0.0767i  -0.3835 + 0.3835i  -0.3835 - 0.3835i
   0.9971 + 0.5369i  -0.3835 - 0.5369i   0.3835 - 0.6903i  -0.3835 - 0.8437i
  -0.0767 + 1.1504i   0.6903 - 0.8437i  -0.2301 + 0.2301i   0.8437 - 0.0767i
  -0.3835 - 1.1504i  -0.6903 - 0.9971i   0.9971 - 0.3835i  -0.9971 + 0.0767i
  -0.0767 + 0.6903i  -0.0767 + 0.8437i   1.1504 + 0.0767i   0.6903 + 1.1504i
```

```
  -0.5369 - 0.9971i  -0.8437 + 0.0767i   0.8437 - 0.3835i  -0.9971 - 1.1504i
   0.2301 - 0.6903i  -0.6903 - 0.5369i  -0.6903 + 1.1504i   0.8437 - 0.2301i
      ⋮
```

Generate PDSCH Symbols for Codewords with Different Modulation Scheme

Specify two random sequences of binary values. The first sequence corresponds to a codeword of 6000 bits using 64-QAM modulation. The second sequence corresponds to a codeword of 8000 bits using 256-QAM modulation. Generate PDSCH modulation symbols for the specified physical layer cell identity number and RNTI using a total of 8 transmission layers.

```
modulation = {'64QAM' '256QAM'};
nlayers = 8;
ncellid = 1;
rnti = 6143;
data = {randi([0 1],6000,1) randi([0 1],8000,1)};
sym = nrPDSCH(data,modulation,nlayers,ncellid,rnti)
```

```
sym = 250×8 complex

  -0.4629 - 0.7715i   0.4629 - 0.4629i   0.4629 + 0.1543i   0.7715 - 1.0801i   0.3835 - 0.9971i
   0.1543 + 0.4629i  -1.0801 + 1.0801i  -0.7715 + 0.7715i  -0.1543 + 0.7715i  -0.2301 + 0.9971i
  -0.1543 + 0.1543i   0.7715 - 1.0801i  -0.4629 + 0.7715i   0.1543 + 1.0801i   0.0767 - 0.8437i
  -0.7715 - 0.4629i  -0.1543 + 0.7715i  -0.7715 - 0.7715i  -0.4629 - 0.1543i  -0.6903 + 0.5369i
   1.0801 - 1.0801i  -1.0801 + 0.7715i   0.1543 - 0.4629i   0.4629 - 0.4629i  -1.1504 + 0.2301i
   0.4629 + 0.4629i   0.1543 + 0.1543i  -0.1543 + 0.1543i   0.1543 - 0.4629i   0.6903 + 0.2301i
  -1.0801 + 0.7715i   0.4629 - 1.0801i   0.4629 + 1.0801i  -0.4629 + 0.4629i  -0.6903 + 0.8437i
  -1.0801 + 0.7715i  -0.1543 - 0.1543i   0.7715 + 1.0801i  -0.4629 - 0.1543i   0.8437 + 0.5369i
  -0.4629 - 1.0801i  -0.7715 - 0.1543i   0.1543 - 1.0801i  -0.1543 + 0.1543i   0.2301 - 0.3835i
   0.7715 + 1.0801i   1.0801 - 0.4629i   1.0801 + 1.0801i  -0.1543 - 1.0801i  -0.0767 + 0.0767i
      ⋮
```

Generate PDSCH Symbols and Indices

Create a carrier configuration object with default properties. Specify the physical layer cell identity as 42 and slot number as 10.

```
carrier = nrCarrierConfig;
carrier.NCellID = 42;
carrier.NSlot = 10;
```

Create a PDSCH configuration object with a 16-QAM modulation scheme. Set the radio network temporary identifier to 1005, size of the BWP to 25, starting PRB index of the BWP to 10, and PRB set to occupy the whole BWP.

```
pdsch = nrPDSCHConfig;
pdsch.Modulation = '16QAM';
pdsch.RNTI = 1005;
pdsch.NID = []; % Set NID equal to the NCellID property of carrier
pdsch.NSizeBWP = 25;
```

```
pdsch.NStartBWP = 10;
pdsch.PRBSet = 0:pdsch.NSizeBWP-1;
```

Generate PDSCH indices in subscript form and set the index orientation to bandwidth part.

```
[ind,info] = nrPDSCHIndices(carrier,pdsch,'IndexStyle','subscript','IndexOrientation','bwp')
```

```
ind = 3900x3 uint32 matrix

    1    1    1
    2    1    1
    3    1    1
    4    1    1
    5    1    1
    6    1    1
    7    1    1
    8    1    1
    9    1    1
   10    1    1
      ⋮

info = struct with fields:
                G: 15600
               Gd: 3900
        NREPerPRB: 156
    DMRSSymbolSet: 2
    PTRSSymbolSet: [1x0 double]
```

Generate PDSCH symbols of data type single.

```
numDataBits = info.G;
cws = randi([0 1],numDataBits,1);
sym = nrPDSCH(carrier,pdsch,cws,'OutputDataType','single')
```

```
sym = 3900x1 single column vector

  -0.9487 + 0.9487i
  -0.9487 - 0.9487i
  -0.3162 - 0.9487i
   0.9487 - 0.3162i
  -0.9487 + 0.3162i
   0.3162 + 0.9487i
   0.3162 + 0.9487i
  -0.3162 + 0.3162i
   0.3162 + 0.3162i
   0.9487 - 0.3162i
      ⋮
```

Input Arguments

cws — DL-SCH codewords
cell array of binary column vectors | binary column vector

DL-SCH codewords, specified as one of these values:

- Cell array of one or two binary column vectors — Use this value to specify one or two DL-SCH codewords, as described in TS 38.212 Section 7.2.6.
- Binary column vector — Use this value to specify one DL-SCH codeword.

Data Types: `double` | `single` | `cell`

`mod` — Modulation scheme
`'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'` | string array | cell array of character vectors

Modulation scheme, specified as `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`, a string array, or a cell array of character vectors. This modulation scheme specifies the modulation type of the codewords and the number of bits used per modulation symbol. If `cws` contains two codewords, the modulation scheme applies to both codewords. Alternatively, you can specify different modulation schemes for each codeword by using a string array or a cell array of character vectors.

Modulation Scheme	Number of Bits Per Symbol
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

Example: To specify different modulation schemes for two codewords, you can use any of these formats: `{'QPSK','16QAM'}` or `["QPSK","16QAM"]`.

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

`nlayers` — Number of transmission layers
integer from 1 to 8

Number of transmission layers, specified as an integer from 1 to 8. For one codeword, use an integer between 1 to 4. For two codewords, use an integer between 5 to 8.

Data Types: `double`

`nid` — Scrambling identity
integer

Scrambling identity, specified as an integer from 0 to 1023. `nid` is the physical layer cell identity number (0 to 1007) or higher layer parameter *dataScramblingIdentityPDSCH* (0 to 1023). For more information, see TS 38.331 Section 6.3.2.

Data Types: `double`

`rnti` — RNTI of UE
integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

`datatype` — Data type of output symbols
`'double'` (default) | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

`carrier` — Carrier configuration parameters
`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function uses only the `NCellID` property of the `nrCarrierConfig` object.

`pdsch` — PDSCH configuration parameters
`nrPDSCHConfig` object

PDSCH configuration parameters, specified as an `nrPDSCHConfig` object. This function uses only these `nrPDSCHConfig` object properties.

- `Modulation`
- `NumLayers`
- `NID`
- `RNTI`

Output Arguments

`sym` — PDSCH modulation symbols
complex matrix

PDSCH modulation symbols, returned as a complex matrix.

Data Types: `single` | `double`
Complex Number Support: Yes

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.212. "NR; Multiplexing and channel coding." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The `datatype` input argument must be compile-time constant. For example, when specifying `'single'` as the output data type, include {coder.Constant('OutputDataType'),coder.Constant('single')} in the -args value of the codegen function. For more information, see the coder.Constant class.

See Also

Functions
nrPDSCHDecode | nrPDSCHPRBS | nrLayerMap | nrSymbolModulate | nrDLSCHInfo | nrPDSCHIndices

Objects
nrCarrierConfig | nrPDSCHConfig

nrPDSCHDecode

Decode PDSCH modulation symbols

Syntax

```
[cws,symbols] = nrPDSCHDecode(sym,mod,nid,rnti)
[cws,symbols] = nrPDSCHDecode(carrier,pdsch,sym)
[cws,symbols] = nrPDSCHDecode( ___ ,nVar)
```

Description

`[cws,symbols] = nrPDSCHDecode(sym,mod,nid,rnti)` returns soft bits `cws` and constellation symbols `symbols` resulting from the inverse operation of the physical downlink shared channel (PDSCH) processing specified in TS 38.211 Sections 7.3.11–3 [1]. The decoding consists of layer demapping, demodulation of `sym` with modulation scheme `mod`, and descrambling with scrambling identity `nid`. The input `rnti` is the radio network temporary identifier (RNTI) of the user equipment (UE).

`[cws,symbols] = nrPDSCHDecode(carrier,pdsch,sym)` returns soft bits `cws` and constellation symbols `symbols` for the specified carrier configuration, `carrier`, and PDSCH configuration, `pdsch`. The input `sym` specifies the received PDSCH symbols to decode.

`[cws,symbols] = nrPDSCHDecode( ___ ,nVar)` specifies the noise variance scaling factor of the soft bits in the PDSCH demodulation, in addition to the input arguments in any of the previous syntaxes.

Examples

Decode PDSCH Modulation Symbols

Generate and decode PDSCH modulation symbols.

Specify a random sequence of binary values corresponding to a codeword of 8000 bits using 256-QAM modulation. Generate PDSCH modulation symbols for the specified physical layer cell identity number, RNTI, and number of transmission layers.

```
modulation = '256QAM';
nlayers = 4;
ncellid = 42;
rnti = 6143;
data = randi([0 1],8000,1);
txsym = nrPDSCH(data,modulation,nlayers,ncellid,rnti);
```

Add an additive white Gaussian noise (AWGN) to the PDSCH symbols. Then demodulate to produce soft bit estimates.

```
SNR = 30; % SNR in dB
rxsym = awgn(txsym,SNR);
rxbits = nrPDSCHDecode(rxsym,modulation,ncellid,rnti);
```

Decode PDSCH Symbols Using Configuration Parameters

Create a carrier configuration object with default properties. Specify the physical layer cell identity as 42.

```
carrier = nrCarrierConfig;
carrier.NCellID = 42;
```

Create a PDSCH configuration object with default properties. Set the radio network temporary identifier to 1005, size of the bandwidth part to 25, starting PRB index of the BWP to 10, and the PRB set to occupy the whole BWP.

```
pdsch = nrPDSCHConfig;
pdsch.RNTI = 1005;
pdsch.NID = []; % Set NID equal to the NCellID property of carrier
pdsch.NSizeBWP = 25;
pdsch.NStartBWP = 10;
pdsch.PRBSet = 0:pdsch.NSizeBWP-1;
```

Generate PDSCH symbols for a single codeword of 8000 bits with the specified carrier configuration and PDSCH configuration.

```
cws = randi([0 1],8000,1);
sym = nrPDSCH(carrier,pdsch,cws);
```

Add an additive white Gaussian noise (AWGN) to the PDSCH symbols. Then demodulate the symbols to produce soft-bit estimates.

```
SNR = 30; % SNR in dB
rxsym = awgn(sym,SNR);
[rxbits,symbols] = nrPDSCHDecode(carrier,pdsch,rxsym);
```

Input Arguments

`sym` — Received PDSCH modulation symbols

complex matrix

Received PDSCH modulation symbols, specified as a complex matrix of size N_{RE}-by-N_{Layers}. N_{RE} is the number of resource elements in a layer, and N_{Layers} is the number of layers. N_{Layers} determines the number of codewords in `cws`.

- If N_{Layers} is from 1 to 4, the function returns one codeword in `cws`.
- If N_{Layers} is from 5 to 8, the function returns two codewords in `cws`.

Data Types: `single` | `double`
Complex Number Support: Yes

`mod` — Modulation scheme

`'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'` | string array | cell array of character vectors

Modulation scheme, specified as `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`, a string array, or a cell array of character vectors. This modulation scheme specifies the modulation type of the codewords and the number of bits used per modulation symbol. If `cws` contains two codewords, the modulation scheme applies to both codewords. Alternatively, you can specify different modulation schemes for each codeword by using a string array or a cell array of character vectors.

Modulation Scheme	Number of Bits Per Symbol
'QPSK'	2
'16QAM'	4
'64QAM'	6
'256QAM'	8

Example: To specify different modulation schemes for two codewords, you can use any of these formats: `{'QPSK','16QAM'}` or `["QPSK","16QAM"]`.

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

`nid` — Scrambling identity
integer

Scrambling identity, specified as an integer from 0 to 1023. `nid` is the physical layer cell identity number (0 to 1007) or higher layer parameter *dataScramblingIdentityPDSCH* (0 to 1023). For more information, see TS 38.331 Section 6.3.2.

Data Types: `double`

`rnti` — RNTI of UE
integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

`nVar` — Noise variance
`1e-10` (default) | nonnegative numeric scalar

Noise variance, specified as a nonnegative numeric scalar. The soft bits are scaled with the variance of additive white Gaussian noise (AWGN). The default value corresponds to an SNR of 100 dB, assuming unit signal power.

Note The default value assumes the decoder and coder are connected back-to-back, where the noise variance is zero. To avoid `-Inf` or `+Inf` values in the output, the function uses `1e-10` as the default value for noise variance. To get appropriate results when the signal is transmitted through a noisy channel, adjust the noise variance accordingly.

Data Types: `double`

`carrier` — Carrier configuration parameters
`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function uses only the `NCellID` property of the `nrCarrierConfig` object.

`pdsch` — PDSCH configuration parameters
`nrPDSCHConfig` object

PDSCH configuration parameters, specified as an `nrPDSCHConfig` object. This function uses only these `nrPDSCHConfig` object properties.

- `Modulation`
- `NID`
- `RNTI`

Output Arguments

`cws` — Approximate LLR soft bits
cell array of real column vectors

Approximate log likelihood ratio (LLR) soft bits, returned as a cell array of one or two real column vectors. The column vectors in `cws` have the same data type as `sym`. The number of column vectors depends on the number of layers in `sym`. The signs in the output values represent the hard bits.

Data Types: `cell`

`symbols` — Symbol constellation for each codeword
cell array of one or two column vectors of complex numbers

Symbol constellation for each codeword in `cws`, returned as a cell array of one or two column vectors of complex numbers. The column vectors in `symbols` have the same data type as `sym`.

Data Types: `cell`

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrPDSCH` | `nrPDSCHPRBS` | `nrLayerMap` | `nrSymbolModulate` | `nrDLSCHInfo` | `nrPDSCHIndices`

Objects
`nrCarrierConfig` | `nrPDSCHConfig`

nrPDSCHDMRS

Generate PDSCH DM-RS symbols

Syntax

```
sym = nrPDSCHDMRS(carrier,pdsch)
sym = nrPDSCHDMRS(carrier,pdsch,'OutputDataType',datatype)
```

Description

`sym = nrPDSCHDMRS(carrier,pdsch)` returns a matrix containing demodulation reference signal (DM-RS) symbols of physical downlink shared channel (PDSCH), as defined in TS 38.211 Section 7.4.1.1.1 [1]. `carrier` specifies the carrier configuration parameters for a specific OFDM numerology. `pdsch` specifies the PDSCH configuration parameters.

`sym = nrPDSCHDMRS(carrier,pdsch,'OutputDataType',datatype)` specifies the data type for the DM-RS symbols.

Examples

Generate PDSCH DM-RS Symbols and Indices

Create a carrier configuration object specifying the slot number as 10.

```
carrier = nrCarrierConfig('NSlot',10);
```

Create a physical downlink shared channel (PDSCH) configuration object, `pdsch`, with physical resource blocks (PRBs) allocated from 0 to 30.

```
pdsch = nrPDSCHConfig;
pdsch.PRBSet = 0:30;
```

Create a PDSCH demodulation reference signal (DM-RS) object, `dmrs`, with specified properties.

```
dmrs = nrPDSCHDMRSConfig;
dmrs.DMRSConfigurationType = 2;
dmrs.DMRSLength = 2;
dmrs.DMRSAdditionalPosition = 1;
dmrs.DMRSTypeAPosition = 2;
dmrs.DMRSPortSet = 5;
dmrs.NIDNSCID = 10;
dmrs.NSCID = 0;
```

Assign the PDSCH DM-RS configuration object to DMRS property of PDSCH configuration object.

```
pdsch.DMRS = dmrs;
```

Generate PDSCH DM-RS symbols and indices for the specified carrier, PDSCH configuration, and output formatting `name-value` pair argument.

```
sym = nrPDSCHDMRS(carrier,pdsch,'OutputDataType','single')
```

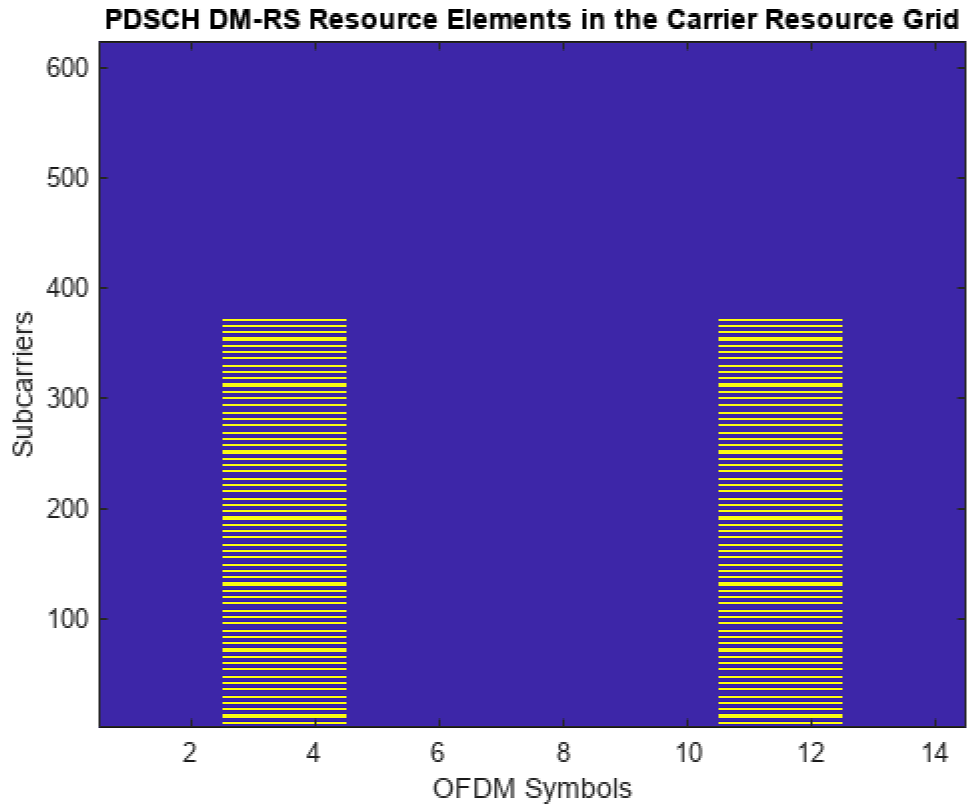
```
sym = 496x1 single column vector

  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
       ⋮
```

```
ind = nrPDSCHDMRSIndices(carrier,pdsch,'IndexBase','0based','IndexOrientation','carrier')
```

```
ind = 496x1 uint32 column vector

   1252
   1253
   1258
   1259
   1264
   1265
   1270
   1271
   1276
   1277
       ⋮
```

Display the generated DM-RS symbols on the carrier resource grid.

```
grid = complex(zeros([carrier.NSizeGrid*12 carrier.SymbolsPerSlot pdsch.NumLayers]));
grid(ind+1) = sym;
imagesc(abs(grid(:,:,1)));
axis xy;
xlabel('OFDM Symbols');
ylabel('Subcarriers');
title('PDSCH DM-RS Resource Elements in the Carrier Resource Grid');
```



Input Arguments

`carrier` — Carrier configuration parameters
`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function uses only these properties of the `nrCarrierConfig` object.

- `NCellID`
- `SubcarrierSpacing`
- `CyclicPrefix`
- `NSizeGrid`
- `NStartGrid`
- `NSlot`

`pdsch` — PDSCH configuration parameters
`nrPDSCHConfig` object

PDSCH configuration parameters, specified as an `nrPDSCHConfig` object. This function uses only these properties of the `nrPDSCHConfig` object.

- `NSizeBWP`

- `NStartBWP`
- `ReservedPRB`
- `NumLayers`
- `MappingType`
- `SymbolAllocation`
- `PRBSet`
- `RNTI`
- `DMRS`

`datatype` — Data type for generated DM-RS symbols
`'double'` (default) | `'single'`

Data type for the generated DM-RS symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

`sym` — DM-RS symbols
complex matrix

DM-RS symbols, returned as a complex matrix. The number of columns correspond to the number of antenna ports configured.

Data Types: `single` | `double`
Complex Number Support: Yes

Version History

Introduced in R2020a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The `datatype` input argument must be compile-time constant. For example, when specifying `'single'` as the output data type, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
nrPDSCH | nrPDSCHDMRSIndices | nrPDSCHPTRS | nrChannelEstimate | nrTimingEstimate

Objects
nrCarrierConfig | nrPDSCHConfig | nrPDSCHReservedConfig | nrPDSCHDMRSConfig

nrPDSCHDMRSIndices

Generate PDSCH DM-RS indices

Syntax

```
ind = nrPDSCHDMRSIndices(carrier,pdsch)
ind = nrPDSCHDMRSIndices(carrier,pdsch,Name,Value)
```

Description

`ind = nrPDSCHDMRSIndices(carrier,pdsch)` returns a matrix containing demodulation reference signal (DM-RS) resource element (RE) indices of a physical downlink shared channel (PDSCH), as defined in TS 38.211 Section 7.4.1.1.2 [1]. `carrier` specifies the carrier configuration parameters for a specific OFDM numerology. `pdsch` specifies the PDSCH configuration parameters. The returned indices are 1-based using linear indexing form.

`ind = nrPDSCHDMRSIndices(carrier,pdsch,Name,Value)` specifies output formatting options using one or more name-value pair arguments. Unspecified options take default values.

Examples

Generate PDSCH DM-RS Symbols and Indices

Create a carrier configuration object specifying the slot number as 10.

```
carrier = nrCarrierConfig('NSlot',10);
```

Create a physical downlink shared channel (PDSCH) configuration object, `pdsch`, with physical resource blocks (PRBs) allocated from 0 to 30.

```
pdsch = nrPDSCHConfig;
pdsch.PRBSet = 0:30;
```

Create a PDSCH demodulation reference signal (DM-RS) object, `dmrs`, with specified properties.

```
dmrs = nrPDSCHDMRSConfig;
dmrs.DMRSConfigurationType = 2;
dmrs.DMRSLength = 2;
dmrs.DMRSAdditionalPosition = 1;
dmrs.DMRSTypeAPosition = 2;
dmrs.DMRSPortSet = 5;
dmrs.NIDNSCID = 10;
dmrs.NSCID = 0;
```

Assign the PDSCH DM-RS configuration object to DMRS property of PDSCH configuration object.

```
pdsch.DMRS = dmrs;
```

Generate PDSCH DM-RS symbols and indices for the specified carrier, PDSCH configuration, and output formatting `name-value` pair argument.

```
sym = nrPDSCHDMRS(carrier,pdsch,'OutputDataType','single')
```

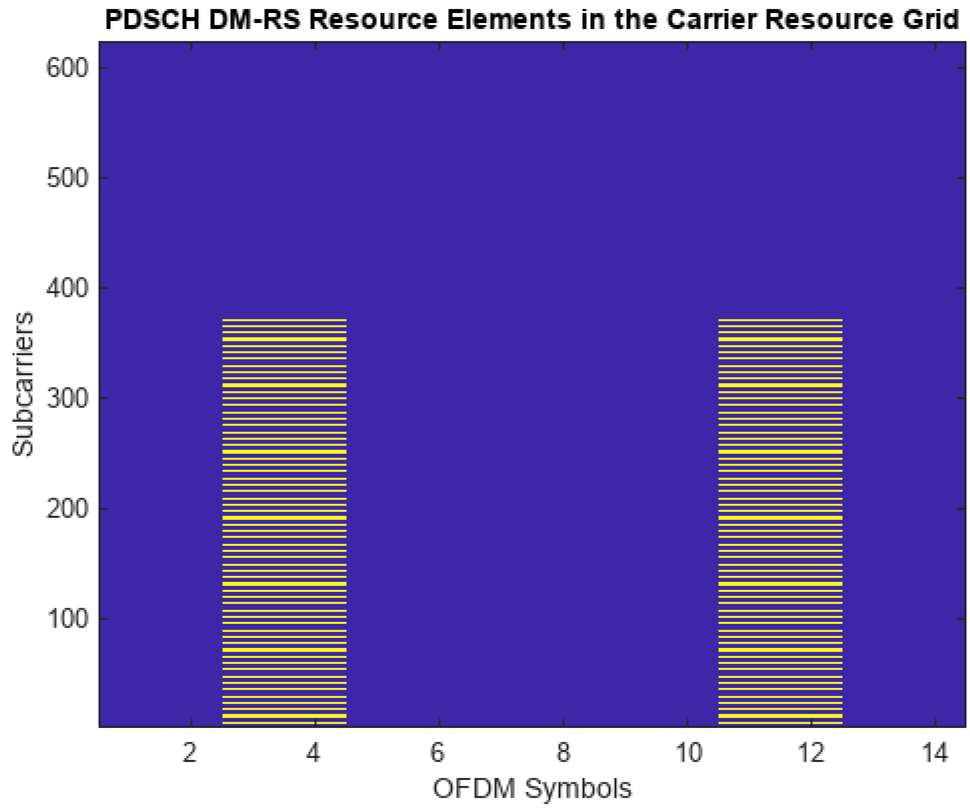
```
sym = 496x1 single column vector

  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
       ⋮
```

```
ind = nrPDSCHDMRSIndices(carrier,pdsch,'IndexBase','0based','IndexOrientation','carrier')
```

```
ind = 496x1 uint32 column vector

   1252
   1253
   1258
   1259
   1264
   1265
   1270
   1271
   1276
   1277
       ⋮
```

Display the generated DM-RS symbols on the carrier resource grid.

```
grid = complex(zeros([carrier.NSizeGrid*12 carrier.SymbolsPerSlot pdsch.NumLayers]));
grid(ind+1) = sym;
imagesc(abs(grid(:,:,1)));
axis xy;
xlabel('OFDM Symbols');
ylabel('Subcarriers');
title('PDSCH DM-RS Resource Elements in the Carrier Resource Grid');
```



Input Arguments

carrier — Carrier configuration parameters
nrCarrierConfig object

Carrier configuration parameters for a specific OFDM numerology, specified as an nrCarrierConfig object. This function uses only these properties of the nrCarrierConfig object.

- SubcarrierSpacing
- CyclicPrefix
- NSizeGrid
- NStartGrid
- NSlot

pdsch — PDSCH configuration parameters
nrPDSCHConfig object

PDSCH configuration parameters, specified as an nrPDSCHConfig object. This function uses only these properties of the nrPDSCHConfig object.

- NSizeBWP
- NStartBWP

- `ReservedPRB`
- `NumLayers`
- `MappingType`
- `SymbolAllocation`
- `PRBSet`
- `DMRS`

Name-Value Pair Arguments

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

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

Example: `'IndexStyle','subscript','IndexBase','0based'` specifies nondefault resource element index formatting properties.

`IndexStyle` — RE indexing form
`'index'` (default) | `'subscript'`

RE indexing form, specified as one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

`IndexBase` — RE indexing base
`'1based'` (default) | `'0based'`

RE indexing base, specified as one of these values:

- `'1based'` — The index counting starts from 1.
- `'0based'` — The index counting starts from 0.

Data Types: `char` | `string`

`IndexOrientation` — Indexing orientation of resource elements
`'carrier'` (default) | `'bwp'`

Indexing orientation of resource elements, specified as the comma-separated pair consisting of `'IndexOrientation'` and one of these values:

- `'carrier'` — Indices are referenced with respect to the carrier grid.
- `'bwp'` — Indices are referenced with respect to the bandwidth part.

Data Types: `char` | `string`

Output Arguments

ind — DM-RS RE indices
N-by-*P* matrix | *M*-by-3 matrix

DM-RS RE indices, returned as one of these values:

- *N*-by-*P* matrix — The function returns this type of value when `'IndexStyle'` is set to `'index'`. The matrix columns correspond to the antenna ports configured.
- *M*-by-3 matrix — The function returns this type of value when `'IndexStyle'` is set to `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and number of antennas, respectively.

Depending on the value of `'IndexBase'`, the function returns either 1-based or 0-based indices. Depending on the value of `'IndexOrientation'`, the function returns either carrier oriented indices or BWP oriented indices.

Data Types: `uint32`

Version History

Introduced in R2020a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrPDSCHIndices` | `nrPDSCHDMRS` | `nrPDSCHPTRSIndices` | `nrChannelEstimate` | `nrTimingEstimate`

Objects

`nrCarrierConfig` | `nrPDSCHConfig` | `nrPDSCHReservedConfig` | `nrPDSCHDMRSConfig`

nrPDSCHIndices

Generate PDSCH resource element indices

Syntax

```
[ind,info] = nrPDSCHIndices(carrier,pdsch)
[ind,info] = nrPDSCHIndices(carrier,pdsch,Name,Value)
```

Description

`[ind,info] = nrPDSCHIndices(carrier,pdsch)` returns `ind` in matrix form, which contains 1-based physical downlink shared channel (PDSCH) resource element (RE) indices, as defined in TS 38.211 Sections 7.3.1.5 and 7.3.1.6 [1]. The number of columns in `ind` is equal to the number of antenna ports configured. `carrier` specifies the carrier configuration parameters for a specific OFDM numerology and `pdsch` specifies the PDSCH configuration . The function also returns the structure `info`, which contains additional information about the associated physical reference signals, PDSCH bit capacity, and PDSCH symbol capacity.

`[ind,info] = nrPDSCHIndices(carrier,pdsch,Name,Value)` specifies output formatting options using one or more name-value pair arguments. Unspecified options take default values.

Examples

Generate PDSCH Symbols and Indices

Create a carrier configuration object with default properties. Specify the physical layer cell identity as 42 and slot number as 10.

```
carrier = nrCarrierConfig;
carrier.NCellID = 42;
carrier.NSlot = 10;
```

Create a PDSCH configuration object with a 16-QAM modulation scheme. Set the radio network temporary identifier to 1005, size of the BWP to 25, starting PRB index of the BWP to 10, and PRB set to occupy the whole BWP.

```
pdsch = nrPDSCHConfig;
pdsch.Modulation = '16QAM';
pdsch.RNTI = 1005;
pdsch.NID = []; % Set NID equal to the NCellID property of carrier
pdsch.NSizeBWP = 25;
pdsch.NStartBWP = 10;
pdsch.PRBSet = 0:pdsch.NSizeBWP-1;
```

Generate PDSCH indices in subscript form and set the index orientation to bandwidth part.

```
[ind,info] = nrPDSCHIndices(carrier,pdsch,'IndexStyle','subscript','IndexOrientation','bwp')
```

```
ind = 3900x3 uint32 matrix
```

```
     1     1     1
     2     1     1
     3     1     1
     4     1     1
     5     1     1
     6     1     1
     7     1     1
     8     1     1
     9     1     1
    10     1     1
       ⋮
```

```
info = struct with fields:
                G: 15600
               Gd: 3900
        NREPerPRB: 156
    DMRSSymbolSet: 2
    PTRSSymbolSet: [1x0 double]
```

Generate PDSCH symbols of data type single.

```
numDataBits = info.G;
cws = randi([0 1],numDataBits,1);
sym = nrPDSCH(carrier,pdsch,cws,'OutputDataType','single')
```

```
sym = 3900x1 single column vector

  -0.9487 + 0.9487i
  -0.9487 - 0.9487i
  -0.3162 - 0.9487i
   0.9487 - 0.3162i
  -0.9487 + 0.3162i
   0.3162 + 0.9487i
   0.3162 + 0.9487i
  -0.3162 + 0.3162i
   0.3162 + 0.3162i
   0.9487 - 0.3162i
       ⋮
```

Input Arguments

carrier — Carrier configuration parameters

nrCarrierConfig object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function uses only these properties of the `nrCarrierConfig` object.

- `SubcarrierSpacing`
- `CyclicPrefix`
- `NSizeGrid`
- `NStartGrid`

- `NSlot`

`pdsch` — PDSCH configuration parameters
`nrPDSCHConfig` object

PDSCH configuration parameters, specified as an `nrPDSCHConfig` object. This function only uses these `nrPDSCHConfig` object properties.

- `NSizeBWP`
- `NStartBWP`
- `ReservedPRB`
- `ReservedRE`
- `Modulation`
- `NumLayers`
- `MappingType`
- `SymbolAllocation`
- `PRBSet`
- `VRBToPRBInterleaving`
- `VRBBundleSize`
- `RNTI`
- `DMRS`
- `EnablePTRS`
- `PTRS`

Name-Value Pair Arguments

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

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

Example: `'IndexStyle','subscript','IndexBase','0based'` specifies the RE indexing form and base, respectively, of the output.

`IndexStyle` — RE indexing form
`'index'` (default) | `'subscript'`

RE indexing form, specified as one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

`IndexBase` — RE indexing base
`'1based'` (default) | `'0based'`

RE indexing base, specified as one of these values:

- `'1based'` — The index counting starts from 1.
- `'0based'` — The index counting starts from 0.

Data Types: `char` | `string`

`IndexOrientation` — Indexing orientation of resource elements
`'carrier'` (default) | `'bwp'`

Indexing orientation of resource elements, specified as the comma-separated pair consisting of `'IndexOrientation'` and one of these values:

- `'carrier'` — Indices are referenced with respect to the carrier grid.
- `'bwp'` — Indices are referenced with respect to the bandwidth part.

Data Types: `char` | `string`

Output Arguments

`ind` — PDSCH resource element indices
N-by-*P* matrix | *M*-by-3 matrix

PDSCH resource element indices, returned as one of these values.

- *N*-by-*P* matrix — The function returns this type of value when `'IndexStyle'` is set to `'index'`.
- *M*-by-3 matrix — The function returns this type of value when `'IndexStyle'` is set to `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and number of antennas, respectively.

Depending on the value of `'IndexBase'`, the function returns either 1-based or 0-based indices. Depending on the value of `'IndexOrientation'`, the function returns either carrier oriented indices or BWP oriented indices.

Data Types: `uint32`

`info` — PDSCH resource information
structure

PDSCH resource information, returned as a structure containing these fields.

Fields	Description
`G`	Bit capacity of the PDSCH. This value must be equal to the length of the codeword from the DL-SCH transport channel. Nominally, the value of **`G`** is set to the `outlen` property of `nrDLSCH` System object.
`Gd`	Number of resource elements per layer or port
`DMRSSymbolSet`	The OFDM symbol locations in a slot containing demodulation reference signal (DM-RS) (0-based)
`NREPerPRB`	Number of REs per PRB allocated to PDSCH. This value excludes any reserved resources.
`PTRSSymbolSet`	The OFDM symbol locations in a slot containing phase tracking reference signal (PT-RS) (0-based)

Version History

Introduced in R2020a

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPDSCH` | `nrPDSCHDecode`

Objects
`nrCarrierConfig` | `nrPDSCHConfig` | `nrPDSCHReservedConfig` | `nrPDSCHDMRSConfig` | `nrPDSCHPTRSConfig`

nrPDSCHPTRS

Generate PDSCH PT-RS symbols

Syntax

```
sym = nrPDSCHPTRS(carrier,pdsch)
sym = nrPDSCHPTRS(carrier,pdsch,'OutputDataType',datatype)
```

Description

`sym = nrPDSCHPTRS(carrier,pdsch)` returns `sym`, which contains phase tracking reference signal (PT-RS) symbols of the physical downlink shared channel (PDSCH), as defined in TS 38.211 Section 7.4.1.2.1 [1]. `carrier` specifies the carrier configuration parameters for a specific OFDM numerology and `pdsch` specifies the PDSCH configuration parameters.

`sym = nrPDSCHPTRS(carrier,pdsch,'OutputDataType',datatype)` specifies the data type of the output PT-RS symbols.

Examples

Generate PDSCH PT-RS Symbols and Indices

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier.

```
carrier = nrCarrierConfig;
```

Create a default PDSCH configuration object, and then enable the PT-RS configuration.

```
pdsch = nrPDSCHConfig;
pdsch.EnablePTRS = 1;
```

Create a PDSCH phase tracking reference signal (PT-RS) configuration object with specified properties.

```
ptrs = nrPDSCHPTRSConfig;
ptrs.TimeDensity = 2;
ptrs.FrequencyDensity = 4;
ptrs.REOffset = '10';
```

Assign the PDSCH PT-RS configuration object to PTRS property of PDSCH configuration object.

```
pdsch.PTRS = ptrs;
```

Generate PDSCH PT-RS symbols of data type single.

```
sym = nrPDSCHPTRS(carrier,pdsch,'OutputDataType','single')
```

```
sym = 78x1 single column vector

  -0.7071 - 0.7071i
```

```
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
   0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
      ⋮
```

Generate PDSCH PT-RS indices in subscript form and set the index orientation to bandwidth part.

```
ind = nrPDSCHPTRSIndices(carrier,pdsch,'IndexStyle','subscript','IndexOrientation','bwp')
```

```
ind = 78x3 uint32 matrix

    19     1     1
    67     1     1
   115     1     1
   163     1     1
   211     1     1
   259     1     1
   307     1     1
   355     1     1
   403     1     1
   451     1     1
      ⋮
```

Input Arguments

carrier — Carrier configuration parameters
nrCarrierConfig object

Carrier configuration parameters for a specific OFDM numerology, specified as an nrCarrierConfig object. This function uses only these properties of the nrCarrierConfig object.

- NCellID
- SubcarrierSpacing
- CyclicPrefix
- NSizeGrid
- NStartGrid
- NSlot

pdsch — PDSCH configuration parameters
nrPDSCHConfig object

PDSCH configuration parameters, specified as an nrPDSCHConfig object. This function only uses these nrPDSCHConfig object properties.

- NSizeBWP

- `NStartBWP`
- `ReservedPRB`
- `ReservedRE`
- `NumLayers`
- `MappingType`
- `SymbolAllocation`
- `PRBSet`
- `RNTI`
- `DMRS`
- `EnablePTRS`
- `PTRS`

`datatype` — Data type of generated PT-RS symbols
`'double'` (default) | `'single'`

Data type for the generated PT-RS symbols, specified as 'double' or 'single'.

Data Types: `char` | `string`

Output Arguments

`sym` — PT-RS symbols
complex column vector

PT-RS symbols, returned as a complex column vector.

Data Types: `double` | `single`
Complex Number Support: Yes

Version History

Introduced in R2020a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The `datatype` input argument must be compile-time constant. For example, when specifying `'single'` as the output data type, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
nrPDSCH | nrPDSCHDecode | nrPDSCHIndices | nrPDSCHPTRSIndices | nrPDSCHDMRS | nrPDSCHDMRSIndices

Objects
nrCarrierConfig | nrPDSCHConfig | nrPDSCHPTRSConfig | nrPDSCHDMRSConfig | nrPDSCHReservedConfig

nrPDSCHPTRSIndices

Generate PDSCH PT-RS Indices

Syntax

```
ind = nrPDSCHPTRSIndices(carrier,pdsch)
ind = nrPDSCHPTRSIndices(carrier,pdsch,Name,Value)
```

Description

`ind = nrPDSCHPTRSIndices(carrier,pdsch)` returns `ind`, which contains 1-based phase tracking reference signal (PT-RS) resource elements (RE) of the physical downlink shared channel (PDSCH), as defined in TS 38.211 Section 7.4.1.2.2 [1]. The function returns `ind` in linear form, for given carrier configuration `carrier` and physical downlink shared channel configuration `pdsch`.

`ind = nrPDSCHPTRSIndices(carrier,pdsch,Name,Value)` specifies output formatting options by using one or more name-value pair arguments. Unspecified options take default values.

Examples

Generate PDSCH PT-RS Symbols and Indices

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier.

```
carrier = nrCarrierConfig;
```

Create a default PDSCH configuration object, and then enable the PT-RS configuration.

```
pdsch = nrPDSCHConfig;
pdsch.EnablePTRS = 1;
```

Create a PDSCH phase tracking reference signal (PT-RS) configuration object with specified properties.

```
ptrs = nrPDSCHPTRSConfig;
ptrs.TimeDensity = 2;
ptrs.FrequencyDensity = 4;
ptrs.REOffset = '10';
```

Assign the PDSCH PT-RS configuration object to PTRS property of PDSCH configuration object.

```
pdsch.PTRS = ptrs;
```

Generate PDSCH PT-RS symbols of data type single.

```
sym = nrPDSCHPTRS(carrier,pdsch,'OutputDataType','single')
```

```
sym = 78x1 single column vector

  -0.7071 - 0.7071i
```

```
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
   0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
      ⋮
```

Generate PDSCH PT-RS indices in subscript form and set the index orientation to bandwidth part.

```
ind = nrPDSCHPTRSIndices(carrier,pdsch,'IndexStyle','subscript','IndexOrientation','bwp')
```

```
ind = 78x3 uint32 matrix

    19     1     1
    67     1     1
   115     1     1
   163     1     1
   211     1     1
   259     1     1
   307     1     1
   355     1     1
   403     1     1
   451     1     1
      ⋮
```

Input Arguments

carrier — Carrier configuration parameters
nrCarrierConfig object

Carrier configuration parameters for a specific OFDM numerology, specified as an nrCarrierConfig object. This function uses only these properties of the nrCarrierConfig object.

- SubcarrierSpacing
- CyclicPrefix
- NSizeGrid
- NStartGrid
- NSlot

pdsch — PDSCH configuration parameters
nrPDSCHConfig object

PDSCH configuration parameters, specified as an nrPDSCHConfig object. This function only uses these nrPDSCHConfig object properties.

- NSizeBWP
- NStartBWP

- `ReservedPRB`
- `ReservedRE`
- `NumLayers`
- `MappingType`
- `SymbolAllocation`
- `PRBSet`
- `RNTI`
- `DMRS`
- `EnablePTRS`
- `PTRS`

Name-Value Pair Arguments

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

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

Example: `'IndexStyle','subscript','IndexBase','0based'` specifies the RE indexing form and base, respectively, of the output.

`IndexStyle` — RE indexing form
`'index'` (default) | `'subscript'`

RE indexing form, specified as one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

`IndexBase` — RE indexing base
`'1based'` (default) | `'0based'`

RE indexing base, specified as one of these values:

- `'1based'` — The index counting starts from 1.
- `'0based'` — The index counting starts from 0.

Data Types: `char` | `string`

`IndexOrientation` — Indexing orientation of resource elements
`'carrier'` (default) | `'bwp'`

Indexing orientation of resource elements, specified as the comma-separated pair consisting of `'IndexOrientation'` and one of these values:

- `'carrier'` — Indices are referenced with respect to the carrier grid.
- `'bwp'` — Indices are referenced with respect to the bandwidth part.

Data Types: `char` | `string`

Output Arguments

ind — PT-RS resource element indices
column vector (default) | *M*-by-3 matrix

PT-RS resource element indices, returned as one of these values.

- Column vector — The function returns this type of value when `'IndexStyle'` is set to `'index'`.
- *M*-by-3 matrix — The function returns this type of value when `'IndexStyle'` is set to `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and number of antennas, respectively.

Depending on the value of `'IndexBase'`, the function returns either 1-based or 0-based indices. Depending on the value of `'IndexOrientation'`, the function returns either carrier oriented indices or BWP oriented indices.

Data Types: `uint32`

Version History

Introduced in R2020a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPDSCH` | `nrPDSCHDecode` | `nrPDSCHPTRS` | `nrPDSCHIndices` | `nrPDSCHDMRS` | `nrPDSCHDMRSIndices`

Objects
`nrCarrierConfig` | `nrPDSCHConfig` | `nrPDSCHDMRSConfig` | `nrPDSCHPTRSConfig` | `nrPDSCHReservedConfig`

nrPDSCHPRBS

Generate PDSCH scrambling sequence

Syntax

```
[seq,cinit] = nrPDSCHPRBS(nid,rnti,q,n)
[seq,cinit] = nrPDSCHPRBS(nid,rnti,q,n,Name,Value)
```

Description

`[seq,cinit] = nrPDSCHPRBS(nid,rnti,q,n)` returns the first `n` elements of the physical downlink shared channel (PDSCH) scrambling sequence. The function also returns the initialization value `cinit` of the pseudorandom binary sequence (PRBS) generator. The initialization value depends on the scrambling identity number `nid`, the radio network temporary identifier (RNTI) of the user equipment (UE) `rnti`, and the codeword number `q`. The function implements TS 38.211 Section 7.3.1.1 [1].

`[seq,cinit] = nrPDSCHPRBS(nid,rnti,q,n,Name,Value)` specifies additional output formatting options by using one or more name-value pair arguments. Unspecified options take their default values.

Examples

Generate PDSCH Scrambling Sequence

Generate the first 300 outputs of the PDSCH scrambling sequence when initialized with the specified physical layer cell identity number, RNTI, and codeword number.

```
ncellid = 17;
rnti = 120;
q = 0;
n = 300;
seq = nrPDSCHPRBS(ncellid,rnti,q,n)
```

```
seq = 300x1 logical array

   0
   1
   1
   0
   1
   1
   0
   1
   0
   0
      ⋮
```

Input Arguments

`nid` — Scrambling identity
integer

Scrambling identity, specified as an integer from 0 to 1023. `nid` is the physical layer cell identity number (0 to 1007) or higher layer parameter *dataScramblingIdentityPDSCH* (0 to 1023). For more information, see TS 38.331 Section 6.3.2.

Data Types: `double`

`rnti` — RNTI of UE
integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

`q` — Codeword number
`0` | `1`

Codeword number, specified as `0` or `1`.

Data Types: `double`

`n` — Number of elements in output sequence
nonnegative integer

Number of elements in output sequence, specified as a nonnegative integer.

Data Types: `double`

Name-Value Pair Arguments

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

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

Example: `'MappingType','signed'` specifies nondefault sequence formatting.

`MappingType` — Output sequence formatting
`'binary'` (default) | `'signed'`

Output sequence formatting, specified as the comma-separated pair consisting of `'MappingType'` and one of these values:

- `'binary'` — This value maps `true` to `1` and `false` to `0`. The data type of the output sequence is `logical`.
- `'signed'` — This value maps `true` to `–1` and `false` to `1`. The data type of the output sequence is `double`. To specify `single` data type, use the `'OutputDataType'` name-value pair.

Data Types: `char` | `string`

`OutputDataType` — Data type of output sequence
`'double'` (default) | `'single'`

Data type of output sequence, specified as the comma-separated pair consisting of `'OutputDataType'` and `'double'` or `'single'`. This name-value pair applies only when `'MappingType'` is set to `'signed'`.

Data Types: `char` | `string`

Output Arguments

`seq` — PDSCH scrambling sequence
logical column vector | numeric column vector

PDSCH scrambling sequence, returned as a logical or numeric column vector. `seq` contains the first `n` elements of the PDSCH scrambling sequence. If you set `'MappingType'` to `'signed'`, the output data type is either `double` or `single`. If you set `'MappingType'` to `'binary'`, the output data type is `logical`.

Data Types: `double` | `single` | `logical`

`cinit` — Initialization value for PRBS generator
nonnegative integer

Initialization value for PRBS generator, returned as a nonnegative integer.

Data Types: `double`

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying `'single'` as the output data type, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPDSCH` | `nrPDSCHDecode` | `nrPRBS`

nrPerfectChannelEstimate

Perfect channel estimation

Syntax

```
h = nrPerfectChannelEstimate(carrier,pathGains,pathFilters)
h = nrPerfectChannelEstimate(pathGains,pathFilters,nrb,scs,initialNSlot)
h = nrPerfectChannelEstimate( ___ ,toffset)
h = nrPerfectChannelEstimate( ___ ,toffset,sampleTimes)
h = nrPerfectChannelEstimate( ___ ,cpl)
h = nrPerfectChannelEstimate( ___ ,Name,Value)
```

Description

`h = nrPerfectChannelEstimate(carrier,pathGains,pathFilters)` performs perfect channel estimation. The function first reconstructs the channel impulse response from the channel path gains `pathGains` and the path filter impulse response `pathFilters`. The function then performs orthogonal frequency division multiplexing (OFDM) demodulation. `carrier` specifies the parameters for the OFDM demodulation.

`h = nrPerfectChannelEstimate(pathGains,pathFilters,nrb,scs,initialNSlot)` performs the OFDM demodulation for `nrb` number of resource blocks with subcarrier spacing `scs` and initial slot number `initialNSlot`.

`h = nrPerfectChannelEstimate( ___ ,toffset)` specifies the timing offset in addition to the input arguments in any of the previous syntaxes. The timing offset indicates the OFDM demodulation starting point on the reconstructed waveform.

`h = nrPerfectChannelEstimate( ___ ,toffset,sampleTimes)` specifies the sample times of the channel snapshots in addition to the input arguments in the previous syntax.

`h = nrPerfectChannelEstimate( ___ ,cpl)` specifies the cyclic prefix length in addition to the input arguments in any of the previous syntaxes that do not include the `carrier` input.

`h = nrPerfectChannelEstimate( ___ ,Name,Value)` specifies options by using one or more name-value pair arguments in addition to the input arguments in any of the previous syntaxes.

Examples

Plot Estimated Channel Magnitude Response for TDL-C Channel Model

Define a channel configuration structure using an `nrTDLChannel` System object. Use delay profileTDL-C from TR 38.901 Section 7.7.2.

```
SR = 7.68e6;
tdl = nrTDLChannel;
tdl.DelayProfile = 'TDL-C';
tdl.DelaySpread = 100e-9;
tdl.MaximumDopplerShift = 300;
tdl.SampleRate = SR;
```

Create a random waveform with a duration of 1 subframe.

```
T = SR*1e-3;
tdlInfo = info(tdl);
Nt = tdlInfo.NumTransmitAntennas;
in = complex(randn(T,Nt),randn(T,Nt));
```

Transmit the input waveform through the channel. Obtain the path filters used in channel filtering.

```
[~,pathGains] = tdl(in);
pathFilters = getPathFilters(tdl);
```

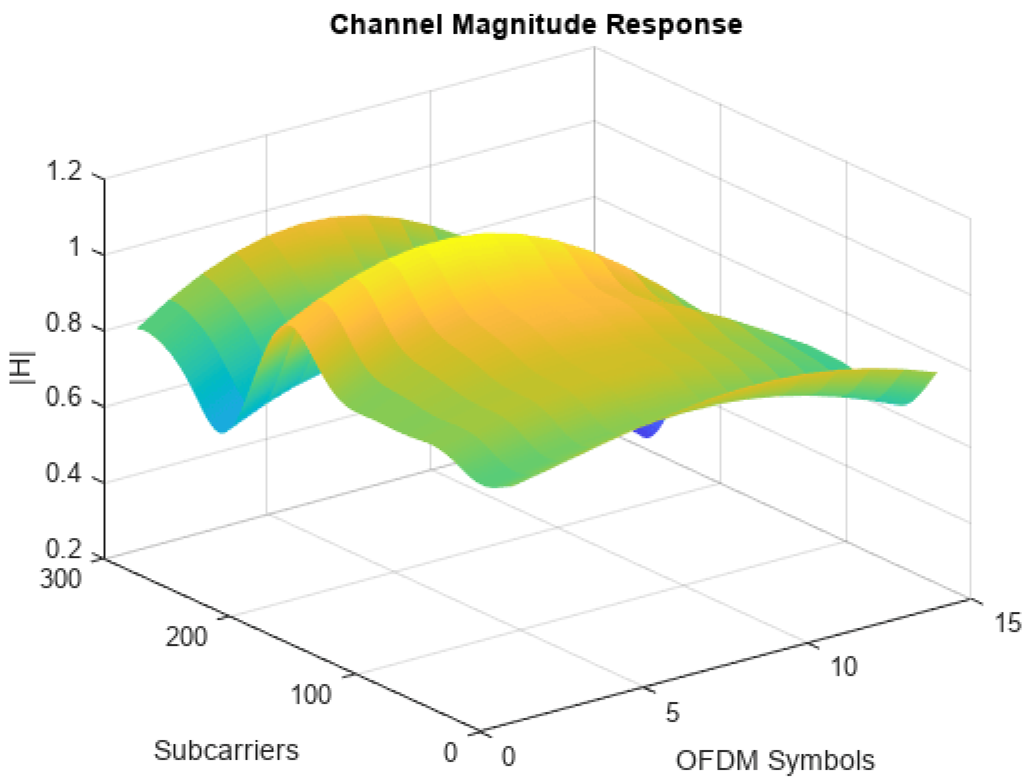
Perform perfect channel estimation using the specified number of blocks, subcarrier spacing, and slot number.

```
NRB = 25;
SCS = 15;
nSlot = 0;

hest = nrPerfectChannelEstimate(pathGains,pathFilters,NRB,SCS,nSlot);
size(hest)
```

```
ans = 1×3

   300    14     2
```

Plot the estimated channel magnitude response for the first receive antenna.

```
figure;
surf(abs(hest(:,:,1)));
shading('flat');
xlabel('OFDM Symbols');
ylabel('Subcarriers');
zlabel('|H|');
title('Channel Magnitude Response');
```



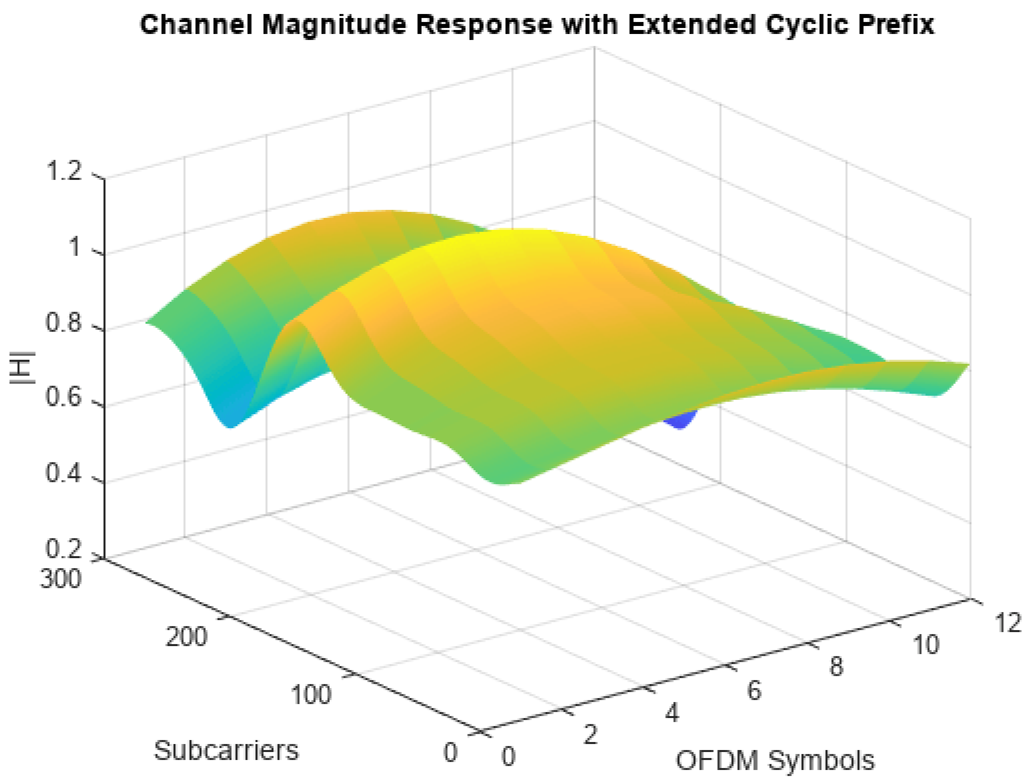
Repeat the channel estimate for extended cyclic prefix.

```
hest = nrPerfectChannelEstimate(pathGains,pathFilters,NRB,SCS, ...
    nSlot,'extended');
size(hest)
```

```
ans = 1×3

   300    12     2
```

Plot the updated results.

```
figure;
surf(abs(hest(:,:,1)));
shading('flat');
xlabel('OFDM Symbols');
ylabel('Subcarriers');
zlabel('|H|');
title('Channel Magnitude Response with Extended Cyclic Prefix');
```



Plot Estimated Channel Magnitude Response for CDL-D Channel Model

Define a channel configuration structure using an `nrCDLChannel` System object. Use delay profile CDL-C from TR 38.901 Section 7.7.1.

```
cdl = nrCDLChannel;
cdl.DelayProfile = 'CDL-D';
cdl.DelaySpread = 30e-9;
cdl.MaximumDopplerShift = 5;
```

Create a random waveform with a duration of 1 subframe.

```
SR = 15.36e6;
T = SR*1e-3;
cdl.SampleRate = SR;
cdlInfo = info(cdl);
Nt = cdlInfo.NumTransmitAntennas;
in = complex(randn(T,Nt),randn(T,Nt));
```

Transmit the input waveform through the channel. Obtain the path filters used in channel filtering.

```
[~,pathGains,sampleTimes] = cdl(in);
pathFilters = getPathFilters(cdl);
```

Perform timing offset estimation using the path filter and path gains.

```
offset = nrPerfectTimingEstimate(pathGains,pathFilters);
```

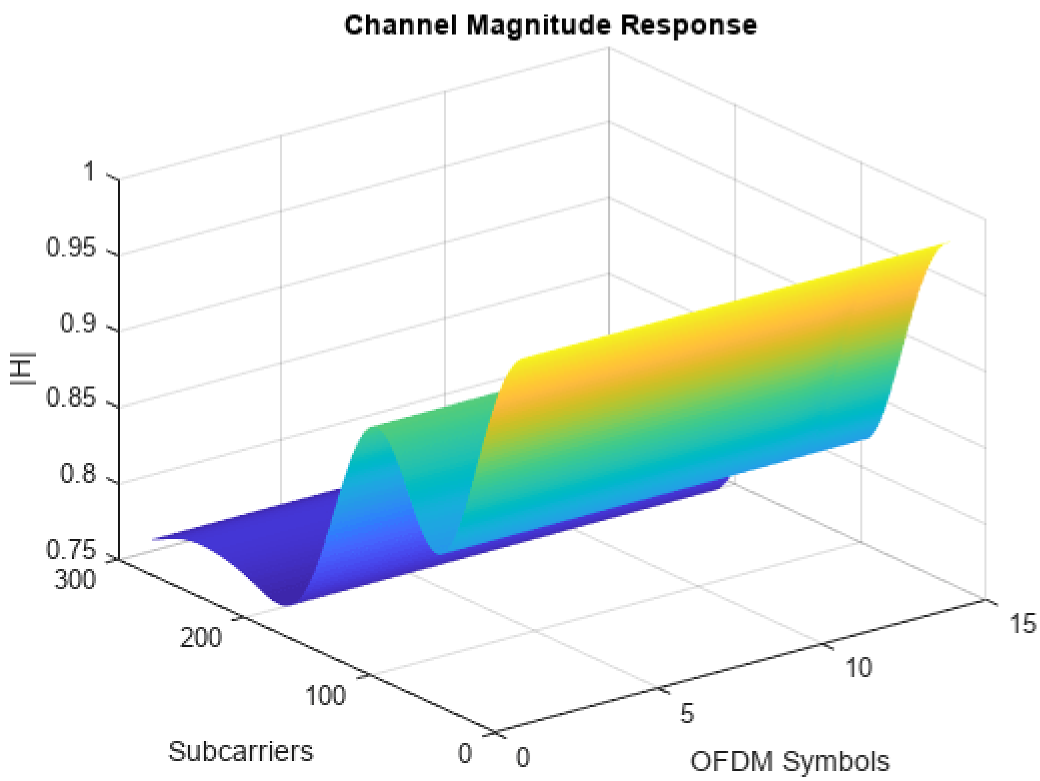
Perform perfect channel estimation. Use the specified number of blocks, subcarrier spacing, slot number, timing offset, and sample times.

```
NRB = 25;
SCS = 15;
nSlot = 0;
hest = nrPerfectChannelEstimate(pathGains,pathFilters,...
    NRB,SCS,nSlot,offset,sampleTimes);
size(hest)
```

```
ans = 1×4

   300    14     2     8
```

Plot the estimated channel magnitude response for the first receive antenna.

```
figure;
surf(abs(hest(:,:,1)));
shading('flat');
xlabel('OFDM Symbols');
ylabel('Subcarriers');
zlabel('|H|');
title('Channel Magnitude Response');
```



Input Arguments

carrier — Carrier configuration parameters
nrCarrierConfig object

Carrier configuration parameters for a specific OFDM numerology, specified as an nrCarrierConfig object. Only these object properties are relevant for this function.

NSizeGrid — Number of RBs in carrier resource grid
52 (default) | integer from 1 to 275

Number of RBs in the carrier resource grid, specified as an integer from 1 to 275. The default value of 52 corresponds to the maximum number of RBs of a 10 MHz carrier with 15 kHz SCS.

Data Types: double

SubcarrierSpacing — Subcarrier spacing in kHz
15 (default) | 30 | 60 | 120 | 240

Subcarrier spacing in kHz, for all channels and reference signals of the carrier, specified as 15, 30, 60, 120, or 240.

Data Types: double

NSlot — Slot number
0 (default) | nonnegative integer

Slot number, specified as a nonnegative integer. You can set NSlot to a value larger than the number of slots per frame. For example, you can set this value using transmission loop counters in a MATLAB simulation. In this case, you may have to ensure that the property value is modulo the number of slots per frame in a calling code.

Data Types: double

CyclicPrefix — Cyclic prefix length
'normal' (default) | 'extended'

Cyclic prefix length, specified as one of these options.

- 'normal' — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- 'extended' — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies for only 60 kHz subcarrier spacing.

Data Types: char | string

pathGains — Channel path gains of fading process
N_{CS}-by-N_P-by-N_T-by-N_R complex matrix

Channel path gains of the fading process, specified as an N_{CS}-by-N_P-by-N_T-by-N_R complex matrix, where:

- N_{CS} is the number of channel snapshots.
- N_P is the number of paths.

- N_T is the number of transmit antennas.
- N_R is the number of receive antennas.

Data Types: `single` | `double`
Complex Number Support: Yes

`pathFilters` — Path filter impulse response
N_H-by-N_P real matrix

Path filter impulse response, specified as an N_H-by-N_P real matrix, where:

- N_H is the number of impulse response samples.
- N_P is the number of paths.

Each column of the matrix contains the filter impulse response for each path of the delay profile.

Data Types: `double`

`nrb` — Number of resource blocks
integer from 1 to 275

Number of resource blocks, specified as an integer from 1 to 275.

Data Types: `double`

`scs` — Subcarrier spacing in kHz
`15` | `30` | `60` | `120` | `240`

Subcarrier spacing in kHz, specified as `15`, `30`, `60`, `120`, or `240`.

Data Types: `double`

`initialNSlot` — Initial slot number
nonnegative integer

Initial slot number, 0-based, specified as a nonnegative integer. The function selects the appropriate cyclic prefix length for the OFDM demodulation based on the value of `initialNSlot` modulo the number of slots per subframe.

Data Types: `double`

`toffset` — Timing offset in samples
nonnegative integer

Timing offset in samples, specified as a nonnegative integer. The timing offset indicates the OFDM demodulation starting point on the reconstructed waveform. The offset accounts for propagation delays, which is essential when obtaining the perfect estimate of the channel seen by a synchronized receiver. `toffset` defaults to the value `nrPerfectTimingEstimate(pathGains,pathFilters)` when not specified as an input argument.

Data Types: `double`

`sampleTimes` — Sample times of channel snapshots
N_{CS}-by-1 column vector of nonnegative real numbers

Sample times of channel snapshots, specified as an N_{CS}-by-1 column vector of nonnegative real numbers. `sampleTimes` specifies the time of occurrence of each channel snapshot. The number of channel snapshots, N_{CS}, is identical to the first dimension of `pathGains`. When not specified, `sampleTimes` defaults to an N_{CS}-by-1 vector of times starting at zero with sampling rate used for the OFDM modulation of the number of resource blocks `nrb` and subcarrier spacing `scs`. Ensure that the channel snapshots span at least one slot. The function performs channel estimation for each complete slot.

Data Types: `double`

`cpl` — Cyclic prefix length
`'normal'` (default) | `'extended'`

Cyclic prefix length, specified as one of these options:

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, the extended cyclic prefix length only applies to 60 kHz subcarrier spacing.

Note

- If you specify the `carrier` input, use the `CyclicPrefix` property of the `carrier` input to specify the cyclic prefix length. You cannot use the `cpl` input together with the `carrier` input.
- If you specify the cyclic prefix length with the `'CyclicPrefix'` name-value pair argument, you cannot use the `cpl` input.

Data Types: `char` | `string`

Name-Value Pair Arguments

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

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

Example: `'CyclicPrefixFraction',0.75` specifies the start location for demodulation relative to the cyclic prefix length.

`CyclicPrefix` — Cyclic prefix length
`'normal'` (default) | `'extended'`

Cyclic prefix length, specified as the comma-separated pair consisting of `'CyclicPrefix'` and one of these values:

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, the extended cyclic prefix length only applies to 60 kHz subcarrier spacing.

Note

- If you specify the carrier input, use the CyclicPrefix property of the carrier input to specify the cyclic prefix length. You cannot use this name-value pair argument together with the carrier input.
- If you specify the cyclic prefix length with the cpl input, you cannot use this name-value pair argument.

Data Types: char | string

Nfft — Number of FFT points
integer greater than 127 (default depends on other input values) | []

Number of fast Fourier transform (FFT) points, specified as the comma-separated pair consisting of 'Nfft' and a nonnegative integer greater than 127 or []. The value you specify must result in integer-valued cyclic prefix lengths and a maximum occupancy of 100%. The occupancy is defined as the value of $(12 \times N_{RB})$/Nfft, where N_{RB} is the number of resource blocks.

If you do not specify this input, or if you specify 'Nfft',[], the function sets an integer value greater than 127 as a default value for this input. The actual default value depends on other input values.

- If you do not specify the SampleRate input, or if you specify 'SampleRate',[], the function sets Nfft satisfying these conditions.
  - Nfft is an integer power of 2.
  - Nfft results in a maximum occupancy of 85%.
- If you specify the SampleRate input, the function sets Nfft satisfying these conditions.
  - Nfft results in integer-valued cyclic prefix lengths.
  - Nfft maximises the value of gcd (Nfft × *SCS*, SampleRate), where *SCS* is specified by the carrier.SubcarrierSpacing property or the scs input.

For more information, see "Configure OFDM Sample Rate and FFT Size".

Data Types: double

SampleRate — Waveform sample rate
positive scalar (default depends on other input values) | []

Waveform sample rate, specified as the comma-separated pair consisting of 'SampleRate' and either a positive scalar or [].

If you do not specify this input, or if you specify 'SampleRate',[], then the function sets this input to the value of $N_{fft} \times SCS$.

- N_{fft} is the value of the 'Nfft' input.
- *SCS* is the subcarrier spacing. Depending on the function syntax you use, *SCS* is specified by the carrier.SubcarrierSpacing property or the scs input.

For more information, see "Configure OFDM Sample Rate and FFT Size".

Data Types: double

`CyclicPrefixFraction` — FFT window position within cyclic prefix
`0.5` (default) | scalar in the interval [0, 1]

Fast Fourier transform (FFT) window position within the cyclic prefix, specified as the comma-separated pair consisting of `'CyclicPrefixFraction'` and a scalar in the interval [0, 1].

The value that you specify indicates the start location for OFDM demodulation relative to the beginning of the cyclic prefix.

Data Types: `double`

Output Arguments

h — Perfect channel estimate
N_{SC}-by-N_{SYM}-by-N_R-by-N_T complex array

Perfect channel estimate, returned as an N_{SC}-by-N_{SYM}-by-N_R-by-N_T complex array, where:

- N_{SC} is the number of subcarriers.
- N_{SYM} is the number of OFDM symbols.
- N_R is the number of receive antennas.
- N_T is the number of transmit antennas.

`h` inherits its data type from `pathGains`.

Data Types: `double` | `single`

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Names and values in name-value pair arguments must be compile-time constants. For example, when specifying extended cyclic prefix, include `{coder.Constant('CyclicPrefix'),coder.Constant('extended')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.
- The input arguments `nrb`, `scs`, and `initialNSlot` must be compile-time constants. For example, include `{coder.Constant(nrb)}`, `{coder.Constant(scs)}`, and `{coder.Constant(initialNSlot)}` in the `-args` value of the `codegen` function.
- The `'SampleRate'` name-value pair argument cannot be used together with the `carrier` input.

See Also

Functions
nrPerfectTimingEstimate | nrTimingEstimate | nrChannelEstimate

Objects
nrTDLChannel | nrCDLChannel | nrCarrierConfig

nrPerfectTimingEstimate

Perfect timing estimation

Syntax

```
[offset,mag] = nrPerfectTimingEstimate(pathGains,pathFilters)
```

Description

`[offset,mag] = nrPerfectTimingEstimate(pathGains,pathFilters)` performs perfect timing estimation. To find the peak of the channel impulse response, the function first reconstructs the impulse response from the channel path gains `pathGains` and the path filter impulse response `pathFilters`. The channel impulse response is averaged across all channel snapshots and summed across all transmit and receive antennas before timing estimation. The function returns the estimated timing offset `offset` and the channel impulse response magnitude `mag`.

Examples

Plot Channel Impulse Magnitude and Timing Offset for TDL-C Channel Model

Define a channel configuration structure using an `nrTDLChannel` System object. Use delay profile TDL-C from TR 38.901 Section 7.7.2.

```
tdl = nrTDLChannel;
tdl.DelayProfile = 'TDL-C';
tdl.DelaySpread = 100e-9;
```

Create a random waveform with a duration of 1 subframe.

```
tdlInfo = info(tdl);
Nt = tdlInfo.NumTransmitAntennas;
in = complex(zeros(100,Nt),zeros(100,Nt));
```

Transmit the input waveform through the channel.

```
[~,pathGains] = tdl(in);
```

Obtain the path filters used in channel filtering.

```
pathFilters = getPathFilters(tdl);
```

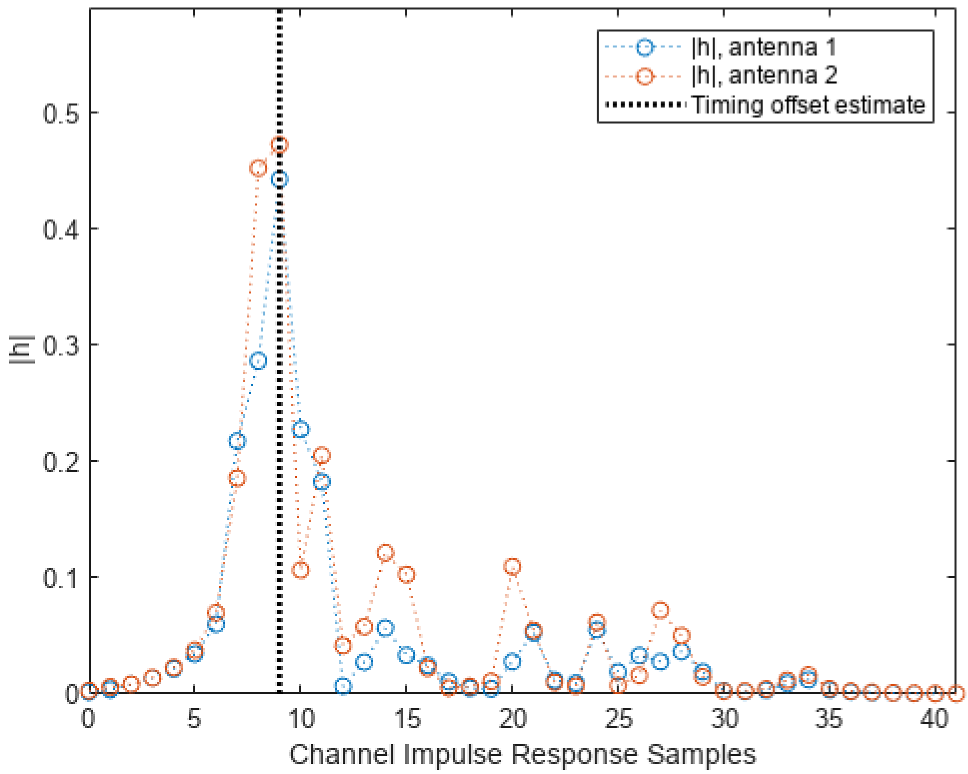
Estimate timing offset.

```
[offset,mag] = nrPerfectTimingEstimate(pathGains,pathFilters);
```

Plot the magnitude of the channel impulse response and the timing offset estimate.

```
[Nh,Nr] = size(mag);
plot(0:(Nh-1),mag,'o:');
hold on;
plot([offset offset],[0 max(mag(:))*1.25],'k:','LineWidth',2);
```

```
axis([0 Nh-1 0 max(mag(:))*1.25]);
legends = "|h|, antenna " + num2cell(1:Nr);
legend([legends "Timing offset estimate"]);
ylabel('|h|');
xlabel('Channel Impulse Response Samples');
```



Input Arguments

pathGains — Channel path gains of fading process
N_{CS}-by-N_P-by-N_T-by-N_R complex matrix

Channel path gains of the fading process, specified as an N_{CS}-by-N_P-by-N_T-by-N_R complex matrix, where:

- N_{CS} is the number of channel snapshots.
- N_P is the number of paths.
- N_T is the number of transmit antennas.
- N_R is the number of receive antennas.

Data Types: `single` | `double`
Complex Number Support: Yes

pathFilters — Path filter impulse response
N_H-by-N_P real matrix

Path filter impulse response, specified as an N_H-by-N_P real matrix, where:

- N_H is the number of impulse response samples.
- N_P is the number of paths.

Each column of the matrix contains the filter impulse response for each path of the delay profile.

Data Types: `double`

Output Arguments

`offset` — Timing offset in samples
nonnegative integer

Timing offset in samples, returned as a nonnegative integer. The number of samples is relative to the first sample of the channel impulse response reconstructed from `pathGains` and `pathFilters`.

Data Types: `double`

`mag` — Channel impulse response magnitude
N_H-by-N_R real matrix

Channel impulse response magnitude for each receive antenna, returned as an N_H-by-N_R real matrix.

- N_H is the number of impulse response samples.
- N_R is the number of receive antennas.

`mag` inherits its data type from `pathGains`.

Data Types: `single` | `double`

Version History

Introduced in R2018b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrPerfectChannelEstimate` | `nrTimingEstimate` | `nrChannelEstimate`

Objects
`nrTDLChannel` | `nrCDLChannel`

nrPolarDecode

Polar decoding

Syntax

```
decbits = nrPolarDecode(rec,K,E,L)
decbits = nrPolarDecode(rec,K,E,L,padCRC)
decbits = nrPolarDecode(rec,K,E,L,padCRC,rnti)
decbits = nrPolarDecode(rec,K,E,L,nmax,iil,CRClen)
```

Description

`decbits = nrPolarDecode(rec,K,E,L)` decodes the rate-recovered input `rec` for an (*N*,K) polar code, where *N* is the length of `rec` and `K` is the length of decoded bits `decbits`, as specified in TS 38.212 Section 5 [1]. The function uses a cyclic redundancy check (CRC)-aided successive-cancellation list decoder of length `L`. By default, output deinterleaving is enabled, the maximum length of the input is 512, and the number of appended CRC bits is 24. Use this syntax for downlink configuration.

`decbits = nrPolarDecode(rec,K,E,L,padCRC)` specifies whether the information block on the transmit end was prepadded with ones before CRC encoding.

`decbits = nrPolarDecode(rec,K,E,L,padCRC,rnti)` specifies a radio network temporary identifier (RNTI). You can use this syntax when the value of `rnti` masks the CRC parity bits at the transmit end.

`decbits = nrPolarDecode(rec,K,E,L,nmax,iil,CRClen)` decodes the input with a specified maximum length of 2^{nmax}, output deinterleaving specified by `iil`, and number of appended CRC bits specified by `CRClen`. This syntax assumes that the information block on the transmit end was not prepadded with ones before CRC encoding and that the RNTI is equal to `0`.

- For downlink (DL) configuration, valid values for `nmax`, `iil`, and `CRClen` are 9, `true`, and 24, respectively.
- For uplink (UL) configuration, valid values for `nmax` and `iil` are 10 and `false`, respectively, and for `CRClen` is 11 or 6.

Examples

Transmit and Decode Polar Encoded Data

Transmit polar-encoded block of data and decode it using successive-cancellation list decoder.

Initial Setup

Create a channel that adds white Gaussian noise (WGN) to an input signal. Set the noise variance to 1.5.

```
nVar = 1.5;
chan = comm.AWGNChannel('NoiseMethod','Variance','Variance',nVar);
```

Create a binary phase shift keying (BSPK) modulator and demodulator.

```
bpskMod = comm.BPSKModulator;
bpskDemod = comm.BPSKDemodulator('DecisionMethod', ...
    'Approximate log-likelihood ratio','Variance',nVar);
```

Simulate a Frame

Perform polar encoding of a random message of length K. The rate-matched output is of length E.

```
K = 132;
E = 256;
msg = randi([0 1],K,1,'int8');
enc = nrPolarEncode(msg,E);
```

Modulate the polar encoded data using BSPK modulation, add WGN, and demodulate.

```
mod = bpskMod(enc);
rSig = chan(mod);
rxLLR = bpskDemod(rSig);
```

Perform polar decoding using successive-cancellation list decoder of length L.

```
L = 8;
rxBits = nrPolarDecode(rxLLR,K,E,L);
```

Determine the number of bit errors.

```
numBitErrs = biterr(rxBits,msg);
disp(['Number of bit errors: ' num2str(numBitErrs)])
```

```
Number of bit errors: 0
```

The transmitted and received messages are identical.

Input Arguments

rec — Rate-recovered input
column vector of real values

Rate-recovered input, specified as a column vector of real values. The input `rec` represents the log-likelihood ratios per bit with a negative bipolar mapping. So a `0` is mapped to `1`, and a `1` is mapped to `-1`. The length of `rec` must be a power of two.

Data Types: `single` | `double`

K — Length of information block in bits
positive integer

Length of information block in bits, specified as a positive integer. `K` includes the CRC bits if applicable.

Data Types: `double`

E — Rate-matched output length in bits
positive integer

Rate-matched output length in bits, specified as a positive integer.

- If 18 ≤ K ≤ 25, E must be in the range K + 3 < E ≤ 8192.
- If K > 30, E must be in the range K < E ≤ 8192.

Data Types: double

L — Length of decoding list
power of two

Length of decoding list, specified as a power of two.

Data Types: double

padCRC — Prepadding before CRC encoding
false (default) | true

Prepadding before CRC encoding, specified as false or true. Set padCRC to true if the information block on the transmit end, before polar encoding, was prepadded with all ones before CRC encoding.

Data Types: logical

rnti — RNTI of UE
integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: double

nmax — Base-2 logarithm of rate-recovered input's maximum length
9 (default) | 10

Base-2 logarithm of rate-recovered input's maximum length, specified as 9 or 10.

- For DL configuration, specify 9.
- For UL configuration, specify 10.

If N is the length of rec in bits, $N \leq 2^{nmax}$, see TS 38.212 Section 5.3.1.2.

Data Types: double

iil — Output deinterleaving
true (default) | false

Output deinterleaving, specified as true or false.

- For DL configuration, specify true.
- For UL configuration, specify false.

Data Types: logical

CRClen — Number of appended CRC bits
24 (default) | 11 | 6

Number of appended CRC bits, specified as 24, 11, or 6.

- For DL configuration, specify 24.
- For UL configuration, specify 11 or 6.

The numbers 24, 11, and 6 correspond to the polynomials gCRC24C, gCRC11, and gCRC6, respectively, as described in TS 38.212. Section 5.1 [1].

Data Types: `double`

Output Arguments

`decbits` — Decoded message
column vector of binary values

Decoded message, returned as a `K`-by-1 column vector of binary values.

Data Types: `int8`

Version History

Introduced in R2018b

Polar decoding metric update
Behavior changed in R2020a

In releases R2019b and before, polar decoding uses the exact form of the expression $\log(1 + e^x)$ for internal metric evaluation. Starting in release R2020a, because the exact form leads to numerical instability for high SNR ranges, polar decoding approximates $\log(1 + e^x)$ as 0 for $x < 0$ and as x for $x \geq 0$. This approximation affects the results of the `nrPolarDecode` function, resulting in a marginal degradation of the BLER performance in a link-level simulation.

References

[1] 3GPP TS 38.212. "NR; Multiplexing and channel coding." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] Tal, I. and Vardy, A., "List decoding of Polar Codes", *IEEE Transactions on Information Theory*. Vol. 61, No. 5, pp. 2213-2226, May 2015.

[3] Niu, K., and Chen, K., "CRC-Aided Decoding of Polar Codes", *IEEE Communications Letters*, Vol. 16, No. 10, pp. 1668-1671, Oct. 2012.

[4] Stimming, A. B., Parizi, M. B., and Burg, A., "LLR-Based Successive Cancellation List Decoding of Polar Codes", *IEEE Transaction on Signal Processing*, Vol. 63, No. 19, pp.5165-5179, 2015.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The input argument `L` must be a compile-time constant. Include `{coder.Constant(L)}` in the `-args` value of the `codegen` function. For more information, see `coder.Constant`.

See Also

Functions
`nrPolarEncode` | `nrRateRecoverPolar` | `nrCRCDecode` | `nrDCIDecode` | `nrUCIDecode`

Topics
"5G New Radio Polar Coding"

nrPolarEncode

Polar encoding

Syntax

```
enc = nrPolarEncode(in,E)
enc = nrPolarEncode(in,E,nmax,iil)
```

Description

`enc = nrPolarEncode(in,E)` returns the polar-encoded output for the input message `in` and rate-matched output length `E` as specified in TS 38.212 Section 5 [1]. By default, input interleaving is enabled and the maximum length of the encoded message is 512. Use this syntax for downlink configuration.

`enc = nrPolarEncode(in,E,nmax,iil)` encodes the input with a specified maximum length of 2^{nmax} and input interleaving specified by `iil`.

- For downlink (DL) configuration, valid values for `nmax` and `iil` are 9 and `true`, respectively.
- For uplink (UL) configuration, valid values for `nmax` and `iil` are 10 and `false`, respectively.

Examples

Perform Polar Encoding

Perform polar encoding of a random message of length `K`. `E` specifies the length of the rate-matched output which is different from the length of the encoded message `enc`. The length of `enc` is always a power of two.

```
K = 132;
E = 300;
msg = randi([0 1],K,1,'int8');
enc = nrPolarEncode(msg,E)
```

```
enc = 512x1 int8 column vector

   0
   0
   0
   0
   0
   0
   1
   1
   1
   0
      ⋮
```

Transmit and Decode Polar Encoded Data

Transmit polar-encoded block of data and decode it using successive-cancellation list decoder.

Initial Setup

Create a channel that adds white Gaussian noise (WGN) to an input signal. Set the noise variance to 1.5.

```
nVar = 1.5;
chan = comm.AWGNChannel('NoiseMethod','Variance','Variance',nVar);
```

Create a binary phase shift keying (BSPK) modulator and demodulator.

```
bpskMod = comm.BPSKModulator;
bpskDemod = comm.BPSKDemodulator('DecisionMethod', ...
    'Approximate log-likelihood ratio','Variance',nVar);
```

Simulate a Frame

Perform polar encoding of a random message of length `K`. The rate-matched output is of length `E`.

```
K = 132;
E = 256;
msg = randi([0 1],K,1,'int8');
enc = nrPolarEncode(msg,E);
```

Modulate the polar encoded data using BSPK modulation, add WGN, and demodulate.

```
mod = bpskMod(enc);
rSig = chan(mod);
rxLLR = bpskDemod(rSig);
```

Perform polar decoding using successive-cancellation list decoder of length `L`.

```
L = 8;
rxBits = nrPolarDecode(rxLLR,K,E,L);
```

Determine the number of bit errors.

```
numBitErrs = biterr(rxBits,msg);
disp(['Number of bit errors: ' num2str(numBitErrs)])
```

```
Number of bit errors: 0
```

The transmitted and received messages are identical.

Input Arguments

in — Input message
column vector of binary values

Input message, specified as a column vector of binary values. `in` includes the CRC bits if applicable.

Data Types: `double` | `int8`

E — Rate-matched output length in bits
positive integer

Rate-matched output length in bits, specified as a positive integer. E depends on K, the length of the input message in.

- If $18 \leq K \leq 25$, E must be in the range $K + 3 <$ E ≤ 8192.
- If $K > 30$, E must be in the range $K <$ E ≤ 8192.

Data Types: double

nmax — Base-2 logarithm of the encoded message's maximum length
9 (default) | 10

Base-2 logarithm of the encoded message's maximum length, specified as 9 or 10.

- For DL configuration, specify 9.
- For UL configuration, specify 10.

If N is the length of the polar-encoded message in bits, then $N \leq 2^{nmax}$. See TS 38.212 Section 5.3.1.2 [1].

Data Types: double

iil — Input interleaving
true (default) | false

Input interleaving, specified as true or false.

- For DL configuration, specify true.
- For UL configuration, specify false.

Data Types: logical

Output Arguments

enc — Polar-encoded message
column vector of binary values

Polar-encoded message, returned as a column vector of binary values. enc inherits its data type from the input message in.

The length of the polar-encoded message, N, is a power of two. For more information, see TS 38.212 Section 5.3.1.

- For DL configuration, $N \leq 512$.
- For UL configuration, $N \leq 1024$.

Data Types: double | int8

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.212. "NR; Multiplexing and channel coding." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*..

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrPolarDecode` | `nrRateMatchPolar` | `nrDCIEncode` | `nrUCIEncode` | `nrCRCEncode`

Topics
"5G New Radio Polar Coding"

nrPRACH

Generate PRACH symbols

Syntax

```
[sym,info] = nrPRACH(carrier,prach)
[sym,info] = nrPRACH(carrier,prach,'OutputDataType',datatype)
```

Description

`[sym,info] = nrPRACH(carrier,prach)` returns the physical random access channel (PRACH) symbols, as defined in TS 38.211 Section 6.3.3 [1]. The input `carrier` specifies carrier configuration parameters for a specific OFDM numerology. The input `prach` specifies PRACH configuration parameters. The function also returns the structure `info`, which contains carrier-dependent information about the PRACH.

`[sym,info] = nrPRACH(carrier,prach,'OutputDataType',datatype)` specifies the data type of the PRACH symbols.

Examples

Generate and Map PRACH Symbols to Grid

Configure the PRACH and the carrier with default properties.

```
carrier = nrCarrierConfig;
prach = nrPRACHConfig;
```

Generate PRACH symbols and indices using the specified carrier and PRACH configuration parameters.

```
prachSym = nrPRACH(carrier,prach);
prachInd = nrPRACHIndices(carrier,prach);
```

Generate a PRACH resource grid of all zeros.

```
prachGrid = nrPRACHGrid(carrier,prach);
```

Map the PRACH symbols to the PRACH resource grid by using the indices.

```
prachGrid(prachInd) = prachSym;
```

Analyze PRACH Root Sequence Indices

Analyze physical root Zadoff-Chu sequence indices by generating 64 orthogonal PRACH preambles for two different PRACH configurations.

Root Sequence Indices with Single Value

Configure the PRACH and the carrier with default properties.

```
carrier = nrCarrierConfig;
prach1 = nrPRACHConfig;
```

Set the PRACH logical root sequence index to `0`. For this value, the physical root sequence index is `129`, as defined in TS 38.211 Table 6.3.3.1-3.

```
prach1.SequenceIndex = 0;
```

Set the PRACH cyclic shift configuration index to `1`. For this value, each PRACH preamble has a different cyclic shift value, based on N_{CS} from TS 38.211 Table 6.3.3.1-5.

```
prach1.ZeroCorrelationZone = 1;
```

Generate 64 PRACH preambles to store the physical root sequence indices and cyclic shift values.

```
rootSequence1 = NaN(1,64);
cyclicShift1 = NaN(1,64);
for preambleIndex = 0:63
    prach1.PreambleIndex = preambleIndex;
    [~,info] = nrPRACH(carrier,prach1);
    rootSequence1(preambleIndex+1) = info.RootSequence;
    cyclicShift1(preambleIndex+1) = info.CyclicShift;
end
```

Verify that in each preamble, the physical root sequence index is `129`, which is the expected value from configuring the logical root sequence index to `0`.

```
disp(rootSequence1)
```

```
  Columns 1 through 13

   129   129   129   129   129   129   129   129   129   129   129   129   129

  Columns 14 through 26

   129   129   129   129   129   129   129   129   129   129   129   129   129

  Columns 27 through 39

   129   129   129   129   129   129   129   129   129   129   129   129   129

  Columns 40 through 52

   129   129   129   129   129   129   129   129   129   129   129   129   129

  Columns 53 through 64

   129   129   129   129   129   129   129   129   129   129   129   129
```

Verify that each preamble has a different cyclic shift value.

```
disp(cyclicShift1)
```

```
  Columns 1 through 13
```

```
    0    13    26    39    52    65    78    91   104   117   130   143   156

  Columns 14 through 26

  169   182   195   208   221   234   247   260   273   286   299   312   325

  Columns 27 through 39

  338   351   364   377   390   403   416   429   442   455   468   481   494

  Columns 40 through 52

  507   520   533   546   559   572   585   598   611   624   637   650   663

  Columns 53 through 64

  676   689   702   715   728   741   754   767   780   793   806   819
```

Root Sequence Indices with Different Values

Configure another PRACH with default properties.

```
prach2 = nrPRACHConfig;
```

Set the PRACH logical root sequence index to `0`. For this value, the physical root sequence index is `129`, as defined in TS 38.211 Table 6.3.3.1-3.

```
prach2.SequenceIndex = 0;
```

Set the PRACH cyclic shift configuration index to `0`. For this value, each PRACH preamble has the same cyclic shift value, equal to `0`, based on TS 38.211 Table 6.3.3.1-5.

```
prach2.ZeroCorrelationZone = 0;
```

Generate 64 PRACH preambles to store the physical root sequence indices and cyclic shift values.

```
rootSequence2 = NaN(1,64);
cyclicShift2 = NaN(1,64);
for preambleIndex = 0:63
    prach2.PreambleIndex = preambleIndex;
    [~,info] = nrPRACH(carrier,prach2);
    rootSequence2(preambleIndex+1) = info.RootSequence;
    cyclicShift2(preambleIndex+1) = info.CyclicShift;
end
```

Check the physical root sequence indices and cyclic shift values. Even though the logical root sequence index, `prach.SequenceIndex`, is `0`, not every physical root sequence index value is the expected value of `129`. Because the cyclic shift value is zero in each preamble, the function `nrPRACH` obtains the physical root sequence indices by taking consecutive logical index values. The returned physical root sequence indices correspond to logical indices 0 to 63 from TS 38.211 Table 6.3.3.1-3.

```
disp(rootSequence2)
```

```
  Columns 1 through 13

  129   710   140   699   120   719   210   629   168   671    84   755   105

  Columns 14 through 26
```

```
   734    93   746    70   769    60   779     2   837     1   838    56   783

  Columns 27 through 39

   112   727   148   691    80   759    42   797    40   799    35   804    73

  Columns 40 through 52

   766   146   693    31   808    28   811    30   809    27   812    29   810

  Columns 53 through 64

    24   815    48   791    68   771    74   765   178   661   136   703
```

```
disp(cyclicShift2)
```

```
  Columns 1 through 13

     0     0     0     0     0     0     0     0     0     0     0     0     0

  Columns 14 through 26

     0     0     0     0     0     0     0     0     0     0     0     0     0

  Columns 27 through 39

     0     0     0     0     0     0     0     0     0     0     0     0     0

  Columns 40 through 52

     0     0     0     0     0     0     0     0     0     0     0     0     0

  Columns 53 through 64

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

Input Arguments

carrier — Carrier configuration parameters
nrCarrierConfig object

Carrier configuration parameters for a specific OFDM numerology, specified as an nrCarrierConfig object.

prach — PRACH configuration parameters
nrPRACHConfig object

PRACH configuration parameters, specified as an nrPRACHConfig object. The function uses only these properties of this input.

- FrequencyRange
- DuplexMode
- ConfigurationIndex
- SubcarrierSpacing

- `LRA`
- `SequenceIndex`
- `PreambleIndex`
- `RestrictedSet`
- `ZeroCorrelationZone`
- `ActivePRACHSlot`
- `NPRACHSlot`

`datatype` — Data type of output symbols
`'double'` (default) | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

`sym` — PRACH symbols
complex column vector | `[]`

PRACH symbols, returned as a complex column vector or an empty array. The number of symbols depends on the PRACH configuration `prach`. The function returns an empty array when the PRACH preamble is not active in the current slot.

Data Types: `single` | `double`

`info` — Carrier-dependent PRACH information
structure

Carrier-dependent PRACH information, returned as a structure containing these fields:

Fields	Description
`RootSequence`	Index or indices of physical root Zadoff-Chu sequence
`CyclicShift`	Cyclic shift or shifts of Zadoff-Chu sequence
`CyclicOffset`	Cyclic shift or shifts corresponding to a Doppler shift of $1/T_{SEQ}$, where T_{SEQ} is the length of the PRACH sequence (applies to restricted set only)
`NumCyclicShifts`	Number of cyclic shifts corresponding to a single PRACH preamble sequence

Note Logical root sequence index `prach.SequenceIndex` determines the returned physical root Zadoff-Chu sequence index `RootSequence`, based on TS 38.211 Table 6.3.3.1-3 and Table 6.3.3.1-4. However, if the preamble index within the cell, specified by `prach.PreambleIndex`, results in insufficient amount of cyclic shifts available at index `prach.SequenceIndex`, the function `nrPRACH` obtains the physical root sequence index by taking consecutive logical root sequence indices, following the process described in TS 38.211 Section 6.3.3.1. In this case, the value of `RootSequence` differs from the expected index, specified by `prach.SequenceIndex`. For an example, see “Analyze PRACH Root Sequence Indices” on page 1-263.

Version History

Introduced in R2020a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The `datatype` input argument must be compile-time constant. For example, when specifying `'single'` as the output data type, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPRACHIndices` | `nrPRACHGrid`

Objects
`nrPRACHConfig`

nrPRACHDetect

Detect PRACH transmission

Syntax

```
[index,offset,info] = nrPRACHDetect(carrier,prach,waveform)
[index,offset,info] = nrPRACHDetect( ___ ,Name=Value)
```

Description

`nrPRACHDetect` detects physical random access channel (PRACH) transmission in a time-domain waveform. The function generates an internal reference waveform to correlate the input waveform with the generated reference waveform and searches for correlation output peaks that are greater than the detection threshold. The function uses the position of the strongest peak in the correlator output to determine the detected preamble index and the associated timing offset.

`[index,offset,info] = nrPRACHDetect(carrier,prach,waveform)` detects PRACH transmission in the time-domain waveform `waveform` for carrier configuration parameters `carrier` and PRACH configuration parameters `prach`. The function generates the internal reference waveform for default PRACH preamble indices from 0 to 63 and selects a default detection threshold based on the input arguments. The function returns the detected PRACH preamble index, `index`, timing offset, `offset`, and detection information, `info`. If `waveform` contains multiple PRACH instances, the function returns the preamble index and timing offset related to the PRACH instance with the strongest peak in the correlation.

`[index,offset,info] = nrPRACHDetect( ___ ,Name=Value)` specifies options as name-value arguments in addition to the input arguments in the previous syntax. For example, `DetectionThreshold=0.5` sets the detection threshold for the correlation.

Examples

Detect PRACH Preamble with Timing Offset

Create a carrier configuration object with six resource blocks.

```
carrier = nrCarrierConfig(NSizeGrid=6);
```

Create a PRACH configuration object. Set the cyclic shift configuration index to `1` and the preamble index within the cell to `44`.

```
prach = nrPRACHConfig;
prach.ZeroCorrelationZone = 1;
prach.PreambleIndex = 44;
```

Generate a PRACH resource grid for the specified carrier and PRACH configuration.

```
prachGrid = nrPRACHGrid(carrier,prach);
```

Generate PRACH symbols and indices for the specified carrier and PRACH configuration.

```
prachSymbols = nrPRACH(carrier,prach);
prachIndices = nrPRACHIndices(carrier,prach);
```

Map the PRACH symbols to the PRACH resource grid.

```
prachGrid(prachIndices) = prachSymbols;
```

OFDM-modulate the PRACH resource grid to obtain a transmit waveform with PRACH instances.

```
tx = nrPRACHOFDMModulate(carrier,prach,prachGrid);
```

Create a received waveform by introducing a delay in the transmit waveform.

```
rx = [zeros(7,1); tx];
```

Detect PRACH instances in the received waveform. Confirm that the delay introduced in the waveform offsets the timing of the PRACH detection.

```
[index,offset] = nrPRACHDetect(carrier,prach,rx)
```

```
index = 44
```

```
offset = 7.1895
```

Input Arguments

`carrier` — Carrier configuration parameters
`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. Only these object properties are relevant for this function.

- `NSizeGrid`
- `SubcarrierSpacing`
- `CyclicPrefix`

`prach` — PRACH configuration parameters
`nrPRACHConfig` object

PRACH configuration parameters, specified as an `nrPRACHConfig` object. Apart from the `PreambleIndex` property of `prach`, all object properties are relevant for this function.

`waveform` — Time-domain waveform
complex-valued matrix

Time-domain waveform, specified as a complex-valued matrix of size *N*-by-*P*.

- *N* is the number of time-domain samples in the waveform. If *N* is less than the number of samples that is needed to analyze this configuration, the function appends zeros at the end of the waveform.
- *P* is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

Name-Value Pair Arguments

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

Example: `DetectionThreshold=0.5` sets the detection threshold for the correlation.

`DetectionThreshold` — Detection threshold

`[]` (default) | real number in the range [0, 1]

Detection threshold for the correlation, specified as `[]` or a real number in the range [0, 1]. When you set this input to `[]`, or if you do not specify this input, the function selects a default value based on these elements of the input arguments `prach` and `waveform`.

- PRACH format, specified by `prach.Format`.
- LRA value, specified by `prach.LRA`.
- N_{SYM} — Number of OFDM symbols per one PRACH preamble, which is equal to `prach.PRACHDuration` – 2 for PRACH format C2 and `prach.PRACHDuration` for all other PRACH formats.
- P — Number of receive antennas, which is the number of columns in the input waveform `waveform`.

This table provides the formulas that the function uses to calculate default detection thresholds for different LRA values.

LRA	Default Detection Threshold
839	$\frac{0.02}{\sqrt{P \times N_{SYM}}}$
139	$\frac{0.1}{\sqrt{P \times N_{SYM}}}$
1151	$\frac{0.01}{\sqrt{P \times N_{SYM}}} + 0.0005$
571	$\frac{0.02}{\sqrt{P \times N_{SYM}}} + 0.001$

Data Types: `double`

`PreambleIndex` — Set of PRACH preamble indices

`[]` (default) | integer-valued array

Set of PRACH preamble indices within the cell for the detection, specified as `[]` or an integer-valued array. The array must have at most 64 elements with integer values in the range from 0 to 63. When you set this input to `[]`, or if you do not specify this input, the function uses default PRACH preamble indices from 0 to 63.

Data Types: `double`

Output Arguments

`index` — Detected PRACH preamble index
nonnegative integer | `[]`

Detected PRACH preamble index, returned as one of these values.

- Nonnegative integer — The index corresponds to the strongest correlation peak across default preamble indices from 0 to 63 or the indices that you specify using the `PreambleIndex` name-value input argument.
- `[]` — No correlation peak exists above the detection threshold.

Data Types: `double`

`offset` — Timing offset of PRACH waveform in samples
real number | `[]`

Timing offset of the PRACH waveform in samples, from the origin of the input waveform, returned as one of these values.

- Real number — The integer part of this number is the sample position of the strongest correlation peak. The fractional part of this number is the fractional delay present in the correlation peak due to the cyclic shift in the frequency domain.
- `[]` — No correlation peak exists above the detection threshold.

Data Types: `double`

`info` — Detection information
structure

Detection information, returned as a structure containing these fields.

Fields	Description
`CorrelationPeaks`	Strongest correlation peak values, where each value corresponds to default preamble indices from 0 to 63 or the indices that you specify using the `PreambleIndex` name-value input argument.
`DetectionThreshold`	Detection threshold that the function uses for the correlation.

Version History

Introduced in R2022b

Extended Capabilities

C/C++ Code Generation

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

Names and values in name-value arguments must be compile-time constants. For example, when specifying a detection threshold of 0.1, include `{coder.Constant('DetectionThreshold'),coder.Constant(0.1)}` in the `-args` value of the `codegen` function. For more information, see `coder.Constant`.

See Also

Functions
`nrPRACH` | `nrPRACHGrid` | `nrPRACHIndices` | `nrPRACHOFDMInfo` | `nrPRACHOFDMModulate`

Objects
`nrPRACHConfig`

nrPRACHGrid

Generate PRACH resource grid

Syntax

```
grid = nrPRACHGrid(carrier,prach)
grid = nrPRACHGrid(carrier,prach,p)
grid = nrPRACHGrid( ___ ,'OutputDataType',datatype)
```

Description

`grid = nrPRACHGrid(carrier,prach)` returns the physical random access channel (PRACH) resource grid for one antenna. The input `carrier` specifies carrier configuration parameters for a specific OFDM numerology. The input `prach` specifies PRACH configuration parameters.

`grid = nrPRACHGrid(carrier,prach,p)` returns the PRACH resource grid for `p` antennas.

`grid = nrPRACHGrid( ___ ,'OutputDataType',datatype)` specifies the resource grid symbol data type in addition to the input arguments in any of the previous syntaxes.

Examples

Generate and Map PRACH Symbols to Grid

Configure the PRACH and the carrier with default properties.

```
carrier = nrCarrierConfig;
prach = nrPRACHConfig;
```

Generate PRACH symbols and indices using the specified carrier and PRACH configuration parameters.

```
prachSym = nrPRACH(carrier,prach);
prachInd = nrPRACHIndices(carrier,prach);
```

Generate a PRACH resource grid of all zeros.

```
prachGrid = nrPRACHGrid(carrier,prach);
```

Map the PRACH symbols to the PRACH resource grid by using the indices.

```
prachGrid(prachInd) = prachSym;
```

Input Arguments

`carrier` — Carrier configuration parameters
`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object.

prach — PRACH configuration parameters
`nrPRACHConfig` object

PRACH configuration parameters, specified as an `nrPRACHConfig` object. The function uses only these properties of this input.

- `FrequencyRange`
- `DuplexMode`
- `ConfigurationIndex`
- `SubcarrierSpacing`

p — Number of antennas
positive integer

Number of antennas, specified as a positive integer.

datatype — Data type of output symbols
`'double'` (default) | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

grid — PRACH resource grid
K-by-*L* or *K*-by-*L*-by-`p` complex array of all zeros

PRACH resource grid, returned as a *K*-by-*L* or *K*-by-*L*-by-`p` complex array of all zeros.

- *K* is equal to (`carrier`.`SubcarrierSpacing` / `prach`.`SubcarrierSpacing`) × `carrier`.`NSizeGrid` × 12
- *L* is the number of OFDM symbols and depends on the PRACH format.
  - For long formats, *L* = `prach`.`PRACHDuration`.
  - For short format C0, *L* = 7.
  - For all other short formats, *L* = 14.

    For more information on PRACH preamble formats, see TS 38.211 Tables 6.3.3.1-1 and 6.3.3.1-2.

Data Types: `double` | `single`

Version History

Introduced in R2020a

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

The `datatype` input argument must be compile-time constant. For example, when specifying `'single'` as the output data type, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPRACH` | `nrPRACHIndices`

Objects
`nrPRACHConfig`

nrPRACHIndices

Generate PRACH resource element indices

Syntax

```
[ind,info] = nrPRACHIndices(carrier,prach)
[ind,info] = nrPRACHIndices(carrier,prach,Name,Value)
```

Description

`[ind,info] = nrPRACHIndices(carrier,prach)` returns resource element indices `ind` for the physical random access channel (PRACH), as defined in TS 38.211 Section 5.3.2 [1]. The input `carrier` specifies carrier configuration parameters for a specific OFDM numerology. The input `prach` specifies PRACH configuration parameters. The function also returns the structure `info`, which contains carrier-dependent information about the PRACH.

`[ind,info] = nrPRACHIndices(carrier,prach,Name,Value)` specifies output formatting options using one or more name-value pair arguments.

Examples

Generate and Map PRACH Symbols to Grid

Configure the PRACH and the carrier with default properties.

```
carrier = nrCarrierConfig;
prach = nrPRACHConfig;
```

Generate PRACH symbols and indices using the specified carrier and PRACH configuration parameters.

```
prachSym = nrPRACH(carrier,prach);
prachInd = nrPRACHIndices(carrier,prach);
```

Generate a PRACH resource grid of all zeros.

```
prachGrid = nrPRACHGrid(carrier,prach);
```

Map the PRACH symbols to the PRACH resource grid by using the indices.

```
prachGrid(prachInd) = prachSym;
```

Input Arguments

`carrier` — Carrier configuration parameters
`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object.

prach — PRACH configuration parameters
nrPRACHConfig object

PRACH configuration parameters, specified as an nrPRACHConfig object. The function uses only these properties of this input.

- FrequencyRange
- DuplexMode
- ConfigurationIndex
- SubcarrierSpacing
- LRA
- RBOffset
- FrequencyStart
- FrequencyIndex
- RBSetOffset
- ActivePRACHSlot
- NPRACHSlot

Name-Value Pair Arguments

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

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

Example: 'IndexStyle','subscript','IndexBase','0based' specifies the indexing form and indexing base of the output.

IndexStyle — RE indexing form
'index' (default) | 'subscript'

RE indexing form, specified as one of these values:

- 'index' — The indices are in linear index form.
- 'subscript' — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: char | string

IndexBase — RE indexing base
'1based' (default) | '0based'

RE indexing base, specified as one of these values:

- '1based' — The index counting starts from 1.
- '0based' — The index counting starts from 0.

Data Types: char | string

Output Arguments

`ind` — PRACH resource element indices
M-by-1 vector (default) | *M*-by-3 matrix

PRACH resource element indices, returned as one of these values.

- *M*-by-1 vector — When '`IndexStyle`' is set to `'index'`.
- *M*-by-3 matrix — When '`IndexStyle`' is set to `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and number of antennas, respectively.

M depends on the length of the Zadoff-Chu preamble sequence and is equal to `prach.LRA`.

Depending on '`IndexBase`', the indices are either 1-based or 0-based.

Data Types: `uint32`

`info` — Carrier-dependent PRACH information
structure

Carrier-dependent PRACH information, returned as a structure containing one field.

Field	Description
`PRBSet`	Physical resource block (PRB) indices occupied by the PRACH preamble for the PUSCH (0-based)

Version History

Introduced in R2020a

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPRACH` | `nrPRACHGrid`

Objects
nrPRACHConfig

nrPRACHOFDMInfo

Get PRACH OFDM information

Syntax

```
info = nrPRACHOFDMInfo(carrier,prach)
info = nrPRACHOFDMInfo(carrier,prach,'Windowing',samples)
```

Description

`info = nrPRACHOFDMInfo(carrier,prach)` provides dimensional information relevant to physical random access channel (PRACH) orthogonal frequency-division multiplexing (OFDM) modulation for carrier configuration parameters `carrier` and PRACH configuration parameters `prach`.

`info = nrPRACHOFDMInfo(carrier,prach,'Windowing',samples)` specifies the number of time-domain samples over which the function applies raised cosine windowing and overlapping of OFDM symbols.

Examples

Get PRACH OFDM Information

Specify carrier configuration parameters.

```
carrier = nrCarrierConfig;
```

Configure PRACH for format A1.

```
prach = nrPRACHConfig('ConfigurationIndex',106,'SubcarrierSpacing',15);
```

Generate and display the PRACH OFDM information.

```
info = nrPRACHOFDMInfo(carrier,prach)
```

```
info = struct with fields:
                   Nfft: 1024
             SampleRate: 15360000
    CyclicPrefixLengths: [152 0 144 0 144 0 152 0 144 0 144 0 0 0]
           GuardLengths: [0 0 0 0 0 0 0 0 0 0 0 0 0 144]
          SymbolLengths: [1176 1024 1168 1024 1168 1024 1176 1024 1168 ... ]
           OffsetLength: 0
              Windowing: 72
```

Get PRACH OFDM Information for Custom Windowing

Set carrier configuration parameters, specifying a subcarrier spacing of 60 kHz.

```
carrier = nrCarrierConfig('SubcarrierSpacing',60);
```

Configure and generate PRACH symbols.

```
prach = nrPRACHConfig;
```

Generate and display the PRACH OFDM information, specifying the number of samples over which the OFDM modulator applies windowing and overlapping of OFDM symbols.

```
samples = 95;
info = nrPRACHOFDMInfo(carrier,prach,'Windowing',samples)
```

```
info = struct with fields:
                  Nfft: 49152
            SampleRate: 61440000
    CyclicPrefixLengths: 6336
          GuardLengths: 5952
         SymbolLengths: 61440
          OffsetLength: 0
             Windowing: 95
```

Input Arguments

carrier — Carrier configuration parameters
nrCarrierConfig object

Carrier configuration parameters for a specific OFDM numerology, specified as an nrCarrierConfig object. Only these object properties are relevant for this function.

- NSizeGrid
- SubcarrierSpacing
- CyclicPrefix

prach — PRACH configuration parameters
nrPRACHConfig object

PRACH configuration parameters, specified as an nrPRACHConfig object. The function uses only these properties of this input.

- FrequencyRange
- DuplexMode
- ConfigurationIndex
- SubcarrierSpacing
- LRA
- NPRACHSlot

samples — Number of time-domain samples for OFDM symbol windowing and overlapping
nonnegative integer (default depends other input values) | []

Number of time-domain samples over which the function applies raised cosine windowing and overlapping of OFDM symbols, specified as a nonnegative integer or [].

If you do not specify this input, or if you specify it as `[]`, the function sets this input to the maximum value E that does not impact error vector magnitude (EVM) tests, as specified in TS 38.101-1 Annex F.5.5 and TS 38.101-2 Annex F.5.5. E is equal to value of `floor`($(N_{CP} - W) \times$ `info.Nfft`$/N_{FFT,\ nominal}$), where N_{CP}, W, and $N_{FFT,\ nominal}$ are the values in the table columns labeled "Cyclic prefix length", "EVM window length", and "FFT size", respectively.

Data Types: `double`

Output Arguments

`info` — OFDM information

structure

OFDM information, returned as a structure containing these fields.

Fields	Values	Description
`Nfft`	Positive integer	Number of FFT points
`SampleRate`	Positive integer	Waveform sample rate
`CyclicPrefixLengths`	1-by-N vector of nonnegative integers, where N is the number of OFDM symbols in a PRACH slot	Cyclic prefix lengths of each OFDM symbol, in samples
`GuardLengths`	1-by-N vector of positive integers, where N is the number of OFDM symbols in a PRACH slot	Guard lengths of OFDM symbols, in samples
`SymbolLengths`	1-by-N vector of nonnegative integers, where N is the number of OFDM symbols in a PRACH slot	OFDM symbol lengths, in samples
`OffsetLength`	Nonnegative integer	Length, in samples, of the initial time offset between the start of the configured PRACH slot period to the start of the cyclic prefix
`Windowing`	Nonnegative integer	Number of time-domain samples over which the function applies raised cosine windowing and overlapping of OFDM symbols

For long formats, for which the `LRA` property of the `prach` input is `839`, the first slot of a PRACH preamble can occur part of the way through the nominal PRACH slot period. In this case, the function increases the value of the `OffsetLength` field, which ensures that the OFDM waveform spans the entire active PRACH preamble. To balance these slots with the nominal PRACH slot period, some inactive PRACH slots have OFDM waveforms that are shorter than the nominal PRACH slot period. The function conveys this by returning the `CyclicPrefixLengths` and `GuardLengths` fields as `[]`, corresponding to no OFDM symbols, and setting the `OffsetLength` field equal to the number of empty subframes required.

Data Types: `struct`

Version History

Introduced in R2020b

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.101-1. "NR; User Equipment (UE) radio transmission and reception; Part 1: Range 1 Standalone." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 38.101-2. "NR; User Equipment (UE) radio transmission and reception; Part 2: Range 2 Standalone." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The `samples` input argument must be compile-time constant. Include `{coder.Constant('Windowing'),coder.Constant(samples)}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPRACH` | `nrPRACHIndices` | `nrPRACHGrid` | `nrPRACHOFDMModulate`

Objects
`nrPRACHConfig`

nrPRACHOFDMModulate

Generate PRACH OFDM modulated waveform

Syntax

```
[waveform,info] = nrPRACHOFDMModulate(carrier,prach,grid)
[waveform,info] = nrPRACHOFDMModulate(carrier,prach,grid,'Windowing',samples)
```

Description

`[waveform,info] = nrPRACHOFDMModulate(carrier,prach,grid)` generates `waveform`, a physical random access channel (PRACH) time-domain waveform, by performing orthogonal frequency-division multiplexing (OFDM) modulation of PRACH carrier resource array `grid` for carrier configuration parameters `carrier` and PRACH configuration parameters `prach`. The function also returns `info`, a structure containing OFDM information.

`[waveform,info] = nrPRACHOFDMModulate(carrier,prach,grid,'Windowing',samples)` specifies the number of time-domain samples over which the function applies raised cosine windowing and overlapping of OFDM symbols.

Examples

Generate PRACH OFDM Modulated Waveform

Generate a PRACH waveform by performing OFDM modulation of a resource array that contains PRACH symbols.

Specify carrier configuration parameters.

```
carrier = nrCarrierConfig;
```

Configure PRACH for format A1.

```
prach = nrPRACHConfig('ConfigurationIndex',106,'SubcarrierSpacing',15);
```

Generate PRACH symbols and map to a PRACH slot resource grid.

```
sym = nrPRACH(carrier,prach);
ind = nrPRACHIndices(carrier,prach);
grid = nrPRACHGrid(carrier,prach);
grid(ind) = sym;
```

Generate the PRACH OFDM waveform by modulating the grid. Display PRACH OFDM information.

```
[waveform,info] = nrPRACHOFDMModulate(carrier,prach,grid);
disp(info)
```

```
                Nfft: 1024
          SampleRate: 15360000
    CyclicPrefixLengths: [152 0 144 0 144 0 152 0 144 0 144 0 0 0]
        GuardLengths: [0 0 0 0 0 0 0 0 0 0 0 0 0 144]
```

```
    SymbolLengths: [1176 1024 1168 1024 1168 1024 1176 1024 1168 ... ]
     OffsetLength: 0
        Windowing: 72
```

Generate PRACH OFDM Modulated Waveform with Custom Windowing

Generate a PRACH OFDM modulated waveform by modulating a carrier containing PRACH symbols, specifying the number of samples over which the modulator applies windowing and overlapping of OFDM symbols.

Specify carrier configuration parameters, specifying a subcarrier spacing of 60 kHz.

```
carrier = nrCarrierConfig('SubcarrierSpacing',60);
```

Configure and generate PRACH symbols and map to a PRACH slot resource grid.

```
prach = nrPRACHConfig('FrequencyRange','FR2','SubcarrierSpacing',60,'DuplexMode','TDD');
sym = nrPRACH(carrier,prach);
ind = nrPRACHIndices(carrier,prach);
grid = nrPRACHGrid(carrier,prach);
grid(ind) = sym;
```

Generate the PRACH OFDM waveform by modulating the grid, specifying the number of time domain samples over which the PRACH OFDM modulator applies windowing and overlapping of OFDM symbols.

```
samples = 80;
[waveform,info] = nrPRACHOFDMModulate(carrier,prach,grid,'Windowing',samples);
```

Input Arguments

carrier — Carrier configuration parameters
nrCarrierConfig object

Carrier configuration parameters for a specific OFDM numerology, specified as an nrCarrierConfig object. Only these object properties are relevant for this function.

- NSizeGrid
- SubcarrierSpacing
- CyclicPrefix

prach — PRACH configuration parameters
nrPRACHConfig object

PRACH configuration parameters, specified as an nrPRACHConfig object. The function uses only these properties of this input.

- FrequencyRange
- DuplexMode
- ConfigurationIndex

- `SubcarrierSpacing`
- `LRA`
- `NPRACHSlot`

`grid` — PRACH resource grid
complex array

PRACH resource grid, specified as a *K*-by-*L*-by-*P* complex array.

- *K* is the number of subcarriers equal to (`carrier`.`SubcarrierSpacing` / `prach`.`SubcarrierSpacing`) × `carrier`.`NSizeGrid` × 12
- *L* is the number of OFDM symbols and depends on the PRACH preamble format.
  - For long formats, *L* = `prach`.`PRACHDuration`.
  - For short format C0, *L* = 7.
  - For all other short formats, *L* = 14.

    For more information on PRACH preamble formats, see TS 38.211 Tables 6.3.3.1-1 and 6.3.3.1-2 [1].
- *P* is the number of transmit antennas.

Data Types: `double`
Complex Number Support: Yes

`samples` — Number of time-domain samples for OFDM symbol windowing and overlapping
nonnegative integer (default depends other input values) | [ ]

Number of time-domain samples over which the function applies raised cosine windowing and overlapping of OFDM symbols, specified as a nonnegative integer or `[]`.

If you do not specify this input, or if you specify it as `[]`, the function sets this input to the maximum value *E* that does not impact error vector magnitude (EVM) tests, as specified in TS 38.101-1 Annex F.5.5 and TS 38.101-2 Annex F.5.5. *E* is equal to value of `floor`($(N_{CP} - W) \times$ `info.Nfft/`$N_{FFT,\ nominal}$), where N_{CP}, W, and $N_{FFT,\ nominal}$ are the values in the table columns labeled "Cyclic prefix length", "EVM window length", and "FFT size", respectively.

Data Types: `double`

Output Arguments

`waveform` — PRACH OFDM modulated waveform
complex-valued matrix

PRACH OFDM modulated waveform, returned as a complex-valued matrix of size *T*-by-*P*.

- *T* is the number of time domain samples in the waveform for the current slot, equal to the value of `info.OffsetLengths` + `sum(info.SymbolLengths)`.

  The `NPRACHSlot` property of the `prach` input determines the current slot.
- *P* is the number of transmit antennas.

Data Types: `double`
Complex Number Support: Yes

info — OFDM information
structure

OFDM information, returned as a structure containing these fields.

Fields	Values	Description
Nfft	Positive integer	Number of FFT points
SampleRate	Positive integer	Waveform sample rate
CyclicPrefixLengths	1-by-*N* vector of nonnegative integers, where *N* is the number of OFDM symbols in a PRACH slot	Cyclic prefix lengths of each OFDM symbol, in samples
GuardLengths	1-by-*N* vector of positive integers, where *N* is the number of OFDM symbols in a PRACH slot	Guard lengths of OFDM symbols, in samples
SymbolLengths	1-by-*N* vector of nonnegative integers, where *N* is the number of OFDM symbols in a PRACH slot	OFDM symbol lengths, in samples
OffsetLength	Nonnegative integer	Length, in samples, of the initial time offset between the start of the configured PRACH slot period to the start of the cyclic prefix
Windowing	Nonnegative integer	Number of time-domain samples over which the function applies raised cosine windowing and overlapping of OFDM symbols

For long formats, for which the `LRA` property of the `prach` input is `839`, the first slot of a PRACH preamble can occur part of the way through the nominal PRACH slot period. In this case, the function increases the value of the `OffsetLength` field, which ensures that the OFDM waveform spans the entire active PRACH preamble. To balance these slots with the nominal PRACH slot period, some inactive PRACH slots have OFDM waveforms that are shorter than the nominal PRACH slot period. The function conveys this by returning the `CyclicPrefixLengths` and `GuardLengths` fields as `[]`, corresponding to no OFDM symbols, and setting the `OffsetLength` field equal to the number of empty subframes required.

Data Types: `struct`

Version History

Introduced in R2020b

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.101-1. “NR; User Equipment (UE) radio transmission and reception; Part 1: Range 1 Standalone.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 38.101-2. “NR; User Equipment (UE) radio transmission and reception; Part 2: Range 2 Standalone.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The `samples` input argument must be compile-time constant. Include `{coder.Constant('Windowing'),coder.Constant(samples)}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPRACH` | `nrPRACHGrid` | `nrPRACHIndices` | `nrPRACHOFDMInfo`

Objects
`nrPRACHConfig`

nrPRBS

Generate PRBS

Syntax

```
[seq,cinit] = nrPRBS(cinit,n)
[seq,cinit] = nrPRBS(cinit,n,Name,Value)
```

Description

`[seq,cinit] = nrPRBS(cinit,n)` returns the elements specified by `n` of the pseudorandom binary sequence (PRBS) generator, when initialized with `cinit`. The function implements the generator specified in TS 38.211 Section 5.2.1 [1] on page 1-291. For uniformity with the channel-specific PRBS functions, the function also returns the initialization value `cinit`.

`[seq,cinit] = nrPRBS(cinit,n,Name,Value)` specifies additional output formatting options by using one or more name-value pair arguments. Unspecified options take their default values.

Examples

Generate Pseudorandom Scrambling Sequence

Generate a 1000-bit binary scrambling sequence. Initialize the PRBS generator with the specified value.

```
cinit = 9;
prbs = nrPRBS(cinit,1000);
```

Input Arguments

`cinit` — Initialization value for PRBS generator

integer from 0 to $2^{31} - 1$

Initialization value for the PRBS generator, specified as an integer from 0 to $2^{31} - 1$.

Data Types: `double`

`n` — Elements in returned sequence

nonnegative integer | `[p m]` row vector

Elements in returned sequence, specified as one of these values:

- Nonnegative integer — `seq` contains the first `n` elements of the PRBS generator.
- `[p m]` row vector — `seq` contains `m` contiguous elements of the PRBS generator, starting at position `p` (0-based).

Data Types: `double`

Name-Value Pair Arguments

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

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

Example: `'MappingType','signed'` specify non-default sequence formatting properties.

`MappingType` — Output sequence formatting

`'binary'` (default) | `'signed'`

Output sequence formatting, specified as the comma-separated pair consisting of `'MappingType'` and one of these values:

- `'binary'` — This value maps `true` to `1` and `false` to `0`. The data type of the output sequence is `logical`.
- `'signed'` — This value maps `true` to `–1` and `false` to `1`. The data type of the output sequence is `double`. To specify `single` data type, use the `'OutputDataType'` name-value pair.

Data Types: `char` | `string`

`OutputDataType` — Data type of output sequence

`'double'` (default) | `'single'`

Data type of output sequence, specified as the comma-separated pair consisting of `'OutputDataType'` and `'double'` or `'single'`. This name-value pair applies only when `'MappingType'` is set to `'signed'`.

Data Types: `char` | `string`

Output Arguments

`seq` — Pseudorandom scrambling sequence

logical column vector | numeric column vector

Pseudorandom scrambling sequence, returned as a logical or numeric column vector. The output `seq` contains the elements of the PRBS generator specified by `n`. If you set `'MappingType'` to `'signed'`, the data type of `seq` is either `double` or `single`. If you set `'MappingType'` to `'binary'`, the output data type is `logical`.

Data Types: `double` | `single` | `logical`

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying `'single'` as the output data type, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPBCHPRBS` | `nrPDCCHPRBS` | `nrPDSCHPRBS`

nrPRS

Generate PRS symbols

Syntax

```
sym = nrPRS(carrier,prs)
sym = nrPRS(carrier,prs,Name,Value)
```

Description

`sym = nrPRS(carrier,prs)` generates positioning reference signal (PRS) symbols `sym`, as defined in TS 38.211 Section 7.4.1.7.2 [1], for carrier configuration `carrier` and PRS configuration `prs`. The function also handles the conditions related to the mapping of PRS resources to slots, as defined in TS 38.211 Section 7.4.1.7.4.

`sym = nrPRS(carrier,prs,Name,Value)` specifies options by using one or more name-value arguments. For example, `'datatype','single'` sets the data type of the output symbols to `single`.

Examples

Generate PRS Symbols for Multiple Resources in Resource Set

Create a default carrier configuration object.

```
carrier = nrCarrierConfig;
```

Specify the extended cyclic prefix for a subcarrier spacing of 60 KHz and slot number 92.

```
carrier.SubcarrierSpacing = 60;
carrier.CyclicPrefix = 'extended';
carrier.NSlot = 92;
```

Create a default PRS configuration object.

```
prs = nrPRSConfig;
```

Set properties that are common to all PRS resources in a resource set.

```
prs.PRSResourceSetPeriod = [640 0];
prs.PRSResourceRepetition = 8;
prs.PRSResourceTimeGap = 4;
prs.MutingPattern1 = [1 0 0 1];
prs.MutingBitRepetition = 2;
prs.MutingPattern2 = [1 0 1 1 1 0 1 0];
prs.NumRB = 52;
prs.RBOffset = 0;
prs.CombSize = 2;
```

Set properties that are unique to each PRS resource in a resource set.

```
prs.PRSResourceOffset = [0 40 80 120];
prs.NumPRSSymbols = [6 4 12 2];
prs.SymbolStart = 0;
prs.REOffset = [0 1 0 1];
prs.NPRSID = 15;
```

Generate PRS symbols of `single` data type with a cell resource format.

```
sym = nrPRS(carrier,prs,'OutputDataType','single','OutputResourceFormat','cell')
```

```
sym=1×4 cell array
    {0x1 single}    {0x1 single}    {3744x1 single}    {0x1 single}
```

Generate PRS Symbols and Indices for Two Resources

Create a default carrier configuration object.

```
carrier = nrCarrierConfig;
```

Create a default PRS configuration object.

```
prs = nrPRSConfig;
```

Set properties related to the PRS slot configuration.

```
prs.PRSResourceSetPeriod = [8 0]; % Resource set periodicity of 8 slots and resource set slot of
prs.PRSResourceOffset = [0 4];    % Configure two PRS resources with slot offsets 0 and 4 relativ
prs.PRSResourceRepetition = 2;    % Repeat each PRS resource twice
prs.PRSResourceTimeGap = 1;       % Configure two PRS resource repetition indices with no time ga
```

Set properties related to the PRS muting configuration.

```
prs.MutingPattern1 = [1 1];       % Transmit all PRS resource set instances
prs.MutingBitRepetition = 1;      % One instance of a PRS resource set corresponding to a single
prs.MutingPattern2 = [1 0];       % Mute second repetition index of all the PRS resources within
```

Set properties related to the PRS time-domain allocation.

```
prs.NumPRSSymbols = [6 12];
prs.SymbolStart = [6 0];
```

Set properties related to the PRS frequency-domain allocation.

```
prs.NumRB = 40;
prs.RBOffset = 4;
prs.CombSize = 4;
prs.REOffset = [1 3];
prs.NPRSID = 5;       % Set PRS sequence identity
```

Get the number of orthogonal frequency division multiplexing (OFDM) symbols per slot.

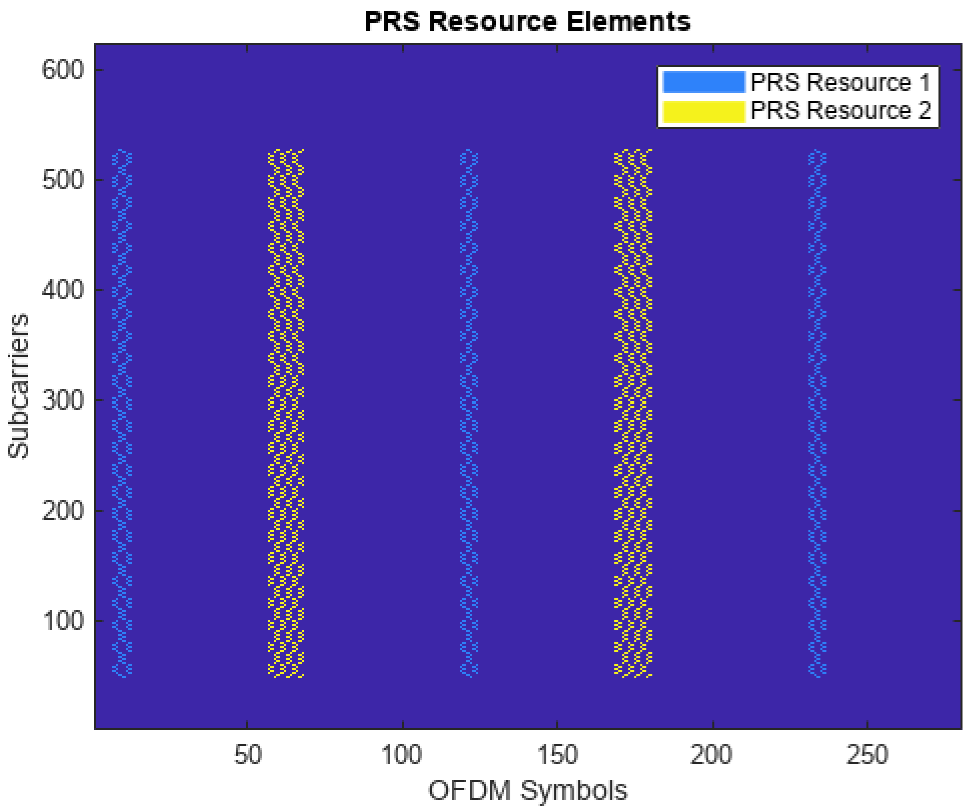
```
numSymPerSlot = carrier.SymbolsPerSlot;
```

Set the number of slots to 20.

```
numSlots = 20;
```

Map the resource elements (RE) for both of the PRS resources on the carrier resource grid.

```
grid = complex(zeros(carrier.NSizeGrid*12,carrier.SymbolsPerSlot*numSlots));
for slotIdx = 0:numSlots-1
    carrier.NSlot = slotIdx;
    indCell = nrPRSIndices(carrier,prs,'OutputResourceFormat','cell');
    symCell = nrPRS(carrier,prs,'OutputResourceFormat','cell');
    slotGrid = nrResourceGrid(carrier);
    slotGrid(indCell{1}) = 70*symCell{1};                          % Resource element mapping
    slotGrid(indCell{2}) = 250*symCell{2};                         % Resource element mapping
    grid(:,(1:numSymPerSlot)+numSymPerSlot*slotIdx) = slotGrid;
end
figure
image(abs(grid));
axis xy;
L = line(ones(2),ones(2),'LineWidth',8);              % Generate lines
set(L,{'color'},{[0.18 0.51 0.98]; [0.96 0.95 0.11]}); % Set the colors
legend('PRS Resource 1','PRS Resource 2');             % Create legend
title('PRS Resource Elements');
xlabel('OFDM Symbols');
ylabel('Subcarriers');
```



Input Arguments

carrier — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function uses only these `nrCarrierConfig` object properties.

`SubcarrierSpacing` — Subcarrier spacing in kHz
`15` (default) | `30` | `60` | `120` | `240`

Subcarrier spacing in kHz, for all channels and reference signals of the carrier, specified as `15`, `30`, `60`, `120`, or `240`.

Data Types: `double`

`CyclicPrefix` — Cyclic prefix length
`'normal'` (default) | `'extended'`

Cyclic prefix length, specified as one of these options.

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies for only 60 kHz subcarrier spacing.

Data Types: `char` | `string`

`NSizeGrid` — Number of RBs in carrier resource grid
`52` (default) | integer from 1 to 275

Number of RBs in the carrier resource grid, specified as an integer from 1 to 275. The default value of `52` corresponds to the maximum number of RBs of a 10 MHz carrier with 15 kHz SCS.

Data Types: `double`

`NStartGrid` — Start of carrier resource grid relative to CRB 0
`0` (default) | integer from 0 to 2199

Start of carrier resource grid relative to CRB 0, specified as an integer from 0 to 2199. This property is the higher-layer parameter *offsetToCarrier*.

Data Types: `double`

`NSlot` — Slot number
`0` (default) | nonnegative integer

Slot number, specified as a nonnegative integer. You can set `NSlot` to a value larger than the number of slots per frame. For example, you can set this value using transmission loop counters in a MATLAB simulation. In this case, you may have to ensure that the property value is modulo the number of slots per frame in a calling code.

Data Types: `double`

`NFrame` — System frame number
`0` (default) | nonnegative integer

System frame number, specified as a nonnegative integer. You can set `NFrame` to a value larger than the maximum frame number 1023. For example, you can set this value using transmission loop counters in a MATLAB simulation. In this case, you may have to ensure that the property value is modulo 1024 in a calling code.

Data Types: `double`

`prs` — PRS configuration parameters
`nrPRSConfig` object

PRS configuration parameters, specified as an `nrPRSConfig` object. This function uses only these `nrPRSConfig` object properties.

`PRSResourceSetPeriod` — PRS resource set slot periodicity and slot offset
`'on'` (default) | `'off'` | two-element vector

PRS resource set slot periodicity and slot offset, specified as one of these options.

- `'on'` — All of the PRS resources are present in the operating slot.
- `'off'` — All of the PRS resources are absent in the operating slot.
- Two-element vector of the form [*TPRSPeriod*, *TPRSOffset*] — *TPRSPeriod* is the resource set slot periodicity. The nominal value of *TPRSPeriod* must equal 2^{μ} multiplied by one of the values in the set {4, 5, 8, 10, 16, 20, 32, 40, 64, 80, 160, 320, 640, 1280, 2560, 5120, 10,240}, where μ is the subcarrier spacing configuration with a value of 0, 1, 2, or 3. *TPRSOffset* is the resource set slot offset and must equal a value in the range [0, *TPRSPeriod* – 1].

Data Types: `double` | `string` | `char`

`PRSResourceOffset` — Slot offset of each PRS resource
`0` (default) | scalar in the range [0, 511] | vector of integers in the range [0, 511]

Slot offset of each PRS resource (0-based) provided by the higher layer parameter *dl-PRS-ResourceSlotOffset-r16*, specified as a scalar in the range [0, 511] or a vector of integers in the range [0, 511]. This property represents the starting slot offset of a PRS resource relative to the PRS resource set offset (*TPRSOffset*).

Configure this property for each resource in a resource set separately based on one of these options.

- When a single resource is present, specify this property as a scalar.
- When multiple resources are present, specify this property as a scalar or vector. If you specify a scalar, the object applies that value to all of the PRS resources in a PRS resource set. If you specify a vector, the object applies the vector element values to the corresponding PRS resource. The length of this vector must be equal to the number of PRS resources to be configured in a PRS resource set.

Dependencies

To enable this property, set the `PRSResourceSetPeriod` property to a two-element vector of the form [*TPRSPeriod*, *TPRSOffset*].

Data Types: `double`

`PRSResourceRepetition` — PRS resource repetition factor
`1` (default) | `2` | `4` | `6` | `8` | `16` | `32`

PRS resource repetition factor provided by the higher layer parameter *dl-PRS-ResourceRepetitionFactor-r16*, specified as `1`, `2`, `4`, `6`, `8`, `16`, or `32`. This property value is the same for all of the PRS resources in a PRS resource set.

Dependencies

To enable this property, set the `PRSResourceSetPeriod` property to a two-element vector of the form [*TPRSPeriod*, *TPRSOffset*].

Data Types: `double`

`PRSResourceTimeGap` — Slot offset between two consecutive repeated instances of a PRS resource

1 (default) | 2 | 4 | 8 | 16 | 32

Slot offset between two consecutive repeated instances of a PRS resource, specified as `1`, `2`, `4`, `8`, `16`, or `32`. The property represents the offset in terms of the number of slots between two repeated instances of a PRS resource. This property value is same the for all of the PRS resources in a PRS resource set. This property is the higher layer parameter *dl-PRS-ResourceTimeGap-r16*.

Dependencies

To enable this property, set the `PRSResourceSetPeriod` property to a two-element vector of the form [*TPRSPeriod*, *TPRSOffset*] and the `PRSResourceRepetition` property to a value greater than 1.

Data Types: `double`

`MutingPattern1` — Muting bit pattern option-1

`[]` (default) | binary-valued vector of length 2, 4, 6, 8, 16, or 32

Muting bit pattern option-1, specified as `[]` or a binary-valued vector of length 2, 4, 6, 8, 16, or 32.

- If the higher layer parameter *mutingOption1-r16* is configured, set this property to a binary-valued vector of length 2, 4, 6, 8, 16, or 32. Each element in the vector corresponds to a number of consecutive instances of a PRS resource set based on the `MutingBitRepetition` property. The vector element also indicates whether all of the PRS resources within the PRS resource set instances are transmitted (binary 1) or muted (binary 0).
- If the higher layer parameter *mutingOption1-r16* is not configured, set this property to `[]`. To disable the muting bit pattern option-1, set this property to `[]`.

Dependencies

To enable this property, set the `PRSResourceSetPeriod` property to a two-element vector of the form [*TPRSPeriod*, *TPRSOffset*].

Data Types: `double`

`MutingBitRepetition` — Muting bit repetition factor

1 (default) | 2 | 4 | 8

Muting bit repetition factor provided by the higher layer parameter *dl-PRS-MutingBitRepetitionFactor-r16*, specified as `1`, `2`, `4`, or `8`. This property indicates the number of consecutive instances of the PRS resource set, *N*, corresponding to each element of the `MutingPattern1` property. The first element in `MutingPattern1` corresponds to the first *N* instances of a PRS resource set, the second element corresponds to the next *N* instances of a PRS resource set, and so on.

Dependencies

To enable this property, set the `PRSResourceSetPeriod` property to a two-element vector of the form [*TPRSPeriod*, *TPRSOffset*] and the `MutingPattern1` property to a value other than `[]`.

Data Types: `double`

`MutingPattern2` — Muting bit pattern option-2
`[]` (default) | binary-valued vector of length 1, 2, 4, 6, 8, 16, or 32

Muting bit pattern option-2, specified as `[]` or a binary-valued vector of length 1, 2, 4, 6, 8, 16, or 32.

- If the higher layer parameter *mutingOption2-r16* is configured, set this property to a binary-valued vector of length 1, 2, 4, 6, 8, 16, or 32. Each element in the vector relates to the corresponding repetition index. Each element in the vector corresponds to a single repetition index of each PRS resource in an active instance of a PRS resource set. The vector element also indicates whether the repetition index for all of the PRS resources is transmitted (binary 1) or muted (binary 0).
- If the higher layer parameter *mutingOption2-r16* is not configured, set this property to `[]`. To disable the muting bit pattern option-2, set this property to `[]`.

Dependencies

To enable this property, set the `PRSResourceSetPeriod` property to a two-element vector of the form [*TPRSPeriod*, *TPRSOffset*].

Data Types: `double`

`NumPRSSymbols` — Number of consecutive OFDM symbols allocated for each PRS resource
`12` (default) | scalar in the range [0, 12] | vector of integers in the range [0, 12]

Number of consecutive OFDM symbols allocated for each PRS resource, specified as a scalar in the range [0, 12] or a vector of integers in the range [0, 12].

Configure this property for each resource in a resource set separately based on one of these options.

- When a single resource is present, specify this property as a scalar.
- When multiple resources are present, specify this property as a scalar or vector. If you specify a scalar, the object applies that value to all of the PRS resources in a PRS resource set. If you specify a vector, the object applies the vector element values to the corresponding PRS resource. The length of this vector must be equal to the number of PRS resources to be configured in a PRS resource set.

This property is the higher layer parameter *dl-PRS-NumSymbols-r16*, and the nominal value of this property must be `2`, `4`, `6`, or `12`. To indicate no PRS resource allocation, set this property to `0`.

Data Types: `double`

`SymbolStart` — Starting OFDM symbol of each PRS resource in slot
`0` (default) | scalar in the range [0, 13] | vector of integers in the range [0, 13]

Starting OFDM symbol of each PRS resource in a slot (0-based), specified as a scalar in the range [0, 13] or a vector of integers in the range [0, 13].

Configure this property for each resource in a resource set separately based on one of these options.

- When a single resource is present, specify this property as a scalar.
- When multiple resources are present, specify this property as a scalar or vector. If you specify a scalar, the object applies that value to all of the PRS resources in a PRS resource set. If you specify a vector, the object applies the vector element values to the corresponding PRS resource. The length of this vector must be equal to the number of PRS resources to be configured in a PRS resource set.

This property is the higher layer parameter *dl-PRS-ResourceSymbolOffset-r16*, and the nominal value of this property must be in the range [0, 12].

Data Types: `double`

`NumRB` — Number of PRBs allocated for all PRS resources
`52` (default) | scalar in the range [0, 275]

Number of PRBs allocated for all PRS resources in a resource set, specified as a scalar in the range [0, 275]. This property is the higher layer parameter *dl-PRS-ResourceBandwidth-r16*, and the nominal value of this property must be in the range [24, 272] with a granularity of 4 PRBs. To indicate no PRS resource allocation, set this property to `0`.

Data Types: `double`

`RBOffset` — Starting PRB index of all PRS resources relative to carrier resource grid
`0` (default) | scalar in the range [0, 274]

Starting PRB index of all PRS resources relative to the carrier resource grid, specified as a scalar in the range [0, 274].

Data Types: `double`

`CombSize` — Comb size of all PRS resources
`2` (default) | `4` | `6` | `12`

Comb size of all PRS resources in a resource set, specified as `2`, `4`, `6`, or `12`. The comb size represents the resource element spacing in each OFDM symbol. This property is the higher layer parameter *dl-PRS-CombSizeN-r16*. The value `2` specifies for the object to allocate every 2nd RE in the PRB for PRS, the value `4` specifies for the object to allocate every 4th RE in the PRB for PRS, and so on.

Data Types: `double`

`NPRSID` — Sequence identity of each PRS resource
`0` (default) | scalar in the range [0, 4095] | vector of integers in the range [0, 4095]

Sequence identity of each PRS resource provided by the higher layer parameter *dl-PRS-SequenceID-r16*, specified as a scalar in the range [0, 4095] or a vector of integers in the range [0, 4095].

Configure this property for each resource in a resource set separately based on one of these options.

- When a single resource is present, specify this property as a scalar.
- When multiple resources are present, specify this property as a scalar or vector. If you specify a scalar, the object applies that value to all of the PRS resources in a PRS resource set. If you specify a vector, the object applies the vector element values to the corresponding PRS resource. The length of this vector must be equal to the number of PRS resources to be configured in a PRS resource set.

Data Types: `double`

Note The maximum vector length of these five properties of `nrPRSConfig` object specifies the number of configured PRS resources.

- `PRSResourceOffset`
- `NumPRSSymbols`
- `SymbolStart`
- `REOffset`
- `NPRSID`

Name-Value Pair Arguments

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

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

Example: `'OutputDataType','single'` sets the data type of the output symbols to `single`.

`OutputDataType` — Data type of output symbols
`'double'` | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

`OutputResourceFormat` — Output format of PRS symbols
`'concatenated'` (default) | `'cell'`

Output format of PRS symbols, specified as one of these values.

- `'concatenated'` — The output `sym` is a single column vector containing all PRS symbols concatenated.
- `'cell'` — The output `sym` is a cell array where each cell corresponds to a single PRS resource.

Data Types: `char` | `string`

Output Arguments

`sym` — PRS symbols
column vector | cell array of column vectors

PRS symbols, returned as a column vector or a cell array of column vectors. If this output is a cell array of column vectors, each cell represents the PRS symbols corresponding to each PRS resource in a PRS resource set.

Data Types: `single` | `double`
Complex Number Support: Yes

Version History

Introduced in R2021a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation (Release 16)." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The `datatype` input argument must be compile-time constant. For example, when specifying `'single'` as the output data type, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Objects
`nrCarrierConfig` | `nrPRSConfig`

Functions
`nrPRSIndices`

Topics
"NR Positioning Reference Signal"

nrPRSIndices

Generate PRS resource element indices

Syntax

```
ind = nrPRSIndices(carrier,prs)
ind = nrPRSIndices(carrier,prs,Name,Value)
```

Description

`ind = nrPRSIndices(carrier,prs)` generates positioning reference signal (PRS) resource element (RE) indices `ind`, as defined in TS 38.211 Section 7.4.1.7.3 [1], for carrier configuration `carrier` and PRS configuration `prs`. The function also handles the conditions related to the mapping of PRS resources to slots, as defined in TS 38.211 Section 7.4.1.7.4.

`ind = nrPRSIndices(carrier,prs,Name,Value)` specifies options by using one or more name-value arguments. For example, `'IndexBase','0based'` specifies the RE indexing base as 0.

Examples

Generate PRS Indices for Multiple Resources in Resource Set

Create a default carrier configuration object.

```
carrier = nrCarrierConfig;
```

Create a default PRS configuration object.

```
prs = nrPRSConfig;
```

Set properties that are common to all PRS resources in a resource set.

```
prs.PRSResourceSetPeriod = [20 0];
prs.PRSResourceRepetition = 4;
prs.PRSResourceTimeGap = 2;
prs.MutingPattern1 = [1 0];
prs.MutingBitRepetition = 2;
prs.MutingPattern2 = [1 0 1 0];
prs.NumRB = 32;
prs.RBOffset = 10;
prs.CombSize = 4;
```

Set properties that are unique to each PRS resource in a resource set. You can specify these properties as a scalar or vector. If you specify a scalar, the object applies that value to all of the PRS resources in a PRS resource set. If you specify a vector, the object applies the vector element values to the corresponding PRS resource. The length of this vector must be equal to the number of PRS resources to be configured in a PRS resource set.

```
prs.PRSResourceOffset = [0 10];
prs.NumPRSSymbols = [6 4];
```

```
prs.SymbolStart = [0 1];
prs.REOffset = 0;
prs.NPRSID = [10 50];
```

Generate 0-based PRS RE indices with subscript RE indexing form.

```
ind = nrPRSIndices(carrier,prs,'IndexStyle','subscript','IndexBase','0based')
```

```
ind = 576x3 uint32 matrix

   120     0     0
   124     0     0
   128     0     0
   132     0     0
   136     0     0
   140     0     0
   144     0     0
   148     0     0
   152     0     0
   156     0     0
      ⋮
```

Generate PRS Symbols and Indices for Two Resources

Create a default carrier configuration object.

```
carrier = nrCarrierConfig;
```

Create a default PRS configuration object.

```
prs = nrPRSConfig;
```

Set properties related to the PRS slot configuration.

```
prs.PRSResourceSetPeriod = [8 0]; % Resource set periodicity of 8 slots and resource set slot of
prs.PRSResourceOffset = [0 4];    % Configure two PRS resources with slot offsets 0 and 4 relativ
prs.PRSResourceRepetition = 2;    % Repeat each PRS resource twice
prs.PRSResourceTimeGap = 1;       % Configure two PRS resource repetition indices with no time ga
```

Set properties related to the PRS muting configuration.

```
prs.MutingPattern1 = [1 1];       % Transmit all PRS resource set instances
prs.MutingBitRepetition = 1;      % One instance of a PRS resource set corresponding to a single
prs.MutingPattern2 = [1 0];       % Mute second repetition index of all the PRS resources within
```

Set properties related to the PRS time-domain allocation.

```
prs.NumPRSSymbols = [6 12];
prs.SymbolStart = [6 0];
```

Set properties related to the PRS frequency-domain allocation.

```
prs.NumRB = 40;
prs.RBOffset = 4;
prs.CombSize = 4;
```

```
prs.REOffset = [1 3];
prs.NPRSID = 5;       % Set PRS sequence identity
```

Get the number of orthogonal frequency division multiplexing (OFDM) symbols per slot.

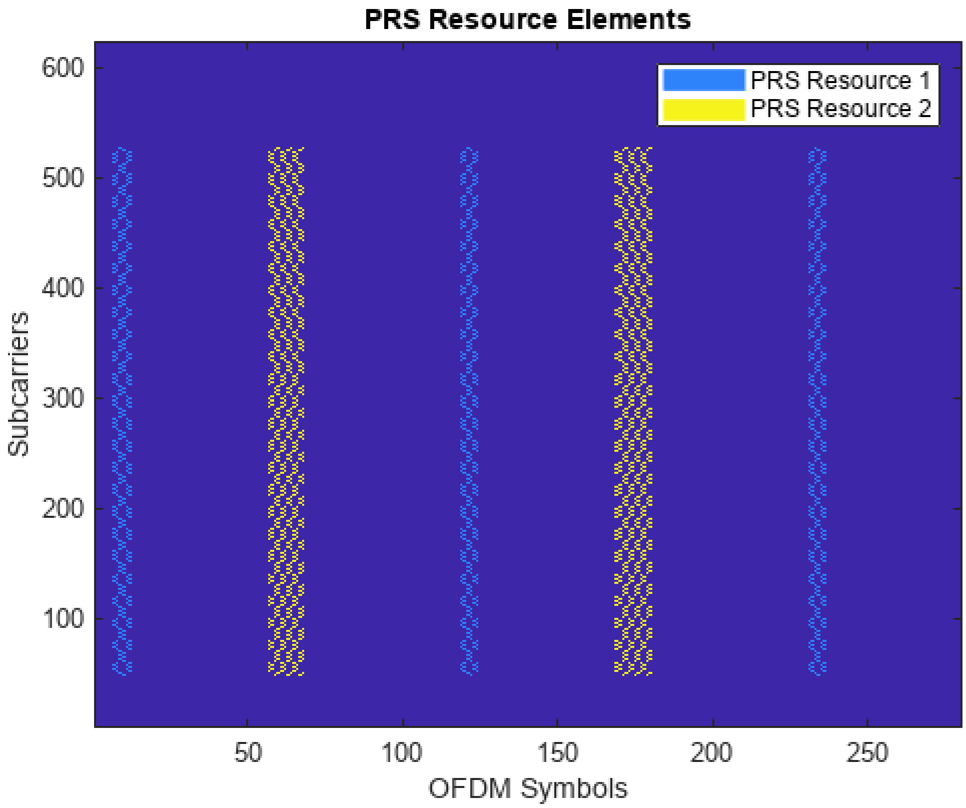
```
numSymPerSlot = carrier.SymbolsPerSlot;
```

Set the number of slots to 20.

```
numSlots = 20;
```

Map the resource elements (RE) for both of the PRS resources on the carrier resource grid.

```
grid = complex(zeros(carrier.NSizeGrid*12,carrier.SymbolsPerSlot*numSlots));
for slotIdx = 0:numSlots-1
    carrier.NSlot = slotIdx;
    indCell = nrPRSIndices(carrier,prs,'OutputResourceFormat','cell');
    symCell = nrPRS(carrier,prs,'OutputResourceFormat','cell');
    slotGrid = nrResourceGrid(carrier);
    slotGrid(indCell{1}) = 70*symCell{1};                            % Resource element mapping
    slotGrid(indCell{2}) = 250*symCell{2};                           % Resource element mapping
    grid(:,(1:numSymPerSlot)+numSymPerSlot*slotIdx) = slotGrid;
end
figure
image(abs(grid));
axis xy;
L = line(ones(2),ones(2),'LineWidth',8);               % Generate lines
set(L,{'color'},{[0.18 0.51 0.98]; [0.96 0.95 0.11]}); % Set the colors
legend('PRS Resource 1','PRS Resource 2');             % Create legend
title('PRS Resource Elements');
xlabel('OFDM Symbols');
ylabel('Subcarriers');
```



Input Arguments

carrier — Carrier configuration parameters
nrCarrierConfig object

Carrier configuration parameters for a specific OFDM numerology, specified as an nrCarrierConfig object. This function uses only these nrCarrierConfig object properties.

SubcarrierSpacing — Subcarrier spacing in kHz
15 (default) | 30 | 60 | 120 | 240

Subcarrier spacing in kHz, for all channels and reference signals of the carrier, specified as 15, 30, 60, 120, or 240.

Data Types: double

CyclicPrefix — Cyclic prefix length
'normal' (default) | 'extended'

Cyclic prefix length, specified as one of these options.

- 'normal' — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.

- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies for only 60 kHz subcarrier spacing.

Data Types: `char` | `string`

`NSizeGrid` — Number of RBs in carrier resource grid
`52` (default) | integer from 1 to 275

Number of RBs in the carrier resource grid, specified as an integer from 1 to 275. The default value of `52` corresponds to the maximum number of RBs of a 10 MHz carrier with 15 kHz SCS.

Data Types: `double`

`NStartGrid` — Start of carrier resource grid relative to CRB 0
`0` (default) | integer from 0 to 2199

Start of carrier resource grid relative to CRB 0, specified as an integer from 0 to 2199. This property is the higher-layer parameter *offsetToCarrier*.

Data Types: `double`

`NSlot` — Slot number
`0` (default) | nonnegative integer

Slot number, specified as a nonnegative integer. You can set `NSlot` to a value larger than the number of slots per frame. For example, you can set this value using transmission loop counters in a MATLAB simulation. In this case, you may have to ensure that the property value is modulo the number of slots per frame in a calling code.

Data Types: `double`

`NFrame` — System frame number
`0` (default) | nonnegative integer

System frame number, specified as a nonnegative integer. You can set `NFrame` to a value larger than the maximum frame number 1023. For example, you can set this value using transmission loop counters in a MATLAB simulation. In this case, you may have to ensure that the property value is modulo 1024 in a calling code.

Data Types: `double`

`prs` — PRS configuration parameters
`nrPRSConfig` object

PRS configuration parameters, specified as an `nrPRSConfig` object. This function uses only these `nrPRSConfig` object properties.

`PRSResourceSetPeriod` — PRS resource set slot periodicity and slot offset
`'on'` (default) | `'off'` | two-element vector

PRS resource set slot periodicity and slot offset, specified as one of these options.

- `'on'` — All of the PRS resources are present in the operating slot.
- `'off'` — All of the PRS resources are absent in the operating slot.
- Two-element vector of the form [*TPRSPeriod*, *TPRSOffset*] — *TPRSPeriod* is the resource set slot periodicity. The nominal value of *TPRSPeriod* must equal 2^{μ} multiplied by one of the values in the

set {4, 5, 8, 10, 16, 20, 32, 40, 64, 80, 160, 320, 640, 1280, 2560, 5120, 10,240}, where μ is the subcarrier spacing configuration with a value of 0, 1, 2, or 3. *TPRSOffset* is the resource set slot offset and must equal a value in the range [0, *TPRSPeriod* – 1].

Data Types: `double` | `string` | `char`

`PRSResourceOffset` — Slot offset of each PRS resource

`0` (default) | scalar in the range [0, 511] | vector of integers in the range [0, 511]

Slot offset of each PRS resource (0-based) provided by the higher layer parameter *dl-PRS-ResourceSlotOffset-r16*, specified as a scalar in the range [0, 511] or a vector of integers in the range [0, 511]. This property represents the starting slot offset of a PRS resource relative to the PRS resource set offset (*TPRSOffset*).

Configure this property for each resource in a resource set separately based on one of these options.

- When a single resource is present, specify this property as a scalar.
- When multiple resources are present, specify this property as a scalar or vector. If you specify a scalar, the object applies that value to all of the PRS resources in a PRS resource set. If you specify a vector, the object applies the vector element values to the corresponding PRS resource. The length of this vector must be equal to the number of PRS resources to be configured in a PRS resource set.

Dependencies

To enable this property, set the `PRSResourceSetPeriod` property to a two-element vector of the form [*TPRSPeriod*, *TPRSOffset*].

Data Types: `double`

`PRSResourceRepetition` — PRS resource repetition factor

`1` (default) | `2` | `4` | `6` | `8` | `16` | `32`

PRS resource repetition factor provided by the higher layer parameter *dl-PRS-ResourceRepetitionFactor-r16*, specified as `1`, `2`, `4`, `6`, `8`, `16`, or `32`. This property value is the same for all of the PRS resources in a PRS resource set.

Dependencies

To enable this property, set the `PRSResourceSetPeriod` property to a two-element vector of the form [*TPRSPeriod*, *TPRSOffset*].

Data Types: `double`

`PRSResourceTimeGap` — Slot offset between two consecutive repeated instances of a PRS resource

`1` (default) | `2` | `4` | `8` | `16` | `32`

Slot offset between two consecutive repeated instances of a PRS resource, specified as `1`, `2`, `4`, `8`, `16`, or `32`. The property represents the offset in terms of the number of slots between two repeated instances of a PRS resource. This property value is same the for all of the PRS resources in a PRS resource set. This property is the higher layer parameter *dl-PRS-ResourceTimeGap-r16*.

Dependencies

To enable this property, set the `PRSResourceSetPeriod` property to a two-element vector of the form [*TPRSPeriod*, *TPRSOffset*] and the `PRSResourceRepetition` property to a value greater than 1.

Data Types: `double`

`MutingPattern1` — Muting bit pattern option-1

`[]` (default) | binary-valued vector of length 2, 4, 6, 8, 16, or 32

Muting bit pattern option-1, specified as `[]` or a binary-valued vector of length 2, 4, 6, 8, 16, or 32.

- If the higher layer parameter *mutingOption1-r16* is configured, set this property to a binary-valued vector of length 2, 4, 6, 8, 16, or 32. Each element in the vector corresponds to a number of consecutive instances of a PRS resource set based on the `MutingBitRepetition` property. The vector element also indicates whether all of the PRS resources within the PRS resource set instances are transmitted (binary 1) or muted (binary 0).
- If the higher layer parameter *mutingOption1-r16* is not configured, set this property to `[]`. To disable the muting bit pattern option-1, set this property to `[]`.

Dependencies

To enable this property, set the `PRSResourceSetPeriod` property to a two-element vector of the form [*TPRSPeriod*, *TPRSOffset*].

Data Types: `double`

`MutingBitRepetition` — Muting bit repetition factor

`1` (default) | `2` | `4` | `8`

Muting bit repetition factor provided by the higher layer parameter *dl-PRS-MutingBitRepetitionFactor-r16*, specified as `1`, `2`, `4`, or `8`. This property indicates the number of consecutive instances of the PRS resource set, *N*, corresponding to each element of the `MutingPattern1` property. The first element in `MutingPattern1` corresponds to the first *N* instances of a PRS resource set, the second element corresponds to the next *N* instances of a PRS resource set, and so on.

Dependencies

To enable this property, set the `PRSResourceSetPeriod` property to a two-element vector of the form [*TPRSPeriod*, *TPRSOffset*] and the `MutingPattern1` property to a value other than `[]`.

Data Types: `double`

`MutingPattern2` — Muting bit pattern option-2

`[]` (default) | binary-valued vector of length 1, 2, 4, 6, 8, 16, or 32

Muting bit pattern option-2, specified as `[]` or a binary-valued vector of length 1, 2, 4, 6, 8, 16, or 32.

- If the higher layer parameter *mutingOption2-r16* is configured, set this property to a binary-valued vector of length 1, 2, 4, 6, 8, 16, or 32. Each element in the vector relates to the corresponding repetition index. Each element in the vector corresponds to a single repetition index of each PRS resource in an active instance of a PRS resource set. The vector element also indicates whether the repetition index for all of the PRS resources is transmitted (binary 1) or muted (binary 0).

- If the higher layer parameter *mutingOption2-r16* is not configured, set this property to `[]`. To disable the muting bit pattern option-2, set this property to `[]`.

Dependencies

To enable this property, set the `PRSResourceSetPeriod` property to a two-element vector of the form [*TPRSPeriod*, *TPRSOffset*].

Data Types: `double`

`NumPRSSymbols` — Number of consecutive OFDM symbols allocated for each PRS resource
`12` (default) | scalar in the range [0, 12] | vector of integers in the range [0, 12]

Number of consecutive OFDM symbols allocated for each PRS resource, specified as a scalar in the range [0, 12] or a vector of integers in the range [0, 12].

Configure this property for each resource in a resource set separately based on one of these options.

- When a single resource is present, specify this property as a scalar.
- When multiple resources are present, specify this property as a scalar or vector. If you specify a scalar, the object applies that value to all of the PRS resources in a PRS resource set. If you specify a vector, the object applies the vector element values to the corresponding PRS resource. The length of this vector must be equal to the number of PRS resources to be configured in a PRS resource set.

This property is the higher layer parameter *dl-PRS-NumSymbols-r16*, and the nominal value of this property must be `2`, `4`, `6`, or `12`. To indicate no PRS resource allocation, set this property to `0`.

Data Types: `double`

`SymbolStart` — Starting OFDM symbol of each PRS resource in slot
`0` (default) | scalar in the range [0, 13] | vector of integers in the range [0, 13]

Starting OFDM symbol of each PRS resource in a slot (0-based), specified as a scalar in the range [0, 13] or a vector of integers in the range [0, 13].

Configure this property for each resource in a resource set separately based on one of these options.

- When a single resource is present, specify this property as a scalar.
- When multiple resources are present, specify this property as a scalar or vector. If you specify a scalar, the object applies that value to all of the PRS resources in a PRS resource set. If you specify a vector, the object applies the vector element values to the corresponding PRS resource. The length of this vector must be equal to the number of PRS resources to be configured in a PRS resource set.

This property is the higher layer parameter *dl-PRS-ResourceSymbolOffset-r16*, and the nominal value of this property must be in the range [0, 12].

Data Types: `double`

`NumRB` — Number of PRBs allocated for all PRS resources
`52` (default) | scalar in the range [0, 275]

Number of PRBs allocated for all PRS resources in a resource set, specified as a scalar in the range [0, 275]. This property is the higher layer parameter *dl-PRS-ResourceBandwidth-r16*, and the nominal value of this property must be in the range [24, 272] with a granularity of 4 PRBs. To indicate no PRS resource allocation, set this property to `0`.

Data Types: `double`

`RBOffset` — Starting PRB index of all PRS resources relative to carrier resource grid
`0` (default) | scalar in the range [0, 274]

Starting PRB index of all PRS resources relative to the carrier resource grid, specified as a scalar in the range [0, 274].

Data Types: `double`

`CombSize` — Comb size of all PRS resources
`2` (default) | `4` | `6` | `12`

Comb size of all PRS resources in a resource set, specified as `2`, `4`, `6`, or `12`. The comb size represents the resource element spacing in each OFDM symbol. This property is the higher layer parameter *dl-PRS-CombSizeN-r16*. The value `2` specifies for the object to allocate every 2nd RE in the PRB for PRS, the value `4` specifies for the object to allocate every 4th RE in the PRB for PRS, and so on.

Data Types: `double`

`REOffset` — Starting RE offset in first PRS OFDM symbol of each PRS resource
`0` (default) | scalar in the range [0, (`CombSize`–1)] | vector of integers in the range [0, (`CombSize`–1)]

Starting RE offset in the first PRS OFDM symbol of each PRS resource, specified as a scalar in the range [0, (`CombSize`–1)] or a vector of integers in the range [0, (`CombSize`–1)]. The relative RE offsets of the next PRS OFDM symbols are defined relative to the `REOffset` value, as described in TS 38.211 Table 7.4.1.7.3-1.

Configure this property for each resource in a resource set separately based on one of these options.

- When a single resource is present, specify this property as a scalar.
- When multiple resources are present, specify this property as a scalar or vector. If you specify a scalar, the object applies that value to all of the PRS resources in a PRS resource set. If you specify a vector, the object applies the vector element values to the corresponding PRS resource. The length of this vector must be equal to the number of PRS resources to be configured in a PRS resource set.

This property is the higher layer parameter *dl-PRS-ReOffset-r16*.

Data Types: `double`

Note The maximum vector length of these five properties of `nrPRSConfig` object specifies the number of configured PRS resources.

- `PRSResourceOffset`
- `NumPRSSymbols`
- `SymbolStart`
- `REOffset`
- `NPRSID`

Name-Value Pair Arguments

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

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

Example: `'IndexStyle','subscript'` specifies the RE indexing style of the output indices as `'subscript'`.

`IndexStyle` — RE indexing form
`'index'` (default) | `'subscript'`

RE indexing form, specified as one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

`IndexBase` — RE indexing base
`'1based'` (default) | `'0based'`

RE indexing base, specified as one of these values:

- `'1based'` — The index counting starts from 1.
- `'0based'` — The index counting starts from 0.

Data Types: `char` | `string`

`OutputResourceFormat` — Output format of PRS resource element indices
`'concatenated'` (default) | `'cell'`

Output format of PRS resource element indices, specified as one of these values:

- `'concatenated'` — The output `ind` is a single column vector containing all PRS resource element indices concatenated.
- `'cell'` — The output `ind` is a cell array where each cell corresponds to a single PRS resource.

Data Types: `char` | `string`

Output Arguments

`ind` — PRS RE indices
column vector | *M*-by-3 matrix | cell array of column vectors | cell array of matrices

PRS RE indices, returned as one of these values.

- Column vector — The function returns the indices output in this format when you set the `IndexStyle` name-value argument to `'index'`.
- *M*-by-3 matrix — The function returns the indices output in this format when you set the `IndexStyle` name-value argument to `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and number of antennas, respectively.

- Cell array of column vectors — The function returns the indices output in this format when you set the `IndexStyle` name-value argument to `'index'` and the `OutputResourceFormat` name-value argument to `'cell'`.
- Cell array of matrices — The function returns the indices output in this format when `IndexStyle` is set to `'subscript'` and the `OutputResourceFormat` name-value argument to `'cell'`. The number of rows in each cell varies based on the PRS resource configurations in a PRS resource set.

Depending on the value of `IndexBase`, the function returns either 1-based or 0-based indices.

Data Types: `uint32`

Version History

Introduced in R2021a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation (Release 16)." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Objects
`nrCarrierConfig` | `nrPRSConfig`

Functions
`nrPRS`

Topics
"NR Positioning Reference Signal"

nrPSS

Generate PSS symbols

Syntax

```
sym = nrPSS(ncellid)
sym = nrPSS(ncellid,'OutputDataType',datatype)
```

Description

`sym = nrPSS(ncellid)` returns the primary synchronization signal (PSS) symbols for the physical layer cell identity number `ncellid`. The function implements TS 38.211 Section 7.4.2.2 [1].

`sym = nrPSS(ncellid,'OutputDataType',datatype)` specifies the data type of the PSS symbol.

Examples

Generate PSS Symbols

Generate the sequence of 127 PSS binary phase shift keying (BPSK) modulation symbols for a given cell identity. The PSS is transmitted in the first symbol of a Synchronization Signal / Physical Broadcast Channel (SS/PBCH) block.

```
ncellid = 17;
pss = nrPSS(ncellid)
```

```
pss = 127×1

    -1
    -1
    -1
    -1
    -1
    -1
     1
     1
     1
    -1
      ⋮
```

Input Arguments

`ncellid` — Physical layer cell identity number
integer

Physical layer cell identity number, specified as an integer from 0 to 1007.

Data Types: double

datatype — Data type of output symbols
'double' (default) | 'single'

Data type of the output symbols, specified as 'double' or 'single'.

Data Types: char | string

Output Arguments

sym — PSS symbols
column vector of real numbers

PSS symbols, returned as a column vector of real numbers.

Data Types: single | double

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The datatype input argument must be compile-time constant. For example, when specifying 'single' as the output data type, include {coder.Constant('OutputDataType'),coder.Constant('single')} in the -args value of the codegen function. For more information, see the coder.Constant class.

See Also

Functions
nrPBCH | nrPSSIndices | nrSSS | nrPBCHDMRS | nrSSBMeasurements

nrPSSIndices

Generate PSS resource element indices

Syntax

```
ind = nrPSSIndices
ind = nrPSSIndices(Name,Value)
```

Description

`ind = nrPSSIndices` returns the resource element indices for the primary synchronization signal (PSS), as defined in TS 38.211 Section 7.4.3.1 [1]. The returned indices are 1-based using linear indexing form. This indexing form can directly index the elements of a 240-by-4 matrix corresponding to the Synchronization Signal / Physical Broadcast Channel (SS/PBCH) block. The order of the indices indicates how the PSS modulation symbols are mapped.

`ind = nrPSSIndices(Name,Value)` specifies index formatting options by using one or more name-value pair arguments. Unspecified options take default values.

Examples

Get PSS Resource Element Indices

Generate the 127 resource element indices associated with the PSS within a single SS/PBCH block.

```
ind = nrPSSIndices
```

```
ind = 127x1 uint32 column vector

    57
    58
    59
    60
    61
    62
    63
    64
    65
    66
      ⋮
```

Input Arguments

Name-Value Pair Arguments

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

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

Example: `'IndexStyle','subscript','IndexBase','0based'` specifies nondefault resource element index formatting options.

`IndexStyle` — RE indexing form
`'index'` (default) | `'subscript'`

RE indexing form, specified as one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

`IndexBase` — RE indexing base
`'1based'` (default) | `'0based'`

RE indexing base, specified as one of these values:

- `'1based'` — The index counting starts from 1.
- `'0based'` — The index counting starts from 0.

Data Types: `char` | `string`

Output Arguments

`ind` — PSS resource element indices
column vector | *M*-by-3 matrix

PSS resource element indices, returned as one of these values:

- Column vector — When `'IndexStyle'` is `'index'`.
- *M*-by-3 matrix — When `'IndexStyle'` is `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers and OFDM symbols in an SS/PBCH block, and the number of antennas, respectively.

Depending on `'IndexBase'`, the indices are either 1-based or 0-based.

Data Types: `uint32`

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPSS` | `nrSSSIndices` | `nrPBCHDMRSIndices` | `nrPBCHIndices` | `nrSSBMeasurements`

nrPUCCH

Generate PUCCH modulation symbols

Syntax

```
sym = nrPUCCH(carrier,pucch,uciBits)
sym = nrPUCCH(carrier,pucch,uciBits,'OutputDataType',datatype)
```

Description

`sym = nrPUCCH(carrier,pucch,uciBits)` generates physical uplink control channel (PUCCH) modulation symbols `sym`, as defined in TS 38.211 Sections 6.3.2.3 to 6.3.2.6 [1] for all PUCCH formats. `carrier` specifies the carrier configuration. `pucch` specifies the format-specific PUCCH configuration. `uciBits` specifies the uplink control information (UCI) bits.

`sym = nrPUCCH(carrier,pucch,uciBits,'OutputDataType',datatype)` specifies the data type of the PUCCH symbols.

Examples

Generate PUCCH Format 0 Symbols for Positive SR Transmission Without HARQ-ACK

Create a default carrier configuration object.

```
carrier = nrCarrierConfig;
```

Specify the physical layer cell identity as 490, cyclic prefix as normal, and slot number as 57.

```
carrier.NCellID = 490;
carrier.CyclicPrefix = 'normal';
carrier.NSlot = 57;
```

Create a default PUCCH format 0 configuration object.

```
pucch0 = nrPUCCH0Config;
```

Specify the first symbol index in the PUCCH transmission slot as 11 and the number of allocated PUCCH symbols as 2. Enable intraslot frequency hopping and group hopping. Set the initial cyclic shift to 7.

```
pucch0.SymbolAllocation = [11 2];
pucch0.FrequencyHopping = 'intraSlot';
pucch0.GroupHopping = 'enable';
pucch0.HoppingID = [];      % Set HoppingID equal to the NCellID property of carrier
pucch0.InitialCyclicShift = 7;
```

Specify a transmission without HARQ-ACK and with a positive SR.

```
sr = 1;                    % Positive SR transmission
ack = [];
uciBits = {ack, sr};
```

Generate PUCCH format 0 modulation symbols.

```
sym = nrPUCCH(carrier,pucch0,uciBits)
```

```
sym = 24×1 complex

  -0.7071 - 0.7071i
  -0.9659 + 0.2588i
  -0.9659 - 0.2588i
   0.7071 - 0.7071i
  -0.2588 - 0.9659i
  -0.9659 - 0.2588i
  -0.7071 - 0.7071i
   0.2588 + 0.9659i
  -0.2588 + 0.9659i
  -0.7071 - 0.7071i
      ⋮
```

Generate PUCCH Format 1 Symbols for Two-Bit HARQ-ACK Transmission

Create a default carrier configuration object.

```
carrier = nrCarrierConfig;
```

Specify a 60 kHz carrier with extended cyclic prefix. Set the number of (resource blocks) RBs in the carrier resource grid to 80 and the slot number to 3.

```
carrier.SubcarrierSpacing = 60;
carrier.CyclicPrefix = 'extended';
carrier.NSizeGrid = 80;
carrier.NSlot = 3;
```

Create a default PUCCH format 1 configuration object.

```
pucch1 = nrPUCCH1Config;
```

Specify the first symbol index in the PUCCH transmission slot as 2 and the number of allocated PUCCH symbols as 10. Enable intraslot frequency hopping and group hopping. Set the hopping identity to 512, the initial cyclic shift to 7, and the orthogonal cover code index (OCCI) to 1.

```
pucch1.SymbolAllocation = [2 10];
pucch1.FrequencyHopping = 'intraSlot';
pucch1.GroupHopping = 'enable';
pucch1.HoppingID = 512;
pucch1.InitialCyclicShift = 7;
pucch1.OCCI = 1;
```

Specify a transmission with two-bit HARQ-ACK.

```
uciBits = [1;0];
```

Generate PUCCH format 1 modulation symbols.

```
sym = nrPUCCH(carrier,pucch1,uciBits)
```

```
sym = 60×1 complex

   1.0000 + 0.0000i
   0.5000 + 0.8660i
  -0.8660 - 0.5000i
   0.0000 + 1.0000i
  -0.8660 + 0.5000i
   0.8660 + 0.5000i
   1.0000 - 0.0000i
   0.8660 - 0.5000i
  -0.8660 - 0.5000i
   1.0000 - 0.0000i
      ⋮
```

Generate PUCCH Format 2 Symbols

Create a default carrier configuration object.

```
carrier = nrCarrierConfig;
```

Create a default PUCCH format 2 configuration object.

```
pucch2 = nrPUCCH2Config;
```

Specify the data scrambling identity as 1000 and the radio network temporary identifier as 160.

```
pucch2.NID = 1000;
pucch2.RNTI = 160;
```

Create a random sequence of binary values corresponding to a UCI codeword of 100 bits.

```
uciCW = randi([0 1],100,1);
```

Generate PUCCH format 2 modulation symbols of data type `single`.

```
sym = nrPUCCH(carrier,pucch2,uciCW,'OutputDataType','single')
```

```
sym = 50x1 single column vector

   0.7071 + 0.7071i
  -0.7071 + 0.7071i
  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
   0.7071 - 0.7071i
   0.7071 + 0.7071i
   0.7071 - 0.7071i
   0.7071 - 0.7071i
      ⋮
```

Generate PUCCH Format 3 Pi/2-BPSK Symbols

Create a default carrier configuration object, and then set the with cell identity as 135.

```
carrier = nrCarrierConfig;
carrier.NCellID = 135;
```

Create a default PUCCH format 3 configuration object.

```
pucch3 = nrPUCCH3Config;
```

Specify the modulation scheme as pi/2-BPSK, the PRB allocation of the PUCCH to range from 70 to 74 (occupying 5 resource blocks), and the radio network temporary identifier as 2560.

```
pucch3.Modulation = 'pi/2-BPSK';
pucch3.PRBSet = 70:74;
pucch3.RNTI = 2560;
```

Create a random sequence of binary values corresponding to a UCI codeword of 120 bits.

```
uciCW = randi([0 1],120,1);
```

Generate PUCCH format 3 modulation symbols of datatype `single`.

```
sym = nrPUCCH(carrier,pucch3,uciCW,'OutputDataType','single')
```

```
sym = 120x1 single column vector

   0.1826 - 0.5477i
  -1.0861 - 0.3479i
   0.6300 + 0.8742i
  -0.5093 + 0.6652i
   0.1911 - 0.7721i
   0.1157 + 0.4320i
   0.3199 - 1.0054i
   1.4349 - 1.7045i
   0.4180 - 0.4832i
   0.0044 - 0.3429i
      ⋮
```

Generate PUCCH Format 4 QPSK Symbols

Create a default carrier configuration object, and then set the cell identity as 140.

```
carrier = nrCarrierConfig;
carrier.NCellID = 140;
```

Create a default PUCCH format 4 configuration object.

```
pucch4 = nrPUCCH4Config;
```

Specify the modulation scheme as QPSK, the spreading factor as 4, the OCCI as 3, and the radio network temporary identifier as 750.

```
pucch4.Modulation = 'QPSK';
pucch4.SpreadingFactor = 4;
pucch4.OCCI = 3;
pucch4.RNTI = 750;
```

Create a random sequence of binary values corresponding to a UCI codeword of 120 bits.

```
uciCW = randi([0 1],120,1);
```

Generate PUCCH format 4 modulation symbols.

```
sym = nrPUCCH(carrier,pucch4,uciCW)
```

```
sym = 240×1 complex

   0.0000 + 0.0000i
   0.5977 + 0.5977i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
  -2.2307 - 2.2307i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
  -0.8165 - 0.8165i
      ⋮
```

Input Arguments

carrier — Carrier configuration parameters
nrCarrierConfig object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function uses only these `nrCarrierConfig` object properties.

- `NCellID`
- `SubcarrierSpacing`
- `CyclicPrefix`
- `NSizeGrid`
- `NStartGrid`
- `NSlot`

pucch — PUCCH configuration parameters
nrPUCCH0Config object | nrPUCCH1Config object | nrPUCCH2Config object | nrPUCCH3Config object | nrPUCCH4Config object

PUCCH configuration parameters, specified as one of these options.

- For format 0, specify an `nrPUCCH0Config` object. The function uses only these object properties.
  - `SymbolAllocation`
  - `FrequencyHopping`

  - `GroupHopping`
  - `HoppingID`
  - `InitialCyclicShift`
- For format 1, specify an `nrPUCCH1Config` object. The function uses only these object properties.
  - `SymbolAllocation`
  - `FrequencyHopping`
  - `GroupHopping`
  - `HoppingID`
  - `InitialCyclicShift`
  - `OCCI`
- For format 2, specify an `nrPUCCH2Config` object. The function uses only these object properties.
  - `NID`
  - `RNTI`
- For format 3, specify an `nrPUCCH3Config` object. The function uses only these object properties.
  - `PRBSet`
  - `Modulation`
  - `NID`
  - `RNTI`
- For format 4, specify an `nrPUCCH4Config` object. The function uses only these object properties.
  - `Modulation`
  - `SpreadingFactor`
  - `OCCI`
  - `NID`
  - `RNTI`

uciBits — UCI bits
binary-valued column vector | one-element cell array | two-element cell array

UCI bits, specified as a binary-valued column vector, a one-element cell array, or a two-element cell array depending on the format type. When you specify this argument as a cell array, each element in the array must be a column vector.

- For format 0, you must specify a binary-valued column vector, a one-element cell array, or a two-element cell array.
  - When you specify this value as a binary-valued column vector or a one-element cell array, the UCI bits are assumed to be hybrid automatic repeat request acknowledgment (HARQ-ACK) bits.
  - When you specify this value as a two-element cell array, the first element is assumed to have HARQ-ACK bits, and the second element is assumed to have scheduling request (SR) bit.
- For format 1, you must specify a binary-valued column vector or a one-element cell array. This argument must contain HARQ-ACK bits or SR bits. To transmit only positive SR bit, specify this argument as `[0]` or `{0}`.

- For formats 2, 3, and 4, you must specify a binary-valued column vector or a one-element cell array. This argument must contain a codeword of encoded UCI bits.

Data Types: `double`

`datatype` — Data type of output symbols
`'double'` | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

`sym` — PUCCH modulation symbols
complex column vector

PUCCH modulation symbols for the specified format, returned as a complex column vector.

Data Types: `single` | `double`
Complex Number Support: Yes

Version History

Introduced in R2021a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The `datatype` input argument must be compile-time constant. For example, when specifying `'single'` as the output data type, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPUCCHIndices` | `nrPUCCHDMRSIndices` | `nrPUCCHDMRS`

Objects
`nrPUCCH0Config` | `nrPUCCH1Config` | `nrPUCCH2Config` | `nrPUCCH3Config` | `nrPUCCH4Config`

nrPUCCH0

Generate PUCCH format 0 modulation symbols

Syntax

```
sym = nrPUCCH0(ack,sr,symAllocation,cp,nslot,nid,groupHopping,initialCS,
freqHopping)
sym = nrPUCCH0( ___ ,'OutputDataType',datatype)
```

Description

`sym = nrPUCCH0(ack,sr,symAllocation,cp,nslot,nid,groupHopping,initialCS,freqHopping)` returns the physical uplink control channel (PUCCH) format 0 modulation symbols, as defined in TS 38.211 Section 6.3.2.3 [1], based on these input arguments:

- Hybrid automatic repeat-request acknowledgment (HARQ-ACK) `ack`
- Scheduling request (SR) `sr`
- PUCCH symbol allocation `symAllocation`
- Cyclic prefix `cp`
- Radio frame slot number `nslot`
- Scrambling identity `nid`
- Group hopping configuration `groupHopping`
- Initial cyclic shift `initialCS`
- Intra-slot frequency hopping configuration `freqHopping`

`sym = nrPUCCH0( ___ ,'OutputDataType',datatype)` specifies the PUCCH symbol data type in addition to the input arguments in the previous syntax.

Examples

Generate PUCCH Format 0 Modulation Symbols for Positive SR Transmission

Specify a transmission without HARQ-ACK and a positive SR.

```
ack = [];
sr = 1;
```

Specify the first symbol index in the PUCCH transmission slot as 11, the number of allocated PUCCH symbols as 2, and the slot number as 63.

```
symAllocation = [11 2];
nslot = 63;
```

Set the scrambling identity to 512 and the initial cyclic shift to 5.

```
nid = 512;
initialCS = 5;
```

Generate the symbols with normal cyclic prefix, intra-slot frequency hopping disabled, and group hopping enabled.

```
cp = 'normal';
freqHopping = 'disabled';
groupHopping = 'enable';
sym = nrPUCCH0(ack,sr,symAllocation,cp,nslot,nid,groupHopping,initialCS,freqHopping)
```

```
sym = 24×1 complex

   0.7071 + 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
      ⋮
```

Generate PUCCH Format 0 Modulation Symbols for HARQ-ACK Transmission

Specify two-bit HARQ-ACK transmission and a negative SR.

```
ack = [1;1];
sr = 0;
```

Specify the first symbol index in the PUCCH transmission slot as 10, the number of allocated PUCCH symbols as 2, and the slot number as 3.

```
symAllocation = [10 2];
nslot = 3;
```

Set the scrambling identity to 12 and the initial cyclic shift to 5.

```
nid = 12;
initialCS = 5;
```

Generate the symbols with extended cyclic prefix, intra-slot frequency hopping disabled, and group hopping enabled.

```
nid = 12;
initialCS = 5;
cp = 'extended';
freqHopping = 'disabled';
groupHopping = 'enable';
sym = nrPUCCH0(ack,sr,symAllocation,cp,nslot,nid,groupHopping,initialCS,freqHopping)
```

```
sym = 24×1 complex

  -0.7071 - 0.7071i
  -0.9659 - 0.2588i
  -0.9659 + 0.2588i
```

```
-0.7071 - 0.7071i
 0.2588 - 0.9659i
-0.2588 - 0.9659i
-0.7071 + 0.7071i
 0.9659 + 0.2588i
 0.2588 + 0.9659i
-0.7071 - 0.7071i
     ⋮
```

Input Arguments

ack — HARQ-ACK bits
empty vector | binary column vector

HARQ-ACK bits, specified as an empty vector or a binary column vector with one or two rows. An empty vector indicates PUCCH transmission without HARQ-ACK. If specifying a binary column vector, the number of rows corresponds to the number of codewords. Vector element `1` denotes positive acknowledgement (ACK), and vector element `0` denotes negative acknowledgment (NACK).

Data Types: `double`

sr — SR bits
empty vector | `1` | `0`

SR bits, specified as an empty vector, `1`, or `0`. An empty vector indicates PUCCH transmission without SR. `1` denotes positive SR. `0` denotes negative SR. For negative SR without HARQ-ACK, the output `sym` is empty.

Data Types: `double`

symAllocation — PUCCH symbol allocation
two-element numeric vector

PUCCH symbol allocation, specified as a two-element numeric vector of the form [*S* *L*], where *S* and *L* are nonnegative integers.

- *S* is the first OFDM symbol index in the PUCCH transmission slot.
- *L* is the number of OFDM symbols allocated for PUCCH transmission. For PUCCH format 0, *L* is either `1` or `2`.

Note *S* and *L* must satisfy these conditions.

- For extended control prefix, $S + L \leq 12$.
- For normal cyclic prefix, $S + L \leq 14$.

Data Types: `double`

cp — Cyclic prefix length
`'normal'` | `'extended'`

Cyclic prefix length, specified as one of these options:

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length only applies for 60 kHz subcarrier spacing.

Data Types: `char` | `string`

`nslot` — Radio frame slot number
integer from 0 to 159

Radio frame slot number, specified as an integer from 0 to 159. For normal cyclic prefix of different numerologies, specify an integer from 0 to 159. For extended cyclic prefix, specify an integer from 0 to 39. For more details, see TS 38.211 Section 4.3.2.

Data Types: `double`

`nid` — Scrambling identity
integer from 0 to 1023

Scrambling identity, specified as an integer from 0 to 1023. `nid` is higher layer parameter *hoppingId*, ranging from 0 to 1023, if the higher layer parameter is configured. Otherwise, `nid` is the physical layer cell identity number *NCellID*, ranging from 0 to 1007. For more information on these values, see TS 38.211 Section 6.3.2.2.1.

Data Types: `double`

`groupHopping` — Group hopping configuration
`'neither'` | `'enable'` | `'disable'`

Group hopping configuration, specified as `'neither'`, `'enable'`, or `'disable'`. The `groupHopping` argument is higher layer parameter *pucch-GroupHopping*.

Note When `groupHopping` is set to `'disable'`, the function enables sequence hopping. In this case, the selected sequence number might not be appropriate for short base sequences.

Data Types: `char` | `string`

`initialCS` — Initial cyclic shift
integer from 0 to 11

Initial cyclic shift, *m_0*, specified as an integer from 0 to 11. `initialCS` is higher layer parameter *initialCyclicShift*.

For more information, see TS 38.213 Section 9.2.1 [2].

Data Types: `double`

`freqHopping` — Intra-slot frequency hopping configuration
`'enabled'` | `'disabled'`

Intra-slot frequency hopping configuration, specified as `'enabled'` or `'disabled'`. The `freqHopping` argument is higher layer parameter *intraSlotFrequencyHopping*.

Data Types: `char` | `string`

datatype — Data type of output symbols
'double' (default) | 'single'

Data type of the output symbols, specified as 'double' or 'single'.

Data Types: char | string

Output Arguments

sym — PUCCH format 0 modulation symbols
empty vector | complex column vector

PUCCH format 0 modulation symbols, returned as an empty vector or a complex column vector. sym is of length 12×*L*, where *L* is the PUCCH symbol allocation length, specified by symAllocation. For negative SR without HARQ-ACK, sym is always empty.

Data Types: single | double
Complex Number Support: Yes

Version History

Introduced in R2019a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.213. "NR; Physical layer procedures for control." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The datatype input argument must be compile-time constant. For example, when specifying 'single' as the output data type, include {coder.Constant('OutputDataType'),coder.Constant('single')} in the -args value of the codegen function. For more information, see the coder.Constant class.

See Also

Functions
nrPUCCH1 | nrPUCCH2 | nrPUCCH3 | nrPUCCH4 | nrPUCCHHoppingInfo | nrLowPAPRS

nrPUCCH1

Generate PUCCH format 1 modulation symbols

Syntax

```
sym = nrPUCCH1(ack,sr,symAllocation,cp,nslot,nid,groupHopping,initialCS,
freqHopping,occi)
sym = nrPUCCH1( ___ ,'OutputDataType',datatype)
```

Description

`sym = nrPUCCH1(ack,sr,symAllocation,cp,nslot,nid,groupHopping,initialCS, freqHopping,occi)` returns the physical uplink control channel (PUCCH) format 1 modulation symbols, as defined in TS 38.211 Section 6.3.2.4 [1], based on these input arguments:

- Hybrid automatic repeat-request acknowledgment (HARQ-ACK) `ack`
- Scheduling request (SR) `sr`
- PUCCH symbol allocation `symAllocation`
- Cyclic prefix `cp`
- Radio frame slot number `nslot`
- Scrambling identity `nid`
- Group hopping configuration `groupHopping`
- Initial cyclic shift `initialCS`
- Intra-slot frequency hopping configuration `freqHopping`
- Orthogonal cover code index `occi`

`sym = nrPUCCH1( ___ ,'OutputDataType',datatype)` specifies the PUCCH symbol data type in addition to the input arguments in the previous syntax.

Examples

Generate PUCCH Format 1 Modulation Symbols for Two-Bit HARQ-ACK with Positive SR Transmission

Specify a transmission with two-bit HARQ-ACK and positive SR.

```
ack = [0;1];
sr = 1;
```

Specify the first symbol index in the PUCCH transmission slot as 0, the number of allocated PUCCH symbols as 14, and the slot number as 3.

```
symAllocation = [0 14];
nslot = 3;
```

Set the scrambling identity to 512 and the initial cyclic shift to 5.

```
nid = 512;
initialCS = 5;
```

Generate the symbols with normal cyclic prefix, intra-slot frequency hopping and group hopping enabled, and orthogonal cover code index 2.

```
cp = 'normal';
freqHopping = 'enabled';
groupHopping = 'enable';
occi = 2;
sym = nrPUCCH1(ack,sr,symAllocation,cp,nslot, ...
    nid,groupHopping,initialCS,freqHopping,occi)
```

```
sym = 84×1 complex

  -1.0000 - 0.0000i
  -0.5000 + 0.8660i
  -0.8660 + 0.5000i
  -0.0000 - 1.0000i
  -0.8660 - 0.5000i
   0.8660 - 0.5000i
  -1.0000 + 0.0000i
   0.8660 + 0.5000i
  -0.8660 + 0.5000i
  -1.0000 + 0.0000i
      ⋮
```

Generate PUCCH Format 1 Modulation Symbols for One-Bit HARQ-ACK Transmission

Specify a transmission with one-bit HARQ-ACK and negative SR.

```
ack = 1;
sr = 0;
```

Specify the first symbol index in the PUCCH transmission slot as 3, the number of allocated PUCCH symbols as 9, and the slot number as 7.

```
symAllocation = [3 9];
nslot = 7;
```

Set the scrambling identity to 512 and the initial cyclic shift to 9.

```
nid = 512;
initialCS = 9;
```

Generate the symbols with extended cyclic prefix, intra-slot frequency hopping and group hopping enabled, and orthogonal cover code index 1.

```
cp = 'extended';
freqHopping = 'enabled';
groupHopping = 'enable';
occi = 1;
sym = nrPUCCH1(ack,sr,symAllocation,cp,nslot, ...
    nid,groupHopping,initialCS,freqHopping,occi)
```

```
sym = 48×1 complex

  -0.0000 + 1.0000i
  -0.8660 + 0.5000i
  -0.5000 + 0.8660i
   1.0000 - 0.0000i
   0.8660 - 0.5000i
   0.8660 + 0.5000i
  -0.0000 - 1.0000i
  -0.8660 + 0.5000i
   0.8660 + 0.5000i
   0.0000 + 1.0000i
      ⋮
```

Input Arguments

ack — HARQ-ACK bits

empty vector | binary column vector

HARQ-ACK bits, specified as an empty vector or a binary column vector with one or two rows. An empty vector indicates PUCCH transmission without HARQ-ACK. If specifying a binary column vector, the number of rows corresponds to the number of codewords. Vector element `1` denotes positive acknowledgement (ACK), and vector element `0` denotes negative acknowledgment (NACK).

Data Types: `double`

sr — SR bits

empty vector | `1` | `0`

SR bits, specified as an empty vector, `1`, or `0`. An empty vector indicates PUCCH transmission without SR. `1` denotes positive SR. `0` denotes negative SR. For either positive or negative SR with HARQ-ACK information bits, only HARQ-ACK transmission occurs. For negative SR without HARQ-ACK, the output `sym` is empty.

Data Types: `double`

symAllocation — PUCCH symbol allocation

two-element numeric vector

PUCCH symbol allocation, specified as a two-element numeric vector of the form [S L], where S and L are nonnegative integers.

- S is the first OFDM symbol index in the PUCCH transmission slot.
- L is the number of OFDM symbols allocated for PUCCH transmission. For PUCCH format 1, L is an integer from 4 or 14.

Note S and L must satisfy these conditions.

- For extended control prefix, $S + L \leq 12$.
- For normal cyclic prefix, $S + L \leq 14$.

Data Types: `double`

cp — Cyclic prefix length
`'normal'` | `'extended'`

Cyclic prefix length, specified as one of these options:

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length only applies for 60 kHz subcarrier spacing.

Data Types: `char` | `string`

nslot — Radio frame slot number
integer from 0 to 159

Radio frame slot number, specified as an integer from 0 to 159. For normal cyclic prefix of different numerologies, specify an integer from 0 to 159. For extended cyclic prefix, specify an integer from 0 to 39. For more details, see TS 38.211 Section 4.3.2.

Data Types: `double`

nid — Scrambling identity
integer from 0 to 1023

Scrambling identity, specified as an integer from 0 to 1023. `nid` is higher layer parameter *hoppingId*, ranging from 0 to 1023, if the higher layer parameter is configured. Otherwise, `nid` is the physical layer cell identity number *NCellID*, ranging from 0 to 1007. For more information on these values, see TS 38.211 Section 6.3.2.2.1.

Data Types: `double`

groupHopping — Group hopping configuration
`'neither'` | `'enable'` | `'disable'`

Group hopping configuration, specified as `'neither'`, `'enable'`, or `'disable'`. The `groupHopping` argument is higher layer parameter *pucch-GroupHopping*.

Note When `groupHopping` is set to `'disable'`, the function enables sequence hopping. In this case, the selected sequence number might not be appropriate for short base sequences.

Data Types: `char` | `string`

initialCS — Initial cyclic shift
integer from 0 to 11

Initial cyclic shift, *m_0*, specified as an integer from 0 to 11. `initialCS` is higher layer parameter *initialCyclicShift*.

For more information, see TS 38.213 Section 9.2.1 [2].

Data Types: `double`

`freqHopping` — Intra-slot frequency hopping configuration
`'enabled'` | `'disabled'`

Intra-slot frequency hopping configuration, specified as `'enabled'` or `'disabled'`. The `freqHopping` argument is higher layer parameter *intraSlotFrequencyHopping*.

Data Types: `char` | `string`

`occi` — Orthogonal cover code index
integer from 0 to 6

Orthogonal cover code index, specified as an integer from 0 to 6. This input argument corresponds to higher layer parameter *timeDomainOCC*. The valid range depends on the number of OFDM symbols that contain control information in a hop.

Data Types: `double`

`datatype` — Data type of output symbols
`'double'` (default) | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

`sym` — PUCCH format 1 modulation symbols
complex column vector | empty vector

PUCCH format 1 modulation symbols, returned as a complex column vector or an empty vector. `sym` is of length `12×floor(L/2)`, where *L* is the PUCCH symbol allocation length, specified by `symAllocation`. For negative SR without HARQ-ACK, the output `sym` is empty.

Data Types: `single` | `double`
Complex Number Support: Yes

Version History
Introduced in R2019a

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.213. “NR; Physical layer procedures for control.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The `datatype` input argument must be compile-time constant. For example, when specifying `'single'` as the output data type, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

`nrPUCCH0` | `nrPUCCH2` | `nrPUCCH3` | `nrPUCCH4` | `nrPUCCHHoppingInfo` | `nrLowPAPRS`

nrPUCCH2

Generate PUCCH format 2 modulation symbols

Syntax

```
sym = nrPUCCH2(uciCW,nid,rnti)
sym = nrPUCCH2( ___ ,'OutputDataType',datatype)
```

Description

`sym = nrPUCCH2(uciCW,nid,rnti)` returns the physical uplink control channel (PUCCH) format 2 modulation symbols, as defined in TS 38.211 Section 6.3.2.5 [1]. `uciCW` is the encoded uplink control information (UCI) codeword, as defined in TS 38.212 Section 6.3.1 [2]. The encoding consists of scrambling using scrambling identity `nid` and QPSK modulation. `rnti` specifies the radio network temporary identifier (RNTI) of the user equipment (UE).

`sym = nrPUCCH2( ___ ,'OutputDataType',datatype)` specifies the PUCCH symbol data type in addition to the input arguments in the previous syntax.

Examples

Generate PUCCH Format 2 Modulation Symbols

Create a random sequence of binary values corresponding to a UCI codeword of 100 bits.

```
uciCW = randi([0 1],100,1);
```

Generate PUCCH format 2 modulation symbols for the specified scrambling identity and RNTI.

```
nid = 148;
rnti = 160;
sym = nrPUCCH2(uciCW,nid,rnti)
```

```
sym = 50×1 complex

  -0.7071 + 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
      ⋮
```

Generate PUCCH Format 2 Modulation Symbols of Nondefault Data Type

Create a random sequence of binary values corresponding to a UCI codeword of 100 bits.

```
uciCW = randi([0 1],100,1);
```

Generate PUCCH format 2 modulation symbols of `single` data type for the specified scrambling identity and RNTI.

```
nid = 512;
rnti = 2563;
sym = nrPUCCH2(uciCW,nid,rnti,'OutputDataType','single')
```

```
sym = 50x1 single column vector

   0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 - 0.7071i
   0.7071 + 0.7071i
      ⋮
```

Input Arguments

`uciCW` — Encoded UCI codeword
logical column vector

Encoded UCI codeword, specified as a logical column vector. For more information, see TS 38.212 Section 6.3.1 [2].

Data Types: `int8` | `double` | `logical`

`nid` — Scrambling identity
integer from 0 to 1023

Scrambling identity, specified as an integer from 0 to 1023. `nid` is higher layer parameter *dataScramblingIdentityPUSCH*, ranging from 0 to 1023, if the higher layer parameter is configured. Otherwise, `nid` is the physical layer cell identity number *NCellID*, ranging from 0 to 1007.

For more information, see TS 38.211 Section 6.3.2.5.1.

Data Types: `double`

`rnti` — RNTI of UE
integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

datatype — Data type of output symbols
'double' (default) | 'single'

Data type of the output symbols, specified as 'double' or 'single'.

Data Types: char | string

Output Arguments

sym — PUCCH format 2 modulation symbols
complex column vector

PUCCH format 2 modulation symbols, returned as a complex column vector.

Data Types: single | double
Complex Number Support: Yes

Version History

Introduced in R2019a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.212. "NR; Multiplexing and channel coding." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The datatype input argument must be compile-time constant. For example, when specifying 'single' as the output data type, include {coder.Constant('OutputDataType'),coder.Constant('single')} in the -args value of the codegen function. For more information, see the coder.Constant class.

See Also

nrPUCCH0 | nrPUCCH1 | nrPUCCH3 | nrPUCCH4 | nrPUCCHPRBS | nrSymbolModulate

nrPUCCH3

Generate PUCCH format 3 modulation symbols

Syntax

```
sym = nrPUCCH3(uciCW,mod,nid,rnti,mrb)
sym = nrPUCCH3( ___ ,'OutputDataType',datatype)
```

Description

`sym = nrPUCCH3(uciCW,mod,nid,rnti,mrb)` returns the physical uplink control channel (PUCCH) format 3 modulation symbols, as defined in TS 38.211 Section 6.3.2.6 [1]. `uciCW` is the encoded uplink control information (UCI) codeword, as defined in TS 38.212 Section 6.3.1 [2]. The encoding consists of scrambling using scrambling identity `nid`, symbol modulation using modulation scheme `mod`, and transform precoding based on the allocated number of resource blocks `mrb`.

`rnti` specifies the radio network temporary identifier (RNTI) of the user equipment (UE).

`sym = nrPUCCH3( ___ ,'OutputDataType',datatype)` specifies the PUCCH symbol data type in addition to the input arguments in the previous syntax.

Examples

Generate PUCCH Format 3 QPSK Modulation Symbols

Create a random sequence of binary values corresponding to a UCI codeword of 96 bits.

```
uciCW = randi([0 1],96,1);
```

Using QPSK modulation, generate PUCCH format 3 modulation symbols for the specified scrambling identity, RNTI, and allocated bandwidth of two resource blocks.

```
modulation = 'QPSK';
nid = 148;
rnti = 160;
Mrb = 2;
sym = nrPUCCH3(uciCW,modulation,nid,rnti,Mrb)
```

```
sym = 48×1 complex

  -0.5774 - 0.2887i
  -0.0288 + 0.6273i
  -1.4717 + 0.3943i
   0.3237 - 0.3237i
   0.3660 + 0.5774i
  -0.3247 + 0.0149i
  -0.2887 + 1.1547i
  -1.0216 - 0.7397i
  -0.7113 - 0.2887i
  -0.6619 - 0.9010i
```

⋮

Generate PUCCH Format 3 Pi/2-BPSK Modulation Symbols

Create a random sequence of binary values corresponding to a UCI codeword of 96 bits.

```
uciCW = randi([0 1],96,1);
```

Using pi/2-BPSK modulation, generate PUCCH Format 3 modulation symbols of `single` data type for the specified scrambling identity, RNTI, and allocated bandwidth of two resource blocks.

```
modulation = 'pi/2-BPSK';
nid = 512;
rnti = 2563;
mrb = 2;
sym = nrPUCCH3(uciCW,modulation,nid,rnti,mrb,'OutputDataType','single')
```

```
sym = 96x1 single column vector

   1.1547 + 0.5774i
  -0.0197 - 0.1773i
   0.2887 + 0.2887i
   0.2887 - 0.1196i
   0.7113 + 0.2887i
  -0.4279 + 0.0475i
  -0.5774 - 0.5774i
   0.0475 - 0.4279i
   0.2887 + 0.7113i
  -0.1196 + 0.2887i
      ⋮
```

Input Arguments

uciCW — Encoded UCI codeword
logical column vector

Encoded UCI codeword, specified as a logical column vector. For more information, see TS 38.212 Section 6.3.1 [2].

Data Types: `int8` | `double` | `logical`

mod — Modulation scheme
`'pi/2-BPSK'` | `'QPSK'`

Modulation scheme, specified as `'pi/2-BPSK'` or `'QPSK'`. The modulation scheme determines the modulation type performed on the input codeword and the number of bits used per modulation symbol.

Modulation Scheme	Number of Bits Per Symbol
'pi/2-BPSK'	1

Modulation Scheme	Number of Bits Per Symbol
`'QPSK'`	2

Data Types: `char` | `string`

`nid` — Scrambling identity
integer from 0 to 1023

Scrambling identity, specified as an integer from 0 to 1023. `nid` is higher layer parameter *dataScramblingIdentityPUSCH*, ranging from 0 to 1023, if the higher layer parameter is configured. Otherwise, `nid` is the physical layer cell identity number *NCellID*, ranging from 0 to 1007.

For more information, see TS 38.211 Section 6.3.2.6.1.

Data Types: `double`

`rnti` — RNTI of UE
integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

`mrb` — Number of resource blocks
positive integer

Number of resource blocks associated with PUCCH format 3 transmission, specified as a positive integer. Preferred `mrb` values are 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, and 16.

Data Types: `double`

`datatype` — Data type of output symbols
`'double'` (default) | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

`sym` — PUCCH format 3 modulation symbols
complex column vector

PUCCH format 3 modulation symbols, returned as a complex column vector.

Data Types: `single` | `double`
Complex Number Support: Yes

Version History

Introduced in R2019a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.212. "NR; Multiplexing and channel coding." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

The `datatype` input argument must be compile-time constant. For example, when specifying `'single'` as the output data type, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

`nrPUCCH0` | `nrPUCCH1` | `nrPUCCH2` | `nrPUCCH4` | `nrPUCCHPRBS` | `nrSymbolModulate` | `nrTransformPrecode`

nrPUCCH4

Generate PUCCH format 4 modulation symbols

Syntax

```
sym = nrPUCCH4(uciCW,mod,nid,rnti,sf,occi)
sym = nrPUCCH4( ___ ,'OutputDataType',datatype)
```

Description

`sym = nrPUCCH4(uciCW,mod,nid,rnti,sf,occi)` returns the physical uplink control channel (PUCCH) format 4 modulation symbols for encoded uplink control information (UCI) codeword `uciCW`. The function implements TS 38.211 Section 6.3.2.6 [1]. The encoding consists of:

- Scrambling using scrambling identity `nid`.
- Symbol modulation using modulation scheme `mod`.
- Block-wise spreading using spreading factor `sf` and orthogonal cover code index `occi`.
- Transform precoding by considering 12 subcarriers associated with the PUCCH format 4 transmission.

`rnti` is the radio network temporary identifier (RNTI) of the user equipment (UE).

`sym = nrPUCCH4( ___ ,'OutputDataType',datatype)` specifies the PUCCH symbol data type in addition to the input arguments in the previous syntax.

Examples

Generate PUCCH Format 4 QPSK Modulation Symbols

Create a random sequence of binary values corresponding to a UCI codeword of 96 bits.

```
uciCW = randi([0 1],96,1);
```

Using QPSK modulation, generate PUCCH format 4 modulation symbols for the specified scrambling identity, RNTI, spreading factor, and orthogonal cover code index.

```
modulation = 'QPSK';
nid = 148;
rnti = 160;
sf = 2;
occi = 1;
sym = nrPUCCH4(uciCW,modulation,nid,rnti,sf,occi)
```

```
sym = 96×1 complex

   0.0000 + 0.0000i
  -0.8165 + 0.8165i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
```

```
   0.0000 + 0.0000i
  -0.8165 + 0.8165i
   0.0000 + 0.0000i
  -1.4142 + 1.4142i
   0.0000 + 0.0000i
  -0.8165 + 0.8165i
      ⋮
```

Generate PUCCH Format 4 Pi/2-BPSK Modulation Symbols

Create a random sequence of binary values corresponding to a UCI codeword of 192 bits.

```
uciCW = randi([0 1],192,1);
```

Using pi/2-BPSK modulation, generate PUCCH format 4 modulation symbols of `single` data type for the specified scrambling identity, RNTI, spreading factor, and orthogonal cover code index.

```
modulation = 'pi/2-BPSK';
nid = 285;
rnti = 897;
sf = 4;
occi = 3;
sym = nrPUCCH4(uciCW,modulation,nid,rnti,sf,occi,'OutputDataType','single')
```

```
sym = 768x1 single column vector

   0.0000 + 0.0000i
  -1.6330 - 1.6330i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
  -1.6330 - 1.6330i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.8165 + 0.8165i
      ⋮
```

Input Arguments

`uciCW` — Encoded UCI codeword
logical column vector

Encoded UCI codeword, specified as a logical column vector. For pi/2-BPSK modulation, the length of `uciCW` must be a multiple of 12. For QPSK modulation, the length of `uciCW` must be a multiple of 24. For more information, see TS 38.212 Section 6.3.1 [2].

Data Types: `int8` | `double` | `logical`

`mod` — Modulation scheme
`'pi/2-BPSK'` | `'QPSK'`

Modulation scheme, specified as `'pi/2-BPSK'` or `'QPSK'`. The modulation scheme determines the modulation type performed on the input codeword and the number of bits used per modulation symbol.

Modulation Scheme	Number of Bits Per Symbol
'pi/2-BPSK'	1
'QPSK'	2

Data Types: `char` | `string`

`nid` — Scrambling identity
integer from 0 to 1023

Scrambling identity, specified as an integer from 0 to 1023. `nid` is higher layer parameter *dataScramblingIdentityPUSCH*, ranging from 0 to 1023, if the higher layer parameter is configured. Otherwise, `nid` is the physical layer cell identity number *NCellID*, ranging from 0 to 1007.

For more information, see TS 38.211 Section 6.3.2.6.1.

Data Types: `double`

`rnti` — RNTI of UE
integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

`sf` — Spreading factor for PUCCH format 4
`2` | `4`

Spreading factor for PUCCH format 4, specified as `2` or `4`.

Data Types: `double`

`occi` — Orthogonal cover code index
nonnegative integer

Orthogonal cover code index, specified as a nonnegative integer. `occi` must be less than the spreading factor `sf`.

Data Types: `double`

`datatype` — Data type of output symbols
`'double'` (default) | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

`sym` — PUCCH format 4 modulation symbols
complex column vector

PUCCH format 4 modulation symbols, returned as a complex column vector.

Data Types: `single` | `double`
Complex Number Support: Yes

Version History

Introduced in R2019a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.213. "NR; Physical layer procedures for control." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

The `datatype` input argument must be compile-time constant. For example, when specifying `'single'` as the output data type, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

`nrPUCCH0` | `nrPUCCH1` | `nrPUCCH2` | `nrPUCCH3` | `nrPUCCHPRBS` | `nrSymbolModulate` | `nrTransformPrecode`

nrPUCCHDecode

Decode PUCCH modulation symbols

Syntax

```
[uciBits,symbols,detMet] = nrPUCCHDecode(carrier,pucch,ouci,sym)
[uciBits,symbols,detMet] = nrPUCCHDecode(carrier,pucch,ouci,sym,nVar)
[uciBits,symbols,detMet] = nrPUCCHDecode( ___ ,'DetectionThreshold',
detectionThreshold)
```

Description

`[uciBits,symbols,detMet] = nrPUCCHDecode(carrier,pucch,ouci,sym)` decodes physical uplink control channel (PUCCH) format-specific decoding and returns the uplink control information (UCI) bits, `uciBits`, as defined in TS 38.211 Sections 6.3.2.3 to 6.3.2.6 for all PUCCH formats. The function also returns received constellation symbols, `symbols`, and the detection metric, `detMet`. Input `carrier` specifies the carrier configuration. Input `pucch` specifies the format-specific PUCCH configuration. Input `ouci` specifies the uncoded UCI bits. When the number of UCI bits is less than 12, the function performs discontinuous transmission (DTX) detection by finding the normalized correlation coefficient of all of the possible reference sequences and then compares the maximum value against a threshold.

`[uciBits,symbols,detMet] = nrPUCCHDecode(carrier,pucch,ouci,sym,nVar)` specifies the variance of additive white Gaussian noise (AWGN) on the received symbols.

`[uciBits,symbols,detMet] = nrPUCCHDecode( ___ ,'DetectionThreshold', detectionThreshold)` specifies the detection threshold to perform DTX in addition to any of the input argument combinations in the previous syntaxes.

Examples

Decode PUCCH Format 0 Symbols for Positive SR Transmission

Create a default carrier configuration object.

```
carrier = nrCarrierConfig;
```

Specify a 15 kHz carrier with a normal cyclic prefix. Set the slot number to 63.

```
carrier.SubcarrierSpacing = 15;
carrier.CyclicPrefix = 'normal';
carrier.NSlot = 63;
```

Create a default PUCCH format 0 configuration object.

```
pucch = nrPUCCH0Config;
```

Specify the first symbol index in the PUCCH transmission slot as 11 and the number of allocated PUCCH symbols as 2. Enable intraslot frequency hopping and group hopping. Set the hopping identity to 512 and the initial cyclic shift to 5.

```
pucch.SymbolAllocation = [11 2];
pucch.FrequencyHopping = 'intraSlot';
pucch.GroupHopping = 'enable';
pucch.HoppingID = 512;
pucch.InitialCyclicShift = 5;
```

Specify a transmission without hybrid automatic repeat request acknowledgment (HARQ-ACK) and with a positive scheduling request (SR).

```
ack = zeros(0,1);
sr = 1;           % Positive SR transmission
```

Generate PUCCH format 0 modulation symbols.

```
sym = nrPUCCH(carrier,pucch,{ack sr});
```

Decode the PUCCH format 0 modulation symbols.

```
uci = nrPUCCHDecode(carrier,pucch,[numel(ack) numel(sr)],sym);
```

Verify that the received UCI bits match the transmitted UCI bits.

```
isequal(uci{1},ack)
```

```
ans = logical
   1
```

```
isequal(uci{2},sr)
```

```
ans = logical
   1
```

Decode PUCCH Format 1 Symbols with Detection Threshold

Create a default carrier configuration object.

```
carrier = nrCarrierConfig;
```

Specify a 60 kHz carrier with an extended cyclic prefix. Set the slot number to 7.

```
carrier.SubcarrierSpacing = 60;
carrier.CyclicPrefix = 'extended';
carrier.NSlot = 7;
```

Create a default PUCCH format 1 configuration object.

```
pucch = nrPUCCH1Config;
```

Specify the first symbol index in the PUCCH transmission slot as 3 and the number of allocated PUCCH symbols as 9. Enable intraslot frequency hopping and group hopping. Set the hopping identity to 512, the initial cyclic shift to 9, and the orthogonal cover code index to 1.

```
pucch.SymbolAllocation = [3 9];
pucch.FrequencyHopping = 'intraSlot';
```

```
pucch.GroupHopping = 'enable';
pucch.HoppingID = 512;
pucch.InitialCyclicShift = 9;
pucch.OCCI = 1;
```

Generate PUCCH format 1 modulation symbols for 1-bit UCI.

```
uci = 1;
sym = nrPUCCH(carrier,pucch,uci);
```

Decode the PUCCH format 1 modulation symbols with a detection threshold of 0.7.

```
rxUCI = nrPUCCHDecode(carrier,pucch,numel(uci),sym,'DetectionThreshold',0.7);
```

Verify that the received UCI bits match the transmitted UCI bits.

```
isequal(rxUCI{1},uci)
```

```
ans = logical
   1
```

Decode PUCCH Format 2 Symbols

Create a default carrier configuration object.

```
carrier = nrCarrierConfig;
```

Create a default PUCCH format 2 configuration object.

```
pucch = nrPUCCH2Config;
```

Specify the data scrambling identity as 1000 and the radio network temporary identifier as 160.

```
pucch.NID = 1000;
pucch.RNTI = 160;
```

Get random UCI bits.

```
ouci = 20;
uci = randi([0 1],ouci,1);
```

Encode the UCI bits.

```
uciCW = nrUCIEncode(uci,100);
```

Generate PUCCH modulation format 2 symbols.

```
sym = nrPUCCH(carrier,pucch,uciCW);
```

Decode the PUCCH modulation format 2 symbols.

```
rxUCI = nrPUCCHDecode(carrier,pucch,ouci,sym);
```

Verify that the received UCI bits match the transmitted UCI bits.

```
isequal(uciCW,double(rxUCI{1}<0))
```

```
ans = logical
   1
```

Decode PUCCH Format 3 Symbols

Create a default carrier configuration object.

```
carrier = nrCarrierConfig;
```

Specify a 30 kHz carrier with an extended cyclic prefix. Set the number of resource blocks in the carrier resource grid to 80.

```
carrier.SubcarrierSpacing = 30;
carrier.NSizeGrid = 80;
```

Create a default PUCCH format 3 configuration object.

```
pucch = nrPUCCH3Config;
```

Specify the modulation scheme as pi/2-BPSK, the phyiscal resource blocks (PRB) allocation of the PUCCH to range from 70 to 74 (occupying 5 resource blocks), and the radio network temporary identifier as 2560.

```
pucch.Modulation = 'pi/2-BPSK';
pucch.PRBSet = 70:74;
pucch.RNTI = 2560;
```

Generate PUCCH format 3 resource element indices.

```
[pucchIndices,pucchInfo] = nrPUCCHIndices(carrier,pucch);
G = pucchInfo.G;                                          % Rate-matched length
```

Get random UCI bits.

```
ouci = 30;
uciBits = randi([0 1],ouci,1);
```

Encode the UCI bits.

```
codedUCI = nrUCIEncode(uciBits,G);
```

Generate PUCCH format 3 modulated symbols.

```
sym = nrPUCCH(carrier,pucch,codedUCI);
```

Generate demodulation reference signal (DM-RS) symbols and indices.

```
dmrsSym = nrPUCCHDMRS(carrier,pucch);
dmrsInd = nrPUCCHDMRSIndices(carrier,pucch);
```

Create a resource grid. Map the PUCCH symbols and the DM-RS symbols to the grid.

```
resGrid = nrResourceGrid(carrier);
resGrid(pucchIndices) = sym;
resGrid(dmrsInd) = dmrsSym;
```

Perform orthogonal frequency division multiplexing (OFDM) modulation.

```
txWaveform = nrOFDMModulate(carrier,resGrid);
```

Perform back-to-back decoding.

```
rxWaveform = txWaveform;
rxGrid = nrOFDMDemodulate(carrier,rxWaveform);
```

Extract PUCCH format 3 symbols from the received grid.

```
rxSym = nrExtractResources(pucchIndices,rxGrid);
```

Decode the PUCCH format 3 modulation symbols, and then decode the UCI bits.

```
rxUCI = nrPUCCHDecode(carrier,pucch,ouci,rxSym);
rxUCIBits = nrUCIDecode(rxUCI{1},ouci);
```

Verify that the received UCI bits match the transmitted UCI bits.

```
isequal(uciBits,rxUCIBits)
```

```
ans = logical
   1
```

Decode PUCCH Format 4 Symbols

Create a default carrier configuration object, and then set the cell identity as 140.

```
carrier = nrCarrierConfig;
carrier.NCellID = 140;
```

Create a default PUCCH format 4 configuration object.

```
pucch = nrPUCCH4Config;
```

Specify the spreading factor as 4, the OCCI as 3, and the radio network temporary identifier as 750.

```
pucch.SpreadingFactor = 4;
pucch.OCCI = 3;
pucch.RNTI = 750;
```

Get random UCI bits.

```
ouci = 20;
uci = randi([0 1],ouci,1);
```

Encode the UCI bits.

```
uciCW = nrUCIEncode(uci,60);
```

Generate PUCCH modulation format 4 symbols.

```
sym = nrPUCCH(carrier,pucch,uciCW);
```

Decode the PUCCH modulation format 4 symbols.

```
rxUCI = nrPUCCHDecode(carrier,pucch,ouci,sym);
```

Verify that the received UCI bits match the transmitted UCI bits.

```
isequal(uciCW,double(rxUCI{1}<0))
```

```
ans = logical
   1
```

Generate and Decode Symbols for Mixed PUCCH Transmission

Compare the transmission and reception of PUCCH formats 0 and 2 for two different user equipments.

Carrier Configuration

Create a default carrier configuration object, and then set the cell identity to 10.

```
carrier = nrCarrierConfig;
carrier.NCellID = 10;
```

PUCCH Configuration

Create a PUCCH format 0 configuration object with the specified properties for user equipment (UE) 1.

```
pucch0 = nrPUCCH0Config;
pucch0.PRBSet = 0;
pucch0.HoppingID = 0;
```

Create a PUCCH format 2 configuration object with the specified properties for UE 2.

```
pucch2 = nrPUCCH2Config;
pucch2.PRBSet = 0;
pucch2.RNTI = 10;
```

Channel Propagation Model Configuration

Define the channel configuration structure using an `nrTDLChannel` System object.

```
channel = nrTDLChannel;
channel.NumReceiveAntennas = 4;
info1 = nrOFDMInfo(carrier);
channel.SampleRate = info1.SampleRate;                         % Set waveform sample rate
chInfo = info(channel);
maxChDelay = ceil(max(chInfo.PathDelays*channel.SampleRate));
maxChDelay = maxChDelay + chInfo.ChannelFilterDelay;
```

Noise Configuration

Normalize the noise power of the carrier, and the configure the random number generator.

```
SNRdB = 21.0;
SNR = 10^(SNRdB/20);
N = 1/(SNR*sqrt(double(info1.Nfft)))/sqrt(2.0);
rng('default');
```

PUCCH Symbols and Indices Generation for Format 0 and Format 2

Specify the HARQ-ACK bits for PUCCH format 0.

```
hi1 = [0;1];
disp('hi1:');
```

```
hi1:
```

```
disp(hi1.');
```

```
     0     1
```

Generate PUCCH format 0 modulation symbols.

```
pucch0Sym = nrPUCCH(carrier,pucch0,hi1);
```

Generate DM-RS symbols for PUCCH formats 0 and 2.

```
pucch0DMRSSym = nrPUCCHDMRS(carrier,pucch0);
pucch2DRSSym = nrPUCCHDMRS(carrier,pucch2);
```

Generate PUCCH format 2 resource element indices.

```
[pucch2Ind,pucch2Info] = nrPUCCHIndices(carrier,pucch2);
```

Create channel quality information (CQI) bits.

```
cqi = [0; 1; 1; 0; 0; 1];
disp('cqi:');
```

```
cqi:
```

```
disp(cqi.');
```

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

Encode the CQI bits.

```
codedcqi = nrUCIEncode(cqi,pucch2Info.G);
```

Generate PUCCH format 2 modulation symbols.

```
pucch2Sym = nrPUCCH(carrier,pucch2,codedcqi);
```

PUCCH Index Generation for Format 0 and Format 2

Generate PUCCH indices for formats 0 and 2.

```
pucch0Indices = nrPUCCHIndices(carrier,pucch0);
pucch2Indices = nrPUCCHIndices(carrier,pucch2);
```

Generate DM-RS indices for PUCCH formats 0 and 2.

```
pucch0DMRSIndices = nrPUCCHDMRSIndices(carrier,pucch0);
pucch2DMRSIndices = nrPUCCHDMRSIndices(carrier,pucch2);
```

Transmission with PUCCH Format 0

Create a resource grid. Map the PUCCH symbols and DM-RS symbols of format 0 to the grid.

```
grid1 = nrResourceGrid(carrier);
grid1(pucch0Indices) = pucch0Sym;
grid1(pucch0DMRSIndices) = pucch0DMRSSym;
```

Perform OFDM modulation and pass the waveform through the tapped delay line (TDL) channel.

```
txwave1 = nrOFDMModulate(carrier,grid1);
txwave1 = [txwave1; zeros(maxChDelay,size(txwave1,2))];
[rxwave1,pathGains1,sampleTimes1] = channel(txwave1);
```

Transmission with PUCCH Format 2

Create a resource grid. Map the PUCCH symbols and DM-RS symbols of format 2 to the grid.

```
grid2 = nrResourceGrid(carrier);
grid2(pucch2Indices) = pucch2Sym;
grid2(pucch2DMRSIndices) = pucch2DRSSym;
```

Perform OFDM modulation, and then pass the waveform through the TDL channel.

```
txwave2 = nrOFDMModulate(carrier,grid2);
release(channel);
channel.Seed = 15;
txwave2 = [txwave2; zeros(maxChDelay,size(txwave2,2))];
[rxwave2,pathGains,sampleTimes] = channel(txwave2);
```

Reception at Base Station

Add AWGN to the received signal.

```
rxwave = rxwave1 + rxwave2;                                  % Add both the faded signals
noise = N*complex(randn(size(rxwave)),randn(size(rxwave)));
rxwave = rxwave + noise;
```

Demodulate the received signal and extract PUCCH format 0 RE indices.

```
rxgrid1 = nrOFDMDemodulate(carrier,rxwave(1:end,:));
rxpucch0 = nrExtractResources(pucch0Indices,rxgrid1);
```

Decode PUCCH Format 0 Symbols

Decode the PUCCH format 0 symbols.

```
rxhi1 = nrPUCCHDecode(carrier,pucch0,length(hi1),rxpucch0);
disp('rxhi1:');
```

```
rxhi1:
```

```
disp(rxhi1{1}.');
```

```
   0   1
```

Verify that the received HARQ-ACK bits match the transmitted HARQ-ACK bits.

```
if isequal(hi1,rxhi1{1})
    disp('PUCCH format 0 data decoded')
else
    disp('PUCCH format 0 data not decoded')
end
```

```
PUCCH format 0 data decoded
```

Decode PUCCH Format 2 Symbols

Estimate the timing offset for the transmission, and then perform the OFDM demodulation on the received waveform.

```
[t,mag] = nrTimingEstimate(carrier,rxwave,pucch2DMRSIndices,pucch2DRSSym);
if t > maxChDelay
    t = 0;
end
rxgrid2 = nrOFDMDemodulate(carrier,rxwave(1+t:end,:));
```

Get the practical channel estimate, and then extract PUCCH format 2 RE indices.

```
[H2, n0] = nrChannelEstimate(carrier,rxgrid2,pucch2DMRSIndices,pucch2DRSSym);
[pucchrx2, pucchH2] = nrExtractResources(pucch2Indices,rxgrid2,H2);
```

Perform minimum mean square error (MMSE) equalization on the extracted PUCCH format 2 resource elements.

```
pucch2eq = nrEqualizeMMSE(pucchrx2,pucchH2,n0);
```

Decode the PUCCH format 2 modulation symbols, and then decode the CQI bits.

```
rxcodedcqi = nrPUCCHDecode(carrier,pucch2,length(cqi),pucch2eq);
rxcqi = nrUCIDecode(rxcodedcqi{1},length(cqi));
disp('rxcqi:');
```

```
rxcqi:
```

```
disp(rxcqi.');
```

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

Verify that the received CQI bits match the transmitted CQI bits.

```
if isequal(cqi,rxcqi)
    disp('PUCCH format 2 data decoded')
else
    disp('PUCCH format 2 data not decoded')
end
```

```
PUCCH format 2 data decoded
```

Input Arguments

carrier — Carrier configuration parameters
`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function uses only these `nrCarrierConfig` object properties.

- `NCellID`
- `SubcarrierSpacing`
- `CyclicPrefix`

- `NSizeGrid`
- `NStartGrid`
- `NSlot`

pucch — PUCCH configuration parameters
`nrPUCCH0Config` object | `nrPUCCH1Config` object | `nrPUCCH2Config` object | `nrPUCCH3Config` object | `nrPUCCH4Config` object

PUCCH configuration parameters, specified as one of these options.

- For format 0, specify an `nrPUCCH0Config` object. The function uses only these object properties.
  - `SymbolAllocation`
  - `FrequencyHopping`
  - `GroupHopping`
  - `HoppingID`
  - `InitialCyclicShift`
- For format 1, specify an `nrPUCCH1Config` object. The function uses only these object properties.
  - `SymbolAllocation`
  - `FrequencyHopping`
  - `GroupHopping`
  - `HoppingID`
  - `InitialCyclicShift`
  - `OCCI`
- For format 2, specify an `nrPUCCH2Config` object. The function uses only these object properties.
  - `NID`
  - `RNTI`
- For format 3, specify an `nrPUCCH3Config` object. The function uses only these object properties.
  - `PRBSet`
  - `Modulation`
  - `NID`
  - `RNTI`
- For format 4, specify an `nrPUCCH4Config` object. The function uses only these object properties.
  - `Modulation`
  - `SpreadingFactor`
  - `OCCI`
  - `NID`
  - `RNTI`

ouci — Number of uncoded UCI bits
nonnegative integer | two-element vector of nonnegative integers

Number of uncoded UCI bits, specified as a nonnegative integer or depending on the format type.

- For format 0, you must specify a nonnegative integer or a two-element vector of nonnegative integers.
  - When you specify this value as a scalar, the uncoded UCI bits represents the hybrid automatic repeat request acknowledgment (HARQ-ACK) bits.
  - When you specify this value as a vector, the first element represents the HARQ-ACK bits, and the second element represents the scheduling request (SR) bit.
- For format 1, you must specify a nonnegative integer. When you specify this value, the uncoded UCI bits represents the HARQ-ACK bits or SR bits.
- For formats 2, 3, and 4, the uncoded UCI bits represents the number of UCI bits in both parts.

Data Types: `double`

`sym` — PUCCH modulation symbols
complex-valued column vector | complex-valued matrix

PUCCH modulation symbols for the specified format, returned as a complex-valued column vector or complex-valued matrix.

For formats 0 and 1, the `sym` argument must be a matrix with the number of columns equal to the number of receive antennas. For formats 2, 3, and 4, the `sym` argument must be a column vector.

Data Types: `single` | `double`
Complex Number Support: Yes

`nVar` — Noise variance
`1e-10` (default) | nonnegative scalar

Noise variance, specified as a nonnegative scalar. The function scales the symbols with the variance of additive white Gaussian noise (AWGN). The default value corresponds to an signal to noise ratio (SNR) of 100 dB, assuming unit signal power.

When the noise variance value is less than 1e-10, the function uses the value of `1e-10`.

Data Types: `double`

`detectionThreshold` — Detection threshold
`[]` | scalar in range from 0 to 1

Detection threshold, specified as `[]` or a scalar in range from 0 to 1. When you do not specify the input or you set it to `[]`, the function selects one of these default values based on the format type.

- For format 0, the default value is `0.49` for one OFDM symbol and `0.42` for two OFDM symbols.
- For format 1, the default value is `0.22`.
- For formats 2, 3, and 4, the default value is `0.45`.

Data Types: `double`

Output Arguments

`uciBits` — UCI bits
one-element cell array | two-element cell array

UCI bits, returned as a one- or two-element cell array depending on the format type. For format 0 and 1, the data type of `uciBits` is `int8`. For format 2, 3, and 4, `uciBits` inherits the data type of the `sym` input.

- For format 0, `uciBits` is a one- or two-element cell array with UCI hard bits.
  - For a one-element cell array, the function returns the HARQ-ACK hard bits.
  - For a two-element cell array, the function returns the first element as HARQ-ACK hard bits and the second element as SR bit.

  PUCCH format 0 decoding involves DTX detection followed by finding the sequence that has the maximum normalized correlation metric.
- For format 1, `uciBits` is a one-element cell array with UCI hard bits. PUCCH format 1 decoding involves DTX detection followed by matched filtering.
- For formats 2, 3, and 4, `uciBits` is a one-element cell array with UCI soft bits. For PUCCH formats 2, 3, and 4, UCI soft bits are returned post DTX detection (applicable up to 12 uncoded uci bits) and inverse processing of PUCCH encoding.

Data Types: `cell`

`symbols` — Received constellation symbols
column-valued column vector

Received constellation symbols, returned as a column-valued column vector. `symbols` inherits the data type of the `sym` input.

Data Types: `double` | `single`
Complex Number Support: Yes

`detMet` — Detection metric
integer

Detection metric, returned as an integer. For PUCCH formats 2, 3, and 4, when the `ouci` input is not in the range from 3 to 11, this output is `0`.

Data Types: `double` | `single`

Version History

Introduced in R2021b

SR bit output value in PUCCH format 0 decoding has changed

Behavior changed in R2022a

For PUCCH format 0, the function now returns `0` instead of `[]` for the SR bit value when the detection metric is below the DTX detection threshold. For PUCCH format 0, the SR bit is returned in the second cell of the `uciBits` output of the function.

Detection threshold in PUCCH format 1 decoding has changed

Behavior changed in R2022a

For PUCCH format 1, the function now uses 0.22 instead of 0.36 as the default DTX detection threshold.

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying `'single'` as the output data type, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrUCIDecode` | `nrPUCCH` | `nrPUCCHIndices`

Objects
`nrPUCCH0Config` | `nrPUCCH1Config` | `nrPUCCH2Config` | `nrPUCCH3Config` | `nrPUCCH4Config`

Topics
"NR PUCCH Block Error Rate"

nrPUCCHDMRS

Generate PUCCH DM-RS symbols

Syntax

```
sym = nrPUCCHDMRS(carrier,pucch)
sym = nrPUCCHDMRS(carrier,pucch,'OutputDataType',datatype)
```

Description

`sym = nrPUCCHDMRS(carrier,pucch)` generates `sym`, which contains demodulation reference signal (DM-RS) symbols of a physical uplink control channel (PUCCH), as defined in TS 38.211 Section 6.4.1.3 [1], for all PUCCH formats. `carrier` specifies the carrier configuration. `pucch` specifies the PUCCH configuration. For PUCCH formats 1, 3, and 4, setting the `GroupHopping` property of `pucch` is set to `'disable'` enables sequence hopping. In this case, the function selects a sequence number, which might be inappropriate for short base sequences. For PUCCH format 0, the function returns an empty value for `sym`.

`sym = nrPUCCHDMRS(carrier,pucch,'OutputDataType',datatype)` specifies the data type of the PUCCH symbols.

Examples

Generate PUCCH DM-RS Format 2 Symbols and Indices

Create a default carrier configuration object.

```
carrier = nrCarrierConfig;
```

Specify the number of RBs in the carrier resource grid as 50, the start of the carrier resource grid relative to CRB 0 as 120, and the slot number as 25.

```
carrier.NSizeGrid = 50;
carrier.NStartGrid = 120;
carrier.NSlot = 25;
```

Create a default PUCCH format 2 configuration object.

```
pucch2 = nrPUCCH2Config;
```

Specify the first symbol index in the PUCCH transmission slot as 7 and the number of allocated PUCCH symbols as 2. Set the PRBs allocated for the PUCCH to range from 40 to 45. Enable intraslot frequency hopping and set the starting PRB index of the second hop to 20. Specify the DM-RS scrambling identity as 1005.

```
pucch2.SymbolAllocation = [7 2];
pucch2.PRBSet = 40:45;
pucch2.FrequencyHopping = 'intraSlot';
pucch2.SecondHopStartPRB = 20;
pucch2.NID0 = 1005;
```

Generate the DM-RS symbols of PUCCH format 2.

```
sym = nrPUCCHDMRS(carrier,pucch2)
```

```
sym = 48×1 complex

   0.7071 - 0.7071i
  -0.7071 + 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
   0.7071 - 0.7071i
   0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 + 0.7071i
      ⋮
```

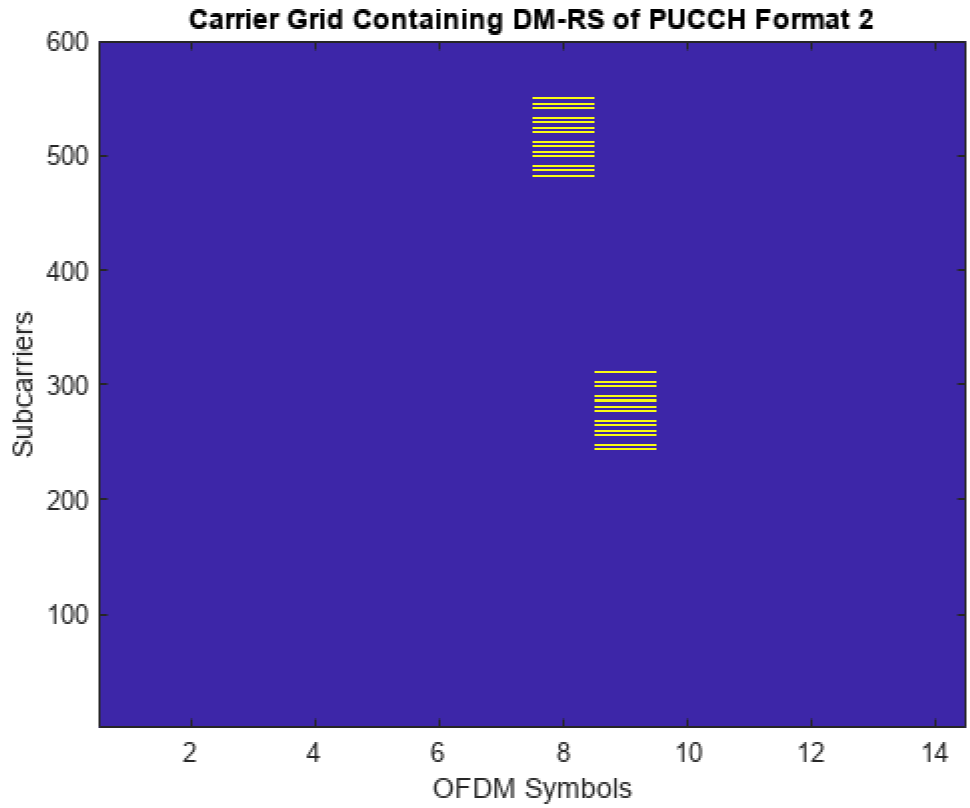
Generate the DM-RS RE indices of PUCCH format 2.

```
ind = nrPUCCHDMRSIndices(carrier,pucch2)
```

```
ind = 48x1 uint32 column vector

   4682
   4685
   4688
   4691
   4694
   4697
   4700
   4703
   4706
   4709
      ⋮
```

Plot the generated symbols and REs in the carrier resource grid.

```
resGrid = nrResourceGrid(carrier);
resGrid(ind) = sym;
imagesc(abs(resGrid))
axis xy
xlabel('OFDM Symbols')
ylabel('Subcarriers')
title('Carrier Grid Containing DM-RS of PUCCH Format 2')
```



Generate PUCCH DM-RS Format 4 Symbols and Indices

Create a default carrier configuration object.

```
carrier = nrCarrierConfig;
```

Specify the number of RBs in the carrier resource grid to 10.

```
carrier.NSizeGrid = 10;
```

Create a default PUCCH format 4 configuration object.

```
pucch4 = nrPUCCH4Config;
```

Set the first symbol index in the PUCCH transmission slot to 2 and the number of allocated PUCCH symbols to 12. Enable group hopping and the additional DM-RS configuration flag. Specify the hopping identity as 120, the spreading factor as 4, and the orthogonal cover code index (OCCI) as 3.

```
pucch4.SymbolAllocation = [2 12];
pucch4.GroupHopping = 'enable';
pucch4.HoppingID = 120;
pucch4.SpreadingFactor = 4;
pucch4.OCCI = 3;
pucch4.AdditionalDMRS = 1;
```

Generate the DM-RS symbols of PUCCH format 4 with data type `single`.

```
sym = nrPUCCHDMRS(carrier,pucch4, 'OutputDataType','single')
```

```
sym = 48x1 single column vector

   0.7071 - 0.7071i
   0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
   0.7071 + 0.7071i
  -0.7071 - 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
      ⋮
```

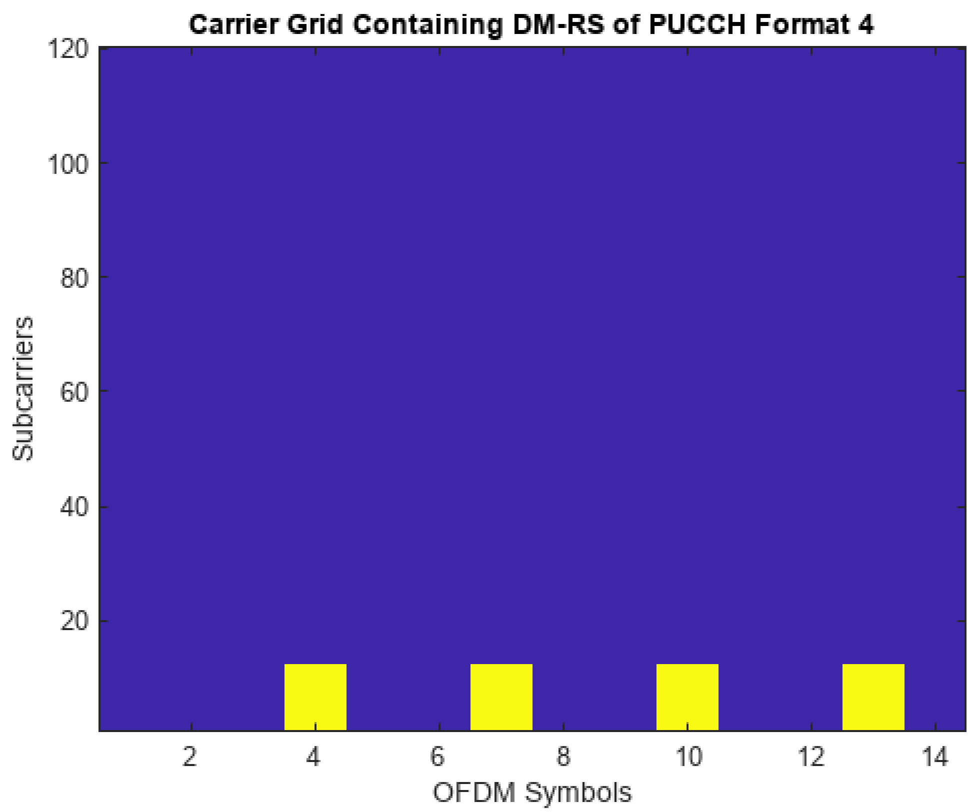
Generate the DM-RS RE indices of PUCCH format 4.

```
ind = nrPUCCHDMRSIndices(carrier,pucch4)
```

```
ind = 48x1 uint32 column vector

   361
   362
   363
   364
   365
   366
   367
   368
   369
   370
      ⋮
```

Plot the generated symbols and REs in the carrier resource grid.

```
resGrid = nrResourceGrid(carrier);
resGrid(ind) = sym;
imagesc(abs(resGrid))
axis xy
xlabel('OFDM Symbols')
ylabel('Subcarriers')
title('Carrier Grid Containing DM-RS of PUCCH Format 4')
```



Input Arguments

`carrier` — Carrier configuration parameters
`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function only uses these `nrCarrierConfig` object properties.

- `NCellID`
- `SubcarrierSpacing`
- `CyclicPrefix`
- `NSizeGrid`
- `NStartGrid`
- `NSlot`

`pucch` — PUCCH configuration parameters
`nrPUCCH1Config` object | `nrPUCCH2Config` object | `nrPUCCH3Config` object | `nrPUCCH4Config` object

PUCCH configuration parameters, specified as one of these options.

- For format 1, specify an `nrPUCCH1Config` object. The function uses only these object properties.

  - `SymbolAllocation`
  - `GroupHopping`
  - `HoppingID`
  - `InitialCyclicShift`
  - `OCCI`
- For format 2, specify an `nrPUCCH2Config` object. The function uses only these object properties.
  - `NStartBWP`
  - `SymbolAllocation`
  - `PRBSet`
  - `FrequencyHopping`
  - `SecondHopStartPRB`
  - `NID0`
- For format 3, specify an `nrPUCCH3Config` object. The function uses only these object properties.
  - `SymbolAllocation`
  - `PRBSet`
  - `FrequencyHopping`
  - `GroupHopping`
  - `HoppingID`
  - `AdditionalDMRS`
- For format 4, specify an `nrPUCCH4Config` object. The function uses only these object properties.
  - `SymbolAllocation`
  - `FrequencyHopping`
  - `GroupHopping`
  - `HoppingID`
  - `SpreadingFactor`
  - `OCCI`
  - `AdditionalDMRS`

For PUCCH format 0, the function returns an empty value.

`datatype` — Data type of output symbols
`'double'` | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

`sym` — PUCCH DM-RS modulation symbols
complex matrix

PUCCH DM-RS modulation symbols, returned as a complex matrix. For PUCCH format 0, this output is empty.

Data Types: `single` | `double`
Complex Number Support: Yes

Version History

Introduced in R2021a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

The `datatype` input argument must be compile-time constant. For example, when specifying `'single'` as the output data type, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrPUCCH` | `nrPUCCHDMRSIndices` | `nrPUCCHIndices`

Objects

`nrPUCCH0Config` | `nrPUCCH1Config` | `nrPUCCH2Config` | `nrPUCCH3Config` | `nrPUCCH4Config`

nrPUCCHDMRSIndices

Generate PUCCH DM-RS resource element indices

Syntax

```
ind = nrPUCCHDMRSIndices(carrier,pucch)
ind = nrPUCCHDMRSIndices(carrier,pucch,Name,Value)
```

Description

`ind = nrPUCCHDMRSIndices(carrier,pucch)` generates `ind`, which contains demodulation reference signal (DM-RS) resource element (RE) indices of a physical uplink control channel (PUCCH), as defined in TS 38.211 Section 6.4.1.3 [1], for all PUCCH formats. `carrier` specifies the carrier configuration. `pucch` specifies the PUCCH configuration. For PUCCH format 0, the function returns an empty value for `sym`.

`ind = nrPUCCHDMRSIndices(carrier,pucch,Name,Value)` specifies output formatting options using one or more name-value arguments.

Examples

Generate PUCCH DM-RS Format 2 Symbols and Indices

Create a default carrier configuration object.

```
carrier = nrCarrierConfig;
```

Specify the number of RBs in the carrier resource grid as 50, the start of the carrier resource grid relative to CRB 0 as 120, and the slot number as 25.

```
carrier.NSizeGrid = 50;
carrier.NStartGrid = 120;
carrier.NSlot = 25;
```

Create a default PUCCH format 2 configuration object.

```
pucch2 = nrPUCCH2Config;
```

Specify the first symbol index in the PUCCH transmission slot as 7 and the number of allocated PUCCH symbols as 2. Set the PRBs allocated for the PUCCH to range from 40 to 45. Enable intraslot frequency hopping and set the starting PRB index of the second hop to 20. Specify the DM-RS scrambling identity as 1005.

```
pucch2.SymbolAllocation = [7 2];
pucch2.PRBSet = 40:45;
pucch2.FrequencyHopping = 'intraSlot';
pucch2.SecondHopStartPRB = 20;
pucch2.NID0 = 1005;
```

Generate the DM-RS symbols of PUCCH format 2.

```
sym = nrPUCCHDMRS(carrier,pucch2)
```

```
sym = 48×1 complex

   0.7071 - 0.7071i
  -0.7071 + 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
   0.7071 - 0.7071i
   0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 + 0.7071i
      ⋮
```

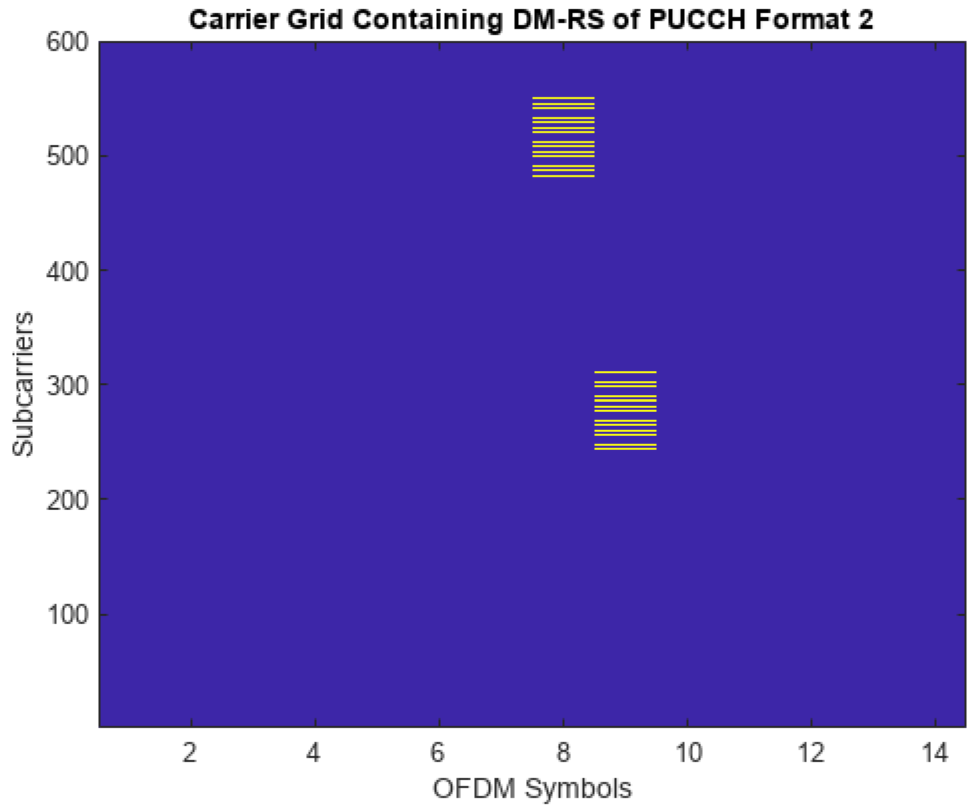
Generate the DM-RS RE indices of PUCCH format 2.

```
ind = nrPUCCHDMRSIndices(carrier,pucch2)
```

```
ind = 48x1 uint32 column vector

   4682
   4685
   4688
   4691
   4694
   4697
   4700
   4703
   4706
   4709
      ⋮
```

Plot the generated symbols and REs in the carrier resource grid.

```
resGrid = nrResourceGrid(carrier);
resGrid(ind) = sym;
imagesc(abs(resGrid))
axis xy
xlabel('OFDM Symbols')
ylabel('Subcarriers')
title('Carrier Grid Containing DM-RS of PUCCH Format 2')
```



Generate PUCCH DM-RS Format 4 Symbols and Indices

Create a default carrier configuration object.

```
carrier = nrCarrierConfig;
```

Specify the number of RBs in the carrier resource grid to 10.

```
carrier.NSizeGrid = 10;
```

Create a default PUCCH format 4 configuration object.

```
pucch4 = nrPUCCH4Config;
```

Set the first symbol index in the PUCCH transmission slot to 2 and the number of allocated PUCCH symbols to 12. Enable group hopping and the additional DM-RS configuration flag. Specify the hopping identity as 120, the spreading factor as 4, and the orthogonal cover code index (OCCI) as 3.

```
pucch4.SymbolAllocation = [2 12];
pucch4.GroupHopping = 'enable';
pucch4.HoppingID = 120;
pucch4.SpreadingFactor = 4;
pucch4.OCCI = 3;
pucch4.AdditionalDMRS = 1;
```

Generate the DM-RS symbols of PUCCH format 4 with data type `single`.

```
sym = nrPUCCHDMRS(carrier,pucch4, 'OutputDataType','single')
```

```
sym = 48x1 single column vector

   0.7071 - 0.7071i
   0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
   0.7071 + 0.7071i
  -0.7071 - 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
       ⋮
```

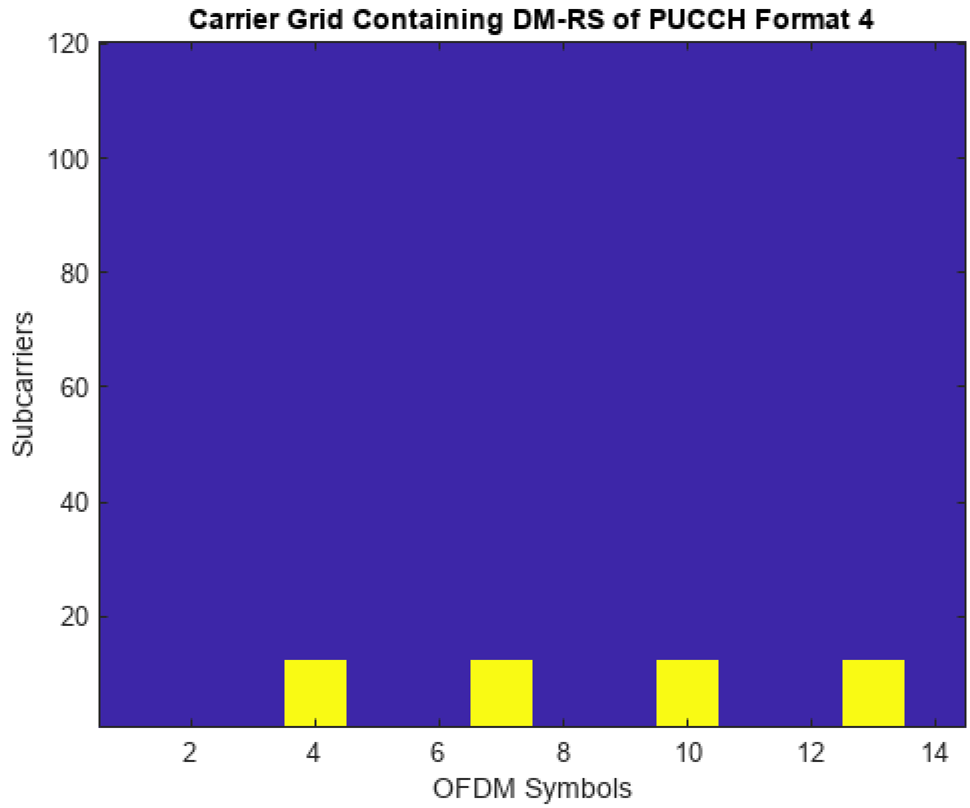
Generate the DM-RS RE indices of PUCCH format 4.

```
ind = nrPUCCHDMRSIndices(carrier,pucch4)
```

```
ind = 48x1 uint32 column vector

   361
   362
   363
   364
   365
   366
   367
   368
   369
   370
       ⋮
```

Plot the generated symbols and REs in the carrier resource grid.

```
resGrid = nrResourceGrid(carrier);
resGrid(ind) = sym;
imagesc(abs(resGrid))
axis xy
xlabel('OFDM Symbols')
ylabel('Subcarriers')
title('Carrier Grid Containing DM-RS of PUCCH Format 4')
```



Input Arguments

carrier — Carrier configuration parameters
nrCarrierConfig object

Carrier configuration parameters for a specific OFDM numerology, specified as an nrCarrierConfig object. This function only uses these nrCarrierConfig object properties.

- SubcarrierSpacing
- CyclicPrefix
- NSizeGrid
- NStartGrid
- NSlot

pucch — PUCCH configuration parameters
nrPUCCH1Config object | nrPUCCH2Config object | nrPUCCH3Config object | nrPUCCH4Config object

PUCCH configuration parameters, specified as one of these options.

- For format 1, specify an nrPUCCH1Config object. The function uses only these object properties.
  - NSizeBWP

  - `NStartBWP`
  - `SymbolAllocation`
  - `PRBSet`
  - `FrequencyHopping`
  - `SecondHopStartPRB`
- For format 2, specify an `nrPUCCH2Config` object. The function uses only these object properties.
  - `NSizeBWP`
  - `NStartBWP`
  - `SymbolAllocation`
  - `PRBSet`
  - `FrequencyHopping`
  - `SecondHopStartPRB`
- For format 3, specify an `nrPUCCH3Config` object. The function uses only these object properties.
  - `NSizeBWP`
  - `NStartBWP`
  - `SymbolAllocation`
  - `PRBSet`
  - `FrequencyHopping`
  - `SecondHopStartPRB`
  - `AdditionalDMRS`
- For format 4, specify an `nrPUCCH4Config` object. The function uses only these object properties.
  - `NSizeBWP`
  - `NStartBWP`
  - `SymbolAllocation`
  - `PRBSet`
  - `FrequencyHopping`
  - `SecondHopStartPRB`
  - `AdditionalDMRS`

For PUCCH format 0, the function returns an empty value.

Name-Value Pair Arguments

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

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

Example: `'IndexStyle','subscript'` specifies the RE indexing form of the output.

`IndexStyle` — RE indexing form

`'index'` (default) | `'subscript'`

RE indexing form, specified as one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

IndexBase — RE indexing base
`'1based'` (default) | `'0based'`

RE indexing base, specified as one of these values:

- `'1based'` — The index counting starts from 1.
- `'0based'` — The index counting starts from 0.

Data Types: `char` | `string`

IndexOrientation — RE indexing orientation
`'carrier'` (default) | `'bwp'`

RE indexing orientation, specified as one of these values:

- `'carrier'` — Indices are referenced with respect to the carrier grid.
- `'bwp'` — Indices are referenced with respect to the BWP.

Data Types: `char` | `string`

Output Arguments

ind — PUCCH DM-RS RE indices
column vector | *M*-by-3 matrix

PUCCH DM-RS RE indices, returned as one of these values.

- Column vector — The function returns this type of value when you set the `IndexStyle` name-value argument to `'index'`.
- *M*-by-3 matrix — The function returns this type of value when you set the `IndexStyle` name-value argument to `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and number of antennas, respectively.

Depending on the value of `IndexBase`, the function returns either 1-based or 0-based indices. Depending on the value of `IndexOrientation`, the function returns either carrier-oriented indices or BWP-oriented indices.

For PUCCH format 0, this output is empty.

Data Types: `uint32`

Version History

Introduced in R2021a

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPUCCHIndices` | `nrPUCCHDMRS` | `nrPUCCH`

Objects
`nrPUCCH0Config` | `nrPUCCH1Config` | `nrPUCCH2Config` | `nrPUCCH3Config` | `nrPUCCH4Config`

nrPUCCHHoppingInfo

Get PUCCH hopping information

Syntax

`info = nrPUCCHHoppingInfo(cp,nslot,nid,groupHopping,initialCS,seqCS)`

Description

`info = nrPUCCHHoppingInfo(cp,nslot,nid,groupHopping,initialCS,seqCS)` returns PUCCH sequence and cyclic shift hopping information. The function assumes intra-slot frequency hopping is enabled. The input arguments are:

- Cyclic prefix `cp`
- Radio frame slot number `nslot`
- Scrambling identity `nid`
- Group hopping configuration `groupHopping`
- Initial cyclic shift `initialCS`
- Sequence cyclic shift `seqCS`

Examples

Get PUCCH Hopping Information

Get PUCCH hopping information for the specified input arguments.

```
cp = 'normal';
nslot = 3;
nid = 512;
groupHopping = 'enable';
initialCS = 5;
seqCS = 0;
info = nrPUCCHHoppingInfo(cp,nslot,nid,groupHopping,initialCS,seqCS)
```

```
info = struct with fields:
          U: [13 22]
          V: [0 0]
      Alpha: [2.0944 2.0944 0 5.7596 2.6180 3.6652 4.1888 5.7596 1.5708 ... ]
        FGH: [11 20]
        FSS: 2
    Hopping: 'groupHopping'
        NCS: [239 107 223 6 24 2 3 66 238 125 209 145 44 233]
```

The output field `Alpha` provides cyclic shifts corresponding to all the symbols in a slot. Since symbol indices are zero-based, to obtain the cyclic shift value corresponding to a symbol index, you must increase the index value.

```
symInd = 0;
alpha = info.Alpha(symInd+1)
```

```
alpha = 2.0944
```

Get PUCCH Hopping Parameters When Intra-Slot Frequency Hopping Is Disabled

Get PUCCH hopping information for the specified input arguments.

```
cp = 'extended';
nslot = 7;
nid = 12;
groupHopping = 'enable';
initialCS = 9;
seqCS = 0;
info = nrPUCCHHoppingInfo(cp,nslot,nid,groupHopping,initialCS,seqCS)
```

```
info = struct with fields:
          U: [20 4]
          V: [0 0]
      Alpha: [1.0472 0 1.0472 0 5.7596 3.1416 5.2360 4.1888 1.5708 ... ]
        FGH: [8 22]
        FSS: 12
    Hopping: 'groupHopping'
        NCS: [149 255 173 255 146 141 25 167 198 12 63 78]
```

To obtain the base sequence group number and base sequence number when intra-slot frequency hopping is disabled, consider only the first elements of `U` and `V`.

```
u = info.U(1)
```

```
u = 20
```

```
v = info.V(1)
```

```
v = 0
```

Input Arguments

cp — Cyclic prefix length
`'normal'` | `'extended'`

Cyclic prefix length, specified as one of these options:

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length only applies for 60 kHz subcarrier spacing.

Data Types: `char` | `string`

`nslot` — Radio frame slot number
integer from 0 to 159

Radio frame slot number, specified as an integer from 0 to 159. For normal cyclic prefix of different numerologies, the valid range is from 0 to 159. For extended cyclic prefix, the valid range is from 0 to 39. For more details, see TS 38.211 Section 4.3.2 [1].

Data Types: `double`

`nid` — Scrambling identity
integer from 0 to 1023

Scrambling identity, specified as an integer from 0 to 1023. `nid` is higher layer parameter *hoppingId*, ranging from 0 to 1023, if the higher layer parameter is configured. Otherwise, `nid` is the physical layer cell identity number *NCellID*, ranging from 0 to 1007. For more information on these values, see TS 38.211 Section 6.3.2.2.1.

Data Types: `double`

`groupHopping` — Group hopping configuration
`'neither'` | `'enable'` | `'disable'`

Group hopping configuration, specified as `'neither'`, `'enable'`, or `'disable'`. The `groupHopping` argument is higher layer parameter *pucch-GroupHopping*.

Data Types: `char` | `string`

`initialCS` — Initial cyclic shift
integer from 0 to 11

Initial cyclic shift, referred to as *m_0* in TS 38.211 Section 6.3.2.2.2, specified as an integer from 0 to 11. For PUCCH formats 0 and 1, `initialCS` is higher layer parameter *initialCyclicShift*. For PUCCH format 3 demodulation reference signals (DMRS), `initialCS` must be 0. For PUCCH format 4 DMRS, `initialCS` must be 0, 3, 6, or 9. For more information, see TS 38.213 Section 9.2.1 [2].

Data Types: `double`

`seqCS` — Sequence cyclic shift
integer from 0 to 11

Sequence cyclic shift, referred to as *m_cs* in TS 38.211 Section 6.3.2.2.2, specified as an integer from 0 to 11. For PUCCH formats 1, 2, 3, and 4, `seqCS` must be 0.

Data Types: `double`

Output Arguments

`info` — Sequence and cyclic shift hopping information
structure

Sequence and cyclic shift hopping information, returned as a structure that contains these fields:

Parameter Field	Values	Description
U	1-by-2 integer vector	Base sequence group numbers, returned as a 1-by-2 integer vector with element values from 0 to 29. The first vector element corresponds to the first hop in a slot. The second vector element corresponds to the second hop in a slot.
V	1-by-2 logical vector	Base sequence numbers, returned a 1-by-2 logical vector. The first vector element corresponds to the first hop in a slot. The second vector element corresponds to the second hop in a slot.
Alpha	1-by-14 integer vector, 1-by-12 integer vector	Cyclic shifts of all symbols in a slot, returned as a 1-by-14 integer vector (for normal cyclic prefix) or 1-by-12 integer vector (for extended cyclic prefix). The first vector element corresponds to the first hop in a slot. The second vector element corresponds to the second hop in a slot.
FGH	1-by-2 integer vector	Sequence-group hopping pattern, returned as 1-by-2 integer vector with values from 0 to 29. The first vector element corresponds to the first hop in a slot. The second vector element corresponds to the second hop in a slot.
FSS	nonnegative integer	Sequence-group shift offset, returned as a nonnegative integer from 0 to 29.
Hopping	`'neither'`, `'groupHopping'`, `'sequenceHopping'`	Hopping configuration, returned as `'neither'`, `'groupHopping'`, or `'sequenceHopping'`. The hopping configuration is based on the input argument `groupHopping`.
NCS	1-by-14 integer vector, 1-by-12 integer vector	Hopping identity of cyclic shifts, referred to as *n_cs* in TS 38.211 Section 6.3.2.2.2, returned as a 1-by-14 integer vector (for normal cyclic prefix) or 1-by-12 integer vector (for extended cyclic prefix). A vector element at position *i* corresponds to the hopping identity of cyclic shift at symbol position *i* in a slot.

Data Types: `struct`

Version History

Introduced in R2019a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.213. "NR; Physical layer procedures for control." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrPUCCH0` | `nrPUCCH1` | `nrPUCCH2` | `nrPUCCH3` | `nrPUCCH4`

nrPUCCHIndices

Generate PUCCH resource element indices

Syntax

```
[ind,info] = nrPUCCHIndices(carrier,pucch)
[ind,info] = nrPUCCHIndices(carrier,pucch,Name,Value)
```

Description

`[ind,info] = nrPUCCHIndices(carrier,pucch)` generates `ind`, which contains 1-based physical uplink control channel (PUCCH) resource element (RE) indices, as defined in TS 38.211 Section 6.3.2 [1] for all PUCCH formats. `carrier` specifies the carrier configuration. `pucch` specifies the PUCCH configuration. The function also generates `info`, which contains information about the bit capacity and symbol capacity of the uplink control information (UCI) on the PUCCH.

`[ind,info] = nrPUCCHIndices(carrier,pucch,Name,Value)` specifies output formatting options using one or more name-value arguments.

Examples

Generate PUCCH Format 0 Indices

Create a default carrier configuration object. This object corresponds to a 10 MHz carrier with 15 kHz subcarrier spacing.

```
carrier = nrCarrierConfig;
```

Create a default PUCCH format 0 configuration object.

```
pucch0 = nrPUCCH0Config;
```

Specify the size of the bandwidth part (BWP) as 35 and the starting PRB index of the BWP relative to CRB 0 as 12. Specify the first symbol index in the PUCCH transmission slot as 3 and the number of allocated PUCCH symbols as 2. Set the PRB allocated for the PUCCH to 20, enable intraslot frequency hopping, and set the starting PRB index of the second hop to 10.

```
pucch0.NSizeBWP = 35;
pucch0.NStartBWP = 12;
pucch0.SymbolAllocation = [3 2];
pucch0.PRBSet = 20;
pucch0.FrequencyHopping = 'intraSlot';
pucch0.SecondHopStartPRB = 10;
```

Generate PUCCH format 0 RE indices, setting the index orientation with respect to the carrier grid.

```
[ind,info] = nrPUCCHIndices(carrier,pucch0,'IndexOrientation','carrier')
```

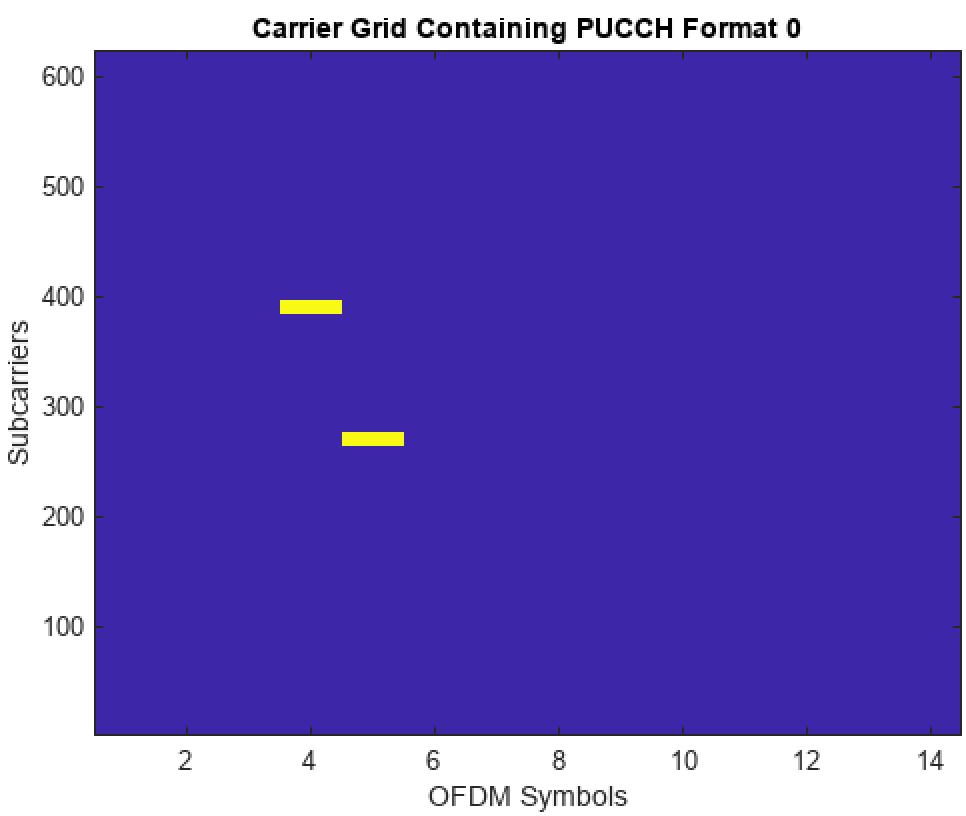
```
ind = 24x1 uint32 column vector

   2257
```

```
    2258
    2259
    2260
    2261
    2262
    2263
    2264
    2265
    2266
       ⋮
```

```
info = struct with fields:
                G: 24
               Gd: 24
        NREPerPRB: 24
    DMRSSymbolSet: [1x0 double]
```

Plot the generated REs for PUCCH format 0 in the carrier resource grid.

```
resGrid = nrResourceGrid(carrier);
resGrid(ind) = 1;
imagesc(resGrid)
axis xy
xlabel('OFDM Symbols')
ylabel('Subcarriers')
title('Carrier Grid Containing PUCCH Format 0')
```



Generate PUCCH Format 1 Indices

Create a default carrier configuration object.

```
carrier = nrCarrierConfig;
```

Specify a 60 kHz carrier with extended cyclic prefix. Set the number of RBs in the carrier resource grid to 55 and the slot number to 12.

```
carrier.SubcarrierSpacing = 60;
carrier.CyclicPrefix = 'extended';
carrier.NSizeGrid = 55;
carrier.NSlot = 12;
```

Create a default PUCCH format 1 configuration object.

```
pucch1 = nrPUCCH1Config;
```

Specify the first symbol index in the PUCCH transmission slot as 3 and the number of allocated PUCCH symbols as 8. Enable intraslot frequency hopping and set the starting PRB index of the second hop to 35.

```
pucch1.NSizeBWP = [];                  % Set NSizeBWP equal to the NSizeGrid property of carrier
pucch1.NStartBWP = [];                 % Set NStartBWP equal to the NStartGrid property of carrie
pucch1.SymbolAllocation = [3 8];
pucch1.FrequencyHopping = 'intraSlot';
pucch1.SecondHopStartPRB = 35;
```

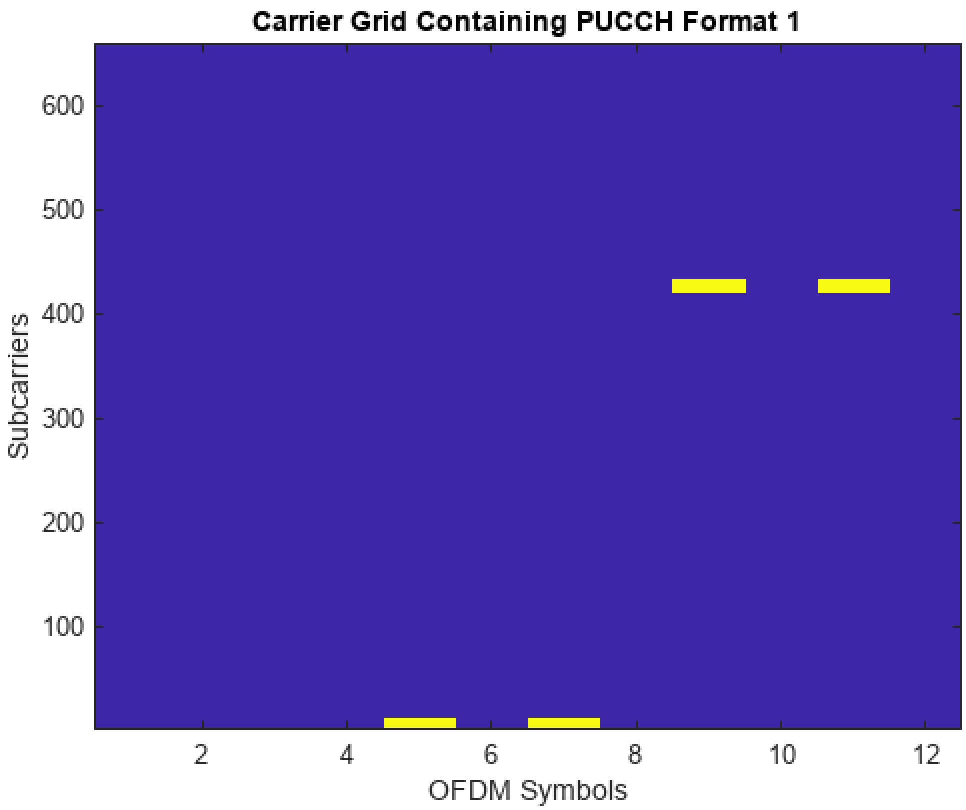
Generate PUCCH format 1 RE indices.

```
[ind,info] = nrPUCCHIndices(carrier,pucch1)
```

```
ind = 48x1 uint32 column vector

   2641
   2642
   2643
   2644
   2645
   2646
   2647
   2648
   2649
   2650
      ⋮

info = struct with fields:
                G: 48
               Gd: 48
         NREPerPRB: 48
    DMRSSymbolSet: [3 5 7 9]
```

Plot the generated REs for PUCCH format 1 in the carrier resource grid.

```
resGrid = nrResourceGrid(carrier);
resGrid(ind) = 1;
imagesc(resGrid)
axis xy
xlabel('OFDM Symbols')
ylabel('Subcarriers')
title('Carrier Grid Containing PUCCH Format 1')
```



Generate PUCCH Format 2 Indices

Create a default carrier configuration object.

```
carrier = nrCarrierConfig;
```

Specify the number of RBs in the carrier resource grid as 24, the start of the carrier resource grid relative to CRB 0 as 20, and the slot number as 43.

```
carrier.NSizeGrid = 24;
carrier.NStartGrid = 20;
carrier.NSlot = 43;
```

Create a default PUCCH format 2 configuration object.

```
pucch2 = nrPUCCH2Config;
```

Specify the first symbol index in the PUCCH transmission slot as 5 and the number of allocated PUCCH symbols as 2. Set the PRBs allocated for the PUCCH to range from 0 to 15 and enable intraslot frequency hopping. Set the starting PRB index of the second hop to 7.

```
pucch2.SymbolAllocation = [5 2];
pucch2.PRBSet = 0:15;
pucch2.FrequencyHopping = 'intraSlot';
pucch2.SecondHopStartPRB = 7;
```

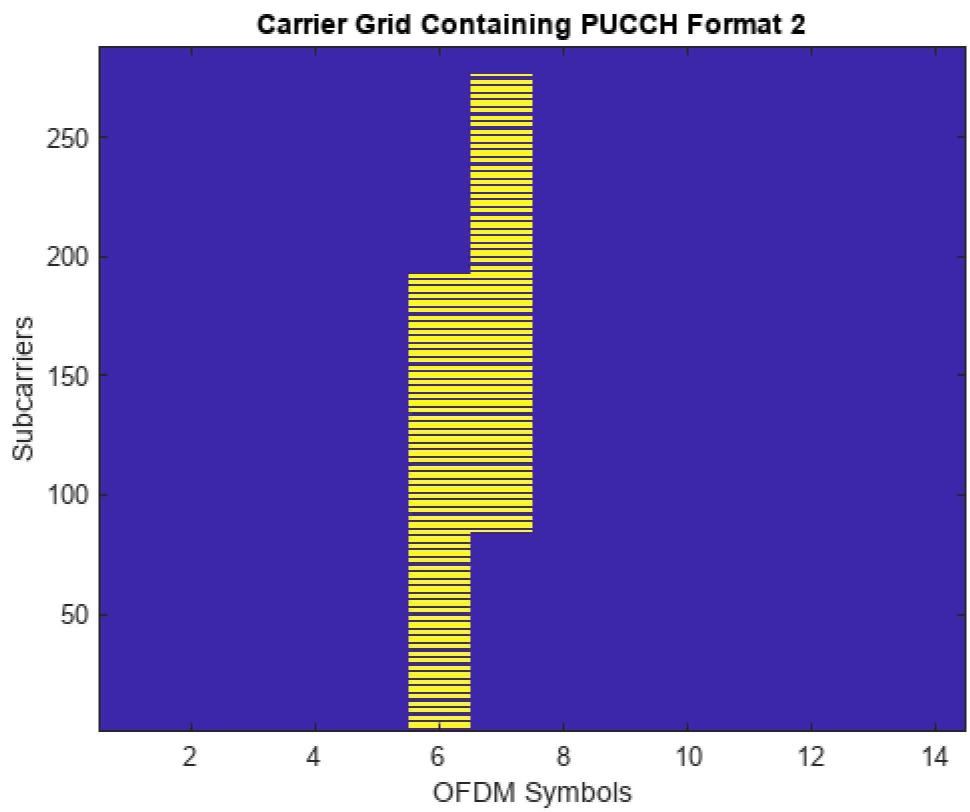
Generate PUCCH format 2 RE indices.

```
[ind,info] = nrPUCCHIndices(carrier,pucch2)
```

```
ind = 256x1 uint32 column vector

   1441
   1443
   1444
   1446
   1447
   1449
   1450
   1452
   1453
   1455
      ⋮

info = struct with fields:
                G: 512
               Gd: 256
         NREPerPRB: 16
     DMRSSymbolSet: [5 6]
```

Plot the generated REs for PUCCH format 2 in the carrier resource grid.

```
resGrid = nrResourceGrid(carrier);
resGrid(ind) = 1;
imagesc(resGrid)
axis xy
xlabel('OFDM Symbols')
ylabel('Subcarriers')
title('Carrier Grid Containing PUCCH Format 2')
```



Generate PUCCH Format 3 Indices

Create a default carrier configuration object.

```
carrier = nrCarrierConfig;
```

Create a default PUCCH format 3 configuration object.

```
pucch3 = nrPUCCH3Config;
```

Specify the PRB allocation of the PUCCH to range from 0 to 11. Set the first symbol index in the PUCCH transmission slot to 2 and the number of allocated PUCCH symbols to 12. Enable the additional DM-RS configuration flag.

```
pucch3.NSizeBWP = [];
pucch3.NStartBWP = [];
pucch3.PRBSet = 0:11;
pucch3.SymbolAllocation = [2 12];
pucch3.AdditionalDMRS = 1;
```

Generate PUCCH format 3 RE indices, setting the index orientation with respect to the carrier grid.

```
[ind,info] = nrPUCCHIndices(carrier,pucch3,'IndexOrientation','carrier')
```

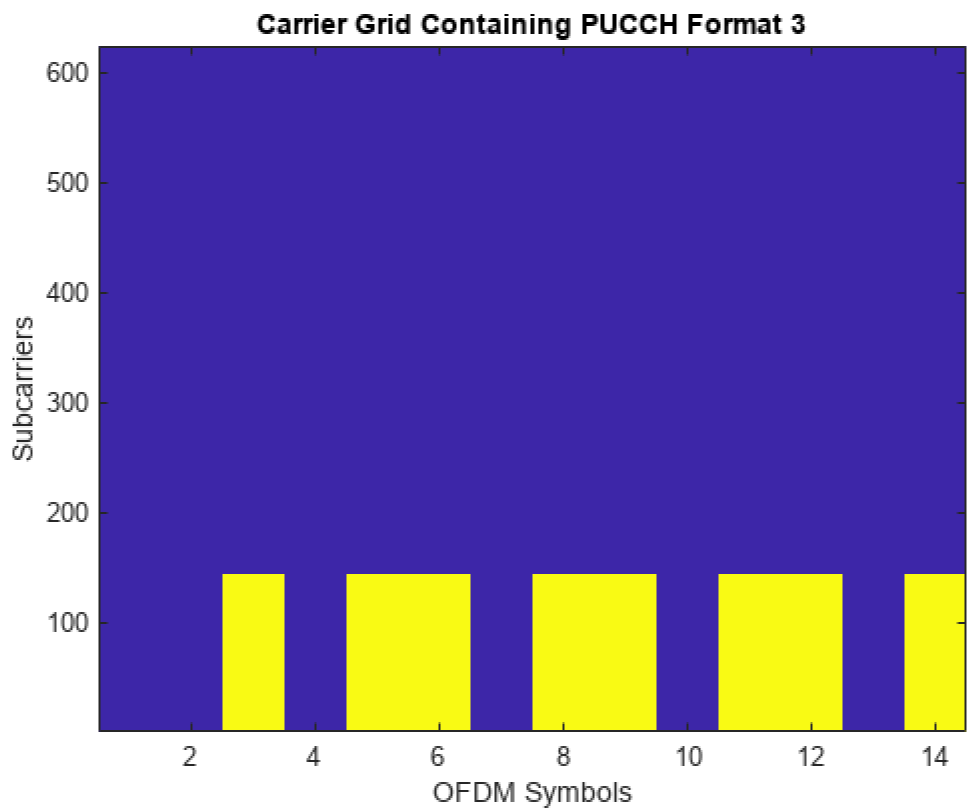
```
ind = 1152x1 uint32 column vector
```

```
    1249
    1250
    1251
    1252
    1253
    1254
    1255
    1256
    1257
    1258
       ⋮
```

```
info = struct with fields:
                G: 2304
               Gd: 1152
        NREPerPRB: 96
    DMRSSymbolSet: [3 6 9 12]
```

Plot the generated REs for PUCCH format 3 in the carrier resource grid.

```
resGrid = nrResourceGrid(carrier);
resGrid(ind) = 1;
imagesc(resGrid)
axis xy
xlabel('OFDM Symbols')
ylabel('Subcarriers')
title('Carrier Grid Containing PUCCH Format 3')
```



Generate PUCCH Format 4 Indices

Create a default carrier configuration object.

```
carrier = nrCarrierConfig;
```

Create a default PUCCH format 4 configuration object.

```
pucch4 = nrPUCCH4Config;
```

Specify the modulation scheme as pi/2-BPSK and the allocated PRB for the PUCCH as 50. Set the first symbol index in the PUCCH transmission slot to 2 and the number of allocated PUCCH symbols to 10. Enable intraslot frequency hopping and the additional DM-RS configuration flag. Set the starting PRB index of the second hop to 10.

```
pucch4.Modulation = 'pi/2-BPSK';
pucch4.PRBSet = 50;
pucch4.SymbolAllocation = [2 10];
pucch4.FrequencyHopping = 'intraSlot';
pucch4.SecondHopStartPRB = 10;
pucch4.AdditionalDMRS = 1;
```

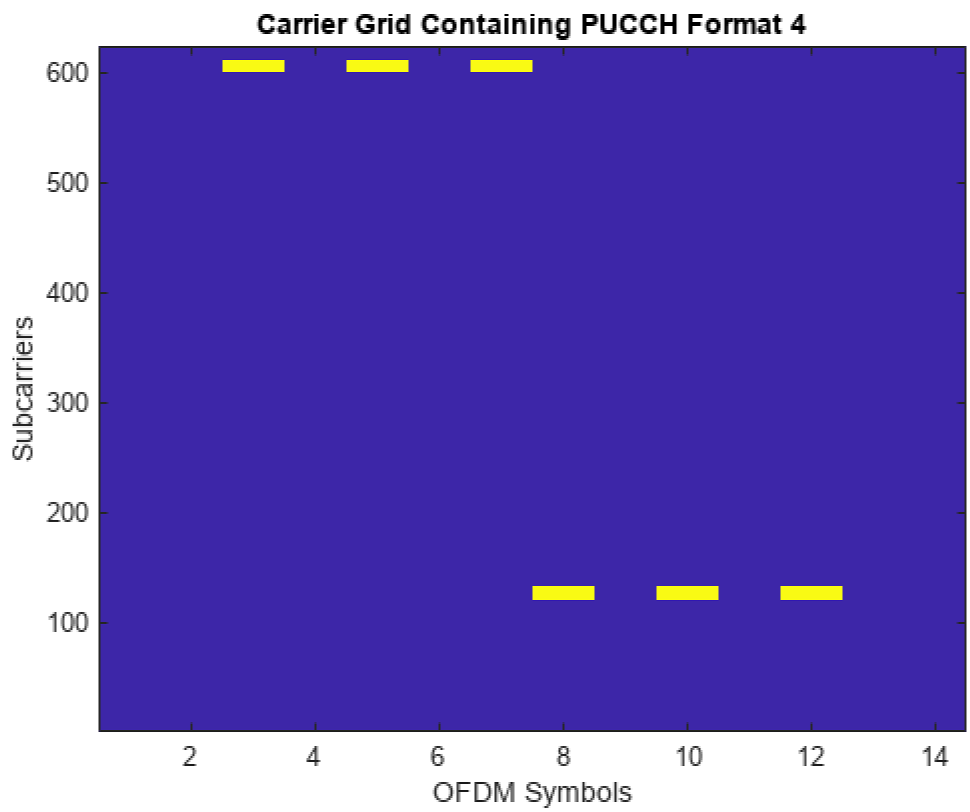
Generate the PUCCH format 4 RE indices.

```
[ind,info] = nrPUCCHIndices(carrier,pucch4)
```

```
ind = 72x1 uint32 column vector

   1849
   1850
   1851
   1852
   1853
   1854
   1855
   1856
   1857
   1858
      ⋮

info = struct with fields:
                G: 36
               Gd: 36
        NREPerPRB: 72
    DMRSSymbolSet: [3 5 8 10]
```

Plot the generated REs for PUCCH format 4 in the carrier resource grid.

```
resGrid = nrResourceGrid(carrier);
resGrid(ind) = 1;
imagesc(resGrid)
axis xy
xlabel('OFDM Symbols')
ylabel('Subcarriers')
title('Carrier Grid Containing PUCCH Format 4')
```



Input Arguments

carrier — Carrier configuration parameters
nrCarrierConfig object

Carrier configuration parameters for a specific OFDM numerology, specified as an nrCarrierConfig object. This function only uses these nrCarrierConfig object properties.

- NCellID
- SubcarrierSpacing
- CyclicPrefix
- NSizeGrid
- NStartGrid
- NSlot

pucch — PUCCH configuration parameters
nrPUCCH0Config object | nrPUCCH1Config object | nrPUCCH2Config object | nrPUCCH3Config object | nrPUCCH4Config object

PUCCH configuration parameters, specified as one of these options.

- For format 0, specify an nrPUCCH0Config object. The function uses only these object properties.

  - NSizeBWP
  - NStartBWP
  - SymbolAllocation
  - PRBSet
  - FrequencyHopping
  - SecondHopStartPRB
- For format 1, specify an nrPUCCH1Config object. The function uses only these object properties.
  - NSizeBWP
  - NStartBWP
  - SymbolAllocation
  - PRBSet
  - FrequencyHopping
  - SecondHopStartPRB
- For format 2, specify an nrPUCCH2Config object. The function uses only these object properties.
  - NSizeBWP
  - NStartBWP
  - SymbolAllocation
  - PRBSet
  - FrequencyHopping
  - SecondHopStartPRB
- For format 3, specify an nrPUCCH3Config object. The function uses only these object properties.
  - NSizeBWP
  - NStartBWP
  - Modulation
  - SymbolAllocation
  - PRBSet
  - FrequencyHopping
  - SecondHopStartPRB
  - AdditionalDMRS
- For format 4, specify an nrPUCCH4Config object. The function uses only these object properties.
  - NSizeBWP
  - NStartBWP
  - Modulation
  - SymbolAllocation
  - PRBSet
  - FrequencyHopping
  - SecondHopStartPRB
  - AdditionalDMRS

Name-Value Pair Arguments

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

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

Example: `'IndexStyle','subscript'` specifies the RE indexing form of the output.

`IndexStyle` — RE indexing form
`'index'` (default) | `'subscript'`

RE indexing form, specified as one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

`IndexBase` — RE indexing base
`'1based'` (default) | `'0based'`

RE indexing base, specified as one of these values:

- `'1based'` — The index counting starts from 1.
- `'0based'` — The index counting starts from 0.

Data Types: `char` | `string`

`IndexOrientation` — RE indexing orientation
`'carrier'` (default) | `'bwp'`

RE indexing orientation, specified as one of these values:

- `'carrier'` — Indices are referenced with respect to the carrier grid.
- `'bwp'` — Indices are referenced with respect to the BWP.

Data Types: `char` | `string`

Output Arguments

`ind` — PUCCH RE indices
column vector | *M*-by-3 matrix

PUCCH RE indices, returned as one of these values:

- Column vector — The function returns this type of value when you set the `IndexStyle` name-value argument to `'index'`.
- *M*-by-3 matrix — The function returns this type of value when you set the `IndexStyle` name-value argument to `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and number of antennas, respectively.

Depending on the value of `IndexBase`, the function returns either 1-based or 0-based indices. Depending on the value of `IndexOrientation`, the function returns either carrier-oriented indices or BWP-oriented indices.

Data Types: `uint32`

`info` — PUCCH resource information
structure

PUCCH resource information, returned as a structure containing these fields. The output data type of each field is `double`.

Field	Description
`G`	Bit capacity of the PUCCH, returned as a value that is equal to the length of the UCI encoded codeword for formats 2, 3, and 4.
`Gd`	Symbol capacity of the PUCCH
`NREPerPRB`	Number of REs per PRB allocated to the PUCCH (including the spreading factor)
`DMRSSymbolSet`	Set of 0-based OFDM symbol locations in a slot containing the demodulation reference signal (DM-RS)

Version History

Introduced in R2021a

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrPUCCH` | `nrPUCCHDMRS` | `nrPUCCHDMRSIndices`

Objects

`nrPUCCH0Config` | `nrPUCCH1Config` | `nrPUCCH2Config` | `nrPUCCH3Config` | `nrPUCCH4Config`

nrPUCCHPRBS

Generate PUCCH scrambling sequence

Syntax

```
[seq,cinit] = nrPUCCHPRBS(nid,rnti,n)
[seq,cinit] = nrPUCCHPRBS(nid,rnti,n,Name,Value)
```

Description

`[seq,cinit] = nrPUCCHPRBS(nid,rnti,n)` returns the first `n` elements of the physical uplink control channel (PUCCH) scrambling sequence. The function also returns initialization value `cinit` of the pseudorandom binary sequence (PRBS) generator. The initialization value depends on scrambling identity `nid` and radio network temporary identifier (RNTI) of the user equipment (UE) `rnti`. The function implements TS 38.211 Section 6.3.2.5.1/6.3.2.6.1 [1].

`[seq,cinit] = nrPUCCHPRBS(nid,rnti,n,Name,Value)` specifies additional output formatting options by using one or more name-value pair arguments. Unspecified name-value pairs take their default values.

Examples

Generate PUCCH Scrambling Sequence

Generate the first 300 elements of the PUCCH scrambling sequence when initialized with the specified physical layer cell identity number and RNTI.

```
ncellid = 17;
rnti = 120;
n = 300;
seq = nrPUCCHPRBS(ncellid,rnti,n)
```

```
seq = 300x1 logical array

   0
   1
   1
   0
   1
   1
   0
   1
   0
   0
      ⋮
```

Input Arguments

nid — Scrambling identity
integer from 0 to 1023

Scrambling identity, specified as an integer from 0 to 1023. `nid` is higher layer parameter *dataScramblingIdentityPUSCH*, ranging from 0 to 1023, if the higher layer parameter is configured. Otherwise, `nid` is the physical layer cell identity number *NCellID*, ranging from 0 to 1007.

For more information, see TS 38.211 Sections 6.3.2.5.1 and 6.3.2.6.1.

Data Types: `double`

rnti — RNTI of UE
integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

n — Number of elements in output sequence
nonnegative integer

Number of elements in output sequence, specified as a nonnegative integer.

Data Types: `double`

Name-Value Pair Arguments

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

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

Example: `'MappingType','signed'` specifies nondefault sequence formatting.

MappingType — Output sequence formatting
`'binary'` (default) | `'signed'`

Output sequence formatting, specified as the comma-separated pair consisting of `'MappingType'` and one of these values:

- `'binary'` — This value maps `true` to `1` and `false` to `0`. The data type of the output sequence is `logical`.
- `'signed'` — This value maps `true` to `–1` and `false` to `1`. The data type of the output sequence is `double`. To specify `single` data type, use the `'OutputDataType'` name-value pair.

Data Types: `char` | `string`

OutputDataType — Data type of output sequence
`'double'` (default) | `'single'`

Data type of output sequence, specified as the comma-separated pair consisting of `'OutputDataType'` and `'double'` or `'single'`. This name-value pair applies only when `'MappingType'` is set to `'signed'`.

Data Types: `char` | `string`

Output Arguments

`seq` — PUCCH scrambling sequence
logical column vector | numeric column vector

PUCCH scrambling sequence, returned as a logical or numeric column vector. `seq` contains the first `n` elements of the PUCCH scrambling sequence. If you set `'MappingType'` to `'signed'`, the output data type is either `double` or `single`. If you set `'MappingType'` to `'binary'`, the output data type is `logical`.

Data Types: `double` | `single` | `logical`

`cinit` — Initialization value for PRBS generator
nonnegative integer

Initialization value for PRBS generator, returned as a nonnegative integer.

Data Types: `double`

Version History

Introduced in R2019a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying `'single'` as the output data type, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPUCCH2` | `nrPUCCH3` | `nrPUCCH4` | `nrPRBS`

nrPUSCH

Generate PUSCH modulation symbols

Syntax

```
[sym,ptrsSym] = nrPUSCH(cw,mod,nLayers,nid,rnti)
[sym,ptrsSym] = nrPUSCH( ___ ,transformPrecode,mrb)
[sym,ptrsSym] = nrPUSCH( ___ ,txScheme,nPorts,tpmi)
[sym,ptrsSym] = nrPUSCH(carrier,pusch,cw)
[sym,ptrsSym] = nrPUSCH( ___ ,'OutputDataType',datatype)
```

Description

`[sym,ptrsSym] = nrPUSCH(cw,mod,nLayers,nid,rnti)` returns physical uplink shared channel (PUSCH) modulation symbols, as defined in TS 38.211 Sections 6.3.1.1 to 6.3.1.5 [1]. The process consists of scrambling with scrambling identity `nid`, performing symbol modulation with modulation scheme `mod`, and layer mapping. `cw` specifies an uplink shared channel (UL-SCH) codeword, as described in TS 38.212 Section 6.2.7 [2]. `nLayers` specifies the number of transmission layers. `rnti` is the radio network temporary identifier (RNTI) of the user equipment (UE). When you use this syntax, the output `ptrsSym` is empty.

`[sym,ptrsSym] = nrPUSCH( ___ ,transformPrecode,mrb)` specifies transform precoding as a logical value in addition to the input arguments in the first syntax. When `transformPrecode` is set to `true`, the function applies the transform precoding defined in TS 38.211 Section 6.3.1.4. `mrb` specifies the allocated number of PUSCH resource blocks. When you use this syntax, the output `ptrsSym` is empty.

`[sym,ptrsSym] = nrPUSCH( ___ ,txScheme,nPorts,tpmi)` specifies the transmission scheme in addition to the input arguments in the second syntax. When `txScheme` is set to `'codebook'`, the function performs multi-input multi-output (MIMO) precoding based on the specified number of layers `nLayers`, number of antenna ports `nPorts`, and the transmitted precoding matrix indicator (TPMI) `tpmi`. When you use this syntax, the output `ptrsSym` is empty.

`[sym,ptrsSym] = nrPUSCH(carrier,pusch,cw)` returns PUSCH modulation symbols, `sym`, for the specified carrier configuration `carrier` and PUSCH configuration `pusch`. The input `cw` specifies the UL-SCH codeword. The function also returns the precoded phase tracking reference signals (PT-RS) symbols, `ptrsSym`, which are mapped to the resource grid. When you use this syntax with transform precoding, the function maps the data modulated symbols and PT-RS symbols at appropriate locations prior to starting the transform precoding process.

`[sym,ptrsSym] = nrPUSCH( ___ ,'OutputDataType',datatype)` specifies the data type for the PUSCH symbols and PT-RS symbols in addition to specifying an input combination from any of the previous syntaxes.

Examples

Generate PUSCH Modulation Symbols

Specify a random sequence of binary values corresponding to a codeword of 8064 bits.

```
cw = randi([0 1],8064,1);
```

Using 16-QAM modulation, generate PUSCH modulation symbols for the specified physical layer cell identity number, RNTI, and two transmission layers. By default, the function disables transform precoding and noncodebook-based transmission.

```
modulation = '16QAM';
nlayers = 2;
ncellid = 17;
rnti = 111;
sym = nrPUSCH(cw,modulation,nlayers,ncellid,rnti)
```

```
sym = 1008×2 complex

  -0.9487 - 0.9487i  -0.3162 + 0.3162i
   0.3162 + 0.3162i  -0.9487 - 0.3162i
   0.3162 + 0.3162i   0.3162 - 0.3162i
   0.9487 - 0.3162i  -0.3162 + 0.9487i
  -0.3162 - 0.9487i   0.3162 - 0.9487i
  -0.3162 + 0.9487i   0.3162 - 0.3162i
   0.3162 + 0.3162i   0.9487 - 0.9487i
  -0.9487 + 0.9487i  -0.3162 + 0.3162i
   0.9487 - 0.9487i  -0.9487 - 0.3162i
  -0.9487 - 0.9487i   0.3162 + 0.9487i
      ⋮
```

Generate PUSCH Symbols Using Codebook-Based Transmission

Specify a random sequence of binary values corresponding to a codeword of 8064 bits.

```
cw = randi([0 1],8064,1);
```

Using 256-QAM modulation, generate PUSCH modulation symbols for the specified physical layer cell identity number, RNTI, bandwidth, and one transmission layer. Enable transform precoding and codebook-based transmission based on the specified TPMI and four antennas.

```
modulation = '256QAM';
ncellid = 17;
rnti = 111;
mrb = 6;
nlayers = 1;
transformPrecode = true;
txScheme = 'codebook';
tpmi = 1;
nports = 4;
sym = nrPUSCH(cw,modulation,nlayers,ncellid,rnti,transformPrecode,mrb,txScheme,nports,tpmi)
```

```
sym = 1008×4 complex

   0.0000 + 0.0000i   0.2169 + 0.2350i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.2296 + 0.3713i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.0797 - 0.9008i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.4767 - 0.0143i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.4124 + 0.2638i   0.0000 + 0.0000i   0.0000 + 0.0000i
```

```
   0.0000 + 0.0000i  -0.1433 - 0.2366i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0885 - 0.1080i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.5507 - 0.1894i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.3039 - 0.9165i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.1498 + 0.3356i   0.0000 + 0.0000i   0.0000 + 0.0000i
      ⋮
```

Generate PUSCH Symbols and Indices

Create a carrier configuration object with default properties. This object corresponds to 30 kHz of subcarrier spacing and 20 MHz transmission bandwidth.

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 30;
carrier.NSizeGrid = 51;
```

Create a PUSCH configuration object with specified properties. When transform precoding is 1, the waveform type is discrete fourier transform spread orthogonal frequency division multiplexing (DFT-s-OFDM).

```
pusch = nrPUSCHConfig;
pusch.NStartBWP = 10;
pusch.NSizeBWP = 41;
pusch.Modulation = '16QAM';
pusch.NID = []; % Set NID equal to the NCellID property of carrier.
pusch.PRBSet = 0:5;
pusch.TransformPrecoding = 1;
pusch.FrequencyHopping = 'intraSlot';
pusch.SecondHopStartPRB = 3;
```

Generate PUSCH indices, setting the index orientation with respect to the carrier grid.

```
[ind,info] = nrPUSCHIndices(carrier,pusch,'IndexOrientation','carrier')
```

```
ind = 864x1 uint32 column vector

   121
   122
   123
   124
   125
   126
   127
   128
   129
   130
      ⋮

info = struct with fields:
                G: 3456
               Gd: 864
        NREPerPRB: 144
    DMRSSymbolSet: [2 7]
```

```
    PTRSSymbolSet: [1x0 double]
```

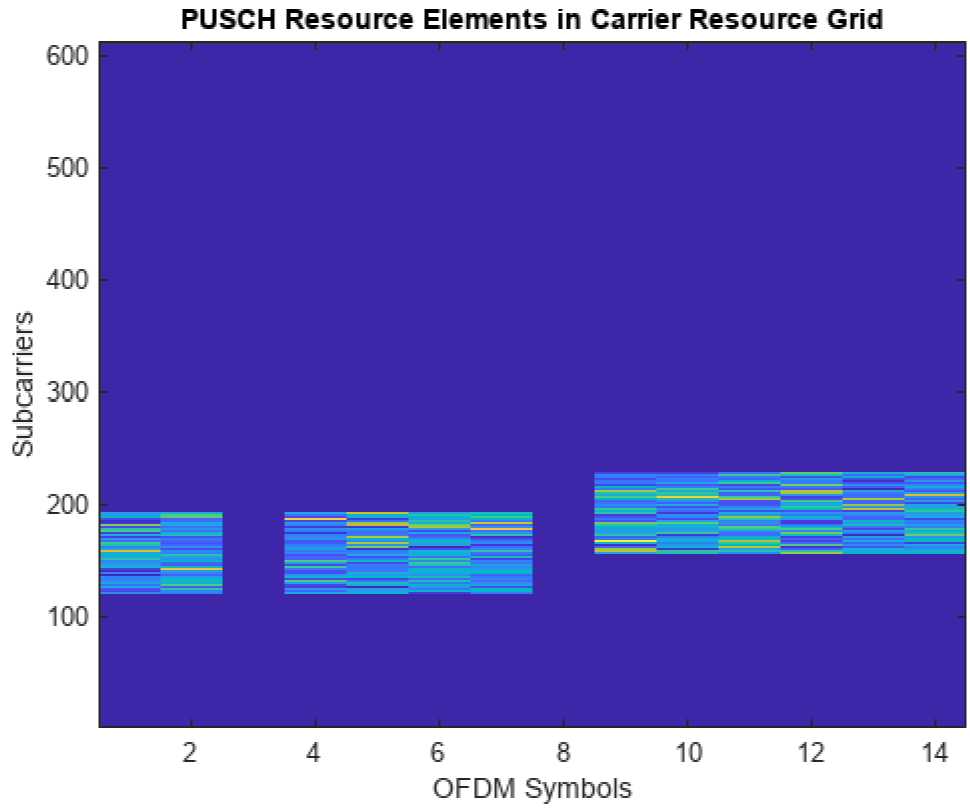
Generate PUSCH symbols of data type single.

```
numDataBits = info.G;
cws = randi([0 1],numDataBits,1);
sym = nrPUSCH(carrier,pusch,cws,'OutputDataType','single')
```

```
sym = 864x1 single column vector

  -0.7454 + 0.2981i
   0.3406 - 0.2312i
  -0.1153 + 0.2756i
   1.1921 - 0.3658i
  -0.3968 - 0.0277i
  -0.8788 - 0.6493i
  -0.8737 + 0.8318i
  -0.5764 + 0.0269i
  -1.6638 + 0.0482i
  -1.0270 - 0.1347i
      ⋮
```

Plot the generated symbols and indices on the carrier resource grid.

```
grid = complex(zeros([carrier.NSizeGrid*12 carrier.SymbolsPerSlot pusch.NumLayers]));
grid(ind) = sym;
imagesc(abs(grid(:,:,1)));
axis xy;
xlabel('OFDM Symbols');
ylabel('Subcarriers');
title('PUSCH Resource Elements in Carrier Resource Grid');
```



Generate PUSCH symbols and PT-RS symbols

Specify a random sequence of binary values corresponding to a codeword of 4032 bits.

```
cw = randi([0 1],4032,1);
```

Create a carrier configuration object with 30 kHz subcarrier spacing and 20 MHz transmission bandwidth.

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 30;
carrier.NSizeGrid = 51;
```

Create a PUSCH configuration object with these specified properties.

```
pusch = nrPUSCHConfig;
pusch.NStartBWP = 10;
pusch.NSizeBWP = 41;
pusch.Modulation = '64QAM';
pusch.PRBSet = 0:5;
pusch.TransformPrecoding = 1;
pusch.FrequencyHopping = 'intraSlot';
pusch.EnablePTRS = 1;
```

Create a PUSCH phase tracking reference signal (PT-RS) configuration object with these specified properties.

```
ptrs = nrPUSCHPTRSConfig;
ptrs.TimeDensity = 2;
ptrs.NumPTRSSamples = 4;
ptrs.NumPTRSGroups = 8;
ptrs.NID = 750;
```

Assign the PUSCH PT-RS configuration object to the PTRS property of the PUSCH configuration object.

```
pusch.PTRS = ptrs;
```

Generate PUSCH PT-RS symbols.

```
[sym,ptrsSym] = nrPUSCH(carrier,pusch,cw)
```

```
sym = 864×1 complex

   0.0000 - 0.3273i
  -0.0879 - 0.3552i
   0.4931 + 0.1194i
  -1.4765 - 0.6074i
  -0.1208 + 0.1961i
   0.8119 - 0.3622i
  -0.4653 + 0.0519i
   0.7790 + 0.9679i
   1.2730 + 1.0205i
  -1.6337 + 0.6741i
      ⋮

ptrsSym = 192×1 complex

   0.0000 - 0.3273i
  -0.0879 - 0.3552i
   0.4931 + 0.1194i
  -1.4765 - 0.6074i
  -0.5758 + 0.1331i
   2.2422 + 0.5442i
  -1.1968 + 1.4190i
   0.7160 + 0.7333i
  -0.7518 - 0.5147i
   1.3432 - 0.0623i
      ⋮
```

Input Arguments

cw — UL-SCH codeword

column vector of integers from –2 to 1

UL-SCH codeword from TS 38.212 Section 6.2.7, specified as a column vector of integers from –2 to 1.

- 0 and 1 represent false and true bit values, respectively.
- –1 and –2 represent *x* and *y* placeholders in the uplink control information (UCI), respectively. For more details, see TS 38.212 Sections 5.3.3.1 and 5.3.3.2.

Data Types: `double` | `int8`

`mod` — Modulation scheme
`'pi/2-BPSK'` | `'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'`

Modulation scheme, specified as `'pi/2-BPSK'`, `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`. This modulation scheme determines the modulation type and number of bits used per modulation symbol.

Modulation Scheme	Number of Bits Per Symbol
`'pi/2-BPSK'`	1
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

Data Types: `char` | `string`

`nLayers` — Number of transmission layers
integer from 1 to 4

Number of transmission layers, specified as an integer from 1 to 4. For more information, see TS 38.211 Section 6.3.1.3.

Data Types: `double`

`nid` — Scrambling identity
integer from 0 to 1023

Scrambling identity, specified as an integer from 0 to 1023. `nid` is higher layer parameter *dataScramblingIdentityPUSCH*, ranging from 0 to 1023, if the higher layer parameter is configured. Otherwise, `nid` is physical layer cell identity number *NCellID*, ranging from 0 to 1007. For more information, see TS 38.211 Section 6.3.1.1.

Data Types: `double`

`rnti` — RNTI of UE
integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

`transformPrecode` — Transform precoding
`false` (default) | `true`

Transform precoding, specified as `false` or `true`. For more information, see TS 38.211 Section 6.3.1.4.

Data Types: `double` | `logical`

mrb — Number of allocated PUSCH resource blocks
integer from 1 to 275

Number of allocated PUSCH resource blocks, specified as an integer from 1 to 275. For more information, see TS 38.214 Section 6.1.2.

Data Types: double

txScheme — Transmission scheme
'nonCodebook' (default) | 'codebook'

Transmission scheme, specified as one of these values:

- 'nonCodebook' — Use this option to disable MIMO precoding.
- 'codebook' — Use this option for codebook-based transmission using MIMO precoding.

For more information, see TS 38.211 Section 6.3.1.4.

Data Types: char | string

tpmi — Transmitted precoding matrix indicator
integer from 0 to 27

Transmitted precoding matrix indicator, specified as an integer from 0 to 27. The valid range of tpmi depends on the specified number of transmission layers, nLayers, and number of antenna ports, nPorts. For more information, see TS 38.211 Tables 6.3.1.5-1 to 6.3.1.5-7.

Data Types: double

nPorts — Number of antenna ports
1 | 2 | 4

Number of antenna ports, specified as 1 , 2, or 4. For more information, see TS 38.211 Section 6.3.1.5.

Data Types: double

datatype — Data type of output symbols
'double' (default) | 'single'

Data type of the output symbols, specified as 'double' or 'single'.

Data Types: char | string

carrier — Carrier configuration parameters
nrCarrierConfig object

Carrier configuration parameters for a specific OFDM numerology, specified as an nrCarrierConfig object. This function uses only these properties of the nrCarrierConfig object.

- NCellID
- SubcarrierSpacing
- CyclicPrefix
- NSizeGrid
- NStartGrid

- NSlot

pusch — PUSCH configuration parameters
nrPUSCHConfig object

PUSCH configuration parameters for a specific OFDM numerology, specified as an nrPUSCHConfig object. This function uses only these properties of the nrPUSCHConfig object.

- NSizeBWP
- NStartBWP
- Modulation
- NumLayers
- MappingType
- SymbolAllocation
- PRBSet
- TransformPrecoding
- TransmissionScheme
- NumAntennaPorts
- TPMI
- FrequencyHopping
- SecondHopStartPRB
- NID
- RNTI
- NRAPID
- DMRS
- EnablePTRS
- PTRS

Output Arguments

sym — PUSCH modulation symbols
complex matrix

PUSCH modulation symbols, returned as a complex matrix. If you set the txScheme input to 'codebook', the number of matrix columns is nPorts. If you set the txScheme input to 'nonCodebook', the number of matrix columns is nLayers. When you enable transform precoding, the output sym represents all of the post transform precoding data modulated symbols and PT-RS symbols.

Data Types: single | double
Complex Number Support: Yes

ptrsSym — PT-RS symbols mapped to resource grid
complex matrix

PT-RS symbols mapped to the resource grid, returned as a complex matrix. When you enable transform precoding, the output ptrsSym is the subset of output sym, at the PT-RS locations prior to transform precoding process. The output `ptrsSym` returns an empty value when you do not specify the `pusch` input argument.

Data Types: `single` | `double`
Complex Number Support: Yes

Version History

Introduced in R2019a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.212. "NR; Multiplexing and channel coding." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The `datatype` input argument must be compile-time constant. For example, when specifying `'single'` as the output data type, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPUSCHDecode` | `nrPUSCHScramble` | `nrPUSCHCodebook` | `nrULSCHInfo`

Objects
`nrCarrierConfig` | `nrPUSCHConfig` | `nrPUSCHPTRSConfig` | `nrPUSCHDMRSConfig`

nrPUSCHCodebook

Generate PUSCH precoding matrix

Syntax

```
w = nrPUSCHCodebook(nLayers,nPorts,tpmi)
w = nrPUSCHCodebook( ___ ,transformPrecode)
```

Description

`w = nrPUSCHCodebook(nLayers,nPorts,tpmi)` returns the physical uplink shared channel (PUSCH) precoding matrix for codebook-based transmission. `nLayers` is the number of layers, `nPorts` is the number of antenna ports, and `tpmi` is the transmitted precoding matrix indicator (TPMI). By default, this function disables transform precoding. The returned matrix, `w`, is the transpose of the precoding matrix defined in TS 38.211 Section 6.3.1.5 [1]. The matrix orientation of `w` allows the precoding operation to be performed by matrix multiplication on the output of the `nrLayerMap` function and `w`.

`w = nrPUSCHCodebook( ___ ,transformPrecode)` specifies transform precoding as a logical value in addition to the input arguments in the previous syntax. When `transformPrecode` is set to `true`, the function applies the transform precoding defined in TS 38.211 Section 6.3.1.4 [1].

Examples

Apply PUSCH Precoding Codebook

Modulate a random sequence of binary values of length 600 by using 64-QAM modulation. Split the modulated symbols into two layers.

```
modulation = '64QAM';
nlayers = 2;
in = randi([0 1],600,1);
data = nrSymbolModulate(in,modulation);
y = nrLayerMap(data,nlayers);
```

Generate the PUSCH precoding matrix for four antennas, two layers, and the specified TPMI.

```
nports = 4;
tpmi = 7;
w = nrPUSCHCodebook(nlayers,nports,tpmi)
```

```
w = 2×4 complex

   0.5000 + 0.0000i   0.0000 + 0.0000i   0.5000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.5000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.5000i
```

Precode the layered modulation symbols by using the codebook matrix.

```
z = y * w
```

```
z = 50×4 complex

  -0.0772 - 0.3858i   0.3858 - 0.5401i  -0.0772 - 0.3858i   0.5401 + 0.3858i
  -0.3858 + 0.0772i  -0.5401 - 0.0772i  -0.3858 + 0.0772i   0.0772 - 0.5401i
  -0.5401 - 0.2315i  -0.2315 + 0.0772i  -0.5401 - 0.2315i  -0.0772 - 0.2315i
  -0.3858 + 0.2315i  -0.2315 - 0.0772i  -0.3858 + 0.2315i   0.0772 - 0.2315i
  -0.0772 - 0.3858i   0.5401 + 0.2315i  -0.0772 - 0.3858i  -0.2315 + 0.5401i
  -0.5401 + 0.5401i  -0.0772 + 0.2315i  -0.5401 + 0.5401i  -0.2315 - 0.0772i
  -0.3858 + 0.2315i  -0.0772 + 0.5401i  -0.3858 + 0.2315i  -0.5401 - 0.0772i
  -0.3858 + 0.5401i   0.5401 + 0.3858i  -0.3858 + 0.5401i  -0.3858 + 0.5401i
  -0.2315 + 0.0772i   0.2315 - 0.5401i  -0.2315 + 0.0772i   0.5401 + 0.2315i
  -0.2315 - 0.0772i   0.3858 - 0.3858i  -0.2315 - 0.0772i   0.3858 + 0.3858i
      ⋮
```

Input Arguments

nLayers — Number of transmission layers
integer from 1 to 4

Number of transmission layers, specified as an integer from 1 to 4. For more information, see TS 38.211 Section 6.3.1.3.

Data Types: double

nPorts — Number of antenna ports
1 | 2 | 4

Number of antenna ports, specified as 1 , 2, or 4. For more information, see TS 38.211 Section 6.3.1.5.

Data Types: double

tpmi — Transmitted precoding matrix indicator
integer from 0 to 27

Transmitted precoding matrix indicator, specified as an integer from 0 to 27. The valid range of tpmi depends on the specified number of transmission layers, nLayers, and number of antenna ports, nPorts. For more information, see TS 38.211 Tables 6.3.1.5-1 to 6.3.1.5-7.

Data Types: double

transformPrecode — Transform precoding
false (default) | true

Transform precoding, specified as false or true. For more information, see TS 38.211 Section 6.3.1.4.

Data Types: double | logical

Output Arguments

w — PUSCH precoding codebook
complex matrix

PUSCH precoding codebook, returned as a complex matrix of size `nLayers`-by-`nPorts`. If `nLayers` and `nPorts` are both `1`, then `w` is `1`. Otherwise, the function returns the transpose of the matrix selected from Tables 6.3.1.5-1 to 6.3.1.5-7 in TS 38.211 [1].

Data Types: `double`
Complex Number Support: Yes

Version History

Introduced in R2019a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrLayerMap` | `nrLayerDemap`

nrPUSCHDecode

Decode PUSCH modulation symbols

Syntax

```
[cw,symbols] = nrPUSCHDecode(sym,mod,nid,rnti)
[cw,symbols] = nrPUSCHDecode( ___ ,nVar)
[cw,symbols] = nrPUSCHDecode( ___ ,transformPrecode,mrb)
[cw,symbols] = nrPUSCHDecode( ___ ,txScheme,nLayers,tpmi)
[cw,symbols] = nrPUSCHDecode(carrier,pusch,sym,nVar)
[cw,symbols] = nrPUSCHDecode(carrier,pusch,tcr,tbs,oack,ocsi1,ocsi2,sym,nVar)
```

Description

`[cw,symbols] = nrPUSCHDecode(sym,mod,nid,rnti)` returns soft bits `cw` and constellation symbols `symbols` resulting from the inverse operation of physical uplink shared channel (PUSCH) processing from TS 38.211 Section 6.3.1 [1]. The decoding consists of layer demapping, demodulation of symbols `sym` with modulation scheme `mod`, and descrambling with scrambling identity `nid`. The input `rnti` is the radio network temporary identifier (RNTI) of the user equipment (UE). multi-input multi-output (MIMO) deprecoding and transform deprecoding both are disabled. When you use this syntax, the function assumes the input symbols contain data symbols only.

`[cw,symbols] = nrPUSCHDecode( ___ ,nVar)` specifies the noise variance scaling factor of the soft bits in the PUSCH demodulation in addition to the input arguments in the first syntax. When you use this syntax, the function assumes the input symbols contain data symbols only.

`[cw,symbols] = nrPUSCHDecode( ___ ,transformPrecode,mrb)` specifies transform deprecoding as a logical value and the number of allocated PUSCH resource blocks. Specify these inputs in addition to the input arguments in the second syntax. When `transformPrecode` is set to `true`, the function applies the inverse of transform precoding defined in TS 38.211 Section 6.3.1.4. `mrb` specifies the allocated number of PUSCH resource blocks. The MIMO deprecoding is disabled. When you use this syntax, the function assumes the input symbols contain data symbols only.

`[cw,symbols] = nrPUSCHDecode( ___ ,txScheme,nLayers,tpmi)` specifies the transmission scheme in addition to the input arguments in the third syntax. When `txScheme` is set to `'codebook'`, the function performs MIMO deprecoding based on the specified number of transmission layers `nLayers` and transmitted precoding matrix indicator (TPMI) `tpmi`. When you use this syntax, the function assumes the input symbols contain data symbols only.

`[cw,symbols] = nrPUSCHDecode(carrier,pusch,sym,nVar)` returns soft bits `cw` and constellation symbols `symbols` for the specified carrier configuration `carrier` and PUSCH configuration `pusch`. The input `sym` is the received PUSCH symbols for each layer and `nVar` specifies the noise variance scaling factor of the soft bits. When you use this syntax with transform precoding, the function assumes the input symbols contain data and PT-RS symbols (if applicable) and uses only data symbols for further processing.

`[cw,symbols] = nrPUSCHDecode(carrier,pusch,tcr,tbs,oack,ocsi1,ocsi2,sym,nVar)` specifies target code rate `tcr`, transport block size `tbs`, and uplink control information. `oack` is the hybrid automatic repeat request acknowledgment (HARQ-ACK) payload length. `ocsi1` is the channel state information (CSI) part 1 payload length. `ocsi2` is the CSI part 2 payload length. When you use this syntax with transform precoding, the function assumes that the input symbols contain data and PT-RS symbols (if applicable) and uses only data symbols for further processing. The function also handles the UCI placeholders when descrambling.

Examples

Generate and Decode PUSCH Modulation Symbols

Specify a random sequence of binary values corresponding to a codeword of 8064 bits.

```
cw = randi([0 1],8064,1);
```

Using 256-QAM modulation, generate PUSCH modulation symbols for the specified physical layer cell identity number, RNTI, and two transmission layers. By default, this function disables transform precoding and noncodebook-based transmission.

```
modulation = '256QAM';
nlayers = 2;
ncellid = 17;
rnti = 111;
sym = nrPUSCH(cw,modulation,nlayers,ncellid,rnti)
```

```
sym = 504×2 complex

  -0.9971 - 0.8437i   0.0767 + 0.2301i
   0.3835 + 0.2301i   0.9971 - 0.5369i
  -0.3835 - 1.1504i  -0.3835 + 0.9971i
   0.5369 + 0.0767i  -0.9971 + 0.8437i
   1.1504 - 0.9971i  -0.8437 - 0.6903i
  -0.6903 + 0.0767i   1.1504 - 0.3835i
   0.8437 + 0.6903i   1.1504 + 0.2301i
  -0.6903 - 0.2301i  -0.8437 + 1.1504i
   0.0767 + 0.8437i  -0.0767 + 0.6903i
   0.3835 - 0.8437i   0.3835 + 0.9971i
      ⋮
```

Decode the PUSCH modulation symbols.

```
demod = nrPUSCHDecode(sym,modulation,ncellid,rnti)
```

```
demod = 8064×1
10^10 ×

   -1.1529
   -0.8471
    0.2118
   -0.0941
   -0.0235
    0.0235
    0.0235
   -0.0235
   -0.0235
   -0.0941
      ⋮
```

Perform hard decision on the soft metric.

```
rxcw = double(demod<0)
```

```
rxcw = 8064×1

     1
     1
     0
     1
     1
     0
     0
     1
     1
     1
      ⋮
```

Compare the result with the original codeword.

```
isequal(cw,rxcw)
```

```
ans = logical
   1
```

Generate and Decode PUSCH Modulation Symbols for Codebook-Based Transmission

Specify a random sequence of binary values corresponding to a codeword of 8064 bits.

```
cw = randi([0 1],8064,1);
```

Using QPSK modulation, generate PUSCH modulation symbols for the specified physical layer cell identity number, RNTI, bandwidth, and one transmission layer. Enable transform precoding and codebook-based transmission based on the specified PUSCH bandwidth, TPMI, and four antennas.

```
modulation = 'QPSK';
ncellid = 17;
rnti = 111;
nlayers = 1;
transformPrecode = true;
txScheme = 'codebook';
mrb = 6;
tpmi = 1;
nports = 4;
sym = nrPUSCH(cw,modulation,nlayers,ncellid,rnti,transformPrecode,mrb,txScheme,nports,tpmi)
```

```
sym = 4032×4 complex

   0.0000 + 0.0000i  -0.1667 + 0.0833i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.0632 - 0.2911i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.1519 - 0.0450i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.3677 + 0.3664i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.3079 - 0.5027i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.8082 - 0.1640i   0.0000 + 0.0000i   0.0000 + 0.0000i
```

```
   0.0000 + 0.0000i  -0.0640 - 0.2388i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.3936 - 0.4160i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0851 - 0.4625i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0345 - 0.3333i   0.0000 + 0.0000i   0.0000 + 0.0000i
       ⋮
```

Decode the PUSCH modulation symbols assuming zero noise variance.

```
nVar = 0;
demod = nrPUSCHDecode(sym,modulation,ncellid,rnti,nVar,transformPrecode,mrb,txScheme,nlayers,tpmi
```

```
demod = 8064×1
10^10 ×

   -2.0000
   -2.0000
    2.0000
   -2.0000
   -2.0000
    2.0000
    2.0000
   -2.0000
   -2.0000
   -2.0000
       ⋮
```

Perform hard decision on the soft metric.

```
rxcv = double(demod<0)
```

```
rxcv = 8064×1

     1
     1
     0
     1
     1
     0
     0
     1
     1
     1
       ⋮
```

Compare the result with the original codeword.

```
isequal(cw,rxcv)
```

```
ans = logical
   1
```

Decode PUSCH Symbols using Configuration Parameters

Create a carrier configuration object with physical layer cell identity as 42.

```
carrier = nrCarrierConfig;
carrier.NCellID = 42;
```

Create a PUSCH configuration object with these properties.

```
pusch = nrPUSCHConfig;
pusch.Modulation = '256QAM';
pusch.NumLayers = 2;
pusch.RNTI = 111;
pusch.TransformPrecoding = 0;
pusch.TransmissionScheme = 'nonCodebook';
pusch.NID = []; % Use empty to be equal to NCellID of carrier
pusch.NSizeBWP = 25;
pusch.NStartBWP = 10;
pusch.PRBSet = 0:pusch.NSizeBWP-1; % Occupy entire bandwidth part
```

Generate PUSCH symbols for a single codeword of 8064 bits with the specified carrier configuration and uplink shared channel configuration.

```
cw = randi([0 1],8064,1);
sym = nrPUSCH(carrier,pusch,cw)
```

```
sym = 504×2 complex

  -0.3835 - 0.8437i  -1.1504 - 0.3835i
   0.6903 + 0.6903i   0.0767 + 0.3835i
  -1.1504 - 0.2301i  -1.1504 + 0.2301i
   0.9971 - 0.6903i  -0.2301 - 0.8437i
  -0.0767 + 0.2301i   0.2301 + 0.8437i
  -0.6903 + 1.1504i   0.8437 + 0.3835i
  -0.0767 + 0.8437i  -0.3835 + 0.5369i
   0.3835 + 0.5369i  -0.8437 - 1.1504i
   0.3835 + 0.2301i  -1.1504 - 0.8437i
   0.2301 - 0.0767i  -0.3835 + 1.1504i
      ⋮
```

Add an additive white Gaussian noise (AWGN) to the PUSCH symbols. Then demodulate to produce soft bit estimates.

```
SNR = 30; % SNR in dB
rxsym = awgn(sym,SNR);
demod = nrPUSCHDecode(carrier,pusch,rxsym)
```

```
demod = 8064×1
10^10 ×

   -0.2106
   -0.8118
    0.0949
   -0.0824
   -0.0231
    0.0294
    0.0239
   -0.0176
```

```
    -1.4404
    -0.1963
       ⋮
```

Perform hard decision on the soft metric.

```
rxcw = double(demod<0);
```

Compare the result with the original codeword.

```
isequal(cw,rxcw)
```

```
ans = logical
   1
```

Encode and Decode Codeword with UCI Placeholders

Create a carrier configuration object with 15 kHz subcarrier spacing and 10 MHz transmission bandwidth.

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 15;
carrier.CyclicPrefix = 'normal';
carrier.NSizeGrid = 52;
```

Create a physical uplink shared channel (PUSCH) configuration object with a QPSK modulation scheme and no frequency hopping. Set the beta offset factor for hybrid automatic repeat request acknowledgment (HARQ-ACK) to 20, and set the beta offset factor for channel state information (CSI) part 1 and CSI part 2 to 6.5 each. Specify the scaling factor as 0.8, which limits the number of resource elements (REs) assigned for the UCI.

```
pusch = nrPUSCHConfig;
pusch.Modulation = 'QPSK';
pusch.FrequencyHopping = 'neither';
pusch.BetaOffsetACK = 20;
pusch.BetaOffsetCSI1 = 6.5;
pusch.BetaOffsetCSI2 = 6.5;
pusch.UCIScaling = 0.8;
```

Set the target code rate, payload lengths of the uplink shared channel (UL-SCH) data, HARQ-ACK, CSI part 1, and CSI part 2.

```
tcr = 0.65;   % Target code rate
tbs = 900;    % Payload length of UL-SCH data (transport block size)
oack = 1;     % Payload length of HARQ-ACK
ocsi1 = 55;   % Payload length of CSI part 1
ocsi2 = 72;   % Payload length of CSI part 2
```

Get the rate-matched lengths of the data, HARQ-ACK, CSI part 1, and CSI part 2.

```
rmInfo = nrULSCHInfo(pusch,tcr,tbs,oack,ocsi1,ocsi2);
```

Create the random payload bits for the UL-SCH data, HARQ-ACK, CSI part 1, and CSI part 2.

```
data = randi([0 1],tbs,1);
ack  = randi([0 1],oack,1);
csi1 = randi([0 1],ocsi1,1);
csi2 = randi([0 1],ocsi2,1);
```

Create a UL-SCH encoder System object™.

```
encUL = nrULSCH;
```

Load the transport block into the UL-SCH encoder.

```
setTransportBlock(encUL,data);
```

Get the coded bits of length `rmInfo.GULSCH` by calling the encoder.

```
rv = 0;  % Redundancy version 0
culsch = encUL(pusch.Modulation,pusch.NumLayers,rmInfo.GULSCH,rv);
```

Encode the random payload of the HARQ-ACK, CSI part 1, and CSI part 2 for the rate-matched output lengths obtained from the `rmInfo` structure.

```
cack  = nrUCIEncode(ack,rmInfo.GACK,pusch.Modulation);
ccsi1 = nrUCIEncode(csi1,rmInfo.GCSI1,pusch.Modulation);
ccsi2 = nrUCIEncode(csi2,rmInfo.GCSI2,pusch.Modulation);
```

Get the codeword from the coded bits of the UL-SCH and the coded bits of UCI types.

```
[cw,info] = nrULSCHMultiplex(pusch,tcr,tbs,culsch,cack,ccsi1,ccsi2)
```

```
cw = 16224x1 int8 column vector

   0
   1
   0
   1
   1
   0
   0
   0
   0
   0
      ⋮

info = struct with fields:
    ULSCHIndices: [3244x1 uint32]
      ACKIndices: [338x1 uint32]
     CSI1Indices: [7252x1 uint32]
     CSI2Indices: [5390x1 uint32]
     UCIXIndices: [0x1 uint32]
     UCIYIndices: [169x1 uint32]
```

Get the PUSCH symbols.

```
sym = nrPUSCH(carrier,pusch,cw);
```

Decode the PUSCH symbols.

```
rxcw = nrPUSCHDecode(carrier,pusch,tcr,tbs,oack,ocsi1,ocsi2,sym);
```

Check that the received soft bits are equal at the locations other than the UCI placeholders.

```
indNoPlaceHolder = setdiff(1:length(cw),[info.UCIXIndices;info.UCIYIndices]);
isequal(cw(indNoPlaceHolder),int8(rxcw(indNoPlaceHolder)<0))
```

```
ans = logical
   1
```

Demultiplex the coded UL-SCH and coded UCI types from the received codeword.

```
[rxculsch,rxcack,rxccsi1,rxccsi2] = nrULSCHDemultiplex(pusch,tcr,tbs,oack,ocsi1,ocsi2,rxcw);
```

Decode the UL-SCH bits.

```
decUL = nrULSCHDecoder;
decUL.TransportBlockLength = tbs;
decUL.TargetCodeRate = tcr;
[decULBits,blkErr] = decUL(rxculsch,pusch.Modulation,pusch.NumLayers,rv);
```

Decode the HARQ-ACK, CSI part 1, and CSI part 2 bits.

```
decack = nrUCIDecode(rxcack,oack,pusch.Modulation);
deccsi1 = nrUCIDecode(rxccsi1,ocsi1,pusch.Modulation);
deccsi2 = nrUCIDecode(rxccsi2,ocsi2,pusch.Modulation);
```

Check the decoded bits of the UL-SCH, HARQ-ACK, CSI part 1, and CSI part 2.

```
isequal(data,decULBits)
```

```
ans = logical
   1
```

```
isequal(decack,ack)
```

```
ans = logical
   1
```

```
isequal(deccsi1,csi1)
```

```
ans = logical
   1
```

```
isequal(deccsi2,csi2)
```

```
ans = logical
   1
```

Input Arguments

sym — Received PUSCH modulation symbols
complex matrix

Received PUSCH modulation symbols, specified as a complex matrix.

Data Types: `single` | `double`
Complex Number Support: Yes

`mod` — Modulation scheme
`'pi/2-BPSK'` | `'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'`

Modulation scheme, specified as `'pi/2-BPSK'`, `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`. This modulation scheme determines the modulation type and number of bits used per modulation symbol.

Modulation Scheme	Number of Bits Per Symbol
`'pi/2-BPSK'`	1
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

Data Types: `char` | `string`

`nid` — Scrambling identity
integer from 0 to 1023

Scrambling identity, specified as an integer from 0 to 1023. `nid` is higher layer parameter *dataScramblingIdentityPUSCH*, ranging from 0 to 1023, if the higher layer parameter is configured. Otherwise, `nid` is physical layer cell identity number *NCellID*, ranging from 0 to 1007. For more information, see TS 38.211 Section 6.3.1.1.

Data Types: `double`

`rnti` — RNTI of UE
integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

`nVar` — Noise variance
`1e-10` (default) | nonnegative numeric scalar

Noise variance, specified as a nonnegative numeric scalar. The soft bits are scaled with the variance of additive white Gaussian noise (AWGN). The default value corresponds to an SNR of 100 dB, assuming unit signal power.

Note The default value assumes the decoder and coder are connected back-to-back, where the noise variance is zero. To avoid `-Inf` or `+Inf` values in the output, the function uses `1e-10` as the default value for noise variance. To get appropriate results when the signal is transmitted through a noisy channel, adjust the noise variance accordingly.

Data Types: `double`

`transformPrecode` — Transform deprecoding
`false` (default) | `true`

Transform deprecoding, specified as `false` or `true`. For more information, see TS 38.211 Section 6.3.1.4.

Data Types: `double` | `logical`

`mrb` — Number of allocated PUSCH resource blocks
integer from 1 to 275

Number of allocated PUSCH resource blocks, specified as an integer from 1 to 275. For more information, see TS 38.214 Section 6.1.2.

Data Types: `double`

`txScheme` — Transmission scheme
`'nonCodebook'` (default) | `'codebook'`

Transmission scheme, specified as one of these values:

- `'nonCodebook'` — Use this option to disable MIMO deprecoding.
- `'codebook'` — Use this option for codebook-based transmission using MIMO deprecoding.

For more information, see TS 38.211 Section 6.3.1.4.

Data Types: `char` | `string`

`nLayers` — Number of transmission layers
integer from 1 to 4

Number of transmission layers, specified as an integer from 1 to 4. For more information, see TS 38.211 Section 6.3.1.3.

Data Types: `double`

`tpmi` — Transmitted precoding matrix indicator
integer from 0 to 27

Transmitted precoding matrix indicator, specified as an integer from 0 to 27. The valid range of `tpmi` depends on the specified number of transmission layers `nLayers` and the number of ports. For more information, see TS 38.211 Tables 6.3.1.5-1 to 6.3.1.5-7.

Data Types: `double`

`carrier` — Carrier configuration parameters
`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function uses only these properties of the `nrCarrierConfig` object.

- `NCellID`
- `SubcarrierSpacing`
- `CyclicPrefix`
- `NSizeGrid`
- `NStartGrid`
- `NSlot`

`pusch` — PUSCH configuration parameters
`nrPUSCHConfig` object

PUSCH configuration parameters for a specific OFDM numerology, specified as an `nrPUSCHConfig` object. This function uses only these properties of the `nrPUSCHConfig` object.

- `Modulation`
- `NumLayers`
- `MappingType`
- `SymbolAllocation`
- `PRBSet`
- `TransformPrecoding`
- `TransmissionScheme`
- `NumAntennaPorts`
- `TPMI`
- `FrequencyHopping`
- `NID`
- `RNTI`
- `NRAPID`
- `BetaOffsetAck`
- `BetaOffsetCSI1`
- `BetaOffsetCSI2`
- `UCIScaling`
- `DMRS`
- `EnablePTRS`
- `PTRS`

`tcr` — Target code rate
scalar in the range (0, 1)

Target code rate for the codeword in the UL-SCH transmission, specified as a scalar in the range (0, 1).

Data Types: `double`

`tbs` — Transport block size
nonnegative integer

Transport block size associated with the codeword in the UL-SCH transmission, specified as a nonnegative integer. A value of `0` indicates no transport block or no UL-SCH transmission on the PUSCH.

Data Types: `double`

`oack` — Payload length of HARQ-ACK bits
nonnegative integer

Payload length of the HARQ-ACK bits, specified as a nonnegative integer. A value of `0` indicates no HARQ-ACK transmission.

Data Types: `double`

`ocsi1` — Payload length of CSI part 1 bits
nonnegative integer

Payload length of the CSI part 1 bits, specified as a nonnegative integer. A value of `0` indicates no CSI part 1 transmission.

Data Types: `double`

`ocsi2` — Payload length of CSI part 2 bits
nonnegative integer

Payload length of the CSI part 2 bits, specified as a nonnegative integer. A value of `0` indicates no CSI part 2 transmission. Nominally, CSI part 2 is present only when CSI part 1 is present.

Data Types: `double`

Output Arguments

`cw` — Approximate LLR soft bits
real column vector

Approximate log-likelihood ratio (LLR) soft bits, returned as a real column vector. `cw` inherits the data type of `sym`. Sign represents hard bits.

Data Types: `double` | `single`

`symbols` — Constellation symbols
column vector of complex numbers

Constellation symbols for `cw`, returned as a column vector of complex numbers. `symbols` inherits the data type of `sym`.

Data Types: `double` | `single`
Complex Number Support: Yes

Version History

Introduced in R2019a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
nrPUSCH | nrPUSCHCodebook | nrPUSCHDescramble | nrULSCHInfo

Objects
nrCarrierConfig | nrPUSCHConfig | nrPUSCHPTRSConfig | nrPUSCHDMRSConfig

nrPUSCHDescramble

Perform PUSCH descrambling

Syntax

```
cw = nrPUSCHDescramble(in,nid,rnti)
cw = nrPUSCHDescramble(in,nid,rnti,nrapid)
cw = nrPUSCHDescramble( ___ ,xInd,yInd)
```

Description

`cw = nrPUSCHDescramble(in,nid,rnti)` returns a column vector of soft bits resulting from the inverse operation of physical uplink shared channel (PUSCH) scrambling from TS 38.211 Section 6.3.1.1 [1]. `in` is a vector of scrambled soft bits, `nid` is the scrambling identity, and `rnti` is the radio network temporary identifier (RNTI) of the user equipment (UE). When you use this syntax, the function descrambles only the data bits because the placeholder bit locations for any uplink control information (UCI), if present, are unknown in this case.

`cw = nrPUSCHDescramble(in,nid,rnti,nrapid)` specifies random access preamble index `nrapid` to initialize the scrambling sequence for *msgA* on PUSCH, as defined in Release 16 of TS 38.211 Section 6.3.1.1.

`cw = nrPUSCHDescramble( ___ ,xInd,yInd)` specifies UCI placeholder X bit locations `xInd` and UCI placeholder Y bit locations `yInd` in addition to the input arguments in any of the previous syntaxes. The inputs `xInd` and `yInd` are 1-based column vectors within the codeword and indicate the respective placeholder locations.

Examples

Perform PUSCH Descrambling

Create a random sequence of binary values corresponding to a codeword containing 3000 bits. Perform PUSCH scrambling initialized with the specified physical layer cell identity number and RNTI.

```
cw = randi([0 1],3000,1);
ncellid = 42;
rnti = 101;
scrambled = nrPUSCHScramble(cw,ncellid,rnti);
```

Modulate the scrambled data using 16-QAM modulation. Demodulate the result.

```
modulation = '16QAM';
sym = nrSymbolModulate(scrambled,modulation);
demod = nrSymbolDemodulate(sym,modulation);
```

Perform PUSCH descrambling of the demodulated symbols.

```
descrambled = nrPUSCHDescramble(demod,ncellid,rnti)
```

```
descrambled = 3000×1
10^10 ×

   -1.6000
   -1.6000
    0.4000
   -0.4000
   -1.6000
    0.4000
    0.4000
   -0.4000
   -0.4000
   -0.4000
      ⋮
```

Perform hard decision on the soft metric.

```
rxcw = double(descrambled<0)
```

```
rxcw = 3000×1

     1
     1
     0
     1
     1
     0
     0
     1
     1
     1
      ⋮
```

Compare the result with the original codeword.

```
isequal(cw,rxcw)
```

```
ans = logical
   1
```

Perform PUSCH Descrambling with UCI Placeholder Indices

Create a codeword for 1 bit.

```
cw = [1 -2 -1 -1]';
```

Specify the scrambling identity as 100 and radio network temporary identifier as 65,350.

```
nid = 100;
rnti = 65350;
```

Perform PUSCH scrambling that is initialized with the UCI placeholder X and Y bit locations.

```
xind = find(cw == -1);
yind = find(cw == -2);
scrambled = nrPUSCHScramble(cw,nid,rnti);
```

Modulate the scrambled data using 16-QAM scheme. Demodulate the result.

```
modulation = '16QAM';
sym = nrSymbolModulate(scrambled,modulation);
demod = nrSymbolDemodulate(sym,modulation);
```

Perform PUSCH descrambling of the demodulated symbols.

```
descrambled = nrPUSCHDescramble(demod,nid,rnti,xind,yind)
```

```
descrambled = 4×1
10^10 ×

   -1.6000
   -1.6000
   -0.4000
   -0.4000
```

Compare the result with the original codeword.

```
isequal(descrambled(1)<0,cw(1))
```

```
ans = logical
   1
```

Input Arguments

in — Approximate LLR soft bits
real column vector

Approximate log-likelihood ratio (LLR) soft bits, specified as a real column vector. Sign represents scrambled hard bit.

Data Types: double | single

nid — Scrambling identity
integer from 0 to 1023

Scrambling identity, specified as an integer from 0 to 1023. nid is higher layer parameter *dataScramblingIdentityPUSCH*, ranging from 0 to 1023, if the higher layer parameter is configured. Otherwise, nid is physical layer cell identity number *NCellID*, ranging from 0 to 1007. For more information, see TS 38.211 Section 6.3.1.1.

Data Types: double

rnti — RNTI of UE
integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: double

`nrapid` — Random access preamble index
`[]` (default) | integer from 0 to 63

Random access preamble index, specified as one of these values.

- `[]` — Use this value to indicate that the scrambling initialization does not consider *msgA* on PUSCH.
- Integer from 0 to 63 — Use this value to initialize the scrambling sequence for *msgA* on PUSCH, as defined in Release 16 of TS 38.211 Section 6.3.1.1.

Data Types: `double`

`xInd` — X placeholder indices of UCI
column vector of positive values

X placeholder indices of the UCI, specified as a column vector of positive values. When you specify this input, the function descrambles the input codeword, `in`, at all locations except the X placeholder locations. A value of `[]` indicates no X placeholder indices.

Data Types: `double`

`yInd` — Y placeholder indices of UCI
column vector of positive values

Y placeholder indices of the UCI, specified as a column vector of positive values. The input codeword, `in`, at the placeholder locations, `yInd`, is descrambled with the previous values of the scrambling sequence. A value of `[]` indicates no Y placeholder indices.

Data Types: `double`

Output Arguments

`cw` — Descrambled approximate LLR soft bits
numeric column vector

Descrambled approximate LLR soft bits, returned as a numeric column vector. Sign represents descrambled hard bit.

Data Types: `double` | `single`

Version History
Introduced in R2019a

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrPUSCHScramble` | `nrPUSCHPRBS`

nrPUSCHDMRS

Generate PUSCH DM-RS symbols

Syntax

```
sym = nrPUSCHDMRS(carrier,pusch)
sym = nrPUSCHDMRS(carrier,pusch,'OutputDataType',datatype)
```

Description

`sym = nrPUSCHDMRS(carrier,pusch)` returns a matrix containing demodulation reference signal (DM-RS) symbols of physical uplink shared channel (PUSCH), as defined in TS 38.211 Section 6.4.1.1.1 [1]. `carrier` specifies the carrier configuration parameters for a specific OFDM numerology. `pusch` specifies the PUSCH configuration parameters.

`sym = nrPUSCHDMRS(carrier,pusch,'OutputDataType',datatype)` specifies the data type for the DM-RS symbols.

Examples

Generate PUSCH DM-RS Symbols for CP-OFDM

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier with 15 kHz subcarrier spacing.

```
carrier = nrCarrierConfig;
```

Create a physical uplink shared channel (PUSCH) configuration object with specified properties. When transform precoding is set to 0, the waveform type is cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM).

```
pusch = nrPUSCHConfig;
pusch.TransformPrecoding = 0;
pusch.TransmissionScheme = 'codebook';
pusch.NumAntennaPorts = 4;
pusch.TPMI = 0;
```

Configure PUSCH demodulation reference signal (DM-RS) with specified parameters.

```
pusch.DMRS.DMRSAdditionalPosition = 1;
pusch.DMRS.DMRSTypeAPosition = 2;
pusch.DMRS.DMRSPortSet = 2;
pusch.DMRS.NIDNSCID = 10;
pusch.DMRS.NSCID = 1;
```

Generate DM-RS symbols associated with PUSCH of `single` data type.

```
sym = nrPUSCHDMRS(carrier,pusch,'OutputDataType','single')
```

```
sym = 624x4 single matrix
```

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

Generate PUSCH DM-RS Symbols and Indices

Create a carrier configuration with 30 kHz subcarrier spacing and 5 MHz transmission bandwidth.

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 30;
carrier.NSizeGrid = 11;
```

Create a physical uplink shared channel (PUSCH) configuration object with specified properties. When transform precoding is set to 1, the waveform type is discrete fourier transform spread orthogonal frequency division multiplexing (DFT-s-OFDM).

```
pusch = nrPUSCHConfig;
pusch.NSizeBWP = 9;
pusch.NStartBWP = 1;
pusch.PRBSet = 0:3;
pusch.TransformPrecoding = 1;
pusch.FrequencyHopping = 'intraSlot';
pusch.SecondHopStartPRB = 5;
```

Create a PUSCH demodulation reference signal (DM-RS) object with specified properties.

```
dmrs = nrPUSCHDMRSConfig;
dmrs.DMRSAdditionalPosition = 1;
dmrs.DMRSTypeAPosition = 2;
dmrs.DMRSPortSet = 3;
dmrs.GroupHopping = 1;
dmrs.SequenceHopping = 0;
dmrs.NRSID = 10;
```

Assign the PUSCH DM-RS configuration object to DMRS property of PUSCH configuration object.

```
pusch.DMRS = dmrs;
```

Generate PUSCH DM-RS symbols and indices for the specified carrier, PUSCH configuration, and output formatting `name-value` pair argument.

```
sym = nrPUSCHDMRS(carrier,pusch,'OutputDataType','single')
```

```
sym = 96x1 single column vector

  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
```

```
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
      ⋮
```

```
ind = nrPUSCHDMRSIndices(carrier,pusch,'IndexBase','0based','IndexOrientation','bwp')
```

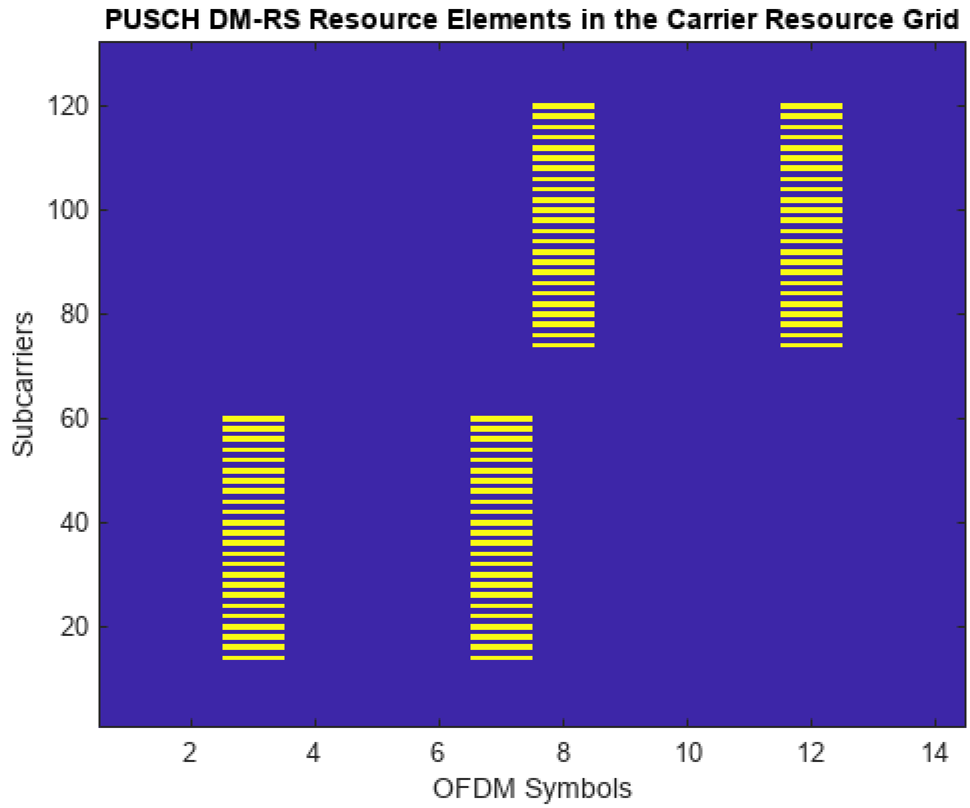
```
ind = 96x1 uint32 column vector

   217
   219
   221
   223
   225
   227
   229
   231
   233
   235
      ⋮
```

Create a bandwidth part (BWP) grid, and then map the DM-RS symbols on the grid.

```
bwp = complex(zeros([pusch.NSizeBWP*12 carrier.SymbolsPerSlot pusch.NumLayers]));
bwp(ind+1) = sym; % Map the DM-RS symbols
```

Map the BWP to the carrier resource grid, and then display the carrier grid.

```
grid = complex(zeros([carrier.NSizeGrid*12 carrier.SymbolsPerSlot pusch.NumLayers])); % Create ca
offset = pusch.NStartBWP-carrier.NStartGrid; % BWP start location in the carrier grid
grid(offset*12+1:(offset+pusch.NSizeBWP)*12,:,:) = bwp;
imagesc(abs(grid(:,:,1)));
axis xy;
xlabel('OFDM Symbols');
ylabel('Subcarriers');
title('PUSCH DM-RS Resource Elements in the Carrier Resource Grid');
```



Input Arguments

`carrier` — Carrier configuration parameters
`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function uses only these properties of the `nrCarrierConfig` object.

- `NCellID`
- `SubcarrierSpacing`
- `CyclicPrefix`
- `NSizeGrid`
- `NStartGrid`
- `NSlot`

`pusch` — PUSCH configuration parameters
`nrPUSCHConfig` object

PUSCH configuration parameters, specified as an `nrPUSCHConfig` object. This function uses only these properties of the `nrPUSCHConfig` object.

- `NSizeBWP`

- `NStartBWP`
- `Modulation`
- `NumLayers`
- `MappingType`
- `SymbolAllocation`
- `PRBSet`
- `TransformPrecoding`
- `TransmissionScheme`
- `NumAntennaPorts`
- `TPMI`
- `FrequencyHopping`
- `SecondHopStartPRB`
- `DMRS`

`datatype` — Data type for generated DM-RS symbols
`'double'` (default) | `'single'`

Data type for the generated DM-RS symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

`sym` — DM-RS symbols
complex matrix

DM-RS symbols, returned as a complex matrix. The number of columns correspond to the number of antenna ports configured.

Data Types: `single` | `double`
Complex Number Support: Yes

Version History

Introduced in R2020a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The `datatype` input argument must be compile-time constant. For example, when specifying `'single'` as the output data type, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPUSCH` | `nrPUSCHDMRSIndices` | `nrPUSCHPTRS` | `nrChannelEstimate` | `nrTimingEstimate`

Objects
`nrCarrierConfig` | `nrPUSCHDMRSConfig` | `nrPUSCHConfig`

nrPUSCHDMRSIndices

Generate PUSCH DM-RS indices

Syntax

```
ind = nrPUSCHDMRSIndices(carrier,pusch)
ind = nrPUSCHDMRSIndices(carrier,pusch,Name,Value)
```

Description

`ind = nrPUSCHDMRSIndices(carrier,pusch)` returns a matrix containing demodulation reference signal (DM-RS) resource element (RE) indices of a physical uplink shared channel (PUSCH), as defined in TS 38.211 Section 6.4.1.1.3 [1]. `carrier` specifies the carrier configuration parameters for a specific OFDM numerology. `pusch` specifies the PUSCH configuration parameters. The returned indices are 1-based using linear indexing form.

`ind = nrPUSCHDMRSIndices(carrier,pusch,Name,Value)` specifies output formatting options by using one or more name-value pair arguments. Unspecified options take default values.

Examples

Generate PUSCH DM-RS Indices for Codebook-Based Transmission

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier with 15 kHz subcarrier spacing.

```
carrier = nrCarrierConfig;
```

Create a physical uplink shared channel (PUSCH) configuration object with specified properties. When transform precoding is set to 0, the waveform type is cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM).

```
pusch = nrPUSCHConfig;
pusch.TransformPrecoding = 0;
pusch.TransmissionScheme = 'codebook';
pusch.NumAntennaPorts = 4;
pusch.TPMI = 0;
```

Configure PUSCH demodulation reference signal (DM-RS) object with specified parameters.

```
pusch.DMRS.DMRSAdditionalPosition = 2;
pusch.DMRS.DMRSTypeAPosition = 2;
pusch.DMRS.DMRSPortSet = 3;
pusch.DMRS.NIDNSCID = 15;
pusch.DMRS.NSCID = 1;
```

Generate DM-RS indices associated to PUSCH of subscript indexing form.

```
ind = nrPUSCHDMRSIndices(carrier,pusch,'IndexStyle','subscript')
```

```
ind = 3744x3 uint32 matrix
```

```
     2     3     1
     4     3     1
     6     3     1
     8     3     1
    10     3     1
    12     3     1
    14     3     1
    16     3     1
    18     3     1
    20     3     1
       ⋮
```

Generate PUSCH DM-RS Symbols and Indices

Create a carrier configuration with 30 kHz subcarrier spacing and 5 MHz transmission bandwidth.

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 30;
carrier.NSizeGrid = 11;
```

Create a physical uplink shared channel (PUSCH) configuration object with specified properties. When transform precoding is set to 1, the waveform type is discrete fourier transform spread orthogonal frequency division multiplexing (DFT-s-OFDM).

```
pusch = nrPUSCHConfig;
pusch.NSizeBWP = 9;
pusch.NStartBWP = 1;
pusch.PRBSet = 0:3;
pusch.TransformPrecoding = 1;
pusch.FrequencyHopping = 'intraSlot';
pusch.SecondHopStartPRB = 5;
```

Create a PUSCH demodulation reference signal (DM-RS) object with specified properties.

```
dmrs = nrPUSCHDMRSConfig;
dmrs.DMRSAdditionalPosition = 1;
dmrs.DMRSTypeAPosition = 2;
dmrs.DMRSPortSet = 3;
dmrs.GroupHopping = 1;
dmrs.SequenceHopping = 0;
dmrs.NRSID = 10;
```

Assign the PUSCH DM-RS configuration object to DMRS property of PUSCH configuration object.

```
pusch.DMRS = dmrs;
```

Generate PUSCH DM-RS symbols and indices for the specified carrier, PUSCH configuration, and output formatting `name-value` pair argument.

```
sym = nrPUSCHDMRS(carrier,pusch,'OutputDataType','single')
```

```
sym = 96x1 single column vector

  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
```

```
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
       ⋮
```

```
ind = nrPUSCHDMRSIndices(carrier,pusch,'IndexBase','0based','IndexOrientation','bwp')
```

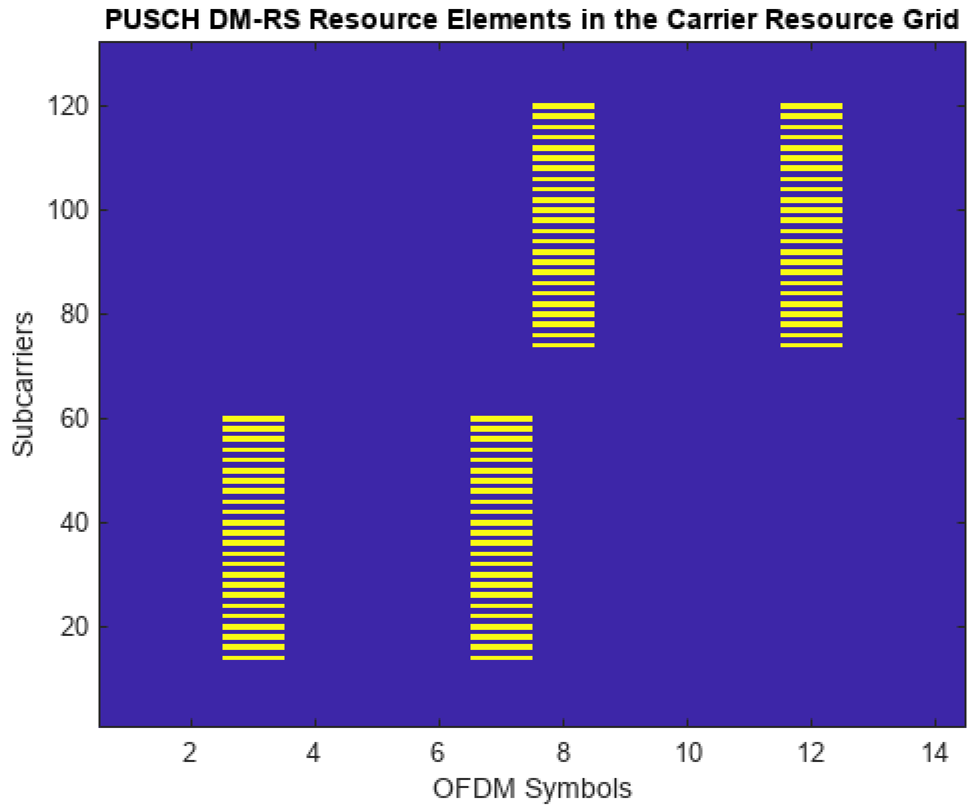
```
ind = 96x1 uint32 column vector

   217
   219
   221
   223
   225
   227
   229
   231
   233
   235
     ⋮
```

Create a bandwidth part (BWP) grid, and then map the DM-RS symbols on the grid.

```
bwp = complex(zeros([pusch.NSizeBWP*12 carrier.SymbolsPerSlot pusch.NumLayers]));
bwp(ind+1) = sym; % Map the DM-RS symbols
```

Map the BWP to the carrier resource grid, and then display the carrier grid.

```
grid = complex(zeros([carrier.NSizeGrid*12 carrier.SymbolsPerSlot pusch.NumLayers])); % Create ca
offset = pusch.NStartBWP-carrier.NStartGrid; % BWP start location in the carrier grid
grid(offset*12+1:(offset+pusch.NSizeBWP)*12,:,:) = bwp;
imagesc(abs(grid(:,:,1)));
axis xy;
xlabel('OFDM Symbols');
ylabel('Subcarriers');
title('PUSCH DM-RS Resource Elements in the Carrier Resource Grid');
```



Input Arguments

carrier — Carrier configuration parameters
nrCarrierConfig object

Carrier configuration parameters for a specific OFDM numerology, specified as an nrCarrierConfig object. This function uses only these properties of the nrCarrierConfig object.

- SubcarrierSpacing
- CyclicPrefix
- NSizeGrid
- NStartGrid
- NSlot

pusch — PUSCH configuration parameters
nrPUSCHConfig object

PUSCH configuration parameters, specified as an nrPUSCHConfig object. This function uses only these properties of the nrPUSCHConfig object.

- NSizeBWP
- NStartBWP

- `NumLayers`
- `MappingType`
- `SymbolAllocation`
- `PRBSet`
- `TransformPrecoding`
- `TransmissionScheme`
- `NumAntennaPorts`
- `TPMI`
- `FrequencyHopping`
- `SecondHopStartPRB`
- `DMRS`

Name-Value Pair Arguments

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

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

Example: `'IndexStyle','subscript','IndexBase','0based'` specifies nondefault resource element index formatting properties.

`IndexStyle` — RE indexing form

`'index'` (default) | `'subscript'`

RE indexing form, specified as one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

`IndexBase` — RE indexing base

`'1based'` (default) | `'0based'`

RE indexing base, specified as one of these values:

- `'1based'` — The index counting starts from 1.
- `'0based'` — The index counting starts from 0.

Data Types: `char` | `string`

`IndexOrientation` — Indexing orientation of resource elements

`'carrier'` (default) | `'bwp'`

Indexing orientation of resource elements, specified as the comma-separated pair consisting of `'IndexOrientation'` and one of these values:

- `'carrier'` — Indices are referenced with respect to the carrier grid.
- `'bwp'` — Indices are referenced with respect to the bandwidth part.

Data Types: `char` | `string`

Output Arguments

ind — DM-RS resource element indices
N-by-*P* matrix | *M*-by-3 matrix

DM-RS resource element indices, returned as one of these values:

- *N*-by-*P* matrix — The function returns this type of value when `'IndexStyle'` is set to `'index'`. The number of columns depends on the `TransmissionScheme` property of `nrPUSCHConfig` object and returned as one these values.
  - Number of transmission layers — When the transmission scheme is non-codebook
  - Number of antenna ports configured — When the transmission scheme is codebook
- *M*-by-3 matrix — The function returns this type of value when `'IndexStyle'` is set to `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and number of antennas, respectively.

Depending on the value of `'IndexBase'`, the function returns either 1-based or 0-based indices. Depending on the value of `'IndexOrientation'`, the function returns either carrier oriented indices or BWP oriented indices.

Data Types: `uint32`

Version History

Introduced in R2020a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPUSCHDMRS` | `nrPUSCHPTRSIndices` | `nrPUSCHIndices` | `nrChannelEstimate` | `nrTimingEstimate`

Objects
`nrCarrierConfig` | `nrPUSCHDMRSConfig` | `nrPUSCHConfig`

nrPUSCHIndices

Generate PUSCH resource element indices

Syntax

```
[ind,info,ptrsInd] = nrPUSCHIndices(carrier,pusch)
[ind,info,ptrsInd] = nrPUSCHIndices(carrier,pusch,Name,Value)
```

Description

`[ind,info,ptrsInd] = nrPUSCHIndices(carrier,pusch)` returns `ind` in matrix form, which contains 1-based physical uplink shared channel (PUSCH) resource element (RE) indices, as defined in TS 38.211 Sections 6.3.1.6 and 6.3.1.7 [1]. The number of columns in `ind` is equal to the number of configured antenna ports. `carrier` specifies the carrier configuration parameters for a specific OFDM numerology and `pusch` specifies the PUSCH configuration. When you enable transform precoding, the output `ind` contains the combined locations of the data and the phase tracking reference signal (PT-RS). The function also returns the structural information, `info`, and PT-RS RE indices, `ptrsInd`. The output `info` contains information about the associated physical reference signals, bit capacity, and symbol capacity. `ptrsInd` is a matrix of PT-RS REs within the carrier resource grid. When you enable transform precoding, the output `ptrsInd` represents the projections of PT-RS locations prior to transform precoding onto the carrier resource grid.

`[ind,info,ptrsInd] = nrPUSCHIndices(carrier,pusch,Name,Value)` specifies output formatting options using one or more name-value pair arguments. Unspecified options take default values.

Examples

Generate PUSCH Indices and PT-RS Indices for Codebook-Based Transmission

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier with 15 kHz subcarrier spacing.

```
carrier = nrCarrierConfig;
```

Create a PUSCH configuration object with codebook-based transmission. Set the number of antenna ports to 4, modulation scheme to pi/2-BPSK, transmitted precoding matrix indicator to 10, and transform precoding to 0. When transform precoding is 0, the waveform type is cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM). Enable phase tracking reference signal (PT-RS).

```
pusch = nrPUSCHConfig;
pusch.TransformPrecoding = 0;
pusch.Modulation = 'pi/2-BPSK';
pusch.TransmissionScheme = 'codebook';
pusch.NumAntennaPorts = 4;
pusch.TPMI = 10;
pusch.EnablePTRS = 1;
```

Generate PUSCH indices and PT-RS indices in subscript form.

```
[ind,info,ptrsInd] = nrPUSCHIndices(carrier,pusch,'IndexStyle','subscript')
```

```
ind = 31096x3 uint32 matrix

    1    1    1
    2    1    1
    3    1    1
    4    1    1
    5    1    1
    6    1    1
    7    1    1
    8    1    1
    9    1    1
   10    1    1
      ⋮

info = struct with fields:
                G: 7774
               Gd: 7774
        NREPerPRB: 156
    DMRSSymbolSet: 2
    PTRSSymbolSet: [0 1 3 4 5 6 7 8 9 10 11 12 13]

ptrsInd = 1352x3 uint32 matrix

    13     1     1
    37     1     1
    61     1     1
    85     1     1
   109     1     1
   133     1     1
   157     1     1
   181     1     1
   205     1     1
   229     1     1
      ⋮
```

Generate PUSCH Symbols and Indices

Create a carrier configuration object with default properties. This object corresponds to 30 kHz of subcarrier spacing and 20 MHz transmission bandwidth.

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 30;
carrier.NSizeGrid = 51;
```

Create a PUSCH configuration object with specified properties. When transform precoding is 1, the waveform type is discrete fourier transform spread orthogonal frequency division multiplexing (DFT-s-OFDM).

```
pusch = nrPUSCHConfig;
pusch.NStartBWP = 10;
pusch.NSizeBWP = 41;
```

```
pusch.Modulation = '16QAM';
pusch.NID = []; % Set NID equal to the NCellID property of carrier.
pusch.PRBSet = 0:5;
pusch.TransformPrecoding = 1;
pusch.FrequencyHopping = 'intraSlot';
pusch.SecondHopStartPRB = 3;
```

Generate PUSCH indices, setting the index orientation with respect to the carrier grid.

```
[ind,info] = nrPUSCHIndices(carrier,pusch,'IndexOrientation','carrier')
```

```
ind = 864x1 uint32 column vector

   121
   122
   123
   124
   125
   126
   127
   128
   129
   130
     ⋮

info = struct with fields:
                G: 3456
               Gd: 864
        NREPerPRB: 144
    DMRSSymbolSet: [2 7]
    PTRSSymbolSet: [1x0 double]
```

Generate PUSCH symbols of data type single.

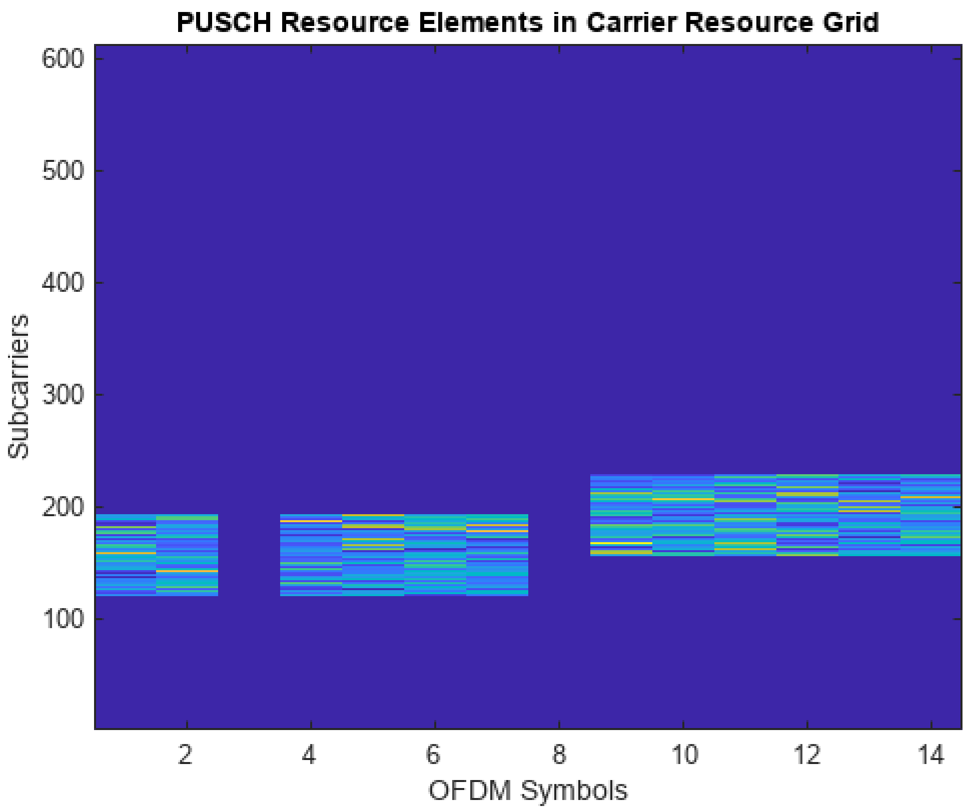
```
numDataBits = info.G;
cws = randi([0 1],numDataBits,1);
sym = nrPUSCH(carrier,pusch,cws,'OutputDataType','single')
```

```
sym = 864x1 single column vector

  -0.7454 + 0.2981i
   0.3406 - 0.2312i
  -0.1153 + 0.2756i
   1.1921 - 0.3658i
  -0.3968 - 0.0277i
  -0.8788 - 0.6493i
  -0.8737 + 0.8318i
  -0.5764 + 0.0269i
  -1.6638 + 0.0482i
  -1.0270 - 0.1347i
      ⋮
```

Plot the generated symbols and indices on the carrier resource grid.

```
grid = complex(zeros([carrier.NSizeGrid*12 carrier.SymbolsPerSlot pusch.NumLayers]));
grid(ind) = sym;
```

```
imagesc(abs(grid(:,:,1)));
axis xy;
xlabel('OFDM Symbols');
ylabel('Subcarriers');
title('PUSCH Resource Elements in Carrier Resource Grid');
```



Input Arguments

carrier — Carrier configuration parameters
nrCarrierConfig object

Carrier configuration parameters for a specific OFDM numerology, specified as an nrCarrierConfig object. This function only uses these nrCarrierConfig object properties.

- SubcarrierSpacing
- CyclicPrefix
- NSizeGrid
- NStartGrid
- NSlot

pusch — PUSCH configuration parameters
nrPUSCHConfig object

PUSCH configuration parameters for a specific OFDM numerology, specified as an `nrPUSCHConfig` object. This function uses only these properties of the `nrPUSCHConfig` object.

- `NSizeBWP`
- `NStartBWP`
- `Modulation`
- `NumLayers`
- `MappingType`
- `SymbolAllocation`
- `PRBSet`
- `TransformPrecoding`
- `TransmissionScheme`
- `NumAntennaPorts`
- `TPMI`
- `FrequencyHopping`
- `SecondHopStartPRB`
- `NID`
- `RNTI`
- `DMRS`
- `EnablePTRS`
- `PTRS`

Name-Value Pair Arguments

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

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

Example: `'IndexStyle','subscript','IndexBase','0based'` specifies the RE indexing form and base, respectively, of the output.

IndexStyle — RE indexing form
`'index'` (default) | `'subscript'`

RE indexing form, specified as one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

IndexBase — RE indexing base
`'1based'` (default) | `'0based'`

RE indexing base, specified as one of these values:

- '1based' — The index counting starts from 1.
- '0based' — The index counting starts from 0.

Data Types: char | string

IndexOrientation — Resource element indexing orientation
'carrier' (default) | 'bwp'

Resource element indexing orientation, specified as the comma-separated pair consisting of 'IndexOrientation' and one of these values:

- 'carrier' — Indices are referenced with respect to the carrier grid.
- 'bwp' — Indices are referenced with respect to the BWP.

Dependencies

This property is applicable only when TransformPrecoding property of nrPUSCHConfig object is set to 0.

Data Types: char | string

Output Arguments

ind — PUSCH RE indices
N-by-*P* matrix | *M*-by-3 matrix

PUSCH RE indices, returned as one of these values.

- *N*-by-*P* matrix — The function returns this type of value when you set 'IndexStyle' to 'index'. The number of columns depends on the TransmissionScheme property of the nrPUSCHConfig object and is returned as one these values.
  - Number of transmission layers — When the transmission scheme is noncodebook
  - Number of antenna ports configured — When the transmission scheme is codebook
- *M*-by-3 matrix — The function returns this type of value when you set 'IndexStyle' to 'subscript'. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and number of antennas, respectively.

Depending on the value of 'IndexBase', the function returns either 1-based or 0-based indices. Depending on the value of 'IndexOrientation', the function returns either carrier-oriented indices or BWP-oriented indices.

Data Types: uint32

info — PUSCH resource information
structure

PUSCH resource information, returned as a structure containing these fields.

Field	Description
G	Bit capacity of the PUSCH. This value must be equal to the length of the codeword from the uplink shared channel (UL-SCH) transport channel.
Gd	Number of REs per layer or port

Field	Description
DMRSSymbolSet	The OFDM symbol locations in a slot containing the demodulation reference signal (DM-RS). The symbols are 0-based.
NREPerPRB	Number of REs per PRB allocated to the PUSCH
PTRSSymbolSet	The OFDM symbol locations in a slot containing the phase tracking reference signal (PT-RS). The symbols are 0-based.

`ptrsInd` — PT-RS RE indices
N-by-*P* matrix | *M*-by-3 matrix

PT-RS RE indices, returned as one of these values.

- *N*-by-*P* matrix — The function returns this type of value when you set `'IndexStyle'` to `'index'`. The number of columns depends on the `TransmissionScheme` property of the `nrPUSCHConfig` object and is returned as one these values.
  - Number of transmission layers — When the transmission scheme is noncodebook
  - Number of antenna ports configured — When the transmission scheme is codebook
- *M*-by-3 matrix — The function returns this type of value when you set `'IndexStyle'` to `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and number of antennas, respectively.

Depending on the value of `'IndexBase'`, the function returns either 1-based or 0-based indices. Depending on the value of `'IndexOrientation'`, the function returns either carrier-oriented indices or BWP-oriented indices.

Data Types: `uint32`

Version History

Introduced in R2020a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
nrPUSCHDecode | nrPUSCH | nrPUSCHPTRSIndices | nrPUSCHDMRSIndices

Objects
nrCarrierConfig | nrPUSCHConfig | nrPUSCHPTRSConfig | nrPUSCHDMRSConfig

nrPUSCHPRBS

Generate PUSCH scrambling sequence

Syntax

```
[seq,cinit] = nrPUSCHPRBS(nid,rnti,n)
[seq,cinit] = nrPUSCHPRBS(nid,rnti,nrapid,n)
[seq,cinit] = nrPUSCHPRBS(___,Name,Value)
```

Description

`[seq,cinit] = nrPUSCHPRBS(nid,rnti,n)` returns the first `n` elements of the physical uplink shared channel (PUSCH) scrambling sequence. The function also returns initialization value `cinit` of the pseudorandom binary sequence (PRBS) generator. The initialization value depends on scrambling identity `nid` and radio network temporary identifier (RNTI) of the user equipment (UE) `rnti`. The function implements TS 38.211 Section 6.3.1.1 [1].

`[seq,cinit] = nrPUSCHPRBS(nid,rnti,nrapid,n)` specifies random access preamble index `nrapid` to initialize the scrambling sequence for *msgA* on PUSCH, as defined in Release 16 of TS 38.211 Section 6.3.1.1.

`[seq,cinit] = nrPUSCHPRBS(___,Name,Value)` specifies additional name-value arguments in addition to the input arguments in any of the previous syntaxes.

Examples

Generate PUSCH Scrambling Sequence

Generate the first 300 elements of the PUSCH scrambling sequence when initialized with the specified physical layer cell identity number and RNTI.

```
ncellid = 17;
rnti = 120;
n = 300;
seq = nrPUSCHPRBS(ncellid,rnti,n)
```

```
seq = 300x1 logical array

   0
   1
   1
   0
   1
   1
   0
   1
   0
   0
      ⋮
```

Input Arguments

nid — Scrambling identity
integer from 0 to 1023

Scrambling identity, specified as an integer from 0 to 1023. `nid` is higher layer parameter *dataScramblingIdentityPUSCH*, ranging from 0 to 1023, if the higher layer parameter is configured. Otherwise, `nid` is physical layer cell identity number *NCellID*, ranging from 0 to 1007. For more information, see TS 38.211 Section 6.3.1.1.

Data Types: `double`

rnti — RNTI of UE
integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

nrapid — Random access preamble index
`[]` (default) | integer from 0 to 63

Random access preamble index, specified as one of these values.

- `[]` — Use this value to indicate that the scrambling initialization does not consider *msgA* on PUSCH.
- Integer from 0 to 63 — Use this value to initialize the scrambling sequence for *msgA* on PUSCH, as defined in Release 16 of TS 38.211 Section 6.3.1.1.

Data Types: `double`

n — Number of elements in output sequence
nonnegative integer

Number of elements in output sequence, specified as a nonnegative integer.

Data Types: `double`

Name-Value Pair Arguments

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

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

Example: `'MappingType','signed'` specifies nondefault sequence formatting.

MappingType — Output sequence formatting
`'binary'` (default) | `'signed'`

Output sequence formatting, specified as the comma-separated pair consisting of `'MappingType'` and one of these values:

- `'binary'` — This value maps `true` to `1` and `false` to `0`. The data type of the output sequence is `logical`.

- 'signed' — This value maps true to −1 and false to 1. The data type of the output sequence is double. To specify single data type, use the 'OutputDataType' name-value pair.

Data Types: char | string

OutputDataType — Data type of output sequence
'double' (default) | 'single'

Data type of output sequence, specified as the comma-separated pair consisting of 'OutputDataType' and 'double' or 'single'. This name-value pair applies only when 'MappingType' is set to 'signed'.

Data Types: char | string

Output Arguments

seq — PUSCH scrambling sequence
logical column vector | numeric column vector

PUSCH scrambling sequence, returned as a logical or numeric column vector. seq contains the first n elements of the PDSCH scrambling sequence. If you set 'MappingType' to 'signed', the output data type is either double or single. If you set 'MappingType' to 'binary', the output data type is logical.

Data Types: double | single | logical

cinit — Initialization value for PRBS generator
nonnegative integer

Initialization value for PRBS generator, returned as a nonnegative integer.

Data Types: double

Version History

Introduced in R2019a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying 'single' as the output data type, include {coder.Constant('OutputDataType'),coder.Constant('single')} in the -args value of the codegen function. For more information, see the coder.Constant class.

See Also

Functions
`nrPUSCHScramble` | `nrPUSCHDescramble` | `nrPRBS`

nrPUSCHPTRS

Generate PUSCH PT-RS symbols

Syntax

```
sym = nrPUSCHPTRS(carrier,pusch)
sym = nrPUSCHPTRS( ___ ,'OutputDataType',datatype)
```

Description

`sym = nrPUSCHPTRS(carrier,pusch)` returns `sym` in matrix form, which contains phase tracking reference signal (PT-RS) symbols of physical uplink shared channel (PUSCH), as defined in TS 38.211 Section 6.4.1.2.1 [1]. The number of columns in `sym` depends on the transmission scheme and transform precoding. `carrier` specifies the carrier configuration parameters for a specific OFDM numerology and `pusch` specifies the PUSCH configuration parameters.

`sym = nrPUSCHPTRS( ___ ,'OutputDataType',datatype)` specifies the data type of output PT-RS symbols `sym`, in addition to the input arguments in the previous syntax.

Examples

Generate PUSCH PT-RS Symbols for CP-OFDM

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier with 15 kHz subcarrier spacing.

```
carrier = nrCarrierConfig;
```

Create a PUSCH configuration object with 'codebook' based transmission and enable the PT-RS configuration. Set the number of antenna ports to 4, transmitted precoding matrix indicator to 5, frequency density to 4, and resource element offset to '11'. When transform precoding is 0, the waveform type is cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM).

```
pusch = nrPUSCHConfig;
pusch.TransformPrecoding = 0;
pusch.TransmissionScheme = 'codebook';
pusch.NumAntennaPorts = 4;
pusch.TPMI = 5;
pusch.EnablePTRS = 1;
pusch.PTRS.FrequencyDensity = 4;
pusch.PTRS.REOffset = '11';
```

Generate PUSCH PT-RS symbols of data type single.

```
sym = nrPUSCHPTRS(carrier,pusch,'OutputDataType','single')
```

```
sym = 169x4 single matrix

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

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

Generate PUSCH PT-RS Symbols and Indices

Create a carrier configuration object with 30 kHz subcarrier spacing and 5 MHz transmission bandwidth.

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 30;
carrier.NSizeGrid = 11;
```

Create a PUSCH configuration object with intraslot frequency hopping and enable the PT-RS configuration. Set the transform precoding to 1, starting physical resource blocks (PRB) index of the second hop to 3 and PRB set to 0:5. When transform precoding is 1, the waveform type is discrete fourier transform spread orthogonal frequency division multiplexing (DFT-s-OFDM).

```
pusch = nrPUSCHConfig;
pusch.PRBSet = 0:5;
pusch.TransformPrecoding = 1;
pusch.FrequencyHopping = 'intraSlot';
pusch.SecondHopStartPRB = 3;
pusch.EnablePTRS = 1;
```

Create a PUSCH phase tracking reference signal (PT-RS) configuration object with specified properties.

```
ptrs = nrPUSCHPTRSConfig;
ptrs.TimeDensity = 2;
ptrs.NumPTRSSamples = 4;
ptrs.NumPTRSGroups = 8;
ptrs.NID = 750;
```

Assign the PUSCH PT-RS configuration object to PTRS property of PUSCH configuration object.

```
pusch.PTRS = ptrs;
```

Generate PUSCH PT-RS symbols of data type single.

```
sym = nrPUSCHPTRS(carrier,pusch,'OutputDataType','single')
```

```
sym = 192x1 single column vector

   0.7071 + 0.7071i
  -0.7071 + 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
```

```
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
       ⋮
```

Generate PUSCH PT-RS indices in subscript form.

```
ind = nrPUSCHPTRSIndices(carrier,pusch,'IndexStyle','subscript')
```

```
ind = 192x3 uint32 matrix

     1     1     1
     2     1     1
     3     1     1
     4     1     1
    12     1     1
    13     1     1
    14     1     1
    15     1     1
    21     1     1
    22     1     1
       ⋮
```

Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function uses only these properties of the `nrCarrierConfig` object.

- `NCellID`
- `SubcarrierSpacing`
- `CyclicPrefix`
- `NSizeGrid`
- `NStartGrid`
- `NSlot`

`pusch` — PUSCH configuration parameters

`nrPUSCHConfig` object

PUSCH configuration parameters, specified as an `nrPUSCHConfig` object. This function uses only these `nrPUSCHConfig` object properties.

- `NSizeBWP`
- `NStartBWP`
- `NumLayers`

- `MappingType`
- `SymbolAllocation`
- `PRBSet`
- `TransformPrecoding`
- `TransmissionScheme`
- `NumAntennaPorts`
- `TPMI`
- `FrequencyHopping`
- `SecondHopStartPRB`
- `RNTI`
- `EnablePTRS`
- `PTRS`
- `DMRS`

`datatype` — Data type of generated PT-RS symbols
`'double'` (default) | `'single'`

Data type for the generated PT-RS symbols, specified as 'double' or 'single'.

Data Types: `char` | `string`

Output Arguments

`sym` — PT-RS symbols
complex matrix

PT-RS symbols, returned as a complex matrix. The number of columns depends on the `TransmissionScheme` and `TransformPrecoding` properties of `nrPUSCHConfig` object.

The number of columns in `sym` is returned as one these values.

- Number of PT-RS antenna ports configured — When transform precoding is disabled and transmission scheme is non-codebook.
- Number of antenna ports configured — When transform precoding is disabled and transmission scheme is codebook.
- Number of transmission layers — When transform precoding is enabled.

Data Types: `double` | `single`
Complex Number Support: Yes

Version History

Introduced in R2020a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

The `datatype` input argument must be compile-time constant. For example, when specifying `'single'` as the output data type, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrPUSCHDecode` | `nrPUSCH` | `nrPUSCHPTRSIndices` | `nrPUSCHDMRS`

Objects

`nrCarrierConfig` | `nrPUSCHConfig` | `nrPUSCHPTRSConfig` | `nrPUSCHDMRSConfig`

nrPUSCHPTRSIndices

Generate PUSCH PT-RS Indices

Syntax

```
ind = nrPUSCHPTRSIndices(carrier,pusch)
ind = nrPUSCHPTRSIndices(carrier,pusch,Name,Value)
```

Description

`ind = nrPUSCHPTRSIndices(carrier,pusch)` returns `ind` in matrix form, which contains phase tracking reference signal (PT-RS) resource elements (RE) of the physical uplink shared channel (PUSCH), as defined in TS 38.211 Section 6.4.1.2.2 [1]. The number of columns in `ind` depends on the transmission scheme and transform precoding. `carrier` specifies the carrier configuration parameters for a specific OFDM numerology and `pusch` specifies the PUSCH configuration parameters. When you enable transform precoding, the indices are generated relative to the start of the PUSCH allocation.

`ind = nrPUSCHPTRSIndices(carrier,pusch,Name,Value)` specifies output formatting options using one or more name-value pair arguments. Unspecified options take default values.

Examples

Generate PUSCH PT-RS Indices for Codebook-Based Transmission

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier with 15 kHz subcarrier spacing.

```
carrier = nrCarrierConfig;
```

Create a PUSCH configuration object with codebook-based transmission and enable the PT-RS configuration. Set the number of antenna ports to 4 and transform precoding to 0. When transform precoding is 0, the waveform type is cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM).

```
pusch = nrPUSCHConfig;
pusch.TransformPrecoding = 0;
pusch.TransmissionScheme = 'codebook';
pusch.NumAntennaPorts = 4;
pusch.EnablePTRS = 1;
```

Create a PUSCH phase tracking reference signal (PT-RS) configuration object with specified properties.

```
ptrs = nrPUSCHPTRSConfig;
ptrs.TimeDensity = 2;
ptrs.FrequencyDensity = 4;
ptrs.REOffset = '11';
```

Assign the PUSCH PT-RS configuration object to PTRS property of PUSCH configuration object.

```
pusch.PTRS = ptrs;
```

Generate PUSCH PT-RS indices in subscript form

```
ind = nrPUSCHPTRSIndices(carrier,pusch,'IndexStyle','subscript')
```

```
ind = 312x3 uint32 matrix

    21     1     1
    69     1     1
   117     1     1
   165     1     1
   213     1     1
   261     1     1
   309     1     1
   357     1     1
   405     1     1
   453     1     1
      ⋮
```

Generate PUSCH PT-RS Symbols and Indices

Create a carrier configuration object with 30 kHz subcarrier spacing and 5 MHz transmission bandwidth.

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 30;
carrier.NSizeGrid = 11;
```

Create a PUSCH configuration object with intraslot frequency hopping and enable the PT-RS configuration. Set the transform precoding to 1, starting physical resource blocks (PRB) index of the second hop to 3 and PRB set to 0:5. When transform precoding is 1, the waveform type is discrete fourier transform spread orthogonal frequency division multiplexing (DFT-s-OFDM).

```
pusch = nrPUSCHConfig;
pusch.PRBSet = 0:5;
pusch.TransformPrecoding = 1;
pusch.FrequencyHopping = 'intraSlot';
pusch.SecondHopStartPRB = 3;
pusch.EnablePTRS = 1;
```

Create a PUSCH phase tracking reference signal (PT-RS) configuration object with specified properties.

```
ptrs = nrPUSCHPTRSConfig;
ptrs.TimeDensity = 2;
ptrs.NumPTRSSamples = 4;
ptrs.NumPTRSGroups = 8;
ptrs.NID = 750;
```

Assign the PUSCH PT-RS configuration object to PTRS property of PUSCH configuration object.

```
pusch.PTRS = ptrs;
```

Generate PUSCH PT-RS symbols of data type single.

```
sym = nrPUSCHPTRS(carrier,pusch,'OutputDataType','single')
```

```
sym = 192x1 single column vector

   0.7071 + 0.7071i
  -0.7071 + 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
      ⋮
```

Generate PUSCH PT-RS indices in subscript form.

```
ind = nrPUSCHPTRSIndices(carrier,pusch,'IndexStyle','subscript')
```

```
ind = 192x3 uint32 matrix

    1     1     1
    2     1     1
    3     1     1
    4     1     1
   12     1     1
   13     1     1
   14     1     1
   15     1     1
   21     1     1
   22     1     1
      ⋮
```

Input Arguments

carrier — Carrier configuration parameters
nrCarrierConfig object

Carrier configuration parameters for a specific OFDM numerology, specified as an nrCarrierConfig object. This function uses only these properties of the nrCarrierConfig object.

- SubcarrierSpacing
- CyclicPrefix
- NSizeGrid
- NStartGrid
- NSlot

pusch — PUSCH configuration parameters
nrPUSCHConfig object

PUSCH configuration parameters, specified as an nrPUSCHConfig object. This function uses only these nrPUSCHConfig object properties.

- `NSizeBWP`
- `NStartBWP`
- `NumLayers`
- `MappingType`
- `SymbolAllocation`
- `PRBSet`
- `TransformPrecoding`
- `TransmissionScheme`
- `NumAntennaPorts`
- `FrequencyHopping`
- `SecondHopStartPRB`
- `RNTI`
- `DMRS`
- `EnablePTRS`
- `PTRS`

Name-Value Pair Arguments

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

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

Example: `'IndexStyle','subscript','IndexBase','0based'` specifies the RE indexing form and base, respectively, of the output.

`IndexStyle` — RE indexing form
`'index'` (default) | `'subscript'`

RE indexing form, specified as one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

`IndexBase` — RE indexing base
`'1based'` (default) | `'0based'`

RE indexing base, specified as one of these values:

- `'1based'` — The index counting starts from 1.
- `'0based'` — The index counting starts from 0.

Data Types: `char` | `string`

`IndexOrientation` — Resource element indexing orientation
`'carrier'` (default) | `'bwp'`

Resource element indexing orientation, specified as the comma-separated pair consisting of `'IndexOrientation'` and one of these values:

- `'carrier'` — Indices are referenced with respect to carrier grid.
- `'bwp'` — Indices are referenced with respect to bandwidth part.

Dependencies

This property is applicable only when `TransformPrecoding` is set to `0`.

Data Types: `char` | `string`

Output Arguments

ind — PT-RS resource element indices
N-by-*P* matrix | *M*-by-3 matrix

PT-RS resource element indices, returned as one of these values.

- *N*-by-*P* matrix — The function returns this type of value when `'IndexStyle'` is set to `'index'`. The number of columns depends on the `TransmissionScheme` and `TransformPrecoding` properties of `nrPUSCHConfig` object and returned as one these values.
  - Number of PT-RS antenna ports configured — When transform precoding is disabled and transmission scheme is non-codebook.
  - Number of antenna ports configured — When transform precoding is disabled and transmission scheme is codebook.
  - Number of transmission layers — When transform precoding is enabled.
- *M*-by-3 matrix — The function returns this type of value when `'IndexStyle'` is set to `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and number of antennas, respectively.

Depending on the value of `'IndexBase'`, the function returns either 1-based or 0-based indices. Depending on the value of `'IndexOrientation'`, the function returns either carrier oriented indices or BWP oriented indices. This index orientation is applicable only when `TransformPrecoding` is set to `0`. For DFT-s-OFDM, the indices are relative to the shared channel allocation.

Data Types: `uint32`

Version History

Introduced in R2020a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPUSCHDecode` | `nrPUSCH` | `nrPUSCHPTRS` | `nrPUSCHDMRSIndices`

Objects
`nrCarrierConfig` | `nrPUSCHConfig` | `nrPUSCHPTRSConfig` | `nrPUSCHDMRSConfig`

nrPUSCHScramble

Perform PUSCH scrambling

Syntax

```
scrambled = nrPUSCHScramble(cw,nid,rnti)
scrambled = nrPUSCHScramble(cw,nid,rnti,nrapid)
```

Description

`scrambled = nrPUSCHScramble(cw,nid,rnti)` returns a column vector resulting from physical uplink shared channel (PUSCH) scrambling, as defined in TS 38.211 Section 6.3.1.1 [1]. `cw` is an uplink shared channel (UL-SCH) codeword, as described in TS 38.212 Section 6.2.7 [2]. `nid` is the scrambling identity, and `rnti` is the radio network temporary identifier (RNTI) of the user equipment (UE).

`scrambled = nrPUSCHScramble(cw,nid,rnti,nrapid)` specifies random access preamble index `nrapid` to initialize the scrambling sequence for *msgA* on the PUSCH, as defined in Release 16 of TS 38.211 Section 6.3.1.1.

Examples

Perform PUSCH Scrambling

Create a random sequence of binary values corresponding to a codeword containing 5000 bits.

```
cw = randi([0 1],5000,1)
```

```
cw = 5000×1

     1
     1
     0
     1
     1
     0
     0
     1
     1
     1
      ⋮
```

Perform PUSCH scrambling initialized with the specified physical layer cell identity number and RNTI.

```
ncellid = 42;
rnti = 101;
scrambled = nrPUSCHScramble(cw,ncellid,rnti)
```

```
scrambled = 5000x1 logical array
```

```
0
1
1
1
1
0
1
0
0
1
   ⋮
```

Input Arguments

cw — UL-SCH codeword
column vector of integers from –2 to 1

UL-SCH codeword from TS 38.212 Section 6.2.7, specified as a column vector of integers from –2 to 1.

- 0 and 1 represent false and true bit values, respectively.
- –1 and –2 represent *x* and *y* placeholders in the uplink control information (UCI), respectively. For more details, see TS 38.212 Sections 5.3.3.1 and 5.3.3.2.

Data Types: `double` | `int8`

nid — Scrambling identity
integer from 0 to 1023

Scrambling identity, specified as an integer from 0 to 1023. `nid` is higher layer parameter *dataScramblingIdentityPUSCH*, ranging from 0 to 1023, if the higher layer parameter is configured. Otherwise, `nid` is physical layer cell identity number *NCellID*, ranging from 0 to 1007. For more information, see TS 38.211 Section 6.3.1.1.

Data Types: `double`

rnti — RNTI of UE
integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

nrapid — Random access preamble index
`[]` (default) | integer from 0 to 63

Random access preamble index, specified as one of these values.

- `[]` — Use this value to indicate that the scrambling initialization does not consider *msgA* on PUSCH.
- Integer from 0 to 63 — Use this value to initialize the scrambling sequence for *msgA* on PUSCH, as defined in Release 16 of TS 38.211 Section 6.3.1.1.

Data Types: `double`

Output Arguments

`scrambled` — Scrambled UL-SCH codeword
logical column vector

Scrambled UL-SCH codeword, returned as a logical column vector.

Data Types: `logical`

Version History

Introduced in R2019a

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrPUSCHDescramble` | `nrPUSCHPRBS`

nrRateMatchLDPC

Low-density parity-check (LDPC) rate matching

Syntax

```
out = nrRateMatchLDPC(in,outlen,rv,mod,nLayers)
out = nrRateMatchLDPC( ___ ,Nref)
```

Description

`out = nrRateMatchLDPC(in,outlen,rv,mod,nLayers)` returns the rate-matched output of length `outlen` for input data matrix `in`. The input `rv` is the redundancy version, `mod` is the modulation type, and `nLayers` is the number of transmission layers. The internal buffer used for the soft input has no size limits.

`nrRateMatchLDPC` includes the stages of bit selection and interleaving defined for LDPC-encoded data and code block concatenation, as specified in TS 38.212 Sections 5.4.2 and 5.5 [1].

`out = nrRateMatchLDPC( ___ ,Nref)` returns the rate-matched output for a limited soft buffer size `Nref`, in addition to the input arguments in the previous syntax. `Nref` is defined in TS 38.212 Section 5.4.2.1 [1].

Examples

Perform LDPC Rate Matching

Create input data corresponding to two LDPC-encoded code blocks of length 3960.

```
encoded = ones(3960,2);
```

Perform LDPC rate matching of the two code blocks to a vector of length 8000. Use single transmission layer with QPSK modulation and zero redundancy version.

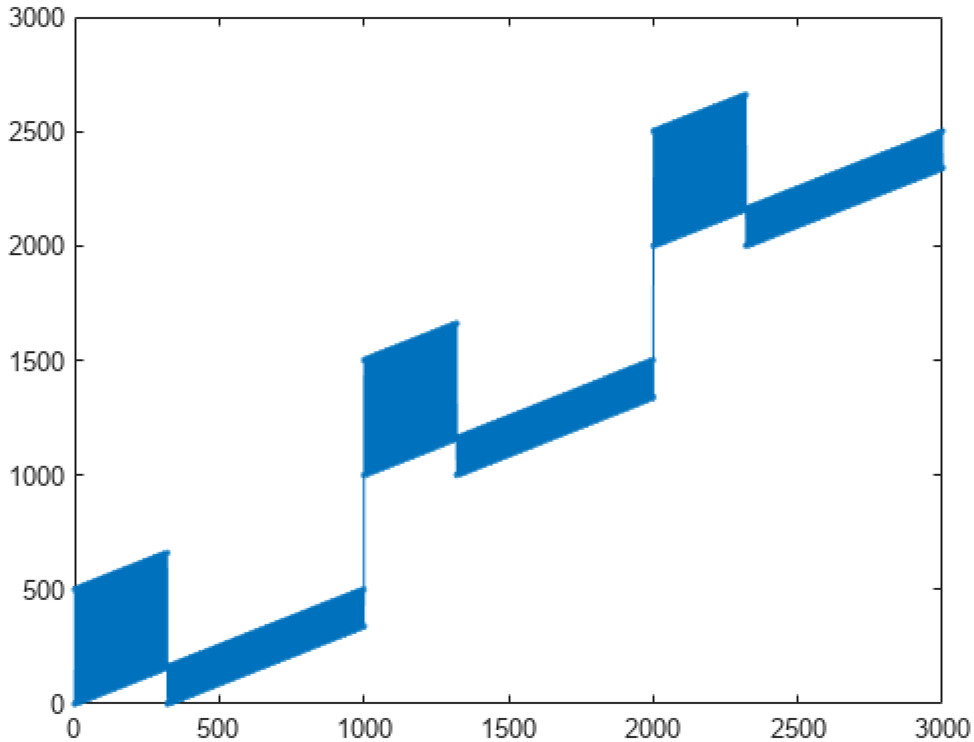
```
rv = 0;
mod = 'QPSK';
nLayers = 1;
outlen = 8000;
ratematched = nrRateMatchLDPC(encoded,outlen,rv,mod,nLayers);
size(ratematched)
```

```
ans = 1×2

        8000           1
```

Plot Output Locations in LDPC Rate-Matched Code Blocks

Create LDPC-encoded input data consisting of integer ramps in separate code blocks. Perform LDPC rate matching of the code blocks to a vector of length 3000. Use single transmission layer with QPSK modulation and zero redundancy version. Plot the locations of the rate-matched output data.

```
encoded = [0 1000 2000] + (1:66*10)';
rv = 0;
mod = 'QPSK';
nLayers = 1;
outlen = 3000;
out = nrRateMatchLDPC(encoded,outlen,rv,mod,nLayers);
plot(out,'.-')
```



Input Arguments

in — LDPC-encoded input data
matrix

LDPC-encoded input data, specified as a matrix. Each column of `in` is a codeword. The number of columns in the input argument `in` is equal to the number of scheduled code blocks in a transport block. Each column is rate-matched separately, and the results are concatenated in `out`.

Data Types: `double` | `int8`

`outlen` — Length of output vector
positive integer

Length of the rate-matched and concatenated output vector, specified as a positive integer. `outlen` is the number of coded bits available for transmission in the transport block, as specified in TS 38.212 Section 5.4.2.1 [1].

The modulation scheme `mod` determines the modulation order Q_m(number of bits used per modulation symbol). If `outlen` is not a multiple of `nLayers` × Q_m, the function sets the length of the output vector to the next multiple of `nLayers` × Q_m.

Data Types: `double`

`rv` — Redundancy version
integer from 0 to 3

Redundancy version, specified as an integer from 0 to 3.

Data Types: `double`

`mod` — Modulation scheme
`'pi/2-BPSK'` | `'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'`

Modulation scheme, specified as `'pi/2-BPSK'`, `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`. This modulation scheme determines the modulation type of the codeword and the number of bits used per modulation symbol.

Modulation Scheme	Number of Bits Per Symbol
`'pi/2-BPSK'`	1
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

Data Types: `char` | `string`

`nLayers` — Number of transmission layers
integer from 1 to 4

Number of transmission layers associated with the transport block, specified as an integer from 1 to 4.

Data Types: `double`

`Nref` — Limited buffer rate matching
positive integer

Limited buffer rate matching, specified as a positive integer. `Nref` is defined in TS 38.212 Section 5.4.2.1.

Data Types: `double`

Output Arguments

`out` — Rate-matched and concatenated code blocks for transport block
vector

Rate-matched and concatenated code blocks for a transport block, returned as a vector of length `outlen`.

Data Types: `double` | `int8`

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.212. "NR; Multiplexing and channel coding." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrCodeBlockSegmentLDPC` | `nrCRCEncode` | `nrRateRecoverLDPC` | `nrLDPCEncode`

nrRateMatchPolar

Polar rate matching

Syntax

```
rm = nrRateMatchPolar(enc,K,E)
rm = nrRateMatchPolar(enc,K,E,ibil)
```

Description

`rm = nrRateMatchPolar(enc,K,E)` returns the rate-matched output of length `E` for the polar-encoded input `enc` and information block length `K`, as specified in TS 38.212 Section 5.4.1 [1]. In this syntax, coded-bit interleaving is disabled. Use this syntax for downlink (DL) configuration.

`rm = nrRateMatchPolar(enc,K,E,ibil)` controls coded-bit interleaving. To enable coded-bit interleaving, set `ibil` to `true`. Use this syntax for uplink (UL) configuration with coded-bit interleaving enabled.

Examples

Perform Polar Rate Matching

Create a polar encoded random block of 512 bits and perform polar rate matching. Specify an information block of 56 bits and a rate-matched output of 864 bits.

```
N = 2^9;
K = 56;
E = 864;
in = randi([0 1],N,1);
out = nrRateMatchPolar(in,K,E)
```

```
out = 864×1

     1
     1
     0
     1
     1
     0
     0
     1
     1
     1
      ⋮
```

Input Arguments

enc — Polar-encoded message
column vector of binary values

Polar-encoded message, specified as a column vector of binary values.

The length of the polar-encoded message, N, is a power of two. For more information, see TS 38.212 Section 5.3.1.

- For DL configuration, $N \le 512$.
- For UL configuration, $N \le 1024$.

Data Types: `double` | `int8`

K — Length of information block in bits
positive integer

Length of information block in bits, specified as a positive integer. `K` includes the CRC bits if applicable.

Data Types: `double`

E — Rate-matched output length in bits
positive integer

Rate-matched output length in bits, specified as a positive integer.

- If 18 ≤ `K` ≤ 25, `E` must be in the range `K` + 3 < `E` ≤ 8192.
- If `K` > 30, `E` must be in the range `K` < `E` ≤ 8192.

Data Types: `double`

ibil — Coded-bit interleaving
`false` (default) | `true`

Coded-bit interleaving, specified as `false` or `true`.

- For DL configuration, specify `false`.
- For UL configuration, specify `true`.

Data Types: `logical`

Output Arguments

rm — Rate-matched output data
column vector of binary values

Rate-matched output data, returned as an `E`-by-1 column vector of binary values. `rm` inherits its data type from the encoded message `enc`.

Data Types: `double` | `int8`

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`nrRateRecoverPolar` | `nrPolarEncode` | `nrDCIEncode` | `nrCRCEncode` | `nrUCIEncode`

Topics

“5G New Radio Polar Coding”

nrRateRecoverLDPC

Low-density parity-check (LDPC) rate recovery

Syntax

```
out = nrRateRecoverLDPC(in,trblklen,R,rv,mod,nLayers)
out = nrRateRecoverLDPC( ___ ,numCB)
out = nrRateRecoverLDPC( ___ ,numCB,Nref)
```

Description

`out = nrRateRecoverLDPC(in,trblklen,R,rv,mod,nLayers)` returns the rate-recovered output representing the LDPC-encoded code blocks for input data vector `in`. The input `trblklen` is the transport block length, `R` is the target code rate, `rv` is the redundancy version, `mod` is the modulation type, and `nLayers` is the number of transmission layers. The internal buffer used for the soft input has no size limits, and the output contains the total number of code blocks.

`nrRateRecoverLDPC` is the inverse of `nrRateMatchLDPC` and performs the inverse of the code block concatenation, bit interleaving, and bit selection stages at the receiver end.

`out = nrRateRecoverLDPC( ___ ,numCB)` specifies the number of code blocks `numCB` to be recovered, in addition to the input arguments in the previous syntax.

`out = nrRateRecoverLDPC( ___ ,numCB,Nref)` returns the rate-recovered output for a limited soft buffer size `Nref` with the specified number of code blocks `numCB` to recover, in addition to the input arguments in the first syntax. `Nref` is defined in TS 38.212 Section 5.4.2.1 [1].

Examples

Perform LDPC Rate Recovery

Create input data of length 4500 corresponding to soft bits. The length of the original transport block is 4000. Perform LDPC rate recovery of the input to one code block. Use single transmission layer with QPSK modulation and zero redundancy version.

```
sbits = ones(4500,1);
trblklen = 4000;
R = 0.5;
rv = 0;
mod = 'QPSK';
nlayers = 1;
numCB = 1;
raterec = nrRateRecoverLDPC(sbits,trblklen,R,rv,mod,nlayers,numCB);
size(raterec)
```

```
ans = 1×2

       12672           1
```

Input Arguments

`in` — Received soft bits before code block desegmentation
vector

Received soft bits before code block desegmentation, specified as a vector.

Data Types: `double` | `single`

`trblklen` — Original transport block length
nonnegative integer

Original transport block length, specified as a nonnegative integer.

Data Types: `double`

`R` — Target code rate
real scalar in the range (0,1)

Target code rate, specified as a real scalar in the range (0,1).

Data Types: `double`

`rv` — Redundancy version
integer from 0 to 3

Redundancy version, specified as an integer from 0 to 3.

Data Types: `double`

`mod` — Modulation scheme
`'pi/2-BPSK'` | `'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'`

Modulation scheme, specified as `'pi/2-BPSK'`, `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`. This modulation scheme determines the modulation type of the codeword and the number of bits used per modulation symbol.

Modulation Scheme	Number of Bits Per Symbol
`'pi/2-BPSK'`	1
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

Data Types: `char` | `string`

`nLayers` — Number of transmission layers
integer from 1 to 4

Number of transmission layers associated with the transport block, specified as an integer from 1 to 4.

Data Types: `double`

numCB — Number of scheduled code block segments
positive integer

Number of scheduled code block segments, specified as a positive integer. numCB is less than or equal to the number of code block segments for a transport block.

Data Types: double

Nref — Limited buffer rate matching
positive integer

Limited buffer rate matching, specified as a positive integer. Nref is defined in TS 38.212 Section 5.4.2.1.

Data Types: double

Output Arguments

out — Rate-recovered scheduled code block segments
matrix

Rate-recovered scheduled code segments, returned as a matrix. The number of rows in out is calculated from trblklen and R. The number of columns in out is equal to numCB, or the total number of code blocks for a transport block. Filler bits are set to Inf to correspond to zeros used during their encoding.

Data Types: double | single

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.212. "NR; Multiplexing and channel coding." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
nrCodeBlockSegmentLDPC | nrCRCDecode | nrRateMatchLDPC | nrLDPCDecode

nrRateRecoverPolar

Polar rate recovering

Syntax

```
rec = nrRateRecoverPolar(llr,K,N)
rec = nrRateRecoverPolar(llr,K,N,ibil)
```

Description

`rec = nrRateRecoverPolar(llr,K,N)` returns the rate-recovered output of length `N` for the soft input `llr` and information block length `K`, as specified in TS 38.212 Section 5.4.1 [1]. In this syntax, coded-bit deinterleaving is disabled. Use this syntax for downlink (DL) configuration.

`rec = nrRateRecoverPolar(llr,K,N,ibil)` controls coded-bit deinterleaving. To enable coded-bit deinterleaving, set `ibil` to `true`. Use this syntax for uplink (UL) configuration with coded-bit deinterleaving enabled.

Examples

Perform Polar Rate Recovery

Create a polar encoded random block of 512 bits and perform polar rate matching using `nrRateMatchPolar`. Perform polar rate recovery. Verify the results are identical to the original polar encoded input.

Specify an information block of 56 bits and an output of 864 bits for rate matching.

```
N = 512;
K = 56;
E = 864;
in = randi([0 1],N,1);
rateMatched = nrRateMatchPolar(in,K,E);
```

Perform rate recovery of the rate-matched data and information block of 56 bits. The length of the rate-recovered output, `N`, is the same as the length of the original polar encoded message.

```
rateRecovered = nrRateRecoverPolar(rateMatched,K,N);
```

Verify that the rate recovered output is identical to the original polar encoded input `in`.

```
isequal(rateRecovered,in)
```

```
ans = logical
   1
```

Input Arguments

`llr` — Log-likelihood ratio value input

column vector of real values

Log-likelihood ratio value input, specified as a column vector of real values. `llr` is the soft-demodulated input of length E, the same length as the rate-matched data vector before modulation.

Data Types: `single` | `double`

`K` — Length of information block in bits

positive integer

Length of information block in bits, specified as a positive integer. `K` includes the CRC bits if applicable.

Data Types: `double`

`N` — Length of polar-encoded message in bits

power of two

Length of polar-encoded message in bits, specified as a power of two.

- `N` ≤ 512 for DL configuration.
- `N` ≤ 1024 for UL configuration.

For more details, see TS 38.212 Section 5.3.1 [1].

Data Types: `double`

`ibil` — Coded-bit deinterleaving

`false` for DL (default) | `true` for UL

Coded-bit deinterleaving, specified as `false` or `true`.

- For DL configuration, specify `false`.
- For UL configuration, specify `true`.

Data Types: `logical`

Output Arguments

`rec` — Rate-recovered output

column vector of real numbers

Rate-recovered output, returned as an `N`-by-1 column vector of real numbers.

Data Types: `single` | `double`

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrRateMatchPolar` | `nrPolarDecode` | `nrCRCDecode` | `nrDCIDecode` | `nrUCIDecode`

Topics
“5G New Radio Polar Coding”

nrResourceGrid

Generate empty carrier slot resource grid

Syntax

```
grid = nrResourceGrid(carrier)
grid = nrResourceGrid(carrier,p)
grid = nrResourceGrid( ___ ,'OutputDataType',datatype)
```

Description

`grid = nrResourceGrid(carrier)` generates an empty carrier slot resource grid for one antenna and the specified carrier configuration parameters.

`grid = nrResourceGrid(carrier,p)` generates an empty carrier slot resource grid for the specified number of antennas.

`grid = nrResourceGrid( ___ ,'OutputDataType',datatype)` specifies the data type of the generated grid in addition to the input arguments in any of the previous syntaxes.

Examples

Generate OFDM Modulated Waveform

Generate a waveform by performing OFDM modulation of a resource array that contains sounding reference signals (SRSs). The resource array spans an entire frame.

Set carrier configuration parameters, specifying a subcarrier spacing of 30 kHz and 24 resource blocks (RBs) in the carrier resource array.

```
carrier = nrCarrierConfig('SubcarrierSpacing',30,'NSizeGrid',24);
```

Configure SRS parameters, setting the slot periodicity to 2 and the offset to zero.

```
srs = nrSRSConfig('SRSPeriod',[2 0]);
```

Get OFDM information for the specified carrier configuration.

```
info = nrOFDMInfo(carrier);
```

Produce the frame resource array by creating and concatenating individual slot resource arrays.

```
grid = [];
for nslot = 0:(info.SlotsPerFrame - 1)
    carrier.NSlot = nslot;
    slotGrid = nrResourceGrid(carrier);
    ind = nrSRSIndices(carrier,srs);
    sym = nrSRS(carrier,srs);
    slotGrid(ind) = sym;
    grid = [grid slotGrid];
end
```

Perform OFDM modulation on the resource array for the specified carrier configuration.

```
[waveform,info] = nrOFDMModulate(carrier,grid);
```

Demodulate OFDM Waveform

Recover a transmitted carrier resource array by demodulating an OFDM waveform.

Set carrier configuration parameters, specifying 106 resource blocks (RBs) in the carrier resource array.

```
carrier = nrCarrierConfig('NSizeGrid',106);
```

Generate physical downlink shared channel (PDSCH) demodulation reference signal (DM-RS) symbols and indices.

```
p = 2;
pdsch = nrPDSCHConfig('NumLayers',p);
sym = nrPDSCHDMRS(carrier,pdsch);
ind = nrPDSCHDMRSIndices(carrier,pdsch);
```

Create a carrier resource array containing the PDSCH DM-RS symbols.

```
txGrid = nrResourceGrid(carrier,p);
txGrid(ind) = sym;
```

Generate OFDM modulated waveform.

```
[txWaveform,~] = nrOFDMModulate(carrier,txGrid);
```

Pass the waveform through a simple 2-by-1 channel.

```
H = [0.6; 0.4];
waveform = txWaveform*H;
```

Recover the carrier resource array by demodulating the received OFDM waveform.

```
grid = nrOFDMDemodulate(carrier,waveform);
```

Generate OFDM Modulated Waveform for Specified Sample Rate

Generate a waveform by performing OFDM modulation of a resource array that contains PDSCH DM-RS symbols.

Set carrier configuration parameters, specifying 106 RBs in the carrier resource array.

```
carrier = nrCarrierConfig('NSizeGrid',106);
```

Configure PDSCH and generate the corresponding symbols and indices.

```
p = 4;
pdsch = nrPDSCHConfig('NumLayers',p);
sym = nrPDSCHDMRS(carrier,pdsch);
ind = nrPDSCHDMRSIndices(carrier,pdsch);
```

Create a carrier resource array and map the PDSCH symbols.

```
grid = nrResourceGrid(carrier,p,'OutputDataType','single');
grid(ind) = sym;
```

Generate OFDM modulated waveform, specifying the sample rate.

```
sr = 1e8;
[waveform,info] = nrOFDMModulate(carrier,grid,'SampleRate',sr);
```

Input Arguments

carrier — Carrier configuration parameters
nrCarrierConfig object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function uses only these properties of the `nrCarrierConfig` object.

- `CyclicPrefix`
- `NSizeGrid`

p — Number of antennas
positive integer

Number of antennas, specified as a positive integer.

Data Types: `double`

datatype — Data type of output symbols
`'double'` (default) | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

grid — Empty carrier slot resource grid
complex-valued array

Empty carrier slot resource array, returned as a complex-valued array of size *K*-by-*L*-by-`p`.

- *K* is the number of subcarriers.
- *L* is the number of OFDM symbols.

Data Types: `single` | `double`
Complex Number Support: Yes

Version History
Introduced in R2020b

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

The `datatype` input argument must be compile-time constant. For example, when specifying `'single'` as the output data type, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrOFDMDemodulate` | `nrOFDMInfo` | `nrOFDMModulate`

Objects

`nrCarrierConfig`

nrSRS

Generate uplink SRS symbols

Syntax

```
[sym,info] = nrSRS(carrier,srs)
[sym,info] = nrSRS(carrier,srs,'OutputDataType',datatype)
```

Description

`[sym,info] = nrSRS(carrier,srs)` returns uplink sounding reference signal (SRS) symbols, as defined in TS 38.211 section 6.4.1.4.2 [1]. The input `carrier` specifies carrier configuration parameters for a specific OFDM numerology. The input `srs` specifies SRS configuration parameters. The function also returns the structure `info`, which contains information about the SRS generation process.

`[sym,info] = nrSRS(carrier,srs,'OutputDataType',datatype)` specifies the data type of the SRS symbols.

Examples

Generate SRS Symbols for Two-Port Transmission

Configure the carrier with default configuration parameters.

```
carrier = nrCarrierConfig;
```

Configure a two-port SRS transmission of 4 OFDM symbols.

```
srs = nrSRSConfig;
srs.NumSRSPorts = 2;
srs.NumSRSSymbols = 4;
```

The SRS must be located in the last six symbols of the slot. Set the time-domain starting position of the SRS to `8` and the bandwidth configuration index to `5`.

```
srs.SymbolStart = 8;
srs.CSRS = 5;
```

Generate SRS symbols for the specified carrier and SRS configuration parameters.

```
[sym,info] = nrSRS(carrier,srs);
```

Verify that the symbols vector contains two columns corresponding to the two-port transmission.

```
size(sym)
```

```
ans = 1×2

   480     2
```

Verify the number of SRS symbols per port.

```
isequal(info.SeqLength*srs.NumSRSSymbols,size(sym,1))
```

```
ans = logical
   1
```

Generate and Map SRS Symbols to Carrier Grid

Configure the SRS and the carrier with default configuration parameters.

```
carrier = nrCarrierConfig;
srs = nrSRSConfig;
```

Generate SRS symbols and indices using the specified carrier and SRS configuration parameters.

```
srsSym = nrSRS(carrier,srs);
srsInd = nrSRSIndices(carrier,srs);
```

Create a carrier grid corresponding to the number of subcarriers, OFDM symbols, and number of antenna ports specified in the configuration objects.

```
K = carrier.NSizeGrid*12;       % Number of subcarriers
L = carrier.SymbolsPerSlot;     % Number of OFDM symbols per slot
P = srs.NumSRSPorts;  % Number of antenna ports
gridSize = [K L P];
```

Initialize the carrier grid for one slot with all zeros.

```
slotGrid = complex(zeros(gridSize));
```

Map the SRS symbols to the carrier grid using the indices.

```
slotGrid(srsInd) = srsSym;
```

Input Arguments

carrier — Carrier configuration parameters
`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object.

srs — SRS configuration parameters
`nrSRSonfig` object

SRS configuration parameters, specified as an `nrSRSConfig` object.

datatype — Data type of output symbols
`'double'` (default) | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

`sym` — SRS symbols

complex column vector | complex matrix | [ ]

SRS symbols, returned as a complex column vector, complex matrix, or empty array. The number of transmission antenna ports specified by the `NumSRSPorts` property of the `srs` input determines the number of columns. The symbols in a column correspond to one antenna port.

The function returns an empty array when the slot number specified by `carrier.NSlot` is not a candidate slot, as defined in TS 38.211 Section 6.4.1.4.4, or when the `srs.SRSPeriod` property is set to `'off'`.

Data Types: `single` | `double`

`info` — Information about SRS generation

structure

Information about the SRS generation, returned as a structure containing these fields:

Fields	Description
`SeqGroup`	Base sequence group number per OFDM symbol (parameter *u* in TS 38.211 Section 6.4.1.4.2)
`NSeq`	Base sequence number per OFDM symbol (parameter *v* in TS 38.211 Section 6.4.1.4.2)
`Alpha`	Reference signal cyclic shift per port (parameter α_i in TS 38.211 Section 6.4.1.4.2)
`SeqLength`	Zadoff-Chu sequence length (parameter $M_{sc,b}^{RS}$ in TS 38.211 Section 6.4.1.4.2)

Version History

Introduced in R2020a

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

The `datatype` input argument must be compile-time constant. For example, when specifying `'single'` as the output data type, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrSRSIndices`

Objects
`nrSRSConfig` | `nrCarrierConfig`

Topics
"NR SRS Configuration"

nrSRSIndices

Generate uplink SRS resource element indices

Syntax

```
[ind,info] = nrSRSIndices(carrier,srs)
[ind,info] = nrSRSIndices(carrier,srs,Name,Value)
```

Description

`[ind,info] = nrSRSIndices(carrier,srs)` returns resource element indices `ind` for the uplink sounding reference signal (SRS), as defined in TS 38.211 section 6.4.1.4.3 [1]. The input `carrier` specifies carrier configuration parameters for a specific OFDM numerology. The input `srs` specifies SRS configuration parameters.

`[ind,info] = nrSRSIndices(carrier,srs,Name,Value)` specifies output formatting options using one or more name-value pair arguments. Unspecified options take default values.

Examples

Generate SRS Indices for Two-Port Transmission

Configure the carrier with default configuration parameters.

```
carrier = nrCarrierConfig;
```

Configure a two-port SRS transmission of 4 OFDM symbols.

```
srs = nrSRSConfig;
srs.NumSRSPorts = 2;
srs.NumSRSSymbols = 4;
```

Set the time-domain starting position of the SRS to `8` and the bandwidth configuration index to `5`.

```
srs.SymbolStart = 8;
srs.CSRS = 5;
```

Generate SRS resource element indices for the specified carrier and SRS configuration parameters.

```
ind = nrSRSIndices(carrier,srs,'IndexStyle','subscript');
```

Verify that the index matrix has three columns corresponding to the [subcarrier, symbol, antenna] subscripts.

```
size(ind)
```

```
ans = 1×2

   960     3
```

Generate and Map SRS Symbols to Carrier Grid

Configure the SRS and the carrier with default configuration parameters.

```
carrier = nrCarrierConfig;
srs = nrSRSConfig;
```

Generate SRS symbols and indices using the specified carrier and SRS configuration parameters.

```
srsSym = nrSRS(carrier,srs);
srsInd = nrSRSIndices(carrier,srs);
```

Create a carrier grid corresponding to the number of subcarriers, OFDM symbols, and number of antenna ports specified in the configuration objects.

```
K = carrier.NSizeGrid*12;       % Number of subcarriers
L = carrier.SymbolsPerSlot;     % Number of OFDM symbols per slot
P = srs.NumSRSPorts;  % Number of antenna ports
gridSize = [K L P];
```

Initialize the carrier grid for one slot with all zeros.

```
slotGrid = complex(zeros(gridSize));
```

Map the SRS symbols to the carrier grid using the indices.

```
slotGrid(srsInd) = srsSym;
```

Input Arguments

`carrier` — Carrier configuration parameters
`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object.

`srs` — SRS configuration parameters
`nrSRSonfig` object

SRS configuration parameters, specified as an `nrSRSConfig` object.

Name-Value Pair Arguments

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

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

Example: `'IndexStyle','subscript','IndexBase','0based'` specifies the indexing style and indexing base of the output.

`IndexStyle` — RE indexing form
`'index'` (default) | `'subscript'`

RE indexing form, specified as one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

IndexBase — RE indexing base
`'1based'` (default) | `'0based'`

RE indexing base, specified as one of these values:

- `'1based'` — The index counting starts from 1.
- `'0based'` — The index counting starts from 0.

Data Types: `char` | `string`

Output Arguments

`ind` — SRS resource element indices
N-by-*P* matrix (default) | *M*-by-3 matrix

SRS resource element indices, returned as one of these values:

- *N*-by-*P* matrix — When '`IndexStyle`' is set to `'index'` and where *P* is the number of antenna ports.
- *M*-by-3 matrix — When '`IndexStyle`' is set to `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and number of antennas, respectively.

The number of rows depends on the SRS configuration specified by `srs`. The `NumSRSPorts` property of `srs` determines the number of antenna ports. Depending on '`IndexBase`', the indices are either 1-based or 0-based.

Data Types: `uint32`

`info` — Information about SRS index generation
structure

Information about the SRS index generation, returned as a structure containing these fields:

Fields	Values	Description
`SubcarrierOffset`	`srs.NumSRSPorts`-by-`srs.NumSRSSymbols` integer matrix	Frequency starting position per antenna port and OFDM symbol (parameter k_0 in TS 38.211 Section 6.4.1.4.3)
`FreqIndex`	(`srs.BSRS` + 1)-by-`srs.NumSRSSymbols` integer matrix	Frequency position index per OFDM symbol (parameter n_b in TS 38.211 Section 6.4.1.4.3, where *b* is an integer from 0 to `srs.BSRS`)
`HoppingOffset`	(`srs.BSRS` – `srs.BHop`)-by-`srs.NumSRSSymbols` integer matrix	Hopping offset per OFDM symbol (parameter F_b in TS 38.211 Section 6.4.1.4.3, where *b* is an integer from `srs.BHop` + 1 to `srs.BSRS`)

Fields	Values	Description
PRBSet	`srs.NRBPerTransmission`-by-`srs.NumSRSSymbols` integer matrix	Resource blocks allocated for SRS per OFDM symbol

Version History

Introduced in R2020a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrSRS`

Objects
`nrSRSConfig` | `nrCarrierConfig`

Topics
"NR SRS Configuration"

nrSSBMeasurements

SSB-based physical layer measurements

Syntax

```
meas = nrSSBMeasurements(ssbGrid,ncellid)
meas = nrSSBMeasurements( ___ ,ibarSSB)
```

Description

`meas = nrSSBMeasurements(ssbGrid,ncellid)` returns physical layer measurements `meas` for synchronization signal block (SSB) `ssbGrid`, as defined in TS 38.215 Sections 5.1.1 and 5.1.3 [1]. The returned structure `meas` contains the reference signal received power (RSRP), received signal strength indicator (RSSI), and reference signal received quality (RSRQ). The input `ssbGrid` specifies a single SS block. The input `ncellid` specifies the physical layer cell identity number.

`meas = nrSSBMeasurements( ___ ,ibarSSB)` also specifies the time-dependent part of the DM-RS scrambling initialization, `ibarSSB`, in addition to the input arguments in the previous syntax. Use this syntax to include the physical broadcast channel demodulation reference signal (PBCH DM-RS) symbols in the RSRP measurement.

Examples

Perform SSB Measurements

Create an SSB for one receive antenna.

```
ssblock = zeros(240,4);
```

Set the cell identity and the corresponding PSS symbols.

```
ncellid = 17;
pssSymbols = nrPSS(ncellid);
ssblock(nrPSSIndices) = pssSymbols;
```

Set the SSS symbols.

```
sssSymbols = nrSSS(ncellid);
ssblock(nrSSSIndices) = sssSymbols;
```

Perform OFDM modulation on the SS block.

```
scs = 15;
initialNSlot = 0;
txWaveform = nrOFDMModulate(ssblock,scs,initialNSlot);
```

Pass the time-domain waveform through an AWGN channel with a signal-to-noise ratio of 10 dB.

```
snr = 10;
rxWaveform = awgn(txWaveform,snr,"measured");
```

Demodulate the received waveform and perform SSB-based measurements.

```
nrb = 20;
ssbGrid = nrOFDMDemodulate(rxWaveform,nrb,scs,initialNSlot);
meas = nrSSBMeasurements(ssbGrid,ncellid)
```

```
meas = struct with fields:
    RSRPPerAntenna: 30.1139
    RSSIPerAntenna: 48.2864
    RSRQPerAntenna: -5.1622
```

Perform SSB Measurements with PBCH DM-RS Symbols

Create an SSB for one receive antenna.

```
ssblock = zeros(240,4);
```

Set the cell identity and the corresponding PSS symbols.

```
ncellid = 17;
pssSymbols = nrPSS(ncellid);
ssblock(nrPSSIndices) = pssSymbols;
```

Set the SSS symbols.

```
sssSymbols = nrSSS(ncellid);
ssblock(nrSSSIndices) = sssSymbols;
```

Create and set the PBCH symbols, using a random codeword, the cell identity, and a scrambling sequence phase of `0`.

```
cw = randi([0 1],864,1);
v = 0;
pbchSymbols = nrPBCH(cw,ncellid,v);
pbchIndices = nrPBCHIndices(ncellid);
ssblock(pbchIndices) = pbchSymbols;
```

Create and set the PBCH DM-RS symbols using the cell identity and the time-dependent part of the DM-RS scrambling initialization.

```
ibarSSB = 0;
dmrsSymbols = nrPBCHDMRS(ncellid,ibarSSB);
dmrsIndices = nrPBCHDMRSIndices(ncellid);
ssblock(dmrsIndices) = dmrsSymbols;
```

Perform OFDM modulation on the SS block.

```
scs = 15;
initialNSlot = 0;
txWaveform = nrOFDMModulate(ssblock,scs,initialNSlot);
```

Apply power scaling to the transmitted waveform.

```
EsdBm = -50;
rxWaveform = txWaveform * sqrt(10^((EsdBm-30)/10));
```

Demodulate the received waveform and perform the SSB-based measurements, including the PBCH DM-RS symbols.

```
nrb = 20;
ssbGrid = nrOFDMDemodulate(rxWaveform,nrb,scs,initialNSlot);
meas = nrSSBMeasurements(ssbGrid,ncellid,ibarSSB)
```

```
meas = struct with fields:
    RSRPPerAntenna: -50
    RSSIPerAntenna: -26.8298
    RSRQPerAntenna: -10.1599
```

Input Arguments

ssbGrid — SSB grid

complex-valued array

SSB grid, specified as a complex-valued array of size 240-by-4-by-*R*, where *R* is the number of receive antennas.

Data Types: `single` | `double`
Complex Number Support: Yes

ncellid — Physical layer cell identity number

integer

Physical layer cell identity number, specified as an integer from 0 to 1007.

Data Types: `double`

ibarSSB — Time-dependent part of DM-RS scrambling initialization

integer from 0 to 7 (default)

Time-dependent part of the DM-RS scrambling initialization, specified as an integer from 0 to 7. Specify this input based on the least significant bits (LSBs) of the SS/PBCH block index and the half-frame number from the SS burst configuration.

- If the number of SS/PBCH blocks per half-frame is 4, `ibarSSB` = $i_{SSB} + 4 \times n_{hf}$, where i_{SSB} is the two LSBs of the SS/PBCH block index (0 to 3). n_{hf} is the half-frame number within the frame (0,1).
- If the number of SS/PBCH blocks per half-frame is 8 or 64, `ibarSSB` is the three LSBs of the SS/PBCH block index (0 to 7).

Data Types: `double`

Output Arguments

meas — SSB-based physical layer measurements

structure

SSB-based physical layer measurements, returned as a structure containing these fields.

Field	Value	Description
RSRPPerAntenna	Real-valued column vector	Vector of RSRP values in dBm relative to 1 milliwatt in 1 ohm
RSSIPerAntenna	Real-valued column vector	Vector of RSSI values in dBm relative to 1 milliwatt in 1 ohm
RSRQPerAntenna	Real-valued column vector	Vector of RSRQ values in dB

The rows of these column vectors correspond to receive antennas.

Data Types: `struct`

Version History

Introduced in R2022b

References

[1] 3GPP TS 38.215. "NR; Physical layer measurements." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrCSIRSMeasurements` | `nrPBCHDMRS` | `nrPBCHIndices` | `nrPSS` | `nrPSSIndices` | `nrSSS` | `nrSSSIndices`

nrSSS

Generate SSS symbols

Syntax

```
sym = nrSSS(ncellid)
sym = nrSSS(ncellid,'OutputDataType',datatype)
```

Description

`sym = nrSSS(ncellid)` returns the secondary synchronization signal (SSS) symbols for the physical layer cell identity number `ncellid`. The function implements TS 38.211 Section 7.4.2.3 [1].

`sym = nrSSS(ncellid,'OutputDataType',datatype)` specifies the data type of the SSS symbol.

Examples

Generate SSS Symbols

Generate the sequence of 127 SSS binary phase shift keying (BPSK) modulation symbols for a given cell identity. The SSS is transmitted in the third symbol of a Synchronization Signal / Physical Broadcast Channel (SS/PBCH) block.

```
ncellid = 17;
sss = nrSSS(ncellid)
```

```
sss = 127×1

    -1
     1
    -1
    -1
    -1
     1
    -1
     1
    -1
     1
      ⋮
```

Input Arguments

`ncellid` — Physical layer cell identity number
integer

Physical layer cell identity number, specified as an integer from 0 to 1007.

Data Types: `double`

`datatype` — Data type of output symbols
`'double'` (default) | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

`sym` — SSS symbols
column vector of real numbers

SSS symbols, returned as a column vector of real numbers.

Data Types: `single` | `double`

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The `datatype` input argument must be compile-time constant. For example, when specifying `'single'` as the output data type, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPBCH` | `nrSSSIndices` | `nrPSS` | `nrPBCHDMRS` | `nrSSBMeasurements`

nrSSSIndices

Generate SSS resource element indices

Syntax

```
ind = nrSSSIndices
ind = nrSSSIndices(Name,Value)
```

Description

`ind = nrSSSIndices` returns the resource element indices for the secondary synchronization signal (SSS), as defined in TS 38.211 Section 7.4.3.1 [1]. The returned indices are 1-based using linear indexing form. This indexing form can directly index the elements of a 240-by-4 matrix corresponding to the Synchronization Signal / Physical Broadcast Channel (SS/PBCH) block. The order of the indices indicates how the SSS modulation symbols are mapped.

`ind = nrSSSIndices(Name,Value)` specifies index formatting options by using one or more name-value pair arguments. Unspecified options take default values.

Examples

Get SSS Resource Element Indices

Generate the 127 resource element indices associated with the SSS within a single SS/PBCH block.

```
ind = nrSSSIndices
```

```
ind = 127x1 uint32 column vector

   537
   538
   539
   540
   541
   542
   543
   544
   545
   546
      ⋮
```

Input Arguments

Name-Value Pair Arguments

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

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

Example: `'IndexStyle','subscript','IndexBase','0based'` specifies nondefault resource element index formatting options.

`IndexStyle` — RE indexing form
`'index'` (default) | `'subscript'`

RE indexing form, specified as one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

`IndexBase` — RE indexing base
`'1based'` (default) | `'0based'`

RE indexing base, specified as one of these values:

- `'1based'` — The index counting starts from 1.
- `'0based'` — The index counting starts from 0.

Data Types: `char` | `string`

Output Arguments

`ind` — SSS resource element indices
column vector (default) | *M*-by-3 matrix

SSS resource element indices, returned as one of these values:

- Column vector — When `'IndexStyle'` is `'index'`.
- *M*-by-3 matrix — When `'IndexStyle'` is `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers and OFDM symbols in an SS/PBCH block, and the number of antennas, respectively.

Depending on `'IndexBase'`, the indices are either 1-based or 0-based.

Data Types: `uint32`

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPBCHDMRSIndices` | `nrSSS` | `nrPSSIndices` | `nrPBCHIndices` | `nrSSBMeasurements`

nrSymbolDemodulate

Demodulate and convert symbols to bits

Syntax

```
out = nrSymbolDemodulate(in,mod)
out = nrSymbolDemodulate(in,mod,nVar)
out = nrSymbolDemodulate(in,mod,'DecisionType',decision)
```

Description

`out = nrSymbolDemodulate(in,mod)` demodulates complex symbols in codeword `in` to soft bits using modulation scheme `mod`. The function implements the inverse of TS 38.211 Section 5.1 [1].

`out = nrSymbolDemodulate(in,mod,nVar)` specifies the noise variance scaling factor for the soft bits.

`out = nrSymbolDemodulate(in,mod,'DecisionType',decision)` specifies the demodulation decision mode by using a name-value pair argument.

Examples

QPSK Demodulation with Soft Decision Mode

Generate a random sequence of binary values of length 40.

```
data = randi([0 1],40,1);
```

Generate modulated symbols using QPSK modulation.

```
modsymb = nrSymbolModulate(data,'QPSK');
```

Perform QPSK demodulation in soft decision mode for a noise variance of 0.1.

```
nVar = 0.1;
recsymb = awgn(modsymb,1/nVar,1,'linear');
out = nrSymbolDemodulate(recsymb,'QPSK',0.1);
```

16-QAM Demodulation with Hard Decision Mode

Generate a random sequence of binary values of length 100.

```
data = randi([0 1],100,1,'int8');
```

Generate modulated symbols using 16-QAM modulation.

```
modsymb = nrSymbolModulate(data,'16QAM');
```

Add a noise to the modulated symbols corresponding to an SNR of 15 dB.

```
recsymb = awgn(modsymb,15);
```

Perform 16-QAM demodulation in hard decision mode.

```
demodbits = nrSymbolDemodulate(recsymb,'16QAM','DecisionType','Hard');
```

Check for bit errors.

```
numErr = biterr(data,demodbits)
```

```
numErr = 1
```

Input Arguments

in — Codeword to demodulate
complex column vector

Codeword to demodulate, specified as a complex column vector.

Data Types: double | single
Complex Number Support: Yes

mod — Modulation scheme
'pi/2-BPSK' | 'BPSK' | 'QPSK' | '16QAM' | '64QAM' | '256QAM'

Modulation scheme, specified as 'pi/2-BPSK', 'BPSK', 'QPSK', '16QAM', '64QAM', or '256QAM'. This modulation scheme determines the modulation type to be performed on the input codeword and the number of bits used per modulation symbol.

Modulation Scheme	Number of Bits Per Symbol
'pi/2-BPSK' 'BPSK'	1
'QPSK'	2
'16QAM'	4
'64QAM'	6
'256QAM'	8

Data Types: char | string

nVar — Noise variance
1e-10 (default) | nonnegative numeric scalar

Noise variance, specified as a nonnegative numeric scalar. The soft bits are scaled with the variance of additive white Gaussian noise (AWGN). The default value corresponds to an SNR of 100 dB, assuming unit signal power. This argument applies only for soft decision mode.

Note The default value assumes the modulator and demodulator are connected back-to-back where the noise variance is zero. To avoid +/-Inf values in the output, the function uses 1e-10 as default value for noise variance. To get appropriate results when the signal is transmitted through a noisy channel, adjust the noise variance accordingly.

Data Types: double

decision — Decision mode
'soft' (default) | 'hard'

Decision mode, specified as 'soft' or 'hard'. The decision mode controls the demodulation type performed on the received symbols.

- 'soft' — Soft decision mode results in a numeric output containing the bitwise approximation to the log-likelihood ratios of the demodulated bits. The output out inherits its data type from the input in.
- 'hard'— Hard decision mode results in a binary output containing groups of bits corresponding to the closest constellation points to the input in. The output out is type-cast to int8.

Data Types: char | string

Output Arguments

out — Demodulated output bits
numeric column vector | binary column vector

Demodulated output bits, returned as a numeric column vector or binary column vector. Demodulation is performed assuming the input constellation power normalization defined in TS 38.211 section 5.1 [1].

Modulation Scheme	Constellation Power Normalization Factor
'pi/2-BPSK' 'BPSK' 'QPSK'	1/sqrt(2)
'16QAM'	1/sqrt(10)
'64QAM'	1/sqrt(42)
'256QAM'	1/sqrt(170)

Each demodulated symbol is mapped to a group of bits corresponding to the number of bits per symbol in the modulation scheme mod. The first bit represents the most significant bit, and the last bit represents the least significant bit. The length of out is the length of the input in multiplied by the number of bits per symbol. The decision mode controls the content and the data type of the demodulated output bits.

Data Types: double | single | int8

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, when specifying soft decision type, include `{coder.Constant('DecisionType'),coder.Constant('soft')}` in the `-args` value of the `codegen` function. For more information, see `coder.Constant`.

See Also

Functions
`nrSymbolModulate` | `nrLayerDemap` | `nrPRBS` | `nrPDCCHDecode` | `nrPDSCHDecode` | `nrPBCHDecode`

nrSymbolModulate

Generate modulated symbols

Syntax

```
out = nrSymbolModulate(in,mod)
out = nrSymbolModulate(in,mod,'OutputDataType',datatype)
```

Description

`out = nrSymbolModulate(in,mod)` maps the bit sequence in codeword `in` to complex modulation symbols using modulation scheme `mod` and returns modulated symbols. The function implements TS 38.211 Section 5.1 [1].

`out = nrSymbolModulate(in,mod,'OutputDataType',datatype)` specifies the data type of the modulated output symbols by using a name-value pair argument. The function uses the specified data type for intermediate computations.

Examples

Generate 16-QAM Modulated Symbols

Generate a random sequence of binary values of length 40. Generate modulated symbols using 16-QAM modulation.

```
data = randi([0 1],40,1);
sym = nrSymbolModulate(data,'16QAM');
```

Generate QPSK-Modulated Symbols

Generate a random sequence of binary values of length 20. Generate modulated symbols using QPSK modulation and specify single-precision data type for the output.

```
data = randi([0 1],20,1,'int8');
sym = nrSymbolModulate(data,'QPSK','OutputDataType','single');
```

Input Arguments

in — Codeword to modulate
column vector of binary values

Codeword to modulate, specified as a column vector of binary values. The codeword length must be a multiple of the number of bits per symbol, specified by the modulation scheme `mod`.

Data Types: `double` | `int8` | `logical`

`mod` — Modulation scheme
`'pi/2-BPSK'` | `'BPSK'` | `'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'`

Modulation scheme, specified as `'pi/2-BPSK'`, `'BPSK'`, `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`. This modulation scheme determines the modulation type to be performed on the input codeword and the number of bits used per modulation symbol.

Modulation Scheme	Number of Bits Per Symbol
`'pi/2-BPSK'` `'BPSK'`	1
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

Data Types: `char` | `string`

`datatype` — Data type of modulated output symbols
`'double'` (default) | `'single'`

Data type of modulated output symbols, specified as `'double'` or `'single'`. The input argument `datatype` determines the data type of the modulated output symbols and the data type that the function uses for intermediate computations.

Data Types: `char` | `string`

Output Arguments

`out` — Modulated output symbols
complex column vector

Modulated output symbols, returned as a complex column vector. The length of `out` is the length of the codeword `in` divided by the number of bits per symbol, specified by the modulation scheme `mod`.

Data Types: `double` | `single`
Complex Number Support: Yes

Version History

Introduced in R2018b

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The `datatype` input argument must be compile-time constant. For example, when specifying `'single'` as the output data type, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrSymbolDemodulate` | `nrLayerMap` | `nrPRBS` | `nrPDCCH` | `nrPDSCH` | `nrPBCH`

nrTBS

Get transport block size

Syntax

```
tbs = nrTBS(mod,nlayers,nPRB,NREPerPRB,tcr)
tbs = nrTBS( ___ ,xOh)
tbs = nrTBS( ___ ,tbScaling)
```

Description

`tbs = nrTBS(mod,nlayers,nPRB,NREPerPRB,tcr)` returns `tbs`, the transport block size (TBS), associated with each codeword for a shared channel transmission, as defined in TS 38.214 Sections 5.1.3.2 and 6.1.4.2. `modulation` is the modulation scheme for each codeword and `nlayers` is the number of transmission layers. `nPRB` is the number of physical resource blocks (PRBs) allocated for the physical shared channel. `NREPerPRB` is the number of resource elements (REs) allocated for the data transmission in the shared channel within one PRB for one slot (without accounting for the additional overhead). `tcr` is the target code rate for each codeword. The additional overhead and scaling factor used for TBS calculation are 0 and 1, respectively.

`tbs = nrTBS( ___ ,xOh)` specifies the additional overhead in addition to the input arguments of the previous syntax. The additional overhead accounts for the overhead from channel state information reference signal (CSI-RS) and control resource set (CORESET). `xOh` controls the number of REs available for the data transmission in the shared channel within one PRB for one slot. The scaling factor used for TBS calculation is 1.

`tbs = nrTBS( ___ ,tbScaling)` specifies the scaling factor in addition to the input arguments of the previous syntax. The function uses `tbScaling` to calculate the intermediate number of information bits, *N_info*, as defined in TS 38.214 Section 5.1.3.2.

Examples

Get TBS for One Codeword

Specify the modulation scheme for one codeword as 16-QAM, the number of transmission layers as 4, and the number of PRBs allocated for the shared channel as 52. Specify the number of REs allocated for the shared channel within one PRB for one slot (without accounting for the additional overhead) as 120. Set the target code rate to 0.48.

```
modulation = '16QAM';
nlayers = 4;
nPRB = 52;
NREPerPRB = 120;
tcr = 0.48;
```

Get the TBS associated with a data transmission having the additional overhead of 6 and scaling factor of 0.25.

```
xOh = 6;
tbScaling = 0.25;
tbs = nrTBS(modulation,nlayers,nPRB,NREPerPRB,tcr,xOh,tbScaling)
```

```
tbs = 11272
```

Get TBS for Two Codewords

Specify the modulation schemes for two codewords as QPSK and 64-QAM. Set the number of transmission layers to 8, and the number of PRBs allocated for the shared channel as 106. Specify the number of REs allocated for the shared channel within one PRB for one slot (without accounting for the additional overhead) as 100.

```
modulation = {'QPSK','64QAM'};
nlayers = 8;
nPRB = 106;
NREPerPRB = 100;
```

Specify the target code rates for two codewords as 0.3701 and 0.4277. Get the payload size of each transport block for a shared channel transmission.

```
tcr = [0.3701 0.4277];
tbs = nrTBS(modulation,nlayers,nPRB,NREPerPRB,tcr)
```

```
tbs = 1×2

       31240      108552
```

Input Arguments

mod — Modulation scheme
'pi/2-BPSK' | 'QPSK' | '16QAM' | '64QAM' | '256QAM' | string scalar | string array | cell array of character vectors

Modulation scheme, specified as `'pi/2-BPSK'`, `'QPSK'`, `'16QAM'`, `'64QAM'`, `'256QAM'`, a string scalar, a string array, or a cell array of character vectors. The modulation scheme for a single codeword is specified as a character vector or a string scalar. If two codewords are present, a single modulation scheme can be applied to both codewords. Alternatively, you can specify different modulation schemes for each codeword by using a string array or a cell array of character vectors.

Modulation Scheme	Number of Bits Per Symbol
'pi/2-BPSK'	1
'QPSK'	2
'16QAM'	4
'64QAM'	6
'256QAM'	8

Data Types: char | string | cell

`nlayers` — Number of transmission layers
integer from 1 to 8

Number of transmission layers, specified as an integer from 1 to 8. For one codeword, use an integer from 1 to 4. For two codewords, use an integer from 5 to 8.

Data Types: `double`

`nPRB` — Number of PRBs
nonnegative integer

Number of PRBs allocated for the physical shared channel, specified as a nonnegative integer. The nominal value of this argument is in the range of 0 to 275.

Data Types: `double`

`NREPerPRB` — Number of REs
nonnegative integer

Number of REs allocated for the data transmission in the physical shared channel within one PRB for one slot, specified as a nonnegative integer. This value excludes any additional overhead.

Data Types: `double`

`tcr` — Target code rate
scalar between 0 and 1 | two-element vector of values between 0 and 1

Target code rate for each codeword, specified as a scalar between 0 and 1 or a two-element vector of values between 0 and 1. Configure two codewords with different target code rates by specifying a two-element vector. Configure two codewords with the same target code rate by specifying a scalar.

Data Types: `double`

`xOh` — Additional overhead
nonnegative integer

Additional overhead, specified as a nonnegative integer. The additional overhead controls the number of REs available for the data transmission in the physical shared channel within one PRB for one slot. The additional overhead accounts for the overhead from channel state information reference signal (CSI-RS) and control resource set (CORESET). The nominal value of the additional overhead is `0`, `6`, `12`, or `18`, provided by the higher-layer parameter *xOverhead* in PDSCH-ServingCellConfig IE or PUSCH-ServingCellConfig IE.

Data Types: `double`

`tbScaling` — Scaling factor
scalar in the range (0, 1] | two-element vector of values in the range (0, 1]

Scaling factor, specified as a scalar in the range (0, 1] or a two-element vector of values in the range (0, 1]. The function uses this value in calculating the intermediate number of information bits, *N_info*, as defined in TS 38.214 Section 5.1.3.2. Configure two codewords with different scaling factors by specifying a two-element vector. Configure two codewords with the same scaling factor by specifying a scalar.

The nominal value of the scaling factor is `0.25`, `0.5`, or `1`, as defined in TS 38.214 Table 5.1.3.2-2.

Data Types: `double`

Output Arguments

`tbs` — Transport block size
nonnegative integer | two-element vector of nonnegative integers

Transport block size associated with each codeword in the shared channel transmission, returned as a nonnegative integer or a two-element vector of nonnegative integers.

The value of `tbs` is `0` in any of these cases.

- When the `NREPerPRB` input is `0`
- When the `nPRB` input is `0`
- When the `NREPerPRB` input is less than the `xOh` input

Data Types: `double`

Version History

Introduced in R2020b

References

[1] 3GPP TS 38.214. "NR; Physical layer procedures for data." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
`nrDLSCH` | `nrULSCH`

nrTimingEstimate

Practical timing estimation

Syntax

```
[offset,mag] = nrTimingEstimate(carrier,waveform,refGrid)
[offset,mag] = nrTimingEstimate(carrier,waveform,refInd,refSym)

[offset,mag] = nrTimingEstimate(waveform,nrb,scs,initialNSlot,refGrid)
[offset,mag] = nrTimingEstimate(waveform,nrb,scs,initialNSlot,refInd,refSym)

[offset,mag] = nrTimingEstimate( ___ ,Name,Value)
```

Description

`[offset,mag] = nrTimingEstimate(carrier,waveform,refGrid)` performs practical timing estimation by cross-correlating input waveform `waveform` with a reference waveform. The function obtains the reference waveform by modulating reference resource grid `refGrid` using orthogonal frequency division multiplexing (OFDM). `carrier` specifies the parameters for the OFDM modulation. The function returns the estimated timing offset, `offset`, and the estimated impulse response magnitude, `mag`, for each receive antenna in the input waveform.

`[offset,mag] = nrTimingEstimate(carrier,waveform,refInd,refSym)` obtains the reference waveform by modulating a reference resource grid containing reference symbols `refSym` at locations `refInd` and using the OFDM modulation specified by `carrier`.

`[offset,mag] = nrTimingEstimate(waveform,nrb,scs,initialNSlot,refGrid)` obtains the reference waveform by modulating reference resource grid `refGrid` and using OFDM modulation that spans `nrb` resource blocks at subcarrier spacing `scs` and initial slot number `initialNSlot`.

`[offset,mag] = nrTimingEstimate(waveform,nrb,scs,initialNSlot,refInd,refSym)` obtains the reference waveform by modulating a resource grid containing reference symbols `refSym` at locations `refInd` and using the OFDM modulation specified by `nrb`, `scs`, and `initialNSlot`.

`[offset,mag] = nrTimingEstimate( ___ ,Name,Value)` specifies options by using one or more name-value pair arguments in addition to the input arguments in any of the previous syntaxes.

Examples

Estimate Timing Offset for TDL-C Channel Transmission

Generate primary synchronization signal (PSS) symbols for physical layer cell identity number 42.

```
ncellid = 42;
pssSym = nrPSS(ncellid);
```

Obtain resource element indices for the PSS.

```
pssInd = nrPSSIndices();
```

Create a resource grid containing the generated PSS symbols.

```
nrb = 20;
scs = 15;
carrier = nrCarrierConfig('NSizeGrid',nrb,'SubcarrierSpacing',scs);
txGrid = nrResourceGrid(carrier);
txGrid(pssInd) = pssSym;
```

OFDM modulate the resource grid.

```
txWaveform = nrOFDMModulate(carrier,txGrid);
```

Transmit the waveform through a TDL-C channel model by using a sample rate of 7.68 MHz.

```
ofdmInfo = nrOFDMInfo(carrier);
channel = nrTDLChannel;
channel.SampleRate = ofdmInfo.SampleRate;
channel.DelayProfile = 'TDL-C';
rxWaveform = channel(txWaveform);
```

Estimate timing offset for the transmission by using the PSS symbols as reference symbols. The OFDM modulation of the reference symbols uses initial slot number 0.

```
initialNSlot = 0;
offset = nrTimingEstimate(rxWaveform,nrb,scs,initialNSlot,pssInd,pssSym);
```

Input Arguments

`carrier` — Carrier configuration parameters
`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. Only these object properties are relevant for this function.

`NSizeGrid` — Number of RBs in carrier resource grid
52 (default) | integer from 1 to 275

Number of RBs in the carrier resource grid, specified as an integer from 1 to 275. The default value of `52` corresponds to the maximum number of RBs of a 10 MHz carrier with 15 kHz SCS.

Data Types: `double`

`SubcarrierSpacing` — Subcarrier spacing in kHz
15 (default) | `30` | `60` | `120` | `240`

Subcarrier spacing in kHz, for all channels and reference signals of the carrier, specified as `15`, `30`, `60`, `120`, or `240`.

Data Types: `double`

`NSlot` — Slot number
`0` (default) | nonnegative integer

Slot number, specified as a nonnegative integer. You can set `NSlot` to a value larger than the number of slots per frame. For example, you can set this value using transmission loop counters in a MATLAB simulation. In this case, you may have to ensure that the property value is modulo the number of slots per frame in a calling code.

Data Types: `double`

`CyclicPrefix` — Cyclic prefix length
`'normal'` (default) | `'extended'`

Cyclic prefix length, specified as one of these options.

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies for only 60 kHz subcarrier spacing.

Data Types: `char` | `string`

`waveform` — Received waveform
T-by-N_R complex matrix

Received waveform, specified as a T-by-N_R complex matrix.

- T is the number of time-domain samples.
- N_R is the number of receive antennas.

Data Types: `single` | `double`
Complex Number Support: Yes

`refGrid` — Predefined reference grid
K-by-N-by-P complex array

Predefined reference grid, specified as a K-by-N-by-P complex array. `refGrid` can span multiple slots.

- K is the number of subcarriers equal to `nrb` × 12.
- N is the number of OFDM symbols in the reference grid.
- P is the number of reference signal ports.

Data Types: `single` | `double`
Complex Number Support: Yes

`refInd` — Reference symbol indices
integer matrix

Reference symbol indices, specified as an integer matrix. The number of rows equals the number of resource elements. You can specify all indices in a single column or distribute them across several columns. The number of elements in `refInd` and `refSym` must be the same but their dimensionality can differ. The function reshapes `refInd` and `refSym` into column vectors before mapping them into a reference grid: `refGrid(refInd(:)) = refSym(:)`.

The elements of `refInd` are 1-based linear indices addressing a K-by-L-by-P resource array.

- K is the number of subcarriers equal to `nrb` × 12.

- *L* is the number of OFDM symbols in a slot. *L* is 12 or 14, depending on the cyclic prefix length specified in the `cpl` input or the `CyclicPrefix` property of the `carrier` input.
- *P* is the number of reference signal ports, inferred from the range of values in `refInd`.

Data Types: `double`

`refSym` — Reference symbols
complex matrix

Reference symbols, specified as a complex matrix. The number of rows equals the number of resource elements. You can specify all symbols in a single column or distribute them across several columns. The number of elements in `refInd` and `refSym` must be the same but their dimensionality can differ. The function reshapes `refInd` and `refSym` into column vectors before mapping them into a reference grid: `refGrid(refInd(:)) = refSym(:)`.

Data Types: `single` | `double`
Complex Number Support: Yes

`nrb` — Number of resource blocks
integer from 1 to 275

Number of resource blocks, specified as an integer from 1 to 275.

Data Types: `double`

`scs` — Subcarrier spacing in kHz
`15` | `30` | `60` | `120` | `240`

Subcarrier spacing in kHz, specified as `15`, `30`, `60`, `120`, or `240`.

Data Types: `double`

`initialNSlot` — Initial slot number
nonnegative integer

Initial slot number, 0-based, specified as a nonnegative integer. The function selects the appropriate cyclic prefix length for the OFDM modulation based on the value of `initialNSlot` modulo the number of slots per subframe.

Data Types: `double`

Name-Value Pair Arguments

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

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

Example: `'SampleRate','1e9'` specifies a sample rate of 1×10^9 Hz.

`CyclicPrefix` — Cyclic prefix length
`'normal'` (default) | `'extended'`

Cyclic prefix length, specified as the comma-separated pair consisting of `'CyclicPrefix'` and one of these values:

- 'normal' — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- 'extended' — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, the extended cyclic prefix length only applies to 60 kHz subcarrier spacing.

Note If you specify the carrier input, use the CyclicPrefix property of the carrier input to specify the cyclic prefix length. You cannot use this name-value pair argument together with the carrier input.

Data Types: char | string

Nfft — Number of FFT points
integer greater than 127 (default depends on other input values) | []

Number of fast Fourier transform (FFT) points, specified as the comma-separated pair consisting of 'Nfft' and a nonnegative integer greater than 127 or []. The value you specify must result in integer-valued cyclic prefix lengths and a maximum occupancy of 100%. The occupancy is defined as the value of $(12 \times N_{RB})$/Nfft, where N_{RB} is the number of resource blocks.

If you do not specify this input, or if you specify 'Nfft',[], the function sets an integer value greater than 127 as a default value for this input. The actual default value depends on other input values.

- If you do not specify the SampleRate input, or if you specify 'SampleRate',[], the function sets Nfft satisfying these conditions.
  - Nfft is an integer power of 2.
  - Nfft results in a maximum occupancy of 85%.
- If you specify the SampleRate input, the function sets Nfft satisfying these conditions.
  - Nfft results in integer-valued cyclic prefix lengths.
  - Nfft maximises the value of gcd (Nfft × *SCS*, SampleRate), where *SCS* is specified by the carrier.SubcarrierSpacing property or the scs input.

For more information, see “Configure OFDM Sample Rate and FFT Size”.

Data Types: double

SampleRate — Waveform sample rate
positive scalar (default depends on other input values) | []

Waveform sample rate, specified as the comma-separated pair consisting of 'SampleRate' and either a positive scalar or [].

If you do not specify this input, or if you specify 'SampleRate',[], then the function sets this input to the value of $N_{fft} \times SCS$.

- N_{fft} is the value of the 'Nfft' input.
- *SCS* is the subcarrier spacing. Depending on the function syntax you use, *SCS* is specified by the carrier.SubcarrierSpacing property or the scs input.

For more information, see “Configure OFDM Sample Rate and FFT Size”.

Data Types: `double`

`CarrierFrequency` — Carrier frequency in Hz
`0` (default) | real number

Carrier frequency in Hz, specified as the comma-separated pair consisting of `'CarrierFrequency'` and a real number. This input corresponds to f_0, defined in TS 38.211 Section 5.4.

Data Types: `double`

Output Arguments

`offset` — Estimated timing offset in samples
nonnegative integer

Estimated timing offset in samples, returned as a nonnegative integer. The number of samples is relative to the first sample of the input waveform `waveform`.

Data Types: `double`

`mag` — Estimated impulse response magnitude
T-by-N_R real matrix

Estimated impulse response magnitude, for each receive antenna in the input waveform `waveform`, returned as a T-by-N_R real matrix.

- T is the number of time-domain samples.
- N_R is the number of receive antennas.

`mag` inherits the data type of the input `waveform`.

Data Types: `single` | `double`

Version History

Introduced in R2019b

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Names and values in name-value pair arguments must be compile-time constants. For example, when specifying extended cyclic prefix, include {coder.Constant('CyclicPrefix'),coder.Constant('extended')} in the -args value of the codegen function. For more information, see the coder.Constant class.

- The input arguments nrb, scs, and initialNSlot must be compile-time constants. For example, include {coder.Constant(nrb)}, {coder.Constant(scs)}, and {coder.Constant(initialNSlot)} in the -args value of the codegen function.
- The 'SampleRate' name-value pair argument cannot be used together with the carrier input.

See Also

Functions
nrChannelEstimate | nrPerfectChannelEstimate | nrPerfectTimingEstimate

Objects
nrCarrierConfig

nrTransformDeprecode

Recover transform deprecoded symbols

Syntax

```
tdpSym = nrTransformDeprecode(modSym,mrb)
```

Description

`tdpSym = nrTransformDeprecode(modSym,mrb)` recovers transform deprecoded symbols for modulation symbols `modSym`, corresponding to the inverse operation of transform precoding from TS 38.211 Section 6.3.1.4 and 6.3.2.6.4 [1]. `mrb` is the number of resource blocks allocated for the physical uplink shared channel (PUSCH), physical uplink control channel (PUCCH) format 3, or PUCCH format 4 transmission. `mrb` determines the length of the subblocks in `modSym` which are transform deprecoded separately.

In the NR uplink, transform deprecoding is used together with CP-OFDM demodulation to demodulate an SC-FDMA (DFT-s-OFDM) waveform. Transform deprecoding applies to only these transmissions:

- After MIMO deprecoding in the PUSCH with single-layer transmission.
- Before symbol demodulation in the PUCCH format 3 transmission.
- Before block-wise despreading in the PUCCH format 4 transmission.

Examples

Recover Transform Deprecoded PUSCH Symbols

Generate a random sequence of binary values corresponding to a PUSCH codeword of 960 bits.

```
cw = randi([0 1],960,1);
```

Perform PUSCH scrambling initialized with the specified physical layer cell identity number and RNTI.

```
ncellid = 42;
rnti = 101;
scrambled = nrPUSCHScramble(cw,ncellid,rnti);
```

Modulate the scrambled PUSCH codeword by using modulation scheme 16-QAM.

```
modulation = '16QAM';
modSym = nrSymbolModulate(scrambled,modulation);
```

Perform layer mapping using a single transmission layer.

```
layeredSym = nrLayerMap(modSym,1);
```

Generate transform precoded symbols by using an allocated PUSCH bandwidth of 2 resource blocks.

```
tpSym = nrTransformPrecode(layeredSym,2);
```

Recover the corresponding transform deprecoded symbols.

```
tdpSym = nrTransformDeprecode(tpSym,2);
```

Input Arguments

`modSym` — Modulation symbols

complex matrix

Modulation symbols, specified as a complex matrix. The number of rows in `modSym` must be a multiple of `mrb`×12. Typically, `modSym` is specified as a column vector, corresponding to single-layer transmission. If `modSym` is a matrix, the `nrTransformDeprecode` function processes each column separately and returns a matrix.

Data Types: `double`

`mrb` — Number of resource blocks

positive integer

Number of resource blocks allocated for the PUSCH, PUCCH format 3, or PUCCH format 4 transmission, specified as a positive integer. `mrb` determines the length of the subblocks in `modSym` which are transform deprecoded separately. Preferred `mrb` values are of the form $2^{\alpha_2} \times 3^{\alpha_3} \times 5^{\alpha_5}$, where α_2, α_3, and α_5 are nonnegative integers, as specified in the standard.

Data Types: `double`

Output Arguments

`tdpSym` — Transform deprecoded symbols

complex matrix

Transform deprecoded symbols, returned as a complex matrix. `tdpSym` inherits the dimensionality of the input `modSym`.

Data Types: `double`
Complex Number Support: Yes

Version History

Introduced in R2019a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions
`nrTransformPrecode`

nrTransformPrecode

Generate transform precoded symbols

Syntax

```
tpSym = nrTransformPrecode(modSym,mrb)
```

Description

`tpSym = nrTransformPrecode(modSym,mrb)` generates transform precoded symbols for modulation symbols `modSym`, as defined in TS 38.211 Section 6.3.1.4 and 6.3.2.6.4 [1]. `mrb` is the number of resource blocks allocated for the physical uplink shared channel (PUSCH), physical uplink control channel (PUCCH) format 3, or PUCCH format 4 transmission. `mrb` determines the length of the subblocks in `modSym` which are transform precoded separately.

In the NR uplink, transform precoding is used together with CP-OFDM modulation to create an SC-FDMA (DFT-s-OFDM) waveform. Transform precoding applies to only these transmissions:

- Before MIMO precoding in the PUSCH with single-layer transmission.
- After symbol modulation in the PUCCH format 3 transmission.
- After block-wise spreading in the PUCCH format 4 transmission.

Examples

Generate Transform Precoded PUSCH Symbols

Generate a random sequence of binary values corresponding to a PUSCH codeword of 960 bits.

```
cw = randi([0 1],960,1);
```

Perform PUSCH scrambling initialized with the specified physical layer cell identity number and RNTI.

```
ncellid = 42;
rnti = 101;
scrambled = nrPUSCHScramble(cw,ncellid,rnti);
```

Modulate the scrambled PUSCH codeword by using modulation scheme 16-QAM.

```
modulation = '16QAM';
modSym = nrSymbolModulate(scrambled,modulation);
```

Perform layer mapping using a single transmission layer.

```
layeredSym = nrLayerMap(modSym,1);
```

Generate transform precoded symbols by using an allocated PUSCH bandwidth of 2 resource blocks.

```
tpSym = nrTransformPrecode(layeredSym,2);
```

Input Arguments

modSym — Modulation symbols
complex matrix

Modulation symbols, specified as a complex matrix. The number of rows in modSym must be a multiple of mrb×12. Typically, modSym is specified as a column vector, corresponding to single-layer transmission. If modSym is a matrix, the nrTransformPrecode function processes each column separately and returns a matrix.

Data Types: double

mrb — Number of resource blocks
positive integer

Number of resource blocks allocated for the PUSCH, PUCCH format 3, or PUCCH format 4 transmission, specified as a positive integer. mrb determines the length of the subblocks in modSym which are transform precoded separately. Preferred mrb values are of the form $2^{\alpha_2} \times 3^{\alpha_3} \times 5^{\alpha_5}$, where α_2, α_3, and α_5 are nonnegative integers, as specified in the standard.

Data Types: double

Output Arguments

tpSym — Transform precoded symbols
complex matrix

Transform precoded symbols, returned as a complex matrix. tpSym inherits the dimensionality of the input modSym.

Data Types: double
Complex Number Support: Yes

Version History

Introduced in R2019a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
nrTransformDeprecode

nrUCIDecode

Decode uplink control information (UCI)

Syntax

```
ucibits = nrUCIDecode(softbits,A)
ucibits = nrUCIDecode(softbits,A,mod)
ucibits = nrUCIDecode( ___ ,'ListLength',L)
[ucibits,err] = nrUCIDecode( ___ )
```

Description

`ucibits = nrUCIDecode(softbits,A)` decodes the input `softbits` and returns the decoded UCI bits of length `A`. The function implements the inverse of the encoding process specified in TS 38.212 Sections 6.3.1.2–6.3.1.5 for the physical uplink control channel (PUCCH) and in Sections 6.3.2.2–6.3.2.5 for the physical uplink shared channel (PUSCH) [1]. The decoding consists of rate recovery, channel decoding, and cyclic redundancy check (CRC) decoding per code block. The particular decoding scheme that the function implements depends on the decoded UCI message length, `A`. For more details, see “Algorithms” on page 1-527.

`ucibits = nrUCIDecode(softbits,A,mod)` also specifies the modulation scheme for the decoding. The specified modulation scheme applies only when the length of `ucibits` is 1 or 2. When not specified, the modulation scheme defaults to QPSK.

`ucibits = nrUCIDecode( ___ ,'ListLength',L)` specifies the list length for polar decoding in addition to the input arguments in any of the previous syntaxes. The specified list length applies only for the successive cancellation list (SCL) decoding when `A` ≥ 12. When not specified, the list length defaults to 8.

`[ucibits,err] = nrUCIDecode( ___ )` also returns an error flag. Use the input arguments in any of the previous syntaxes. A value of `1` in `err` indicates that an error occurred during code block decoding. The `err` output applies only for CRC-based decoding schemes. For more information, see “Algorithms” on page 1-527.

Examples

Decode UCI Codeword

Create a random sequence of binary values corresponding to a UCI message of 32 bits. Encode the message based on the specified length of the rate-matched UCI codeword.

```
A = 32;
E = 120;
uciBits = randi([0 1],A,1);
ucicw   = nrUCIEncode(uciBits,E);
```

Decode the soft bits representing UCI codeword `ucicw`. Set the length of the polar decoding list to 4. The error flag in the output indicates that no errors occurred during code block decoding.

```
L = 4;
[recBits,err] = nrUCIDecode(1-2*ucicw,A,'ListLength',L)
```

```
recBits = 32x1 int8 column vector

   1
   1
   0
   1
   1
   0
   0
   1
   1
   1
      ⋮

err = logical
   0
```

Verify that the transmitted and received message bits are identical.

```
isequal(recBits,uciBits)
```

```
ans = logical
   1
```

Decode UCI Codeword with 16-QAM Modulation Scheme and AWGN

Create a random sequence of binary values corresponding to a two-bit UCI message.

```
K = 2;
uci = randi([0 1],K,1,'int8');
```

Encode the message for the specified length of the rate-matched output and 16-QAM modulation scheme.

```
mod = '16QAM';
E = 4*3;
encUCI = nrUCIEncode(uci,E,mod);
```

Replace placeholders -1 and -2 in the output through scrambling.

```
encUCI(encUCI==-1) = 1;
encUCI(encUCI==-2) = encUCI(find(encUCI==-2)-1);
```

Modulate the encoded UCI message.

```
modOut = nrSymbolModulate(encUCI,mod);
```

Add white Gaussian noise (AWGN) to the modulated symbols using a signal-to-noise ratio of 0 dB.

```
snrdB = 0;
rxSig = awgn(modOut,snrdB);
```

Demodulate the received signal.

```
rxSoftBits = nrSymbolDemodulate(rxSig,mod);
```

Decode the soft bits representing the demodulated UCI codeword.

```
decBits = nrUCIDecode(rxSoftBits,K,mod);
```

Verify that the transmitted and received message bits are identical.

```
isequal(decBits,uci)
```

```
ans = logical
   1
```

Input Arguments

`softbits` — Approximate LLR soft bits
real column vector

Approximate log-likelihood ratio (LLR) soft bits corresponding to encoded UCI bits, specified as a real column vector.

Data Types: `double` | `single`

`A` — Length of decoded UCI message bits
integer from 1 to 1706

Length of decoded UCI message bits, specified as an integer from 1 to 1706.

Data Types: `double`

`mod` — Modulation scheme
`'QPSK'` (default) | `'pi/2-BPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'`

Modulation scheme, specified as `'QPSK'`, `'pi/2-BPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`. This modulation scheme determines the modulation type and number of bits used per modulation symbol, as shown in this table.

Modulation Scheme	Number of Bits Per Symbol
`'pi/2-BPSK'`	1
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

This input argument applies only when the input `A` is `1` or `2`.

Data Types: `char` | `string`

`L` — Length of polar decoding list
8 (default) | power of two

Length of polar decoding list, specified as 8 or a power of two.

Data Types: double

Output Arguments

ucibits — Decoded UCI message bits
A-by-1 column vector of binary values

Decoded UCI message bits, returned as an A-by-1 column vector of binary values.

Data Types: int8

err — Result of UCI code block decoding
logical scalar | logical vector

Result of UCI code block decoding for each code block, returned as a logical scalar or logical vector of length 2. 1 in err indicates that an error has occurred during code block decoding.

Data Types: logical

Algorithms

The particular UCI decoding scheme that nrUCIDecode implements depends on the specified output length A.

A	Deconcatenation	Decoding	CRC Bits
1–11	N/A	Maximum likelihood	N/A
12–19	N/A	CRC-aided SCL	6
20–1706	Depends on A and the length of softbits	CRC-aided SCL	11

Version History
Introduced in R2019a

Polar decoding metric update
Behavior changed in R2020a

In releases R2019b and before, polar decoding uses the exact form of the expression $\log(1 + e^x)$ for internal metric evaluation. Starting in release R2020a, because the exact form leads to numerical instability for high SNR ranges, polar decoding approximates $\log(1 + e^x)$ as 0 for $x < 0$ and as x for $x \geq 0$. This approximation affects the results of the nrUCIDecode function, resulting in a marginal degradation of the BLER performance in a link-level simulation.

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify the list length, include `{coder.Constant('ListLength'),coder.Constant(L)}` in the `-args` value of `codegen`. For more information, see `coder.Constant`.

See Also

Functions
`nrUCIEncode` | `nrRateRecoverPolar` | `nrPolarDecode` | `nrCRCDecode`

nrUCIEncode

Encode uplink control information (UCI)

Syntax

```
codeduci = nrUCIEncode(ucibits,E)
codeduci = nrUCIEncode(ucibits,E,mod)
```

Description

`codeduci = nrUCIEncode(ucibits,E)` encodes UCI message bits `ucibits` and returns concatenated, rate-matched, and encoded UCI blocks of length `E`. The returned blocks can be mapped to either the physical uplink control channel (PUCCH) or the physical uplink shared channel (PUSCH). The function implements TS 38.212 Sections 6.3.1.2–6.3.1.5 for PUCCH and Sections 6.3.2.2–6.3.2.5 for PUSCH [1]. The encoding consists of code block segmentation, cyclic redundancy check (CRC) attachment, channel coding, rate matching, and code block concatenation. The function supports polar encoding and small block lengths. The particular encoding scheme that the function implements depends on the input UCI message length. For more details, see "Algorithms" on page 1-531.

`codeduci = nrUCIEncode(ucibits,E,mod)` also specifies the modulation scheme for the encoding. The specified modulation scheme applies only when the length of `ucibits` is 1 or 2. When not specified, the modulation scheme defaults to QPSK. In the output, –1 and –2 represent the *x* and *y* placeholders, respectively, in Tables 5.3.3.1-1 and 5.3.3.2-1.

Examples

Encode UCI Message Bits

Create a random sequence of binary values corresponding to a UCI message of 32 bits.

```
ucibits = randi([0 1],32,1,'int8');
```

Encode the message for the specified rate-matched output length.

```
E = 120;
codeduci = nrUCIEncode(ucibits,E)
```

```
codeduci = 120x1 int8 column vector

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

⋮

Encode UCI Message Bits for 16-QAM Modulation Scheme

Create a random sequence of binary values corresponding to a two-bit UCI message.

```
ucibits = randi([0 1],2,1,'int8');
```

Encode the message for the specified rate-matched output length and 16-QAM modulation scheme.

```
E = 12;
codeduci = nrUCIEncode(ucibits,E,'16QAM')
```

```
codeduci = 12x1 int8 column vector

    1
    1
   -1
   -1
    0
    1
   -1
   -1
    1
    0
      ⋮
```

Input Arguments

ucibits — UCI message bits
column vector of binary values

UCI message bits, specified as a column vector of binary values. `ucibits` is the information bits encoded before transmission on the PUCCH or PUSCH.

Data Types: `double` | `int8`

E — Length of rate-matched UCI codeword
positive integer

Length of rate-matched UCI codeword, in bits, specified as a positive integer. The valid range of `E` depends on *A*, the length of the input `ucibits`, as shown in this table.

A	Valid Range of E
1	E ≥ *A*
2–11	E > *A*
12–19	E > *A* + 9
20–1706	E > *A* + 11

Data Types: `double`

`mod` — Modulation scheme
'QPSK' (default) | 'pi/2-BPSK' | '16QAM' | '64QAM' | '256QAM'

Modulation scheme, specified as `'QPSK'`, `'pi/2-BPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`. This modulation scheme determines the modulation type and number of bits used per modulation symbol, as shown in this table.

Modulation Scheme	Number of Bits Per Symbol
'pi/2-BPSK'	1
'QPSK'	2
'16QAM'	4
'64QAM'	6
'256QAM'	8

This input argument applies only when the input `ucibits` is one or two bits.

Data Types: char | string

Output Arguments

`codeduci` — Coded UCI bits
E-by-1 column vector of integers from –2 to 1

Coded UCI bits, returned as an E-by-1 column vector of integers from –2 to 1. `codeduci` inherits its data type from the input `ucibits`. Element values –1 and –2 represent the *x* and *y* placeholders, respectively, in Tables 5.3.3.1-1 and 5.3.3.2-1.

Data Types: double | int8

Algorithms

The UCI encoding consists of code block segmentation, cyclic redundancy check (CRC) attachment, channel coding, rate matching, and code block concatenation. The particular UCI encoding scheme that `nrUCIEncode` implements depends on *A*, the length of input `ucibits`.

A	Code Block Segmentation	CRC Bits	Encoding
1	N/A	N/A	Repetition
2	N/A	N/A	Simplex
3–11	N/A	N/A	Reed-Muller
12–19	N/A	6	Parity-check Polar
20–1706	Occurs only when $A \geq 1013$ or when $A \geq 360$ and $E \geq 1088$	11	Polar

Version History

Introduced in R2019a

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrUCIDecode` | `nrPUCCH2` | `nrPUCCH3` | `nrPUCCH4` | `nrPUSCH` | `nrPolarEncode` | `nrRateMatchPolar`

nrULSCHDemultiplex

Perform UL-SCH data and control demultiplexing

Syntax

`[culsch,cack,ccsi1,ccsi2] = nrULSCHDemultiplex(pusch,tcr,tbs,oack,ocsi1,ocsi2,cw)`

Description

`[culsch,cack,ccsi1,ccsi2] = nrULSCHDemultiplex(pusch,tcr,tbs,oack,ocsi1,ocsi2,cw)` returns demultiplexed encoded data vector `culsch`, and encoded uplink control information (UCI) vectors `cack`, `ccsi1`, and `ccsi2` post the demultiplexing of received codeword `cw`. This demultiplexing reverses the processing defined in TS 38.212 Section 6.2.7 [1]. `pusch` is the physical uplink shared channel (PUSCH) configuration. `tcr` is the target code rate. `tbs` is the transport block size for the uplink shared channel (UL-SCH) transmission. `oack` is number of the hybrid automatic repeat request acknowledgment (HARQ-ACK) payload bits. `ocsi1` is the number of channel state information (CSI) part 1 payload bits, and `ocsi2` is the number of CSI part 2 payload bits. `cw` is a column vector of the received log-likelihood ratio (LLR) soft bits.

Examples

Perform UL-SCH Multiplexing and Demultiplexing

Create a PUSCH configuration object with a pi/2-BPSK modulation scheme and no frequency hopping. Set the beta offset factor for the HARQ-ACK to 20, and the beta offset factor for CSI part 1 and CSI part 2 to 6.25 each. Specify the scaling factor as 1, which limits the number of resource elements (REs) assigned for the UCI.

```
pusch = nrPUSCHConfig;
pusch.Modulation = 'pi/2-BPSK';
pusch.FrequencyHopping = 'neither';
pusch.BetaOffsetACK = 20;
pusch.BetaOffsetCSI1 = 6.25;
pusch.BetaOffsetCSI2 = 6.25;
pusch.UCIScaling = 1;
```

Set the target code rate, payload lengths of the UL-SCH data, HARQ-ACK, CSI part 1, and CSI part 2.

```
tcr = 0.5;   % Target code rate
tbs = 3848;  % Payload length of UL-SCH data (transport block size)
oack = 8;    % Payload length of HARQ-ACK
ocsi1 = 88;  % Payload length of CSI part 1
ocsi2 = 100; % Payload length of CSI part 2
```

Get the rate matched lengths of the UL-SCH data, HARQ-ACK, CSI part 1, and CSI part 2.

```
rmInfo = nrULSCHInfo(pusch,tcr,tbs,oack,ocsi1,ocsi2);
```

Create the randomly coded bits of the UL-SCH, HARQ-ACK, CSI part 1, and CSI part 2.

```
culsch = randi([0 1],rmInfo.GULSCH,1);
cack = randi([0 1],rmInfo.GACK,1);
ccsi1 = randi([0 1],rmInfo.GCSI1,1);
ccsi2 = randi([0 1],rmInfo.GCSI2,1);
```

Get the codeword from the randomly coded bits of the UL-SCH and coded bits of UCI types.

```
cw = nrULSCHMultiplex(pusch,tcr,tbs,culsch,cack,ccsi1,ccsi2);
```

Get the demultiplexed UL-SCH and UCI bits from the codeword.

```
[rxculsch,rxcack,rxccsi1,rxccsi2] = nrULSCHDemultiplex(pusch,tcr,tbs,oack,ocsi1,ocsi2,1-2*cw);
```

Verify that the randomly coded bits and demultiplexed coded bits of the UL-SCH and the coded UCI types are identical.

```
isequal(rxculsch<0,culsch)
```

```
ans = logical
   1
```

```
isequal(rxcack<0,cack)
```

```
ans = logical
   1
```

```
isequal(rxccsi1<0,ccsi1)
```

```
ans = logical
   1
```

```
isequal(rxccsi2<0,ccsi2)
```

```
ans = logical
   1
```

Input Arguments

pusch — PUSCH configuration parameters
nrPUSCHConfig object

PUSCH configuration parameters, specified as an `nrPUSCHConfig` object. This function uses only these `nrPUSCHConfig` object properties.

- `Modulation`
- `NumLayers`
- `MappingType`
- `SymbolAllocation`
- `PRBSet`
- `TransformPrecoding`

- FrequencyHopping
- RNTI
- BetaOffsetAck
- BetaOffsetCSI1
- BetaOffsetCSI2
- UCIScaling
- DMRS
- EnablePTRS
- PTRS

tcr — Target code rate
scalar in the range (0, 1)

Target code rate for the codeword in the UL-SCH transmission, specified as a scalar in the range (0, 1).

Data Types: double

tbs — Transport block size
nonnegative integer

Transport block size associated with the codeword in the UL-SCH transmission, specified as a nonnegative integer. A value of 0 indicates no transport block or no UL-SCH transmission on the PUSCH.

Data Types: double

oack — Payload length of HARQ-ACK bits
nonnegative integer

Payload length of the HARQ-ACK bits, specified as a nonnegative integer. A value of 0 indicates no HARQ-ACK transmission.

Data Types: double

ocsi1 — Payload length of CSI part 1 bits
nonnegative integer

Payload length of the CSI part 1 bits, specified as a nonnegative integer. A value of 0 indicates no CSI part 1 transmission.

Data Types: double

ocsi2 — Payload length of CSI part 2 bits
nonnegative integer

Payload length of the CSI part 2 bits, specified as a nonnegative integer. A value of 0 indicates no CSI part 2 transmission. Nominally, the CSI part 2 is present only when CSI part 1 is present.

Data Types: double

cw — Received LLR soft bits
real-valued column vector | [ ]

Received log likelihood ratio (LLR) soft bits, returned as a real-valued column vector or `[]`. The length of `cw` must be equal to the bit capacity of the PUSCH.

Data Types: `single` | `double`

Output Arguments

`culsch` — Coded UL-SCH LLR soft bits
real-valued column vector

Coded UL-SCH LLR soft bits, specified as a real-valued column vector. If the input argument `cw` is empty, then the output argument `culsch` is also empty. The output data type of `culsch` matches that of the input argument `cw`.

Data Types: `single` | `double`

`cack` — Coded HARQ-ACK LLR soft bits
real-valued column vector

Coded HARQ-ACK LLR soft bits, specified as a real-valued column vector. If the input argument `cw` is empty, then the output argument `cack` is also empty. The output data type of `cack` matches that of the input argument `cw`.

Data Types: `single` | `double`

`ccsi1` — Coded CSI part 1 LLR soft bits
real-valued column vector

Coded CSI part 1 LLR soft bits, specified as a real-valued column vector. If the input argument `cw` is empty, then the output argument `ccsi1` is also empty. The output data type of `ccsi1` matches that of the input argument `cw`.

Data Types: `single` | `double`

`ccsi2` — Coded CSI part 2 LLR soft bits
real-valued column vector

Coded CSI part 2 LLR soft bits, specified as a real-valued column vector. If the input argument `cw` is empty, then the output argument `ccsi2` is also empty. The output data type of `ccsi2` matches that of the input argument `cw`.

Data Types: `single` | `double`

Version History
Introduced in R2020b

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.213. “NR; Physical layer procedures for control.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
nrULSCHDecoder | nrPUSCHConfig

Functions
nrULSCHMultiplex | nrUCIDecode | nrPUSCHDecode | nrULSCHInfo | nrRateRecoverLDPC | nrRateRecoverPolar | nrPUSCHDescramble

nrULSCHInfo

Get uplink shared channel (UL-SCH) information

Syntax

```
info = nrULSCHInfo(tbs,tcr)
info = nrULSCHInfo(pusch,tcr,tbs,oack,ocsi1,ocsi2)
```

Description

`info = nrULSCHInfo(tbs,tcr)` returns a structure containing the uplink shared transmission (UL-SCH) information for an input transport block size `tbs` and target code rate `tcr`. The UL-SCH information includes the cyclic redundancy check (CRC) attachment, code block segmentation (CBS), and channel coding. When you use this syntax, the function provides the UL-SCH coding information and does not handle the uplink control information (UCI) multiplexing on the physical uplink shared channel (PUSCH), because the information of PUSCH resources is not known.

`info = nrULSCHInfo(pusch,tcr,tbs,oack,ocsi1,ocsi2)` returns a structure, which contains the UL-SCH information related to the encoding process and UCI multiplexing, for the PUSCH configuration `pusch`, target code rate `tcr`, and transport block size `tbs`. The `oack` input is the hybrid automatic repeat request acknowledgment (HARQ-ACK) payload length. The `ocsi1` input is the channel state information (CSI) part 1 payload length. The `ocsi2` input is the CSI part 2 payload length.

The function performs the multiplexing process on one of these options.

- UL-SCH data and UCI data (HARQ-ACK, CSI part 1, and CSI part 2)
- UCI data (HARQ-ACK, CSI part 1, and CSI part 2) only

Examples

Get UL-SCH Information

Show UL-SCH information before rate matching for an input transport block of length 8456 and target code rate 517/1024. The displayed UL-SCH information shows:

- The transport block has 312 <NULL> filler bits per code block.
- The number of bits per code block, after CBS, is 4576.
- The number of bits per code block, after low-density parity-check (LDPC) coding, is 13,728.

```
tBlkLen = 8456;
targetCodeRate = 517/1024;
nrULSCHInfo(tBlkLen,targetCodeRate)
```

```
ans = struct with fields:
        CRC: '24A'
          L: 24
        BGN: 1
          C: 2
```

```
        Lcb: 24
          F: 312
         Zc: 208
          K: 4576
          N: 13728
     GULSCH: []
       GACK: 0
      GCSI1: 0
      GCSI2: 0
    GACKRvd: 0
      QdACK: 0
     QdCSI1: 0
     QdCSI2: 0
```

Get Bit Capacity Information for UL-SCH Data and UCI

Create a PUSCH object with a default configuration.

```
pusch = nrPUSCHConfig;
```

Set the beta offsets of the UCI types in the PUSCH configuration. Set the UCI scaling factor.

```
pusch.BetaOffsetACK = 10;  % Beta offset factor for HARQ-ACK
pusch.BetaOffsetCSI1 = 10; % Beta offset factor for CSI part 1
pusch.BetaOffsetCSI2 = 10; % Beta offset factor for CSI part 2
pusch.UCIScaling = 1;      % Scaling factor
```

Set the target code rate for the shared channel transmission.

```
tcr = 517/1024; % Target code rate
```

Set the payload lengths of the data, HARQ-ACK, CSI part 1, and CSI part 2.

```
tbs = 8456; % Payload length of UL-SCH data (transport block size)
oack = 6;   % Payload length of HARQ-ACK
ocsi1 = 40; % Payload length of CSI part 1
ocsi2 = 10; % Payload length of CSI part 2
```

Obtain the bit capacity information of the data and UCI.

```
info = nrULSCHInfo(pusch,tcr,tbs,ocsi1,ocsi2,oack)
```

```
info = struct with fields:
        CRC: '24A'
          L: 24
        BGN: 1
          C: 2
        Lcb: 24
          F: 312
         Zc: 208
          K: 4576
          N: 13728
     GULSCH: 15032
       GACK: 906
      GCSI1: 178
```

```
    GCSI2: 108
  GACKRvd: 0
    QdACK: 453
   QdCSI1: 89
   QdCSI2: 54
```

Input Arguments

tbs — Transport block size
nonnegative integer

Transport block size associated with the codeword in the UL-SCH transmission, specified as a nonnegative integer. A value of `0` indicates no transport block or no UL-SCH transmission on the PUSCH.

Data Types: `double`

tcr — Target code rate
scalar in the range (0, 1)

Target code rate for the codeword in the UL-SCH transmission, specified as a scalar in the range (0, 1).

Data Types: `double`

pusch — PUSCH configuration parameters
`nrPUSCHConfig` object

PUSCH configuration parameters, specified as an `nrPUSCHConfig` object. This function uses only these `nrPUSCHConfig` object properties.

- `Modulation`
- `NumLayers`
- `MappingType`
- `SymbolAllocation`
- `PRBSet`
- `TransformPrecoding`
- `FrequencyHopping`
- `NID`
- `BetaOffsetAck`
- `BetaOffsetCSI1`
- `BetaOffsetCSI2`
- `UCIScaling`
- `DMRS`
- `EnablePTRS`
- `PTRS`

oack — Payload length of HARQ-ACK bits
nonnegative integer

Payload length of the HARQ-ACK bits, specified as a nonnegative integer. A value of 0 indicates no HARQ-ACK transmission.

Data Types: double

ocsi1 — Payload length of CSI part 1 bits
nonnegative integer

Payload length of the CSI part 1 bits, specified as a nonnegative integer. A value of 0 indicates no CSI part 1 transmission.

Data Types: double

ocsi2 — Payload length of CSI part 2 bits
nonnegative integer

Payload length of the CSI part 2 bits, specified as a nonnegative integer. A value of 0 indicates no CSI part 2 transmission. Nominally, the CSI part 2 is present only when CSI part 1 is present.

Data Types: double

Output Arguments

info — UL-SCH information
structure

UL-SCH information, returned as a structure containing these fields.

Fields	Values	Description
CRC	'16', '24A'	CRC polynomial selection
L	0, 16, 24	Number of CRC bits
BGN	1, 2	LDPC base graph selection
C	Positive integer	Number of code blocks
Lcb	0, 24	Number of parity bits per code block
F	Nonnegative integer	Number of <NULL> filler bits per code block
Zc	Positive integer	Lifting size selection
K	Nonnegative integer	Number of bits per code block after CBS
N	Nonnegative integer	Number of bits per code block after LDPC coding
GULSCH	Nonnegative integer	Number of coded and rate matched UL-SCH data bits
GACK	Nonnegative integer	Number of coded and rate matched HARQ-ACK bits
GCSI1	Nonnegative integer	Number of coded and rate matched CSI part 1 bits
GCSI2	Nonnegative integer	Number of coded and rate matched CSI part 2 bits
GACKRvd	Nonnegative integer	Number of reserved bits for HARQ-ACK
QdACK	Nonnegative integer	Number of coded HARQ-ACK symbols per layer (Q'_{ACK})
QdCSI1	Nonnegative integer	Number of coded CSI part 1 symbols per layer (Q'_{CSI1})
QdCSI2	Nonnegative integer	Number of coded CSI part 2 symbols per layer (Q'_{CSI2})

Version History

Introduced in R2019a

References

[1] 3GPP TS 38.212. "NR; Multiplexing and channel coding." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
`nrULSCHDecoder` | `nrULSCH` | `nrPUSCHConfig`

Functions
`nrPUSCH` | `nrPUSCHDecode` | `nrULSCHMultiplex` | `nrULSCHDemultiplex`

nrULSCHMultiplex

Perform UL-SCH data and control multiplexing

Syntax

```
[cw,info] = nrULSCHMultiplex(pusch,tcr,tbs,culsch,cack,ccsi1,ccsi2)
```

Description

`[cw,info] = nrULSCHMultiplex(pusch,tcr,tbs,culsch,cack,ccsi1,ccsi2)` returns codeword `cw` by performing uplink shared channel (UL-SCH) multiplexing on the encoded UL-SCH data and the encoded uplink control information (UCI), as defined in TS 38.212 Section 6.2.7 [1]. `pusch` is the physical uplink shared channel configuration (PUSCH). `tcr` is the target code rate. `tbs` is the transport block size for the UL-SCH transmission. `culsch` is the encoded UL-SCH data. `cack`, `ccsi1`, and `ccsi2` are the encoded UCI types.

The function internally calculates the number of reserved bits for the hybrid automatic repeat request acknowledgment (HARQ-ACK) transmission, `GACKRvd` and then compares against the lengths of the coded inputs. This comparison determines the processing of the HARQ-ACK for rate-matching or puncturing.

The length of `cw` equals the bit capacity of the PUSCH. `cw` contains the encoded information up to the bit capacity of PUSCH and ignores any other additional information in the inputs. The output `cw` contains zeros when not enough encoded UL-SCH and encoded UCI (HARQ-ACK, channel state information (CSI) 1, or CSI part 2) data is present to achieve the bit capacity. The function also returns the structure `info`, which contains information about the 1-based locations of each type in the codeword.

Examples

Generate Codeword with Predefined Coded UL-SCH Data and Coded UCI Types

Create a default PUSCH configuration object. Allocate the first 21 resource blocks of the bandwidth part to the PUSCH.

```
pusch = nrPUSCHConfig;
pusch.PRBSet = 0:20;
```

Set the target code rate, payload lengths of the UL-SCH data, HARQ-ACK, CSI part 1, and CSI part 2.

```
tcr = 0.5;  % Target code rate
tbs = 100;  % Payload length of UL-SCH data (transport block size)
oack = 3;   % Payload length of HARQ-ACK
ocsi1 = 10; % Payload length of CSI part 1
ocsi2 = 10; % Payload length of CSI part 2
```

Get the rate matched lengths of the UL-SCH data, HARQ-ACK, CSI part 1, and CSI part 2.

```
rmInfo = nrULSCHInfo(pusch,tcr,tbs,oack,ocsi1,ocsi2);
```

Create the predefined coded bits of the UL-SCH, HARQ-ACK, CSI part 1, and CSI part 2 for the rate-matched output length obtained from the `rmInfo` structure.

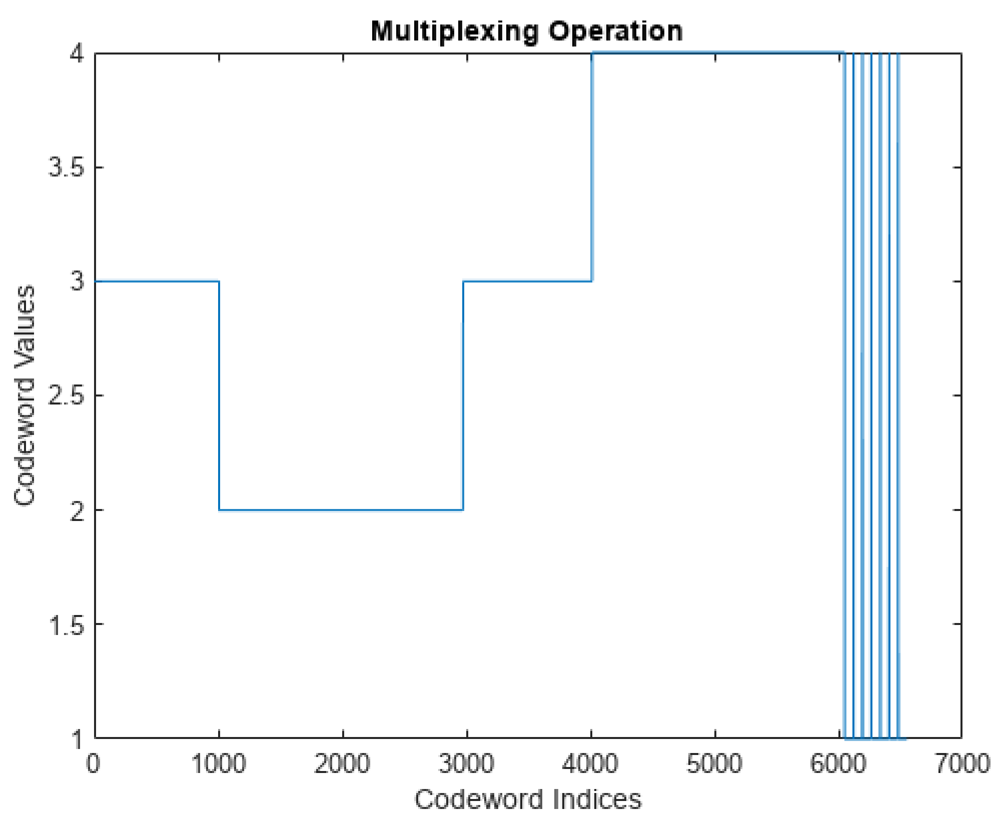
```
culsch = ones(rmInfo.GULSCH,1);
cack = 2*ones(rmInfo.GACK,1);
ccsi1 = 3*ones(rmInfo.GCSI1,1);
ccsi2 = 4*ones(rmInfo.GCSI2,1);
```

Get the codeword from the predefined coded UL-SCH data and coded UCI types.

```
cw = nrULSCHMultiplex(pusch,tcr,tbs,culsch,cack,ccsi1,ccsi2);
```

To see the multiplexing operation, plot the codeword. The codeword starts with the elements of CSI part 1, followed by HARQ-ACK, CSI part 1, CSI part 2, and the mix of UL-SCH data and CSI part 2.

```
plot(cw)
xlabel('Codeword Indices')
ylabel('Codeword Values')
title('Multiplexing Operation')
```



Generate Codeword with Coded UL-SCH Data and Coded UCI

Create a PUSCH configuration object with a pi/2-BPSK modulation scheme and no frequency hopping. Set the beta offset factor for HARQ-ACK to 20, and the beta offset factor for CSI part 1 and

CSI part 2 to 6.25 each. Specify the scaling factor as 0.8, which limits the number of resource elements (REs) assigned for the UCI.

```
pusch = nrPUSCHConfig;
pusch.Modulation = 'pi/2-BPSK';
pusch.FrequencyHopping = 'neither';
pusch.BetaOffsetACK = 20;
pusch.BetaOffsetCSI1 = 6.25;
pusch.BetaOffsetCSI2 = 6.25;
pusch.UCIScaling = 0.8;
```

Set the target code rate, payload lengths of the UL-SCH data, HARQ-ACK, CSI part 1, and CSI part 2.

```
tcr = 0.5;   % Target code rate
tbs = 1032;  % Payload length of UL-SCH data (transport block size)
oack = 8;    % Payload length of HARQ-ACK
ocsi1 = 88;  % Payload length of CSI part 1
ocsi2 = 720; % Payload length of CSI part 2
```

Get the rate matched lengths of the data, HARQ-ACK, CSI part 1, and CSI part 2.

```
rmInfo = nrULSCHInfo(pusch,tcr,tbs,oack,ocsi1,ocsi2);
```

Create the random payload bits for the UL-SCH data, HARQ-ACK, CSI part 1, and CSI part 2.

```
data = randi([0 1],tbs,1);
ack  = randi([0 1],oack,1);
csi1 = randi([0 1],ocsi1,1);
csi2 = randi([0 1],ocsi2,1);
```

Create a UL-SCH encoder System object.

```
encUL = nrULSCH;
```

Load the transport block into the UL-SCH encoder.

```
setTransportBlock(encUL,data);
```

Get the coded bits of length `rmInfo.GULSCH` by calling the encoder.

```
rv = 0; % Redundancy version is 0
culsch = encUL(pusch.Modulation,pusch.NumLayers,rmInfo.GULSCH,rv);
```

Encode the random payload of the HARQ-ACK, CSI part 1, and CSI part 2 for the rate-matched output lengths obtained from the `rmInfo` structure.

```
cack  = nrUCIEncode(ack,rmInfo.GACK,pusch.Modulation);
ccsi1 = nrUCIEncode(csi1,rmInfo.GCSI1,pusch.Modulation);
ccsi2 = nrUCIEncode(csi2,rmInfo.GCSI2,pusch.Modulation);
```

Get the codeword from the coded bits of the UL-SCH and the coded bits of UCI types.

```
[cw,info] = nrULSCHMultiplex(pusch,tcr,tbs,culsch,cack,ccsi1,ccsi2)
```

```
cw = 8112x1 int8 column vector

   1
   0
   1
```

```
   0
   0
   0
   1
   1
   1
   0
      ⋮

info = struct with fields:
    ULSCHIndices: [1622x1 uint32]
      ACKIndices: [1159x1 uint32]
     CSI1Indices: [4482x1 uint32]
     CSI2Indices: [849x1 uint32]
     UCIXIndices: [0x1 uint32]
     UCIYIndices: [0x1 uint32]
```

Input Arguments

pusch — PUSCH configuration parameters
nrPUSCHConfig object

PUSCH configuration parameters, specified as an nrPUSCHConfig object. This function uses only these nrPUSCHConfig object properties.

- Modulation
- NumLayers
- MappingType
- SymbolAllocation
- PRBSet
- TransformPrecoding
- FrequencyHopping
- RNTI
- BetaOffsetAck
- UCIScaling
- DMRS
- EnablePTRS
- PTRS

tcr — Target code rate
scalar in the range (0, 1)

Target code rate for the codeword in the UL-SCH transmission, specified as a scalar in the range (0, 1).

Data Types: double

tbs — Transport block size
nonnegative integer

Transport block size associated with the codeword in the UL-SCH transmission, specified as a nonnegative integer. A value of `0` indicates no transport block or no UL-SCH transmission on the PUSCH.

Data Types: `double`

`culsch` — Coded UL-SCH data bits
binary-valued column vector

Coded UL-SCH data bits, specified as a binary-valued column vector of length `gulsch`. The `gulsch` is the number of coded and rate matched UL-SCH data bits returned in the `info` output argument of the `nrULSCHInfo` function. The encoded UL-SCH bits, `culsch`, is the encoded and rate-matched bits obtained by processing the transport block. A value of `[]` indicates the absence of UL-SCH data transmission. If you specify the `tbs` input argument as `0`, the `culsch` input argument must be empty.

The `gulsch` must be equal to the bit capacity of the data.

Data Types: `int8` | `double`

`cack` — Coded HARQ-ACK bits
real-valued column vector

Coded HARQ-ACK bits, specified as a real-valued column vector of length `gack`. The `gack` is the number of coded and rate matched HARQ-ACK bits returned in the `info` output argument of the `nrULSCHInfo` function. A value of `[]` indicates the absence of HARQ-ACK transmission. The nominal values of the HAQR-ACK bits are `0`, `1`, `-1`, and `-2`.

The `gack` must be a multiple of the product of the number of transmission layers and the modulation order.

Data Types: `int8` | `double`

`ccsi1` — Coded CSI part 1 bits
real-valued column vector

Coded CSI part 1 bits, specified as a real-valued column vector of length `gcsi1`. The `gcsi1` is the number of coded and rate matched CSI part 1 bits returned in the `info` output argument of the `nrULSCHInfo` function. A value of `[]` indicates the absence of the CSI part 1 transmission. The nominal values of the CSI part 1 bits are `0`, `1`, `-1`, and `-2`.

The `gcsi1` must be a multiple of the product of the number of transmission layers and the modulation order.

Data Types: `int8` | `double`

`ccsi2` — Coded CSI part 2 bits
real-valued column vector

Coded CSI part 2 bits, specified as a real-valued column vector of length `gcsi2`. The `gcsi2` is the number of coded and rate matched CSI part 2 bits returned in the `info` output argument of the `nrULSCHInfo` function. A value of `[]` indicates the absence of the CSI part 2 transmission. The nominal values of the CSI part 2 bits are `0`, `1`, `-1`, and `-2`. Nominally, the CSI part 2 is present only when CSI part 1 is present.

The `gcsi2` must be a multiple of the product of the number of transmission layers and the modulation order.

Data Types: int8 | double

Output Arguments

cw — Codeword
real-valued column vector

Codeword for the transmission on the PUSCH, returned as a real-valued column vector. If you provide any of the input arguments as data type int8, the output data type of the codeword is int8. The nominal values of the bits in the codewords are 0, 1, -1, and -2.

The length of cw equals the bit capacity of the PUSCH.

The cw output is an empty value of size 0-by-1 for all of these cases.

- When the PRBSet property of the pusch argument is [].
- When the SymbolAllocation property of the pusch argument is [] or when the number of contiguous OFDM symbols allocated for PUSCH is zero
- When all of the input arguments culsch, cack, ccsi1, and ccsi2 are empty

Data Types: int8 | double

info — Location information
structure

Location information about the 1-based locations of each type in output codeword cw, returned as a structure containing these fields. The output data type of each field is uint32.

Field	Description
ULSCHIndices	Locations of coded UL-SCH bits in codeword
ACKIndices	Locations of coded HARQ-ACK bits in codeword
CSI1Indices	Locations of coded CSI part 1 bits in codeword
CSI2Indices	Locations of coded CSI part 2 bits in codeword
UCIXIndices	Locations of X UCI placeholders in codeword
UCIYIndices	Locations of Y UCI placeholders in codeword

If the returned codeword, cw, is an empty array, each field in this structure is also an empty array.

Version History

Introduced in R2020b

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.213. “NR; Physical layer procedures for control.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
nrULSCH | nrPUSCHConfig

Functions
nrULSCHDemultiplex | nrUCIEncode | nrPUSCH | nrULSCHInfo | nrRateMatchLDPC | nrRateMatchPolar | nrPUSCHScramble

Topics
"NR UCI Multiplexing on PUSCH"

nrWaveformGenerator

Generate 5G NR waveform

Syntax

`[wave,info] = nrWaveformGenerator(cfg)`

`nrWaveformGenerator`

Description

`[wave,info] = nrWaveformGenerator(cfg)` generates 5G NR waveform `wave` for specified configuration `cfg`. The input `cfg` specifies either downlink or uplink configuration parameters for single or multiple subcarrier spacing (SCS) carriers and bandwidth parts (BWPs).

- If `cfg` is an `nrDLCarrierConfig` object, the configuration also specifies the synchronization signal (SS) burst, control resource sets (CORESETs), search spaces, physical downlink control channels (PDCCH) and associated demodulation reference signals (DM-RS), physical downlink shared channels (PDSCH) and associated DM-RS and phase tracking reference signals (PT-RS), and channel-state information reference signals (CSI-RS).
- If `cfg` is an `nrULCarrierConfig` object, the configuration also specifies the physical uplink shared channels (PUSCH) and associated DM-RS and PT-RS, the physical uplink control channels (PUCCH) and associated DM-RS, and the sounding reference signals (SRS).

The function also returns a structure, `info`, containing information about the resource grid and waveform resources.

`nrWaveformGenerator` opens the **5G Waveform Generator** app.

Examples

Configure and Generate Single-User 5G Downlink Waveform

Create an SCS carrier configuration object with the default SCS of 15 kHz and 100 resource blocks.

```
carrier = nrSCSCarrierConfig('NSizeGrid',100);
```

Create a customized BWP configuration object for the SCS carrier.

```
bwp = nrWavegenBWPConfig('NStartBWP',carrier.NStartGrid+10);
```

Create an SS burst configuration object with block pattern Case A.

```
ssb = nrWavegenSSBurstConfig('BlockPattern','Case A');
```

Create a PDCCH configuration object, specifying an aggregation of size two and the fourth candidate for the PDCCH instance.

```
pdcch = nrWavegenPDCCHConfig('AggregationLevel',2,'AllocatedCandidate',4);
```

Create a CORESET configuration object, specifying four frequency resources and a duration of three OFDM symbols.

```
coreset = nrCORESETConfig;
coreset.FrequencyResources = [1 1 1 1];
coreset.Duration = 3;
```

Create a search space set configuration object, specifying two aggregation levels.

```
ss = nrSearchSpaceConfig;
ss.NumCandidates = [8 4 0 0 0];
```

Create a PDSCH configuration object, specifying the modulation scheme and the target code rate. Enable the PDSCH PT-RS.

```
pdsch = nrWavegenPDSCHConfig( ...
    'Modulation','16QAM','TargetCodeRate',658/1024,'EnablePTRS',true);
```

Create a PDSCH DM-RS and a PDSCH PT-RS configuration object with the specified property values.

```
dmrs = nrPDSCHDMRSConfig('DMRSTypeAPosition',3);
pdsch.DMRS = dmrs;
ptrs = nrPDSCHPTRSConfig('TimeDensity',2);
pdsch.PTRS = ptrs;
```

Create a CSI-RS configuration object with the specified property values.

```
csirs = nrWavegenCSIRSConfig('RowNumber',4,'RBOffset',10,'NumRB',10,'SymbolLocations',5);
```

Create a single-user 5G downlink waveform configuration object, specifying the previously defined configurations.

```
cfgDL = nrDLCarrierConfig( ...
    'FrequencyRange','FR1', ...
    'ChannelBandwidth',40, ...
    'NumSubframes',20, ...
    'SCSCarriers',{carrier}, ...
    'BandwidthParts',{bwp}, ...
    'SSBurst',ssb, ...
    'CORESET',{coreset}, ...
    'SearchSpaces',{ss}, ...
    'PDCCH',{pdcch}, ...
    'PDSCH',{pdsch}, ...
    'CSIRS',{csirs});
```

Generate a 5G downlink waveform using the specified configuration.

```
waveform = nrWaveformGenerator(cfgDL);
```

Configure and Generate Multiuser 5G Downlink Waveform

Create two SCS carrier configuration objects with mixed numerologies and custom numbers of resource blocks.

```
carriers = {
    nrSCSCarrierConfig('SubcarrierSpacing',15,'NStartGrid',10,'NSizeGrid',100), ...
```

```
    nrSCSCarrierConfig('SubcarrierSpacing',30,'NStartGrid',0,'NSizeGrid',70)};
```

Create two custom BWP configuration objects, one for each of the carriers.

```
bwp = {
    nrWavegenBWPConfig('BandwidthPartID',1,'SubcarrierSpacing',15,'NStartBWP',10,'NSizeBWP',80),
    nrWavegenBWPConfig('BandwidthPartID',2,'SubcarrierSpacing',30,'NStartBWP',0,'NSizeBWP',60)};
```

Create an SS burst configuration object with block pattern Case A, corresponding to an SCS of 15 kHz.

```
ssb = nrWavegenSSBurstConfig('BlockPattern','Case A');
```

Create two PDCCH configuration objects.

```
pdcch = {
    nrWavegenPDCCHConfig('SearchSpaceID',1,'BandwidthPartID',1,'RNTI',1,'DMRSScramblingID',1), .
    nrWavegenPDCCHConfig('SearchSpaceID',2,'BandwidthPartID',2,'RNTI',2,'DMRSScramblingID',2, ..
        'AggregationLevel',4)};
```

Create two CORESET configuration objects and two search space set configuration objects for the two PDCCH.

```
coreset = {
    nrCORESETConfig('CORESETID',1,'FrequencyResources',[1 1 1 1 1 0 0 0 0 0 1],'Duration',3), ..
    nrCORESETConfig('CORESETID',2,'FrequencyResources',[0 0 0 0 0 0 0 0 1 1])};

ss = {
    nrSearchSpaceConfig('SearchSpaceID',1,'CORESETID',1,'StartSymbolWithinSlot',4), ...
    nrSearchSpaceConfig('SearchSpaceID',2,'CORESETID',2,'NumCandidates',[8 8 4 0 0])};
```

Create two PDSCH configuration objects with mixed modulation schemes.

```
pdsch = {
    nrWavegenPDSCHConfig('BandwidthPartID',1,'Modulation','16QAM','RNTI',1,'NID',1,'PRBSet',10:5
    nrWavegenPDSCHConfig('BandwidthPartID',2,'Modulation','QPSK','RNTI',2,'NID',2, ...
            'PRBSet', 50:59)};
```

Create two CSI-RS configuration objects.

```
 csirs = {
     nrWavegenCSIRSConfig('BandwidthPartID',1,'RowNumber',2,'RBOffset',20), ...
     nrWavegenCSIRSConfig('BandwidthPartID',2,'Density','one','RowNumber',4,'NumRB',10)};
```

Create a multiuser 5G downlink waveform configuration object, specifying the previously defined configurations.

```
cfgDL = nrDLCarrierConfig( ...
    'FrequencyRange','FR1', ...
    'ChannelBandwidth',40, ...
    'NumSubframes',20, ...
    'SCSCarriers',carriers, ...
    'BandwidthParts',bwp, ...
    'SSBurst',ssb, ...
    'CORESET',coreset, ...
    'SearchSpaces',ss, ...
    'PDCCH',pdcch, ...
```

```
    'PDSCH',pdsch, ...
    'CSIRS',csirs);
```

Generate a 5G downlink waveform using the specified configuration.

```
waveform = nrWaveformGenerator(cfgDL);
```

Configure and Generate Single-User 5G Uplink Waveform

Create an SCS carrier configuration object with the default SCS of 15 kHz and 100 resource blocks.

```
carrier = nrSCSCarrierConfig('NSizeGrid',100);
```

Create a customized BWP configuration object for the SCS carrier.

```
bwp = nrWavegenBWPConfig('NStartBWP',carrier.NStartGrid+10);
```

Create a single-user 5G uplink waveform configuration object, specifying the previously defined configurations. In the uplink configuration object, by default, the PUSCH is enabled, while the PUCCH and the SRS are disabled.

```
cfgUL = nrULCarrierConfig( ...
    'FrequencyRange','FR1', ...
    'ChannelBandwidth',40, ...
    'NumSubframes',20, ...
    'SCSCarriers',{carrier}, ...
    'BandwidthParts',{bwp});
```

Generate a 5G uplink waveform using the specified configuration.

```
waveform = nrWaveformGenerator(cfgUL);
```

Configure and Generate Multiuser 5G Uplink Waveform

Create two SCS carrier configuration objects with mixed numerologies and custom numbers of resource blocks.

```
carriers = {
    nrSCSCarrierConfig('SubcarrierSpacing',15,'NStartGrid',10,'NSizeGrid',100), ...
    nrSCSCarrierConfig('SubcarrierSpacing',30,'NStartGrid',0,'NSizeGrid',70)};
```

Create two custom BWP configuration objects, one for each of the carriers.

```
bwp = {
    nrWavegenBWPConfig('BandwidthPartID',0,'SubcarrierSpacing',15,'NStartBWP',30,'NSizeBWP',80),
    nrWavegenBWPConfig('BandwidthPartID',1,'SubcarrierSpacing',30,'NStartBWP',0,'NSizeBWP',60)};
```

Create two PUSCH configuration objects, one for each of the carriers, with mixed modulation schemes.

```
pusch = {
    nrWavegenPUSCHConfig('BandwidthPartID',0,'Modulation','16QAM','SlotAllocation',0:2:9,'PRBSet
    nrWavegenPUSCHConfig('BandwidthPartID',1,'Modulation','QPSK','RNTI',2,'NID',2,'PRBSet',50:59
```

Create a single PUCCH configuration object, only for the second carrier. By default, the PUCCH is enabled in this configuration.

```
pucch = {nrWavegenPUCCH0Config('BandwidthPartID',1,'SlotAllocation',0:9,'PRBSet',2,'DataSourceUCI
```

Create two SRS configuration objects, one for each of the carriers. By default, the SRS is enabled in both configurations.

```
srs = {
    nrWavegenSRSConfig('BandwidthPartID',0,'SlotAllocation',1:2:9,'NumSRSPorts',2), ...
    nrWavegenSRSConfig('BandwidthPartID',1,'FrequencyStart',4)};
```

Create a multiuser 5G uplink waveform configuration object, specifying the previously defined configurations.

```
cfgUL = nrULCarrierConfig( ...
    'FrequencyRange','FR1', ...
    'ChannelBandwidth',40, ...
    'NumSubframes',20, ...
    'SCSCarriers',carriers, ...
    'BandwidthParts',bwp, ...
    'PUSCH',pusch, ...
    'PUCCH',pucch, ...
    'SRS',srs);
```

Generate a 5G uplink waveform using the specified configuration.

```
waveform = nrWaveformGenerator(cfgUL);
```

Input Arguments

cfg — Configuration parameters for 5G NR waveform generation

`nrDLCarrierConfig` object | `nrULCarrierConfig` object

Configuration parameters for 5G NR waveform generation, specified as an `nrDLCarrierConfig` or `nrULCarrierConfig` object.

Output Arguments

wave — Time-domain 5G NR waveform

complex matrix

Time-domain 5G NR waveform, returned as a complex matrix. The number of matrix columns correspond to the number of transmit antennas.

Data Types: `double`

info — Metadata of 5G waveform

structure

Metadata of 5G waveform, returned as a structure with these fields.

ResourceGrids — BWP information

structure

BWP information, returned as a structure with these fields.

<table>
<tr><th>Field</th><th>Value</th><th>Description</th></tr>
<tr><td>ResourceGridBWP</td><td>Complex 2-D or 3-D array</td><td>BWP resource grid</td></tr>
<tr><td>ResourceGridInCarrier</td><td>Complex 2-D or 3-D array</td><td>BWP resource grid in carrier</td></tr>
<tr><td>Info</td><td>Structure array</td><td>Each structure in the array contains these fields.
<table>
<tr><th>Field</th><th>Value</th><th>Description</th></tr>
<tr><td>Nfft</td><td>Positive integer</td><td>Number of fast Fourier transform (FFT) points</td></tr>
<tr><td>SampleRate</td><td>Real number</td><td>Waveform sample rate</td></tr>
<tr><td>CyclicPrefixLengths</td><td>Row vector of positive integers</td><td>Cyclic prefix lengths of each OFDM symbol in a subframe, in samples</td></tr>
<tr><td>SymbolLengths</td><td>Row vector of positive integers</td><td>OFDM symbol lengths, in samples</td></tr>
<tr><td>Windowing</td><td>Positive integer</td><td>Number of time-domain samples over which the function applies raised cosine windowing and overlapping of OFDM symbols</td></tr>
<tr><td>SymbolPhases</td><td>Vector of integers</td><td>Phase compensation of each OFDM symbol, in radians</td></tr>
<tr><td>SymbolsPerSlot</td><td>12 or 14</td><td>Number of OFDM symbols in a slot</td></tr>
<tr><td>SlotsPerSubframe</td><td>1, 2, 4, or 8</td><td>Number of slots in a 1 ms subframe</td></tr>
<tr><td>SlotsPerFrame</td><td>Positive integer</td><td>Number of slots in a 10 ms frame</td></tr>
<tr><td>k0</td><td>Nonnegative integer</td><td>Frequency starting position per antenna port and OFDM symbol</td></tr>
</table>
</td></tr>
</table>

Data Types: `struct`

`WaveformResources` — Information about waveform resources
structure

Information about waveform resources, returned as a structure with these fields.

Field	Value	Description
PDCCH (returned for only downlink waveforms)	1-by-N_{PDCCH} structure array, N_{PDCC} numb confi PDCC input	Each structure in the array contains these fields.

Field	Value	Description
Name	Character array	Name of PDCCH configuration
CDMLengths	Two-element vector of integers	CDM arrangement for reference signals
Resources	1-by-M_{PDCCH} str ar wh M th of slo th sp PD	Each structure in the array contains these fields.

Field	Value	Description
NSlot	Nonnegative integer	Slot number
DCIBits	Binary-valued column vector	Downlink control information (DCI) bits
Codeword	Binary-valued column vector	Encoded DCI codeword
G	Nonnegative integer	Bit capacity of the PDCCH
Gd	Nonnegative integer	Number of resource elements per layer or port
ChannelIndices	Column vector of positive integers	PDCCH indices relative to the associated BWP
ChannelSymbols	Complex column vector	PDCCH symbols
DMRSIndices	Column vector of positive integers	PDCCH DM-RS indices relative to the associated BWP
DMRSSymbols	Complex column vector	PDCCH DM-RS symbols

Field	Value	Description
PDSCH (returned for only downlink waveforms)	1-by-N_{PDSCH} structure array, N_{PDSC} numb confi PDSC	Each structure in the array contains these fields.

Field	Value	Description
Name	Character array	Name of PDSCH configuration
CDMLengths	Two-element integer vector	CDM arrangement for reference signals
Resources	1-by-M_{PDSCH} str ar wh M th of slo th sp PD	Each structure in the array contains these fields.

Fields	Values	
NSlot	Nonnegative integer	Slot number
TransportBlockSize	Nonnegative integer	Size of PDSCH transport block
TransportBlock	Binary-valued column vector	PDSCH transport block
RV	Nonnegative integer	Redundancy version
Codeword	Binary-valued column vector Cell array of two binary-valued column vectors	Codeword(s) from DL-SCH transport channel
G	Nonnegative integer	Bit capacity of the PDSCH. This value is equal to the length of the codeword from the DL-SCH transport channel.
Gd	Nonnegative integer	Number of resource elements per layer or port
ChannelIndices	Column vector of	PDSCH indices relative to the associated BWP

Field	Value	Description

Field	Value	Description

Fields	Values	
	positive integers	
ChannelSymbols	Complex column vector	PDSCH symbols
DMRSIndices	Column vector of positive integers	PDSCH DM-RS indices relative to the associated BWP
DMRSSymbols	Complex column vector	PDSCH DM-RS symbols
DMRSSymbolSet	Vector of nonnegative integers	OFDM symbol locations in a slot containing the DM-RS (0-based)
PTRSIndices	Column vector of positive integers	PDSCH PT-RS indices relative to the associated BWP
PTRSSymbols	Complex column vector	PDSCH PT-RS symbols
PTRSSymbolSet	Vector of nonnegative integers	OFDM symbol locations in a slot containing PT-RS (0-based)

Field	Value	Description
PUSCH (returned for only uplink waveforms)	1-by-N_{PUSCH} structure array, N_{PUSC} numb confi PUSC	Each structure in the array contains these fields.

Field	Value	Description
Name	Character array	Name of PUSCH configuration
CDMLengths	Two-element integer vector	CDM arrangement for reference signals
Resources	1-by-M_{PUSCH} st ar wh M th of slo th sp PU	Each structure in the array contains these fields.

Field	Value	
NSlot	Nonnegative integer	Slot number
TransportBlockSize	Nonnegative integer	Size of PUSCH transport block
TransportBlock	Binary-valued column vector	PUSCH transport block
RV	Nonnegative integer	Redundancy version
Codeword	Binary-valued column vector	Codeword from UL-SCH transport channel
G	Nonnegative integer	Bit capacity of the PUSCH. This value is equal to the length of the codeword from the UL-SCH transport channel.
Gd	Nonnegative integer	Number of resource elements per layer or port
ChannelIndices	Column vector of positive integers	PUSCH indices relative to the associated BWP
ChannelSymbols	Complex column vector	PUSCH symbols
DMRSIndices	Column vector of	PUSCH DM-RS indices relative to the associated BWP

Field	Value	Description
	positive integers	
DMRSSymbols	Complex column vector	PUSCH DM-RS symbols
DMRSSymbolSet	Vector of nonnegative integers	OFDM symbol locations in a slot containing the DM-RS (0-based)
PTRSIndices	Column vector of positive integers	PUSCH PT-RS indices relative to the associated BWP
PTRSSymbols	Complex column vector	PUSCH PT-RS symbols
PTRSSymbolSet	Vector of nonnegative integers	OFDM symbol locations in a slot containing PT-RS (0-based)

<table>
<tr><th>Field</th><th>Value</th><th>Description</th></tr>
<tr><td>PUCCH

(returned for only uplink waveforms)</td><td>1-by-N_{PUCCH} structure array, N_{PUCC} numb confi PUCC</td><td>Each structure in the array contains these fields.
<table>
<tr><th>Field</th><th>Value</th><th>Description</th></tr>
<tr><td>Name</td><td>Character array</td><td>Name of PUCCH configuration</td></tr>
<tr><td>Format</td><td>Integer from 0 to 4 or []</td><td>PUCCH format</td></tr>
<tr><td>CDMLength</td><td>Two-element integer vector</td><td>CDM arrangement for reference signals</td></tr>
<tr><td>Resources</td><td>1-by-M_{PUCCH} str ar wh M th of slo th sp PU</td><td>Each structure in the array contains these fields.
<table>
<tr><th>Field</th><th>Value</th><th></th></tr>
<tr><td>NSlot</td><td>Nonnegative integer</td><td>Slot number</td></tr>
<tr><td>SRBit (returned only for format 0)</td><td>Binary-valued scalar</td><td>Scheduling request (SR) bit</td></tr>
<tr><td>UCIBits</td><td>Binary-valued column vector</td><td>Uplink control information (UCI) bits</td></tr>
<tr><td>UCI2Bits (returned only for formats 3 and 4)</td><td>Binary-valued column vector</td><td>UCI part 2 bits</td></tr>
<tr><td>Codeword</td><td>Binary-valued column vector

Two-element cell array of binary-valued column vectors</td><td>Codeword containing the UCI bits</td></tr>
</table>
</td></tr>
</table>
</td></tr>
</table>

Field	Value	Description
	(applies to only format 0)	
G	Nonnegative integer	Bit capacity of the PUCCH. This value is equal to the length of the codeword.
Gd	Nonnegative integer	Symbol capacity of the PUCCH
ChannelIndices	Column vector of positive integers	PUCCH indices relative to the associated BWP
ChannelSymbols	Complex column vector	PUCCH symbols
DMRSIndices (returned only for formats 1, 2, 3, and 4)	Column vector of positive integers	PUCCH DM-RS indices relative to the associated BWP
DMRSSymbols (returned only for formats 1, 2, 3, and 4)	Complex column vector	PUCCH DM-RS symbols

<table>
<tr><th>Field</th><th>Value</th><th>Description</th></tr>
<tr><td>SRS

(returned for only uplink waveforms)</td><td>1-by-N_{SRS} structure array, N_{SRS} i numb confi SRS i</td><td>Each structure in the array contains these fields.
<table>
<tr><th>Field</th><th>Value</th><th>Description</th></tr>
<tr><td>Name</td><td>Character array</td><td>Name of SRS configuration</td></tr>
<tr><td>Resources</td><td>1-by-M_{SRS} structure ar wh is nu all slo th sp SR</td><td>Each structure in the array contains these fields.
<table>
<tr><th>Field</th><th>Value</th><th></th></tr>
<tr><td>NSlot</td><td>Nonnegative integer</td><td>Slot number</td></tr>
<tr><td>SignalIndices</td><td>Column vector of positive integers</td><td>SRS indices relative to the associated BWP</td></tr>
<tr><td>SignalSymbols</td><td>Complex column vector</td><td>SRS symbols</td></tr>
</table></td></tr>
</table></td></tr>
</table>

Data Types: `struct`

Data Types: `struct`

Version History

Introduced in R2020b

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

5G Downlink Waveform Generation

When the `cfg` input is an `nrDLCarrierConfig` object, these limitations apply.

- The `cfg.WindowingPercent` property must be set to `0` or `[]`.
- The `cfg.CarrierFrequency` property must be set to `0`.
- The `cfg.SampleRate` property must be set to `[]` or a value equal to `cfg.BWP.SubcarrierSpacing` times the FFT size. The FFT size must be a power of two in the range from 128 to FFT_{MAX}, where:
  - FFT_{MAX} = 4096 × 60 ÷ `SubcarrierSpacing` for FR1
  - FFT_{MAX} = 4096 × 240 ÷ `SubcarrierSpacing` for FR2

- The `cfg.PDCCH.DataSource`, `cfg.PDSCH.DataSource`, and `cfg.SSBurst.DataSource` properties cannot be set using a random seed. Set these properties by using the binary vector format or by using one of the predefined character vectors.
- The `info.WaveformResources.PDSCH.Resources.Codeword` output is always a cell array. When only one codeword exists, the second cell element is the empty array (for example, `{[1 0 ...],[]}`).

5G Uplink Waveform Generation

When the `cfg` input is an `nrULCarrierConfig` object, these limitations apply.

- The `cfg.WindowingPercent` property must be set to `0` or `[]`.
- The `cfg.CarrierFrequency` property must be set to `0`.
- The `cfg.SampleRate` property must be set to `[]` or a value equal to `cfg.BWP.SubcarrierSpacing` times the FFT size. The FFT size must be a power of two in the range from 128 to FFT_{MAX}, where:
  - FFT_{MAX} = 4096 × 60 ÷ `SubcarrierSpacing` for FR1
  - FFT_{MAX} = 4096 × 240 ÷ `SubcarrierSpacing` for FR2
- The following properties cannot be set using a random seed. Set these properties by using the binary vector format or by using one of the predefined character vectors.
  - `cfg.PUSCH.DataSource`
  - `cfg.PUSCH.DataSourceACK`
  - `cfg.PUSCH.DataSourceCSI1`
  - `cfg.PUSCH.DataSourceCSI2`
  - `cfg.PUSCH.DataSourceCGUCI`
  - `cfg.PUCCH.DataSourceSR` (applies to only PUCCH format 0)
  - `cfg.PUCCH.DataSourceUCI` (applies to all PUCCH formats)
  - `cfg.PUCCH.DataSourceUCI2` (applies to only PUCCH formats 3 and 4)
- The `info.WaveformResources.PUSCH.Resources.Codeword` and `info.WaveformResources.PUCCH.Resources.Codeword` (for formats 1, 2, 3, and 4) outputs are always returned as a cell array, where the second cell element is the empty array (for example, `{[1 0 ...],[]}`).
- The `info.WaveformResources.PUCCH.Resources.SRBit` output is always returned as an empty array for formats 1, 2, 3, and 4.
- The `info.WaveformResources.PUCCH.Resources.UCI2Bits` output is always returned as an empty array for formats 0, 1, and 2.
- The `info.WaveformResources.PUCCH.Resources.DMRSIndices` and `info.WaveformResources.PUCCH.Resources.DMRSSymbols` outputs are always returned as an empty array for format 0.

See Also

Objects
`nrDLCarrierConfig` | `nrULCarrierConfig`

Topics

"5G NR Downlink Vector Waveform Generation"
"5G NR Uplink Vector Waveform Generation"

read

Read next protocol packet from PCAP file

Syntax

```
protocolPacket = read(pcap)
protocolPacket = read(pcap,packetFilter)
```

Description

`protocolPacket = read(pcap)` reads the next protocol packet from the packet capture (PCAP) file specified by the input PCAP file reader and returns the decoded packet.

`protocolPacket = read(pcap,packetFilter)` returns only the decoded packets that match the specified filter, `packetFilter`.

Examples

Read eCPRI Packets in Streaming Mode from PCAP File

Create a PCAP file reader object, specifying the name of a PCAP file.

```
pcapReaderObj = pcapReader('ethernetSamplePackets.pcap');
```

Create a filter for the message types of eCPRI packets.

```
filterString = ['ecpri.MessageType == IQData || ecpri.MessageType == BitSequence ' ...
    '|| ecpri.MessageType == RemoteReset'];
```

In streaming mode, read the eCPRI packets that match the specified filter to the MATLAB® workspace.

```
for packetCount = 1:3
    ecpriFilteredPackets = read(pcapReaderObj,filterString)
end
```

```
ecpriFilteredPackets = struct with fields:
             SNo: 21
       Timestamp: 1.6128e+15
        LinkType: 1
        Protocol: 'eth'
    PacketLength: 64
          Packet: [1x1 struct]
        RawBytes: [1x0 double]


ecpriFilteredPackets = struct with fields:
             SNo: 22
       Timestamp: 1.6128e+15
        LinkType: 1
        Protocol: 'eth'
```

```
    PacketLength: 64
          Packet: [1x1 struct]
        RawBytes: [1x0 double]


ecpriFilteredPackets = struct with fields:
             SNo: 31
       Timestamp: 1.6128e+15
        LinkType: 1
        Protocol: 'eth'
    PacketLength: 64
          Packet: [1x1 struct]
        RawBytes: [1x0 double]
```

Input Arguments

pcap — PCAP file reader
`pcapReader` object

PCAP file reader, specified as a `pcapReader` object.

`packetFilter` — Packet filter
character vector | string scalar

Packet filter, specified as a character vector or a string scalar. Ignoring the white spaces, this object function reads the protocol packets that match this value. If the next packet does not match this value, the object function continues to read the next packet until one of these actions occurs.

- The object function detects a packet that matches the specified filter.
- The object function reaches the end of the PCAP file.

You can specify this argument as a combination of one or more filters connected by logical AND (& or &&) or logical OR (`|` or `||`). This syntax shows a typical packet filter.

`protocolName`.*`FieldName`* `RELATIONALOPERATOR` *`value`*

The valid values for *`protocolName`* are `pcap`, `eth`, and `ecpri`.

This table shows the valid values of *`FieldName`* and the data type of the corresponding value if *`protocolName`* is `pcap`.

`FieldName` Value	**`value` Value**
`SNo`	Integer
`TimestampSec`	Integer
`PacketLength`	Integer

This table shows the valid values of *`FieldName`* and the data type of the corresponding value if *`protocolName`* is `eth`.

FieldName Value	*value* Value
SourceAddress	Character vector or string scalar representing hexadecimal bytes
DestinationAddress	Character vector or string scalar representing hexadecimal bytes
Type	Integer

This table shows the valid values of *FieldName* and the data type of the corresponding value if *protocolName* is ecpri.

FieldName Value	*value* Value
ProtocolRevision	Integer
MessageType	Character vector or string scalar representing eCPRI message type
PC_ID	Integer
SEQ_ID	Integer

Note For more information about the *FieldName* values, see the protocolPacket output.

The valid values for RELATIONALOPERATOR are <=, >=, >, <, ==, ~=, and ~.

Example: ‘pcap.PacketLength == 100’ filters the protocol packets of size 100 bytes.

Example: ‘eth.SourceAddress == 44FB5A9710AC & eth.Type == 2048’ filters the Ethernet packets with the specified source address and type of upper-layer protocol.

Example: ‘ecpri.MessageType == IQData || ecpri.MessageType == RemoteReset || ecpri.MessageType == EventIndication’ filters the eCPRI packets with message type IQ data, remote reset, or event indication.

Data Types: char | string

Output Arguments

protocolPacket — Decoded protocol packet

structure

Decoded protocol packet, returned as a structure containing these fields.

Field	Description
SNo	Serial number of the protocol packet
Timestamp	Timestamp of the packet in the format specified by the OutputTimestampFormat property.
TimestampSec	Timestamp of the packet in seconds. To enable this field, set the OutputTimestampFormat property of the pcap input to 'datetime'.

Field	Description
Protocol	Name of the link type
LinkType	Link type specified in the PCAP global header
PacketLength	Original length of the packet transmitted in the network, in bytes If this value is greater than the SnapLength property of the pcapReader object, the length of the decoded protocol packet is truncated to contain SnapLength bytes.
Packet	Decoded packet structure or packet bytes If the PCAP file includes Ethernet packets or eCPRI packets encapsulated in Ethernet, this field contains a decoded packet structure. Otherwise, this field contains packet bytes.
RawBytes	Undecoded bytes because of padding or decode error

If the PCAP file includes Ethernet packets, the Packet field of this argument contains a structure, eth, containing these fields.

Field	Description
SourceAddress	Medium access control (MAC) address of the source host in decimal bytes, returned as a column vector. You can access the source address of the first Ethernet packet by using the command protocolPacket(1).Packet.eth.SourceAddress.
DestinationAddress	MAC address of the destination host in decimal bytes, returned as a column vector. You can access the destination address of the first Ethernet packet by using the command protocolPacket(1).Packet.eth.DestinationAddress.
Type	Field in the Ethernet header that identifies the upper-layer protocol. You can access the upper-layer protocol type of the first Ethernet packet by using the command protocolPacket(1).Packet.eth.Type.

Field	Description
Payload	Upper-layer protocol packet, returned as a vector of octets where each element is in the range [0, 255]. You can access the upper-layer protocol payload of the first Ethernet packet by using the command protocolPacket(1).Packet.eth.Payload.
RawBytes	Undecoded bytes returned by the Ethernet decoder because of padding or decode error. If the Ethernet decoder does not return any undecoded bytes, this field is not present in the eth structure. You can access the raw bytes of the first Ethernet packet by using the command protocolPacket(1).Packet.eth.RawBytes.

If the PCAP file includes eCPRI packets encapsulated in Ethernet, the Packet field of this argument contains a structure or a cell array of structures, ecpri, containing these fields.

Field	Description
ProtocolRevision	Protocol version of eCPRI, specified as a nonnegative integer.
Concatenation	Concatenation indicator of eCPRI messages, specified as 0 or 1. • A value of 0 indicates the corresponding eCPRI message is the last one inside the eCPRI protocol data unit (PDU). • A value of 1 indicates that another eCPRI message follows this one within the eCPRI PDU.
MessageType	Type of service supported by the eCPRI message, specified as a character vector or a string scalar.

The output of the eCPRI decoder varies according to the MessageType value. This table shows the MessageType values and the corresponding additional fields supported by the ecpri structure.

MessageType Value	MessageType Description	Additional Fields Supported by ecpri Structure
IQData	Transfer time domain or frequency domain in-phase and quadrature (IQ) samples	PC_ID, SEQ_ID, UserData, and Payload
BitSequence	Transfer user data in the form of bit sequence	PC_ID, SEQ_ID, UserData, and Payload

MessageType Value	MessageType Description	Additional Fields Supported by ecpri Structure
RealTimeControlData	Transfer vendor specific real-time control messages	PC_ID, SEQ_ID, UserData, and Payload
GenericDataTransfer	Transfer user plane data or related control	PC_ID, SEQ_ID, UserData, and Payload
RemoteMemoryAccess	Enable reading or writing from or to a specific memory address	RemoteMemoryAccessID, ReadWrite, RequestResponse, ElementID, Address, DataLength, UserData, and Payload
OnewayDelayMeasurement	Estimate the one-way delay between two eCPRI ports in one direction	MeasurementID, ActionType, TimestampSec, TimestampNanoSec, CompensationValue, UserData, and Payload
RemoteReset	Request to reset an eCPRI node	ResetID, ResetCodeOp, UserData, and Payload
EventIndication	Indicates an event has occurred	EventID, EventType, SequenceNumber, NumberOfFaultsOrNotifications, and Elements

Note For more information on the message types and the corresponding additional fields supported in the ecpri structure, see Section 3.2.4 of the eCPRI Specification V1.2 [1].

Data Types: struct

Version History

Introduced in R2021b

References

[1] “Common Public Radio Interface: eCPRI Interface Specification V1.2 ” Accessed June 22, 2021.

See Also

Objects
pcapReader

Functions
addLinkTypeDecoder | addUpperLayerDecoder | readAll | reset

readAll

Read all protocol packets from current position to end of PCAP file

Syntax

```
protocolPackets = readAll(pcap)
protocolPackets = readAll(pcap,packetFilter)
```

Description

`protocolPackets = readAll(pcap)` reads all of the protocol packets from the current position to the end of the packet capture (PCAP) file specified by the input PCAP file reader and returns the decoded packets.

`protocolPackets = readAll(pcap,packetFilter)` returns only the decoded packets that match the specified filter, `packetFilter`.

Examples

Read Ethernet Packets from PCAP File

Create a PCAP file reader object, specifying the name of a PCAP file.

```
pcapReaderObj = pcapReader('ethernetSamplePackets.pcap');
```

Read all of the packets from the PCAP file to the MATLAB® workspace.

```
decodedPackets = readAll(pcapReaderObj)
```

```
decodedPackets=1×35 struct array with fields:
    SNo
    Timestamp
    LinkType
    Protocol
    PacketLength
    Packet
    RawBytes
```

Input Arguments

pcap — PCAP file reader
`pcapReader` object

PCAP file reader, specified as a `pcapReader` object.

`packetFilter` — Packet filter
character vector | string scalar

Packet filter, specified as a character vector or a string scalar. Ignoring the white spaces, this object function reads the protocol packets that match this value.

You can specify this argument as a combination of one or more filters connected by logical AND (& or &&) or logical OR (`|` or `||`). This syntax shows a typical packet filter.

`protocolName``.`*`FieldName`* `RELATIONALOPERATOR` *`value`*

The valid values for *`protocolName`* are `pcap`, `eth`, and `ecpri`.

This table shows the valid values of *`FieldName`* and the data type of the corresponding value if *`protocolName`* is `pcap`.

FieldName Value	**value Value**
`SNo`	Integer
`TimestampSec`	Integer
`PacketLength`	Integer

This table shows the valid values of *`FieldName`* and the data type of the corresponding value if *`protocolName`* is `eth`.

***FieldName* Value**	***value* Value**
`SourceAddress`	Character vector or string scalar representing hexadecimal bytes
`DestinationAddress`	Character vector or string scalar representing hexadecimal bytes
`Type`	Integer

This table shows the valid values of *`FieldName`* and the data type of the corresponding value if *`protocolName`* is `ecpri`.

***FieldName* Value**	***value* Value**
`ProtocolRevision`	Integer
`MessageType`	Character vector or string scalar representing eCPRI message type
`PC_ID`	Integer
`SEQ_ID`	Integer

Note For more information about the *`FieldName`* values, see the `protocolPacket` output.

The valid values for `RELATIONALOPERATOR` are <=, >=, >, <, ==, ~=, and ~.

Example: `‘pcap.PacketLength == 100’` filters the protocol packets of size 100 bytes.

Example: `‘eth.SourceAddress == 44FB5A9710AC & eth.Type == 2048’` filters the Ethernet packets with the specified source address and type of upper-layer protocol.

Example: `‘ecpri.MessageType == IQData || ecpri.MessageType == RemoteReset || ecpri.MessageType == EventIndication’` filters the eCPRI packets with message type IQ data, remote reset, or event indication.

Data Types: char | string

Output Arguments

protocolPackets — Decoded protocol packets
array of structures

Decoded protocol packets, returned as an array of structures containing these fields.

Field	Description
SNo	Serial number of the protocol packet
Timestamp	Timestamp of the packet in the format specified by the OutputTimestampFormat property.
TimestampSec	Timestamp of the packet in seconds. To enable this field, set the OutputTimestampFormat property of the pcap input to 'datetime'.
Protocol	Name of the link type
LinkType	Link type specified in the PCAP global header
PacketLength	Original length of the packet transmitted in the network, in bytes If this value is greater than the SnapLength property of the pcapReader object, the length of the decoded protocol packet is truncated to contain SnapLength bytes.
Packet	Decoded packet structure or packet bytes If the PCAP file includes Ethernet packets or eCPRI packets encapsulated in Ethernet, this field contains a decoded packet structure. Otherwise, this field contains packet bytes.
RawBytes	Undecoded bytes because of padding or decode error

If the PCAP file includes Ethernet packets, the Packet field of this argument contains a structure, eth, containing these fields.

Field	Description
SourceAddress	Medium access control (MAC) address of the source host in decimal bytes, returned as a column vector. You can access the source address of the first Ethernet packet by using the command protocolPacket(1).Packet.eth.SourceAddress.

Field	Description
DestinationAddress	MAC address of the destination host in decimal bytes, returned as a column vector. You can access the destination address of the first Ethernet packet by using the command protocolPacket(1).Packet.eth.DestinationAddress.
Type	Field in the Ethernet header that identifies the upper-layer protocol. You can access the upper-layer protocol type of the first Ethernet packet by using the command protocolPacket(1).Packet.eth.Type.
Payload	Upper-layer protocol packet, returned as a vector of octets where each element is in the range [0, 255]. You can access the upper-layer protocol payload of the first Ethernet packet by using the command protocolPacket(1).Packet.eth.Payload.
RawBytes	Undecoded bytes returned by the Ethernet decoder because of padding or decode error. If the Ethernet decoder does not return any undecoded bytes, this field is not present in the eth structure. You can access the raw bytes of the first Ethernet packet by using the command protocolPacket(1).Packet.eth.RawBytes.

If the PCAP file includes eCPRI packets encapsulated in Ethernet, the Packet field of this argument also contains a structure or a cell array of structures, ecpri, containing these fields.

Field	Description
ProtocolRevision	Protocol version of eCPRI, specified as a nonnegative integer.
Concatenation	Concatenation indicator of eCPRI messages, specified as 0 or 1. • A value of 0 indicates the corresponding eCPRI message is the last one inside the eCPRI protocol data unit (PDU). • A value of 1 indicates that another eCPRI message follows this one within the eCPRI PDU.

Field	Description
MessageType	Type of service supported by the eCPRI message, specified as a character vector or a string scalar.

The output of the eCPRI decoder varies according to the MessageType value. This table shows the MessageType values and the corresponding additional fields supported by the ecpri structure.

MessageType Value	MessageType Description	Additional Fields Supported by ecpri Structure
IQData	Transfer time domain or frequency domain in-phase and quadrature (IQ) samples	PC_ID, SEQ_ID, UserData, and Payload
BitSequence	Transfer user data in the form of bit sequence	PC_ID, SEQ_ID, UserData, and Payload
RealTimeControlData	Transfer vendor specific real-time control messages	PC_ID, SEQ_ID, UserData, and Payload
GenericDataTransfer	Transfer user plane data or related control	PC_ID, SEQ_ID, UserData, and Payload
RemoteMemoryAccess	Enable reading or writing from or to a specific memory address	RemoteMemoryAccessID, ReadWrite, RequestResponse, ElementID, Address, DataLength, UserData, and Payload
OnewayDelayMeasurement	Estimate the one-way delay between two eCPRI ports in one direction	MeasurementID, ActionType, TimestampSec, TimestampNanoSec, CompensationValue, UserData, and Payload
RemoteReset	Request to reset an eCPRI node	ResetID, ResetCodeOp, UserData, and Payload
EventIndication	Indicates an event has occurred	EventID, EventType, SequenceNumber, NumberOfFaultsOrNotifications, and Elements

Note For more information on the message types and the corresponding additional fields supported in the ecpri structure, see Section 3.2.4 of the eCPRI Specification V1.2 [1].

Data Types: struct

Version History

Introduced in R2021b

References

[1] "Common Public Radio Interface: eCPRI Interface Specification V1.2 " Accessed June 22, 2021.

See Also

Objects
`pcapReader`

Functions
`addLinkTypeDecoder` | `addUpperLayerDecoder` | `read` | `reset`

reset

Reset position of PCAP file reader to first protocol packet of PCAP file

Syntax

```
reset(pcap)
```

Description

`reset(pcap)` resets the position of the specified PCAP file reader to the first protocol packet in the PCAP file.

Examples

Reset PCAP File Reader

Create a PCAP file reader object, specifying the name of a PCAP file.

```
pcapReaderObj = pcapReader('ethernetSamplePackets.pcap');
```

Create a filter for the eCPRI packets, specifying the message type as a bit sequence.

```
filterString = 'ecpri.MessageType == BitSequence';
```

Read the first eCPRI packet that matches the specified filter to the MATLAB® workspace.

```
ecpriFilteredFirstPacket = read(pcapReaderObj,filterString)
```

```
ecpriFilteredFirstPacket = struct with fields:
             SNo: 22
       Timestamp: 1.6128e+15
        LinkType: 1
        Protocol: 'eth'
    PacketLength: 64
          Packet: [1x1 struct]
        RawBytes: [1x0 double]
```

Reset the position of the PCAP file reader to the first packet of the PCAP file.

```
reset(pcapReaderObj);
```

Create a new filter for the eCPRI packets, specifying the message type as a bit sequence.

```
filterString = 'ecpri.MessageType == IQData';
```

Read the eCPRI packets that match the specified filter to the MATLAB workspace.

```
ecpriFilteredPackets = readAll(pcapReaderObj,filterString)
```

```
ecpriFilteredPackets = struct with fields:
             SNo: 21
```

```
      Timestamp: 1.6128e+15
       LinkType: 1
       Protocol: 'eth'
   PacketLength: 64
         Packet: [1x1 struct]
       RawBytes: [1x0 double]
```

Input Arguments

pcap — PCAP file reader
`pcapReader` object

PCAP file reader, specified as a `pcapReader` object.

Version History

Introduced in R2021b

References

[1] “Common Public Radio Interface: eCPRI Interface Specification V1.2 ” Accessed June 22, 2021.

See Also

Objects
`pcapReader`

Functions
`addLinkTypeDecoder` | `addUpperLayerDecoder` | `read` | `readAll`

resetSoftBuffer

Reset soft buffer for HARQ process in UL-SCH or DL-SCH decoder

Syntax

```
resetSoftBuffer(decUL)
resetSoftBuffer(decDL,cwid)
resetSoftBuffer( ___ ,harqID)
```

Description

`resetSoftBuffer(decUL)` resets the soft buffer for hybrid automatic repeat-request (HARQ) process number 0 in the specified UL-SCH decoder `decUL`.

`resetSoftBuffer(decDL,cwid)` resets the soft buffer for codeword index `cwid` and HARQ process number 0 in the specified DL-SCH decoder `decDL`. The codeword index `cwid` specifies one of the two possible codewords for DL-SCH decoding.

`resetSoftBuffer( ___ ,harqID)` resets the soft buffer for the specified HARQ process number `harqID`. Specify `harqID` in addition to the input arguments in any of the previous syntaxes.

To enable soft combining of retransmissions before low-density parity-check (LDPC) decoding, each decoder object maintains a soft buffer for each HARQ process. Upon successful decoding of the input, the object automatically resets the soft buffer for the HARQ process. Calling the `resetSoftBuffer` function resets the soft buffer manually. Call this function when decoding different transport blocks for the same HARQ process subsequently or when all redundancy versions for a HARQ process are complete.

Examples

Reset Soft Buffer in UL-SCH Decoder with Multiple HARQ Processes

Generate a random sequence of binary values corresponding to one transport block of length 5120.

```
trBlkLen1 = 5120;
trBlk1 = randi([0 1],trBlkLen1,1,'int8');
```

Create and configure a UL-SCH encoder System object with multiple HARQ processes.

```
encUL = nrULSCH('MultipleHARQProcesses',true);
```

Load the transport block into the UL-SCH encoder for HARQ process number 1.

```
harqID = 1;
setTransportBlock(encUL,trBlk1,harqID);
```

Call the encoder with QPSK modulation scheme, 1 transmission layer, an output length of 10,240 bits, redundancy version 0, and HARQ process number 1. The encoder applies the UL-SCH processing chain to the transport block loaded into the object using HARQ process number 1.

```
rv = 0;
codedTrBlock1 = encUL('QPSK',1,10240,rv,harqID);
```

Create and configure an UL-SCH decoder System object with multiple HARQ processes.

```
decUL = nrULSCHDecoder('MultipleHARQProcesses',true);
```

Configure the decoder for the encoded transport block.

```
decUL.TransportBlockLength = trBlkLen1;
```

Add noise to the soft bits representing the encoded transport block. Call the UL-SCH decoder on the modified soft bits for HARQ process number 1.

```
rxSoftBits1 = awgn(1-2*double(codedTrBlock1),5);
[decBits1,blkErr1] = decUL(rxSoftBits1,'QPSK',1,rv,harqID);
```

The added noise results in an error during the decoding.

```
blkErr1
```

```
blkErr1 = logical
   1
```

Repeat the encoding operation for a new transport block of length 4400 and HARQ process number 1.

```
trBlkLen2 = 4400;
trBlk2 = randi([0 1],trBlkLen2,1,'int8');
setTransportBlock(encUL,trBlk2,harqID);
codedTrBlock2 = encUL('QPSK',1,8800,rv,harqID);
```

Configure the decoder for the second transport block.

```
decUL.TransportBlockLength = trBlkLen2;
```

If an error occurred during the previous decoding with HARQ process number 1, you must reset the soft buffer of the HARQ process before decoding the second transport block.

```
if blkErr1
    resetSoftBuffer(decUL,harqID);
end
```

Call the decoder on the soft bits representing the second encoded transport block using HARQ process number 1.

```
rxBits2 = 1-2*double(codedTrBlock2);
[decBits2,blkErr2] = decUL(rxBits2,'QPSK',1,rv,harqID);
blkErr2
```

```
blkErr2 = logical
   0
```

Verify that the second transmitted and decoded message bits are identical.

```
isequal(decBits2,trBlk2)
```

```
ans = logical
   1
```

Reset Soft Buffer in DL-SCH Decoder with Multiple HARQ Processes

Generate a random sequence of binary values corresponding to one transport block of length 5120.

```
trBlkLen1 = 5120;
trBlk1 = randi([0 1],trBlkLen1,1,'int8');
```

Create and configure a DL-SCH encoder System object with multiple HARQ processes.

```
encDL = nrDLSCH('MultipleHARQProcesses',true);
```

Load the transport block into the DL-SCH encoder for HARQ process number 1 and codeword index 1.

```
harqID = 1;
cwID = 0;
setTransportBlock(encDL,trBlk1,cwID,harqID);
```

Call the encoder with QPSK modulation scheme, 1 transmission layer, an output length of 10,240 bits, redundancy version 0, and HARQ process number 1. The encoder applies the DL-SCH processing chain to the transport block loaded into the object for HARQ process number 1.

```
rv = 0;
codedTrBlock1 = encDL('QPSK',1,10240,rv,harqID);
```

Create and configure a DL-SCH decoder System object with multiple HARQ processes.

```
decDL = nrDLSCHDecoder('MultipleHARQProcesses',true);
```

Configure the decoder for the encoded transport block.

```
decDL.TransportBlockLength = trBlkLen1;
```

Add noise to the soft bits representing the encoded transport block. Call the DL-SCH decoder on the modified soft bits for HARQ process number 1.

```
rxSoftBits1 = awgn(1-2*double(codedTrBlock1),5);
[decBits1,blkErr1] = decDL(rxSoftBits1,'QPSK',1,rv,harqID);
```

The added noise results in an error during decoding.

```
blkErr1
```

```
blkErr1 = logical
   1
```

Repeat the encoding operation for a new transport block of length 4400 and HARQ process number 1.

```
trBlkLen2 = 4400;
trBlk2 = randi([0 1],trBlkLen2,1,'int8');
```

```
setTransportBlock(encDL,trBlk2,cwID,harqID);
codedTrBlock2 = encDL('QPSK',1,8800,rv,harqID);
```

Configure the decoder for the second transport block.

```
decDL.TransportBlockLength = trBlkLen2;
```

If an error occurred during the previous decoding with HARQ process number 1, you must reset the soft buffer of the HARQ process before decoding the second transport block.

```
if blkErr1
    resetSoftBuffer(decDL,harqID);
end
```

Call the decoder on the soft bits representing the second encoded transport block using HARQ process number 1.

```
rxBits2 = 1-2*double(codedTrBlock2);
[decBits2,blkErr2] = decDL(rxBits2,'QPSK',1,rv,harqID);
blkErr2
```

```
blkErr2 = logical
   0
```

Verify that the second transmitted and decoded message bits are identical.

```
isequal(decBits2,trBlk2)
```

```
ans = logical
   1
```

Input Arguments

decUL — UL-SCH decoder
nrULSCHDecoder System object

UL-SCH decoder, specified as a `nrULSCHDecoder` System object. The object implements the UL-SCH decoder processing chain corresponding to the inverse operation of UL-SCH encoding specified in TR 38.212 Section 6.2.

decDL — DL-SCH decoder
nrDLSCHDecoder System object

DL-SCH decoder, specified as a `nrDLSCHDecoder` System object. The object implements the DL-SCH decoder processing chain corresponding to the inverse operation of DL-SCH encoding specified in TR 38.212 Section 7.2.

cwid — DL-SCH codeword index
0 | 1

DL-SCH codeword index, specified as `0` or `1`.

Data Types: `double`

`harqID` — HARQ process number
integer from 0 to 15

HARQ process number, specified as an integer from 0 to 15.

Data Types: `double`

Version History

Introduced in R2019a

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
`nrULSCHDecoder` | `nrDLSCHDecoder`

setTransportBlock

Load transport block into UL-SCH or DL-SCH encoder

Syntax

```
setTransportBlock(enc,trblk)
setTransportBlock(encDL,trblk,trblkID)
setTransportBlock( ___ ,harqID)
```

Description

`setTransportBlock(enc,trblk)` loads the transport block `trblk` into the specified uplink (UL) or downlink (DL) shared channel (SCH) encoder System object `enc`. Call this function before calling `enc`.

`setTransportBlock(encDL,trblk,trblkID)` loads the transport block `trblk` into the specified DL-SCH encoder System object `encDL` for the specified transport block number `trblkID`. Call this function before calling `encDL`.

`setTransportBlock( ___ ,harqID)` loads the transport block for the specified hybrid automatic repeat-request (HARQ) process number `harqID`. Specify `harqID` in addition to the input arguments in any of the previous syntaxes.

Examples

UL-SCH Encoding with Multiple HARQ Processes

Generate a random sequence of binary values corresponding to one transport block of length 5120.

```
trBlkLen = 5120;
trBlk = randi([0 1],trBlkLen,1,'int8');
```

Create and configure an UL-SCH encoder System object for use with multiple HARQ processes.

```
encUL = nrULSCH;
encUL.MultipleHARQProcesses = true;
```

Load the transport block into the UL-SCH encoder, specifying HARQ process number 2.

```
harqID = 2;
setTransportBlock(encUL,trBlk,harqID);
```

Call the encoder with QPSK modulation scheme, 3 transmission layers, an output length of 10,002 bits, and redundancy version 3. The encoder applies the UL-SCH processing chain to the transport block loaded into the object for HARQ process number 2.

```
mod = 'QPSK';
nLayers = 3;
outlen = 10002;
rv = 3;
codedTrBlock = encUL(mod,nLayers,outlen,rv,harqID);
```

Verify that the encoded transport block has the required number of bits.

```
isequal(length(codedTrBlock),outlen)
```

```
ans = logical
   1
```

DL-SCH Encoding with Multiple HARQ Processes

Generate a random sequence of binary values corresponding to one transport block of length 5120.

```
trBlkLen = 5120;
trBlk = randi([0 1],trBlkLen,1,'int8');
```

Create and configure a DL-SCH encoder System object for use with multiple HARQ processes.

```
encDL = nrDLSCH;
encDL.MultipleHARQProcesses = true;
```

Load transport block `trBlk` for transport block number 0 into the DL-SCH encoder, specifying HARQ process number 2.

```
harqID = 2;
trBlkID = 0;
setTransportBlock(encDL,trBlk,trBlkID,harqID);
```

Call the encoder with QPSK modulation scheme, 3 transmission layers, an output length of 10,002 bits, and redundancy version 3. The encoder applies the DL-SCH processing chain to the transport block loaded into the object for HARQ process number 2.

```
mod = 'QPSK';
nLayers = 3;
outlen = 10002;
rv = 3;
codedTrBlock = encDL(mod,nLayers,outlen,rv,harqID);
```

Verify that the encoded transport block has the required number of bits.

```
isequal(length(codedTrBlock),outlen)
```

```
ans = logical
   1
```

Input Arguments

enc — UL-SCH or DL-SCH encoder

`nrULSCH` System object | `nrDLSCH` System object

UL-SCH or DL-SCH encoder, specified as an `nrULSCH` or `nrDLSCH` System object. The objects implement the UL-SCH and DL-SCH processing chains specified in TS 38.212 Section 6.2 and Section 7.2, respectively.

trblk — Transport block
binary column vector | cell array of one or two binary column vectors

Transport block, specified as a binary column vector or a cell array of one or two binary column vectors. The cell array option applies only when `enc` is a DL-SCH encoder System object. The two-element cell array option applies only when the transport block number `trblkID` is not specified for DL-SCH processing.

Data Types: `int8` | `double` | `logical`

encDL — DL-SCH encoder
`nrDLSCH` System object

DL-SCH encoder, specified as an `nrDLSCH` System object. The object implements the DL-SCH processing chain specified in TS 38.212 Section 7.2.

trblkID — Transport block number in DL-SCH processing
`0` (default) | `1`

Transport block number in DL-SCH processing, specified as `0` or `1`.

Data Types: `double`

harqID — HARQ process number
integer from 0 to 15

HARQ process number, specified as an integer from 0 to 15.

Data Types: `double`

Version History

Introduced in R2019a

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`getTransportBlock`

Objects
`nrULSCH` | `nrDLSCH`

swapTransmitAndReceive

Reverse link direction in CDL channel model

Syntax

`swapTransmitAndReceive(cdl)`

Description

`swapTransmitAndReceive(cdl)` reverses the link direction in the specified clustered delay line (CDL) channel model by swapping the roles of the transmit and receive antennas in the channel. The swap results in a reciprocal channel model but does not alter the channel fading. Sending waveforms through the channel for both link directions (original and reversed) results in channel path gains that differ only in swapped transmit and receive antenna dimensions. Calling the function again reverts the channel to its original link direction by swapping back the roles of the transmit and receive antennas.

You can use this function during a simulation to model time division duplex (TDD) operation by sending waveforms for each link direction through the channel while maintaining channel reciprocity.

The function swaps the values of these property pairs of the input `cdl` to reflect the link direction change.

- `TransmitAntennaArray` and `ReceiveAntennaArray`
- `TransmitArrayOrientation` and `ReceiveArrayOrientation`
- `AnglesAoD` and `AnglesAoA` (only when `DelayProfile` is set to `'Custom'`)
- `AnglesZoD` and `AnglesZoA` (only when `DelayProfile` is set to `'Custom'`)
- Elements related to departure and arrival angles in the `AngleSpreads` and `MeanAngles` properties
- `NumTransmitAntennas` and `NumReceiveAntennas` output structure fields of the function call `info(cdl)`
- `NumInputSignals` and `NumOutputSignals` output structure fields of the function call `info(cdl)`

The function also toggles the `TransmitAndReceiveSwapped` property value of the input `cdl` between `true` and `false` to reflect the actual status of the link direction.

Note To preserve the reciprocity of a multi-antenna channel, set `cdl.NormalizeChannelOutputs` to `false` before calling `swapTransmitAndReceive(cdl)`.

Examples

Utilize Channel Model Reciprocity in Downlink and Uplink Transmission

Use a single channel model for both a downlink and an uplink transmission by swapping the roles of the transmit and receive antennas in the channel. Confirm that the channel model is reciprocal.

Create a default carrier configuration object for both the downlink and the uplink transmission.

```
carrier = nrCarrierConfig;
```

Create and configure the channel model. You can specify either a CDL channel model or a TDL channel model.

```
channel = nrCDLChannel;                 % For TDL channel model, use nrTDLChannel
channel.MaximumDopplerShift = 0;
channel.NormalizeChannelOutputs = false; % Avoid normalization by the number of Rx antennas
ofdmInfo = nrOFDMInfo(carrier);
channel.SampleRate = ofdmInfo.SampleRate;
```

Create a signal for downlink transmission.

```
channelinfo = info(channel);
txgridDL = nrResourceGrid(carrier,channelinfo.NumTransmitAntennas);
txDL = nrOFDMModulate(carrier,txgridDL);
```

Account for the maximum channel delay.

```
channelinfo = info(channel);
maxChDelay = ceil(max(channelinfo.PathDelays*channel.SampleRate)) + channelinfo.ChannelFilterDela
txDL = [txDL; zeros([maxChDelay size(txDL,2)])];
```

Send the downlink signal through the channel model.

```
[rxDL,pathGainsDL] = channel(txDL);
```

Reconfigure the channel model for uplink transmission by swapping the roles of the transmit and receive antennas.

```
swapTransmitAndReceive(channel);
```

Create a signal for uplink transmission.

```
channelInfoUL = info(channel);
txGridUL = nrResourceGrid(carrier,channelInfoUL.NumTransmitAntennas);
txUL = nrOFDMModulate(carrier,txGridUL);
```

Account for the maximum channel delay.

```
txUL = [txUL; zeros([maxChDelay size(txUL,2)])];
```

Send the uplink signal through the reconfigured channel model.

```
[rxUL,pathGainsUL] = channel(txUL);
```

Confirm that the channel is reciprocal, that is, the channel fading is the same in both transmissions. The uplink channel path gain and the downlink channel path gain differ only in swapped transmit and receive antenna dimensions.

```
isequal(pathGainsUL,permute(pathGainsDL,[1 2 4 3]))
```

```
ans = logical
   1
```

Input Arguments

cdl — CDL channel model
nrCDLChannel System object

CDL channel model, specified as an nrCDLChannel System object. The nrCDLChannel object implements the multi-input multi-output (MIMO) link-level fading channel specified in TR 38.901 Section 7.7.1.

Version History

Introduced in R2021a

References

[1] 3GPP TR 38.901. “Study on channel model for frequencies from 0.5 to 100 GHz.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
nrCDLChannel

Topics
“TDD Reciprocity-Based PDSCH MU-MIMO Using SRS”

swapTransmitAndReceive

Reverse link direction in TDL channel model

Syntax

```
swapTransmitAndReceive(tdl)
```

Description

`swapTransmitAndReceive(tdl)` reverses the link direction in the specified tapped delay line (TDL) channel model by swapping the roles of the transmit and receive antennas in the channel. The swap results in a reciprocal channel model but does not alter the channel fading. Sending waveforms through the channel for both link directions (original and reversed) results in channel path gains that differ only in swapped transmit and receive antenna dimensions. Calling the function again reverts the channel to its original link direction by swapping back the roles of the transmit and receive antennas.

You can use this function during a simulation to model time division duplex (TDD) operation by sending waveforms for each link direction through the channel while maintaining channel reciprocity.

The function swaps the values of these property pairs of the input `tdl` to reflect the link direction change.

- `NumTransmitAntennas` and `NumReceiveAntennas`
- `TransmitCorrelationMatrix` and `ReceiveCorrelationMatrix`
- `TransmitPolarizationAngles` and `ReceivePolarizationAngles`
- `NumTransmitAntennas` and `NumReceiveAntennas` output structure fields of the function call `info(tdl)`
- Matrix elements related to transmit and receive antennas in the `SpatialCorrelationMatrix` property and the corresponding output structure field of the function call `info(tdl)`

The function also toggles the `TransmitAndReceiveSwapped` property value of the input `tdl` between `true` and `false` to reflect the actual status of the link direction.

Note To preserve the reciprocity of a multi-antenna channel, set `tdl.NormalizeChannelOutputs` to `false` before calling `swapTransmitAndReceive(tdl)`.

Examples

Utilize Channel Model Reciprocity in Downlink and Uplink Transmission

Use a single channel model for both a downlink and an uplink transmission by swapping the roles of the transmit and receive antennas in the channel. Confirm that the channel model is reciprocal.

Create a default carrier configuration object for both the downlink and the uplink transmission.

```
carrier = nrCarrierConfig;
```

Create and configure the channel model. You can specify either a CDL channel model or a TDL channel model.

```
channel = nrCDLChannel;                  % For TDL channel model, use nrTDLChannel
channel.MaximumDopplerShift = 0;
channel.NormalizeChannelOutputs = false; % Avoid normalization by the number of Rx antennas
ofdmInfo = nrOFDMInfo(carrier);
channel.SampleRate = ofdmInfo.SampleRate;
```

Create a signal for downlink transmission.

```
channelinfo = info(channel);
txgridDL = nrResourceGrid(carrier,channelinfo.NumTransmitAntennas);
txDL = nrOFDMModulate(carrier,txgridDL);
```

Account for the maximum channel delay.

```
channelinfo = info(channel);
maxChDelay = ceil(max(channelinfo.PathDelays*channel.SampleRate)) + channelinfo.ChannelFilterDela
txDL = [txDL; zeros([maxChDelay size(txDL,2)])];
```

Send the downlink signal through the channel model.

```
[rxDL,pathGainsDL] = channel(txDL);
```

Reconfigure the channel model for uplink transmission by swapping the roles of the transmit and receive antennas.

```
swapTransmitAndReceive(channel);
```

Create a signal for uplink transmission.

```
channelInfoUL = info(channel);
txGridUL = nrResourceGrid(carrier,channelInfoUL.NumTransmitAntennas);
txUL = nrOFDMModulate(carrier,txGridUL);
```

Account for the maximum channel delay.

```
txUL = [txUL; zeros([maxChDelay size(txUL,2)])];
```

Send the uplink signal through the reconfigured channel model.

```
[rxUL,pathGainsUL] = channel(txUL);
```

Confirm that the channel is reciprocal, that is, the channel fading is the same in both transmissions. The uplink channel path gain and the downlink channel path gain differ only in swapped transmit and receive antenna dimensions.

```
isequal(pathGainsUL,permute(pathGainsDL,[1 2 4 3]))
```

```
ans = logical
   1
```

Input Arguments

tdl — TDL channel model
nrTDLChannel System object

TDL channel model, specified as an `nrTDLChannel` System object. The `nrTDLChannel` object implements the multi-input multi-output (MIMO) link-level fading channel specified in TR 38.901 Section 7.7.2.

Version History

Introduced in R2021a

References

[1] 3GPP TR 38.901. “Study on channel model for frequencies from 0.5 to 100 GHz.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`nrTDLChannel`

Topics

“TDD Reciprocity-Based PDSCH MU-MIMO Using SRS”

write

Write protocol packet data to PCAP or PCAPNG file

Syntax

```
write(pcapObj,packet,timestamp)
write(pcapngObj,packet,timestamp,interfaceID)
write( ___ ,Name,Value)
```

Description

`write(pcapObj,packet,timestamp)` writes the protocol packet data to the PCAP file specified in the PCAP file writer object, `pcapObj`. Input `packet` specifies the protocol packet and input `timestamp` specifies the packet arrival time.

`write(pcapngObj,packet,timestamp,interfaceID)` writes protocol packet data to a PCAPNG file specified in the PCAPNG file writer object, `pcapngObj`. Input `packet`, `timestamp`, and `interfaceID` specifies the protocol packet, packet arrival time, and interface identifier, respectively.

`write( ___ ,Name,Value)` specifies options using one or more name-value pair arguments in addition to the input argument combinations from any of the previous syntaxes. For example, `'PacketFormat','bits'` sets the format of the protocol packets to bits.

Examples

Write 5G NR Packet Data to PCAP File

Create a PCAP file writer object, specifying the name of the PCAP file.

```
pcapObj = pcapWriter('FileName','write5GNRpacket');
timestamp = 300;                                            % Timestamp
```

5G NR packets do not have a valid link type. As per Tcpdump, if a valid link type is not present, specify the link type of SLL packet.

```
linkType = 113;
```

Write a global header to the PCAP file.

```
writeGlobalHeader(pcapObj,linkType);
```

5G NR packets are not directly supported by Wireshark. To enable Wireshark to parse 5G NR packets, add encapsulation and metadata to the 5G NR packet.

```
payload = [59; 205];                                        % MAC subPDU (contains truncated buffe
radioType = 1;                                              % Frequency division duplexing
linkDir = 0;                                                % Uplink packet
rntiType = 3;                                               % Cell-RNTI
startString = [109; 97; 99; 45; 110; 114];                  % Tag to indicate start of NR MAC sign
payloadTag = 1;                                             % Payload tag for NR packets
```

```
signature = [startString; radioType; linkDir; rntiType];
macNRInfoPacket = [signature; payloadTag; payload];
```

Construct a user datagram protocol (UDP) header.

```
udpPacketLength = 8 + length(macNRInfoPacket);          % Length of header (8 bytes) and paylo
udpHeader = [163; 76;                                    % Source port number
    39; 15;                                              % Destination port number
    fix(udpPacketLength/256); mod(udpPacketLength,256);  % Total length of UDP packet
    0; 0];                                               % Checksum
```

Construct an IPv4 header.

```
ipPacketLength = 20 + udpPacketLength;                   % Length of header (20 bytes) and payl
ipHeader = [69;                                          % Version of IP protocol and priority
    0;                                                   % Type of service
    fix(ipPacketLength/256); mod(ipPacketLength,256);    % Total length of the IPv4 packet
    0; 1;                                                % Identification
    0; 0;                                                % Flags and fragmentation offset
    64;                                                  % Time to live in seconds
    17;                                                  % UDP protocol number
    0; 0;                                                % Header checksum
    127; 0; 0; 1;                                        % Source IP address
    127; 0; 0; 1];                                       % Destination IP address
```

Construct an SLL header.

```
sllHeader = [0; 0;                                       % Packet type
    3; 4;                                                % Address resolution protocol hardware
    0; 0;                                                % Link layer address length
    0; 0; 0; 0; 0; 0; 0; 0;                              % Link layer address
    8; 0];                                               % Protocol type
```

Construct an 5G NR packet by adding encapsulation and metadata.

```
packet = [sllHeader; ipHeader; udpHeader; macNRInfoPacket];
```

Write the 5G NR packet data to the PCAP file.

```
write(pcapObj,packet,timestamp);
```

Write 5G NR Packet Data to PCAPNG File

Create a PCAPNG file writer object, specifying the name of the PCAPNG file.

```
pcapngObj = pcapngWriter('FileName','write5GNRpacket');
```

Write the interface block for 5G New Radio (NR). 5G NR packets do not have a valid link type. As per Tcpdump, if a valid link type is not present, specify the link type of SLL packet.

```
interface = '5GNR';                                      % Interface name
linkType = 113;                                          % Link type of SLL packet
timestamp = 300;                                         % Timestamp
interfaceID = writeInterfaceDescriptionBlock(pcapngObj,linkType,interface);
```

5G NR packets are not directly supported by Wireshark. To enable Wireshark to parse 5G NR packets, add encapsulation and metadata to the 5G NR packet.

```
payload = [59; 205];                                        % MAC subPDU (contains short truncated
radioType = 1;                                              % Frequency division duplexing
linkDir = 0;                                                % Uplink packet
rntiType = 3;                                               % Cell-RNTI
startString = [109; 97; 99; 45; 110; 114];                  % Tag to indicate the start of NR MAC
payloadTag = 1;                                             % Payload tag for NR packets
signature = [startString; radioType; linkDir; rntiType];
macNRInfoPacket = [signature; payloadTag; payload];
```

Construct a UDP header.

```
udpPacketLength = 8 + length(macNRInfoPacket);              % Length of header (8 bytes) and paylo
udpHeader = [163; 76;                                       % Source port number
    39; 15;                                                 % Destination port number
    fix(udpPacketLength/256); mod(udpPacketLength,256);     % Total length of UDP packet
    0; 0];                                                  % Checksum
```

Construct an IPv4 header.

```
ipPacketLength = 20 + udpPacketLength;                      % Length of header (20 bytes) and payl
ipHeader = [69;                                             % Version of IP protocol and Priority
    0;                                                      % Type of service
    fix(ipPacketLength/256);mod(ipPacketLength,256);        % Total length of the IPv4 packet
    0; 1;                                                   % Identification
    0; 0;                                                   % Flags and fragmentation offset
    64;                                                     % Time to live in seconds
    17;                                                     % UDP protocol number
    0; 0;                                                   % Header checksum
    127; 0; 0; 1;                                           % Source IP address
    127; 0; 0; 1];                                          % Destination IP address
```

Construct an SLL header.

```
sllHeader = [0;0;                                           % Packet type
    3; 4;                                                   % Address resolution protocol hardware
    0; 0;                                                   % Link layer address length
    0; 0; 0; 0; 0; 0; 0; 0;                                 % Link layer address
    8; 0];                                                  % Protocol type
```

Construct an 5G NR packet by adding encapsulation and metadata.

```
packet = [sllHeader; ipHeader; udpHeader; macNRInfoPacket];
```

Write the 5G NR packet data to the PCAPNG file.

```
packetComment = 'This is 5G NR MAC packet';                 % Packet comment
write(pcapngObj,packet,timestamp,interfaceID,'PacketComment',packetComment);
```

Input Arguments

Note The `pcapWriter` and `pcapngWriter` objects do not overwrite the existing PCAP or PCAPNG files, respectively. Each time when you create these objects, specify a unique PCAP or PCAPNG file name.

`pcapObj` — PCAP file writer object
`pcapWriter` object

PCAP file writer object, specified as a `pcapWriter` object.

`packet` — Protocol packet
binary-valued vector | character vector | string scalar | numeric vector | *n*-by-2 character array

Protocol packet, specified as one of these values.

- Binary-valued vector – This value represents bits.
- Character vector – This value represents octets in hexadecimal format.
- String scalar – This value represents octets in hexadecimal format.
- Numeric vector with each element in the range [0, 255] – This value represents octets in decimal format.
- *n*-by-2 character array – In this value, each row represents an octet in hexadecimal format .

Data Types: `char` | `string` | `double`

`timestamp` — Packet arrival time
nonnegative integer

Packet arrival time in POSIX® microseconds elapsed since 1/1/1970, specified as a nonnegative integer.

Data Types: `double`

`pcapngObj` — PCAPNG file writer object
`pcapngWriter` object

PCAPNG file writer object, specified as a `pcapngWriter` object.

`interfaceID` — Unique identifier for an interface
nonnegative scalar

Unique identifier for an interface, specified as a nonnegative scalar.

Data Types: `double`

Name-Value Pair Arguments

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

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

Example: `'PacketFormat','bits'` specifies the format of the protocol packet as bits.

`PacketFormat` — Format of the protocol packet
`'octets'` (default) | `'bits'`

Format of the protocol packet, specified as the comma-separated pair consisting of `PacketFormat` and `'octets'` or `'bits'`. If this value is specified as `'octets'`, `packet` is specified as one of these values.

- Binary-valued vector – This value represents bits.
- Character vector – This value represents octets in hexadecimal format.
- String scalar – This value represents octets in hexadecimal format.
- Numeric vector with each element in the range [0, 255] – This value represents octets in decimal format.
- *n*-by-2 character array – In this value, each row represents an octet in hexadecimal format .

Data Types: `char` | `string` | `double`

`PacketComment` — Comment for protocol packet
`''` (default) | character vector | string scalar

Comment for the protocol packet, specified as the comma-separated pair consisting of `PacketComment` and a character vector or a string scalar.

Dependencies

To enable this name-value pair argument, specify the `pcapngObj` input argument.

Data Types: `char` | `string`

Version History

Introduced in R2020b

References

[1] Tuexen, M. "PCAP Next Generation (Pcapng) Capture File Format." 2020. https://www.ietf.org/.

[2] Group, The Tcpdump. "Tcpdump/Libpcap Public Repository." Accessed May 20, 2020. https://www.tcpdump.org.

[3] "Development/LibpcapFileFormat - The Wireshark Wiki." Accessed May 20, 2020. https://www.wireshark.org.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`writeGlobalHeader` | `writeCustomBlock` | `writeInterfaceDescriptionBlock`

Objects
`pcapWriter` | `pcapngWriter`

write

Write 5G NR MAC packet to PCAP or PCAPNG file

Syntax

```
write(nrPCAPW,packet,timestamp)
write(nrPCAPW,packet,timestamp,Name=Value)
```

Description

`write(nrPCAPW,packet,timestamp)` writes 5G new radio (NR) medium access control (MAC) packets to a packet capture (PCAP) or packet capture next generation (PCAPNG) file (`.pcap` or `.pcapng`, respectively) specified by the NR PCAP or PCAPNG file writer object, `nrPCAPW`. Input `packet` specifies the 5G NR MAC packet, and input `timestamp` specifies the packet arrival time.

`write(nrPCAPW,packet,timestamp,Name=Value)` specifies options using one or more name-value arguments. For example, `PacketFormat='bits'` sets the format of the 5G NR MAC packets to bits.

Examples

Write 5G NR MAC Packet to PCAPNG File

Create a 5G NR PCAPNG file writer object, specifying the name of the file, extension of the file, and file comment.

```
nrpcapw = nrPCAPWriter(FileName='nrPCAPNGExample', ...
    FileExtension='pcapng', ...
    FileComment='Export NR MAC packet to PCAPNG file');
```

Create a 5G NR MAC packet.

```
nrMACPDU = [6; 68; 64; 0; ones(66,1); 62; 4; 7; 74; 96; 102];
```

Set the timestamp for the packet.

```
timestamp = 1000000;                        % In microseconds
```

Create the packet information structure for the MAC packet by using the constants defined in the created 5G NR PCAPNG file writer object.

```
packetInfo = struct();
packetInfo.RadioType = nrpcapw.RadioTDD;    % Frequency division duplexing
packetInfo.LinkDir = nrpcapw.Downlink;      % Link direction
packetInfo.RNTIType = nrpcapw.CellRNTI;     % Type of radio network temporary identifier (RNTI)
```

Write the 5G NR MAC packet to the PCAPNG file.

```
write(nrpcapw,nrMACPDU,timestamp,PacketInfo=packetInfo);
```

Input Arguments

Note The `nrPCAPWriter` object does not overwrite the existing PCAP or PCAPNG file. Each time when you create this object, specify a unique PCAP or PCAPNG file name.

`nrPCAPW` — 5G NR PCAP or PCAPNG file writer
`nrPCAPWriter` object

5G NR PCAP or PCAPNG file writer, specified as an `nrPCAPWriter` object.

`packet` — 5G NR MAC packet
binary-valued vector | character vector | string scalar | numeric vector | *n*-by-2 character array

5G NR MAC packet, specified as one of these options.

- Binary-valued vector — This value represents bits.
- Character vector — This value represents octets in hexadecimal format.
- String scalar — This value represents octets in hexadecimal format.
- Numeric vector with each element in the range [0, 255] — This value represents octets in decimal format.
- *n*-by-2 character array — In this value, each row represents an octet in hexadecimal format and *n* denotes an integer.

Data Types: `char` | `string` | `double`

`timestamp` — Packet arrival time
nonnegative integer

Packet arrival time in POSIX microseconds elapsed since 1/1/1970, specified as a nonnegative integer.

Data Types: `double`

Name-Value Pair Arguments

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

Example: `PacketFormat='bits'` specifies the format of the 5G NR MAC packet as bits.

`PacketInfo` — Packet information
structure

Packet information, specified as a structure containing these case-sensitive fields.

Field	Description
`RadioType`	Mode of duplex, specified as one of these values. • `1` (default) — RadioFDD • `2` — RadioTDD

Field	Description
LinkDir	Direction of the link, specified as one of these values. • 0 (default) — Uplink • 1 — Downlink
RNTIType	Type of radio network temporary identifier (RNTI), specified as one of these values. • 0 — No RNTI • 1 — Paging RNTI • 2 — Random access RNTI • 3 (default) — Cell RNTI • 4 — System information RNTI • 5 — Configured scheduling RNTI
RNTI	Value of the RNTI, specified as a 2-byte decimal value in the range [0, 65,535].
UEID	User equipment identifier, specified as a 2-byte decimal value in the range [0, 65,535].
PHRType2OtherCell	For a multiple entry power headroom report MAC control element, this field decides the presence of the type 2 power headroom field for a special cell. Specify this field as a binary value.
HARQID	Hybrid automatic repeat request process identifier, specified as a 1-byte decimal value in the range [0, 15].
SystemFrameNumber	System frame number, specified as an integer in the range [0, 1023].
SlotNumber	Slot number, specified as an integer in the range [0, 159]. This field identifies the slot in the 10 ms frame.

If you do not specify this name-value argument, the object function adds the default packet information to the MAC packet. The default packet information includes the default values of the *RadioType*, *LinkDir*, and *RNTIType* fields.

Data Types: `struct`

`PacketComment` — Comment for 5G NR MAC packet

`''` (default) | character vector | string scalar

Comment for 5G NR MAC packet, specified as a character vector or a string scalar.

Dependencies

To enable this name-value argument, specify the `FileExtension` property of the `nrPCAPWriter` object as `'pcapng'`. If you set the `FileExtension` property to `'pcap'`, the object function ignores this name-value argument.

Data Types: `char` | `string`

`PacketFormat` — Format of 5G NR MAC packet

`'octets'` (default) | `'bits'`

Format of the 5G NR MAC packet, specified as `'octets'` or `'bits'`. If this value is specified as `'octets'`, `packet` is specified as one of these values.

- Binary-valued vector – This value represents bits.
- Character vector – This value represents octets in hexadecimal format.
- String scalar – This value represents octets in hexadecimal format.
- Numeric vector with each element in the range [0, 255] – This value represents octets in decimal format.
- *n*-by-2 character array – In this value, each row represents an octet in hexadecimal format .

If this value is specified as `'bits'`, `packet` is specified as a binary-valued vector.

Data Types: `char` | `string` | `double`

Version History

Introduced in R2021b

References

[1] Tuexen, M. "PCAP Next Generation (Pcapng) Capture File Format." 2020. https://www.ietf.org/.

[2] "Development/LibpcapFileFormat - The Wireshark Wiki." Accessed May 20, 2020. https://www.wireshark.org.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
`nrPCAPWriter`

writeCustomBlock

Write custom block to PCAPNG file

Syntax

`writeCustomBlock(pcapngObj,customData)`

Description

`writeCustomBlock(pcapngObj,customData)` writes the custom block data, `customData`, to a PCAPNG file specified in the PCAPNG file writer object.

Examples

Write 5G NR User-Defined Custom Block to PCAPNG File

Create a default PCAPNG file writer object.

```
pcapngObj = pcapngWriter;
```

Write the interface block for 5G New Radio (NR). 5G NR packets do not have a valid link type. As per Tcpdump, if a valid link type is not present, specify the link type of SLL packet.

```
interface = '5GNR';         % Interface name
linkType = 113;             % Link type of SLL packet
interfaceID = writeInterfaceDescriptionBlock(pcapngObj,linkType,interface);
```

Write the custom block to specify user-defined data.

```
writeCustomBlock(pcapngObj,"This block writes user-defined data");
```

Input Arguments

Note The `pcapngWriter` object does not overwrite the existing PCAPNG file. Each time when you create this object, specify a unique PCAPNG file name.

`pcapngObj` — PCAPNG file writer object

`pcapngWriter` object

PCAPNG file writer object, specified as a `pcapngWriter` object.

`customData` — User-defined data

character vector | string scalar

User-defined data, specified as a character vector or a string scalar.

Data Types: `char` | `string`

Version History

Introduced in R2020b

References

[1] Tuexen, M. “PCAP Next Generation (Pcapng) Capture File Format.” 2020. https://www.ietf.org/.

[2] Group, The Tcpdump. “Tcpdump/Libpcap Public Repository.” Accessed May 20, 2020. https://www.tcpdump.org.

[3] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`write` | `writeInterfaceDescriptionBlock`

Objects
`pcapWriter` | `pcapngWriter`

writeGlobalHeader

Write global header to PCAP file

Syntax

`writeGlobalHeader(pcapObj,linkType)`

Description

`writeGlobalHeader(pcapObj,linkType)` writes a global header to the PCAP file specified in the PCAP file writer object, `pcapObj`. Input `linkType` specifies the unique identifier for a protocol.

Examples

Write Global Header for 5G NR Packet to PCAP File

Create a default PCAP file writer object.

```
pcapObj = pcapWriter;
```

5G NR packets do not have a valid link type. As per Tcpdump, if a valid link type is not present, specify the link type of SLL packet.

```
linkType = 113;
```

Use the PCAP file writer object and the link type of the SLL packet to write a global header for the 5G NR packet to the PCAP file.

```
writeGlobalHeader(pcapObj,linkType);
```

Input Arguments

Note The `pcapWriter` object does not overwrite the existing PCAP file. Each time when you create this object, specify a unique PCAP file name.

`pcapObj` — PCAP file writer object
`pcapWriter` object

PCAP file writer object, specified as a `pcapWriter` object.

`linkType` — Unique identifier for a protocol
integer in the range [0, 65,535].

Unique identifier for a protocol, specified as an integer in the range [0, 65535].

Data Types: `double`

Version History

Introduced in R2020b

References

[1] Group, The Tcpdump. “Tcpdump/Libpcap Public Repository.” Accessed May 20, 2020. https://www.tcpdump.org.

[2] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`write`

Objects
`pcapWriter` | `pcapngWriter`

writeInterfaceDescriptionBlock

Write interface description block to PCAPNG file

Syntax

`interfaceID = writeInterfaceDescriptionBlock(pcapngObj,linkType,interface)`

Description

`interfaceID = writeInterfaceDescriptionBlock(pcapngObj,linkType,interface)` writes an interface description block to the PCAPNG file specified in the PCAPNG file writer object, `pcapngObj`. Input `linkType` specifies the unique identifier for the protocol and input `interface` specifies the interface on which the protocol packets are captured. This object function returns the unique identifier for the interface.

Examples

Write 5G NR Interface Description Block to PCAPNG File

Create a default PCAPNG file writer object.

```
pcapngObj = pcapngWriter;
```

Write the interface block for 5G New Radio (NR). 5G NR packets do not have a valid link type. As per Tcpdump, if a valid link type is not present, specify the link type of SLL packet.

```
interface = '5GNR';        % Interface name
linkType = 113;            % Link type of SLL packet
interfaceID = writeInterfaceDescriptionBlock(pcapngObj,linkType,interface);
```

Input Arguments

Note The `pcapngWriter` object does not overwrite the existing PCAPNG file. Each time when you create this object, specify a unique PCAPNG file name.

`pcapngObj` — PCAPNG file writer object

`pcapngWriter` object

PCAPNG file writer object, specified as a `pcapngWriter` object.

`linkType` — Unique identifier for protocol

integer in the range [0, 65,535].

Unique identifier for a protocol, specified as an integer in the range [0, 65,535].

Data Types: `double`

`interface` — Name of the interface on which protocol packets are captured
character vector | string scalar

Name of the interface on which protocol packets are captured, specified as a character vector or a string scalar in 8-bit unicode transformation format (UTF-8) format.

Data Types: `char` | `string`

Output Arguments

`interfaceID` — Unique identifier for an interface
nonnegative scalar

Unique identifier for an interface, specified as a nonnegative scalar.

Data Types: `double`

Version History

Introduced in R2020b

References

[1] Tuexen, M. “PCAP Next Generation (Pcapng) Capture File Format.” 2020. https://www.ietf.org/.

[2] Group, The Tcpdump. “Tcpdump/Libpcap Public Repository.” Accessed May 20, 2020. https://www.tcpdump.org.

[3] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`write` | `writeCustomBlock`

Objects
`pcapWriter` | `pcapngWriter`

2

System Objects

nrCDLChannel

Send signal through CDL channel model

Description

The `nrCDLChannel` System object sends an input signal through a clustered delay line (CDL) multi-input multi-output (MIMO) link-level fading channel to obtain the channel-impaired signal. The object implements the following aspects of TR 38.901 [1]:

- Section 7.7.1: CDL models
- Section 7.7.3: Scaling of delays
- Section 7.7.5.1: Scaling of angles
- Section 7.7.6: K-factor for LOS channel models

To send a signal through the CDL MIMO channel model:

1. Create the `nrCDLChannel` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
cdl = nrCDLChannel
cdl = nrCDLChannel(Name,Value)
```

Description

`cdl = nrCDLChannel` creates a CDL MIMO channel System object.

`cdl = nrCDLChannel(Name,Value)` creates the object with properties set by using one or more name-value pairs. Enclose the property name inside quotes, followed by the specified value. Unspecified properties take default values.

Example: `cdl = nrCDLChannel('DelayProfile','CDL-D','DelaySpread',2e-6)` creates the channel object with CDL-D delay profile and 2-microseconds delay spread.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

Configurable Channel Properties

DelayProfile — CDL delay profile
'CDL-A' (default) | 'CDL-B' | 'CDL-C' | 'CDL-D' | 'CDL-E' | 'Custom'

CDL delay profile, specified as 'CDL-A', 'CDL-B', 'CDL-C', 'CDL-D', 'CDL-E', or 'Custom'. See TR 38.901 Section 7.7.1, Tables 7.7.1-1 to 7.7.1-5.

When you set this property to 'Custom', configure the delay profile using properties PathDelays, AveragePathGains, AnglesAoA, AnglesAoD, AnglesZoA, AnglesZoD, HasLOSCluster, KFactorFirstCluster, AngleSpreads, XPR, and NumStrongestClusters.

Data Types: char | string

PathDelays — Discrete path delays in seconds
0.0 (default) | numeric scalar | row vector

Discrete path delays in seconds, specified as a numeric scalar or row vector. AveragePathGains and PathDelays must have the same size.

Dependencies

To enable this property, set DelayProfile to 'Custom'.

Data Types: double

AveragePathGains — Average path gains in dB
0.0 (default) | numeric scalar | row vector

Average path gains in dB, also referred to as cluster powers in TR 38.901, specified as a numeric scalar or row vector. AveragePathGains and PathDelays must have the same size.

Dependencies

To enable this property, set DelayProfile to 'Custom'.

Data Types: double

AnglesAoA — Azimuth of arrival angle in degrees
0.0 (default) | numeric scalar | row vector

Azimuth of arrival angle in degrees, specified as a numeric scalar or row vector. The vector elements specify the angles for each cluster.

Dependencies

To enable this property, set DelayProfile to 'Custom'.

Data Types: double

AnglesAoD — Azimuth of departure angle in degrees
0.0 (default) | numeric scalar | row vector

Azimuth of departure angle in degrees, specified as a numeric scalar or row vector. The vector elements specify the angles for each cluster.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'`.

Data Types: `double`

`AnglesZoA` — Zenith of arrival angle in degrees

`0.0` (default) | numeric scalar | row vector

Zenith of arrival angle in degrees, specified as a numeric scalar or row vector. The vector elements specify the angles for each cluster.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'`.

Data Types: `double`

`AnglesZoD` — Zenith of departure angle in degrees

`0.0` (default) | numeric scalar | row vector

Zenith of departure angle in degrees, specified as a numeric scalar or row vector. The vector elements specify the angles for each cluster.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'`.

Data Types: `double`

`HasLOSCluster` — Line of sight cluster of the delay profile

`false` (default) | `true`

Line of sight (LOS) cluster of the delay profile, specified as `false` or `true`. The `PathDelays`, `AveragePathGains`, `AnglesAoA`, `AnglesAoD`, `AnglesZoA`, and `AnglesZoD` properties define the delay profile. To enable the LOS cluster of the delay profile, set `HasLOSCluster` to `true`.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'`.

Data Types: `logical`

`KFactorFirstCluster` — K-factor in first cluster of delay profile in dB

`13.3` (default) | numeric scalar

K-factor in the first cluster of the delay profile in dB, specified as a numeric scalar. The default value corresponds to the K-factor in the first cluster of CDL-D as defined in TR 38.901 Section 7.7.1, Table 7.7.1-4.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'` and `HasLOSCluster` to `true`.

Data Types: `double`

`AngleScaling` — Apply scaling of angles

`false` (default) | `true`

Apply scaling of angles, specified as `false` or `true` according to TR 38.901 Section 7.7.5.1. When set to `true`, the `AngleSpreads` and `MeanAngles` properties define the scaling of angles.

Dependencies

To enable this property, set `DelayProfile` to `'CDL-A'`, `'CDL-B'`, `'CDL-C'`, `'CDL-D'`, or `'CDL-E'`. This property does not apply for custom delay profile.

Data Types: `logical`

`AngleSpreads` — Scaled or cluster-wise RMS angle spreads in degrees
`[5.0 11.0 3.0 3.0]` (default) | four-element row vector

Scaled or cluster-wise root mean square (RMS) angle spreads in degrees, specified as a four-element row vector in one of these forms:

- [*ASD ASA ZSD ZSA*] — Use this vector to specify the desired RMS angle spreads of the channel, as described in TR 38.901 Section 7.7.5.1 ($AS_{desired}$), where:

  - *ASD* is the RMS azimuth spread of departure angles
  - *ASA* is the RMS azimuth spread of arrival angles
  - *ZSD* is the RMS zenith spread of departure angles
  - *ZSA* is the RMS zenith spread of arrival angles

  To use this form, set `AngleScaling` to `true` and `DelayProfile` to `'CDL-A'`, `'CDL-B'`, `'CDL-C'`, `'CDL-D'`, or `'CDL-E'`.
- [C_{ASD} C_{ASA} C_{ZSD} C_{ZSA}] — Use this vector to specify cluster-wise RMS angle spreads for scaling ray offset angles within a cluster, as described in TR 38.901 Section 7.7.1, Step1, where:

  - C_{ASD} is the cluster-wise RMS azimuth spread of departure angles
  - C_{ASA} is the cluster-wise RMS azimuth spread of arrival angles
  - C_{ZSD} is the cluster-wise RMS zenith spread of departure angles
  - C_{ZSA} is the cluster-wise RMS zenith spread of arrival angles

  To use this form, set `DelayProfile` to `'Custom'`. Based on TR 38.901 Section 7.7.5.1, the object does not perform angle scaling in this case.

The default value corresponds to the default cluster-wise angle spreads of CDL-A as defined in TR 38.901 Section 7.7.1 Table 7.7.1-1.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'` or `AngleScaling` to `true`.

Data Types: `double`

`MeanAngles` — Scaled mean angles in degrees
`[0.0 0.0 0.0 0.0]` (default) | four-element row vector

Scaled mean angles in degrees, specified as a four-element row vector of the form [*AoD AoA ZoD ZoA*].

- *AoD* is the mean azimuth of departure angles after scaling
- *AoA* is the mean azimuth of arrival angles after scaling

- *ZoD* is the mean zenith of departure angles after scaling
- *ZoA* is the mean zenith of arrival angles after scaling

Use this vector to specify the desired mean angles of the channel used for angle scaling, as described in TR 38.901 Section 7.7.5.1 ($\mu_{\Phi,\text{desired}}$).

Dependencies

To enable this property, set `AngleScaling` to `true`.

Data Types: `double`

`RayCoupling` — Coupling of departure and arrival rays within a cluster
`'Random'` (default) | *N*-by-*M*-by-3 numeric array

Coupling of departure and arrival rays within a cluster for azimuth and elevation, specified as one of these values.

- `'Random'` — The object randomly couples the rays, as defined in TR 38.901 Section 7.5 Step 8, using the random number stream specified by the `RandomStream` property.
- *N*-by-*M*-by-3 numeric array — Use this array to explicitly define the ray coupling. *N* is the number of clusters, equal to the number of path delays, specified by the `PathDelays` property. *M* is the number of rays per cluster, equal to 20. The three *N*-by-*M* planes, in the third dimension, correspond to the *AoD*/*AoA*, *ZoD*/*ZoA*, and *AoD*/*ZoD* ray couplings, respectively. Each row in each *N*-by-*M* plane specifies the ray coupling within the corresponding cluster by using a permutation of ray indices from 1 to *M*.

  Note

  - *N* is the number of clusters before any splitting into subclusters (see the `NumStrongestClusters` property).
  - *N* does not count the LOS cluster that is specified by the `HasLOSCluster` property.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'`.

Data Types: `double` | `char` | `string`

XPR — Cross-polarization power ratio in dB
`10.0` (default) | numeric scalar | *N*-by-*M* numeric matrix

Cross-polarization power ratio in dB, specified as a numeric scalar or an *N*-by-*M* numeric matrix. *N* is the number of clusters, equal to the number of path delays, specified by the `PathDelays` property. *M* is the number of rays per cluster, equal to 20. The default value corresponds to the cluster-wise cross-polarization power ratio of CDL-A as defined in TR 38.901 Section 7.7.1, Table 7.7.1-1.

Note

- *N* is the number of clusters before any splitting into subclusters (see the `NumStrongestClusters` property).
- *N* does not count the LOS cluster that is specified by the `HasLOSCluster` property.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'`.

Data Types: `double`

`InitialPhases` — Initial phases in degrees

`'Random'` (default) | *N*-by-*M*-by-4 numeric array

Initial phases of all rays for the four polarization combinations in degrees, specified as one of these values.

- `'Random'` — The object draws uniformly distributed random phases, as defined in TR 38.901 Section 7.5 Step 10, using the random number stream specified by the `RandomStream` property.
- *N*-by-*M*-by-4 numeric array — Use this option to explicitly define the initial phases. *N* is the number of clusters, equal to the number of path delays, specified by the `PathDelays` property. *M* is the number of rays per cluster, equal to 20. The four *N*-by-*M* planes, in the third dimension, correspond to the θ/θ, θ/ϕ, ϕ/θ, ϕ/ϕ polarization combinations, respectively.

Note

- *N* is the number of clusters before any splitting into subclusters (see the `NumStrongestClusters` property).
- *N* does not count the LOS cluster that is specified by the `HasLOSCluster` property.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'`.

Data Types: `double` | `char` | `string`

`DelaySpread` — Desired RMS delay spread in seconds

`30e-9` (default) | numeric scalar

Desired RMS delay spread in seconds, specified as a numeric scalar. For examples of desired RMS delay spreads, $DS_{desired}$, see TR 38.901 Section 7.7.3 and Tables 7.7.3-1 and 7.7.3-2.

Dependencies

To enable this property, set `DelayProfile` to `'CDL-A'`, `'CDL-B'`, `'CDL-C'`, `'CDL-D'`, or `'CDL-E'`. This property does not apply for custom delay profile.

Data Types: `double`

`CarrierFrequency` — Carrier frequency in Hz

`4e9` (default) | numeric scalar

Carrier frequency in Hz, specified as a numeric scalar.

Data Types: `double`

`MaximumDopplerShift` — Maximum Doppler shift in Hz

`5` (default) | nonnegative numeric scalar

Maximum Doppler shift in Hz, specified as a nonnegative numeric scalar. This property applies to all channel paths. When the maximum Doppler shift is set to 0, the channel remains static for the entire input. To generate a new channel realization, reset the object by calling the `reset` function.

Data Types: `double`

`UTDirectionOfTravel` — User terminal direction of travel in degrees
`[0; 90]` (default) | two-element column vector

User terminal (or user equipment) direction of travel in degrees, specified as a two-element column vector. The vector elements specify the azimuth and the zenith components [azimuth; zenith].

Data Types: `double`

`KFactorScaling` — K-factor scaling
`false` (default) | `true`

K-factor scaling, specified as `false` or `true`. When set to `true`, the `KFactor` property specifies the desired K-factor and the object applies K-factor scaling as described in TR 38.901 Section 7.7.6.

Note K-factor scaling modifies both the path delays and path powers.

Dependencies

To enable this property, set `DelayProfile` to `'CDL-D'` or `'CDL-E'`.

Data Types: `double`

`KFactor` — Desired K-factor for scaling in dB
`9.0` (default) | numeric scalar

Desired K-factor for scaling in dB, specified as a numeric scalar. For typical K-factor values, see TR 38.901 Section 7.7.6 and Table 7.5-6.

Note

- K-factor scaling modifies both the path delays and path powers.
- `K-factor` applies to the overall delay profile. Specifically, the K-factor before the scaling is K_{model}, as described in TR 38.901 Section 7.7.6. K_{model} is the ratio of the power of the first path LOS to the total power of all the Laplacian clusters, including the Laplacian part of the first cluster.

Dependencies

To enable this property, set `KFactorScaling` to `true`.

Data Types: `double`

`SampleRate` — Sample rate of input signal in Hz
`30720000` (default) | positive numeric scalar

Sample rate of the input signal in Hz, specified as a positive numeric scalar.

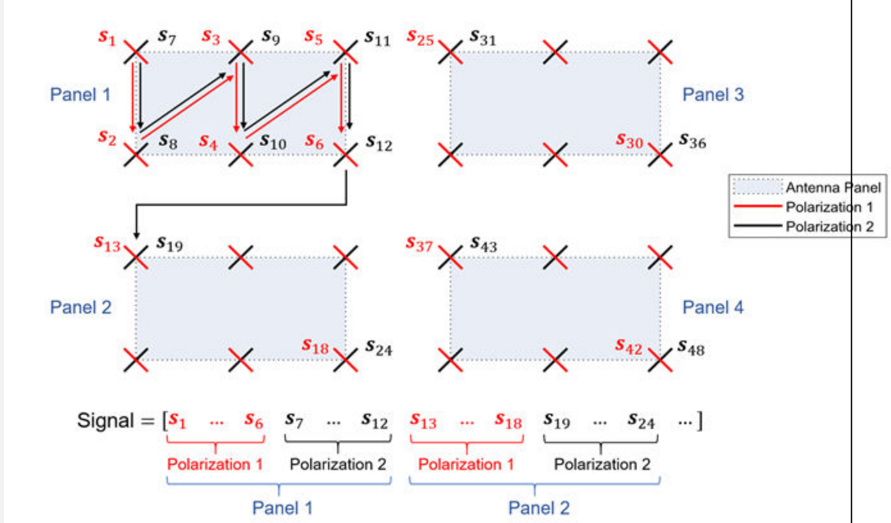
Data Types: `double`

`TransmitAntennaArray` — Transmit antenna array characteristics
structure (default) | phased array

Transmit antenna array characteristics, specified as a structure or a phased array (requires Phased Array System Toolbox™).

Phased arrays enable you to specify different antenna array configurations, including predefined and custom antenna elements. You can design custom antenna elements by using Phased Array System Toolbox or Antenna Toolbox™ features. To specify custom antenna elements in a 5G rectangular multipanel array, as defined in TR 38.901 Section 7.3, use the `phased.NRRectangularPanelArray` object. For an overview of phased arrays, see "Array Geometries and Analysis" (Phased Array System Toolbox).

When specified as a structure, this property contains these fields:

Parameter Field	Values	Description
Size	`[2 2 2 1 1]` (default), row vector	Size of antenna array [M N P M_g N_g], where: • M and N are the number of rows and columns in the antenna array, respectively. • P is the number of polarizations (1 or 2). • M_g and N_g are the number of row and column array panels, respectively. The `nrCDLChannel` System object maps the input signal `signalIn` to the antenna array elements panel-wise, in the order that a 5-D array of size M-by-N-by-P-by-M_g-by-N_g is linearly indexed across the first dimension to the last. For example, this figure shows how the object maps the input signal `signalIn` to an antenna array of size `[2 3 2 2 2]`. The antenna array consists of 2-by-2 antenna panels of 2-by-3 elements with 2 polarizations. The object maps the first M = 2 columns of the input signal (s_1 and s_2) to the first column of antenna elements with the first polarization angle of the first panel. The next M = 2 columns of the input signal (s_3 and s_4) are mapped to the next column of antenna elements, and so on. Following this pattern, the object maps the first $M \times N$ = 6 columns of the input signal (s_1 to s_6) to the antenna elements with the first polarization angle of the complete first panel. Similarly, the next 6 columns of the input signal (s_7 to s_{12}) are mapped to the antenna elements with the second polarization angle of the first panel. Subsequent sets of $M \times N \times P$ = 12 columns of the input signal (s_{13} to s_{24}, s_{25} to s_{36}, s_{37} to s_{48}) are mapped to consecutive panels, taking panel rows first, then panel columns. 

Parameter Field	Values	Description
ElementSpacing	[0.5 0.5 1.0 1.0] (default), row vector	Element spacing, in wavelengths, specified as a row vector of the form [λ_v λ_h dg_v dg_h]. The vector elements represent the vertical and horizontal element spacing and the vertical and horizontal panel spacing, respectively. The panel spacing is measured from the center of the panels.
PolarizationAngles	[45 -45] (default), row vector	Polarization angles in degrees, specified as a row vector of the form [θ ρ].
Orientation (to be removed)	[0; 0; 0] (default), column vector	**Note** This field will be removed in a future release. Use the TransmitArrayOrientation property instead. Mechanical orientation of the array, in degrees, specified as a column vector of the form [α; β; γ]. The vector elements specify the bearing, downtilt, and slant, respectively. The default value indicates that the broadside direction of the array points to the positive *x*-axis.
Element	'38.901' (default), 'isotropic'	Antenna element radiation pattern as described in TR 38.901 Section 7.3. (Note that TR 38.901 superseded TR 38.900.)
PolarizationModel	'Model-2' (default), 'Model-1'	Model that determines the radiation field patterns based on a defined radiation power pattern. For more information, see TR 38.901 Section 7.3.2.

TransmitArrayOrientation — Mechanical orientation of transmit antenna array
[0; 0; 0] (default) | three-element numeric column vector

Mechanical orientation of the transmit antenna array, specified as a three-element numeric column vector of the form [α; β; γ]. The vector elements specify the bearing, downtilt, and slant rotation angles in degrees, respectively, as specified in TR 38.901 Section 7.1.3. The object applies these rotation angles relative to the default array orientation in the local coordinate system. The default array orientation, corresponding to the value [0; 0; 0], depends on the TransmitAntennaArray property.

- If you specify the TransmitAntennaArray property as a structure (default), in the default array orientation, the broadside direction points to the positive *x*-axis.
- If you specify the TransmitAntennaArray property as a phased array (requires Phased Array System Toolbox), you can configure the default array orientation by setting the relevant array properties of the specified phased array object.

To visualize and evaluate the resulting array orientation, call the displayChannel function on the nrCDLChannel channel model.

For an example of orienting transmit and receive antennas towards each other, see “Orient Transmit and Receive Antennas Using LOS Path Angles” on page 2-24.

Data Types: double

ReceiveAntennaArray — Receive antenna array characteristics
structure (default) | phased array

Receive antenna array characteristics, specified as a structure or a phased array (requires Phased Array System Toolbox).

Phased arrays enable you to specify different antenna array configurations, including predefined and custom antenna elements. You can design custom antenna elements by using Phased Array System Toolbox or Antenna Toolbox features. To specify custom antenna elements in a 5G rectangular multipanel array, as defined in TR 38.901 Section 7.3, use the `phased.NRRectangularPanelArray` object. For an overview of phased arrays, see “Array Geometries and Analysis” (Phased Array System Toolbox).

When specified as a structure, this property contains these fields:

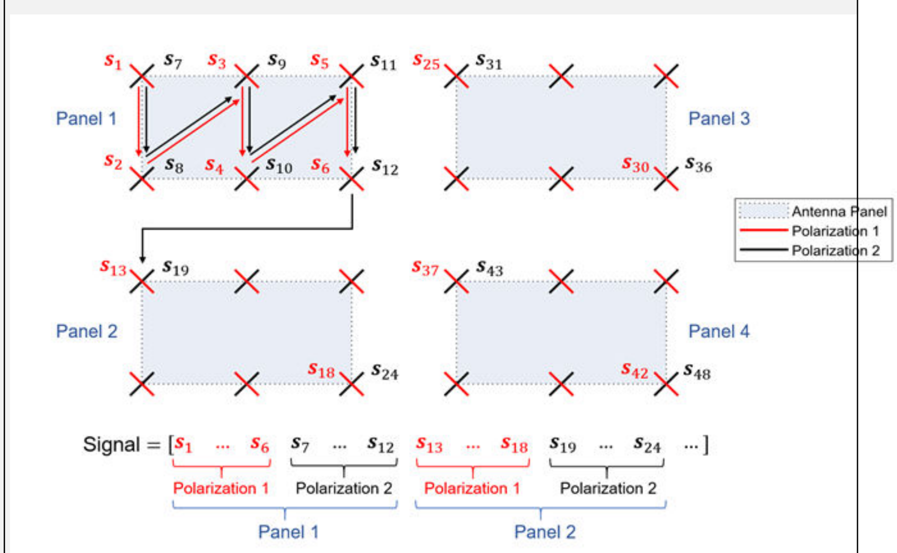
<table>
<tr><th>Parameter Field</th><th>Values</th><th>Description</th></tr>
<tr>
<td><code>Size</code></td>
<td><code>[1 1 2 1 1]</code> (default),

row vector</td>
<td>
Size of antenna array $[M\ N\ P\ M_g\ N_g]$, where:

M and N are the number of rows and columns in the antenna array, respectively.
P is the number of polarizations (1 or 2).
M_g and N_g are the number of row and column array panels, respectively.

The <code>nrCDLChannel</code> System object maps the antenna array elements to the output signal <code>signalOut</code> panel-wise, in the order that a 5-D array of size M-by-N-by-P-by-M_g-by-N_g is linearly indexed across the first dimension to the last.

For example, this figure shows how the object maps an antenna array of size <code>[2 3 2 2 2]</code> to the output signal <code>signalOut</code>. The antenna array consists of 2-by-2 antenna panels of 2-by-3 elements with 2 polarizations. The first column of antenna elements with the first polarization angle of the first panel are mapped to the first M = 2 columns of the output signal (s_1 and s_2). The next column of antenna elements are mapped to the next M = 2 columns of the output signal (s_3 and s_4), and so on. Following this pattern, the object maps the antenna elements with the first polarization angle of the complete first panel to the first $M \times N$ = 6 columns of the output signal (s_1 to s_6). Similarly, the antenna elements with the second polarization angle of the first panel are mapped to the next 6 columns of the output signal (s_7 to s_{12}). Consecutive panels are mapped to subsequent sets of $M \times N \times P$ = 12 columns of the output signal (s_{13} to s_{24}, s_{25} to s_{36}, s_{37} to s_{48}), taking panel rows first, then panel columns.



</td>
</tr>
</table>

Parameter Field	Values	Description
ElementSpacing	`[0.5 0.5 0.5 0.5]` (default), row vector	Element spacing, in wavelengths, specified as a row vector of the form [λ_v λ_h dg_v dg_h]. The vector elements represent the vertical and horizontal element spacing and the vertical and horizontal panel spacing, respectively. The panel spacing is measured from the center of the panels.
PolarizationAngles	`[0 90]` (default), row vector	Polarization angles in degrees, specified as a row vector of the form [θ ρ].
Orientation (to be removed)	`[0; 0; 0]` (default), column vector	**Note** This field will be removed in a future release. Use the `ReceiveArrayOrientation` property instead. Mechanical orientation of the array, in degrees, specified as a column vector of the form [α; β; γ]. The vector elements specify the bearing, downtilt, and slant, respectively. The default value indicates that the broadside direction of the array points to the positive *x*-axis.
Element	`'isotropic'` (default), `'38.901'`	Antenna element radiation pattern as described in TR 38.901 Section 7.3. (Note that TR 38.901 superseded TR 38.900.)
PolarizationModel	`'Model-2'` (default), `'Model-1'`	Model that determines the radiation field patterns based on a defined radiation power pattern. For more information, see TR 38.901 Section 7.3.2.

ReceiveArrayOrientation — Mechanical orientation of receive antenna array
`[0; 0; 0]` (default) | three-element numeric column vector

Mechanical orientation of the receive antenna array, specified as a three-element numeric column vector of the form [α; β; γ]. The vector elements specify the bearing, downtilt, and slant rotation angles in degrees, respectively, as specified in TR 38.901 Section 7.1.3. The object applies these rotation angles relative to the default array orientation in the local coordinate system. The default array orientation, corresponding to the value `[0; 0; 0]`, depends on the `ReceiveAntennaArray` property.

- If you specify the `ReceiveAntennaArray` property as a structure (default), in the default array orientation, the broadside direction points to the positive x-axis.
- If you specify the `ReceiveAntennaArray` property as a phased array (requires Phased Array System Toolbox), you can configure the default array orientation by setting the relevant array properties of the specified phased array object.

To visualize and evaluate the resulting array orientation, call the `displayChannel` function on the `nrCDLChannel` channel model.

For an example of orienting transmit and receive antennas towards each other, see “Orient Transmit and Receive Antennas Using LOS Path Angles” on page 2-24.

Data Types: `double`

SampleDensity — Number of time samples per half wavelength
`64` (default) | `Inf` | numeric scalar

Number of time samples per half wavelength, specified as `Inf` or a numeric scalar. The `SampleDensity` and `MaximumDopplerShift` properties control the coefficient generation sampling rate, *Fcg*, given by

Fcg = `MaximumDopplerShift` × 2 × `SampleDensity`.

Setting `SampleDensity` to `Inf` assigns *Fcg* the value of the `SampleRate` property.

For an example of how sample density affects the channel output and path gains, see “Explore the Effect of SampleDensity Property in CDL Channel Output” on page 2-22.

Data Types: `double`

`NormalizePathGains` — Normalized channel fading process
`true` (default) | `false`

Normalized channel fading process, specified as `true` or `false`. When this property is set to `true`, the amplitude of the channel fading process is normalized by the average path gains (also referred to as cluster powers in TR 38.901). This normalization does not include other channel gains, for example, polarization and antenna element directivity. When this property is set to `false`, the channel fading process is not normalized. The `DelayProfile` property determines the average path gains, based on TR 38.901 Section 7.7.1, Tables 7.7.1-1 to 7.7.1-5. When you set `DelayProfile` to `'Custom'`, you can specify the average path gains with the `AveragePathGains` property.

Data Types: `logical`

`InitialTime` — Time offset of fading process in seconds
`0.0` (default) | numeric scalar

Time offset of fading process in seconds, specified as a numeric scalar.

Tunable: Yes

Data Types: `double`

`NumStrongestClusters` — Number of strongest clusters to split into subclusters
`0` (default) | numeric scalar

Number of strongest clusters to split into subclusters, specified as a numeric scalar. See TR 38.901 Section 7.5, Step 11.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'`.

Data Types: `double`

`ClusterDelaySpread` — Cluster delay spread in seconds
`3.90625e-9` (default) | nonnegative scalar

Cluster delay spread in seconds, specified as a nonnegative scalar. Use this property to specify the delay offset between subclusters for clusters split into subclusters. See TR 38.901 Section 7.5, Step 11.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'` and `NumStrongestClusters` to a value greater than zero.

Data Types: double

RandomStream — Source of random number stream
'mt19937ar with seed' (default) | 'Global stream'

Source of the random number stream to initialize the ray phases and coupling using uniformly distributed random numbers, specified as one of these values.

- 'mt19937ar with seed' — The object uses the mt19937ar algorithm for the random number generation. Calling the reset function resets the filters and reinitializes the random number stream to the value of the Seed property. Specifying this value results in repeatable channel fading.
- 'Global stream' — The object uses the current global random number stream for the random number generation. Calling the reset function resets only the filters.

Dependencies

To enable this property, set the RayCoupling or InitialPhases properties to 'Random'.

Seed — Initial seed of mt19937ar random number stream
73 (default) | nonnegative numeric scalar

Initial seed of mt19937ar random number stream, specified as a nonnegative numeric scalar.

Dependencies

To enable this property, set RandomStream to 'mt19937ar with seed'. When calling the reset function, the seed reinitializes the mt19937ar random number stream.

Data Types: double

NormalizeChannelOutputs — Normalize channel outputs
true (default) | false

Normalize channel outputs, specified as true or false. When this property is set to true, the channel outputs are normalized. The normalization is by N_R, where N_R is the number of receive antenna elements or the number of antenna subarrays (only when you specify the ReceiveAntennaArray property as a phased.ReplicatedSubarray or phased.PartitionedArray phased array object). To determine the value of N_R, check the NumOutputSignals structure field in the output of the info(cdl) object function call.

Note When you call the swapTransmitAndReceive function to reverse the role of the transmit and receive antennas within the channel, the function also swaps these output structure fields of the info(cdl) function call:

- NumTransmitAntennas and NumReceiveAntennas
- NumInputSignals and NumOutputSignals

Hence the normalization is always by N_R.

Dependencies

To enable this property, set ChannelFiltering to true.

Data Types: logical

ChannelFiltering — Fading channel filtering
true (default) | false

Fading channel filtering, specified as true or false. When this property is set to false, these conditions apply.

- The object takes no input signal and returns only the path gains and sample times.
- The SampleDensity property determines when to sample the channel coefficients.
- The NumTimeSamples property controls the duration of the fading process realization at a sampling rate given by the SampleRate property.

For a use case of disabling channel filtering, see the “CDL Channel Model Customization with Ray Tracing” example.

Data Types: logical

NumTimeSamples — Number of time samples
30720 (default) | positive integer

Number of time samples, specified as a positive integer. Use this property to set the duration of the fading process realization.

Tunable: Yes

Dependencies

To enable this property, set ChannelFiltering to false.

Data Types: double

OutputDataType — Data type of generated path gains
'double' (default) | 'single'

Data type of generated path gains, specified as 'double' or 'single'.

Dependencies

To enable this property, set ChannelFiltering to false.

Data Types: double

Nonconfigurable Channel Properties

TransmitAndReceiveSwapped — Reversed channel link direction
false (default) | true

This property is read-only.

Reversed channel link direction, returned as one of these values.

- false — The role of the transmit and receive antennas within the channel model corresponds to the original channel link direction. Calling the swapTransmitAndReceive function on the nrCDLChannel object reverses the link direction of the channel and toggles this property value from false to true.
- true — The role of the transmit and receive antennas within the channel model are swapped. Calling the swapTransmitAndReceive function on the nrCDLChannel object restores the original link direction of the channel and toggles this property value from true to false.

Data Types: `logical`

Usage

Syntax

```
signalOut = cdl(signalIn)
[signalOut,pathGains] = cdl(signalIn)
[signalOut,pathGains,sampleTimes] = cdl(signalIn)

[pathGains,sampleTimes] = cdl()
```

Description

`signalOut = cdl(signalIn)` sends the input signal through a CDL MIMO fading channel and returns the channel-impaired signal.

`[signalOut,pathGains] = cdl(signalIn)` also returns the MIMO channel path gains of the underlying fading process.

`[signalOut,pathGains,sampleTimes] = cdl(signalIn)` also returns the sample times of the channel snapshots of `pathGains` (first-dimension elements).

`[pathGains,sampleTimes] = cdl()` returns only the path gains and the sample times. The `cdl` object acts as a source of the path gains and sample times without filtering an input signal. The `NumTimeSamples` object property specifies the duration of the fading process and the `OutputDataType` object property specifies the data type of the generated path gains. To use this syntax, you must set the `ChannelFiltering` object property to `false`.

Input Arguments

`signalIn` — Input signal
complex scalar | column vector | N_S-by-N_T matrix

Input signal, specified as a complex scalar, column vector, or N_S-by-N_T matrix, where:

- N_S is the number of samples.
- N_T is the number of transmit antenna elements or the number of antenna subarrays (only when you specify the `TransmitAntennaArray` property as a `phased.ReplicatedSubarray` or `phased.PartitionedArray` phased array object). To determine the value of N_T, check the `NumInputSignals` structure field in the output of the `info(cdl)` object function call.

Data Types: `single` | `double`
Complex Number Support: Yes

Output Arguments

`signalOut` — Output signal
complex scalar | vector | N_S-by-N_R matrix

Output signal, returned as a complex scalar, vector, or N_S-by-N_R matrix, where:

- N_S is the number of samples.

- N_R is the number of receive antenna elements or the number of antenna subarrays (only when you specify the `ReceiveAntennaArray` property as a `phased.ReplicatedSubarray` or `phased.PartitionedArray` phased array object). To determine the value of N_R, check the `NumOutputSignals` structure field in the output of the `info(cdl)` object function call.

The output signal data type is of the same precision as the input signal data type.

Data Types: `single` | `double`
Complex Number Support: Yes

`pathGains` — MIMO channel path gains of fading process
N_{CS}-by-N_P-by-N_T-by-N_R complex array

MIMO channel path gains of the fading process, returned as an N_{CS}-by-N_P-by-N_T-by-N_R complex array, where:

- N_{CS} is the number of channel snapshots, controlled by the `SampleDensity` property of `cdl`.
- N_P is the number of paths, specified by the size of the `PathDelays` property of `cdl`.
- N_T is the number of transmit antenna elements or the number of antenna subarrays (only when you specify the `TransmitAntennaArray` property as a `phased.ReplicatedSubarray` or `phased.PartitionedArray` phased array object). To determine the value of N_T, check the `NumInputSignals` structure field in the output of the `info(cdl)` object function call.
- N_R is the number of receive antenna elements or the number of antenna subarrays (only when you specify the `ReceiveAntennaArray` property as a `phased.ReplicatedSubarray` or `phased.PartitionedArray` phased array object). To determine the value of N_R, check the `NumOutputSignals` structure field in the output of the `info(cdl)` object function call.

The path gains data type is of the same precision as the input signal data type.

Data Types: `single` | `double`
Complex Number Support: Yes

`sampleTimes` — Sample times of channel snapshots
N_{CS}-by-1 column vector

Sample times of channel snapshots, returned as an N_{CS}-by-1 column vector, where N_{CS} is the number of channel snapshots controlled by the `SampleDensity` property.

Data Types: `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to `nrCDLChannel`

info	Characteristic information of link-level MIMO channel
getPathFilters	Get path filter impulse response for link-level MIMO channel
displayChannel	Visualize and explore CDL channel model characteristics
swapTransmitAndReceive	Reverse link direction in CDL channel model

Common to All System Objects

step	Run System object algorithm
clone	Create duplicate System object
isLocked	Determine if System object is in use
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Transmission Over Channel Model with Delay Profile CDL-D

Transmit waveform through a clustered delay line (CDL) channel model with delay profile CDL-D from TR 38.901 Section 7.7.1.

Define the channel configuration structure using an `nrCDLChannel` System object. Use delay profile CDL-D, a delay spread of 10 ns, and UE velocity of 15 km/h:

```
v = 15.0;                    % UE velocity in km/h
fc = 4e9;                    % carrier frequency in Hz
c = physconst('lightspeed'); % speed of light in m/s
fd = (v*1000/3600)/c*fc;     % UE max Doppler frequency in Hz

cdl = nrCDLChannel;
cdl.DelayProfile = 'CDL-D';
cdl.DelaySpread = 10e-9;
cdl.CarrierFrequency = fc;
cdl.MaximumDopplerShift = fd;
```

Configure the transmit array layout as a vector of the form [*M N P Mg Ng*] = [2 4 2 1 2], representing 2 panels (*Mg*=1, *Ng*=2) with a 2-by-4 antenna array (*M*=2, *N*=4) and two polarization angles (*P*=2). Configure the receive antenna array as a vector of the form [*M N P Mg Ng*] = [1 1 2 1 1], representing a single pair of cross-polarized co-located antennas.

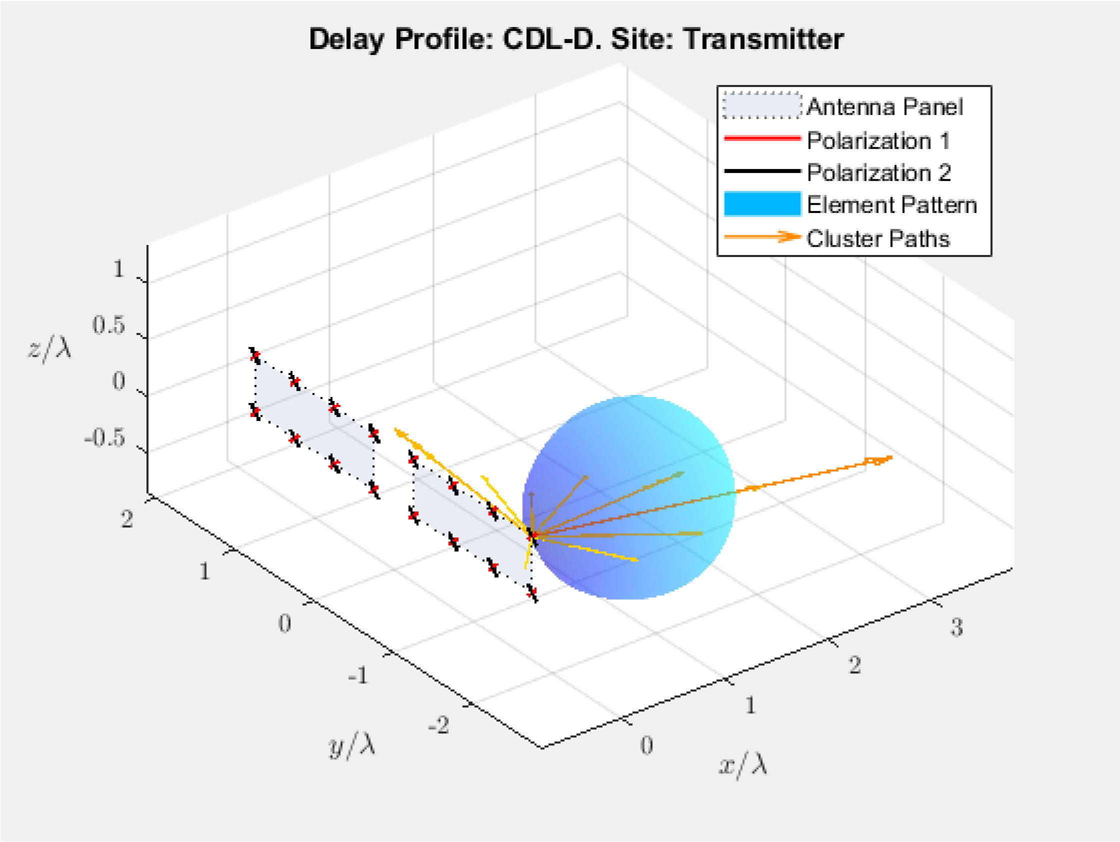
```
cdl.TransmitAntennaArray.Size = [2 4 2 1 2];
cdl.ReceiveAntennaArray.Size = [1 1 2 1 1];
```

Set the distance between the transmit antenna elements to half wavelength. Specify the distance between the antenna panel centers to evenly distribute the antenna elements of all panels and avoid panel overlapping.

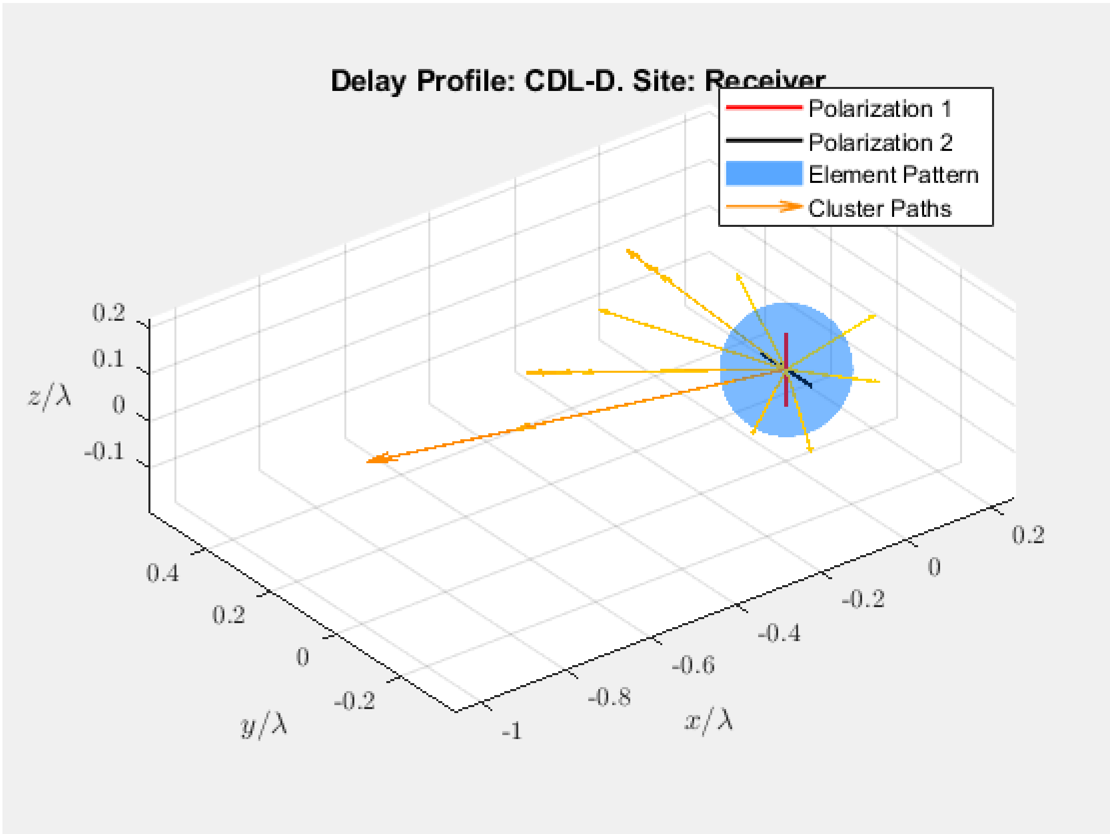
```
cdl.TransmitAntennaArray.ElementSpacing(1:2) = 0.5;
cdl.TransmitAntennaArray.ElementSpacing(3:4) = cdl.TransmitAntennaArray.ElementSpacing(1:2).*(cd
```

Verify the configuration by displaying the channel.

```
displayChannel(cdl,'LinkEnd','Tx')
```




```
displayChannel(cdl,'LinkEnd','Rx')
```



Create a random waveform of 1 subframe duration with 8 antennas.

```
SR = 15.36e6;
T = SR * 1e-3;
cdl.SampleRate = SR;
cdlinfo = info(cdl);
Nt = cdlinfo.NumInputSignals;

txWaveform = complex(randn(T,Nt),randn(T,Nt));
```

Transmit the input waveform through the channel.

```
rxWaveform = cdl(txWaveform);
```

Explore the Effect of SampleDensity Property in CDL Channel Output

Plot channel output and path gain snapshots for various sample density values while using an `nrCDLChannel` System object.

Configure a channel with delay profile CDL-B from TR 38.901 Section 7.7.1. Set the maximum Doppler shift to 300 Hz and the channel sampling rate to 10 kHz.

```
cdl = nrCDLChannel;
cdl.DelayProfile = 'CDL-B';
cdl.MaximumDopplerShift = 300.0;
```

```
cdl.SampleRate = 10e3;
cdl.Seed = 19;
```

Configure the transmit and receive antenna arrays for single-input/single-output (SISO) operation.

```
cdl.TransmitAntennaArray.Size = [1 1 1 1 1];
cdl.ReceiveAntennaArray.Size = [1 1 1 1 1];
```

Create an input waveform with a length of 40 samples.

```
T = 40;
in = ones(T,1);
```

Plot the step response of the channel (displayed as lines) and the corresponding path gain snapshots (displayed circles) for various values of the `SampleDensity` property. The sample density property controls how often the channel snapshots are taken relative to the Doppler frequency.

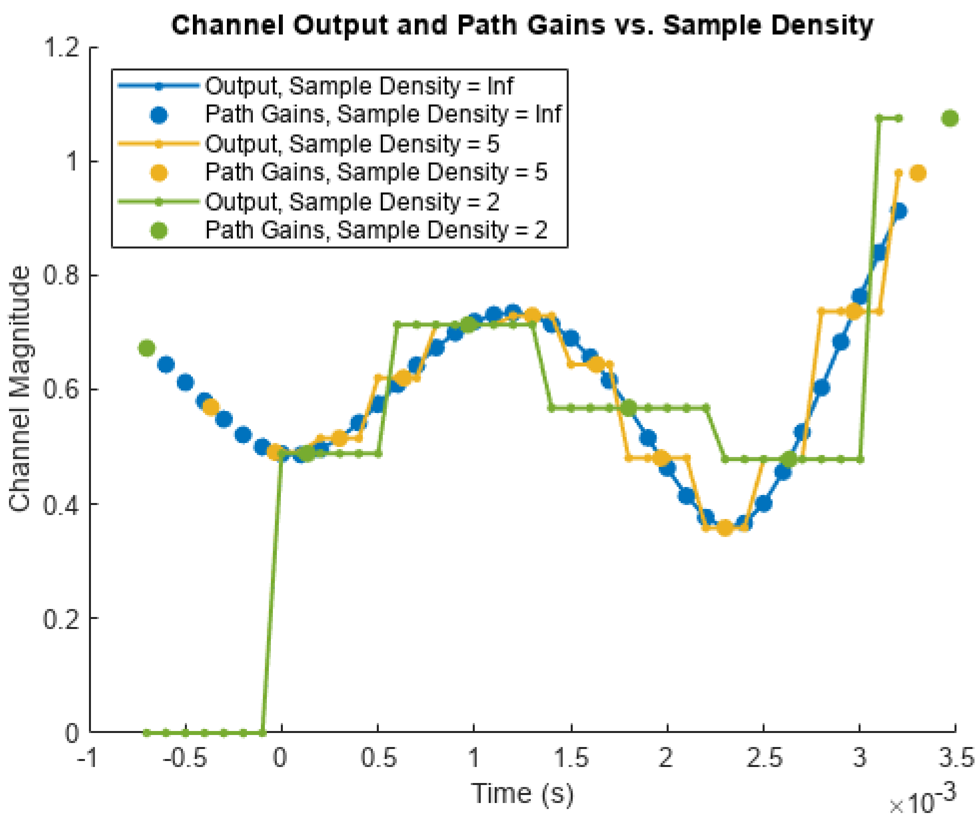
- When `SampleDensity` is set to `Inf`, a channel snapshot is taken for every input sample.
- When `SampleDensity` is set to a scalar *S*, a channel snapshot is taken at a rate of $F_{CS} = 2S \times MaximumDopplerShift$.

The `nrCDLChannel` object applies the channel snapshots to the input waveform by means of zero-order hold interpolation. The object takes an extra snapshot beyond the end of the input. Some of the final output samples use this extra value to minimize the interpolation error. The channel output contains a transient (and a delay) due to the filters that implement the path delays.

```
s = [Inf 5 2]; % sample densities

legends = {};
figure; hold on;
SR = cdl.SampleRate;
for i = 1:length(s)

    % call channel with chosen sample density
    release(cdl); cdl.SampleDensity = s(i);
    [out,pathgains,sampletimes] = cdl(in);
    chInfo = info(cdl); tau = chInfo.ChannelFilterDelay;

    % plot channel output against time
    t = cdl.InitialTime + ((0:(T-1)) - tau).' / SR;
    h = plot(t,abs(out),'o-');
    h.MarkerSize = 2;
    h.LineWidth = 1.5;
    desc = ['Sample Density = ' num2str(s(i))];
    legends = [legends ['Output, ' desc]];
    disp([desc ', Ncs = ' num2str(length(sampletimes))]);

    % plot path gains against sample times
    h2 = plot(sampletimes-tau/SR,abs(sum(pathgains,2)),'o');
    h2.Color = h.Color;
    h2.MarkerFaceColor = h.Color;
    legends = [legends ['Path Gains, ' desc]];
end
```

```
Sample Density = Inf, Ncs = 40
Sample Density = 5, Ncs = 13
Sample Density = 2, Ncs = 6
```

```
xlabel('Time (s)');
title('Channel Output and Path Gains vs. Sample Density');
ylabel('Channel Magnitude');
legend(legends,'Location','NorthWest');
```



Orient Transmit and Receive Antennas Using LOS Path Angles

Create a CDL channel model. Then specify a light-of-sight (LOS) channel.

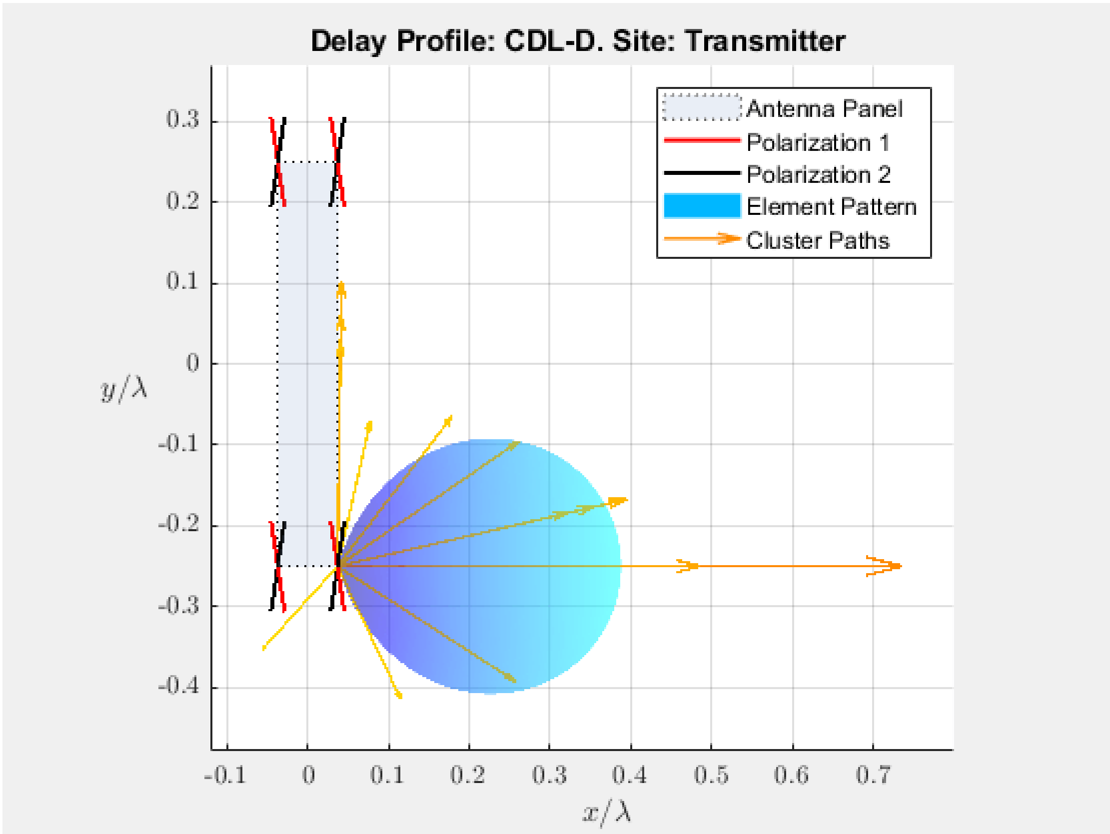
```
cdl = nrCDLChannel;
cdl.DelayProfile = 'CDL-D'; % LOS channel
cdl.TransmitAntennaArray.Element = '38.901';
cdl.ReceiveAntennaArray.Element = '38.901';
```

Retrieve channel characteristic information. Orient the transmit and receive antenna arrays to point at each other by using the LOS path angles returned in the characteristic information.

```
cdlInfo = cdl.info;
cdl.TransmitArrayOrientation = [cdlInfo.AnglesAoD(1) cdlInfo.AnglesZoD(1)-90 0]';
cdl.ReceiveArrayOrientation = [cdlInfo.AnglesAoA(1) cdlInfo.AnglesZoA(1)-90 0]';
```

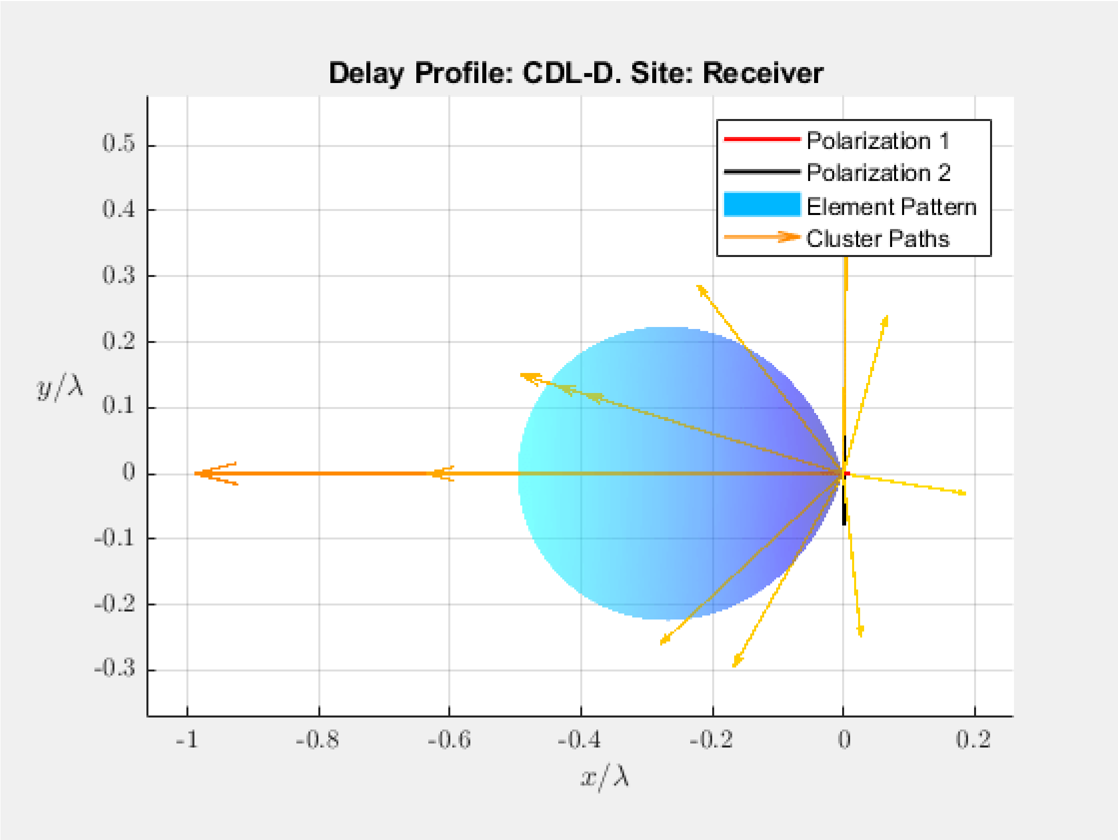
Visualize the channel characteristics at the transmitter end.

```
cdl.displayChannel('LinkEnd','Tx');
view(0,90)
```



Visualize the channel characteristics at the receiver end. The strongest path (LOS) passes through the maximum of the antenna element radiation pattern, which confirms that the antennas point at each other.

```
cdl.displayChannel('LinkEnd','Rx')
view(0,90)
```



Configure CDL Channel Antenna Using Phased Array

Create a CDL channel model. Then specify a phased array for the transmit antenna array.

```
cdl = nrCDLChannel;
cdl.TransmitAntennaArray = phased.URA;
```

Specify a cross-dipole transmit antenna array element to generate circularly polarized fields.

```
cdl.TransmitAntennaArray.Element = phased.CrossedDipoleAntennaElement;
```

Set the broadside direction of the array toward the positive *y*-axis. Add a 30 degree downtilt.

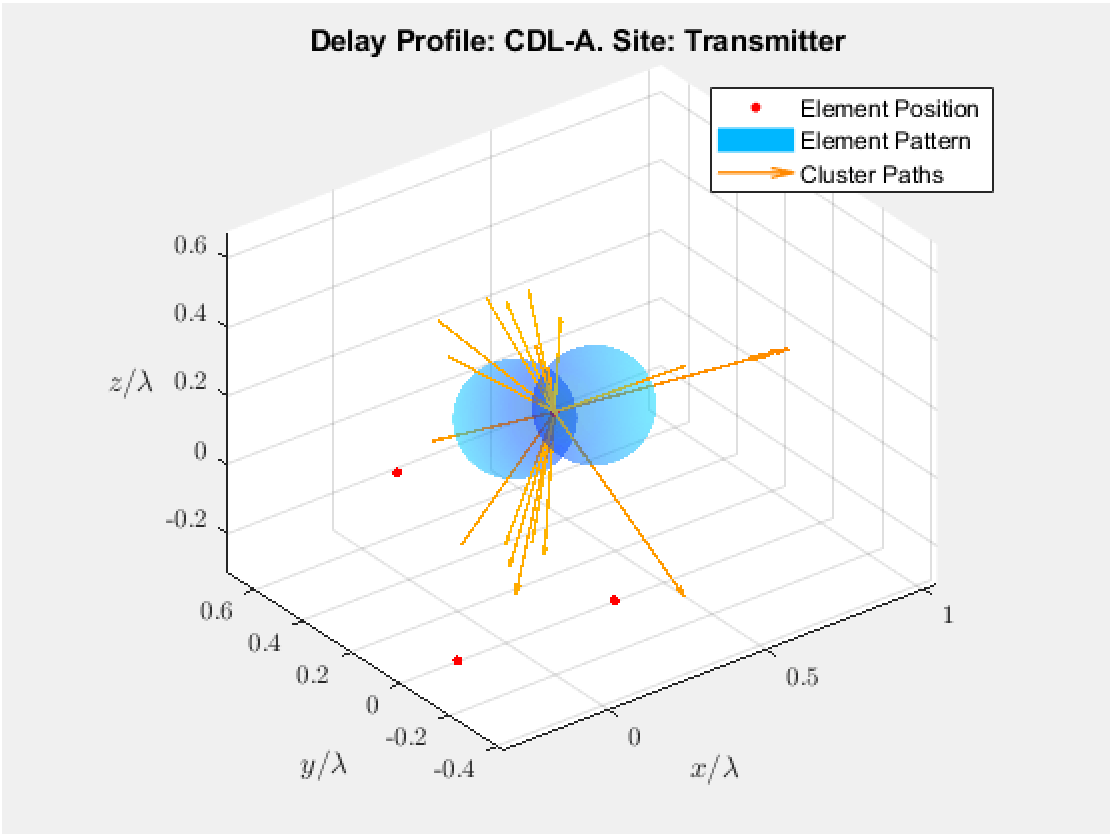
```
cdl.TransmitAntennaArray.ArrayNormal = 'y';
cdl.TransmitArrayOrientation = [0; 30; 0];
```

Set the antenna element spacing to half wavelength.

```
lambda = physconst('lightspeed')/cdl.CarrierFrequency;
cdl.TransmitAntennaArray.ElementSpacing = [lambda/2 lambda/2];
```

Visualize the channel characteristics at the transmitter end.

```
cdl.displayChannel('LinkEnd','Tx');
```



Version History

Introduced in R2018b

`Orientation` field of antenna array properties will be removed

Warns starting in R2021a

- The `Orientation` field of the `TransmitAntennaArray` property will be removed in a future release. Use the `TransmitArrayOrientation` property instead.
- The `Orientation` field of the `ReceiveAntennaArray` property will be removed in a future release. Use the `ReceiveArrayOrientation` property instead.

References

[1] 3GPP TR 38.901. "Study on channel model for frequencies from 0.5 to 100 GHz." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

Functions
`nrPerfectTimingEstimate` | `nrPerfectChannelEstimate`

Objects
`nrTDLChannel` | `comm.MIMOChannel`

Topics
“Visualize CDL Channel Model Characteristics”
“TDD Reciprocity-Based PDSCH MU-MIMO Using SRS”

nrDLSCH

Apply DL-SCH encoder processing chain

Description

The `nrDLSCH` System object applies the downlink shared channel (DL-SCH) encoder processing chain to one or two transport blocks. The DL-SCH encoding process consists of cyclic redundancy check (CRC), code block segmentation and CRC, low-density parity-check (LDPC) encoding, rate matching, and code block concatenation. The System object implements TS 38.212 Section 7.2 [1].

To apply the DL-SCH encoder processing chain:

1. Create the `nrDLSCH` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
encDL = nrDLSCH
encDL = nrDLSCH(Name,Value)
```

Description

`encDL = nrDLSCH` creates a DL-SCH encoder System object.

`encDL = nrDLSCH(Name,Value)` creates the object with properties set by using one or more name-value pairs. Enclose the property name inside quotes, followed by the specified value. Unspecified properties take default values.

Example: For example, `nrDLSCH('MultipleHARQProcesses',true)` creates the object and enables multiple hybrid automatic repeat-request (HARQ) processes.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`MultipleHARQProcesses` — Enable multiple HARQ processes
`false` (default) | `true`

Enable multiple HARQ processes, specified as `false` or `true`. When set to `false`, the object uses a single process. When set to `true`, the object uses multiple HARQ processes, at most 16. In both cases, to enable retransmissions when a failure occurs, the object buffers the input data.

Data Types: `logical`

`TargetCodeRate` — Target code rate
`0.5137` (default) | numeric scalar | 1-by-2 numeric vector

Target code rate, specified as a numeric scalar or a 1-by-2 numeric vector. The values must be in the interval (0, 1). The default value corresponds to 526/1024. If you specify `TargetCodeRate` as a scalar, the object applies scalar expansion when processing two transport blocks. To specify different target code rates for each transport block, specify `TargetCodeRate` as a vector.

Tunable: Yes

Data Types: `double`

`LimitedBufferSize` — Limited buffer size
`25344` (default) | positive integer

Limited buffer size used for rate matching, specified as a positive integer. The default value corresponds to 384×66, which is the maximum coded length of a code block. The default value implies no limit on the buffer size.

Data Types: `double`

Usage

Syntax

```
codedBits = encDL(mod,nLayers,outlen,rv)
codedBits = encDL( ___ ,harqID)
```

Description

`codedBits = encDL(mod,nLayers,outlen,rv)` applies the DL-SCH encoder processing chain to one or two transport blocks. The object returns encoded, rate-matched, and concatenated code blocks as one or two codewords of length `outlen`. Before you call this object, you must load the transport blocks into the object by using the `setTransportBlock` object function. `mod` specifies the modulation scheme. `nLayers` specifies the number of transmission layers. `rv` specifies the redundancy version of the transmission.

`codedBits = encDL( ___ ,harqID)` specifies the HARQ process number `harqID` used with the current transmission in addition to the input arguments in the previous syntax. To use this syntax, set the `MultipleHARQProcesses` property to `true`. When the property is set to `false`, the object uses HARQ process number 0.

When processing two transport blocks, specify the same HARQ process number for each transport block when calling the `setTransportBlock` function.

Input Arguments

`mod` — Modulation scheme
`'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'` | string array | cell array of character vectors

Modulation scheme, specified as 'QPSK', '16QAM', '64QAM', '256QAM', a string array, or cell array of character vectors. This modulation scheme determines the modulation type and number of bits used per modulation symbol. For two transport blocks, the modulation scheme applies to both blocks. Alternatively, you can specify different modulation schemes for each transport block by using a string array or a cell array of character vectors.

Modulation Scheme	Number of Bits Per Symbol
'QPSK'	2
'16QAM'	4
'64QAM'	6
'256QAM'	8

Data Types: char | string

nLayers — Number of transmission layers
integer from 1 to 8

Number of transmission layers, specified as an integer from 1 to 8. For nLayers > 4, the object expects two transports blocks.

Data Types: double

outlen — Output codeword length
nonnegative integer | 1-by-2 integer vector

Output codeword length, in bits, specified as a nonnegative integer or a 1-by-2 integer vector. If you specify outlen as a scalar, the object applies scalar expansion when processing two transport blocks. To specify a different codeword length for each transport block, specify outlen as a vector.

The actual output length is a multiple of the product of the number of bits per symbol and the number of transmission layers. For example, for 64-QAM and 1 transmission layer, if you specify 16 for outlen, the actual output length is 6x1x3 = 18.

Data Types: double

rv — Redundancy version
integer from 0 to 3 | 1-by-2 vector of integers from 0 to 3

Redundancy version, specified as one of these options.

- Integer from 0 to 3 — Use this option to specify the redundancy version for a single transport block.
- 1-by-2 vector of integers from 0 to 3 — Use this option to specify the redundancy version for two transport blocks.

Data Types: double

harqID — HARQ process number
integer from 0 to 15

HARQ process number, specified as an integer from 0 to 15.

Data Types: double

Output Arguments

codedBits — One or two DL-SCH codewords
binary column vector | cell array of two binary column vectors

One or two DL-SCH codewords, returned as a binary column vector or a cell array of two binary column vectors. A codeword is the encoded, rate-matched, and concatenated code blocks obtained by processing one transport block. Specify the length of the codewords by using the `outlen` input argument.

Data Types: `int8`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to `nrDLSCH`

getTransportBlock	Get transport block from UL-SCH or DL-SCH encoder
setTransportBlock	Load transport block into UL-SCH or DL-SCH encoder

Common to All System Objects

step	Run System object algorithm
clone	Create duplicate System object
isLocked	Determine if System object is in use
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Connect DL-SCH Encoder and Decoder Back to Back

Generate a random sequence of binary values corresponding to one transport block of length 5120.

```
trBlkLen = 5120;
trBlk = randi([0 1],trBlkLen,1,'int8');
```

Create and configure a DL-SCH encoder System object with the specified target code rate.

```
targetCodeRate = 567/1024;
encDL = nrDLSCH;
encDL.TargetCodeRate = targetCodeRate;
```

Load the transport block into the DL-SCH encoder.

```
setTransportBlock(encDL,trBlk);
```

Call the encoder with 64-QAM modulation scheme, 1 transmission layer, an output length of 10,240 bits, and redundancy version 0. The encoder applies the DL-SCH processing chain to the transport block loaded into the object.

```
mod = '64QAM';
nLayers = 1;
outlen = 10240;
rv = 0;
codedTrBlock = encDL(mod,nLayers,outlen,rv);
```

Create and configure a DL-SCH decoder System object.

```
decDL = nrDLSCHDecoder;
decDL.TargetCodeRate = targetCodeRate;
decDL.TransportBlockLength = trBlkLen;
```

Call the DL-SCH decoder on the soft bits representing the encoded transport block. Use the configuration parameters specified for the encoder. The error flag in the output indicates that the block decoding does not have errors.

```
rxSoftBits = 1.0 - 2.0*double(codedTrBlock);
[decbits,blkerr] = decDL(rxSoftBits,mod,nLayers,rv)
```

```
decbits = 5120x1 int8 column vector

   1
   1
   0
   1
   1
   0
   0
   1
   1
   1
      ⋮

blkerr = logical
   0
```

Verify that the transmitted and received message bits are identical.

```
isequal(decbits,trBlk)
```

```
ans = logical
   1
```

DL-SCH Encoding with Multiple HARQ Processes

Generate a random sequence of binary values corresponding to one transport block of length 5120.

```
trBlkLen = 5120;
trBlk = randi([0 1],trBlkLen,1,'int8');
```

Create and configure a DL-SCH encoder System object for use with multiple HARQ processes.

```
encDL = nrDLSCH;
encDL.MultipleHARQProcesses = true;
```

Load transport block `trBlk` for transport block number 0 into the DL-SCH encoder, specifying HARQ process number 2.

```
harqID = 2;
trBlkID = 0;
setTransportBlock(encDL,trBlk,trBlkID,harqID);
```

Call the encoder with QPSK modulation scheme, 3 transmission layers, an output length of 10,002 bits, and redundancy version 3. The encoder applies the DL-SCH processing chain to the transport block loaded into the object for HARQ process number 2.

```
mod = 'QPSK';
nLayers = 3;
outlen = 10002;
rv = 3;
codedTrBlock = encDL(mod,nLayers,outlen,rv,harqID);
```

Verify that the encoded transport block has the required number of bits.

```
isequal(length(codedTrBlock),outlen)
```

```
ans = logical
   1
```

Version History

Introduced in R2019a

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- The `nlayers` input argument must be compile-time constant. For example, include `{coder.Constant(nlayers)}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.
- See also “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

Objects

`nrDLSCHDecoder` | `nrULSCH`

Functions

`nrPDSCH` | `nrDLSCHInfo`

Topics

"NR PDSCH Throughput"

"5G NR Downlink Vector Waveform Generation"

nrDLSCHDecoder

Apply DL-SCH decoder processing chain

Description

The `nrDLSCHDecoder` System object applies the downlink shared channel (DL-SCH) decoder processing chain to the soft bits corresponding to one or two DL-SCH-encoded transport blocks. The DL-SCH decoding process consists of rate recovery, low-density parity-check (LDPC) decoding, desegmentation, and cyclic redundancy check (CRC) decoding. The object implements the inverse operation of the DL-SCH encoding process specified in TS 38.212 Section 7.2 [1].

To apply the DL-SCH decoder processing chain:

1. Create the `nrDLSCHDecoder` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
decDL = nrDLSCHDecoder
decDL = nrDLSCHDecoder(Name,Value)
```

Description

`decDL = nrDLSCHDecoder` creates a DL-SCH decoder System object.

`decDL = nrDLSCHDecoder(Name,Value)` creates the object with properties set by using one or more name-value pairs. Enclose the property name inside quotes, followed by the specified value. Unspecified properties take default values.

Example: For example, `nrDLSCHDecoder('MultipleHARQProcesses',true)` creates the object and enables multiple hybrid automatic repeat-request (HARQ) processes.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`MultipleHARQProcesses` — Enable multiple HARQ processes

`false` (default) | `true`

Enable multiple HARQ processes, specified as `false` or `true`. When set to `false`, the object uses a single process. When set to `true`, the object uses multiple HARQ processes, at most 16. To enable soft combining of retransmissions before LDPC decoding, the object maintains a soft buffer for each HARQ process.

Data Types: `logical`

`TargetCodeRate` — Target code rate
`0.5137` (default) | numeric scalar | 1-by-2 numeric vector

Target code rate, specified as a numeric scalar or a 1-by-2 numeric vector. The values must be in the interval (0, 1). The default value corresponds to 526/1024. If you specify `TargetCodeRate` as a scalar, the object applies scalar expansion when processing two transport blocks. To specify different target code rates for each transport block, specify `TargetCodeRate` as a vector.

Tunable: Yes

Data Types: `double`

`TransportBlockLength` — Length of decoded transport block
`5120` (default) | positive scalar integer | 1-by-2 integer vector

Length of decoded transport block, or transport blocks, in bits, specified as a positive scalar integer or a 1-by-2 integer vector. If you specify `TransportBlockLength` as a scalar, the object applies scalar expansion when processing two transport blocks. To specify a different length for the decoded transport blocks, specify `TransportBlockLength` as a vector.

Tunable: Yes

Data Types: `double`

`LimitedBufferSize` — Limited buffer size
`25344` (default) | positive integer

Limited buffer size used for rate recovery, specified as a positive integer. The default value corresponds to 384×66, which is the maximum coded length of a code block. The default value implies no limit on the buffer size.

Data Types: `double`

`MaximumLDPCIterationCount` — Maximum LDPC decoding iterations
`12` (default) | positive integer

Maximum LDPC decoding iterations, specified as a positive integer. Since early termination is enabled, decoding stops once parity-checks are satisfied. In this case, fewer iterations take place than the maximum specified by this argument.

Data Types: `double`

`LDPCDecodingAlgorithm` — LDPC decoding algorithm
`'Belief propagation'` (default) | `'Layered belief propagation'` | `'Normalized min-sum'` | `'Offset min-sum'`

LDPC decoding algorithm, specified as one of these values:

- `'Belief propagation'` — Use this option to specify the belief-passing or message-passing algorithm.

- 'Layered belief propagation' — Use this option to specify the layered belief-passing algorithm, which is suitable for quasi-cyclic parity-check matrices (PCMs).
- 'Normalized min-sum' — Use this option to specify the layered belief propagation algorithm with normalized min-sum approximation.
- 'Offset min-sum' — Use this option to specify the layered belief propagation algorithm with offset min-sum approximation.

For more information on these algorithms, see LDPC Decoding Algorithms on page 2-41.

Data Types: char | string

ScalingFactor — Scaling factor for normalized min-sum decoding
0.75 (default) | real scalar in the range (0, 1]

Scaling factor for normalized min-sum decoding, specified as a real scalar in the range (0, 1].

Dependencies

To enable this property, set the LDPCDecodingAlgorithm property to 'Normalized min-sum'.

Data Types: double

Offset — Offset for offset min-sum decoding
0.5 (default) | nonnegative finite real scalar

Offset for offset min-sum decoding, specified as a nonnegative finite real scalar.

Dependencies

To enable this property, set the LDPCDecodingAlgorithm property to 'Offset min-sum'.

Data Types: double

Usage

Syntax

```
trblk = decDL(softbits,mod,nLayers,rv)
trblk = decDL( ___ ,harqID)
[trblk,blkerr] = decDL( ___ )
```

Description

trblk = decDL(softbits,mod,nLayers,rv) applies the DL-SCH decoder processing chain to the input softbits and returns the decoded bits. mod specifies the modulation scheme. nLayers specifies the number of transmission layers. rv specifies the redundancy version of the transmission.

trblk = decDL(___ ,harqID) specifies the HARQ process number harqID used with the current transmission in addition to the input arguments in the previous syntax. To use this syntax, set the MultipleHARQProcesses property to true. When the property is set to false, the object uses HARQ process number 0.

When the object receives codewords with different redundancy version for an individual HARQ process, the object uses soft buffer state retention to enable soft combining of retransmissions. When you enable multiple HARQ processes, the object maintains independent buffers for each process.

`[trblk,blkerr] = decDL( ___ )` returns an error flag, using the input arguments in any of the previous syntaxes. A value of 1 in `blkerr` indicates an error during transport block decoding.

Input Arguments

`softbits` — Approximate LLR soft bits
real column vector | cell array of two real column vectors

Approximate log-likelihood ratio (LLR) soft bits, corresponding to one or two DL-SCH-encoded transport blocks, specified as a real column vector or a cell array of two real column vectors.

Data Types: `single` | `double`

`mod` — Modulation scheme
`'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'` | string array | cell array of character vectors

Modulation scheme, specified as `'QPSK'`, `'16QAM'`, `'64QAM'`, `'256QAM'`, a string array, or cell array of character vectors. This modulation scheme determines the modulation type and number of bits used per modulation symbol. For two transport blocks, the modulation scheme applies to both blocks. Alternatively, you can specify different modulation schemes for each transport block by using a string array or a cell array of character vectors.

Modulation Scheme	Number of Bits Per Symbol
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

Data Types: `char` | `string`

`nLayers` — Number of transmission layers
integer from 1 to 8

Number of transmission layers, specified as an integer from 1 to 8. For `nLayers` > 4, the object expects two encoded transports blocks as input.

Data Types: `double`

`rv` — Redundancy version
integer from 0 to 3 | 1-by-2 vector of integers from 0 to 3

Redundancy version, specified as one of these options.

- Integer from 0 to 3 — Use this option to specify the redundancy version for a single encoded transport block.
- 1-by-2 vector of integers from 0 to 3 — Use this option to specify the redundancy version for two encoded transport blocks.

Data Types: `double`

`harqID` — HARQ process number
integer from 0 to 15

HARQ process number, specified as an integer from 0 to 15.

Data Types: double

Output Arguments

trblk — Decoded DL-SCH transport blocks
binary column vector | cell array of two binary column vectors

Decoded DL-SCH transport blocks, returned as a binary column vector or cell array of two binary column vectors. The TransportBlockLength property specifies the length of the column vectors.

blkerr — Result of DL-SCH transport block decoding
logical scalar | logical vector

Result of DL-SCH transport block decoding for each transport block, returned as a logical scalar or logical vector of length 2. A value of 1 in blkerr indicates an error during transport block decoding.

Data Types: logical

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named obj, use this syntax:

```
release(obj)
```

Specific to nrDLSCHDecoder

resetSoftBuffer	Reset soft buffer for HARQ process in UL-SCH or DL-SCH decoder

Common to All System Objects

step	Run System object algorithm
clone	Create duplicate System object
isLocked	Determine if System object is in use
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Connect DL-SCH Encoder and Decoder Back to Back

Generate a random sequence of binary values corresponding to one transport block of length 5120.

```
trBlkLen = 5120;
trBlk = randi([0 1],trBlkLen,1,'int8');
```

Create and configure a DL-SCH encoder System object with the specified target code rate.

```
targetCodeRate = 567/1024;
encDL = nrDLSCH;
encDL.TargetCodeRate = targetCodeRate;
```

Load the transport block into the DL-SCH encoder.

```
setTransportBlock(encDL,trBlk);
```

Call the encoder with 64-QAM modulation scheme, 1 transmission layer, an output length of 10,240 bits, and redundancy version 0. The encoder applies the DL-SCH processing chain to the transport block loaded into the object.

```
mod = '64QAM';
nLayers = 1;
outlen = 10240;
rv = 0;
codedTrBlock = encDL(mod,nLayers,outlen,rv);
```

Create and configure a DL-SCH decoder System object.

```
decDL = nrDLSCHDecoder;
decDL.TargetCodeRate = targetCodeRate;
decDL.TransportBlockLength = trBlkLen;
```

Call the DL-SCH decoder on the soft bits representing the encoded transport block. Use the configuration parameters specified for the encoder. The error flag in the output indicates that the block decoding does not have errors.

```
rxSoftBits = 1.0 - 2.0*double(codedTrBlock);
[decbits,blkerr] = decDL(rxSoftBits,mod,nLayers,rv)
```

```
decbits = 5120x1 int8 column vector

   1
   1
   0
   1
   1
   0
   0
   1
   1
   1
      ⋮

blkerr = logical
   0
```

Verify that the transmitted and received message bits are identical.

```
isequal(decbits,trBlk)
```

```
ans = logical
   1
```

Algorithms

LDPC Decoding Algorithms

The `nrDLSCHDecoder` object supports these four LDPC decoding algorithms.

Belief Propagation Decoding

The implementation of the belief propagation algorithm is based on the decoding algorithm presented in [2]. For transmitted LDPC-encoded codeword, c, where $c = (c_0, c_1, \ldots, c_{n-1})$, the input to the LDPC decoder is the log-likelihood ratio (LLR) value $L(c_i) = \log\left(\frac{\Pr(c_i = 0 \mid \text{channel output for } c_i)}{\Pr(c_i = 1 \mid \text{channel output for } c_i)}\right)$.

In each iteration, the key components of the algorithm are updated based on these equations:

$$L(r_{ji}) = 2\,\text{atanh}\left(\prod_{i' \in V_j \backslash i} \tanh\left(\frac{1}{2}L(q_{i'j})\right)\right),$$

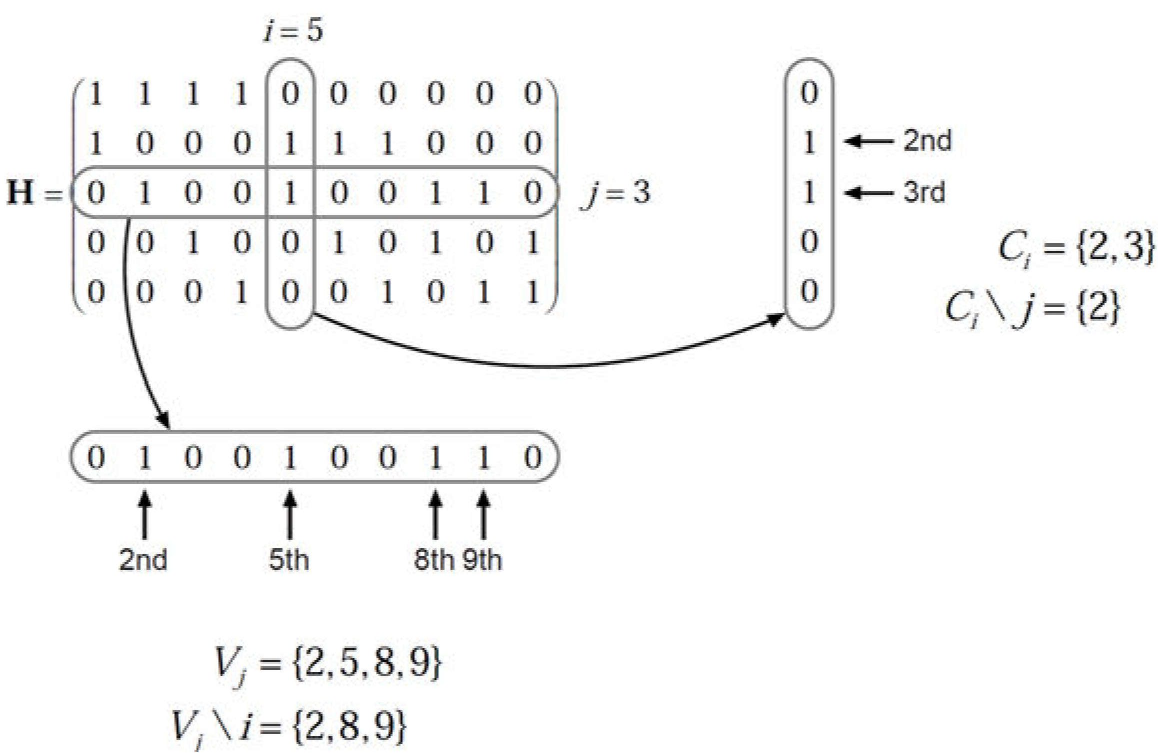
$L(q_{ij}) = L(c_i) + \sum_{j' \in C_i \backslash j} L(r_{j'i})$, initialized as $L(q_{ij}) = L(c_i)$ before the first iteration, and

$$L(Q_i) = L(c_i) + \sum_{j' \in C_i} L(r_{j'i}).$$

At the end of each iteration, $L(Q_i)$ is an updated estimate of the LLR value for the transmitted bit c_i. The value $L(Q_i)$ is the soft-decision output for c_i. If $L(Q_i) < 0$, the hard-decision output for c_i is 1. Otherwise, the output is 0.

Index sets $C_i \backslash j$ and $V_j \backslash i$ are based on the parity-check matrix (PCM). Index sets C_i and V_j correspond to all nonzero elements in column i and row j of the PCM, respectively.

This figure highlights the computation of these index sets in a given PCM for $i = 5$ and $j = 3$.



To avoid infinite numbers in the algorithm equations, atanh(1) and atanh(-1) are set to 19.07 and –19.07, respectively. Due to finite precision, MATLAB returns 1 for tanh(19.07) and –1 for tanh(–19.07).

The decoding terminates when all parity checks are satisfied ($\mathbf{Hc}^{\mathrm{T}} = 0$) or after `MaximumLDPCIterationCount` number of iterations.

Layered Belief Propagation Decoding

The implementation of the layered belief propagation algorithm is based on the decoding algorithm presented in [3], Section II.A. The decoding loop iterates over subsets of rows (layers) of the PCM. For each row, m, in a layer and each bit index, j, the implementation updates the key components of the algorithm based on these equations:

(1) $L(q_{mj}) = L(q_j) - R_{mj}$,

(2) $A_{mj} = \sum_{\substack{n \in N(m) \\ n \neq j}} \psi(L(q_{mn}))$,

(3) $s_{mj} = \prod_{\substack{n \in N(m) \\ n \neq j}} \mathrm{sign}(L(q_{mn}))$,

(4) $R_{mj} = -s_{mj}\psi(A_{mj})$, and

(5) $L(q_j) = L(q_{mj}) + R_{mj}$.

For each layer, the decoding equation (5) works on the combined input obtained from the current LLR inputs $L(q_{mj})$ and the previous layer updates R_{mj}.

Because only a subset of the nodes is updated in a layer, the layered belief propagation algorithm is faster compared to the belief propagation algorithm. To achieve the same error rate as attained with belief propagation decoding, use half the number of decoding iterations when using the layered belief propagation algorithm.

Normalized Min-Sum Decoding

The implementation of the normalized min-sum decoding algorithm follows the layered belief propagation algorithm with equation (2) replaced by

$$A_{mj} = \min_{\substack{n \in N(m) \\ n \neq j}} (|L(q_{mn})| \cdot \alpha),$$

where α is in the range (0, 1] and is the scaling factor specified by `ScalingFactor`. This equation is an adaptation of equation (4) presented in [4].

Offset Min-Sum Decoding

The implementation of the offset min-sum decoding algorithm follows the layered belief propagation algorithm with equation (2) replaced by

$$A_{mj} = \max\left(\min_{\substack{n \in N(m) \\ n \neq j}} (|L(q_{mn})| - \beta), 0\right),$$

where $\beta \geq 0$ and is the offset specified by `Offset`. This equation is an adaptation of equation (5) presented in [4].

Version History

Introduced in R2019a

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] Gallager, Robert G. *Low-Density Parity-Check Codes*, Cambridge, MA, MIT Press, 1963.

[3] Hocevar, D.E. "A reduced complexity decoder architecture via layered decoding of LDPC codes." In *IEEE Workshop on Signal Processing Systems, 2004. SIPS 2004*. dois: 10.1109/SIPS.2004.1363033

[4] Chen, Jinghu, R.M. Tanner, C. Jones, and Yan Li. "Improved min-sum decoding algorithms for irregular LDPC codes." In *Proceedings. International Symposium on Information Theory, 2005. ISIT 2005*. doi: 10.1109/ISIT.2005.1523374

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- The `nlayers` input argument must be compile-time constant. For example, include `{coder.Constant(nlayers)}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.
- See also “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

Objects

`nrDLSCH` | `nrULSCHDecoder`

Functions

`nrPDSCHDecode` | `nrDLSCHInfo`

Topics

“NR PDSCH Throughput”

nrHSTChannel

Send signal through HST channel model

Description

The `nrHSTChannel` System object sends an input signal through a high-speed train (HST) multi-input multi-output (MIMO) link-level propagation channel to obtain the channel-impaired signal. The object implements these aspects of TS 38.101-4 [1]:

- Annex B.1: Static propagation condition
- Annex B.3.1: Single-tap channel profile (same as TS 38.104 Annex G.3: High-speed train condition [2])
- Annex B.3.2: HST-SFN channel profile
- Annex B.3.3: HST-DPS channel profile

To send a signal through the HST MIMO channel model:

1. Create the `nrHSTChannel` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
hst = nrHSTChannel
hst = nrHSTChannel(Name=Value)
```

Description

`hst = nrHSTChannel` creates an HST MIMO channel System object.

`hst = nrHSTChannel(Name=Value)` sets properties on page 2-45 using one or more optional name-value arguments. For example, `Velocity=350` sets the train velocity to 350 km/h.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

ChannelProfile — HST channel profile
'HST' (default) | 'HST-SFN' | 'HST-DPS'

HST channel profile, specified as one of these values.

- 'HST' — Use this value to specify the HST single-tap channel profile, as defined in TS 38.101-4 Annex B.3.1.
- 'HST-SFN' — Use this value to specify the HST single frequency network (SFN) multi-tap fading channel profile, as defined in TS 38.101-4 Annex B.3.2.
- 'HST-DPS' — Use this value to specify the HST dynamic point switching (DPS) single-tap channel profile, as defined in TS 38.101-4 Annex B.3.3.

Data Types: char | string

Ds — Distance between gNodeBs in meters
300 (default) | positive numeric scalar

Distance between gNodeBs in meters, specified as a positive numeric scalar. When you set the ChannelProfile property to 'HST', Ds/2 is the initial distance of the train from a gNodeB.

Data Types: double

Dmin — Minimum distance between gNodeB and railway track in meters
2 (default) | positive numeric scalar

Minimum distance between a gNodeB and the railway track in meters, specified as a positive numeric scalar.

Data Types: double

Velocity — Train velocity in km/h
300 (default) | nonnegative numeric scalar

Train velocity in km/h, specified as a nonnegative numeric scalar. For static propagation conditions, as defined in TS 38.101-4 Annex B.1, set this property and the MaximumDopplerShift property to 0.

Data Types: double

MaximumDopplerShift — Maximum Doppler shift in Hz
750 (default) | nonnegative numeric scalar

Maximum Doppler shift in Hz, specified as a nonnegative numeric scalar. This property applies to all channel paths. For static propagation conditions, as defined in TS 38.101-4 Annex B.1, set this property and the Velocity property to 0.

Data Types: double

NumTaps — Number of taps in HST-SFN channel profile
4 (default) | positive integer

Number of taps in the HST-SFN channel profile, specified as a positive integer.

Dependencies

To enable this property, set the ChannelProfile property to 'HST-SFN'.

Data Types: double

`SampleRate` — Sample rate of input signal in Hz
`30720000` (default) | positive numeric scalar

Sample rate of the input signal in Hz, specified as a positive numeric scalar.

Data Types: `double`

`NumTransmitAntennas` — Number of transmit antennas
`1` (default) | positive integer

Number of transmit antennas, specified as a positive integer. When you set `ChannelFiltering` to `true`, the object ignores this property and uses the number of columns of the input signal to set the number of transmit antennas.

Dependencies

To enable this property, set `ChannelFiltering` to `false`.

Data Types: `double`

`NumReceiveAntennas` — Number of receive antennas
2 (default) | positive integer

Number of receive antennas, specified as a positive integer.

Data Types: `double`

`InitialTime` — Time offset of channel process in seconds
`0.0` (default) | nonnegative scalar

Time offset of the channel process in seconds, specified as a nonnegative scalar.

Tunable: Yes

Data Types: `double`

`NormalizeChannelOutputs` — Normalize channel outputs
`true` (default) | `false`

Normalize channel outputs, specified as `true` or `false`. When this property is set to `true`, the channel outputs are normalized by the number of receive antenna elements.

Data Types: `logical`

`ChannelFiltering` — Channel filtering
`true` (default) | `false`

Channel filtering, specified as `true` or `false`. When this property is set to `false`, these conditions apply.

- The object takes no input signal and returns only the path gains and sample times.
- The `NumTimeSamples` property controls the duration of the channel process at a sampling rate specified by the `SampleRate` property.
- The channel coefficients sampling rate is one sample per each time sample from 0 to `NumTimeSamples` – 1.

Data Types: `logical`

NumTimeSamples — Number of time samples
30720 (default) | positive integer

Number of time samples, specified as a positive integer. Use this property to set the duration of the channel process.

Tunable: Yes

Dependencies

To enable this property, set ChannelFiltering to false.

Data Types: double

OutputDataType — Data type of generated path gains
'double' (default) | 'single'

Data type of the generated path gains, specified as 'double' or 'single'. When you set ChannelFiltering to true, the object ignores this property and uses the data type of the input signal to set the data type of the path gains.

Dependencies

To enable this property, set ChannelFiltering to false.

Data Types: double

Usage

Syntax

```
signalOut = hst(signalIn)
[signalOut,pathGains] = hst(signalIn)
[signalOut,pathGains,sampleTimes] = hst(signalIn)

[pathGains,sampleTimes] = hst()
```

Description

signalOut = hst(signalIn) sends the input signal through an HST MIMO channel and returns the channel-impaired signal.

[signalOut,pathGains] = hst(signalIn) also returns the channel path gains.

[signalOut,pathGains,sampleTimes] = hst(signalIn) also returns the sample times of the channel snapshots of the path gains, pathGains.

[pathGains,sampleTimes] = hst() returns only the path gains and the sample times. The hst object acts as a source of the path gains and sample times without filtering an input signal. The NumTimeSamples property specifies the duration of the channel process and the OutputDataType property specifies the data type of the generated path gains. To use this syntax, you must set the ChannelFiltering property to false.

Input Arguments

`signalIn` — Input signal
complex scalar | vector | N_S-by-N_T matrix

Input signal, specified as a complex scalar, vector, or N_S-by-N_T matrix, where:

- N_S is the number of samples.
- N_T is the number of transmit antennas that must be constant across all object calls.

Data Types: `single` | `double`
Complex Number Support: Yes

Output Arguments

`signalOut` — Output signal
complex scalar | vector | N_S-by-N_R matrix

Output signal, returned as a complex scalar, vector, or N_S-by-N_R matrix, where:

- N_S is the number of samples.
- N_R is the number of receive antennas.

The output signal is of the same precision as the input signal, `signalIn`.

Data Types: `single` | `double`
Complex Number Support: Yes

`pathGains` — Channel path gains
N_S-by-N_P-by-N_T-by-N_R complex matrix

Channel path gains, returned as an N_S-by-N_P-by-N_T-by-N_R complex matrix, where:

- N_S is the number of samples.
- N_P is the number of paths.
- N_T is the number of transmit antennas.
- N_R is the number of receive antennas.

The path gains are of the same precision as the input signal, `signalIn`.

Data Types: `single` | `double`
Complex Number Support: Yes

`sampleTimes` — Sample times of channel snapshots
N_S-by-1 column vector of real numbers

Sample times of the channel snapshots of the path gains, `pathGains`, returned as an N_S-by-1 column vector of real numbers. N_S is the first dimension of `pathGains` that corresponds to the number of samples.

Data Types: `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to `nrHSTChannel`

info	Characteristic information of link-level MIMO channel
getPathFilters	Get path filter impulse response for link-level MIMO channel

Common to All System Objects

step	Run System object algorithm
clone	Create duplicate System object
isLocked	Determine if System object is in use
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Send Waveform Through HST Single-Tap Channel

Create an HST single-tap channel model.

```
hst = nrHSTChannel;
hst.ChannelProfile = 'HST';
```

Set the distance between the gNodeBs to 300 m. Set the minimum distance between the railway track and the gNodeBs to 2 m.

```
hst.Ds = 300;
hst.Dmin = 2;
```

Set the train velocity to 300 km/h. Set the maximum Doppler shift to 750 Hz.

```
hst.Velocity = 300;
hst.MaximumDopplerShift = 750;
```

Create a random waveform of one subframe duration with one transmit antenna.

```
hst.SampleRate = 30.72e6;
T = hst.SampleRate * 1e-3;
Nt = 1;
txWaveform = complex(randn(T,Nt),randn(T,Nt));
```

Send the waveform through the channel.

```
rxWaveform = hst(txWaveform);
```

Estimate HST-SFN Channel Delay

Create an HST-SFN multi-tap channel model with one receive antenna.

```
hst = nrHSTChannel( ...
    ChannelProfile='HST-SFN', ...
    NumReceiveAntennas=1);
```

Set the distance between the gNodeBs to 700 m. Set the minimum distance between the railway track and the gNodeBs to 150 m.

```
hst.Ds = 700;
hst.Dmin = 150;
```

Set the train velocity to 500 km/h. Set the maximum Doppler shift to 870 Hz.

```
hst.Velocity = 500;
hst.MaximumDopplerShift = 870;
```

Disable channel filtering.

```
hst.ChannelFiltering = false;
```

Set the sample rate and the number of channel samples to calculate 1 ms of path gains samples.

```
hst.SampleRate = 30.72e6;
hst.NumTimeSamples = hst.SampleRate*1e-3;
```

Set the initial time of the channel to configure the starting position of the train. The train position relative to a remote gNodeB determines the delay of each gNodeB signal.

```
hst.InitialTime = (hst.Ds/3)/(hst.Velocity/3.6);
```

Retrieve the path gains from the channel.

```
pathGains = hst();
```

Obtain the channel path filter responses relative to the previous channel call. The delay of each gNodeB signal changes over time.

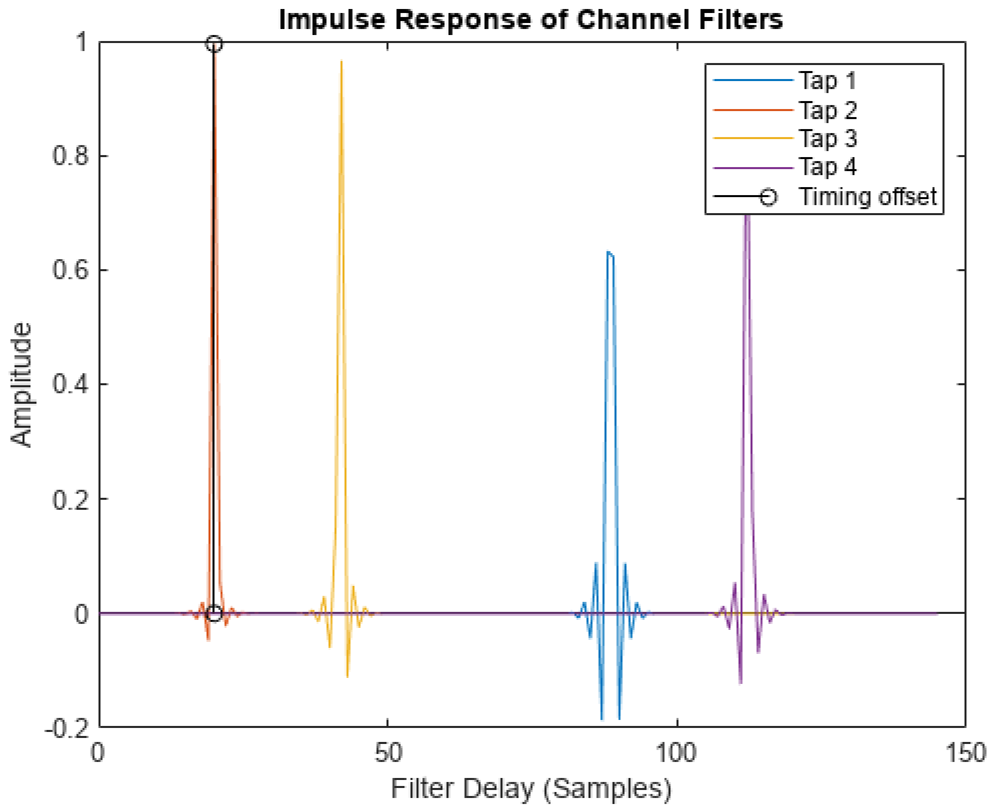
```
pathFilters = getPathFilters(hst);
```

Estimate the channel delay by obtaining the timing offset.

```
offset = nrPerfectTimingEstimate(pathGains,pathFilters);
```

Display the path filters and the estimated channel delay.

```
plot(0:size(pathFilters,1)-1,pathFilters);
hold on
stem(repmat(offset,1,hst.NumTaps),pathFilters(1+offset,:),'k')
legend(["Tap "+ (1:hst.NumTaps) "Timing offset"])
xlabel('Filter Delay (Samples)')
ylabel('Amplitude')
title('Impulse Response of Channel Filters')
```



Version History

Introduced in R2022b

References

[1] 3GPP TS 38.101-4. "NR; User Equipment (UE) radio transmission and reception; Part 4: Performance requirements." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.104. "NR; Base Station (BS) radio transmission and reception." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also

Functions
`nrPerfectTimingEstimate` | `nrPerfectChannelEstimate`

Objects
`nrCDLChannel` | `nrTDLChannel`

nrULSCH

Apply UL-SCH encoder processing chain

Description

The `nrULSCH` System object applies the uplink shared channel (UL-SCH) encoder processing chain to a transport block. The UL-SCH encoding process consists of cyclic redundancy check (CRC), code block segmentation and CRC, low-density parity-check (LDPC) encoding, rate matching, and code block concatenation. The object implements these aspects of TS 38.212 [1]:

- Sections 6.2.1: Transport block CRC attachment
- Sections 6.2.2: LDPC base graph selection
- Sections 6.2.3: Code block segmentation and code block CRC attachment
- Sections 6.2.4: Channel coding of UL-SCH
- Sections 6.2.5: Rate matching
- Sections 6.2.6: Code block concatenation

To apply the UL-SCH encoder processing chain:

1. Create the `nrULSCH` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
encUL = nrULSCH
encUL = nrULSCH(Name,Value)
```

Description

`encUL = nrULSCH` creates a UL-SCH encoder System object.

`encUL = nrULSCH(Name,Value)` creates the object with properties set by using one or more name-value pairs. Enclose the property name inside quotes, followed by the specified value. Unspecified properties take default values.

Example: For example, `nrULSCH('MultipleHARQProcesses',true)` creates the object and enables multiple hybrid automatic repeat-request (HARQ) processes.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`MultipleHARQProcesses` — Enable multiple HARQ processes
`false` (default) | `true`

Enable multiple HARQ processes, specified as `false` or `true`. When set to `false`, the object uses a single process. When set to `true`, the object uses multiple HARQ processes, at most 16. In both cases, to enable retransmissions when a failure occurs, the object buffers the input data.

Data Types: `logical`

`TargetCodeRate` — Target code rate
`0.5137` (default) | real number

Target code rate, specified as a real number in the interval (0, 1). The default value corresponds to 526/1024.

Tunable: Yes

Data Types: `double`

`LimitedBufferRateMatching` — Enable limited buffer rate matching
`false` (default) | `true`

Enable limited buffer rate matching, specified as `false` or `true`. When set to `false`, the size of the internal buffer used for rate matching is the full coded length of each code block. When set to `true`, you can specify the size of the internal buffer used for rate matching by setting the `LimitedBufferSize` property.

Data Types: `logical`

`LimitedBufferSize` — Limited buffer size
`25344` (default) | positive integer

Limited buffer size used for rate matching, specified as a positive integer. The default value corresponds to 384×66, which is the maximum coded length of a code block. The default value implies no limit on the buffer size.

Dependencies

To enable this property, set `LimitedBufferRateMatching` to `true`.

Data Types: `double`

Usage

Syntax

```
codedBits = encUL(mod,nLayers,outlen,rv)
codedBits = encUL(___,harqID)
```

Description

`codedBits = encUL(mod,nLayers,outlen,rv)` applies the UL-SCH encoder processing chain to the transport block previously loaded into the object. The object returns the encoded, rate-matched, and concatenated code blocks as a codeword of length `outlen`. Before you call this object, you must load a transport block into the object by using the `setTransportBlock` object function. `mod` specifies the modulation scheme. `nLayers` specifies the number of transmission layers. `rv` specifies the redundancy version of the transmission.

`codedBits = encUL( ___ ,harqID)` specifies the HARQ process number `harqID` used with the current transport block in addition to the input arguments in the previous syntax. To use this syntax, set the `MultipleHARQProcesses` property to `true`. When the property is set to `false`, the object uses HARQ process number 0.

Input Arguments

`mod` — Modulation scheme
`'pi/2-BPSK'` | `'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'`

Modulation scheme, specified as `'pi/2-BPSK'`, `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`. This modulation scheme determines the modulation type and number of bits used per modulation symbol.

Modulation Scheme	Number of Bits Per Symbol
`'pi/2-BPSK'`	1
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

Data Types: `char` | `string`

`nLayers` — Number of transmission layers
integer from 1 to 4

Number of transmission layers, specified as an integer from 1 to 4. For more information, see TS 38.211 Section 6.3.1.3.

Data Types: `double`

`outlen` — Output codeword length
nonnegative integer

Output codeword length, in bits, specified as a nonnegative integer. The actual output length is a multiple of the product of the number of bits per symbol and the number of transmission layers. For example, for 64-QAM and 1 transmission layer, if you specify 16 for `outlen`, the actual output length is 6x1x3 = 18.

Data Types: `double`

`rv` — Redundancy version
integer from 0 to 3

Redundancy version, specified as an integer from 0 to 3.

Data Types: double

harqID — HARQ process number
integer from 0 to 15

HARQ process number, specified as an integer from 0 to 15.

Data Types: double

Output Arguments

codedBits — UL-SCH codeword
binary column vector

UL-SCH codeword, returned as a binary column vector of length `outlen`. A codeword is the encoded, rate-matched, and concatenated code blocks obtained by processing the transport block.

Data Types: int8

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to nrULSCH

getTransportBlock	Get transport block from UL-SCH or DL-SCH encoder
setTransportBlock	Load transport block into UL-SCH or DL-SCH encoder

Common to All System Objects

step	Run System object algorithm
clone	Create duplicate System object
isLocked	Determine if System object is in use
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Connect UL-SCH Encoder and Decoder Back to Back

Generate a random sequence of binary values corresponding to one transport block of length 5120.

```
trBlkLen = 5120;
trBlk = randi([0 1],trBlkLen,1,'int8');
```

Create and configure an UL-SCH encoder System object with the specified target code rate.

```
targetCodeRate = 567/1024;
encUL = nrULSCH;
encUL.TargetCodeRate = targetCodeRate;
```

Load the transport block into the UL-SCH encoder.

```
setTransportBlock(encUL,trBlk);
```

Call the encoder with 64-QAM modulation scheme, 1 transmission layer, an output length of 10,240 bits, and redundancy version 0. The encoder applies the UL-SCH processing chain to the transport block loaded into the object.

```
mod = '64QAM';
nLayers = 1;
outlen = 10240;
rv = 0;
codedTrBlock = encUL(mod,nLayers,outlen,rv);
```

Create and configure an UL-SCH decoder System object.

```
decUL = nrULSCHDecoder;
decUL.TargetCodeRate = targetCodeRate;
decUL.TransportBlockLength = trBlkLen;
```

Call the UL-SCH decoder on the soft bits representing the encoded transport block. Use the configuration parameters specified for the encoder. The error flag in the output indicates that the block decoding does not have errors.

```
rxSoftBits = 1.0 - 2.0*double(codedTrBlock);
[decbits,blkerr] = decUL(rxSoftBits,mod,nLayers,rv)
```

```
decbits = 5120x1 int8 column vector

   1
   1
   0
   1
   1
   0
   0
   1
   1
   1
      ⋮

blkerr = logical
   0
```

Verify that the transmitted and received message bits are identical.

```
isequal(decbits,trBlk)
```

```
ans = logical
   1
```

UL-SCH Encoding with Multiple HARQ Processes

Generate a random sequence of binary values corresponding to one transport block of length 5120.

```
trBlkLen = 5120;
trBlk = randi([0 1],trBlkLen,1,'int8');
```

Create and configure an UL-SCH encoder System object for use with multiple HARQ processes.

```
encUL = nrULSCH;
encUL.MultipleHARQProcesses = true;
```

Load the transport block into the UL-SCH encoder, specifying HARQ process number 2.

```
harqID = 2;
setTransportBlock(encUL,trBlk,harqID);
```

Call the encoder with QPSK modulation scheme, 3 transmission layers, an output length of 10,002 bits, and redundancy version 3. The encoder applies the UL-SCH processing chain to the transport block loaded into the object for HARQ process number 2.

```
mod = 'QPSK';
nLayers = 3;
outlen = 10002;
rv = 3;
codedTrBlock = encUL(mod,nLayers,outlen,rv,harqID);
```

Verify that the encoded transport block has the required number of bits.

```
isequal(length(codedTrBlock),outlen)
```

```
ans = logical
   1
```

Version History

Introduced in R2019a

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

Objects
nrULSCHDecoder | nrDLSCH

Functions
nrPUSCH | nrULSCHInfo

nrULSCHDecoder

Apply UL-SCH decoder processing chain

Description

The `nrULSCHDecoder` System object applies the uplink shared channel (UL-SCH) decoder processing chain to the soft bits corresponding to a UL-SCH-encoded transport block. The UL-SCH decoding process consists of rate recovery, low-density parity-check (LDPC) decoding, desegmentation, and cyclic redundancy check (CRC) decoding. The object implements the inverse operation of the UL-SCH encoding process specified in these sections of TS 38.212 [1]:

- Sections 6.2.1: Transport block CRC attachment
- Sections 6.2.2: LDPC base graph selection
- Sections 6.2.3: Code block segmentation and code block CRC attachment
- Sections 6.2.4: Channel coding of UL-SCH
- Sections 6.2.5: Rate matching
- Sections 6.2.6: Code block concatenation

To apply the UL-SCH decoder processing chain:

1. Create the `nrULSCHDecoder` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
decUL = nrULSCHDecoder
decUL = nrULSCHDecoder(Name,Value)
```

Description

`decUL = nrULSCHDecoder` creates a UL-SCH decoder System object.

`decUL = nrULSCHDecoder(Name,Value)` creates the object with properties set by using one or more name-value pairs. Enclose the property name inside quotes, followed by the specified value. Unspecified properties take default values.

Example: For example, `nrULSCHDecoder('MultipleHARQProcesses',true)` creates the object and enables multiple hybrid automatic repeat-request (HARQ) processes.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`MultipleHARQProcesses` — Enable multiple HARQ processes
`false` (default) | `true`

Enable multiple HARQ processes, specified as `false` or `true`. When set to `false`, the object uses a single process. When set to `true`, the object uses multiple HARQ processes, at most 16. To enable soft combining of retransmissions before LDPC decoding, the object maintains a soft buffer for each HARQ process.

Data Types: `logical`

`TargetCodeRate` — Target code rate
`0.5137` (default) | real number

Target code rate, specified as a real number in the interval (0, 1). The default value corresponds to 526/1024.

Tunable: Yes

Data Types: `double`

`TransportBlockLength` — Length of decoded transport block
`5120` (default) | positive integer

Length of decoded transport block, in bits, specified as a positive integer.

Tunable: Yes

Data Types: `double`

`LimitedBufferRateRecovery` — Enable limited buffer rate recovery
`false` (default) | `true`

Enable limited buffer rate recovery, specified as `false` or `true`. When set to `false`, the size of the internal buffer used for rate recovery is the full coded length of each code block. When set to `true`, you can specify the size of the internal buffer used for rate recovery by setting the `LimitedBufferSize` property.

Data Types: `logical`

`LimitedBufferSize` — Limited buffer size
`25344` (default) | positive integer

Limited buffer size used for rate recovery, specified as a positive integer. The default value corresponds to 384×66, which is the maximum coded length of a code block. The default value implies no limit on the buffer size.

Dependencies

To enable this property, set `LimitedBufferRateRecovery` to `true`.

Data Types: `double`

`MaximumLDPCIterationCount` — Maximum LDPC decoding iterations
`12` (default) | positive integer

Maximum LDPC decoding iterations, specified as a positive integer. Since early termination is enabled, decoding stops once parity-checks are satisfied. In this case, fewer iterations take place than the maximum specified by this argument.

Data Types: `double`

`LDPCDecodingAlgorithm` — LDPC decoding algorithm
`'Belief propagation'` (default) | `'Layered belief propagation'` | `'Normalized min-sum'` | `'Offset min-sum'`

LDPC decoding algorithm, specified as one of these values:

- `'Belief propagation'` — Use this option to specify the belief-passing or message-passing algorithm.
- `'Layered belief propagation'` — Use this option to specify the layered belief-passing algorithm, which is suitable for quasi-cyclic parity-check matrices (PCMs).
- `'Normalized min-sum'` — Use this option to specify the layered belief propagation algorithm with normalized min-sum approximation.
- `'Offset min-sum'` — Use this option to specify the layered belief propagation algorithm with offset min-sum approximation.

For more information on these algorithms, see LDPC Decoding Algorithms on page 2-67.

Data Types: `char` | `string`

`ScalingFactor` — Scaling factor for normalized min-sum decoding
`0.75` (default) | real scalar in the range (0, 1]

Scaling factor for normalized min-sum decoding, specified as a real scalar in the range (0, 1].

Dependencies

To enable this property, set the `LDPCDecodingAlgorithm` property to `'Normalized min-sum'`.

Data Types: `double`

`Offset` — Offset for offset min-sum decoding
`0.5` (default) | nonnegative finite real scalar

Offset for offset min-sum decoding, specified as a nonnegative finite real scalar.

Dependencies

To enable this property, set the `LDPCDecodingAlgorithm` property to `'Offset min-sum'`.

Data Types: `double`

Usage

Syntax

```
trblk = decUL(softbits,mod,nLayers,rv)
trblk = decUL(___,harqID)
[trblk,blkerr] = decUL(___)
```

Description

`trblk = decUL(softbits,mod,nLayers,rv)` applies the UL-SCH decoder processing chain to the input `softbits` and returns the decoded bits. `mod` specifies the modulation scheme. `nLayers` specifies the number of transmission layers. `rv` specifies the redundancy version of the transmission.

`trblk = decUL(___,harqID)` specifies the HARQ process number `harqID` used with the current transport block in addition to the input arguments in the previous syntax. To use this syntax, set the `MultipleHARQProcesses` property to `true`. When the property is set to `false`, the object uses HARQ process number 0.

When the object receives codewords with different redundancy version for an individual HARQ process, the object uses soft buffer state retention to enable soft combining of retransmissions. When you enable multiple HARQ processes, the object maintains independent buffers for each process.

`[trblk,blkerr] = decUL(___)` returns an error flag, using the input arguments in any of the previous syntaxes. A value of 1 in `blkerr` indicates an error during transport block decoding.

Input Arguments

`softbits` — Approximate LLR soft bits
real column vector

Approximate log-likelihood ratio (LLR) soft bits, corresponding to the UL-SCH-encoded transport block, specified as a real column vector.

Data Types: `single` | `double`

`mod` — Modulation scheme
`'pi/2-BPSK'` | `'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'`

Modulation scheme, specified as `'pi/2-BPSK'`, `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`. This modulation scheme determines the modulation type and number of bits used per modulation symbol.

Modulation Scheme	Number of Bits Per Symbol
'pi/2-BPSK'	1
'QPSK'	2
'16QAM'	4
'64QAM'	6
'256QAM'	8

Data Types: `char` | `string`

`nLayers` — Number of transmission layers
integer from 1 to 4

Number of transmission layers, specified as an integer from 1 to 4. For more information, see TS 38.211 Section 6.3.1.3.

Data Types: `double`

`rv` — Redundancy version
integer from 0 to 3

Redundancy version, specified as an integer from 0 to 3.

Data Types: `double`

`harqID` — HARQ process number
integer from 0 to 15

HARQ process number, specified as an integer from 0 to 15.

Data Types: `double`

Output Arguments

`trblk` — Decoded UL-SCH transport blocks
binary column vector

Decoded UL-SCH transport block, returned as a binary column vector of length specified by the `TransportBlockLength` property.

`blkerr` — Result of UL-SCH transport block decoding
logical scalar

Result of UL-SCH transport block decoding, returned as a logical scalar. A value of `1` indicates an error during transport block decoding.

Data Types: `logical`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to `nrULSCHDecoder`

resetSoftBuffer	Reset soft buffer for HARQ process in UL-SCH or DL-SCH decoder

Common to All System Objects

step	Run System object algorithm
clone	Create duplicate System object
isLocked	Determine if System object is in use
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Connect UL-SCH Encoder and Decoder Back to Back

Generate a random sequence of binary values corresponding to one transport block of length 5120.

```
trBlkLen = 5120;
trBlk = randi([0 1],trBlkLen,1,'int8');
```

Create and configure an UL-SCH encoder System object with the specified target code rate.

```
targetCodeRate = 567/1024;
encUL = nrULSCH;
encUL.TargetCodeRate = targetCodeRate;
```

Load the transport block into the UL-SCH encoder.

```
setTransportBlock(encUL,trBlk);
```

Call the encoder with 64-QAM modulation scheme, 1 transmission layer, an output length of 10,240 bits, and redundancy version 0. The encoder applies the UL-SCH processing chain to the transport block loaded into the object.

```
mod = '64QAM';
nLayers = 1;
outlen = 10240;
rv = 0;
codedTrBlock = encUL(mod,nLayers,outlen,rv);
```

Create and configure an UL-SCH decoder System object.

```
decUL = nrULSCHDecoder;
decUL.TargetCodeRate = targetCodeRate;
decUL.TransportBlockLength = trBlkLen;
```

Call the UL-SCH decoder on the soft bits representing the encoded transport block. Use the configuration parameters specified for the encoder. The error flag in the output indicates that the block decoding does not have errors.

```
rxSoftBits = 1.0 - 2.0*double(codedTrBlock);
[decbits,blkerr] = decUL(rxSoftBits,mod,nLayers,rv)
```

```
decbits = 5120x1 int8 column vector

   1
   1
   0
   1
   1
   0
   0
   1
   1
   1
      ⋮
```

```
blkerr = logical
   0
```

Verify that the transmitted and received message bits are identical.

```
isequal(decbits,trBlk)
```

```
ans = logical
   1
```

Algorithms

LDPC Decoding Algorithms

The `nrULSCHDecoder` object supports these four LDPC decoding algorithms.

Belief Propagation Decoding

The implementation of the belief propagation algorithm is based on the decoding algorithm presented in [3]. For transmitted LDPC-encoded codeword, c, where $c = (c_0, c_1, \ldots, c_{n-1})$, the input to the LDPC decoder is the log-likelihood ratio (LLR) value $L(c_i) = \log\left(\frac{\Pr(c_i = 0 \,|\, \text{channel output for } c_i)}{\Pr(c_i = 1 \,|\, \text{channel output for } c_i)}\right)$.

In each iteration, the key components of the algorithm are updated based on these equations:

$$L(r_{ji}) = 2\,\text{atanh}\left(\prod_{i' \in V_j \backslash i} \tanh\left(\frac{1}{2} L(q_{i'j})\right)\right),$$

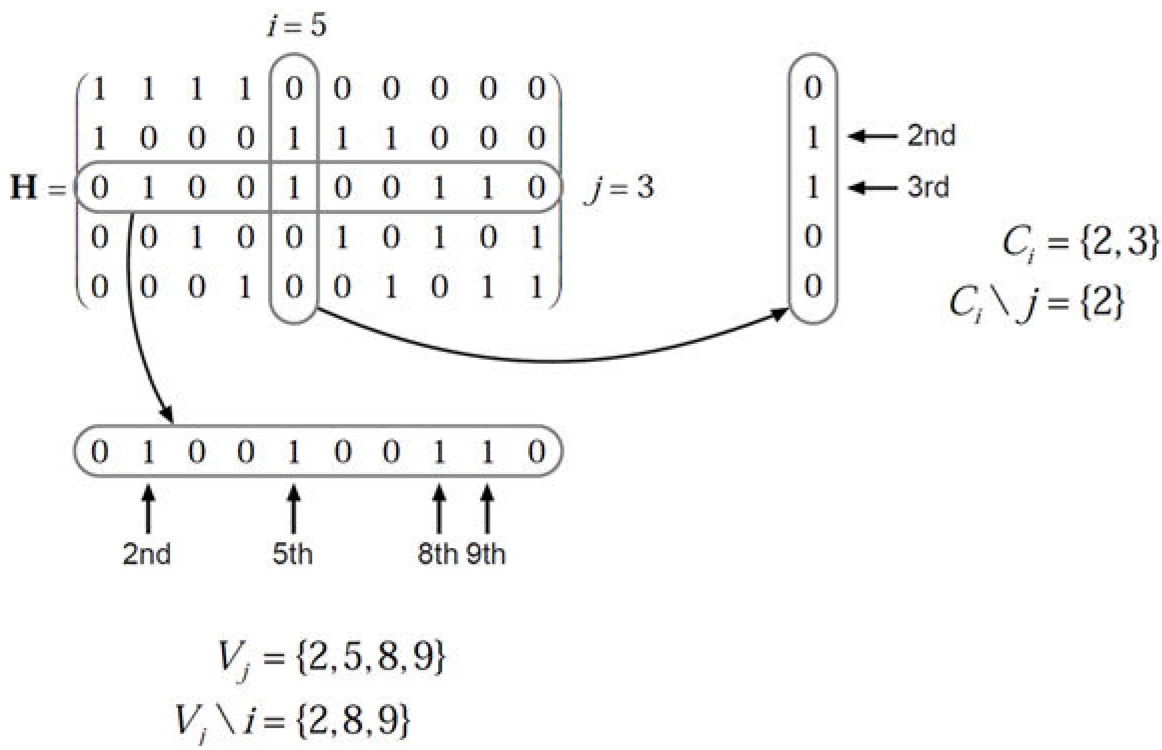
$L(q_{ij}) = L(c_i) + \sum_{j' \in C_i \backslash j} L(r_{j'i})$, initialized as $L(q_{ij}) = L(c_i)$ before the first iteration, and

$$L(Q_i) = L(c_i) + \sum_{j' \in C_i} L(r_{j'i}).$$

At the end of each iteration, $L(Q_i)$ is an updated estimate of the LLR value for the transmitted bit c_i. The value $L(Q_i)$ is the soft-decision output for c_i. If $L(Q_i) < 0$, the hard-decision output for c_i is 1. Otherwise, the output is 0.

Index sets $C_i \backslash j$ and $V_j \backslash i$ are based on the parity-check matrix (PCM). Index sets C_i and V_j correspond to all nonzero elements in column i and row j of the PCM, respectively.

This figure highlights the computation of these index sets in a given PCM for $i = 5$ and $j = 3$.



To avoid infinite numbers in the algorithm equations, atanh(1) and atanh(-1) are set to 19.07 and –19.07, respectively. Due to finite precision, MATLAB returns 1 for tanh(19.07) and –1 for tanh(–19.07).

The decoding terminates when all parity checks are satisfied ($\mathbf{Hc}^{\mathrm{T}} = 0$) or after `MaximumLDPCIterationCount` number of iterations.

Layered Belief Propagation Decoding

The implementation of the layered belief propagation algorithm is based on the decoding algorithm presented in [4], Section II.A. The decoding loop iterates over subsets of rows (layers) of the PCM. For each row, *m*, in a layer and each bit index, *j*, the implementation updates the key components of the algorithm based on these equations:

(1) $L(q_{mj}) = L(q_j) - R_{mj}$,

(2) $A_{mj} = \sum_{\substack{n \in N(m) \\ n \neq j}} \psi(L(q_{mn}))$,

(3) $s_{mj} = \prod_{\substack{n \in N(m) \\ n \neq j}} \mathrm{sign}(L(q_{mn}))$,

(4) $R_{mj} = -s_{mj}\psi(A_{mj})$, and

(5) $L(q_j) = L(q_{mj}) + R_{mj}$.

For each layer, the decoding equation (5) works on the combined input obtained from the current LLR inputs $L(q_{mj})$ and the previous layer updates R_{mj}.

Because only a subset of the nodes is updated in a layer, the layered belief propagation algorithm is faster compared to the belief propagation algorithm. To achieve the same error rate as attained with belief propagation decoding, use half the number of decoding iterations when using the layered belief propagation algorithm.

Normalized Min-Sum Decoding

The implementation of the normalized min-sum decoding algorithm follows the layered belief propagation algorithm with equation (2) replaced by

$$A_{mj} = \min_{\substack{n \in N(m) \\ n \neq j}} (|L(q_{mn})| \cdot \alpha),$$

where α is in the range (0, 1] and is the scaling factor specified by `ScalingFactor`. This equation is an adaptation of equation (4) presented in [5].

Offset Min-Sum Decoding

The implementation of the offset min-sum decoding algorithm follows the layered belief propagation algorithm with equation (2) replaced by

$$A_{mj} = \max\left(\min_{\substack{n \in N(m) \\ n \neq j}} (|L(q_{mn})| - \beta), 0\right),$$

where $\beta \geq 0$ and is the offset specified by `Offset`. This equation is an adaptation of equation (5) presented in [5].

Version History

Introduced in R2019a

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] Gallager, Robert G. *Low-Density Parity-Check Codes*, Cambridge, MA, MIT Press, 1963.

[4] Hocevar, D.E. "A reduced complexity decoder architecture via layered decoding of LDPC codes." In *IEEE Workshop on Signal Processing Systems, 2004. SIPS 2004*. doi: 10.1109/SIPS.2004.1363033

[5] Chen, Jinghu, R.M. Tanner, C. Jones, and Yan Li. "Improved min-sum decoding algorithms for irregular LDPC codes." In *Proceedings. International Symposium on Information Theory, 2005. ISIT 2005*. doi: 10.1109/ISIT.2005.1523374

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also

Objects
`nrULSCH` | `nrDLSCHDecoder`

Functions
`nrPUSCHDecode` | `nrULSCHInfo`

nrTDLChannel

Send signal through TDL channel model

Description

The `nrTDLChannel` System object sends an input signal through a tapped delay line (TDL) multi-input multi-output (MIMO) link-level fading channel to obtain the channel-impaired signal. The object implements the following aspects of TR 38.901 [1]:

- Section 7.7.2: TDL models
- Section 7.7.3: Scaling of delays
- Section 7.7.5.2 TDL extension: Applying a correlation matrix
- Section 7.7.6: K-factor for LOS channel models

To send a signal through the TDL MIMO channel model:

1. Create the `nrTDLChannel` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
tdl = nrTDLChannel
tdl = nrTDLChannel(Name,Value)
```

Description

`tdl = nrTDLChannel` creates a TDL MIMO channel System object.

`tdl = nrTDLChannel(Name,Value)` creates the object with properties set by using one or more name-value pairs. Enclose the property name inside quotes, followed by the specified value. Unspecified properties take default values.

Example: `tdl = nrTDLChannel('DelayProfile','TDL-D','DelaySpread',2e-6)` creates a TDL channel model with TDL-D delay profile and a 2-microseconds delay spread.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

Configurable Channel Properties

DelayProfile — TDL delay profile
'TDL-A' (default) | 'TDL-B' | 'TDL-C' | 'TDL-D' | 'TDL-E' | 'TDLA30' | 'TDLB100' | 'TDLC300' | 'TDLC60' | 'Custom'

TDL delay profile, specified as one of these values.

- 'TDL-A', 'TDL-B', 'TDL-C', 'TDL-D', or 'TDL-E' — These values correspond to the delay profiles defined in TR 38.901 Section 7.7.2, Tables 7.7.2-1 to 7.7.2-5.
- 'TDLA30', 'TDLB100', 'TDLC300', or 'TDLC60' — These values correspond to the simplified delay profiles defined in TS 38.101-4 Annex B.2.1 and TS 38.104 Annex G.2.1.
- 'Custom' — Configure the delay profile using the PathDelays, AveragePathGains, FadingDistribution, and KFactorFirstTap properties.

Data Types: char | string

PathDelays — Discrete path delays in seconds
0.0 (default) | numeric scalar | row vector

Discrete path delays in seconds, specified as a numeric scalar or row vector. AveragePathGains and PathDelays must have the same size.

Dependencies

To enable this property, set DelayProfile to 'Custom'.

Data Types: double

AveragePathGains — Average path gains in dB
0.0 (default) | numeric scalar | row vector

Average path gains in dB, specified as a numeric scalar or row vector. AveragePathGains and PathDelays must have the same size.

Dependencies

To enable this property, set DelayProfile to 'Custom'.

Data Types: double

FadingDistribution — Fading process statistical distribution
'Rayleigh' (default) | 'Rician'

Fading process statistical distribution, specified as 'Rayleigh' or 'Rician'.

Dependencies

To enable this property, set DelayProfile to 'Custom'.

Data Types: char | string

KFactorFirstTap — K-factor of first tap of delay profile in dB
13.3 (default) | numeric scalar

K-factor of first tap of delay profile in dB, specified as a numerical scalar. The default value corresponds to the K-factor of the first tap of TDL-D as defined in TR 38.901 Section 7.7.2, Table 7.7.2-4.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'` and `FadingDistribution` to `'Rician'`.

Data Types: `double`

`DelaySpread` — Desired RMS delay spread in seconds

`30e-9` (default) | numeric scalar

Desired root mean square (RMS) delay spread in seconds, specified as a numeric scalar. For examples of desired RMS delay spreads, $DS_{desired}$, see TR 38.901 Section 7.7.3 and Tables 7.7.3-1 and 7.7.3-2.

Dependencies

To enable this property, set `DelayProfile` to `'TDL-A'`, `'TDL-B'`, `'TDL-C'`, `'TDL-D'`, or `'TDL-E'`. This property does not apply for custom delay profile.

Data Types: `double`

`MaximumDopplerShift` — Maximum Doppler shift in Hz

`5` (default) | nonnegative numeric scalar

Maximum Doppler shift in Hz, specified as a nonnegative numeric scalar. This property applies to all channel paths. When the maximum Doppler shift is set to 0, the channel remains static for the entire input. To generate a new channel realization, reset the object by calling the `reset` function.

Data Types: `double`

`KFactorScaling` — K-factor scaling

`false` (default) | `true`

K-factor scaling, specified as `false` or `true`. When set to `true`, the `KFactor` property specifies the desired K-factor, and the object applies K-factor scaling as described in TR 38.901 Section 7.7.6.

Note K-factor scaling modifies both the path delays and path powers.

Dependencies

To enable this property, set `DelayProfile` to `'TDL-D'` or `'TDL-E'`.

Data Types: `double`

`KFactor` — Desired K-factor for scaling in dB

`9.0` (default) | numeric scalar

Desired K-factor for scaling in dB, specified as a numeric scalar. For typical K-factor values, see TR 38.901 Section 7.7.6 and Table 7.5-6.

Note

- K-factor scaling modifies both the path delays and path powers.

- `K-factor` applies to the overall delay profile. Specifically, the K-factor after the scaling is K_{model} as described in TR 38.901 Section 7.7.6. K_{model} is the ratio of the power of the first path LOS to the total power of all the Rayleigh paths, including the Rayleigh part of the first path.

Dependencies

To enable this property, set `KFactorScaling` to `true`.

Data Types: `double`

`SampleRate` — Sample rate of input signal in Hz
`30720000` (default) | positive numeric scalar

Sample rate of the input signal in Hz, specified as a positive numeric scalar.

Data Types: `double`

`MIMOCorrelation` — Correlation between UE and BS antennas
`'Low'` (default) | `'Medium'` | `'Medium-A'` | `'UplinkMedium'` | `'High'` | `'Custom'`

Correlation between user equipment (UE) and base station (BS) antennas, specified as one of these values:

- `'Low'` or `'High'` — Applies to both uplink and downlink. `'Low'` is equivalent to no correlation between antennas.
- `'Medium'` or `'Medium-A'` — For downlink, see TS 36.101 Annex B.2.3.2. For uplink, see TS 36.104 Annex B.5.2. The `TransmissionDirection` property controls the transmission direction.
- `'UplinkMedium'` — See TS 36.104, Annex B.5.2.
- `'Custom'` — The `ReceiveCorrelationMatrix` property specifies the correlation between UE antennas, and the `TransmitCorrelationMatrix` property specifies the correlation between BS antennas. See TR 38.901 Section 7.7.5.2.

For more details on correlation between UE and BS antennas, see TS 36.101 [2] and TS 36.104 [3]

Data Types: `char` | `string`

`Polarization` — Antenna polarization arrangement
`'Co-Polar'` (default) | `'Cross-Polar'` | `'Custom'`

Antenna polarization arrangement, specified as `'Co-Polar'`, `'Cross-Polar'`, `'Custom'`.

Data Types: `char` | `string`

`TransmissionDirection` — Transmission direction
`'Downlink'` (default) | `'Uplink'`

Transmission direction, specified as `'Downlink'` or `'Uplink'`.

Dependencies

To enable this property, set `MIMOCorrelation` to `'Low'`, `'Medium'`, `'Medium-A'`, `'UplinkMedium'`, or `'High'`.

Note This property describes the transmission direction corresponding to the channel status in which the role of the transmit and receive antennas are not swapped. If the antennas are swapped, the opposite transmission direction applies to this property. To determine the current link direction of the channel, inspect the `TransmitAndReceiveSwapped` property value.

Data Types: `char` | `string`

`NumTransmitAntennas` — Number of transmit antennas
`1` (default) | positive integer

Number of transmit antennas, specified as a positive integer.

Dependencies

To enable this property, set `MIMOCorrelation` to `'Low'`, `'Medium'`, `'Medium-A'`, `'UplinkMedium'`, or `'High'`, or set both `MIMOCorrelation` and `Polarization` to `'Custom'`.

Data Types: `double`

`NumReceiveAntennas` — Number of receive antennas
2 (default) | positive integer

Number of receive antennas, specified as a positive integer.

Dependencies

To enable this property, set `MIMOCorrelation` to `'Low'`, `'Medium'`, `'Medium-A'`, `'UplinkMedium'`, or `'High'`.

Data Types: `double`

`TransmitCorrelationMatrix` — Spatial correlation of transmitter
`[1]` (default) | 2-D matrix | 3-D array

Spatial correlation of transmitter, specified as a 2-D matrix or 3-D array.

- If the channel is frequency-flat (`PathDelays` is a scalar), specify `TransmitCorrelationMatrix` as a 2-D Hermitian matrix of size N_T-by-N_T. N_T is the number of transmit antennas. The main diagonal elements must be all ones, and the off-diagonal elements must have a magnitude smaller than or equal to one.
- If the channel is frequency-selective (`PathDelays` is a row vector of length N_P), specify `TransmitCorrelationMatrix` as one of these arrays:
  - 2-D Hermitian matrix of size N_T-by-N_T with element properties as previously described. Each path has the same transmit correlation matrix.
  - 3-D array of size N_T-by-N_T-by-N_P, where each submatrix of size N_T-by-N_T is a Hermitian matrix with element properties as previously described. Each path has its own transmit correlation matrix.

Dependencies

To enable this property, set `MIMOCorrelation` to `'Custom'` and `Polarization` to either `'Co-Polar'` or `'Cross-Polar'`.

Data Types: `double`

`ReceiveCorrelationMatrix` — Spatial correlation of receiver
`[1 0; 0 1]` (default) | 2-D matrix | 3-D array

Spatial correlation of receiver, specified as a 2-D matrix or 3-D array.

- If the channel is frequency-flat (`PathDelays` is a scalar), specify `ReceiveCorrelationMatrix` as a 2-D Hermitian matrix of size N_R-by-N_R. N_R is the number of receive antennas. The main diagonal elements must be all ones, and the off-diagonal elements must have a magnitude smaller than or equal to one.
- If the channel is frequency-selective (`PathDelays` is a row vector of length N_P), specify `ReceiveCorrelationMatrix` as one of these arrays:
  - 2-D Hermitian matrix of size N_R-by-N_R with element properties as previously described. Each path has the same receive correlation matrix.
  - 3-D array of size N_R-by-N_R-by-N_P, where each submatrix of size N_R-by-N_R is a Hermitian matrix with element properties as previously described. Each path has its own receive correlation matrix.

Dependencies

To enable this property, set `MIMOCorrelation` to `'Custom'` and `Polarization` to either `'Co-Polar'` or `'Cross-Polar'`.

Data Types: `double`

`TransmitPolarizationAngles` — Transmit polarization slant angles in degrees
`[45 -45]` (default) | row vector

Transmit polarization slant angles in degrees, specified as a row vector.

Dependencies

To enable this property, set `MIMOCorrelation` to `'Custom'` and `Polarization` to `'Cross-Polar'`.

Data Types: `double`

`ReceivePolarizationAngles` — Receive polarization slant angles in degrees
`[90 0]` (default) | row vector

Receive polarization slant angles in degrees, specified as a row vector.

Dependencies

To enable this property, set `MIMOCorrelation` to `'Custom'` and `Polarization` to `'Cross-Polar'`.

Data Types: `double`

XPR — Cross-polarization power ratio in dB
`10.0` (default) | numeric scalar | row vector

Cross-polarization power ratio in dB, specified as a numeric scalar or a row vector. This property corresponds to the ratio between the vertical-to-vertical (P_{VV}) and vertical-to-horizontal (P_{VH}) polarizations defined for the clustered delay line (CDL) models in TR 38.901 Section 7.7.1.

- If the channel is frequency-flat (`PathDelays` is a scalar), specify `XPR` as a scalar.
- If the channel is frequency-selective (`PathDelays` is a row vector of length N_P), specify `XPR` as one of these values:

- Scalar — Each path has the same cross-polarization power ratio.
- Row vector of size 1-by-N_P — Each path has its own cross-polarization power ratio.

The default value corresponds to the cluster-wise cross-polarization power ratio of CDL-A as defined in TR 38.901 Section 7.7.1, Table 7.7.1-1.

Dependencies

To enable this property, set `MIMOCorrelation` to `'Custom'` and `Polarization` to `'Cross-Polar'`.

Data Types: `double`

`SpatialCorrelationMatrix` — Combined correlation for channel

`[1 0; 0 1]` (default) | 2-D matrix | 3-D array

Combined correlation for the channel, specified as 2-D matrix or 3-D array. The matrix determines the product of the number of transmit antennas (N_T) and the number of receive antennas (N_R).

- If the channel is frequency-flat (`PathDelays` is a scalar), specify `SpatialCorrelationMatrix` as a 2-D Hermitian matrix of size ($N_T \times N_R$)-by-($N_T \times N_R$).The magnitude of any off-diagonal element must be no larger than the geometric mean of the two corresponding diagonal elements.
- If the channel is frequency-selective (`PathDelays` is a row vector of length N_P), specify `SpatialCorrelationMatrix` as one of these arrays:
  - 2-D Hermitian matrix of size ($N_T \times N_R$)-by-($N_T \times N_R$) with off-diagonal element properties as previously described. Each path has the same spatial correlation matrix.
  - 3-D array of size ($N_T \times N_R$)-by-($N_T \times N_R$)-by-N_P array — where each matrix of size ($N_T \times N_R$)-by-($N_T \times N_R$) is a Hermitian matrix with off-diagonal element properties as previously described. Each path has its own spatial correlation matrix.

Dependencies

To enable this property, set `MIMOCorrelation` to `'Custom'` and `Polarization` to `'Custom'`.

Data Types: `double`

`NormalizePathGains` — Normalize path gains

`true` (default) | `false`

Normalize path gains, specified as `true` or `false`. Use this property to normalize the fading processes. When this property is set to `true`, the total power of the path gains, averaged over time, is 0 dB. When this property is set to `false`, the path gains are not normalized. The average powers of the path gains are specified by the selected delay profile, or if `DelayProfile` is set to `'Custom'`, by the `AveragePathGains` property.

Data Types: `logical`

`InitialTime` — Time offset of fading process in seconds

`0.0` (default) | numeric scalar

Time offset of fading process in seconds, specified as a numeric scalar.

Data Types: `double`

`NumSinusoids` — Number of modeling sinusoids

`48` (default) | positive integer

Number of modeling sinusoids, specified as a positive integer. These sinusoids model the fading process.

Data Types: double

RandomStream — Source of random number stream
'mt19937ar with seed' (default) | 'Global stream'

Source of the random number stream to initialize the sinusoid phases using uniformly distributed random numbers, specified as one of these values.

- 'mt19937ar with seed' — The object uses the mt19937ar algorithm for the random number generation. Calling the reset function resets the filters and reinitializes the random number stream to the value of the Seed property. Specifying this value results in repeatable channel fading.
- 'Global stream' — The object uses the current global random number stream for the random number generation. Calling the reset function resets only the filters.

Seed — Initial seed of mt19937ar random number stream
73 (default) | nonnegative numeric scalar

Initial seed of mt19937ar random number stream, specified as a nonnegative numeric scalar.

Dependencies

To enable this property, set RandomStream to 'mt19937ar with seed'. When calling the reset function, the seed reinitializes the mt19937ar random number stream.

Data Types: double

NormalizeChannelOutputs — Normalize channel outputs
true (default) | false

Normalize channel outputs, specified as true or false. When this property is set to true, the channel outputs are normalized by the number of receive antenna elements.

Note When you call the swapTransmitAndReceive function to reverse the role of the transmit and receive antennas within the channel, the function also swaps the NumTransmitAntennas and NumReceiveAntennas properties. Hence the normalization is always by the number of receive antenna elements, specified by the NumReceiveAntennas property.

Data Types: logical

ChannelFiltering — Fading channel filtering
true (default) | false

Fading channel filtering, specified as true or false. When this property is set to false, these conditions apply.

- The object takes no input signal and returns only the path gains and sample times.
- The NumTimeSamples property controls the duration of the fading process realization at a sampling rate given by the SampleRate property.
- The channel coefficients sampling rate is one sample per each time sample from 0 to NumTimeSamples – 1.

Data Types: `logical`

`NumTimeSamples` — Number of time samples
`30720` (default) | positive integer

Number of time samples, specified as a positive integer. Use this property to set the duration of the fading process realization.

Tunable: Yes

Dependencies

To enable this property, set `ChannelFiltering` to `false`.

Data Types: `double`

`OutputDataType` — Data type of generated path gains
`'double'` (default) | 'single'

Data type of generated path gains, specified as `'double'` or `'single'`.

Dependencies

To enable this property, set `ChannelFiltering` to `false`.

Data Types: `double`

Nonconfigurable Channel Properties

`TransmitAndReceiveSwapped` — Reversed channel link direction
`false` (default) | `true`

This property is read-only.

Reversed channel link direction, returned as one of these values.

- `false` — The role of the transmit and receive antennas within the channel model corresponds to the original channel link direction. Calling the `swapTransmitAndReceive` function on the `nrTDLChannel` object reverses the link direction of the channel and toggles this property value from `false` to `true`.
- `true` — The role of the transmit and receive antennas within the channel model are swapped. Calling the `swapTransmitAndReceive` function on the `nrTDLChannel` object restores the original link direction of the channel and toggles this property value from `true` to `false`.

Data Types: `logical`

Usage

Syntax

```
signalOut = tdl(signalIn)
[signalOut,pathGains] = tdl(signalIn)
[signalOut,pathGains,sampleTimes] = tdl(signalIn)

[pathGains,sampleTimes] = tdl()
```

Description

`signalOut = tdl(signalIn)` sends the input signal through a TDL MIMO fading channel and returns the channel-impaired signal.

`[signalOut,pathGains] = tdl(signalIn)` also returns the MIMO channel path gains of the underlying fading process.

`[signalOut,pathGains,sampleTimes] = tdl(signalIn)` also returns the sample times of the channel snapshots of the path gains.

`[pathGains,sampleTimes] = tdl()` returns only the path gains and the sample times. The `tdl` object acts as a source of the path gains and sample times without filtering an input signal. The `NumTimeSamples` object property specifies the duration of the fading process and the `OutputDataType` object property specifies the data type of the generated path gains. To use this syntax, you must set the `ChannelFiltering` object property to `false`.

Input Arguments

`signalIn` — Input signal

complex scalar | vector | N_S-by-N_T matrix

Input signal, specified as a complex scalar, vector, or N_S-by-N_T matrix, where:

- N_S is the number of samples.
- N_T is the number of transmit antennas.

Data Types: `single` | `double`
Complex Number Support: Yes

Output Arguments

`signalOut` — Output signal

complex scalar | vector | N_S-by-N_R matrix

Output signal, returned as a complex scalar, vector, or N_S-by-N_R matrix, where:

- N_S is the number of samples.
- N_R is the number of receive antennas.

The output signal data type is of the same precision as the input signal data type.

Data Types: `single` | `double`
Complex Number Support: Yes

`pathGains` — MIMO channel path gains of fading process

N_S-by-N_P-by-N_T-by-N_R complex matrix

MIMO channel path gains of the fading process, returned as an N_S-by-N_P-by-N_T-by-N_R complex matrix, where:

- N_S is the number of samples.
- N_P is the number of paths, specified by the length of the `PathDelays` property of `tdl`.
- N_T is the number of transmit antennas.

- N_R is the number of receive antennas.

The path gains data type is of the same precision as the input signal data type.

Data Types: `single` | `double`
Complex Number Support: Yes

`sampleTimes` — Sample times of channel snapshots
N_S-by-1 column vector of real numbers

Sample times of the channel snapshots of the path gains, returned as an N_S-by-1 column vector of real numbers. N_S is the first dimension of `pathGains` that corresponds to the number of samples.

Data Types: `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to `nrTDLChannel`

info	Characteristic information of link-level MIMO channel
getPathFilters	Get path filter impulse response for link-level MIMO channel
swapTransmitAndReceive	Reverse link direction in TDL channel model

Common to All System Objects

step	Run System object algorithm
clone	Create duplicate System object
isLocked	Determine if System object is in use
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Transmission Over MIMO Channel Model with Delay Profile TDL

Display the waveform spectrum received through a tapped delay line (TDL) multi-input/multi-output (MIMO) channel model from TR 38.901 Section 7.7.2 using an `nrTDLChannel` System object.

Define the channel configuration structure using an `nrTDLChannel` System object. Use delay profile TDL-C from TR 38.901 Section 7.7.2, a delay spread of 300 ns, and UE velocity of 30 km/h:

```
v = 30.0;                    % UE velocity in km/h
fc = 4e9;                    % carrier frequency in Hz
c = physconst('lightspeed'); % speed of light in m/s
fd = (v*1000/3600)/c*fc;     % UE max Doppler frequency in Hz

tdl = nrTDLChannel;
tdl.DelayProfile = 'TDL-C';
```

```
tdl.DelaySpread = 300e-9;
tdl.MaximumDopplerShift = fd;
```

Create a random waveform of 1 subframe duration with 1 antenna.

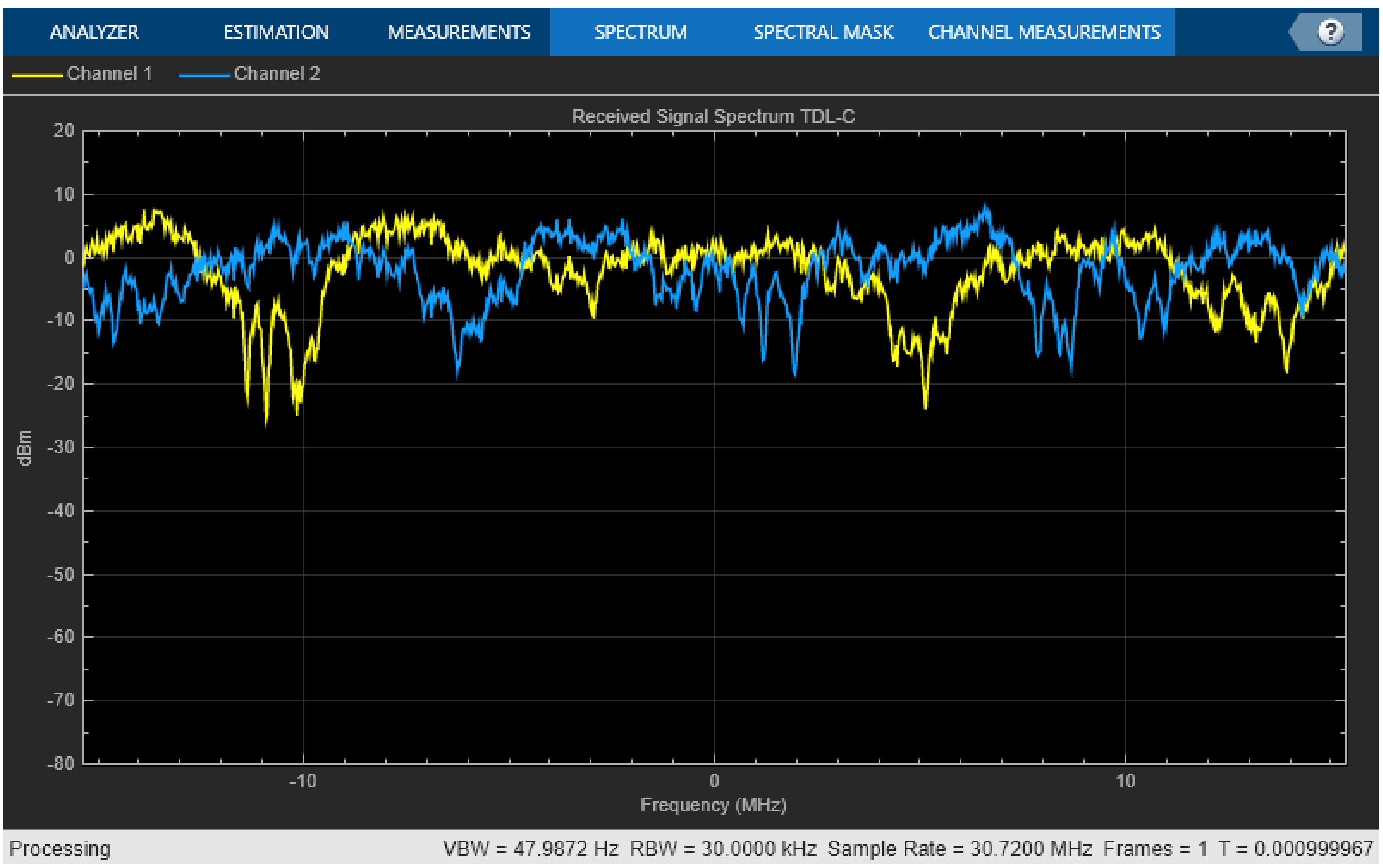
```
SR = 30.72e6;
T = SR * 1e-3;
tdl.SampleRate = SR;
tdlinfo = info(tdl);
Nt = tdlinfo.NumTransmitAntennas;

txWaveform = complex(randn(T,Nt),randn(T,Nt));
```

Transmit the input waveform through the channel.

```
rxWaveform = tdl(txWaveform);
```

Plot the received waveform spectrum.

```
analyzer = spectrumAnalyzer('SampleRate',tdl.SampleRate,...
    'AveragingMethod','exponential','ForgettingFactor',0.99 );
analyzer.Title = ['Received Signal Spectrum ' tdl.DelayProfile];
analyzer(rxWaveform);
```



Plot Path Gains for TDL-E Delay Profile with SISO

Plot the path gains of a tapped delay line (TDL) single-input/single-output (SISO) channel using an `nrTDLChannel` System object.

Configure a channel with delay profile TDL-E from TR 38.901 Section 7.7.2. Set the maximum Doppler shift to 70 Hz and enable path gain output.

```
tdl = nrTDLChannel;
tdl.SampleRate = 500e3;
tdl.MaximumDopplerShift = 70;
tdl.DelayProfile = 'TDL-E';
```

Configure the transmit and receive antenna arrays for SISO operation.

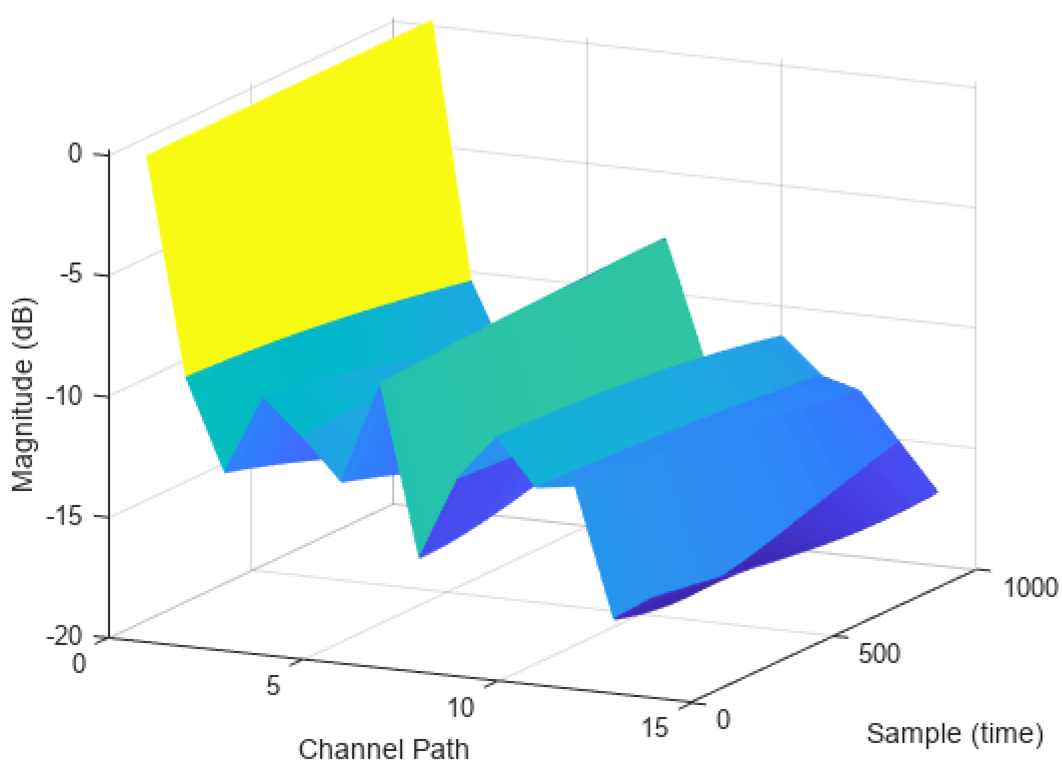
```
tdl.NumTransmitAntennas = 1;
tdl.NumReceiveAntennas = 1;
```

Create a dummy input signal. The length of the input determines the time samples of the generated path gain.

```
in = zeros(1000,tdl.NumTransmitAntennas);
```

To generate the path gains, call the channel on the input. Plot the results.

```
[~, pathGains] = tdl(in);
mesh(10*log10(abs(pathGains)));
view(26,17); xlabel('Channel Path');
ylabel('Sample (time)'); zlabel('Magnitude (dB)');
```



Transmission Over TDL-D Channel Model with Cross-Polar Antennas

Display the waveform spectrum received through a tapped delay line (TDL) channel model using delay profile TDL-D from TR 38.901 Section 7.7.2.

Configure 4-by-2, high-correlation, cross-polar antennas as specified in TS 36.101 Annex B.2.3A.3.

```
tdl = nrTDLChannel;
tdl.NumTransmitAntennas = 4;
tdl.DelayProfile = 'TDL-D';
tdl.DelaySpread = 10e-9;
tdl.KFactorScaling = true;
tdl.KFactor = 7.0;
tdl.MIMOCorrelation = 'High';
tdl.Polarization = 'Cross-Polar';
```

Create a random waveform of 1 subframe duration with 4 antennas.

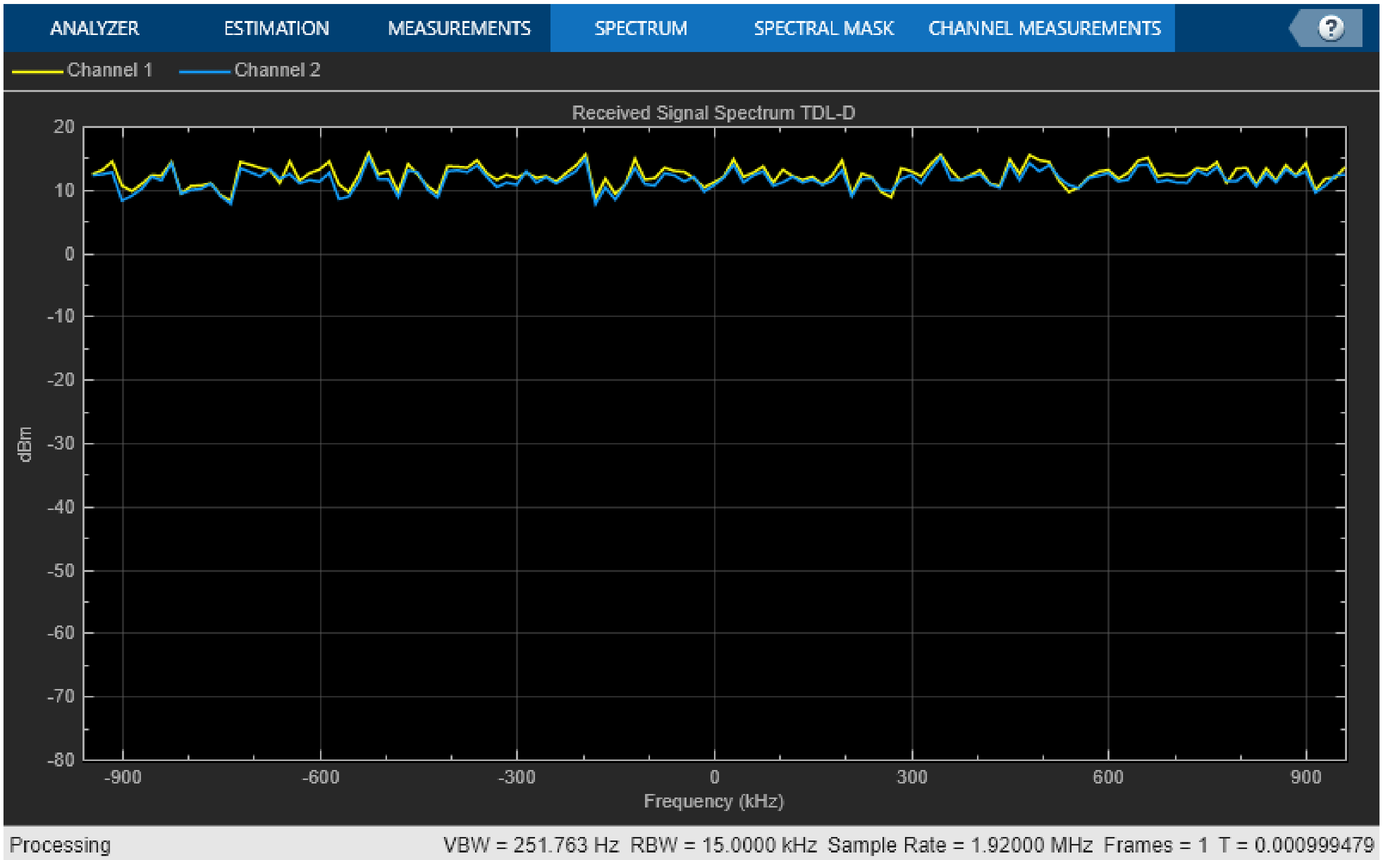
```
SR = 1.92e6;
T = SR * 1e-3;
tdl.SampleRate = SR;
tdlinfo = info(tdl);
Nt = tdlinfo.NumTransmitAntennas;

txWaveform = complex(randn(T,Nt),randn(T,Nt));
```

Transmit the input waveform through the channel.

```
rxWaveform = tdl(txWaveform);
```

Plot the received waveform spectrum.

```
analyzer = spectrumAnalyzer('SampleRate',tdl.SampleRate,'RBWSource','property','RBW',15e3);
analyzer.Title = ['Received Signal Spectrum ' tdl.DelayProfile];
analyzer(rxWaveform);
```



Transmission Over TDL Channel Model with Custom Delay Profile

Transmit waveform through a tapped delay line (TDL) channel model from TR 38.901 Section 7.7.2 with customized delay profile.

Define the channel configuration structure using an `nrTDLChannel` System object. Customize the delay profile with two taps.

- First tap: Rician with average power 0 dB, K-factor 10 dB, and zero delay.
- Second tap: Rayleigh with average power −5 dB, and 45 ns path delay using TDL-D.

```
tdl = nrTDLChannel;
tdl.NumTransmitAntennas = 1;
tdl.DelayProfile = 'Custom';
tdl.FadingDistribution = 'Rician';
tdl.KFactorFirstTap = 10.0;
tdl.PathDelays = [0.0 45e-9];
tdl.AveragePathGains = [0.0 -5.0];
```

Create a random waveform of 1 subframe duration with 1 antenna.

```
SR = 30.72e6;
T = SR * 1e-3;
```

```
tdl.SampleRate = SR;
tdlinfo = info(tdl);
Nt = tdlinfo.NumTransmitAntennas;

txWaveform = complex(randn(T,Nt),randn(T,Nt));
```

Transmit the input waveform through the channel.

```
rxWaveform = tdl(txWaveform);
```

Version History

Introduced in R2018b

References

[1] 3GPP TR 38.901. “Study on channel model for frequencies from 0.5 to 100 GHz.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 36.104. “Evolved Universal Terrestrial Radio Access (E-UTRA); Base Station (BS) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

Functions

`nrPerfectTimingEstimate` | `nrPerfectChannelEstimate`

Objects

`comm.MIMOChannel` | `nrCDLChannel`

Topics

“TDD Reciprocity-Based PDSCH MU-MIMO Using SRS”

3

Objects

networkTrafficFTP

FTP application traffic pattern generator

Description

The `networkTrafficFTP` object specifies the configuration parameters to generate a file transfer protocol (FTP) application traffic pattern based on the 3GPP TR 36.814 specification and the IEEE® 802.11ax™ Evaluation Methodology.

You can use the FTP application traffic pattern in 5G and WLAN (requires WLAN Toolbox™) system-level simulations to accurately model the real-world data traffic.

Creation

Syntax

```
cfgFTP = networkTrafficFTP
cfgFTP = networkTrafficFTP(Name,Value)
```

Description

`cfgFTP = networkTrafficFTP` creates a default FTP application traffic pattern object.

`cfgFTP = networkTrafficFTP(Name,Value)` sets properties on page 3-2 using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'GeneratePacket',true` generates an FTP packet.

Properties

`FixedFileSize` — Custom size of file to be transmitted
[ ] (default) | positive scalar

Custom size of the file to be transmitted, specified as a positive scalar. This property must be expressed in megabytes. If you do not specify this property, the object uses the truncated Lognormal distribution to calculate the file size.

Data Types: `double`

`LogNormalMu` — Truncated Lognormal distribution mu value
`14.45` (default) | positive scalar

Truncated Lognormal distribution mu value, specified as a positive scalar. The object uses this property to calculate the file size.

Dependencies

To enable this property, set the `FixedFileSize` property to `[ ]`.

Data Types: `double`

LogNormalSigma — Truncated Lognormal distribution sigma value
0.35 (default) | positive scalar

Truncated Lognormal distribution sigma value, specified as a positive scalar. The object uses this value to calculate the file size.

Dependencies

To enable this property, set the FixedFileSize property to [ ].

Data Types: double

UpperLimit — Truncated Lognormal distribution upper limit
5 (default) | positive scalar

Truncated Lognormal distribution upper limit, specified as a positive scalar. The object uses this value to calculate the file size. The generated file size value must be less than or equal to the UpperLimit value. If the generated file size value is greater than UpperLimit, the object discards the file size and creates a new one.

Dependencies

To enable this property, set the FixedFileSize property to [ ].

Data Types: double

ReadingTime — Time interval between two successive file transfers
[ ] (default) | positive scalar

Time interval between two consecutive file transfers, specified as a positive scalar. This property must be expressed in milliseconds. To specify a customized value for the reading time, specify this property. If you do not specify this property, the object uses the exponential distribution to calculate the reading time.

Data Types: double

ExponentialMean — Exponential distribution mean value
180000 (default) | nonnegative scalar

Exponential distribution mean value, specified as a nonnegative scalar. This property must be expressed in milliseconds. The object uses this property to calculate the reading time.

Dependencies

To enable this property, set the ReadingTime property to [ ].

Data Types: double

PoissonMean — Poisson distribution mean value
[ ] (default) | nonnegative scalar

Poisson distribution mean value, specified as a nonnegative scalar. This property must be expressed in milliseconds. The object uses this property to calculate the packet interarrival time.

Data Types: double

PacketInterArrivalTime — Time interval between two successively generated packets
0 (default) | nonnegative scalar

Time interval between two successively generated packets, specified as a nonnegative scalar. This property must be expressed in milliseconds.

Dependencies

To enable this property, set the `PoissonMean` property to `[ ]`.

Data Types: `double`

`GeneratePacket` — Flag to indicate whether to generate FTP packet
`false` or `0` (default) | `true` or `1`

Flag to indicate whether the object generates an FTP packet, specified as a `logical` `1` (`true`) or `0` (`false`).

Data Types: `logical`

`ApplicationData` — Application data to be added in FTP packet
1500-by-1 vector of 1s (default) | column vector of integers in the range [0, 255]

Application data to be added in the FTP packet, specified as a column vector of integers in the range [0, 255]. If the size of the application data is greater than the packet size, the object truncates the application data. If the size of the application data is less than the packet size, the object appends zeros.

Dependencies

To enable this property, set the `GeneratePacket` property to `true` or `1`.

Data Types: `double`

Object Functions

Specific to This Object

generate   Generate next FTP, On-Off, VoIP, or video conference application traffic packet

Examples

Generate FTP Application Traffic Pattern

Create a default FTP application traffic pattern object.

```
cfgFTP = networkTrafficFTP;
```

Generate an FTP application traffic pattern.

```
[dt,packetSize] = generate(cfgFTP);
```

Generate FTP Application Traffic Pattern Using Reading Time

Create an FTP application traffic pattern object, specifying the reading time.

```
cfgFTP = networkTrafficFTP('ReadingTime',5);
```

Generate an FTP application traffic pattern.

```
[dt,packetSize] = generate(cfgFTP);
```

Generate FTP Application Traffic Pattern and Data Packet

Create an FTP application traffic pattern object to generate an FTP data packet.

```
cfgFTP = networkTrafficFTP('GeneratePacket',true);
```

Generate an FTP application traffic pattern and data packet.

```
[dt,packetSize,packet] = generate(cfgFTP);
```

Generate FTP Application Traffic Pattern to Visualize Packet Sizes and Intervals

Create a default FTP application traffic pattern object.

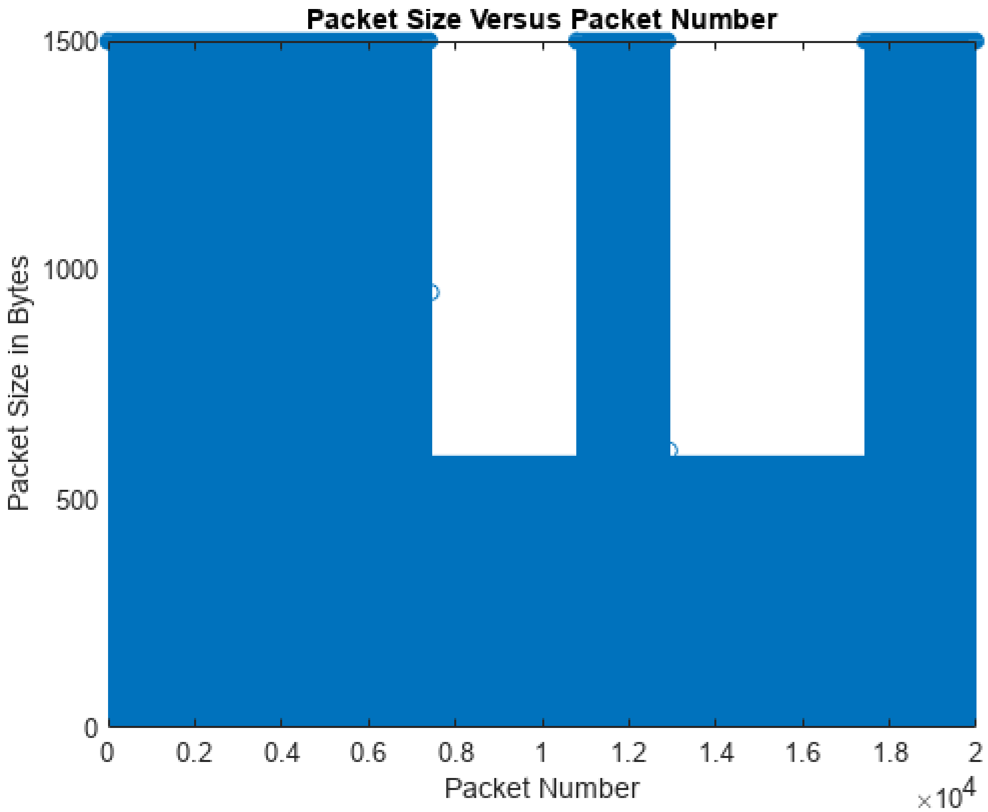
```
cfgFTP = networkTrafficFTP;
```

Generate an FTP application traffic pattern with 20,000 FTP packets.

```
for packetCount = 1:20000
    [dt(packetCount),packetSize(packetCount)] = generate(cfgFTP);
end
```

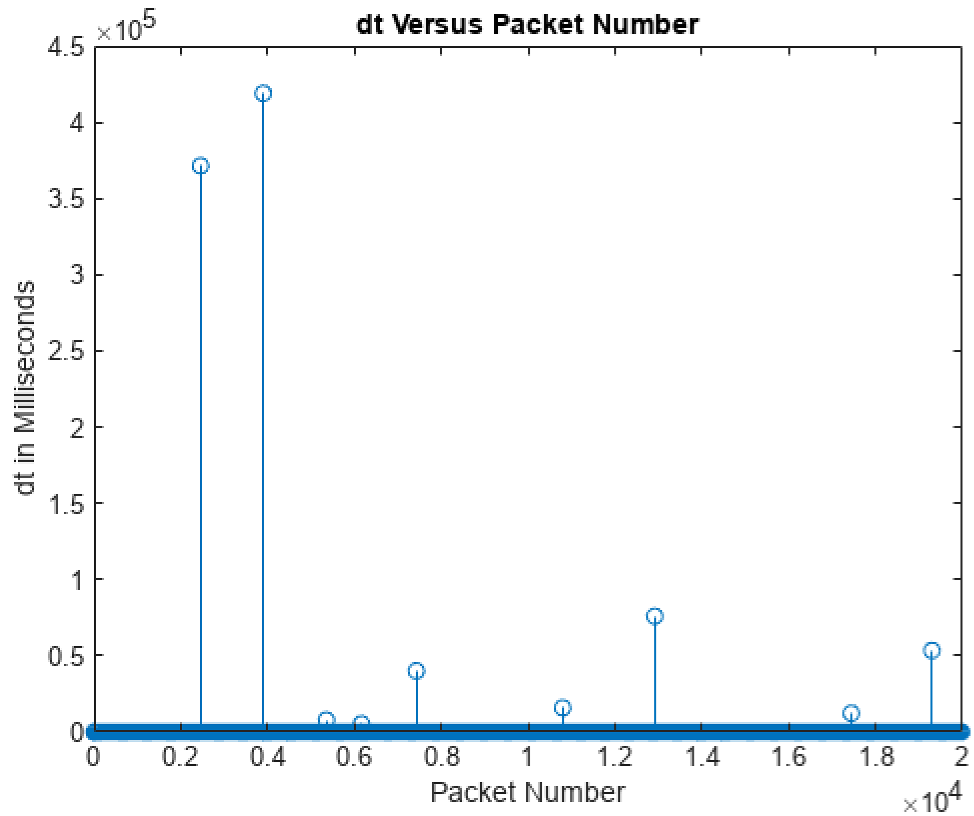
Visualize the FTP packet sizes.

```
stem(packetSize);
title('Packet Size Versus Packet Number');
xlabel('Packet Number');
ylabel('Packet Size in Bytes');
```



Visualize the FTP packet intervals.

```
figure;
stem(dt);
title('dt Versus Packet Number');
xlabel('Packet Number');
ylabel('dt in Milliseconds');
```



Version History

Introduced in R2020b

References

[1] 3GPP TR 36.814. "Evolved Universal Terrestrial Radio Access (E-UTRA). Further advancements for E-UTRA physical layer aspects". Release 15. *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. https://www.3gpp.org.

[2] IEEE 802.11-14/0571r12. "11ax Evaluation Methodology." IEEE P802.11. Wireless LANs. https://www.ieee.org.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`networkTrafficOnOff` | `networkTrafficVoIP` | `networkTrafficVideoConference`

Topics

"Generate and Visualize FTP Application Traffic Pattern"

networkTrafficOnOff

On-Off application traffic pattern generator

Description

The `networkTrafficOnOff` object specifies the configuration parameters to generate an On-Off application traffic pattern.

You can use the On-Off application traffic pattern in 5G, Bluetooth (requires Bluetooth® Toolbox), and WLAN (requires WLAN Toolbox) system-level simulations to accurately model the real-world data traffic.

Creation

Syntax

```
cfgOnOff = networkTrafficOnOff
cfgOnOff = networkTrafficOnOff(Name,Value)
```

Description

`cfgOnOff = networkTrafficOnOff` creates a default On-Off application traffic pattern object.

`cfgOnOff = networkTrafficOnOff(Name,Value)` sets properties on page 3-9 using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'GeneratePacket',true` generates an application packet.

Properties

`OnTime` — On state duration
[ ] (default) | positive scalar

On state duration, specified as a positive scalar. Units are in seconds. To specify a customized value for the On time, specify this property. If you do not specify this property, the object uses the exponential distribution to calculate the On time.

Data Types: `double`

`OffTime` — Off state duration
[ ] (default) | nonnegative scalar

Off state duration, specified as a nonnegative scalar. Units are in seconds. To specify a customized value for the Off time, specify this property. If you do not specify this property, the object uses the exponential distribution to calculate the Off time.

Data Types: `double`

`OnExponentialMean` — Exponential distribution mean value to calculate On state duration
`1` (default) | positive scalar

Exponential distribution mean value to calculate the On state duration, specified as a positive scalar. Units are in seconds.

Dependencies

To enable this property, set the `OnTime` property to `[ ]`.

Data Types: `double`

`OffExponentialMean` — Exponential distribution mean value to calculate Off state duration
2 (default) | nonnegative scalar

Exponential distribution mean value to calculate the Off state duration, specified as a nonnegative scalar. Units are in seconds.

Dependencies

To enable this property, set the `OffTime` property to `[ ]`.

Data Types: `double`

`DataRate` — Packet generation rate during On state
5 (default) | positive scalar

Packet generation rate during the On state, specified as a positive scalar. Units are in Kbps. If the value of this property is low and the `PacketSize` is large, the object might not generate packets in the On state.

Data Types: `double`

`PacketSize` — Length of packet to be generated
`1500` (default) | positive scalar

Length of the packet to be generated, specified as a positive scalar. Units are in bytes. If the value of this property is greater than the `DataRate` property value, the object accumulates the data across multiple On times to generate a packet.

Data Types: `double`

`GeneratePacket` — Flag to indicate whether to generate application packet
`false` or `0` (default) | `true` or `1`

Flag to indicate whether the object generates an application packet, specified as a `logical` `1` (`true`) or `0` (`false`).

Data Types: `logical`

`ApplicationData` — Application data to be added in packet
1500-by-1 vector of 1s (default) | column vector of integers in the range [0, 255]

Application data to be added in the packet, specified as a column vector of integers in the range [0, 255]. If the size of the application data is greater than the `PacketSize` property value, the object truncates the application data. If the size of the application data is less than the `PacketSize` property value, the object appends zeros.

Dependencies

To enable this property, set the `GeneratePacket` property to `1` (`true`).

Data Types: double

Object Functions

Specific to This Object

generate   Generate next FTP, On-Off, VoIP, or video conference application traffic packet

Examples

Generate On-Off Application Traffic Pattern

Create a default On-Off application traffic pattern object.

```
cfgOnOff = networkTrafficOnOff;
```

Generate an On-Off application traffic pattern.

```
[dt,packetSize] = generate(cfgOnOff);
```

Generate On-Off Application Traffic Pattern Using On State Mean Value

Create an On-Off application traffic pattern object, specifying the exponentially distributed mean value of the On state.

```
cfgOnOff = networkTrafficOnOff('OnExponentialMean',5);
```

Generate an On-Off application traffic pattern.

```
[dt,packetSize] = generate(cfgOnOff);
```

Generate On-Off Application Traffic Pattern and Data Packet

Create an On-Off application traffic pattern object to generate an On-Off data packet.

```
cfgOnOff = networkTrafficOnOff('GeneratePacket',true);
```

Generate an On-Off application traffic pattern and data packet.

```
[dt,packetSize,packet] = generate(cfgOnOff);
```

Generate On-Off Application Traffic Pattern to Visualize Packet Sizes and Intervals

Create a default On-Off application traffic pattern object.

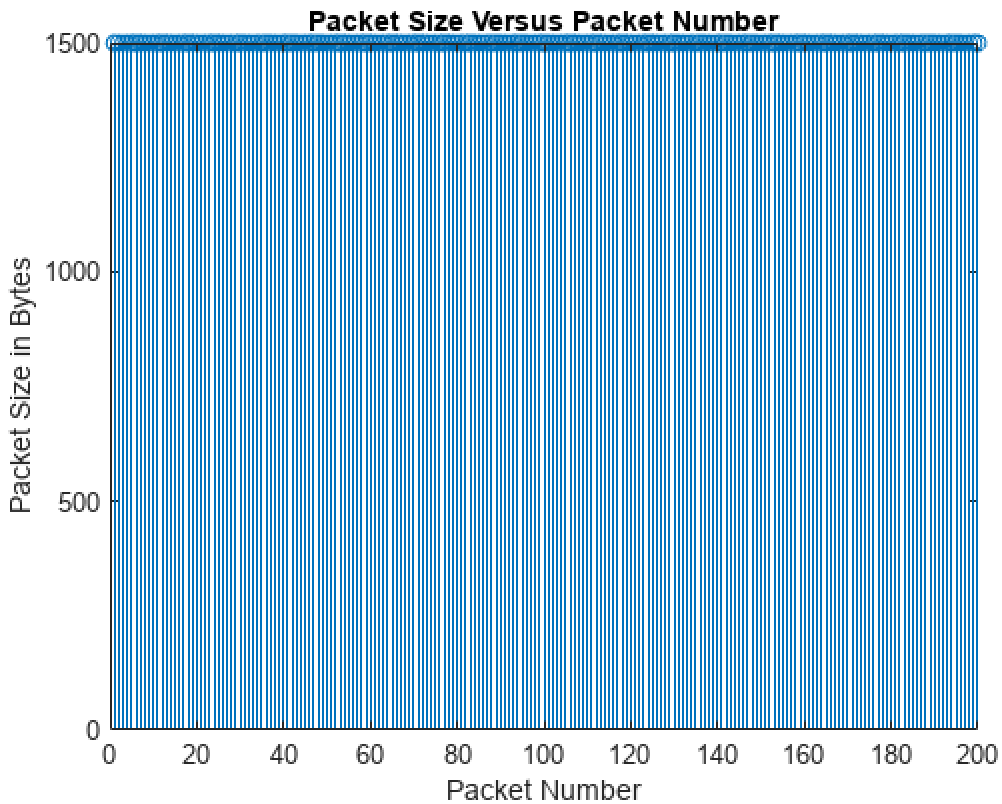
```
cfgOnOff = networkTrafficOnOff;
```

Generate an On-Off application traffic pattern with 200 On-Off packets.

```
for packetCount = 1:200
    [dt(packetCount),packetSize(packetCount)] = generate(cfgOnOff);
end
```

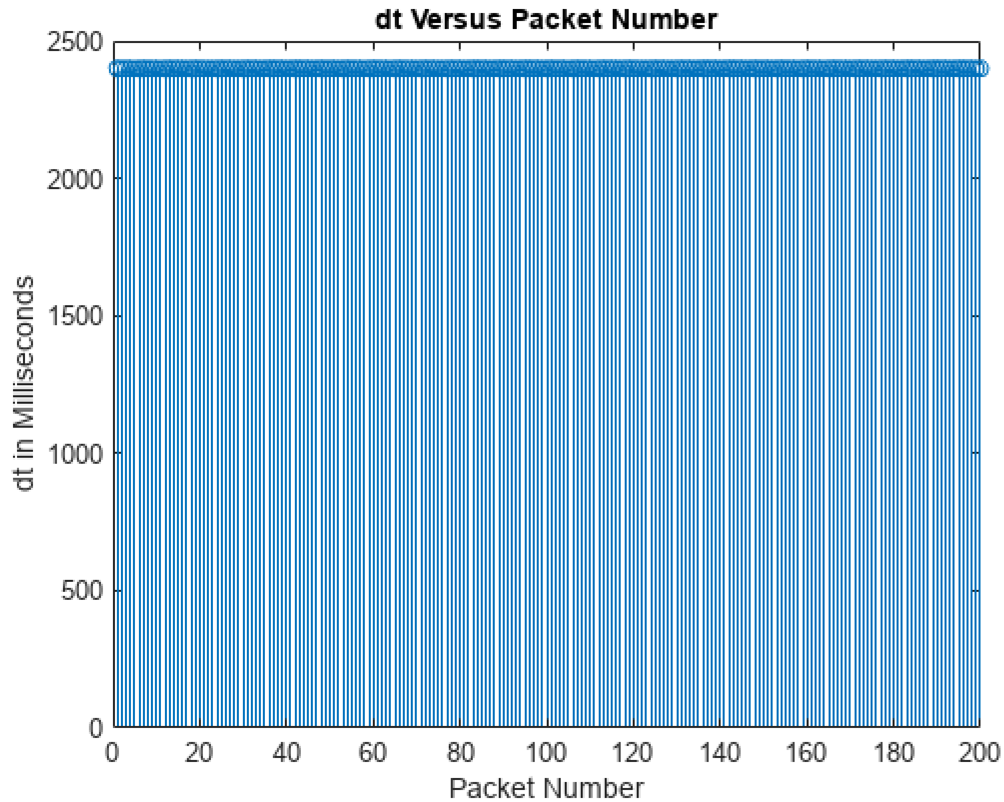
Visualize the On-Off packet sizes.

```
stem(packetSize);
title('Packet Size Versus Packet Number');
xlabel('Packet Number');
ylabel('Packet Size in Bytes');
```



Visualize the On-Off packet intervals.

```
figure;
stem(dt);
title('dt Versus Packet Number');
xlabel('Packet Number');
ylabel('dt in Milliseconds');
```



Version History

Introduced in R2020b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
`networkTrafficFTP` | `networkTrafficVoIP` | `networkTrafficVideoConference`

networkTrafficVoIP

VoIP application traffic pattern generator

Description

The `networkTrafficVoIP` object specifies the configuration parameters to generate a voice over Internet protocol (VoIP) application traffic pattern based on the IEEE 802.11ax Evaluation Methodology.

You can use the VoIP application traffic pattern in 5G and WLAN (requires WLAN Toolbox) system-level simulations to accurately model the real-world data traffic.

Creation

Syntax

```
cfgVoIP = networkTrafficVoIP
cfgVoIP = networkTrafficVoIP(Name,Value)
```

Description

`cfgVoIP = networkTrafficVoIP` creates a default VoIP application traffic pattern object.

`cfgVoIP = networkTrafficVoIP(Name,Value)` sets properties on page 3-14 using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'HasJitter',false` specifies that the VoIP application pattern does not model jitter.

Properties

`ExponentialMean` — Mean value for exponential distribution
`1250` (default) | nonnegative integer

Mean value for the exponential distribution, specified as a nonnegative integer. The object uses this property to calculate the exponentially distributed active or silent state duration in the VoIP traffic.

Data Types: `double`

`HasJitter` — Flag to indicate whether to model jitter
`true` or `1` (default) | `false` or `0`

Flag to indicate whether the object models jitter, specified as a `logical` `1` (`true`) or `0` (`false`).

Data Types: `logical`

`LaplaceScale` — Scale parameter for Laplace distribution
`5.11` (default) | scalar in the range [1, 100]

Scale parameter for the Laplace distribution, specified as a scalar in the range [1, 100]. The object uses this property and the `LaplaceMu` property to calculate the packet arrival delay jitter in milliseconds.

Dependencies

To enable this property, set the `HasJitter` property to `1` (`true`).

Data Types: `double`

`LaplaceMu` — Location parameter for Laplace distribution

`0` (default) | scalar in the range [0, 100]

Location parameter for the Laplace distribution, specified as a scalar in the range [0, 100]. The object uses this property and the `LaplaceScale` property to calculate packet arrival delay jitter in milliseconds.

Dependencies

To enable this property, set the `HasJitter` property to `1` (`true`).

Data Types: `double`

`GeneratePacket` — Flag to indicate whether to generate a VoIP packet

`false` or `0` (default) | `true` or `1`

Flag to indicate whether the object generates a VoIP packet, specified as a `logical` `1` (`true`) or `0` (`false`).

Data Types: `logical`

`ApplicationData` — Application data to be added in VoIP packet

36-element column vector of 1s (default) | column vector of integers in the range [0, 255]

Application data to be added in the VoIP packet, specified as a column vector of integers in the range [0, 255]. If the size of the application data is greater than the packet size, the object truncates the application data. If the size of the application data is less than the packet size, the object appends zeros.

Dependencies

To enable this property, set the `GeneratePacket` property to `1` (`true`).

Data Types: `double`

Object Functions

Specific to This Object

generate	Generate next FTP, On-Off, VoIP, or video conference application traffic packet

Examples

Generate VoIP Application Traffic Pattern

Create a default VoIP application traffic pattern object.

```
cfgVoIP = networkTrafficVoIP;
```

Generate a VoIP application traffic pattern.

```
[dt,packetSize] = generate(cfgVoIP);
```

Generate VoIP Application Traffic Pattern Using Mean Value of Exponential Distribution

Create a VoIP application traffic pattern object, specifying the mean value of the exponential distribution.

```
cfgVoIP = networkTrafficVoIP('ExponentialMean',5);
```

Generate a VoIP application traffic pattern.

```
[dt,packetSize] = generate(cfgVoIP);
```

Generate VoIP Application Traffic Pattern and Data Packet

Create a VoIP application traffic pattern object to generate a VoIP data packet.

```
cfgVoIP = networkTrafficVoIP('GeneratePacket',true);
```

Generate a VoIP application traffic pattern and data packet.

```
[dt,packetSize,packet] = generate(cfgVoIP);
```

Generate VoIP Application Traffic Pattern to Visualize Packet Sizes and Intervals

Create a default VoIP application traffic pattern object.

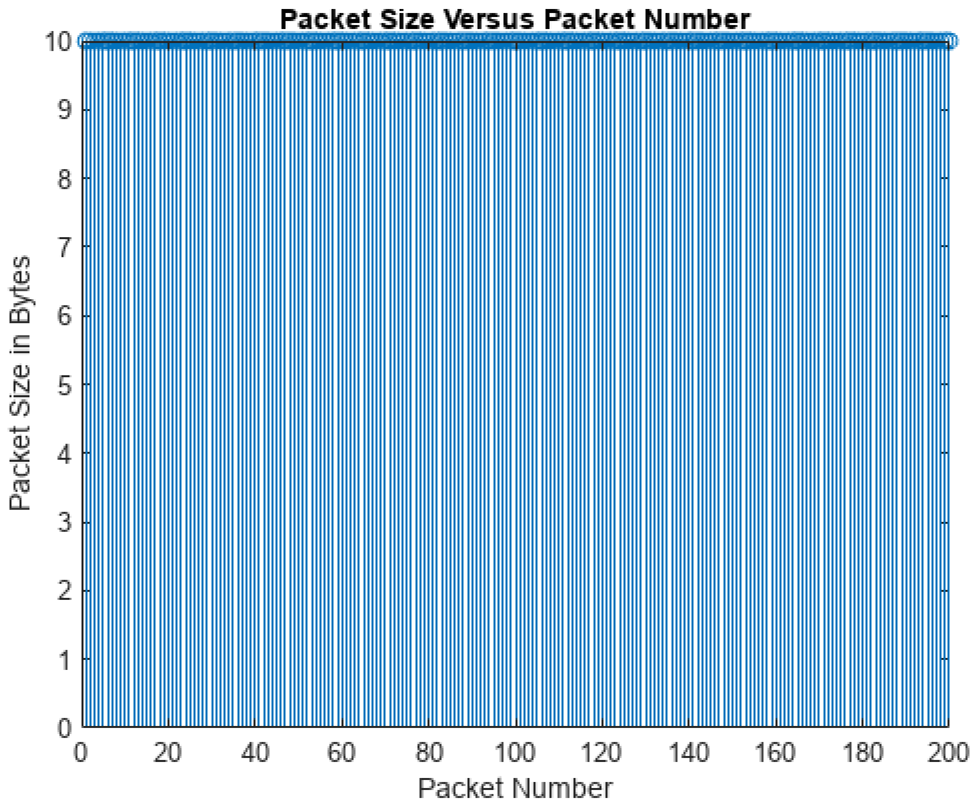
```
cfgVoIP = networkTrafficVoIP;
```

Generate a VoIP application traffic pattern with 200 VoIP packets.

```
for packetCount = 1:200
    [dt(packetCount),packetSize(packetCount)] = generate(cfgVoIP);
end
```

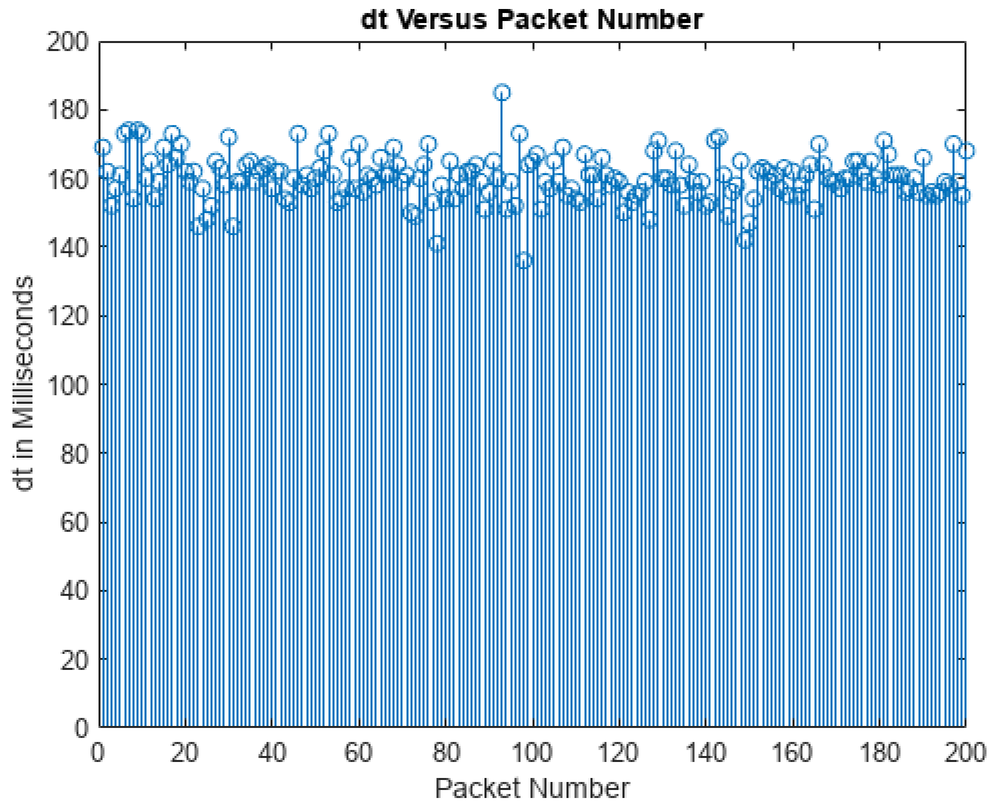
Visualize the VoIP packet sizes.

```
stem(packetSize);
title('Packet Size Versus Packet Number');
xlabel('Packet Number');
ylabel('Packet Size in Bytes');
```



Visualize the VoIP packet intervals.

```
figure;
stem(dt);
title('dt Versus Packet Number');
xlabel('Packet Number');
ylabel('dt in Milliseconds');
```



Version History

Introduced in R2020b

References

[1] IEEE 802.11-14/0571r12. "11ax Evaluation Methodology." IEEE P802.11. Wireless LANs. https://www.ieee.org.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
`networkTrafficFTP` | `networkTrafficOnOff` | `networkTrafficVideoConference`

networkTrafficVideoConference

Video conference application traffic pattern generator

Description

The `networkTrafficVideoConference` object specifies the configuration parameters to generate a video conference application traffic pattern based on the IEEE 802.11ax Evaluation Methodology.

You can use the video conference application traffic pattern in 5G and WLAN (requires WLAN Toolbox) system-level simulations to accurately model real-world data traffic.

Creation

Syntax

```
cfgVideo = networkTrafficVideoConference
cfgVideo = networkTrafficVideoConference(Name,Value)
```

Description

`cfgVideo = networkTrafficVideoConference` creates a default video application traffic pattern object.

`cfgVideo = networkTrafficVideoConference(Name,Value)` sets "Properties" on page 3-19 using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'HasJitter',false` specifies that the video application pattern does not model jitter.

Properties

`FrameInterval` — Time interval between two consecutive video frames
`40` (default) | positive integer

Time interval between two consecutive video frames, specified as a positive integer. This value is expressed in milliseconds.

Data Types: `double`

`FrameSizeMethod` — Option to set source for video frame size
`'WeibullDistribution'` (default) | `'FixedSize'`

Option to set source for video frame size, specified as one of these values.

- `'WeibullDistribution'` — Use the video frame size value calculated by the Weibull probability distribution function.
- `'FixedSize'` — Use the video frame size value specified by the FixedFrameSize property.

Data Types: `char` | `string`

FixedFrameSize — Size of video frame
5000 (default) | integer in the range [1, 15,000]

Size of the video frame, specified as an integer in the range [1, 15,000]. This value is expressed in bytes. The video frame can be segmented into multiple network packets based on this value.

Dependencies

To enable this property, set the FrameSizeMethod property to 'FixedSize'.

Data Types: double

WeibullScale — Scale parameter for Weibull distribution
6950 (default) | scalar in the range (0, 54,210]

Scale parameter for the Weibull distribution to calculate the video frame size, specified as a scalar in the range (0, 54,210].

Dependencies

To enable this property, set the FrameSizeMethod property to 'WeibullDistribution'.

Data Types: double

WeibullShape — Shape parameter for Weibull distribution
0.8099 (default) | scalar in the range (0, 1]

Shape parameter for the Weibull distribution to calculate the video frame size, specified as a scalar in the range (0, 1].

Dependencies

To enable this property, set the FrameSizeMethod property to 'WeibullDistribution'.

Data Types: double

HasJitter — Flag to indicate whether to model network jitter
true or 1 (default) | false or 0

Flag to indicate whether to model network jitter, specified as a logical 1 (true) or 0 (false). The object applies jitter between the segmented packets. If you set this property to 1 (true), the object models jitter using the Gamma probability distribution function.

Data Types: logical

GammaShape — Shape parameter for Gamma distribution
0.2463 (default) | scalar in the range (0, 5]

Shape parameter for the Gamma distribution to calculate network jitter, specified as a scalar in the range (0, 5].

Dependencies

To enable this property, set the HasJitter property to 1 (true).

Data Types: double

GammaScale — Scale parameter for Gamma distribution
6.227 (default) | scalar in the range (0, 10]

Scale parameter for the Gamma distribution to calculate network jitter, specified as a scalar in the range (0, 10].

Dependencies

To enable this property, set the `HasJitter` property to `1` (`true`).

Data Types: `double`

`ProtocolOverhead` — Protocol overheads in network traffic
28 (default) | integer in the range [0, 60]

Protocol overheads in network traffic, specified as an integer in the range [0, 60]. To add layer 3, layer 4, and application protocol overheads in network traffic, enable this property. This value is expressed in bytes.

Data Types: `double`

`GeneratePacket` — Flag to indicate whether to generate video packet with payload
`false` or `0` (default) | `true` or `1`

Flag to indicate whether to generate a video packet with payload, specified as a `logical` `1` (`true`) or `0` (`false`). To generate a video packet with payload, set this property to `1` (`true`). If you set this property to `0` (`false`), the `generate` object function generates no application data packet.

Data Types: `logical`

`ApplicationData` — Application data to be added in video packet
1500-by-1 vector of 1s (default) | column vector of integers in the range [0, 255]

Application data to be added in the video packet, specified as a column vector of integers in the range [0, 255].

- If the size of the application data is greater than the packet size, the object truncates the application data.
- If the size of the application data is less than the packet size, the object appends zeros.

Dependencies

To enable this property, set the GeneratePacket property to `1` (`true`).

Data Types: `double`

Object Functions

Specific to This Object

generate   Generate next FTP, On-Off, VoIP, or video conference application traffic packet

Examples

Generate Video Conference Traffic Using Default Values

Create a default video application traffic pattern object.

```
cfgVideo = networkTrafficVideoConference
```

```
cfgVideo = 
  networkTrafficVideoConference with properties:

       FrameInterval: 40
     FrameSizeMethod: 'WeibullDistribution'
        WeibullScale: 6950
        WeibullShape: 0.8099
           HasJitter: 0
    ProtocolOverhead: 28
      GeneratePacket: 0
```

Generate a video application traffic pattern.

```
[dt,packetSize] = generate(cfgVideo);
```

Generate Video Conference Traffic with Frame Interval and Data Packet

Create a video application traffic pattern object to generate a video data packet. Specify the frame interval in milliseconds.

```
cfgVideo = networkTrafficVideoConference('GeneratePacket',true);
cfgVideo.FrameInterval = 60;
```

Generate a video application traffic pattern and data packet.

```
[dt,packetSize,packet] = generate(cfgVideo);
```

Generate Multiple Video Data Packets Using Elapsed Time

Create a video application traffic pattern object to generate a video frame of size 400 bytes.

```
cfgVideo = networkTrafficVideoConference('FrameSizeMethod','FixedSize','FixedFrameSize',400);
```

Specify the elapsed time in milliseconds.

```
elapsedTime = 10;
```

After every elapsed time value, invoke the video application traffic pattern object to generate five video data packets.

```
for i = 1:5
    while true
        [dt,packetSize] = generate(cfgVideo,elapsedTime);
        if packetSize
            fprintf('Video data packet %d generated\n',i);
            break;
        end
    end
end
```

```
Video data packet 1 generated
Video data packet 2 generated
```

```
Video data packet 3 generated
Video data packet 4 generated
Video data packet 5 generated
```

Generate Video Conference Traffic Pattern to Visualize Packet Sizes and Intervals

Create a default video application traffic pattern object.

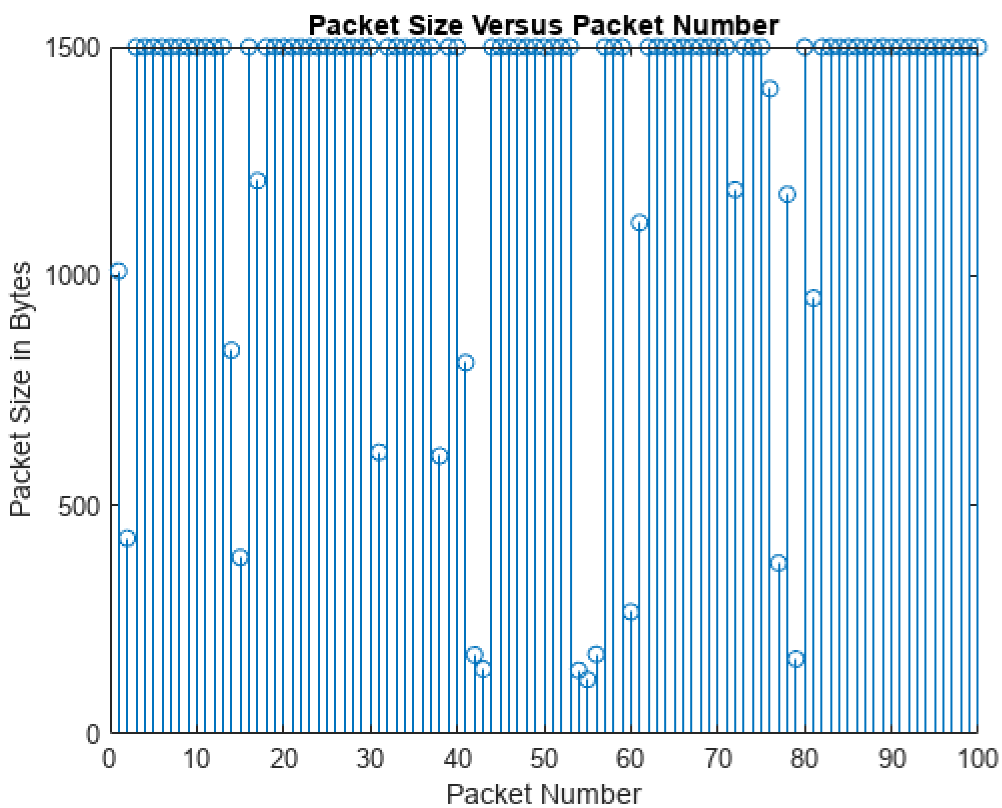
```
cfgVideo = networkTrafficVideoConference;
```

Generate a video application traffic pattern with 100 video packets.

```
for packetCount = 1:100
    [dt(packetCount),packetSize(packetCount)] = generate(cfgVideo);
end
```

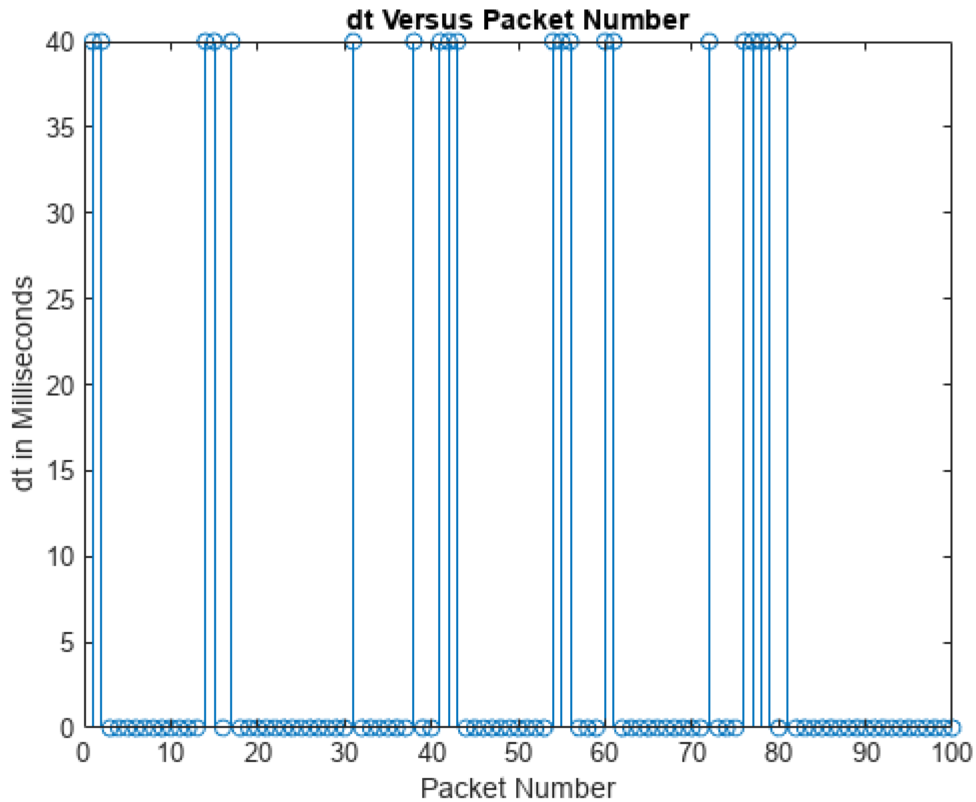
Visualize the video packet sizes.

```
stem(packetSize);
title('Packet Size Versus Packet Number');
xlabel('Packet Number');
ylabel('Packet Size in Bytes');
```



Visualize the video packet intervals.

```
figure;
stem(dt);
title('dt Versus Packet Number');
xlabel('Packet Number');
ylabel('dt in Milliseconds');
```



Version History

Introduced in R2021a

References

[1] IEEE 802.11-14/0571r12. "11ax Evaluation Methodology." IEEE P802.11. Wireless LANs. https://www.ieee.org.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects
`networkTrafficFTP` | `networkTrafficOnOff` | `networkTrafficVoIP`

nrCarrierConfig

Carrier configuration parameters

Description

The `nrCarrierConfig` object sets carrier configuration parameters for a specific OFDM numerology, as defined in TS 38.211 Sections 4.2, 4.3, and 4.4 [1].

The object defines the carrier subcarrier spacing, bandwidth, and offset parameters from point *A*, the center of subcarrier 0 in the common resource block 0 (CRB 0). For a 60 kHz subcarrier spacing, you can specify either normal or extended cyclic prefix. The read-only properties of this object provide the carrier resource grid time-domain dimensions. By default, the object specifies a 10 MHz carrier corresponding to 52 resource blocks (RBs) and 15 kHz subcarrier spacing. You can use the object in slot-oriented processing by specifying the current slot and frame numbers.

Creation

Syntax

```
carrier = nrCarrierConfig
carrier = nrCarrierConfig(Name,Value)
```

Description

`carrier = nrCarrierConfig` creates a carrier configuration object with default properties.

`carrier = nrCarrierConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'SubcarrierSpacing',30,'NSizeGrid',273` specifies a 100 MHz carrier corresponding to 273 RBs and 30 kHz subcarrier spacing. Unspecified properties take their default values.

Properties

`NCellID` — Physical layer cell identity
`1` (default) | integer from 0 to 1007

Physical layer cell identity, specified as an integer from 0 to 1007.

Data Types: `double`

`SubcarrierSpacing` — Subcarrier spacing in kHz
`15` (default) | `30` | `60` | `120` | `240`

Subcarrier spacing in kHz, for all channels and reference signals of the carrier, specified as `15`, `30`, `60`, `120`, or `240`.

Data Types: `double`

`CyclicPrefix` — Cyclic prefix length
`'normal'` (default) | `'extended'`

Cyclic prefix length, specified as one of these options.

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies for only 60 kHz subcarrier spacing.

Data Types: `char` | `string`

`NSizeGrid` — Number of RBs in carrier resource grid
`52` (default) | integer from 1 to 275

Number of RBs in the carrier resource grid, specified as an integer from 1 to 275. The default value of `52` corresponds to the maximum number of RBs of a 10 MHz carrier with 15 kHz SCS.

Data Types: `double`

`NStartGrid` — Start of carrier resource grid relative to CRB 0
`0` (default) | integer from 0 to 2199

Start of carrier resource grid relative to CRB 0, specified as an integer from 0 to 2199. This property is the higher-layer parameter *offsetToCarrier*.

Data Types: `double`

`NSlot` — Slot number
`0` (default) | nonnegative integer

Slot number, specified as a nonnegative integer. You can set `NSlot` to a value larger than the number of slots per frame. For example, you can set this value using transmission loop counters in a MATLAB simulation. In this case, you may have to ensure that the property value is modulo the number of slots per frame in a calling code.

Data Types: `double`

`NFrame` — System frame number
`0` (default) | nonnegative integer

System frame number, specified as a nonnegative integer. You can set `NFrame` to a value larger than the maximum frame number 1023. For example, you can set this value using transmission loop counters in a MATLAB simulation. In this case, you may have to ensure that the property value is modulo 1024 in a calling code.

Data Types: `double`

`SymbolsPerSlot` — Number of OFDM symbols per slot
`14` (default) | `12`

This property is read-only.

Number of OFDM symbols per slot, specified as `14` for normal cyclic prefix or `12` for extended cyclic prefix. The object sets this property based on the `CyclicPrefix` property.

Data Types: double

SlotsPerSubframe — Number of slots per 1 ms subframe
1 (default) | 2 | 4 | 8 | 16

This property is read-only.

Number of slots per 1 ms subframe, specified as 1, 2, 4, 8, or 16. The object sets this property based on the SubcarrierSpacing property values 15, 30, 60, 120, and 240, respectively.

Data Types: double

SlotsPerFrame — Number of slots per 10 ms frame
10 (default) | 20 | 40 | 80 | 160

This property is read-only.

Number of slots per 10 ms frame, specified as 10, 20, 40, 80, or 160. The object sets this property based on the SubcarrierSpacing property values 15, 30, 60, 120, and 240, respectively.

Data Types: double

Examples

Generate CSI-RS Symbols and Indices for 10 MHz Carrier

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier.

```
carrier = nrCarrierConfig;
```

Create a CSI-RS configuration object with default properties.

```
csirs = nrCSIRSConfig;
```

Generate CSI-RS symbols of single data type.

```
[sym,info_sym] = nrCSIRS(carrier,csirs,'OutputDataType','single');
```

Generate resource element indices for CSI-RS.

```
[ind,info_ind] = nrCSIRSIndices(carrier,csirs);
```

Generate ZP and NZP-CSI-RS Symbols and Indices

Create a carrier configuration object, specifying the slot number as 10.

```
carrier = nrCarrierConfig('NSlot',10);
```

Create a CSI-RS resource configuration object for two periodic resources. Specify one NZP resource and one ZP resource with row numbers 3 and 5, symbol locations 13 and 9, and subcarrier locations 6 and 4, respectively. For both resources, set the periodicity to 5, offset to 1, and density to 'one'.

```
csirs = nrCSIRSConfig;
csirs.CSIRSType = {'nzp','zp'};
csirs.CSIRSPeriod = {[5 1],[5 1]};
csirs.RowNumber = [3 5];
csirs.Density = {'one','one'};
csirs.SymbolLocations = {13,9};
csirs.SubcarrierLocations = {6,4};
```

Generate CSI-RS symbols and indices for the specified carrier, CSI-RS resource configuration, and output formatting name-value pair arguments. Verify the format of the symbols and indices.

```
[sym,info_sym] = nrCSIRS(carrier,csirs,...
                'OutputResourceFormat','cell')
```

```
sym=1×2 cell array
    {0x1 double}    {0x1 double}


info_sym = struct with fields:
      ResourceOrder: [2 1]
           KBarLBar: {{1x1 cell}  {1x2 cell}}
    CDMGroupIndices: {[0]  [0 1]}
             KPrime: {[0 1]  [0 1]}
             LPrime: {[0]  [0]}
```

```
[ind,info_ind] = nrCSIRSIndices(carrier,csirs,...
                'IndexStyle','subscript','OutputResourceFormat','cell')
```

```
ind=1×2 cell array
    {0x3 uint32}    {0x3 uint32}


info_ind = struct with fields:
      ResourceOrder: [2 1]
           KBarLBar: {{1x1 cell}  {1x2 cell}}
    CDMGroupIndices: {[0]  [0 1]}
             KPrime: {[0 1]  [0 1]}
             LPrime: {[0]  [0]}
```

Verify that the generated outputs are in the order of ZP-CSI-RS resources followed by NZP-CSI-RS resources in terms of the specified `csirs.CSIRSType` indices.

```
info_sym.ResourceOrder
```

```
ans = 1×2

     2     1
```

```
info_ind.ResourceOrder
```

```
ans = 1×2

     2     1
```

Version History

Introduced in R2019b

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrResourceGrid` | `nrOFDMModulate` | `nrOFDMDemodulate` | `nrOFDMInfo`

nrCORESETConfig

Control resource set (CORESET) configuration parameters

Description

The `nrCORESETConfig` object sets CORESET configuration parameters for the physical downlink control channel (PDCCH), as defined in TS 38.211 Section 7.3.2 [1]. Use this object when setting the `CORESET` property of the `nrPDCCHConfig` or `nrDLCarrierConfig` objects.

Creation

Syntax

```
crst = nrCORESETConfig
crst = nrCORESETConfig(Name,Value)
```

Description

`crst = nrCORESETConfig` creates a CORESET configuration object with default properties.

`crst = nrCORESETConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'REGBundleSize',3,'Duration',3` configures the CORESET with a duration of 3 OFDM symbols and resource element group (REG) bundle size of 3. Unspecified properties take their default values.

Properties

`CORESETID` — CORESET ID
`1` (default) | integer from 0 to 11

CORESET ID, specified as an integer from 0 to 11. When this object and `nrSearchSpaceConfig` object specify the `SearchSpace` and `CORESET` properties, respectively, of the same `nrPDCCHConfig` object, the `CORESETID` properties of these objects must match.

The reference point for the demodulation reference signal (DM-RS) sequence-to-subcarrier resource mapping for CORESET ID 0 is the lowest physical resource block of the CORESET. All other CORESET ID values use the common resource block 0 for the DM-RS reference point.

Data Types: `double`

`Label` — Name of CORESET configuration
`'CORESET1'` (default) | character array | string scalar

Name of CORESET configuration, specified as a character array or string scalar. Use this property to set a description to the CORESET configuration.

Data Types: `char` | `string`

FrequencyResources — Frequency-domain resources
[1 1 1 1 1 1 1 1] (default) | binary row vector

Frequency-domain resources, specified as a binary row vector. An element value of 1 indicates an allocated frequency resource of six resource blocks (RBs). An element value of 0 indicates no allocation. The maximum number of vector elements is 45. Grouping starts from the first RB group in the bandwidth part (BWP). The first vector element corresponds to the first RB group in the BWP.

This property determines the total number of RBs allocated in the frequency domain, which is given by $numRBs = 6 \times$ sum(FrequencyResources).

Data Types: double

Duration — CORESET duration
2 (default) | 1 | 3

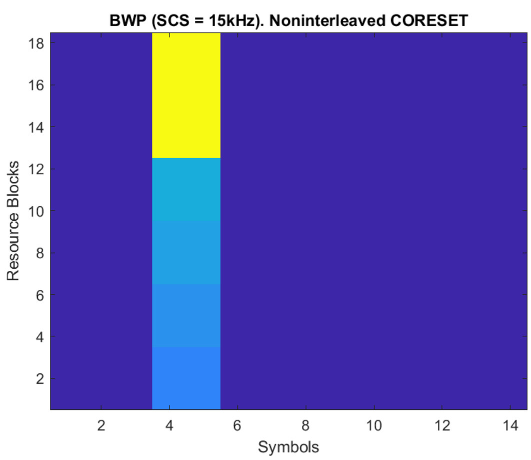
CORESET duration, in OFDM symbols, specified as 1, 2, or 3.

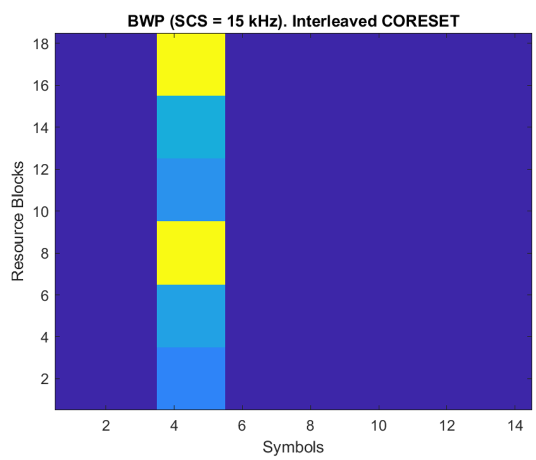
Data Types: double

CCEREGMapping — CCE-to-REG mapping
'interleaved' (default) | 'noninterleaved'

Control channel elements (CCE) to REG mapping, specified as 'interleaved' or 'noninterleaved'.

These diagrams illustrate the difference between noninterleaved and interleaved CORESETs. Both CORESETS are configured with the Duration property set to 2 and the REGBundleSize property set to 6. The InterleaverSize property of the interleaved CORESET is set to 2.





Data Types: char | string

REGBundleSize — Size of REG bundles
6 (default) | 2 | 3

Size of REG bundles, specified as 2, 3, or 6.

- When the Duration property is set to 3, set REGBundleSize to 3 or 6.
- When Duration is set to 1 or 2, set REGBundleSize to 2 or 6.

The number of REGs, *numREGs*, depends on the total number of resource blocks, *numRBs*, allocated in the frequency domain and is given by

numREGs = *numRBs* × `Duration`, where *numRBs* = 6 × `sum(FrequencyResources)`.

When the `CCEREGMapping` property is set to `'interleaved'`, *numREGs* must be a multiple of `REGBundleSize` × `InterleaverSize`.

Dependencies

To enable this property, set the `CCEREGMapping` property to `'interleaved'`.

Data Types: `double`

`InterleaverSize` — Interleaver size

2 (default) | 3 | 6

Interleaver size for interleaved CCE-to-REG mapping, specified as `2`, `3`, or `6`.

The number of REGs, *numREGs*, depends on the total number of resource blocks, *numRBs*, allocated in the frequency domain and is given by

numREGs = *numRBs* × `Duration`, where *numRBs* = 6 × `sum(FrequencyResources)`.

When the `CCEREGMapping` property is set to `'interleaved'`, *numREGs* must be a multiple of `REGBundleSize` × `InterleaverSize`.

Dependencies

To enable this property, set the `CCEREGMapping` property to `'interleaved'`.

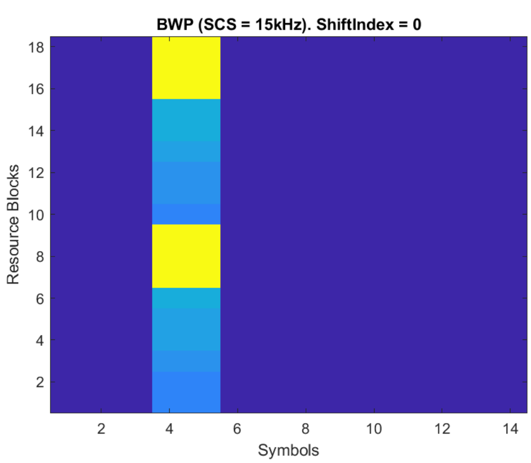
Data Types: `double`

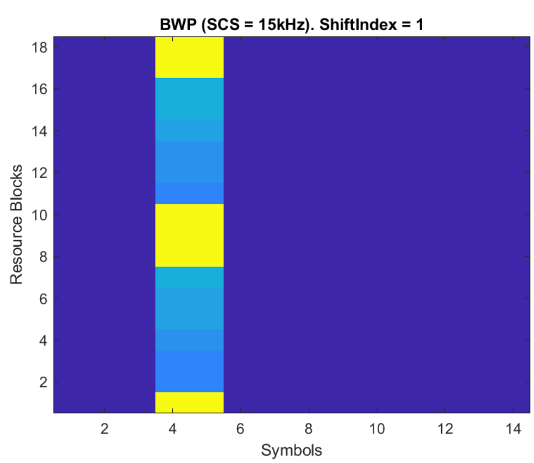
`ShiftIndex` — Shift index

`0` (default) | integer from 0 to 274 | *NCellID* with integer value from 0 to 1007

Shift index, specified as an integer from 0 to 274 or the physical layer cell identity *NCellID* with integer value from 0 to 1007.

For example, these diagrams illustrate the effect of shift index on CORESET mapping. Both CORESETs are configured with the `Duration` property set to `2`, the `REGBundleSize` property set to `2`, and the `InterleaverSize` property set to 2.





Dependencies

To enable this property, set the `CCEREGMapping` property to `'interleaved'`.

Data Types: `double`

`PrecoderGranularity` — Precoder granularity
`'sameAsREG-bundle'` (default) | `'allContiguousRBs'`

Precoder granularity associated with the CORESET, as defined in TS 38.211 Sections 7.3.2.2 and 7.4.1.3.2., specified as one of these values.

- `'sameAsREG-bundle'` — The precoding on associated PDCCH transmissions is the same within a REG bundle. In this case, the PDCCH DM-RS transmission is across the PRBs associated with the PDCCH.
- `'allContiguousRBs'` — The precoding is the same across all REGs within the set of contiguous RBs in the CORESET. In this case, the PDCCH DM-RS transmission is across the entire CORESET region.

Data Types: `char` | `string`

`RBOffset` — RB offset of CORESET in BWP
`[]` (default) | integer from 0 to 5

RB offset of the start of the CORESET from the start of the BWP, specified as one these values.

- `[]` — CORESET frequency resources start at the first complete group of six common resource blocks (CRBs) in the BWP.
- Integer from 0 to 5 — Specify the RB offset explicitly. This option corresponds to the higher-layer parameter *rb-Offset-r16*.

Data Types: `char` | `string`

`NCCE` — Number of CCE available in CORESET
positive integer

This property is read-only.

Number of CCE available for use in the CORESET, as defined in TS 38.213 Section 10.1 [2], returned as a positive integer. The value depends on the `FrequencyResources` and `Duration` property values.

Data Types: `double`

Examples

Generate PDCCH DM-RS Symbols and Indices

Configure the carrier with default configuration parameters.

```
carrier = nrCarrierConfig;
```

Configure the CORESET with 6 frequency resources, a duration of 3 OFDM symbols, and a REG bundle size of 3.

```
crst = nrCORESETConfig;
crst.FrequencyResources = ones(1,6);
crst.Duration = 3;
crst.REGBundleSize = 3;
```

Configure the PDCCH with the specified bandwidth part and CORESET.

```
pdcch = nrPDCCHConfig;
pdcch.NStartBWP = 6;
pdcch.NSizeBWP = 36;
pdcch.CORESET = crst;
pdcch.AggregationLevel = 16;
```

Generate PDCCH DM-RS symbols and indices for the specified carrier and PDCCH.

```
[~,dmrs,dmrsInd] = nrPDCCHResources(carrier,pdcch);
```

Generate PDCCH and DM-RS Indices Relative to BWP Grid

Configure a carrier grid of 60 resource blocks (RBs), where the starting RB index relative to the common resource block 0 (CRB 0) is 3.

```
carrier = nrCarrierConfig;
carrier.NStartGrid = 3;
carrier.NSizeGrid = 60;
```

Configure noninterleaved CORESET with 6 frequency resources and a duration of 3 OFDM symbols.

```
crst = nrCORESETConfig;
crst.FrequencyResources = ones(1,6);
crst.Duration = 3;
crst.CCEREGMapping = 'noninterleaved';
```

Configure the PDCCH with the specified bandwidth part and CORESET.

```
pdcch = nrPDCCHConfig;
pdcch.NStartBWP = 5;
pdcch.NSizeBWP = 48;
pdcch.CORESET = crst;
pdcch.AggregationLevel = 16;
```

Generate PDCCH resource element indices and DM-RS symbol indices using 1-based, subscript indexing form relative to the BWP grid.

```
[ind,~,dmrsInd] = nrPDCCHResources(carrier,pdcch,...
    'IndexOrientation','bwp','IndexStyle','subscript');
```

Version History

Introduced in R2020a

Specify precoder granularity

The `PrecoderGranularity` property enables you to specify the precoder granularity of the associated PDCCH, as defined in TS 38.211 Sections 7.3.2.2 and 7.4.1.3.2.

Specify RB offset

The `RBOffset` property enables you to specify the RB offset of the CORESET in the BWP.

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.213. "NR; Physical layer procedures for control." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrPDCCH` | `nrPDCCHResources` | `nrPDCCHSpace`

Objects
`nrSearchSpaceConfig` | `nrPDCCHConfig` | `nrDLCarrierConfig` | `nrCarrierConfig`

nrCSIRSConfig

CSI-RS configuration parameters

Description

The `nrCSIRSConfig` object sets channel state information reference signal (CSI-RS) configuration parameters for one or more zero-power (ZP) or non-zero-power (NZP) CSI-RS resources, as defined in TS 38.211 Section 7.4.1.5 [1].

Creation

Syntax

```
csirs = nrCSIRSConfig
csirs = nrCSIRSConfig(Name,Value)
```

Description

`csirs = nrCSIRSConfig` creates a CSI-RS configuration object with default properties.

`csirs = nrCSIRSConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'CSIRSType',{'zp','nzp','zp'},'Density',{'one','dot5odd','three'},'SubcarrierLocations',{0,4,[0 4]}` specifies three CSI-RS resources with different frequency density values and different frequency-domain locations. Unspecified properties take their default values.

Properties

`CSIRSType` — Type of one or more CSI-RS resource configurations
`'nzp'` (default) | `'zp'` | cell array | string scalar | string array

Type of one or more CSI-RS resource configurations, specified as one of these options.

- `'nzp'` — Use this option to specify a single NZP-CSI-RS resource.
- `'zp'` — Use this option to specify a single ZP-CSI-RS resource.
- Cell array with elements `'nzp'` or `'zp'` — Use this option to specify multiple CSI-RS resources.

Alternatively, you can specify this property by using `"nzp"` and `"zp"` as string scalars or as elements of a string array.

The number of CSI-RS resource configurations is equal to the number of values provided for this property.

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

`CSIRSPeriod` — Slot periodicity and offset of CSI-RS resource
`'on'` (default) | `'off'` | vector of integers | cell array | string scalar | string array

Slot periodicity and offset of the CSI-RS resource, specified as one of these options.

For Single CSI-RS Resource

- `'on'` — Use this option to indicate that the resource is present in all slots.
- `'off'` — Use this option to indicate that the resource is absent in all slots.
- Vector of integers of the form [*Tcsi-rs Toffset*] — Use this option to specify slot periodicity *Tcsi-rs* and offset *Toffset* for scheduling the CSI-RS resource in specific slots.

  Tcsi-rs is 4, 5, 8, 10, 16, 20, 32, 40, 64, 80, 160, 320, or 640. For a particular value of *Tcsi-rs*, the value of *Toffset* is in the range from 0 to *Tcsi-rs*–1.

For Multiple CSI-RS Resources

- Cell array with elements `'on'`, `'off'`, or [*Tcsi-rs Toffset*] — The number of elements in the cell array must be one or equal the number of CSI-RS resources specified by the `CSIRSType` property. When the cell array contains only one element, the specified value applies to all CSI-RS resources.

Alternatively, you can specify this property by using `"on"` and `"off"` as string scalars or as elements of a string array.

This property is the higher-layer parameter *CSI-ResourcePeriodicityAndOffset* or *slotConfig* defined in the *CSI-RS-CellMobility* IE.

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

`RowNumber` — Row number of CSI-RS resource
3 (default) | integer from 1 to 18 | vector of integers

Row number of CSI-RS resource, as defined in TS 38.211 Table 7.4.1.5.3-1, specified as one of these options.

For Single CSI-RS Resource

- Integer from 1 to 18

For Multiple CSI-RS Resources

- Vector of integers in the range from 1 to 18 — The number of vector elements must equal the number of CSI-RS resources specified by the `CSIRSType` property.

Data Types: `double`

`Density` — Frequency density of CSI-RS resource
`'one'` (default) | `'three'` | `'dot5even'` | `'dot5odd'` | cell array | string scalar | string array

Frequency density of the CSI-RS resource, as defined in TS 38.211 Table 7.4.1.5.3-1, specified as one of these options.

For Single CSI-RS Resource

- `'one'` — This option corresponds to $\rho = 1$ from the specified table.
- `'three'` — This option corresponds to $\rho = 3$ from the specified table.
- `'dot5even'` — This option corresponds to $\rho = 0.5$ from the specified table with even resource block (RB) allocation regarding the common resource block 0 (CRB 0).

- `'dot5odd'` — This option corresponds to ρ = 0.5 from the specified table with odd RB allocation regarding CRB 0.

For Multiple CSI-RS Resources

- Cell array of the character vectors `'one'`, `'three'`, `'dot5even'`, or `'dot5odd'` — The number of elements in the cell array must equal the number of CSI-RS resources specified by the `CSIRSType` property.

Alternatively, you can specify this property by using `"one"`, `"three"`, `"dot5even"`, and `"dot5odd"` as string scalars or as elements of a string array.

The value of ρ is the higher-layer parameter *density* defined in the *CSI-RS-ResourceMapping* IE or the *CSI-RS-CellMobility* IE.

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

`SymbolLocations` — Time-domain locations of CSI-RS resource

`0` (default) | integer from 0 to 13 | vector of integers | cell array

Time-domain locations of the CSI-RS resource (l_0 and l_1 values in the TS 38.211 Table 7.4.1.5.3-1), specified as one of these options.

For Single CSI-RS Resource

- Integer from 0 to 13 — This option corresponds to the l_0 value in the specified table.
- Vector of integers of the form [l_0 l_1] or [l_0; l_1], where l_0 and l_1 are the corresponding l_0 and l_1 values in the specified table — The l_1 values are required only in table rows 13, 14, 16, and 17. l_0 is an integer from 0 to 13, and l_1 is an integer from 2 to 12.

For Multiple CSI-RS Resources

- Cell array of l_0 values or vectors of the form [l_0 l_1] or [l_0; l_1] — The number of elements in the cell array must equal the number of CSI-RS resources specified by the `CSIRSType` property.

The values of l_0 and l_1 are the higher-layer parameters *firstOFDMSymbolInTimeDomain* and *firstOFDMSymbolInTimeDomain2*, respectively, in the *CSI-RS-ResourceMapping* IE or *CSI-RS-Resource-Mobility* IE.

Data Types: `double`

`SubcarrierLocations` — Frequency-domain locations of CSI-RS resource

`0` (default) | numeric vector | cell array

Frequency-domain locations of the CSI-RS resource (k_i values in the TS 38.211 Table 7.4.1.5.3-1), specified as one of these options.

For Single CSI-RS Resource

- Numeric vector with elements 1, 2, 3, 4, or 6 — The vector elements correspond to the possible lengths of subcarrier locations.

For Multiple CSI-RS Resources

- Cell array of numeric vectors with elements 1, 2, 3, 4, or 6 — The number of elements in the cell array must equal the number of CSI-RS resources specified by the `CSIRSType` property.

Data Types: `double`

`NumRB` — CSI-RS resource bandwidth
`52` (default) | integer from 1 to 275 | vector of integers

CSI-RS resource bandwidth, in terms of the number of allocated RBs, specified as one of these options.

For Single CSI-RS Resource

- Integer from 1 to 275

For Multiple CSI-RS Resources

- Vector of integers in the range from 1 to 275 — The number of vector elements must equal to one or the number of CSI-RS resources specified by the `CSIRSType` property. When the vector contains only one element, the specified value applies to all CSI-RS resources.

This property is the higher-layer parameter *nrOfRBs* in *FrequencyOccupation* IE or *nrOfPRBs* in *CSI-RS-ResourceConfigMobility* IE.

Data Types: `double`

`RBOffset` — Starting RB index of CSI-RS resource allocation
`0` (default) | integer from 0 to 274 | vector of integers

Starting RB index of the CSI-RS resource allocation, relative to the carrier resource grid, specified as one of these options.

For Single CSI-RS Resource

- Integer from 0 to 274

For Multiple CSI-RS Resources

- Vector of integers in the range from 0 to 274 — The number of vector elements must be one or equal the number of CSI-RS resources specified by the `CSIRSType` property. When the vector contains only one element, the specified value applies to all CSI-RS resources.

Data Types: `double`

`NID` — Scrambling identity
`0` (default) | integer from 0 to 1023 | vector of integers

Scrambling identity, specified as one of these options.

For Single CSI-RS Resource

- Integer from 0 to 1023

For Multiple CSI-RS Resources

- Vector of integers in the range from 0 to 1023 — The number of vector elements must be one or equal the number of CSI-RS resources specified by the `CSIRSType` property. When the vector contains only one element, the specified value applies to all CSI-RS resources.

This property is the higher-layer parameter *scramblingID* in *NZP-CSI-RS-Resource* IE or *sequenceGenerationConfig* in *CSI-RS-ResourceConfigMobility* IE.

When the `CSIRSType` property defines only ZP resources, this property is hidden.

Data Types: `double`

`NumCSIRSPorts` — Number of CSI-RS antenna ports
`2` (default) | `1` | `4` | `8` | `12` | `16` | `24` | `32` | vector of integers

This property is read-only.

Number of CSI-RS antenna ports, specified as `1`, `2`, `4`, `8`, `12`, `16`, `24`, `32`, or a vector of integers from this list. The object sets this property based on the `RowNumber` property.

Data Types: `double`

`CDMType` — CDM type of CSI-RS resource
`'FD-CDM2'` (default) | `'noCDM'` | `'CDM4'` | `'CDM8'` | cell array

This property is read-only.

CDM type of CSI-RS resource, specified as `'noCDM'`, `'FD-CDM2'`, `'CDM4'`, `'CDM8'`, or a cell array of character vectors from this list. The object sets this property based on the `RowNumber` property.

Data Types: `char`

Examples

Generate ZP and NZP-CSI-RS Symbols and Indices

Create a carrier configuration object, specifying the slot number as `10`.

```
carrier = nrCarrierConfig('NSlot',10);
```

Create a CSI-RS resource configuration object for two periodic resources. Specify one NZP resource and one ZP resource with row numbers `3` and `5`, symbol locations `13` and `9`, and subcarrier locations `6` and `4`, respectively. For both resources, set the periodicity to `5`, offset to `1`, and density to `'one'`.

```
csirs = nrCSIRSConfig;
csirs.CSIRSType = {'nzp','zp'};
csirs.CSIRSPeriod = {[5 1],[5 1]};
csirs.RowNumber = [3 5];
csirs.Density = {'one','one'};
csirs.SymbolLocations = {13,9};
csirs.SubcarrierLocations = {6,4};
```

Generate CSI-RS symbols and indices for the specified carrier, CSI-RS resource configuration, and output formatting name-value pair arguments. Verify the format of the symbols and indices.

```
[sym,info_sym] = nrCSIRS(carrier,csirs,...
                'OutputResourceFormat','cell')
```

```
sym=1×2 cell array
    {0x1 double}    {0x1 double}
```

```
info_sym = struct with fields:
      ResourceOrder: [2 1]
           KBarLBar: {{1x1 cell}  {1x2 cell}}
    CDMGroupIndices: {[0]  [0 1]}
             KPrime: {[0 1]  [0 1]}
             LPrime: {[0]  [0]}
```

```
[ind,info_ind] = nrCSIRSIndices(carrier,csirs,...
                'IndexStyle','subscript','OutputResourceFormat','cell')
```

```
ind=1×2 cell array
    {0x3 uint32}    {0x3 uint32}
```

```
info_ind = struct with fields:
      ResourceOrder: [2 1]
           KBarLBar: {{1x1 cell}  {1x2 cell}}
    CDMGroupIndices: {[0]  [0 1]}
             KPrime: {[0 1]  [0 1]}
             LPrime: {[0]  [0]}
```

Verify that the generated outputs are in the order of ZP-CSI-RS resources followed by NZP-CSI-RS resources in terms of the specified `csirs.CSIRSType` indices.

```
info_sym.ResourceOrder
```

```
ans = 1×2

     2     1
```

```
info_ind.ResourceOrder
```

```
ans = 1×2

     2     1
```

Generate and Map CSI-RS Symbols Used for Tracking

Create a carrier configuration object with default properties.

```
carrier = nrCarrierConfig;
```

Create a CSI-RS resource configuration object with CSI-RS parameters set for tracking. Specify four periodic NZP-CSI-RS resources in two consecutive slots. Specify for each slot to contain two periodic NZP-CSI-RS resources with periodicity set to `20`. Set the offset for the first two resources to `0`. Set the offset for the next two resources to `1`. Set the row number to `1` and density to `'three'` for all resources.

```
csirs = nrCSIRSConfig;
csirs.CSIRSType = {'nzp','nzp','nzp','nzp'};
csirs.CSIRSPeriod = {[20 0],[20 0],[20 1],[20 1]};
csirs.RowNumber = [1 1 1 1];
```

```
csirs.Density = {'three','three','three','three'};
csirs.SymbolLocations = {6,10,6,10};
csirs.SubcarrierLocations = {0,0,0,0};
```

Generate CSI-RS symbols and indices for the default slot number of the carrier configuration object (slot number 0).

```
ind0 = nrCSIRSIndices(carrier,csirs);
sym0 = nrCSIRS(carrier,csirs);
```

Map the symbols to a carrier grid of one slot duration.

```
gridSize = [12*carrier.NSizeGrid carrier.SymbolsPerSlot max(csirs.NumCSIRSPorts)];
slotgrid0 = complex(zeros(gridSize));
slotgrid0(ind0) = sym0;
```

Change the absolute slot number in the carrier configuration from `0` to `1`.

```
carrier.NSlot = 1;
```

Generate CSI-RS symbols and indices for slot number 1.

```
ind1 = nrCSIRSIndices(carrier,csirs);
sym1 = nrCSIRS(carrier,csirs);
```

Map the symbols to another carrier grid of one slot duration.

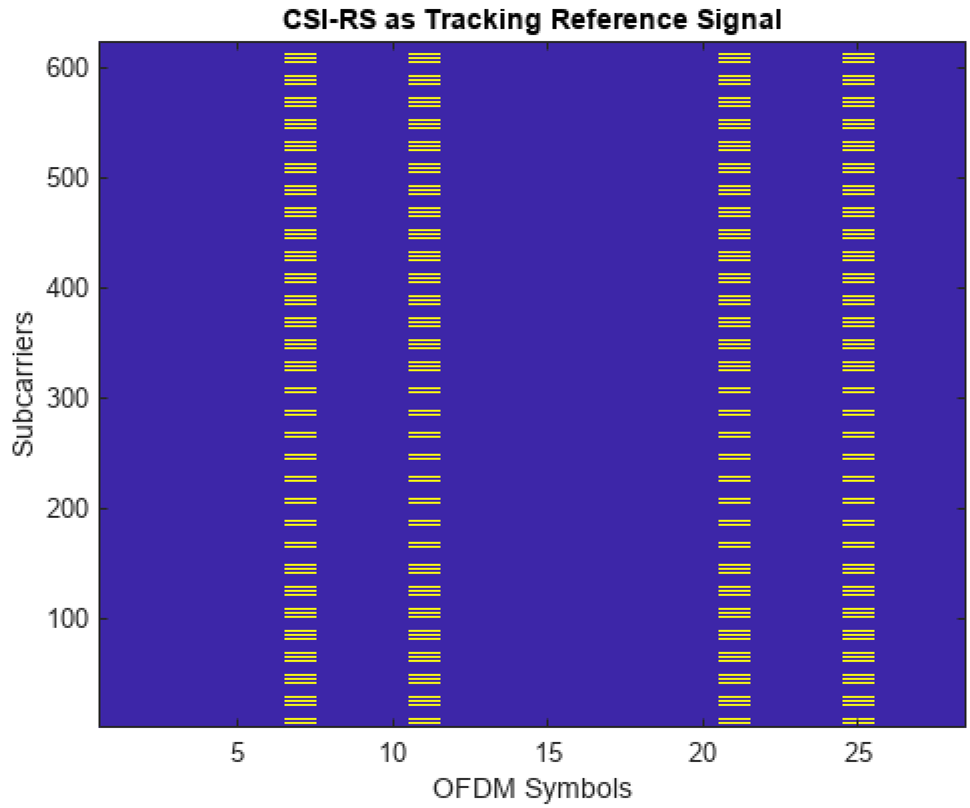
```
slotgrid1 = complex(zeros(gridSize));
slotgrid1(ind1) = sym1;
```

Concatenate the two slots to form the final grid.

```
grid = [slotgrid0 slotgrid1];
```

Plot the grid.

```
imagesc(abs(grid(:,:,1)));
axis xy;
title('CSI-RS as Tracking Reference Signal');
xlabel('OFDM Symbols');
ylabel('Subcarriers');
```



Version History

Introduced in R2019b

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrCSIRS` | `nrCSIRSIndices` | `nrCSIRSMeasurements`

Objects
`nrCarrierConfig`

nrDLCarrierConfig

5G downlink waveform configuration parameters

Description

The `nrDLCarrierConfig` object sets the parameters of a single-component-carrier 5G downlink waveform. Use this object to configure 5G downlink waveform generation when calling the `nrWaveformGenerator` function.

This object defines these aspects of the downlink waveform:

- Frequency range
- Channel bandwidth
- Cell identity
- Waveform duration
- Subcarrier spacing (SCS) carriers
- Bandwidth parts (BWPs)
- Synchronization signal (SS) burst
- Control resource sets (CORESETs)
- Search spaces
- Physical downlink control channel (PDCCH) and PDCCH demodulation reference signal (DM-RS)
- Physical downlink shared channel (PDSCH), PDSCH DM-RS, and PDSCH phase-tracking reference signal (PT-RS)
- Channel state information reference signal (CSI-RS)

Creation

Syntax

```
cfgDL = nrDLCarrierConfig
cfgDL = nrDLCarrierConfig(Name,Value)
```

Description

`cfgDL = nrDLCarrierConfig` creates a default single-component-carrier 5G downlink waveform configuration object.

`cfgDL = nrDLCarrierConfig(Name,Value)` sets properties on page 3-46 using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'FrequencyRange','FR2'` specifies a downlink waveform for frequency range 2 (FR2).

Properties

`Label` — Name of downlink carrier configuration
`'Downlink carrier 1'` (default) | character array | string scalar

Name of the downlink carrier configuration, specified as a character array or string scalar. Use this property to set a description to the downlink carrier configuration.

Data Types: `char` | `string`

`FrequencyRange` — Frequency range
`'FR1'` (default) | `'FR2'`

Frequency range, specified as one of these values.

- `'FR1'` for frequency range 1 (FR1)
- `'FR2'` for frequency range 2 (FR2)

Data Types: `char` | `string`

`ChannelBandwidth` — Channel bandwidth
`50` (default) | `5` | `10` | `15` | `20` | `25` | `30` | `40` | `50` | `60` | `70` | `80` | `90` | `100` | `200` | `400`

Channel bandwidth, in MHz, specified as one of these values.

- `5`, `10`, `15`, `20`, `25`, `30`, `40`, `50`, `60`, `70`, `80`, `90`, or `100` for FR1
- `50`, `100`, `200`, or `400` for FR2

Set the frequency range with the `FrequencyRange` property.

Data Types: `double`

`NCellID` — Physical layer cell identity
`1` (default) | integer from 0 to 1007

Physical layer cell identity, specified as an integer from 0 to 1007.

Data Types: `double`

`NumSubframes` — Waveform duration in subframes
`10` (default) | positive integer

Waveform duration in subframes (multiples of 1 ms), specified as a positive integer. The default value of 10 subframes corresponds to one frame.

Data Types: `double`

`WindowingPercent` — Windowing percentage relative to FFT length
`0` (default) | real scalar from 0 to 50 | six-element row vector | `[ ]`

Windowing percentage relative to the fast Fourier transform (FFT) length, specified as one of these values.

- Real scalar from 0 to 50 — The object sets the same windowing percentage for all combinations of SCS and cyclic prefix.

- Six-element row vector of the form [*w1 w2 w3 w4 w5 w6*], where each element is a real scalar from 0 to 50 — The object sets individual windowing percentage for the different SCS and cyclic prefix combinations.
  - *w1* specifies the windowing percentage for 15 kHz SCS.
  - *w2* specifies the windowing percentage for 30 kHz SCS.
  - *w3* specifies the windowing percentage for 60 kHz SCS and normal cyclic prefix.
  - *w4* specifies the windowing percentage for 60 kHz SCS and extended cyclic prefix.
  - *w5* specifies the windowing percentage for 120 kHz SCS.
  - *w6* specifies the windowing percentage for 240 kHz SCS.
- `[]` — The object automatically selects the windowing percentage of each SCS carrier (specified by `SCSCarriers`) based on the `SampleRate` property and these additional properties.
  - The `NSizeGrid` and `SubcarrierSpacing` properties of the actual SCS carrier.
  - The `CyclicPrefix` property of the actual BWP specified by the `BandwidthParts` property.

  For more information, see the `'Windowing'` name-value argument description of the `nrOFDMModulate` function.

This property configures the number of time-domain samples, as a percentage of the FFT length, over which windowing and overlapping of the OFDM symbols take place.

Data Types: `double`

`SampleRate` — Sample rate of OFDM-modulated waveform
`[]` (default) | positive integer scalar

Sample rate of the OFDM-modulated waveform, specified as `[]` or a positive integer scalar. When you set this value to `[]`, the object sets the sample rate to the minimum value that accommodates all carriers in the waveform without aliasing.

`CarrierFrequency` — Carrier frequency in Hz
`0` (default) | real number

Carrier frequency in Hz, specified as a real number. This property corresponds to f_0, defined in TS 38.211 Section 5.4, and is used for symbol phase compensation before OFDM modulation.

Data Types: `double`

`SCSCarriers` — One or more SCS carrier configurations
`{nrSCSCarrierConfig}` (default) | cell array of `nrSCSCarrierConfig` objects

One or more SCS carrier configurations, specified as a cell array of `nrSCSCarrierConfig` objects. Because this property configures the subcarrier spacing and grid size of each numerology, each `nrSCSCarrierConfig` object in the cell array must have a unique `SubcarrierSpacing` property value.

`BandwidthParts` — One or more BWP configurations
`{nrWavegenBWPConfig}` (default) | cell array of `nrWavegenBWPConfig` objects

One or more BWP configurations, specified as a cell array of `nrWavegenBWPConfig` objects. The `SubcarrierSpacing` properties of these BWP objects must be one of the values defined by the `SubcarrierSpacing` properties of the carriers specified by the `SCSCarriers` property.

`SSBurst` — SS burst configuration
default `nrWavegenSSBurstConfig` object (default) | `nrWavegenSSBurstConfig` object

SS burst configuration, specified as an `nrWavegenSSBurstConfig` object. Use this property to configure the SS burst and blocks.

`CORESET` — One or more CORESET configurations
`{nrCORESETConfig}` (default) | cell array of `nrCORESETConfig` objects

One or more CORESET configurations, specified as a cell array of `nrCORESETConfig` objects. Use this property to specify different CORESET configurations for multiple search spaces and PDCCH.

`SearchSpaces` — One or more search space set configurations
`{nrSearchSpaceConfig}` (default) | cell array of `nrSearchSpaceConfig` objects

One or more search space set configurations, specified as a cell array of `nrSearchSpaceConfig` objects. Use this property to specify different search space set configurations for linking to a CORESET and for multiple PDCCH.

`PDCCH` — One or more PDCCH configurations
`{nrWavegenPDCCHConfig}` (default) | cell array of `nrWavegenPDCCHConfig` objects

One or more PDCCH configurations, specified as a cell array of `nrWavegenPDCCHConfig` objects. Use this property to configure different PDCCH and associated DM-RS.

`PDSCH` — One or more PDSCH configurations
`{nrWavegenPDSCHConfig}` (default) | cell array of `nrWavegenPDSCHConfig` objects

One or more PDSCH configurations, specified as a cell array of `nrWavegenPDSCHConfig` objects. Use this property to configure different PDSCH and associated DM-RS and PT-RS.

`CSIRS` — One or more CSI-RS configurations
`{nrWavegenCSIRSConfig('Enable',0)}` (default) | cell array of `nrWavegenCSIRSConfig` objects

One or more CSI-RS configurations, specified as a cell array of `nrWavegenCSIRSConfig` objects.

Examples

Configure and Generate Single-User 5G Downlink Waveform

Create an SCS carrier configuration object with the default SCS of 15 kHz and 100 resource blocks.

```
carrier = nrSCSCarrierConfig('NSizeGrid',100);
```

Create a customized BWP configuration object for the SCS carrier.

```
bwp = nrWavegenBWPConfig('NStartBWP',carrier.NStartGrid+10);
```

Create an SS burst configuration object with block pattern Case A.

```
ssb = nrWavegenSSBurstConfig('BlockPattern','Case A');
```

Create a PDCCH configuration object, specifying an aggregation of size two and the fourth candidate for the PDCCH instance.

```
pdcch = nrWavegenPDCCHConfig('AggregationLevel',2,'AllocatedCandidate',4);
```

Create a CORESET configuration object, specifying four frequency resources and a duration of three OFDM symbols.

```
coreset = nrCORESETConfig;
coreset.FrequencyResources = [1 1 1 1];
coreset.Duration = 3;
```

Create a search space set configuration object, specifying two aggregation levels.

```
ss = nrSearchSpaceConfig;
ss.NumCandidates = [8 4 0 0 0];
```

Create a PDSCH configuration object, specifying the modulation scheme and the target code rate. Enable the PDSCH PT-RS.

```
pdsch = nrWavegenPDSCHConfig( ...
    'Modulation','16QAM','TargetCodeRate',658/1024,'EnablePTRS',true);
```

Create a PDSCH DM-RS and a PDSCH PT-RS configuration object with the specified property values.

```
dmrs = nrPDSCHDMRSConfig('DMRSTypeAPosition',3);
pdsch.DMRS = dmrs;
ptrs = nrPDSCHPTRSConfig('TimeDensity',2);
pdsch.PTRS = ptrs;
```

Create a CSI-RS configuration object with the specified property values.

```
csirs = nrWavegenCSIRSConfig('RowNumber',4,'RBOffset',10,'NumRB',10,'SymbolLocations',5);
```

Create a single-user 5G downlink waveform configuration object, specifying the previously defined configurations.

```
cfgDL = nrDLCarrierConfig( ...
    'FrequencyRange','FR1', ...
    'ChannelBandwidth',40, ...
    'NumSubframes',20, ...
    'SCSCarriers',{carrier}, ...
    'BandwidthParts',{bwp}, ...
    'SSBurst',ssb, ...
    'CORESET',{coreset}, ...
    'SearchSpaces',{ss}, ...
    'PDCCH',{pdcch}, ...
    'PDSCH',{pdsch}, ...
    'CSIRS',{csirs});
```

Generate a 5G downlink waveform using the specified configuration.

```
waveform = nrWaveformGenerator(cfgDL);
```

Configure and Generate Multiuser 5G Downlink Waveform

Create two SCS carrier configuration objects with mixed numerologies and custom numbers of resource blocks.

```
carriers = {
    nrSCSCarrierConfig('SubcarrierSpacing',15,'NStartGrid',10,'NSizeGrid',100), ...
    nrSCSCarrierConfig('SubcarrierSpacing',30,'NStartGrid',0,'NSizeGrid',70)};
```

Create two custom BWP configuration objects, one for each of the carriers.

```
bwp = {
    nrWavegenBWPConfig('BandwidthPartID',1,'SubcarrierSpacing',15,'NStartBWP',10,'NSizeBWP',80),
    nrWavegenBWPConfig('BandwidthPartID',2,'SubcarrierSpacing',30,'NStartBWP',0,'NSizeBWP',60)};
```

Create an SS burst configuration object with block pattern Case A, corresponding to an SCS of 15 kHz.

```
ssb = nrWavegenSSBurstConfig('BlockPattern','Case A');
```

Create two PDCCH configuration objects.

```
pdcch = {
    nrWavegenPDCCHConfig('SearchSpaceID',1,'BandwidthPartID',1,'RNTI',1,'DMRSScramblingID',1), .
    nrWavegenPDCCHConfig('SearchSpaceID',2,'BandwidthPartID',2,'RNTI',2,'DMRSScramblingID',2, ..
        'AggregationLevel',4)};
```

Create two CORESET configuration objects and two search space set configuration objects for the two PDCCH.

```
coreset = {
    nrCORESETConfig('CORESETID',1,'FrequencyResources',[1 1 1 1 1 0 0 0 0 0 1],'Duration',3), ..
    nrCORESETConfig('CORESETID',2,'FrequencyResources',[0 0 0 0 0 0 0 0 1 1])};

ss = {
    nrSearchSpaceConfig('SearchSpaceID',1,'CORESETID',1,'StartSymbolWithinSlot',4), ...
    nrSearchSpaceConfig('SearchSpaceID',2,'CORESETID',2,'NumCandidates',[8 8 4 0 0])};
```

Create two PDSCH configuration objects with mixed modulation schemes.

```
pdsch = {
    nrWavegenPDSCHConfig('BandwidthPartID',1,'Modulation','16QAM','RNTI',1,'NID',1,'PRBSet',10:5
    nrWavegenPDSCHConfig('BandwidthPartID',2,'Modulation','QPSK','RNTI',2,'NID',2, ...
        'PRBSet', 50:59)};
```

Create two CSI-RS configuration objects.

```
 csirs = {
     nrWavegenCSIRSConfig('BandwidthPartID',1,'RowNumber',2,'RBOffset',20), ...
     nrWavegenCSIRSConfig('BandwidthPartID',2,'Density','one','RowNumber',4,'NumRB',10)};
```

Create a multiuser 5G downlink waveform configuration object, specifying the previously defined configurations.

```
cfgDL = nrDLCarrierConfig( ...
    'FrequencyRange','FR1', ...
    'ChannelBandwidth',40, ...
    'NumSubframes',20, ...
    'SCSCarriers',carriers, ...
    'BandwidthParts',bwp, ...
    'SSBurst',ssb, ...
    'CORESET',coreset, ...
```

```
    'SearchSpaces',ss, ...
    'PDCCH',pdcch, ...
    'PDSCH',pdsch, ...
    'CSIRS',csirs);
```

Generate a 5G downlink waveform using the specified configuration.

```
waveform = nrWaveformGenerator(cfgDL);
```

Version History

Introduced in R2020b

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

nrWaveformGenerator

Objects

nrWavegenBWPConfig | nrWavegenSSBurstConfig | nrWavegenPDSCHConfig | nrWavegenPDCCHConfig | nrWavegenCSIRSConfig | nrSCSCarrierConfig | nrPDSCHPTRSConfig | nrPDSCHDMRSConfig | nrSearchSpaceConfig | nrCORESETConfig | nrULCarrierConfig

nrPathLossConfig

Path loss configuration parameters

Description

The `nrPathLossConfig` object sets path loss configuration parameters for a specific scenario, as defined in TR 38.901 Section 7.4.1 [1].

The default `nrPathLossConfig` object configures an urban macrocell scenario with an environment height of 1 m. The default configuration does not use an optional path loss model.

Creation

Syntax

```
pathlossconf = nrPathLossConfig
pathlossconf = nrPathLossConfig(Name=Value)
```

Description

`pathlossconf = nrPathLossConfig` creates a path loss configuration object.

`pathlossconf = nrPathLossConfig(Name=Value)` sets properties on page 3-52 using one or more name-value arguments. For example, `BuildingHeight=4` specifies 4 m as the average building height.

Properties

`Scenario` — Scenario characteristic
`'UMa'` (default) | `'UMi'` | `'RMa'` | `'InH'` | `'InF-SL'` | `'InF-DL'` | `'InF-SH'` | `'InF-DH'` | `'InF-HH'`

Scenario characteristic, specified as one of these values.

- `'UMa'` — Urban macrocell
- `'UMi'` — Urban microcell
- `'RMa'` — Rural macrocell
- `'InH'` — Indoor hotspot
- `'InF-SL'` — Indoor factory with sparse clutter and low base station (BS) height
- `'InF-DL'` — Indoor factory with dense clutter and low BS height
- `'InF-SH'` — Indoor factory with sparse clutter and high BS height
- `'InF-DH'` — Indoor factory with dense clutter and high BS height
- `'InF-HH'` — Indoor factory with high Tx and high Rx

Data Types: `char` | `string`

BuildingHeight — Average building height
5 (default) | numeric scalar in the range from 5 to 50

Average building height, in meters, in a rural macrocell scenario, specified as a numeric scalar in the range from 5 to 50.

Dependencies

To enable this property, set the Scenario property to 'RMa'.

Data Types: double

StreetWidth — Average street width
20 (default) | numeric scalar in the range from 5 to 50

Average street width, in meters, in a rural macrocell scenario, specified as a numeric scalar in the range from 5 to 50.

Dependencies

To enable this property, set the Scenario property to 'RMa'.

Data Types: double

EnvironmentHeight — Average environment height
1 (default) | numeric scalar | numeric matrix

Average environment height, in meters, in an urban macrocell or microcell scenario, specified as a numeric scalar or numeric matrix of size N_{BS}-by-N_{UE}. N_{BS} is the number of BSs. N_{UE} is the number of user equipments (UEs).

Dependencies

To enable this property, set the Scenario property to 'UMa' or 'UMi'.

Data Types: double

OptionalModel — Optional path loss model
false or 0 (default) | true or 1

Optional path loss model, specified as one of these numeric or logical values.

- 0 (false) — The configuration does not use an optional path loss model.
- 1 (true) — The configuration uses an optional path loss model, as defined in TR 38.901 Section 7.4 Table 7.4.1-1 [1] for urban macrocell, urban microcell, and indoor hotspot scenarios.

Dependencies

To enable this property, set the Scenario property to 'UMa', 'UMi', or 'InH'.

Data Types: double | logical

Examples

Calculate Path Loss Between Multiple BSs and UEs

Create a path loss configuration object. Set the characteristics such that they correspond to a rural macrocell scenario with an average building height of 7 m and street width of 25 m.

```
pathlossconf = nrPathLossConfig;
pathlossconf.Scenario = "RMa";
pathlossconf.BuildingHeight = 7;
pathlossconf.StreetWidth = 25;
```

Specify the carrier frequency in Hz.

```
freq = 3.5e9;
```

Specify the coordinates of two BSs and ten UEs. The BSs are 1 km apart. The UEs are randomly placed inside a 2 km-by-2 km square region at elevations between 1 m and 2 m.

```
bs = [-500 500; 0 0; 30 50];
nbs = size(bs,2);
nue = 10;
ue = zeros(3,nue);
ue(1:2,:) = 2e3*(rand(2,nue)-0.5);
ue(3,:) = 1 + rand(1,nue);
```

Specify the LOS condition between each BS and UE pair.

```
los = randi([0 1],nbs,nue);
```

Calculate the path loss between each BS and UE pair.

```
pathloss = nrPathLoss(pathlossconf,freq,los,bs,ue)
```

```
pathloss = 2×10

  109.9708   131.3512   133.6539    85.0968   112.2626   131.3747   110.1649   137.4889   130.5121   110.2
  123.4839   134.9305   104.4709   125.2657   126.9743   133.3612    97.4411   121.9781   106.9546   105.4
```

Version History

Introduced in R2021b

References

[1] 3GPP TR 38.901. “Study on channel model for frequencies from 0.5 to 100 GHz.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
nrPathLoss

Topics
"Include Path Loss in NR Link-Level Simulations"

nrPCAPWriter

PCAP or PCAPNG file writer of 5G NR MAC packets

Description

The `nrPCAPWriter` object creates a packet capture (PCAP) or packet capture next generation (PCAPNG) file writer object to write generated and recovered 5G new radio (NR) medium access control (MAC) packets to a PCAP or PCAPNG file (`.pcap` or `.pcapng`, respectively). To write 5G NR MAC packets, no native link type is available for NR. The object writes the 5G NR MAC packets to the PCAP or PCAPNG file by encapsulating the packets into a pseudo protocol with a link type. The object prepends each MAC packet with socket address link layer (SLL), internet protocol (IP), and user datagram protocol (UDP) headers followed by per-packet information. The encapsulation of packets enables the object to write a MAC packet with a maximum size of 65,482 bytes at a time. If the size of the MAC packet is greater than 65,482 bytes, the object truncates the remaining bytes.

Creation

Syntax

```
nrPCAPW = nrPCAPWriter
nrPCAPW = nrPCAPWriter(Name=Value)
```

Description

`nrPCAPW = nrPCAPWriter` creates a default 5G NR PCAP or PCAPNG file writer object.

`nrPCAPW = nrPCAPWriter(Name=Value)` sets properties on page 3-56 using one or more optional name-value arguments. For example, `FileExtension='pcapng'` sets the extension of the file as `.pcapng`.

Properties

Note The `nrPCAPWriter` object does not overwrite the existing PCAP or PCAPNG file. Each time when you create this object, specify a unique PCAP or PCAPNG file name.

`FileName` — Name of PCAP or PCAPNG file
`'capture'` (default) | character vector | string scalar

Name of a PCAP or PCAPNG file, specified as a character vector or a string scalar.

Data Types: `char` | `string`

`ByteOrder` — Byte order
`'little-endian'` (default) | `'big-endian'`

Byte order, specified as `'little-endian'` or `'big-endian'`.

Data Types: char | string

FileExtension — Type of file
'pcap' (default) | 'pcapng'

Type of file, specified as 'pcap' or 'pcapng'.

Data Types: char | string

FileComment — Comment for PCAPNG file
'' (default) | character vector | string scalar

Comment for the PCAPNG file, specified as a character vector or a string scalar.

Data Types: char | string

Interface — Name of interface on which NR MAC packets are captured
'5GNR' (default) | character vector | string scalar

Name of the interface on which NR MAC packets are captured, specified as a character vector or a string scalar.

Data Types: char | string

PCAPWriter — PCAP or PCAPNG file writer object
pcapWriter object | pcapngWriter object

PCAP or PCAPNG file writer object, specified as a pcapWriter or pcapngWriter object.

Object Functions

Specific to This Object

write   Write 5G NR MAC packet to PCAP or PCAPNG file

Examples

Write 5G NR MAC Packet to PCAP File

Create a 5G NR PCAP file writer object, specifying the name and extension of the file.

```
nrpcapw = nrPCAPWriter(FileName='nrPCAPExample',FileExtension='pcap');
```

Create a 5G NR MAC packet.

```
nrMACPDU = [6; 68; 64; 0; ones(66,1); 62; 4; 7; 74; 96; 102];
```

Set the timestamp for the packet.

```
timestamp = 1000000;                          % In microseconds
```

Create the packet information structure for the MAC packet by using the constants defined in the created 5G NR PCAP file writer object.

```
packetInfo = struct();
packetInfo.RadioType = nrpcapw.RadioFDD;      % Frequency division duplexing
```

```
packetInfo.LinkDir = nrpcapw.Uplink;          % Link direction
packetInfo.RNTIType = nrpcapw.CellRNTI;       % Type of radio network temporary identifier (RNTI)
```

Write the 5G NR MAC packet to the PCAP file.

```
write(nrpcapw,nrMACPDU,timestamp,PacketInfo=packetInfo);
```

Write SIB1 Packet with Comment to PCAPNG File

Create a 5G NR PCAPNG file writer object, specifying the name of the file, extension of the file, and file comment.

```
nrpcapw = nrPCAPWriter(FileName='nrPCAPExample2', ...
    FileExtension='pcapng', ...
    FileComment='SIB1 Packet');
```

Create a system information block 1 (SIB1) packet.

```
sib1Packet = [64; 0; 0; 36; 104; 21; 0; 10; 156; 1; 15; zeros(13,1)];
```

Set the timestamp for the packet.

```
timestamp = 1000000;                      % In microseconds
```

Create the packet information structure for the SIB1 packet by using the constants defined in the created 5G NR PCAPNG file writer object.

```
packetInfo = struct();
packetInfo.RadioType = nrpcapw.RadioFDD;         % Frequency division duplexing
packetInfo.LinkDir = nrpcapw.Downlink;           % Link direction
packetInfo.RNTIType = nrpcapw.SystemInfoRNTI;    % Type of radio network temporary identifier (RNTI)
```

Write the SIB1 packet to the PCAPNG file.

```
write(nrpcapw,sib1Packet,timestamp,PacketInfo=packetInfo);
```

Write 5G NR MAC BSR Packet to PCAPNG File

Create a 5G NR PCAPNG file writer object, specifying the name of the file, extension of the file, and packet comment.

```
nrpcapw = nrPCAPWriter(FileName='nrPCAPExample3', ...
    FileExtension='pcapng', ...
    FileComment='Sample file');
```

Create a 5G NR MAC packet containing short truncated buffer status report (BSR).

```
nrMACPDU = [59; 205];
```

Set the timestamp for the packet.

```
timestamp = 1000000;                       % In microseconds
```

Create the packet information structure for the MAC packet by using the constants defined in the created 5G NR PCAPNG file writer object.

```
packetInfo = struct();
packetInfo.RadioType = nrpcapw.RadioFDD;    % Frequency division duplexing
packetInfo.LinkDir = nrpcapw.Uplink;        % Link direction
packetInfo.RNTIType = nrpcapw.CellRNTI;     % Type of radio network temporary identifier (RNTI)
packetInfo.RNTI = 15;                       % RNTI value in the range [0, 65535]
packetInfo.UEID = 1022;                     % User equipment identifier in the range [0, 65535]
packetInfo.SystemFrameNumber = 10;          % System frame number in the range [0, 1023]
```

Write the 5G NR MAC packet to the PCAPNG file.

```
write(nrpcapw,nrMACPDU,timestamp,PacketInfo=packetInfo, ...
    PacketComment='This is a NR MAC BSR packet');
```

Version History

Introduced in R2021b

References

[1] Tuexen, M. “PCAP Next Generation (Pcapng) Capture File Format.” 2020. https://www.ietf.org/.

[2] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`pcapWriter` | `pcapngWriter`

nrPDCCHConfig

PDCCH configuration parameters

Description

The `nrPDCCHConfig` object sets physical downlink control channel (PDCCH) configuration parameters, as defined in TS 38.211 Section 7.3.2 [1] and TS 38.213 Section 10 [2].

Creation

Syntax

```
pdcch = nrPDCCHConfig
pdcch = nrPDCCHConfig(Name,Value)
```

Description

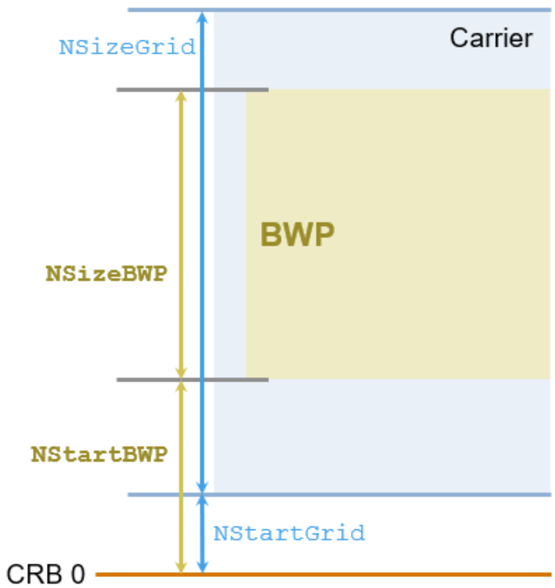
`pdcch = nrPDCCHConfig` creates a PDCCH configuration object with default properties.

`pdcch = nrPDCCHConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'NSizeBWP',36` configures the PDCCH with a bandwidth part (BWP) consisting of 36 resource blocks (RBs). Unspecified properties take their default values.

Properties

`NStartBWP` — Start of BWP resource grid relative to CRB 0
`0` (default) | nonnegative integer

Start of BWP resource grid relative to the common resource block 0 (CRB 0), specified as a nonnegative integer. Set this property relative to the carrier such that the property value is in this range: `NStartGrid` ≤ `NStartBWP` < (`NStartGrid` + `NSizeGrid`), where `NStartGrid` and `NSizeGrid` are properties of the carrier configuration object `nrCarrierConfig`. This figure shows where in the carrier the BWP is located in terms of this property and the `NSizeBWP` property.



Data Types: `double`

NSizeBWP — Number of RBs in BWP resource grid
48 (default) | integer from 1 to 275

Number of RBs in BWP resource grid, specified as an integer from 1 to 275. This property must be less than or equal to the size of the carrier, which is specified by the `NSizeGrid` property of the carrier configuration object `nrCarrierConfig`.

Data Types: `double`

CORESET — CORESET configuration
`nrCORESETConfig` object with default properties (default)

Control resource set (CORESET) configuration, specified as an `nrCORESETConfig` object.

SearchSpace — Search space set configuration
`nrSearchSpaceConfig` object with default properties (default)

Search space set configuration, specified as an `nrSearchSpaceConfig` object.

RNTI — Radio network temporary identifier
1 (default) | integer from 0 to 65,519

Radio network temporary identifier (RNTI), specified as an integer from 0 to 65,519.

- If the higher layer parameter *pdcch-DMRS-ScramblingID* is configured, `RNTI` is the cell radio network temporary identifier (C-RNTI) with integer value from 1 to 65,519.
- If *pdcch-DMRS-ScramblingID* is not configured, `RNTI` is `0`.

Data Types: `double`

DMRSScramblingID — PDCCH DM-RS scrambling identity
2 (default) | integer from 0 to 65,535 | [ ]

PDCCH demodulation reference signal (DM-RS) scrambling identity, specified as an integer from 0 to 65,535 if the higher layer parameter *pdcch-DMRS-ScramblingID* is configured or as `[]` if *pdcch-DMRS-ScramblingID* is not configured. When you specify this property as `[]`, the object sets the PDCCH DM-RS scrambling identity to the physical layer cell identity specified by the `NCellID` property of the carrier.

Data Types: `double`

`AggregationLevel` — PDCCH aggregation level
8 (default) | 1 | 2 | 4 | 16

PDCCH aggregation level, specified as `1`, `2`, `4`, `8`, or `16`.

Data Types: `double`

`AllocatedCandidate` — Candidate used for PDCCH instance
`1` (default) | integer from 1 to 8

Candidate used for the PDCCH instance, specified as an integer from 1 to 8. The value of this property is an index from the set of candidates specified for the aggregation level by the `SearchSpace.NumCandidates` property.

This property does not apply when the `CCEOffset` property is set to a nonempty value.

Data Types: `double`

`CCEOffset` — CCE offset used for PDCCH instance
`[]` (default) | positive integer

Control channel element (CCE) offset used for the PDCCH instance, specified as one of these options.

- `[]` — The `AllocatedCandidate` property determines the first CCE used for the PDCCH instance.
- Positive integer — This value explicitly specifies the index of the first CCE used for the PDCCH instance. The value must be a multiple of the `AggregationLevel` property value and less than or equal to the CCE capacity of the associated CORESET. This option overrides the allocation specified by the `AllocatedCandidate` property.

Data Types: `double`

Examples

Generate PDCCH DM-RS Symbols and Indices

Configure the carrier with default configuration parameters.

```
carrier = nrCarrierConfig;
```

Configure the CORESET with 6 frequency resources, a duration of 3 OFDM symbols, and a REG bundle size of 3.

```
crst = nrCORESETConfig;
crst.FrequencyResources = ones(1,6);
crst.Duration = 3;
crst.REGBundleSize = 3;
```

Configure the PDCCH with the specified bandwidth part and CORESET.

```
pdcch = nrPDCCHConfig;
pdcch.NStartBWP = 6;
pdcch.NSizeBWP = 36;
pdcch.CORESET = crst;
pdcch.AggregationLevel = 16;
```

Generate PDCCH DM-RS symbols and indices for the specified carrier and PDCCH.

```
[~,dmrs,dmrsInd] = nrPDCCHResources(carrier,pdcch);
```

Generate PDCCH and DM-RS Indices Relative to BWP Grid

Configure a carrier grid of 60 resource blocks (RBs), where the starting RB index relative to the common resource block 0 (CRB 0) is 3.

```
carrier = nrCarrierConfig;
carrier.NStartGrid = 3;
carrier.NSizeGrid = 60;
```

Configure noninterleaved CORESET with 6 frequency resources and a duration of 3 OFDM symbols.

```
crst = nrCORESETConfig;
crst.FrequencyResources = ones(1,6);
crst.Duration = 3;
crst.CCEREGMapping = 'noninterleaved';
```

Configure the PDCCH with the specified bandwidth part and CORESET.

```
pdcch = nrPDCCHConfig;
pdcch.NStartBWP = 5;
pdcch.NSizeBWP = 48;
pdcch.CORESET = crst;
pdcch.AggregationLevel = 16;
```

Generate PDCCH resource element indices and DM-RS symbol indices using 1-based, subscript indexing form relative to the BWP grid.

```
[ind,~,dmrsInd] = nrPDCCHResources(carrier,pdcch,...
    'IndexOrientation','bwp','IndexStyle','subscript');
```

Generate PDCCH DM-RS Symbols for All Candidates and Aggregation Levels

Configure the carrier with default configuration parameters.

```
carrier = nrCarrierConfig;
```

Configure the CORESET with 6 frequency resources, a duration of 3 OFDM symbols, and a REG bundle size of 3.

```
crst = nrCORESETConfig;
crst.FrequencyResources = ones(1,6);
```

```
crst.Duration = 3;
crst.REGBundleSize = 3;
```

Configure the search space set for the PDCCH with the specified number of candidates at each aggregation level.

```
cfgSS = nrSearchSpaceConfig;
cfgSS.NumCandidates = [5 5 3 2 1];
```

Configure the PDCCH with the specified bandwidth part, CORESET, and search space set.

```
pdcch = nrPDCCHConfig;
pdcch.NStartBWP = 6;
pdcch.NSizeBWP = 36;
pdcch.CORESET = crst;
pdcch.SearchSpace = cfgSS;
```

Generate PDCCH DM-RS symbols for all candidates and aggregation levels.

```
[~,allDMRS] = nrPDCCHSpace(carrier,pdcch)
```

```
allDMRS=5×1 cell array
    { 18x5 double}
    { 36x5 double}
    { 72x3 double}
    {144x2 double}
    {288x1 double}
```

Verify that the number of generated candidates for the PDCCH DM-RS symbols at each aggregation level matches the number of candidates specified by the search space set.

```
numCandidates = [...
    size(allDMRS{1},2) ...
    size(allDMRS{2},2) ...
    size(allDMRS{3},2) ...
    size(allDMRS{4},2) ...
    size(allDMRS{5},2)];
isequaln(cfgSS.NumCandidates,numCandidates)
```

```
ans = logical
   1
```

Version History

Introduced in R2020a

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.213. “NR; Physical layer procedures for control.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrPDCCH` | `nrPDCCHResources` | `nrPDCCHSpace`

Objects
`nrCarrierConfig` | `nrSearchSpaceConfig` | `nrCORESETConfig`

nrPDSCHConfig

PDSCH configuration parameters

Description

The `nrPDSCHConfig` object sets physical downlink shared channel (PDSCH) configuration parameters, as defined in TS 38.211 Sections 7.3.1, 7.4.1.1, and 7.4.1.2 [1].

This object defines all the properties involved in the PDSCH processing chain, including scrambling, symbol modulation, layer mapping, virtual resource blocks (VRB) to physical resource blocks (PRB) interleaving, and resource element (RE) mapping with the reserved resources patterns. The object also contains properties of the associated physical reference signals, such as demodulation reference signal (DM-RS) and phase tracking reference signal (PT-RS).

The default `nrPDSCHConfig` object configures a single-layer PDSCH with mapping type A, QPSK modulation, a resource allocation of 52 resource blocks and 14 OFDM symbols in a slot, and single-symbol DM-RS type 1. This configuration corresponds to a full resource allocation with respect to the default `nrCarrierConfig` object.

Creation

Syntax

```
pdsch = nrPDSCHConfig
pdsch = nrPDSCHConfig(Name,Value)
```

Description

`pdsch = nrPDSCHConfig` creates a PDSCH configuration object with default properties.

`pdsch = nrPDSCHConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'NSizeBWP',200,'NumLayers',7` specifies 200 PRBs in the bandwidth part (BWP) and 7 transmission layers. Unspecified properties take their default values.

Properties

Channel Configuration

`NSizeBWP` — Number of PRBs in BWP
`[]` (default) | integer from 1 to 275

Number of PRBs in BWP, specified as an integer from 1 to 275. Use `[]` to set this property to the `NSizeGrid` property of the `nrCarrierConfig` object.

Data Types: `double`

NStartBWP — Starting PRB index of BWP relative to CRB 0
[] (default) | integer from 0 to 2473

Starting PRB index of BWP relative to common resource block (CRB) 0, specified as an integer from 0 to 2473. Use [] to set this property to the NStartGrid property of the nrCarrierConfig object.

Data Types: double

ReservedPRB — Reserved PRBs and OFDM symbols pattern in BWP
default nrPDSCHReservedConfig object (default) | cell array of nrPDSCHReservedConfig objects

Reserved PRBs and OFDM symbols pattern in the BWP, specified as a cell array of nrPDSCHReservedConfig objects.

Data Types: cell

ReservedRE — Reserved RE indices within BWP
[] (default) | vector of nonnegative integers

Reserved RE indices within the BWP, specified as a vector of nonnegative integers. This property specifies RE indices (0-based) that are unavailable for a PDSCH due to the channel state information reference signal (CSI-RS) or cell-specific reference signal being present in a particular slot.

Data Types: double

Modulation — Modulation scheme
'QPSK' (default) | '16QAM' | '64QAM' | '256QAM' | string scalar | string array | cell array of character vectors

Modulation scheme, specified as 'QPSK', '16QAM', '64QAM', or '256QAM', a string scalar, a string array, or a cell array of character vectors. This modulation scheme specifies the modulation type of the codewords and the number of bits used per modulation symbol. For one codeword, specify the modulation scheme as a character vector or string scalar. If two codewords are present (NumLayers > 4), the same modulation scheme applies to both codewords or you can specify different modulation schemes for each codeword by using a string array or a cell array of character vectors.

Modulation Scheme	Number of Bits Per Symbol
'QPSK'	2
'16QAM'	4
'64QAM'	6
'256QAM'	8

Example: {'QPSK','16QAM'} or ["QPSK","16QAM"] specifies different modulation schemes for two codewords.

Data Types: char | string | cell

NumLayers — Number of transmission layers
1 (default) | integer from 1 to 8

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

- For one codeword, specify an integer from 1 to 4.
- For two codewords, specify an integer from 5 to 8.

Data Types: double

MappingType — Mapping type
'A' (default) | 'B'

Mapping type of the physical shared channel, specified as 'A' or 'B'.

Data Types: char | string

SymbolAllocation — OFDM symbol allocation
[0 14] (default) | two-element vector of nonnegative integers

OFDM symbol allocation of the physical shared channel, specified as a two-element vector of nonnegative integers. The first element of this property represents the start of symbol allocation (0-based). The second element represents the number of allocated OFDM symbols.

When you set this property to [] or the second element of the vector to 0, no symbol is allocated for the channel.

Data Types: double

PRBSet — PRB allocation
[0:51] (default) | vector of integers from 0 to 274

PRB allocation of the PDSCH in the BWP, specified as a vector of integers from 0 to 274.

Data Types: double

VRBToPRBInterleaving — Enable VRB-to-PRB interleaving
0 (default) | 1

Enable VRB-to-PRB interleaving, specified as one of these values.

- 0 — Disable VRB-to-PRB interleaving.
- 1 — Enable VRB-to-PRB interleaving.

Data Types: double | logical

VRBBundleSize — VRB bundle size
2 (default) | 4

VRB bundle size, in terms of the number of PRBs for VRB-to-PRB interleaving, specified as 2 or 4.

Dependencies

To enable this property, set the VRBToPRBInterleaving property to 1.

Data Types: double

NID — PDSCH scrambling identity
[] (default) | integer from 0 to 1023

PDSCH scrambling identity, specified as [] or an integer from 0 to 1023.

- If the higher layer parameter *dataScramblingIdentityPDSCH* is configured, NID must be in the range from 0 to 1023.
- If the higher layer parameter *dataScramblingIdentityPDSCH* is not configured, NID must be in the range from 0 to 1007.

When you specify this property as [], the object sets the PDSCH scrambling identity to the physical layer cell identity, specified by the NCellID property of the carrier.

Data Types: double

RNTI — Radio network temporary identifier
1 (default) | integer from 0 to 65,535

Radio network temporary identifier of the user equipment (UE), specified as an integer from 0 to 65,535.

Data Types: double

Reference Signals Configuration

DMRS — PDSCH DM-RS configuration parameters
nrPDSCHDMRSConfig object with default properties (default) | nrPDSCHDMRSConfig object

PDSCH DM-RS configuration parameters, specified as an nrPDSCHDMRSConfig object.

EnablePTRS — Enable PT-RS
0 or false (default) | 1 or true

Enable the PT-RS, specified as one of these values.

- 0 (false) — Disable the PT-RS configuration.
- 1 (true) — Enable the PT-RS configuration.

Data Types: double | logical

PTRS — PDSCH PT-RS configuration parameters
nrPDSCHPTRSConfig object with default properties (default) | nrPDSCHPTRSConfig object

PDSCH PT-RS configuration, specified as an nrPDSCHPTRSConfig object.

Read-Only Properties

NumCodewords — Number of codewords
1 (default) | 2

This property is read-only.

Number of codewords, specified as 1 or 2. This property is updated based on the DMRSPortSet property of nrPDSCHDMRSConfig object. Use NumLayers property to calculate the number of codewords, when DMRSPortSet property is empty.

Data Types: double

Examples

Create PDSCH Configuration Object

Create a physical downlink shared channel (PDSCH) configuration object that occupies a bandwidth of 10 MHz bandwidth with 15 kHz subcarrier spacing.

Specify 52 PRBs in the bandwidth part (BWP), a scrambling identity of 750, and a 16-QAM modulation scheme. Enable VRB-to-PRB interleaving and PT-RS configuration.

```
pdsch = nrPDSCHConfig;
pdsch.NSizeBWP = 52;
pdsch.NID = 750;
pdsch.Modulation = '16QAM';
pdsch.VRBToPRBInterleaving = 1;
pdsch.EnablePTRS = 1;
disp(pdsch)
```

```
  nrPDSCHConfig with properties:

                NSizeBWP: 52
               NStartBWP: []
             ReservedPRB: {[1x1 nrPDSCHReservedConfig]}
              ReservedRE: []
              Modulation: '16QAM'
               NumLayers: 1
             MappingType: 'A'
        SymbolAllocation: [0 14]
                  PRBSet: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 ... ]
    VRBToPRBInterleaving: 1
           VRBBundleSize: 2
                     NID: 750
                    RNTI: 1
                    DMRS: [1x1 nrPDSCHDMRSConfig]
              EnablePTRS: 1
                    PTRS: [1x1 nrPDSCHPTRSConfig]

   Read-only properties:
             NumCodewords: 1
```

Create PDSCH Object to Configure Two Reserved PRB Patterns

Create a PDSCH configuration object with two reserved PRB patterns.

```
pdsch = nrPDSCHConfig('ReservedPRB',{nrPDSCHReservedConfig,nrPDSCHReservedConfig});
```

For each reserved PRB pattern, specify the reserved PRB indices in the BWP, the OFDM symbols associated with those reserved PRBs, and the period for the number of slots in the pattern.

```
pdsch.ReservedPRB{1}.PRBSet = (0:15);
pdsch.ReservedPRB{1}.SymbolSet = (5:6);
pdsch.ReservedPRB{1}.Period = 5;
pdsch.ReservedPRB{2}.PRBSet = (0:23);
pdsch.ReservedPRB{2}.SymbolSet = [2:4 7:9];
pdsch.ReservedPRB{2}.Period = 3;
```

Display the two PRB patterns.

```
PRBPattern1 = pdsch.ReservedPRB{1}
```

```
PRBPattern1 =
  nrPDSCHReservedConfig with properties:
```

```
        PRBSet: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]
     SymbolSet: [5 6]
        Period: 5
```

```
PRBPattern2 = pdsch.ReservedPRB{2}
```

```
PRBPattern2 = 
  nrPDSCHReservedConfig with properties:

        PRBSet: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
     SymbolSet: [2 3 4 7 8 9]
        Period: 3
```

Generate PDSCH Symbols and Indices

Create a carrier configuration object with default properties. Specify the physical layer cell identity as 42 and slot number as 10.

```
carrier = nrCarrierConfig;
carrier.NCellID = 42;
carrier.NSlot = 10;
```

Create a PDSCH configuration object with a 16-QAM modulation scheme. Set the radio network temporary identifier to 1005, size of the BWP to 25, starting PRB index of the BWP to 10, and PRB set to occupy the whole BWP.

```
pdsch = nrPDSCHConfig;
pdsch.Modulation = '16QAM';
pdsch.RNTI = 1005;
pdsch.NID = []; % Set NID equal to the NCellID property of carrier
pdsch.NSizeBWP = 25;
pdsch.NStartBWP = 10;
pdsch.PRBSet = 0:pdsch.NSizeBWP-1;
```

Generate PDSCH indices in subscript form and set the index orientation to bandwidth part.

```
[ind,info] = nrPDSCHIndices(carrier,pdsch,'IndexStyle','subscript','IndexOrientation','bwp')
```

```
ind = 3900x3 uint32 matrix

    1    1    1
    2    1    1
    3    1    1
    4    1    1
    5    1    1
    6    1    1
    7    1    1
    8    1    1
    9    1    1
   10    1    1
      ⋮

info = struct with fields:
                G: 15600
```

```
               Gd: 3900
        NREPerPRB: 156
    DMRSSymbolSet: 2
    PTRSSymbolSet: [1x0 double]
```

Generate PDSCH symbols of data type single.

```
numDataBits = info.G;
cws = randi([0 1],numDataBits,1);
sym = nrPDSCH(carrier,pdsch,cws,'OutputDataType','single')
```

```
sym = 3900x1 single column vector

  -0.9487 + 0.9487i
  -0.9487 - 0.9487i
  -0.3162 - 0.9487i
   0.9487 - 0.3162i
  -0.9487 + 0.3162i
   0.3162 + 0.9487i
   0.3162 + 0.9487i
  -0.3162 + 0.3162i
   0.3162 + 0.3162i
   0.9487 - 0.3162i
        ⋮
```

Version History

Introduced in R2020a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`nrPDSCH` | `nrPDSCHDecode` | `nrPDSCHIndices` | `nrPDSCHPTRS` | `nrPDSCHPTRSIndices` | `nrPDSCHDMRS` | `nrPDSCHDMRSIndices`

Objects

`nrCarrierConfig` | `nrPDSCHReservedConfig` | `nrPDSCHPTRSConfig` | `nrPDSCHDMRSConfig`

nrPDSCHDMRSConfig

PDSCH DM-RS configuration parameters

Description

The `nrPDSCHDMRSConfig` object sets demodulation reference signal (DM-RS) configuration parameters for a physical downlink shared channel (PDSCH), as defined in TS 38.211 Section 7.4.1.1 [1]. Use this object when setting the `DMRS` property of the `nrPDSCHConfig` or `nrWavegenPDSCHConfig` objects.

The object defines the properties of PDSCH DM-RS symbols and indices generation and the resource elements pattern not used for data in DM-RS symbol locations. The read-only properties of this object provide the DM-RS subcarrier locations within a resource block (RB), code division multiplexing (CDM) groups, and time and frequency weights for DM-RS symbols. By default, the object specifies a single symbol DM-RS at symbol index 2 (0-based) with configuration type 1 and antenna port 0.

Creation

Syntax

```
dmrs = nrPDSCHDMRSConfig
dmrs = nrPDSCHDMRSConfig(Name,Value)
```

Description

`dmrs = nrPDSCHDMRSConfig` creates a DM-RS configuration object for a PDSCH with default properties.

`dmrs = nrPDSCHDMRSConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'DMRSConfigurationType',1,'DMRSLength',2` specifies a double-symbol DM-RS with configuration type 1. Unspecified properties take their default values.

Properties

`DMRSConfigurationType` — DM-RS configuration type
`1` (default) | `2`

DM-RS configuration type, specified as `1` or `2`. This property is the higher-layer parameter *dmrs-Type*.

Data Types: `double`

`DMRSReferencePoint` — Reference point for DM-RS sequence to subcarrier resource mapping
`CRB0` (default) | `PRB0`

Reference point for the DM-RS sequence to subcarrier resource mapping, specified as one of these options.

- `PRB0` — When the reference point is subcarrier 0 of the physical resource block 0 (PRB 0) of the bandwidth part (BWP). Use this option when PDSCH is signalled by control resource set 0 (CORESET 0). For this case, the BWP parameters must align with CORESET 0.
- `CRB0` — When the reference point is subcarrier 0 of the common resource block 0 (CRB 0)

Data Types: `char` | `string`

`DMRSTypeAPosition` — Position of first DM-RS OFDM symbol
`2` (default) | `3`

Position of first DM-RS OFDM symbol, provided by higher layer parameter *dmrs-TypeA-Position*, specified as `2` or `3`.

This property only applies when the `MappingType` property of the `nrPDSCHConfig` or `nrWavegenPDSCHConfig` objects is set to `'A'`.

Data Types: `double`

`DMRSAdditionalPosition` — Maximum number of DM-RS additional positions
`0` (default) | `1` | `2` | `3`

Maximum number of DM-RS additional positions, specified as `0`, `1`, `2`, or `3`. This property is the higher layer parameter *dmrs-AdditionalPosition*.

Data Types: `double`

`DMRSLength` — Number of consecutive front-loaded DM-RS OFDM symbols
`1` (default) | `2`

Number of consecutive front-loaded DM-RS OFDM symbols, specified as `1` (single-symbol DM-RS) or `2` (double-symbol DM-RS).

Data Types: `double`

`CustomSymbolSet` — DM-RS OFDM symbol locations
`[]` (default) | integer from 0 to 13 | vector of nonnegative integers

DM-RS OFDM symbol locations that are 0-based, specified as one of these options.

- Integer from 0 to 13 — For one DM-RS symbol
- Vector of nonnegative integers from 0 to 13 — For multiple DM-RS symbols

Each input symbol location is assumed to be a single-symbol DM-RS within the physical shared channel symbol allocation.

The default value, `[]`, corresponds to the DM-RS symbol locations, as defined in TS 38.211 Table 7.4.1.1.2-3 or 7.4.1.1.2-4. Setting this property overrides the corresponding DM-RS symbol locations in these standard lookup tables.

Data Types: `double`

`DMRSPortSet` — DM-RS antenna ports
`[]` (default) | integer scalar | vector of nonnegative integers

DM-RS antenna ports, specified as one of these options.

- Integer from 0 to 11 — For a single antenna port
- Vector of nonnegative integers from 0 to 11 — For multiple antenna ports

Nominal antenna ports supported depend on `DMRSLength` and `DMRSConfigurationType` property values.

DMRSLength Value	DMRSConfigurationType Value	Nominal Range of Antenna Ports Supported
1	1	[0, 3]
	2	[0, 5]
2	1	[0, 7]
	2	[0, 11]

The default value, `[]`, implies that `DMRSPortSet` is in the range from 0 to `NumLayers`– 1, where `NumLayers` is a property of `nrPDSCHConfig` or `nrWavegenPDSCHConfig`.

Data Types: `double`

`NIDNSCID` — DM-RS scrambling identities
`[]` (default) | 1-by-2 integer vector | scalar integer

DM-RS scrambling identities (NID^0 and NID^1), specified as one of these options.

- 1-by-2 integer vector of values from 0 to 65,535 — The vector elements define NID^0 and NID^1.
- Scalar integer from 0 to 65,535 — This option specifies equal values for both NID^0 and NID^1.
- `[]` — Use this option to set the DM-RS scrambling identity to the physical layer cell identity, specified by the `NCellID` property of the carrier configuration.

Data Types: `double`

`NSCID` — DM-RS scrambling initialization
`0` (default) | `1`

DM-RS scrambling initialization, specified as `0` or `1`.

Data Types: `double`

`NumCDMGroupsWithoutData` — Number of DM-RS CDM groups without data
`2` (default) | `1` | `3`

Number of DM-RS CDM groups without data, specified as `1`, `2`, or `3`.

Each value indicates a different set of CDM group numbers, according to TS 38.214 Section 5.1.6.2.

- `1` — CDM group number 0
- `2` — CDM group numbers 0 and 1
- `3` — CDM group numbers 0, 1, and 2

Data Types: `double`

`DMRSDownlinkR16` — Low PAPR DM-RS sequence
`0` (default) | `1`

Low peak-to-average-power ratio (PAPR) DM-RS sequence, specified as one of these logical values.

- `0` (`false`) — Disable the use of low PAPR DM-RS.
- `1` (`true`) — Enable the use of low PAPR DM-RS. The DM-RS sequence generation depends on the DM-RS antenna port indices (`DMRSPortSet`) as well as the scrambling initialization (`NSCID`).

Data Types: `double`

`CDMGroups` — CDM group numbers corresponding to each DM-RS port
`0` (default) | integer from 0 to 2 | row vector of integers

This property is read-only.

CDM group numbers corresponding to each DM-RS port, specified as one of these options.

- Integer from 0 to 2 — When `DMRSPortSet` property is a scalar, specifying single DM-RS antenna port.
- Row vector of integers from 0 to 2 — When `DMRSPortSet` property is a vector, specifying multiple DM-RS antenna ports. Each element corresponds to a CDM group number for that port.

Value of this property depends on the `DMRSConfigurationType` property according to TS 38.211 Table 7.4.1.1.2-1 or 7.4.1.1.2-2 [1].

Data Types: `double`

`DeltaShifts` — Delta shifts corresponding to each CDM group
`0` (default) | integer from the set {0, 1, 2, 4} | row vector of integers

This property is read-only.

Delta shifts corresponding to each CDM group, specified as one of these options.

- Integer from the set {0, 1, 2, 4} — When `DMRSPortSet` property is a scalar, specifying single DM-RS antenna port.
- Row vector of integers from the set {0, 1, 2, 4} — When `DMRSPortSet` property is a vector, specifying multiple DM-RS antenna ports. Each element corresponds to the delta shift to be applied for that port.

Value of this property depends on the `DMRSConfigurationType` according to TS 38.211 Table 7.4.1.1.2-1 or 7.4.1.1.2-2 [1].

Data Types: `double`

`FrequencyWeights` — Frequency weights
`[1; 1]` (default) | column vector of integers | matrix of integers

This property is read-only.

Frequency weights for the DM-RS symbols, specified as one of these options.

- Column vector of integers — When `DMRSPortSet` property is a scalar, specifying single DM-RS antenna port.
- Matrix of integers — When `DMRSPortSet` property is a vector, specifying multiple DM-RS antenna ports. Each column corresponds to the weights for that port.

Value of this property depends on the `DMRSConfigurationType` according to TS 38.211 Table 7.4.1.1.2-1 or 7.4.1.1.2-2 [1].

Data Types: `double`

`TimeWeights` — Time weights
`[1; 1]` (default) | column vector of integers | matrix of integers

This property is read-only.

Time weights for to the DM-RS symbols, specified as one of these options.

- Column vector of integers — When `DMRSPortSet` property is a scalar, specifying single DM-RS antenna port.
- Matrix of integers — When `DMRSPortSet` property is a vector, specifying multiple DM-RS antenna ports. Each column corresponds to the weights for that port.

Value of this property depends on the `DMRSConfigurationType` according to TS 38.211 Table 7.4.1.1.2-1 or 7.4.1.1.2-2 [1].

Data Types: `double`

`DMRSSubcarrierLocations` — Subcarrier locations in RB for each port
`[0; 2; 4; 6; 8; 10]` (default) | column vector of integers | matrix of integers

This property is read-only.

Subcarrier locations in an RB for each port, specified as one of these options.

- Column vector of integers — When `DMRSPortSet` property is a scalar, specifying single DM-RS antenna port.
- Matrix of integers — When `DMRSPortSet` property is a vector, specifying multiple DM-RS antenna ports. Each column corresponds to the subcarrier locations for that port.

Data Types: `double`

`CDMLengths` — CDM arrangement for reference signals
`[1 1]` (default) | two-element row vector

This property is read-only.

CDM arrangement for reference signals, specified as the comma-separated pair consisting of `'CDMLengths'` and a two-element row vector of nonnegative integers [*FD TD*]. Array elements *FD* and *TD* specify the length of CDM despreading in the frequency domain (FD-CDM) and time domain (TD-CDM), respectively. A value of `1` for an element specifies no CDM.

Data Types: `double`

Examples

Create PDSCH DM-RS Object

Create a physical downlink shared channel (PDSCH) demodulation reference signal (DM-RS) object.

Specify a single-symbol DMRS with configuration type as 2, number of DM-RS additional positions as 2, and antenna ports as 0, 1, and 3.

View the corresponding properties.

```
dmrs = nrPDSCHDMRSConfig;
dmrs.DMRSConfigurationType = 2;
dmrs.DMRSLength = 1;
dmrs.DMRSAdditionalPosition = 2;
dmrs.DMRSPortSet = [0 1 3];
dmrs
```

```
dmrs = 
  nrPDSCHDMRSConfig with properties:

      DMRSConfigurationType: 2
         DMRSReferencePoint: 'CRB0'
          DMRSTypeAPosition: 2
     DMRSAdditionalPosition: 2
                 DMRSLength: 1
            CustomSymbolSet: []
                DMRSPortSet: [0 1 3]
                   NIDNSCID: []
                      NSCID: 0
    NumCDMGroupsWithoutData: 2
            DMRSDownlinkR16: 0

   Read-only properties:
                  CDMGroups: [0 0 1]
                DeltaShifts: [0 0 2]
          FrequencyWeights: [2x3 double]
               TimeWeights: [2x3 double]
    DMRSSubcarrierLocations: [4x3 double]
                 CDMLengths: [2 1]
```

Generate PDSCH DM-RS Symbols and Indices

Create a carrier configuration object specifying the slot number as 10.

```
carrier = nrCarrierConfig('NSlot',10);
```

Create a physical downlink shared channel (PDSCH) configuration object, `pdsch`, with physical resource blocks (PRBs) allocated from 0 to 30.

```
pdsch = nrPDSCHConfig;
pdsch.PRBSet = 0:30;
```

Create a PDSCH demodulation reference signal (DM-RS) object, `dmrs`, with specified properties.

```
dmrs = nrPDSCHDMRSConfig;
dmrs.DMRSConfigurationType = 2;
dmrs.DMRSLength = 2;
dmrs.DMRSAdditionalPosition = 1;
dmrs.DMRSTypeAPosition = 2;
dmrs.DMRSPortSet = 5;
```

```
dmrs.NIDNSCID = 10;
dmrs.NSCID = 0;
```

Assign the PDSCH DM-RS configuration object to DMRS property of PDSCH configuration object.

```
pdsch.DMRS = dmrs;
```

Generate PDSCH DM-RS symbols and indices for the specified carrier, PDSCH configuration, and output formatting `name-value` pair argument.

```
sym = nrPDSCHDMRS(carrier,pdsch,'OutputDataType','single')
```

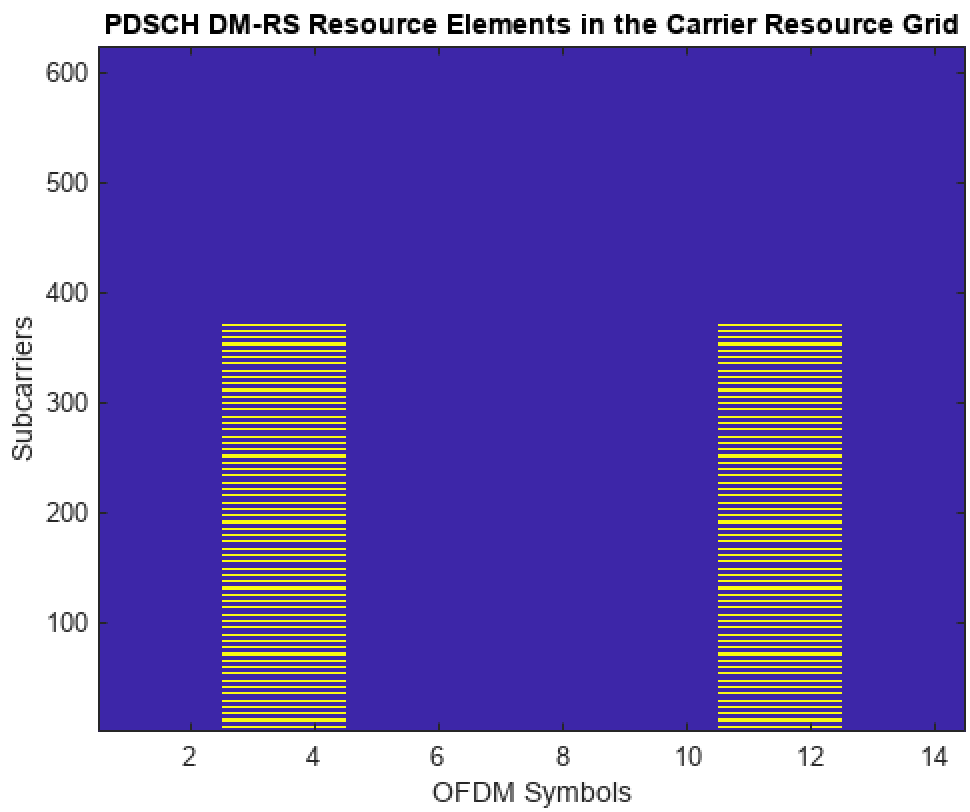
```
sym = 496x1 single column vector

  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
       ⋮
```

```
ind = nrPDSCHDMRSIndices(carrier,pdsch,'IndexBase','0based','IndexOrientation','carrier')
```

```
ind = 496x1 uint32 column vector

   1252
   1253
   1258
   1259
   1264
   1265
   1270
   1271
   1276
   1277
      ⋮
```

Display the generated DM-RS symbols on the carrier resource grid.

```
grid = complex(zeros([carrier.NSizeGrid*12 carrier.SymbolsPerSlot pdsch.NumLayers]));
grid(ind+1) = sym;
imagesc(abs(grid(:,:,1)));
axis xy;
xlabel('OFDM Symbols');
ylabel('Subcarriers');
title('PDSCH DM-RS Resource Elements in the Carrier Resource Grid');
```



Version History

Introduced in R2020a

Release 16 updates for low PAPR sequences, DM-RS scrambling identities, and type B mapping

- The `DMRSDownlinkR16` property enables low peak-to-average-power ratio (PAPR) physical downlink shared channel (PDSCH) DM-RS sequences, as defined in Release 16 of TS 38.211 Section 7.4.1.1.
- The `NIDNSCID` property supports dynamic ID selection.
- PDSCH DM-RS symbol generation now supports type B mapping DM-RS symbol positions, as defined in Release 16 of TS 38.211 Tables 7.4.1.1.2-3 and 7.4.1.1.2-4.

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*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrPDSCHDMRS` | `nrPDSCHDMRSIndices` | `nrPDSCH`

Objects
`nrCarrierConfig` | `nrPDSCHConfig` | `nrPDSCHPTRSConfig`

nrPDSCHPTRSConfig

PDSCH PT-RS configuration parameters

Description

The `nrPDSCHPTRSConfig` object sets phase tracking reference signal (PT-RS) configuration parameters for a physical downlink shared channel (PDSCH), as defined in TS 38.211 Section 7.4.1.2 [1]. By default, the object defines the PT-RS with time density 1, frequency density 2, resource element offset '00' and PTRS port set [ ]. Use this object when setting the `PTRS` property of the `nrPDSCHConfig` or `nrWavegenPDSCHConfig` objects.

Creation

Syntax

```
ptrs = nrPDSCHPTRSConfig
ptrs = nrPDSCHPTRSConfig(Name,Value)
```

Description

`ptrs = nrPDSCHPTRSConfig` creates a PT-RS configuration object for a PDSCH with default properties.

`ptrs = nrPDSCHPTRSConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'TimeDensity',2,'FrequencyDensity',4` sets the time density to 2 and frequency density to 4. Unspecified properties take their default values.

Properties

`TimeDensity` — PT-RS time density
`1` (default) | `2` | `4`

PT-RS time density, specified as `1`, `2` or `4`. This property is the higher layer parameter *timeDensity*.

Data Types: `double`

`FrequencyDensity` — PT-RS frequency density
`2` (default) | `4`

PT-RS frequency density, specified as `2` or `4`. This property is the higher layer parameter *frequencyDensity*.

Data Types: `double`

`REOffset` — Resource element offset
`'00'` (default) | `'01'` | `'10'` | `'11'`

Resource element offset with a specific subcarrier offset, specified as `'00'`, `'01'`,`'10'`, or `'11'`. This property is the higher layer parameter *resourceElementOffset*.

Data Types: `char` | `string`

`PTRSPortSet` — PT-RS antenna port set
`[]` (default) | nonnegative integer

PT-RS antenna port set, specified as a nonnegative integer. Specify `[]` to set this property to the lowest value in the `DMRSPortSet` property of `nrPDSCHDMRSConfig` object. This usage of `[]` value is applicable only when `nrPDSCHPTRSConfig` object is used as a property of `nrPDSCHConfig` or `nrWavegenPDSCHConfig`.

Data Types: `double`

Examples

Create PDSCH PT-RS Object

Create a PT-RS configuration object for a PDSCH. Set the time density to 2, frequency density to 4, and resource element offset to '10'.

```
ptrs = nrPDSCHPTRSConfig;
ptrs.TimeDensity = 2;
ptrs.FrequencyDensity = 4;
ptrs.REOffset = '10';
disp(ptrs)
```

```
  nrPDSCHPTRSConfig with properties:

         TimeDensity: 2
    FrequencyDensity: 4
            REOffset: '10'
         PTRSPortSet: []
```

Generate PDSCH PT-RS Symbols and Indices

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier.

```
carrier = nrCarrierConfig;
```

Create a default PDSCH configuration object, and then enable the PT-RS configuration.

```
pdsch = nrPDSCHConfig;
pdsch.EnablePTRS = 1;
```

Create a PDSCH phase tracking reference signal (PT-RS) configuration object with specified properties.

```
ptrs = nrPDSCHPTRSConfig;
ptrs.TimeDensity = 2;
ptrs.FrequencyDensity = 4;
ptrs.REOffset = '10';
```

Assign the PDSCH PT-RS configuration object to PTRS property of PDSCH configuration object.

```
pdsch.PTRS = ptrs;
```

Generate PDSCH PT-RS symbols of data type single.

```
sym = nrPDSCHPTRS(carrier,pdsch,'OutputDataType','single')
```

```
sym = 78x1 single column vector

  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
   0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
      ⋮
```

Generate PDSCH PT-RS indices in subscript form and set the index orientation to bandwidth part.

```
ind = nrPDSCHPTRSIndices(carrier,pdsch,'IndexStyle','subscript','IndexOrientation','bwp')
```

```
ind = 78x3 uint32 matrix

    19      1      1
    67      1      1
   115      1      1
   163      1      1
   211      1      1
   259      1      1
   307      1      1
   355      1      1
   403      1      1
   451      1      1
      ⋮
```

Version History

Introduced in R2020a

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions
`nrPDSCHIndices` | `nrPDSCHPTRS` | `nrPDSCHPTRSIndices`

Objects
`nrCarrierConfig` | `nrPDSCHConfig` | `nrPDSCHReservedConfig` | `nrPDSCHDMRSConfig`

nrPDSCHReservedConfig

PDSCH reserved PRB configuration parameters

Description

The `nrPDSCHReservedConfig` object sets physical downlink shared channel (PDSCH) reserved physical resource block (PRB) configuration parameters, as defined in TS 38.214 Section 5.1.4.1 [1].

The object configures the reserved PRB pattern for the PDSCH. By default, the object configures the empty reserved PRB pattern.

Creation

Syntax

```
reservedPRB = nrPDSCHReservedConfig
reservedPRB = nrPDSCHReservedConfig(Name,Value)
```

Description

`reservedPRB = nrPDSCHReservedConfig` creates a PDSCH reserved PRB configuration object with default properties.

`reservedPRB = nrPDSCHReservedConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'SymbolSet',(0:3),'Period',5` specifies the OFDM symbols associated with reserved PRBs as (0:3) and the period of the total number of slots in the pattern as 5. Unspecified properties take their default values.

Properties

`PRBSet` — Reserved PRB indices within BWP
`[]` (default) | vector of nonnegative integer

Reserved PRB indices within the bandwidth part (BWP), specified as a nonnegative integer vector. The PRB indices are 0-based. If this property is `[]` value, no reserved PRBs are defined.

Data Types: `double`

`SymbolSet` — OFDM symbols associated with reserved PRBs
`[]` (default) | vector of nonnegative integer

OFDM symbols associated with reserved PRBs spanning over one or more slots, specified a nonnegative integer vector. The symbol indices are 0-based. If this property is `[]` value, no reserved OFDM symbols are defined.

Data Types: `double`

`Period` — Period of number of slots in pattern

[ ] (default) | positive integer

Period of the number of slots in the pattern, specified as a positive integer. The SymbolSet property specifies provides the entire OFDM symbols pattern. This pattern repeats itself for every `Period` slots.

If this property is `[]` value, the OFDM symbols pattern does not cyclically repeat itself.

Data Types: `double`

Examples

Create PDSCH Object to Configure Two Reserved PRB Patterns

Create a PDSCH configuration object with two reserved PRB patterns.

```
pdsch = nrPDSCHConfig('ReservedPRB',{nrPDSCHReservedConfig,nrPDSCHReservedConfig});
```

For each reserved PRB pattern, specify the reserved PRB indices in the BWP, the OFDM symbols associated with those reserved PRBs, and the period for the number of slots in the pattern.

```
pdsch.ReservedPRB{1}.PRBSet = (0:15);
pdsch.ReservedPRB{1}.SymbolSet = (5:6);
pdsch.ReservedPRB{1}.Period = 5;
pdsch.ReservedPRB{2}.PRBSet = (0:23);
pdsch.ReservedPRB{2}.SymbolSet = [2:4 7:9];
pdsch.ReservedPRB{2}.Period = 3;
```

Display the two PRB patterns.

```
PRBPattern1 = pdsch.ReservedPRB{1}
```

```
PRBPattern1 = 
  nrPDSCHReservedConfig with properties:

       PRBSet: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]
    SymbolSet: [5 6]
       Period: 5
```

```
PRBPattern2 = pdsch.ReservedPRB{2}
```

```
PRBPattern2 = 
  nrPDSCHReservedConfig with properties:

       PRBSet: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
    SymbolSet: [2 3 4 7 8 9]
       Period: 3
```

Version History

Introduced in R2020a

References

[1] 3GPP TS 38.214. "NR; Physical layer procedures for data." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrPDSCH` | `nrPDSCHDecode`

Objects
`nrCarrierConfig` | `nrPDSCHConfig` | `nrPDSCHDMRSConfig` | `nrPDSCHPTRSConfig`

nrPRACHConfig

PRACH configuration parameters

Description

The `nrPRACHConfig` object sets physical random access channel (PRACH) configuration parameters for a PRACH preamble, as defined in TS 38.211 Section 5.3.2 and Section 6.3.3 [1]. The default PRACH configuration corresponds to a PRACH preamble format 0 placed at the start of the allocated resources that is active in all subframes for frequency range 1 (FR1) and frequency division duplex (FDD) mode for paired spectrum.

Creation

Syntax

```
prach = nrPRACHConfig
prach = nrPRACHConfig(Name,Value)
```

Description

`prach = nrPRACHConfig` creates a PRACH configuration object with default properties.

`prach = nrPRACHConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'ConfigurationIndex',106,'SubcarrierSpacing',30` specifies the time resource and subcarrier spacing for the PRACH preamble. Unspecified properties take their default values.

Properties

Configurable PRACH Properties

`FrequencyRange` — Frequency range
`'FR1'` (default) | `'FR2'`

Frequency range, specified as `'FR1'` or `'FR2'`.

Use this property together with the `DuplexMode` property to specify these PRACH configuration tables from TS 38.211.

- To specify Table 6.3.3.2-2, set `FrequencyRange` to `'FR1'` and `DuplexMode` to `'FDD'` or `'SUL'`.
- To specify Table 6.3.3.2-3, set `FrequencyRange` to `'FR1'` and `DuplexMode` to `'TDD'`.
- To specify Table 6.3.3.2-4, set `FrequencyRange` to `'FR2'` and `DuplexMode` to `'TDD'`.

Data Types: `char` | `string`

`DuplexMode` — Duplex mode for uplink transmission
`'FDD'` (default) | `'TDD'` | `'SUL'`

Duplex mode for uplink transmission, specified as one of these values:

- 'FDD' — Use this value to specify FDD mode for paired spectrum.
- 'TDD' — Use this value to specify time division duplex (TDD) mode for unpaired spectrum.
- 'SUL' — Use this value to specify supplementary uplink.

Use this property together with the FrequencyRange property to specify these PRACH configuration tables from TS 38.211:

- To specify Table 6.3.3.2-2, set FrequencyRange to 'FR1' and DuplexMode to 'FDD' or 'SUL'.
- To specify Table 6.3.3.2-3, set FrequencyRange to 'FR1' and DuplexMode to 'TDD'.
- To specify Table 6.3.3.2-4, set FrequencyRange to 'FR2' and DuplexMode to 'TDD'.

Data Types: char | string

ConfigurationIndex — Time resource of PRACH preamble
27 (default) | integer from 0 to 262

Time resource of PRACH preamble, specified as an integer from 0 to 262. This property specifies a configuration index from Tables 6.3.3.2-2 to 6.3.3.2-4 in TS 38.211. The FrequencyRange and DuplexMode properties determine the actual configuration table to consider. When you set FrequencyRange to 'FR1' and DuplexMode to 'FDD' or when you set FrequencyRange to 'FR2', the property value must be an integer from 0 to 255.

This property corresponds to the *prach-ConfigurationIndex* and *prach-ConfigurationIndex-v1610* higher layer parameters.

Data Types: double

SubcarrierSpacing — Subcarrier spacing for PRACH in kHz
1.25 (default) | 5 | 15 | 30 | 60 | 120

Subcarrier spacing for the PRACH in kHz, specified as 1.25, 5, 15, or 30 for frequency range FR1 and 60 or 120 for frequency range FR2.

Set this property in relation to the preamble format property Format. To identify valid preamble format and subcarrier spacing combinations, see the LongPreambleFormats and ShortPreambleFormats fields of the Tables property. For more information, see Table 6.3.3.1-1 for long preambles and Table 6.3.3.1-2 for short preambles.

Data Types: double

LRA — Length of Zadoff-Chu preamble sequence
139 | 571 | 839 | 1151

Length of Zadoff-Chu preamble sequence, specified as 139, 571, 839, or 1151. The default value depends on the Format property.

- For long preambles (formats 0, 1, 2, and 3), the default value is 839. You cannot change the property value from its default value.
- For short preambles (formats A1, A2, A3, B1, B2, B3, B4, C0, and C2), the default value is 139.
  - When you set the SubcarrierSpacing property to 1.25, 5, 60, or 120, you cannot change the property value from its default value.

- When you set the `SubcarrierSpacing` property to `15`, you can set this property to `139` or `1151`.
- When you set the `SubcarrierSpacing` property to `30`, you can set this property to `139` or `571`.

This property corresponds to the L_{RA} parameter from TS 38.211 Section 6.3.3.

Data Types: `double`

`SequenceIndex` — Logical root sequence index
`0` (default) | integer from 0 to 1149

Logical root sequence index, specified as an integer from 0 to 1149.

This property corresponds to the *prach-RootSequenceIndex-r16* and *prach-RootSequenceIndex* higher layer parameters and *i* parameter from TS 38.211 Tables 6.3.3.1-3, 6.3.3.1-4, 6.3.3.1-4A, and 6.3.3.1-4B.

Data Types: `double`

`PreambleIndex` — Preamble index within cell
`0` (default) | integer from 0 to 63

Preamble index within the cell, specified as an integer from 0 to 63.

This property is the higher layer parameter *ra-PreambleIndex*.

Data Types: `double`

`RestrictedSet` — Type of restricted set
`'UnrestrictedSet'` (default) | `'RestrictedSetTypeA'` | `'RestrictedSetTypeB'`

Type of restricted set, specified as `'UnrestrictedSet'`, `'RestrictedSetTypeA'`, or `'RestrictedSetTypeB'`. Set this property in relation to the cyclic shift configuration index property `ZeroCorrelationZone`, as defined by N_{CS} in Tables 6.3.3.1-5 to 6.3.3.1-7 from TS 38.211.

Data Types: `char` | `string`

`ZeroCorrelationZone` — Cyclic shift configuration index
`0` (default) | integer from 0 to 15

Cyclic shift configuration index, specified as an integer from 0 to 15. Use this property together with the `RestrictedSet` and `SubcarrierSpacing` properties to retrieve the number of cyclic shifts for the sequence generation. For more information, see TS 38.211 Tables 6.3.3.1-5 to 6.3.3.1-7.

Data Types: `double`

`RBOffset` — Starting RB index of initial uplink BWP
`0` (default) | integer from 0 to 274

Starting resource block (RB) index of the initial uplink bandwidth part (BWP), relative to the carrier resource grid, specified as an integer from 0 to 274.

Data Types: `double`

`FrequencyStart` — Offset of lowest PRACH transmission occasion
`0` (default) | integer from 0 to 274

Offset of lowest PRACH transmission occasion, in frequency domain, relative to the physical resource block (PRB) 0, specified as an integer from 0 to 274.

This property corresponds to parameter n_{RA}^{start} in TS 38.211 Section 5.3.2 and is the higher layer parameter *msg1-FrequencyStart*.

Data Types: `double`

`RBSetOffset` — Starting RB index of uplink RB set
`0` (default) | integer from 0 to 274

Starting RB index of the uplink RB set for the configured PRACH transmission occasion, specified as an integer from 0 to 274. This property determines the PRACH indices and corresponds to $N_{\mathrm{RB,UL},\,n_0+n_{\mathrm{RA}}}^{\mathrm{start},\,\mu} - N_{\mathrm{RB,UL},\,n_0}^{\mathrm{start},\,\mu}$ in TS 38.211 Section 5.3.2.

Dependencies

To enable this property, set the `LRA` property to `571` or `1151`.

Data Types: `double`

`FrequencyIndex` — Index of PRACH transmission occasion
`0` (default) | integer from 0 to 7

Index of PRACH transmission occasion, in frequency domain, specified as an integer from 0 to 7. The frequency index must be in the range from 0 to $M - 1$, where M is 1, 2, 4, or 8.

This property corresponds to parameter n_{RA} in TS 38.211 Sections 5.3.2 and 6.3.3.2 and is the higher layer parameter *msg1-FDM* defined in TS 38.331 Section 6.3.2.

Dependencies

To enable this property, set the `LRA` property to `139` or `839`.

Data Types: `double`

`TimeIndex` — Index of PRACH transmission occasion
`0` (default) | integer from 0 to 6

Index of the PRACH transmission occasion, in time domain, specified as an integer from 0 to 6. Set this property in relation to the `LRA` property.

- When `LRA` is `839`, set `TimeIndex` to `0`.
- When `LRA` is `139`, `571`, or `1151`, set `TimeIndex` to an integer from 0 to `NumTimeOccasions` − 1.

This property corresponds to parameter n_t^{RA} in TS 38.211 Section 5.3.2.

Data Types: `double`

`ActivePRACHSlot` — Position of active PRACH slot within subframe or 60 kHz slot
`0` (default) | `1`

Position of active PRACH slot within a subframe (for FR1) or a 60 kHz slot (for FR2), specified as `0` or `1`.

- If the SubcarrierSpacing property is set to 1.25, 5, 15, or 60, then ActivePRACHSlot must be 0.
- If SubcarrierSpacing is set to 30 or 120, then ActivePRACHSlot must be set based on configuration tables TS 38.211 Table 6.3.3.2-2 to Table 6.3.3.2-4.

To specify the frequency range of the carrier as FR1 or FR2, use the FrequencyRange property.

This property corresponds to parameter n_{slot}^{RA} in TS 38.211 Section 5.3.2.

Data Types: double

NPRACHSlot — PRACH slot number
0 (default) | nonnegative integer

PRACH slot number, specified as a nonnegative integer. You can set NPRACHSlot to a value larger than the number of slots per frame. For example, you can set this value using transmission loop counters in a MATLAB simulation. In this case, you might have to ensure that the property value is modulo the number of slots per frame in a calling code.

Data Types: double

Nonconfigurable PRACH Properties

The object automatically sets these properties based on configurable PRACH property values by using the configuration tables from TS 38.211 Section 6.3.3.

Format — Preamble format
'0' | '1' | '2' | '3' | 'A1' | 'A2' | 'A3' | 'B1' | 'B2' | 'B3' | 'B4' | 'C0' | 'C2'

This property is read-only.

Preamble format, defined in TS 38.211 Tables 6.3.3.1-1 and 6.3.3.1-2, returned as '0', '1', '2', '3', 'A1', 'A2', 'A3', 'B1', 'B2', 'B3', 'B4', 'C0', or 'C2'.

For short preamble format C0, each preamble has one active sequence period. Because the preamble spans two OFDM symbols, including the guard and the cyclic prefix, the grid related to format C0 has 7 OFDM symbols instead of 14.

Data Types: char | string

NumTimeOccasions — Number of time-domain PRACH occasions within PRACH slot
integer from 0 to 7

This property is read-only.

Number of time-domain PRACH occasions within a PRACH slot, returned as an integer from 0 to 7. For long preambles, NumTimeOccasions is always 1. For more details, see TS 38.211 Section 5.3.2.

This property corresponds to parameter $N_t^{RA,slot}$ in TS 38.211 Tables 6.3.3.2-2 to 6.3.3.2-4.

Data Types: double

PRACHDuration — Number of OFDM symbols in PRACH slot grid
integer from 1 to 12

This property is read-only.

Number of OFDM symbols in the PRACH slot grid, corresponding to one transmission occasion, returned as an integer from 1 to 12.

This property corresponds to parameter $N_{dur}^{RA, slot}$ in TS 38.211 Tables 6.3.3.2-2 to 6.3.3.2-4. For format C0, because the grid has 7 OFDM symbols instead of 14, the object sets `PRACHDuration` to $N_{dur}^{RA, slot}$ / 2.

For long preamble formats 0 and 1, `PRACHDuration` is `1` and `2`, respectively. For long preamble formats 2 and 3, `PRACHDuration` is `4`. For more information on long preamble formats, see Table 6.3.3.1-1.

Data Types: `double`

`SymbolLocation` — First OFDM symbol location in current PRACH occasion
integer from 0 to 26

This property is read-only.

First OFDM symbol location in current PRACH occasion within a slot, returned as an integer from 0 to 26. If the `ActivePRACHSlot` property is set to `1`, this location can fall outside a PRACH slot.

This property corresponds to parameter l in TS 38.211 Section 5.3.2 with these exceptions.

- For format C0, because the grid has 7 OFDM symbols instead of 14, the object sets `SymbolLocation` to l / 2.
- For long preamble formats characterized by starting symbol location 7 in Table 6.3.3.2-3, the object sets the `SymbolLocation` to `0`.

Data Types: `double`

`SubframesPerPRACHSlot` — Total number of subframes per nominal PRACH slot
`0.125` | `0.25` | `0.5` | `1` | `3` | `4`

This property is read-only.

Total number of subframes per nominal PRACH slot, returned as `0.125`, `0.25`, `0.5`, `1`, `3`, or `4`.

Data Types: `double`

`PRACHSlotsPerPeriod` — Number of PRACH slots per overall period
`5` | `10` | `20` | `40` | `80` | `160` | `320` | `640`

This property is read-only.

Number of PRACH slots per overall period, returned as `5`, `10`, `20`, `40`, `80`, `160`, `320`, or `640`. The overall period spans an integer multiple of x frames, where x is defined in TS 38.211 Tables 6.3.3.2-2, 6.3.3.2-3, and 6.3.3.2-4.

Data Types: `double`

PRACH Lookup Tables

`Tables` — PRACH configuration tables
constant structure

This property is read-only.

PRACH configuration tables, from TS 38.211 Section 6.3.3, returned as a constant structure containing these fields:

Fields	Values	Description
LongPreambleFormats	4-by-6 table	Table 6.3.3.1-1: Long PRACH preamble formats
ShortPreambleFormats	9-by-8 table	Table 6.3.3.1-2: Short PRACH preamble formats
NCSFormat012	16-by-4 table	Table 6.3.3.1-5: N_{CS} for long preamble formats with 1.25 kHz subcarrier spacing
NCSFormat3	16-by-4 table	Table 6.3.3.1-6: N_{CS} for long preamble formats with 5 kHz subcarrier spacing
NCSFormatABC	16-by-4 table	Table 6.3.3.1-7: N_{CS} for short preamble formats
SupportedSCSCombinations	22-by-5 table	Table 6.3.3.2-1: Supported combinations of subcarrier spacing for the PRACH and the physical uplink shared channel (PUSCH)
ConfigurationsFR1PairedSUL	263-by-9 table	Table 6.3.3.2-2: PRACH configurations for FR1 and paired spectrum or FR1 and supplementary uplink
ConfigurationsFR1Unpaired	263-by-9 table	Table 6.3.3.2-3: PRACH configurations for FR1 and unpaired spectrum
ConfigurationsFR2	263-by-9 table	Table 6.3.3.2-4: PRACH configurations for FR2 and unpaired spectrum

Invalid PRACH Configurations

Based on the configuration tables in TS 38.211 Section 6.3.3, these property setting combinations and scenarios lead to invalid PRACH configurations.

- Setting `'FrequencyRange'` to `'FR2'` and `'DuplexMode'` to `'FDD'` is invalid.
- Setting `'FrequencyRange'` to `'FR2'` and `'DuplexMode'` to `'SUL'` is invalid.
- Setting `'FrequencyRange'` to `'FR1'` and `'SubcarrierSpacing'` to `60` is invalid.
- Setting `'FrequencyRange'` to `'FR1'` and `'SubcarrierSpacing'` to `120` is invalid.
- Setting `'FrequencyRange'` to `'FR2'` and `'SubcarrierSpacing'` to `1.25` is invalid.
- Setting `'FrequencyRange'` to `'FR2'` and `'SubcarrierSpacing'` to `5` is invalid.
- Setting `'FrequencyRange'` to `'FR2'` and `'SubcarrierSpacing'` to `15` is invalid.
- Setting `'FrequencyRange'` to `'FR2'` and `'SubcarrierSpacing'` to `30` is invalid.
- Any combination of properties `Format` and `SubcarrierSpacing` not listed in Table 6.3.3.1-1 for long preambles or Table 6.3.3.1-2 for short preambles is invalid. You can identify valid combinations in the `LongPreambleFormats` and `ShortPreambleFormats` fields of the `Tables` property.
- Any combination of properties `ZeroCorrelationZone` and `RestrictedSet` not listed in Tables 6.3.3.1-5, 6.3.3.1-6, and 6.3.3.1-7 is invalid. You can identify valid combinations in the `NCSFormat012`, `NCSFormat3`, and `NCSFormatABC` fields, respectively, of the `Tables` property.
- Any combination of properties `ActivePRACHSlot`, `FrequencyRange`, `DuplexMode`, `ConfigurationIndex`, and `SubcarrierSpacing` not covered in Section 5.3.2 is invalid.

- Any configuration where `TimeIndex` ≥ `NumTimeOccasions` is invalid.

Examples

Configure PRACH Preamble Format

Create a PRACH configuration object with default properties. The default configuration object defines a PRACH configuration with long preamble format 0, based on TS 38.211 Table 6.3.3.2-2.

```
prach = nrPRACHConfig;
```

To consider a different PRACH configuration table as a basis, for example Table 6.3.3.2-3 for FR1 and unpaired spectrum, update the duplex mode property.

```
prach.DuplexMode = 'TDD';
```

To change the PRACH preamble format, you must update the `ConfigurationIndex` property of the object based on Table 6.3.3.2-3. To lookup a suitable value, access this table through the `ConfigurationsFR1Unpaired` field of the `Tables` property.

```
prach.Tables.ConfigurationsFR1Unpaired(:,:)
```

```
ans=263×9 table
    ConfigurationIndex    PreambleFormat    x      y      SubframeNumber    StartingSymbol    PI
    __________________    ______________    __    _____    ______________    ______________    _

            0                 {'0'}         16    {[1]}      {[   9]}              0
            1                 {'0'}          8    {[1]}      {[   9]}              0
            2                 {'0'}          4    {[1]}      {[   9]}              0
            3                 {'0'}          2    {[0]}      {[   9]}              0
            4                 {'0'}          2    {[1]}      {[   9]}              0
            5                 {'0'}          2    {[0]}      {[   4]}              0
            6                 {'0'}          2    {[1]}      {[   4]}              0
            7                 {'0'}          1    {[0]}      {[   9]}              0
            8                 {'0'}          1    {[0]}      {[   8]}              0
            9                 {'0'}          1    {[0]}      {[   7]}              0
           10                 {'0'}          1    {[0]}      {[   6]}              0
           11                 {'0'}          1    {[0]}      {[   5]}              0
           12                 {'0'}          1    {[0]}      {[   4]}              0
           13                 {'0'}          1    {[0]}      {[   3]}              0
           14                 {'0'}          1    {[0]}      {[   2]}              0
           15                 {'0'}          1    {[0]}      {[1 6]}              0
      ⋮
```

To change the preamble from format 0 to format A1, set the `ConfigurationIndex` property to any value from 67 to 86.

```
prach.ConfigurationIndex = 86;
```

Verify that the object updates the preamble format correctly.

```
isequal(prach.Format,'A1')
```

```
ans = logical
   1
```

Version History

Introduced in R2020a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`nrPRACH` | `nrPRACHGrid` | `nrPRACHIndices`

Topics

"5G NR PRACH Configuration"

nrPRSConfig

PRS configuration parameters

Description

The `nrPRSConfig` object sets the positioning reference signal (PRS) configuration parameters, as defined in TS 38.211 Section 7.4.1.7 [1]. By default, the object defines a PRS resource set with a single PRS resource occupying 52 resource blocks and spanning over the first 12 orthogonal frequency division multiplexing (OFDM) symbols per slot. The default object allocates the complete bandwidth for PRS resource, when used in combination with a default `nrCarrierConfig` object.

Creation

Syntax

```
prs = nrPRSConfig
prs = nrPRSConfig(Name,Value)
```

Description

`prs = nrPRSConfig` creates a default PRS configuration object.

`prs = nrPRSConfig(Name,Value)` specifies properties on page 3-98 using one or more name-value arguments. Enclose each property name in quotes. For example, `'PRSResourceSetPeriod','on'` specifies that all of the PRS resources are present in the operating slot.

Properties

`PRSResourceSetPeriod` — PRS resource set slot periodicity and slot offset
`'on'` (default) | `'off'` | two-element vector

PRS resource set slot periodicity and slot offset, specified as one of these options.

- `'on'` — All of the PRS resources are present in the operating slot.
- `'off'` — All of the PRS resources are absent in the operating slot.
- Two-element vector of the form [*TPRSPeriod*, *TPRSOffset*] — *TPRSPeriod* is the resource set slot periodicity. The nominal value of *TPRSPeriod* must equal 2^{μ} multiplied by one of the values in the set {4, 5, 8, 10, 16, 20, 32, 40, 64, 80, 160, 320, 640, 1280, 2560, 5120, 10,240}, where μ is the subcarrier spacing configuration with a value of 0, 1, 2, or 3. *TPRSOffset* is the resource set slot offset and must equal a value in the range [0, *TPRSPeriod* – 1].

Data Types: `double` | `string` | `char`

`PRSResourceOffset` — Slot offset of each PRS resource
`0` (default) | scalar in the range [0, 511] | vector of integers in the range [0, 511]

Slot offset of each PRS resource (0-based) provided by the higher layer parameter *dl-PRS-ResourceSlotOffset-r16*, specified as a scalar in the range [0, 511] or a vector of integers in the range [0, 511]. This property represents the starting slot offset of a PRS resource relative to the PRS resource set offset (*TPRSOffset*).

Configure this property for each resource in a resource set separately based on one of these options.

- When a single resource is present, specify this property as a scalar.
- When multiple resources are present, specify this property as a scalar or vector. If you specify a scalar, the object applies that value to all of the PRS resources in a PRS resource set. If you specify a vector, the object applies the vector element values to the corresponding PRS resource. The length of this vector must be equal to the number of PRS resources to be configured in a PRS resource set.

Dependencies

To enable this property, set the `PRSResourceSetPeriod` property to a two-element vector of the form [*TPRSPeriod*, *TPRSOffset*].

Data Types: `double`

`PRSResourceRepetition` — PRS resource repetition factor

1 (default) | 2 | 4 | 6 | 8 | 16 | 32

PRS resource repetition factor provided by the higher layer parameter *dl-PRS-ResourceRepetitionFactor-r16*, specified as `1`, `2`, `4`, `6`, `8`, `16`, or `32`. This property value is the same for all of the PRS resources in a PRS resource set.

Dependencies

To enable this property, set the `PRSResourceSetPeriod` property to a two-element vector of the form [*TPRSPeriod*, *TPRSOffset*].

Data Types: `double`

`PRSResourceTimeGap` — Slot offset between two consecutive repeated instances of a PRS resource

1 (default) | 2 | 4 | 8 | 16 | 32

Slot offset between two consecutive repeated instances of a PRS resource, specified as `1`, `2`, `4`, `8`, `16`, or `32`. The property represents the offset in terms of the number of slots between two repeated instances of a PRS resource. This property value is same the for all of the PRS resources in a PRS resource set. This property is the higher layer parameter *dl-PRS-ResourceTimeGap-r16*.

Dependencies

To enable this property, set the `PRSResourceSetPeriod` property to a two-element vector of the form [*TPRSPeriod*, *TPRSOffset*] and the `PRSResourceRepetition` property to a value greater than 1.

Data Types: `double`

`MutingPattern1` — Muting bit pattern option-1

`[]` (default) | binary-valued vector of length 2, 4, 6, 8, 16, or 32

Muting bit pattern option-1, specified as `[]` or a binary-valued vector of length 2, 4, 6, 8, 16, or 32.

- If the higher layer parameter *mutingOption1-r16* is configured, set this property to a binary-valued vector of length 2, 4, 6, 8, 16, or 32. Each element in the vector corresponds to a number of consecutive instances of a PRS resource set based on the `MutingBitRepetition` property. The vector element also indicates whether all of the PRS resources within the PRS resource set instances are transmitted (binary 1) or muted (binary 0).
- If the higher layer parameter *mutingOption1-r16* is not configured, set this property to `[]`. To disable the muting bit pattern option-1, set this property to `[]`.

Dependencies

To enable this property, set the `PRSResourceSetPeriod` property to a two-element vector of the form [*TPRSPeriod*, *TPRSOffset*].

Data Types: `double`

`MutingBitRepetition` — Muting bit repetition factor

`1` (default) | `2` | `4` | `8`

Muting bit repetition factor provided by the higher layer parameter *dl-PRS-MutingBitRepetitionFactor-r16*, specified as `1`, `2`, `4`, or `8`. This property indicates the number of consecutive instances of the PRS resource set, *N*, corresponding to each element of the `MutingPattern1` property. The first element in `MutingPattern1` corresponds to the first *N* instances of a PRS resource set, the second element corresponds to the next *N* instances of a PRS resource set, and so on.

Dependencies

To enable this property, set the `PRSResourceSetPeriod` property to a two-element vector of the form [*TPRSPeriod*, *TPRSOffset*] and the `MutingPattern1` property to a value other than `[]`.

Data Types: `double`

`MutingPattern2` — Muting bit pattern option-2

`[]` (default) | binary-valued vector of length 1, 2, 4, 6, 8, 16, or 32

Muting bit pattern option-2, specified as `[]` or a binary-valued vector of length 1, 2, 4, 6, 8, 16, or 32.

- If the higher layer parameter *mutingOption2-r16* is configured, set this property to a binary-valued vector of length 1, 2, 4, 6, 8, 16, or 32. Each element in the vector relates to the corresponding repetition index. Each element in the vector corresponds to a single repetition index of each PRS resource in an active instance of a PRS resource set. The vector element also indicates whether the repetition index for all of the PRS resources is transmitted (binary 1) or muted (binary 0).
- If the higher layer parameter *mutingOption2-r16* is not configured, set this property to `[]`. To disable the muting bit pattern option-2, set this property to `[]`.

Dependencies

To enable this property, set the `PRSResourceSetPeriod` property to a two-element vector of the form [*TPRSPeriod*, *TPRSOffset*].

Data Types: `double`

`NumPRSSymbols` — Number of consecutive OFDM symbols allocated for each PRS resource

`12` (default) | scalar in the range [0, 12] | vector of integers in the range [0, 12]

Number of consecutive OFDM symbols allocated for each PRS resource, specified as a scalar in the range [0, 12] or a vector of integers in the range [0, 12].

Configure this property for each resource in a resource set separately based on one of these options.

- When a single resource is present, specify this property as a scalar.
- When multiple resources are present, specify this property as a scalar or vector. If you specify a scalar, the object applies that value to all of the PRS resources in a PRS resource set. If you specify a vector, the object applies the vector element values to the corresponding PRS resource. The length of this vector must be equal to the number of PRS resources to be configured in a PRS resource set.

This property is the higher layer parameter *dl-PRS-NumSymbols-r16*, and the nominal value of this property must be `2`, `4`, `6`, or `12`. To indicate no PRS resource allocation, set this property to `0`.

Data Types: `double`

`SymbolStart` — Starting OFDM symbol of each PRS resource in slot
`0` (default) | scalar in the range [0, 13] | vector of integers in the range [0, 13]

Starting OFDM symbol of each PRS resource in a slot (0-based), specified as a scalar in the range [0, 13] or a vector of integers in the range [0, 13].

Configure this property for each resource in a resource set separately based on one of these options.

- When a single resource is present, specify this property as a scalar.
- When multiple resources are present, specify this property as a scalar or vector. If you specify a scalar, the object applies that value to all of the PRS resources in a PRS resource set. If you specify a vector, the object applies the vector element values to the corresponding PRS resource. The length of this vector must be equal to the number of PRS resources to be configured in a PRS resource set.

This property is the higher layer parameter *dl-PRS-ResourceSymbolOffset-r16*, and the nominal value of this property must be in the range [0, 12].

Data Types: `double`

`NumRB` — Number of PRBs allocated for all PRS resources
`52` (default) | scalar in the range [0, 275]

Number of PRBs allocated for all PRS resources in a resource set, specified as a scalar in the range [0, 275]. This property is the higher layer parameter *dl-PRS-ResourceBandwidth-r16*, and the nominal value of this property must be in the range [24, 272] with a granularity of 4 PRBs. To indicate no PRS resource allocation, set this property to `0`.

Data Types: `double`

`RBOffset` — Starting PRB index of all PRS resources relative to carrier resource grid
`0` (default) | scalar in the range [0, 274]

Starting PRB index of all PRS resources relative to the carrier resource grid, specified as a scalar in the range [0, 274].

Data Types: `double`

CombSize — Comb size of all PRS resources
2 (default) | 4 | 6 | 12

Comb size of all PRS resources in a resource set, specified as 2, 4, 6, or 12. The comb size represents the resource element spacing in each OFDM symbol. This property is the higher layer parameter *dl-PRS-CombSizeN-r16*. The value 2 specifies for the object to allocate every 2nd RE in the PRB for PRS, the value 4 specifies for the object to allocate every 4th RE in the PRB for PRS, and so on.

Data Types: double

REOffset — Starting RE offset in first PRS OFDM symbol of each PRS resource
0 (default) | scalar in the range [0, (CombSize–1)] | vector of integers in the range [0, (CombSize–1)]

Starting RE offset in the first PRS OFDM symbol of each PRS resource, specified as a scalar in the range [0, (CombSize–1)] or a vector of integers in the range [0, (CombSize–1)]. The relative RE offsets of the next PRS OFDM symbols are defined relative to the REOffset value, as described in TS 38.211 Table 7.4.1.7.3-1.

Configure this property for each resource in a resource set separately based on one of these options.

- When a single resource is present, specify this property as a scalar.
- When multiple resources are present, specify this property as a scalar or vector. If you specify a scalar, the object applies that value to all of the PRS resources in a PRS resource set. If you specify a vector, the object applies the vector element values to the corresponding PRS resource. The length of this vector must be equal to the number of PRS resources to be configured in a PRS resource set.

This property is the higher layer parameter *dl-PRS-ReOffset-r16*.

Data Types: double

NPRSID — Sequence identity of each PRS resource
0 (default) | scalar in the range [0, 4095] | vector of integers in the range [0, 4095]

Sequence identity of each PRS resource provided by the higher layer parameter *dl-PRS-SequenceID-r16*, specified as a scalar in the range [0, 4095] or a vector of integers in the range [0, 4095].

Configure this property for each resource in a resource set separately based on one of these options.

- When a single resource is present, specify this property as a scalar.
- When multiple resources are present, specify this property as a scalar or vector. If you specify a scalar, the object applies that value to all of the PRS resources in a PRS resource set. If you specify a vector, the object applies the vector element values to the corresponding PRS resource. The length of this vector must be equal to the number of PRS resources to be configured in a PRS resource set.

Data Types: double

FrequencyOffsetTable — Frequency offsets table
4-by-13 table

This property is read-only.

Frequency offsets table, specified as a 4-by-13 table. The table contains the relative resource element offsets in each PRS OFDM symbol defined relative to the REOffset property, according to TS 38.211

Table 7.4.1.7.3-1. The 4 rows in the table correspond to the four `CombSize` property values. The 13 columns in the table correspond to the OFDM symbol numbers within the PRS resource allocation (0 to 12).

The object automatically sets this property based on the TS 38.211 Table 7.4.1.7.3-1.

Data Types: `double`

Note The maximum vector length of these five properties specifies the number of configured PRS resources.

- `PRSResourceOffset`
- `NumPRSSymbols`
- `SymbolStart`
- `REOffset`
- `NPRSID`

Examples

Set PRS Configuration Parameters

Create a default PRS configuration object. Display its properties.

```
prs = nrPRSConfig
```

```
prs = 
  nrPRSConfig with properties:

     PRSResourceSetPeriod: 'on'
        PRSResourceOffset: 0
    PRSResourceRepetition: 1
       PRSResourceTimeGap: 1
           MutingPattern1: []
      MutingBitRepetition: 1
           MutingPattern2: []
            NumPRSSymbols: 12
              SymbolStart: 0
                    NumRB: 52
                 RBOffset: 0
                 CombSize: 2
                 REOffset: 0
                   NPRSID: 0

   Constant properties:
     FrequencyOffsetTable: [4x13 table]
```

Create PRS Configuration Object with Two PRS Resources

Create a default PRS configuration object.

```
prs = nrPRSConfig;
```

Set properties that are common to all of the PRS resources in a resource set.

```
prs.PRSResourceSetPeriod = [20 0];
prs.PRSResourceRepetition = 4;
prs.PRSResourceTimeGap = 2;
prs.MutingPattern1 = [1 0];
prs.MutingBitRepetition = 2;
prs.MutingPattern2 = [1 0 1 0];
prs.NumRB = 32;
prs.RBOffset = 10;
prs.CombSize = 4;
```

Set properties that are unique to each PRS resource in a resource set. You can specify these properties as a scalar or vector. If you specify a scalar, the object applies that value to all of the PRS resources in a PRS resource set. If you specify a vector, the object applies the vector element values to the corresponding PRS resource. The length of this vector must be equal to the number of configured PRS resources in a PRS resource set.

```
prs.PRSResourceOffset = [0 10];
prs.NumPRSSymbols = [6 4];
prs.SymbolStart = [0 1];
prs.REOffset = 0;
prs.NPRSID = [10 50];
```

Display the object properties.

```
disp(prs)
```

```
  nrPRSConfig with properties:

     PRSResourceSetPeriod: [20 0]
        PRSResourceOffset: [0 10]
    PRSResourceRepetition: 4
       PRSResourceTimeGap: 2
           MutingPattern1: [1 0]
      MutingBitRepetition: 2
           MutingPattern2: [1 0 1 0]
            NumPRSSymbols: [6 4]
              SymbolStart: [0 1]
                    NumRB: 32
                 RBOffset: 10
                 CombSize: 4
                 REOffset: 0
                   NPRSID: [10 50]

   Constant properties:
     FrequencyOffsetTable: [4x13 table]
```

Version History

Introduced in R2021a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation (Release 16)." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
`nrCarrierConfig`

Functions
`nrPRS` | `nrPRSIndices`

Topics
"NR Positioning Reference Signal"

nrPUCCH0Config

PUCCH format 0 configuration parameters

Description

The `nrPUCCH0Config` object sets physical uplink control channel (PUCCH) configuration parameters for format 0, as defined in TS 38.211 Sections 6.3.2.1 to 6.3.2.3 [1]. The object also configures the bandwidth part (BWP) containing the PUCCH and the number of resource blocks (RBs) that the PUCCH occupies within the BWP.

The default `nrPUCCH0Config` object allocates a PUCCH format 0 in the first RB of the BWP and the last OFDM symbol in the slot of 14 OFDM symbols.

Creation

Syntax

```
pucch = nrPUCCH0Config
pucch = nrPUCCH0Config(Name,Value)
```

Description

`pucch = nrPUCCH0Config` creates a default PUCCH configuration object for format 0.

`pucch = nrPUCCH0Config(Name,Value)` specifies properties on page 3-106 using one or more name-value pairs. Enclose each property in quotes. For example, `'NSizeBWP',200` specifies 200 physical resource blocks (PRB) in the BWP.

Properties

`NSizeBWP` — Number of PRBs in BWP
`[]` (default) | integer from 1 to 275

Number of PRBs in the BWP, specified as `[]` or an integer from 1 to 275. To set this property to the `NSizeGrid` property of the `nrCarrierConfig` object, use `[]`.

Data Types: `double`

`NStartBWP` — Starting PRB index of BWP relative to CRB 0
`[]` (default) | integer from 0 to 2473

Starting PRB index of the BWP relative to the common resource block 0 (CRB 0), specified as `[]` or an integer from 0 to 2473. To set this property to the `NStartGrid` property of the `nrCarrierConfig` object, use `[]`.

Data Types: `double`

`SymbolAllocation` — OFDM symbol allocation of PUCCH
`[13 1]` (default) | `[]` | two-element vector of nonnegative integers

OFDM symbol allocation of the PUCCH within a slot, specified as [] or a two-element vector of nonnegative integers. The first element of this property represents the start of symbol allocation. The second element represents the number of consecutively allocated OFDM symbols and must equal 1 or 2. To indicate that no OFDM symbols are allocated for the PUCCH, use [] or specify the second element of the vector as 0.

Data Types: double

PRBSet — PRB allocation of PUCCH
0 (default) | integer from 0 to 274

PRB (0-based) allocation of the PUCCH within the BWP, specified as an integer from 0 to 274. To indicate that no resource blocks are allocated for the PUCCH, use [].

Data Types: double

FrequencyHopping — Frequency hopping configuration for PUCCH
'neither' (default) | 'intraSlot' | 'interSlot'

Frequency hopping configuration for the PUCCH, specified as one of these values.

- 'neither' — Indicates no frequency hopping
- 'intraSlot' — Enables intraslot frequency hopping
- 'interSlot' — Enables interslot frequency hopping

When you enable interslot frequency hopping, the slot number determines the starting resource block for the allocated OFDM symbols. For even-numbered slots, the resource block starts from the minimum resource block allocated in the PRBSet property. For odd-numbered slots, the resource block starts from the resource block provided in the SecondHopStartPRB property.

Data Types: char | string

SecondHopStartPRB — Starting PRB index of second hop relative to BWP
1 (default) | integer from 0 to 274

Starting PRB index of the second hop relative to the BWP, specified as an integer from 0 to 274.

Data Types: double

GroupHopping — Group hopping configuration for PUCCH
'neither' (default) | 'enable' | 'disable'

Group hopping configuration for the PUCCH, specified as one of these values.

- 'neither' — Disables group hopping and sequence hopping
- 'enable' — Enables group hopping and disables sequence hopping
- 'disable' — Disables group hopping and enables sequence hopping

This property is the higher-layer parameter *pucch-GroupHopping*.

Data Types: char | string

HoppingID — Hopping identity of PUCCH
[] (default) | integer from 0 to 1023

Hopping identity of the PUCCH, specified as [] or an integer from 0 to 1023.

- If the higher-layer parameter *hoppingId* is configured, this property must be in the range from 0 to 1023.
- If the higher-layer parameter *hoppingId* is not configured, this property must be equal to the physical layer cell identity and be in the range from 0 to 1007.

When you specify this property as `[]`, the object sets the PUCCH hopping identity to the physical layer cell identity specified by the `NCellID` property of the `nrCarrierConfig` object.

Data Types: `double`

`InitialCyclicShift` — Initial cyclic shift
`0` (default) | integer from 0 to 11

Initial cyclic shift, specified as an integer from 0 to 11.

This property is the higher-layer parameter *initialCyclicShift*.

Data Types: `double`

Examples

Create PUCCH Format 0 Configuration Object

Create a default PUCCH configuration object for format 0.

```
pucch = nrPUCCH0Config;
```

Specify the size of the BWP as 45, the PRB allocation of the PUCCH as 40, and the initial cyclic shift as 5. Enable group hopping and set the hopping identity to 400. Display the object properties.

```
pucch.NSizeBWP = 45;
pucch.PRBSet = 40;
pucch.GroupHopping = 'enable';
pucch.HoppingID = 400;
pucch.InitialCyclicShift = 5;
disp(pucch)
```

```
  nrPUCCH0Config with properties:

             NSizeBWP: 45
            NStartBWP: []
     SymbolAllocation: [13 1]
               PRBSet: 40
     FrequencyHopping: 'neither'
    SecondHopStartPRB: 1
         GroupHopping: 'enable'
            HoppingID: 400
   InitialCyclicShift: 5
```

Version History

Introduced in R2021a

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
`nrPUCCH1Config` | `nrPUCCH2Config` | `nrPUCCH3Config` | `nrPUCCH4Config`

Functions
`nrPUCCH` | `nrPUCCHIndices`

nrPUCCH1Config

PUCCH format 1 configuration parameters

Description

The `nrPUCCH1Config` object sets physical uplink control channel (PUCCH) configuration parameters for format 1, as defined in TS 38.211 Sections 6.3.2.1, 6.3.2.2, 6.3.2.4, and 6.4.1.3.1 [1]. The object also configures the bandwidth part (BWP) containing the PUCCH and the number of resource blocks (RBs) that the PUCCH occupies within the BWP.

The default `nrPUCCH1Config` object allocates a PUCCH format 1 in the first RB in the BWP and in all OFDM symbols in a slot.

Creation

Syntax

```
pucch = nrPUCCH1Config
pucch = nrPUCCH1Config(Name,Value)
```

Description

`pucch = nrPUCCH1Config` creates a default PUCCH configuration object for format 1.

`pucch = nrPUCCH1Config(Name,Value)` specifies properties on page 3-110 using one or more name-value pairs. Enclose each property in quotes. For example, `'NSizeBWP',200` specifies 200 physical resource blocks (PRB) in the BWP.

Properties

`NSizeBWP` — Number of PRBs in BWP
`[]` (default) | integer from 1 to 275

Number of PRBs in the BWP, specified as `[]` or an integer from 1 to 275. To set this property to the `NSizeGrid` property of the `nrCarrierConfig` object, use `[]`.

Data Types: `double`

`NStartBWP` — Starting PRB index of BWP relative to CRB 0
`[]` (default) | integer from 0 to 2473

Starting PRB index of the BWP relative to the common resource block 0 (CRB 0), specified as `[]` or an integer from 0 to 2473. To set this property to the `NStartGrid` property of the `nrCarrierConfig` object, use `[]`.

Data Types: `double`

`SymbolAllocation` — OFDM symbol allocation of PUCCH
`[0 14]` (default) | `[]` | two-element vector of nonnegative integers

OFDM symbol allocation of the PUCCH within a slot, specified as [] or a two-element vector of nonnegative integers. The first element of this property represents the start of symbol allocation. The second element represents the number of consecutively allocated OFDM symbols and must be in the range [4, 14]. To indicate that no OFDM symbols are allocated for the PUCCH, use [] or specify the second element of the vector as 0.

Data Types: double

PRBSet — PRB allocation of PUCCH
0 (default) | integer from 0 to 274

PRB (0-based) allocation of the PUCCH within the BWP, specified as an integer from 0 to 274. To indicate that no resource blocks are allocated for the PUCCH, use [].

Data Types: double

FrequencyHopping — Frequency hopping configuration for PUCCH
'neither' (default) | 'intraSlot' | 'interSlot'

Frequency hopping configuration for the PUCCH, specified as one of these values.

- 'neither' — Indicates no frequency hopping
- 'intraSlot' — Enables intraslot frequency hopping
- 'interSlot' — Enables interslot frequency hopping

When you enable interslot frequency hopping, the slot number determines the starting resource block for the allocated OFDM symbols. For even-numbered slots, the resource block starts from the minimum resource block allocated in the PRBSet property. For odd-numbered slots, the resource block starts from the resource block provided in the SecondHopStartPRB property.

Data Types: char | string

SecondHopStartPRB — Starting PRB index of second hop relative to BWP
1 (default) | integer from 0 to 274

Starting PRB index of the second hop relative to the BWP, specified as an integer from 0 to 274.

Data Types: double

GroupHopping — Group hopping configuration for PUCCH
'neither' (default) | 'enable' | 'disable'

Group hopping configuration for the PUCCH, specified as one of these values.

- 'neither' — Disables group hopping and sequence hopping
- 'enable' — Enables group hopping and disables sequence hopping
- 'disable' — Disables group hopping and enables sequence hopping

This property is the higher-layer parameter *pucch-GroupHopping*.

Data Types: char | string

HoppingID — Hopping identity of PUCCH
[] (default) | integer from 0 to 1023

Hopping identity of the PUCCH, specified as [] or an integer from 0 to 1023.

- If the higher-layer parameter *hoppingId* is configured, this property must be in the range from 0 to 1023.
- If the higher-layer parameter *hoppingId* is not configured, this property must be equal to the physical layer cell identity and be in the range from 0 to 1007.

When you specify this property as `[]`, the object sets the PUCCH hopping identity to the physical layer cell identity specified by the `NCellID` property of the `nrCarrierConfig` object.

Data Types: `double`

`InitialCyclicShift` — Initial cyclic shift
`0` (default) | integer from 0 to 11

Initial cyclic shift, specified as an integer from 0 to 11.

This property is the higher-layer parameter *initialCyclicShift*.

Data Types: `double`

`OCCI` — Orthogonal cover code index
`0` (default) | integer from 0 to 6

Orthogonal cover code index (OCCI), specified as an integer from 0 to 6.

- When you disable intraslot frequency hopping, the OCCI value must be less than the floor of half of the number of OFDM symbols allocated for the PUCCH.
- When you enable intraslot frequency hopping, the OCCI value must be less than the floor of one-fourth of the number of OFDM symbols allocated for the PUCCH.

Data Types: `double`

Examples

Create PUCCH Format 1 Configuration Object

Create a default PUCCH configuration object for format 1.

```
pucch = nrPUCCH1Config;
```

Specify the PRB allocation of the PUCCH as 75, the initial cyclic shift as 3, and the OCCI as 5. Enable group hopping and set the hopping identity to 450. Display the object properties.

```
pucch.PRBSet = 75;
pucch.GroupHopping = 'enable';
pucch.HoppingID = 450;
pucch.InitialCyclicShift = 3;
pucch.OCCI = 5;
disp(pucch)
```

```
  nrPUCCH1Config with properties:

              NSizeBWP: []
             NStartBWP: []
      SymbolAllocation: [0 14]
                PRBSet: 75
```

```
      FrequencyHopping: 'neither'
     SecondHopStartPRB: 1
          GroupHopping: 'enable'
             HoppingID: 450
    InitialCyclicShift: 3
                  OCCI: 5
```

Version History

Introduced in R2021a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`nrPUCCH0Config` | `nrPUCCH2Config` | `nrPUCCH3Config` | `nrPUCCH4Config`

Functions

`nrPUCCH` | `nrPUCCHIndices`

nrPUCCH2Config

PUCCH format 2 configuration parameters

Description

The nrPUCCH2Config object sets physical uplink control channel (PUCCH) configuration parameters for format 2, as defined in TS 38.211 Sections 6.3.2.1, 6.3.2.5, and 6.4.1.3.2 [1]. The object also configures the bandwidth part (BWP) containing the PUCCH and the number of resource blocks (RBs) that the PUCCH occupies within the BWP.

The default nrPUCCH2Config object allocates a PUCCH format 2 in the first RB in the BWP and the last OFDM symbol in the slot of 14 OFDM symbols.

Creation

Syntax

```
pucch = nrPUCCH2Config
pucch = nrPUCCH2Config(Name,Value)
```

Description

pucch = nrPUCCH2Config creates a default PUCCH configuration object for format 2.

pucch = nrPUCCH2Config(Name,Value) specifies properties on page 3-114 using one or more name-value pairs. Enclose each property in quotes. For example, 'NSizeBWP',200 specifies 200 physical resource blocks (PRB) in the BWP.

Properties

NSizeBWP — Number of PRBs in BWP
[] (default) | integer from 1 to 275

Number of PRBs in the BWP, specified as [] or an integer from 1 to 275. To set this property to the NSizeGrid property of the nrCarrierConfig object, use [].

Data Types: double

NStartBWP — Starting PRB index of BWP relative to CRB 0
[] (default) | integer from 0 to 2473

Starting PRB index of the BWP relative to the common resource block 0 (CRB 0), specified as [] or an integer from 0 to 2473. To set this property to the NStartGrid property of the nrCarrierConfig object, use [].

Data Types: double

SymbolAllocation — OFDM symbol allocation of PUCCH
[13 1] (default) | [] | two-element vector of nonnegative integers

OFDM symbol allocation of the PUCCH within a slot, specified as [] or a two-element vector of nonnegative integers. The first element of this property represents the start of symbol allocation. The second element represents the number of consecutively allocated OFDM symbols and must equal 1 or 2. To indicate that no OFDM symbols are allocated for the PUCCH, use [] or specify the second element of the vector as 0.

Data Types: double

PRBSet — PRB allocation of PUCCH
0 (default) | vector of integers from 0 to 274

PRB (0-based) allocation of the PUCCH within the BWP, specified as a vector of integers from 0 to 274. To indicate that no resource blocks are allocated for the PUCCH, use [].

Data Types: double

FrequencyHopping — Frequency hopping configuration for PUCCH
'neither' (default) | 'intraSlot' | 'interSlot'

Frequency hopping configuration for the PUCCH, specified as one of these values.

- 'neither' — Indicates no frequency hopping
- 'intraSlot' — Enables intraslot frequency hopping
- 'interSlot' — Enables interslot frequency hopping

When you enable interslot frequency hopping, the slot number determines the starting resource block for the allocated OFDM symbols. For even-numbered slots, the resource block starts from the minimum resource block allocated in the PRBSet property. For odd-numbered slots, the resource block starts from the resource block provided in the SecondHopStartPRB property.

Data Types: char | string

SecondHopStartPRB — Starting PRB index of second hop relative to BWP
1 (default) | integer from 0 to 274

Starting PRB index of the second hop relative to the BWP, specified as an integer from 0 to 274.

Data Types: double

NID — PUCCH scrambling identity
[] (default) | integer from 0 to 1023

PUCCH scrambling identity, specified as [] or an integer from 0 to 1023.

- If the higher-layer parameter *dataScramblingIdentityPUSCH* is configured, this property must be in the range from 0 to 1023.
- If the higher-layer parameter *dataScramblingIdentityPUSCH* is not configured, this property must be equal to the physical layer cell identity and be in the range from 0 to 1007.

When you specify this property as [], the object sets the PUCCH scrambling identity to the physical layer cell identity specified by the NCellID property of the nrCarrierConfig object.

Data Types: double

RNTI — Radio network temporary identifier
1 (default) | integer from 0 to 65,535

Radio network temporary identifier of the user equipment (UE), specified as an integer from 0 to 65,535.

Data Types: double

NID0 — DM-RS scrambling identity
[] (default) | integer from 0 to 65,535

Demodulation reference signal (DM-RS) scrambling identity, specified as [] or an integer from 0 to 65,535.

- If the higher-layer parameter *scramblingID0* is configured, this property must be in the range from 0 to 65,535.
- If the higher-layer parameter *scramblingID0* is not configured, this property must be equal to the physical layer cell identity and be in the range from 0 to 1007.

When you specify this property as [], the object sets the DM-RS scrambling identity to the physical layer cell identity specified by the NCellID property of the nrCarrierConfig object.

Data Types: double

Examples

Create PUCCH Format 2 Configuration Object

Create a default PUCCH configuration object for format 2.

```
pucch = nrPUCCH2Config;
```

Specify the starting PRB index of the BWP relative to the CRB 0 as 1005, the PRB allocation of the PUCCH to range from 70 to 74 (occupying 5 resource blocks), and the PUCCH to span the first two OFDM symbols in each slot. Enable intraslot frequency hopping and set the starting PRB index of the second hop as 35. Set the data scrambling identity to 450 and the radio network temporary identifier of the UE to 12,034. Display the object properties.

```
pucch.NStartBWP = 1005;
pucch.SymbolAllocation = [0 2];
pucch.PRBSet = 70:74;
pucch.FrequencyHopping = 'intraSlot';
pucch.SecondHopStartPRB = 35;
pucch.NID = 450;
pucch.RNTI = 12034;
disp(pucch)
```

```
  nrPUCCH2Config with properties:

              NSizeBWP: []
             NStartBWP: 1005
      SymbolAllocation: [0 2]
                PRBSet: [70 71 72 73 74]
      FrequencyHopping: 'intraSlot'
     SecondHopStartPRB: 35
                   NID: 450
                  RNTI: 12034
                  NID0: []
```

Version History

Introduced in R2021a

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
`nrPUCCH0Config` | `nrPUCCH1Config` | `nrPUCCH3Config` | `nrPUCCH4Config`

Functions
`nrPUCCH` | `nrPUCCHIndices`

nrPUCCH3Config

PUCCH format 3 configuration parameters

Description

The `nrPUCCH3Config` object sets physical uplink control channel (PUCCH) configuration parameters for format 3, as defined in TS 38.211 Sections 6.3.2.1, 6.3.2.2, 6.3.2.6, and 6.4.1.3.3 [1]. The object also configures the bandwidth part (BWP) containing the PUCCH and the number of resource blocks (RBs) that the PUCCH occupies within the BWP.

The default `nrPUCCH3Config` object allocates a PUCCH format 3 in the first RB in the BWP and in all OFDM symbols in a slot.

Creation

Syntax

```
pucch = nrPUCCH3Config
pucch = nrPUCCH3Config(Name,Value)
```

Description

`pucch = nrPUCCH3Config` creates a default PUCCH configuration object for format 3.

`pucch = nrPUCCH3Config(Name,Value)` specifies properties on page 3-118 using one or more name-value pairs. Enclose each property in quotes. For example, `'NSizeBWP',200` specifies 200 physical resource blocks (PRB) in the BWP.

Properties

`NSizeBWP` — Number of PRBs in BWP
`[]` (default) | integer from 1 to 275

Number of PRBs in the BWP, specified as `[]` or an integer from 1 to 275. To set this property to the `NSizeGrid` property of the `nrCarrierConfig` object, use `[]`.

Data Types: `double`

`NStartBWP` — Starting PRB index of BWP relative to CRB 0
`[]` (default) | integer from 0 to 2473

Starting PRB index of the BWP relative to the common resource block 0 (CRB 0), specified as `[]` or an integer from 0 to 2473. To set this property to the `NStartGrid` property of the `nrCarrierConfig` object, use `[]`.

Data Types: `double`

`Modulation` — Modulation scheme
`'QPSK'` (default) | `'pi/2-BPSK'`

Modulation scheme, specified as 'QPSK' or 'pi/2-BPSK'.

Modulation Scheme	Number of Bits Per Symbol
'pi/2-BPSK'	1
'QPSK'	2

Data Types: char | string

SymbolAllocation — OFDM symbol allocation of PUCCH
[0 14] (default) | [] | two-element vector of nonnegative integers

OFDM symbol allocation of the PUCCH within a slot, specified as [] or a two-element vector of nonnegative integers. The first element of this property represents the start of symbol allocation. The second element represents the number of consecutively allocated OFDM symbols and must be in the range [4, 14]. To indicate that no OFDM symbols are allocated for the PUCCH, use [] or specify the second element of the vector as 0.

Data Types: double

PRBSet — PRB allocation of PUCCH
0 (default) | vector of integers from 0 to 274

PRB (0-based) allocation of the PUCCH within the BWP, specified as a vector of integers from 0 to 274. To indicate that no resource blocks are allocated for the PUCCH, use [].

Data Types: double

FrequencyHopping — Frequency hopping configuration for PUCCH
'neither' (default) | 'intraSlot' | 'interSlot'

Frequency hopping configuration for the PUCCH, specified as one of these values.

- 'neither' — Indicates no frequency hopping
- 'intraSlot' — Enables intraslot frequency hopping
- 'interSlot' — Enables interslot frequency hopping

When you enable interslot frequency hopping, the slot number determines the starting resource block for the allocated OFDM symbols. For even-numbered slots, the resource block starts from the minimum resource block allocated in the PRBSet property. For odd-numbered slots, the resource block starts from the resource block provided in the SecondHopStartPRB property.

Data Types: char | string

SecondHopStartPRB — Starting PRB index of second hop relative to BWP
1 (default) | integer from 0 to 274

Starting PRB index of the second hop relative to the BWP, specified as an integer from 0 to 274.

Data Types: double

GroupHopping — Group hopping configuration for PUCCH
'neither' (default) | 'enable' | 'disable'

Group hopping configuration for the PUCCH, specified as one of these values.

- `'neither'` — Disables group hopping and sequence hopping
- `'enable'` — Enables group hopping and disables sequence hopping
- `'disable'` — Disables group hopping and enables sequence hopping

This property is the higher-layer parameter *pucch-GroupHopping*.

Data Types: `char` | `string`

`HoppingID` — Hopping identity of PUCCH
`[]` (default) | integer from 0 to 1023

Hopping identity of the PUCCH, specified as `[]` or an integer from 0 to 1023.

- If the higher-layer parameter *hoppingId* is configured, this property must be in the range from 0 to 1023.
- If the higher-layer parameter *hoppingId* is not configured, this property must be equal to the physical layer cell identity and be in the range from 0 to 1007.

When you specify this property as `[]`, the object sets the PUCCH hopping identity to the physical layer cell identity specified by the `NCellID` property of the `nrCarrierConfig` object.

Data Types: `double`

`NID` — PUCCH scrambling identity
`[]` (default) | integer from 0 to 1023

PUCCH scrambling identity, specified as `[]` or an integer from 0 to 1023.

- If the higher-layer parameter *dataScramblingIdentityPUSCH* is configured, this property must be in the range from 0 to 1023.
- If the higher-layer parameter *dataScramblingIdentityPUSCH* is not configured, this property must be equal to the physical layer cell identity and be in the range from 0 to 1007.

When you specify this property as `[]`, the object sets the PUCCH scrambling identity to the physical layer cell identity specified by the `NCellID` property of the `nrCarrierConfig` object.

Data Types: `double`

`RNTI` — Radio network temporary identifier
`1` (default) | integer from 0 to 65,535

Radio network temporary identifier of the user equipment (UE), specified as an integer from 0 to 65,535.

Data Types: `double`

`AdditionalDMRS` — Option to enable additional DM-RS
`false` or `0` (default) | `true` or `1`

Option to enable additional demodulation reference signal (DM-RS), provided by the higher-layer parameter *additionalDMRS*, specified as one of these numeric or logical values.

- `0` (`false`) — Disable additional DM-RS. When the number of the PUCCH OFDM symbols is greater than 9, two DM-RS symbols are present.
- `1` (`true`) — Enable additional DM-RS. When the number of the PUCCH OFDM symbols is greater than 9, four DM-RS symbols are present.

Data Types: double | logical

Examples

Create PUCCH Format 3 Configuration Object

Create a default PUCCH configuration object for format 3.

```
pucch = nrPUCCH3Config;
```

Specify the number of PRBs in the BWP as 150, the modulation scheme as pi/2-BPSK, the PRB allocation of the PUCCH to range from 90 to 97, and the data scrambling identity as 650. Enable group hopping and the additional DM-RS configuration flag. Display the object properties.

```
pucch.NSizeBWP = 150;
pucch.Modulation = 'pi/2-BPSK';
pucch.PRBSet = 90:97;
pucch.GroupHopping = 'enable';
pucch.NID = 650;
pucch.AdditionalDMRS = 1;
disp(pucch)
```

```
  nrPUCCH3Config with properties:

             NSizeBWP: 150
            NStartBWP: []
           Modulation: 'pi/2-BPSK'
     SymbolAllocation: [0 14]
               PRBSet: [90 91 92 93 94 95 96 97]
     FrequencyHopping: 'neither'
    SecondHopStartPRB: 1
         GroupHopping: 'enable'
            HoppingID: []
                  NID: 650
                 RNTI: 1
       AdditionalDMRS: 1
```

Version History

Introduced in R2021a

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects
nrPUCCH0Config | nrPUCCH1Config | nrPUCCH2Config | nrPUCCH4Config

Functions
nrPUCCH | nrPUCCHIndices | nrPUCCHDMRS | nrPUCCHDMRSIndices

nrPUCCH4Config

PUCCH format 4 configuration parameters

Description

The `nrPUCCH4Config` object sets physical uplink control channel (PUCCH) configuration parameters for format 4, as defined in TS 38.211 Sections 6.3.2.1, 6.3.2.2, 6.3.2.6, and 6.4.1.3.3 [1]. The object also configures the bandwidth part (BWP) containing the PUCCH and the number of resource blocks (RBs) that the PUCCH occupies within the BWP.

The default `nrPUCCH4Config` object configures a PUCCH format 4 in the first RB in the BWP and in all OFDM symbols in a slot.

Creation

Syntax

```
pucch = nrPUCCH4Config
pucch = nrPUCCH4Config(Name,Value)
```

Description

`pucch = nrPUCCH4Config` creates a default PUCCH configuration object for format 4.

`pucch = nrPUCCH4Config(Name,Value)` specifies properties on page 3-123 using one or more name-value pairs. Enclose each property in quotes. For example, `'NSizeBWP',200` specifies 200 physical resource blocks (PRB) in the BWP.

Properties

`NSizeBWP` — Number of PRBs in BWP
`[]` (default) | integer from 1 to 275

Number of PRBs in the BWP, specified as `[]` or an integer from 1 to 275. To set this property to the `NSizeGrid` property of the `nrCarrierConfig` object, use `[]`.

Data Types: `double`

`NStartBWP` — Starting PRB index of BWP relative to CRB 0
`[]` (default) | integer from 0 to 2473

Starting PRB index of the BWP relative to the common resource block 0 (CRB 0), specified as `[]` or an integer from 0 to 2473. To set this property to the `NStartGrid` property of the `nrCarrierConfig` object, use `[]`.

Data Types: `double`

`Modulation` — Modulation scheme
`'QPSK'` (default) | `'pi/2-BPSK'`

Modulation scheme, specified as 'QPSK' or 'pi/2-BPSK'.

Modulation Scheme	Number of Bits Per Symbol
'pi/2-BPSK'	1
'QPSK'	2

Data Types: char | string

SymbolAllocation — OFDM symbol allocation of PUCCH
[0 14] (default) | [] | two-element vector of nonnegative integers

OFDM symbol allocation of the PUCCH within a slot, specified as [] or a two-element vector of nonnegative integers. The first element of this property represents the start of symbol allocation. The second element represents the number of consecutively allocated OFDM symbols and must be in the range [4, 14]. To indicate that no OFDM symbols are allocated for the PUCCH, use [] or specify the second element of the vector as 0.

Data Types: double

PRBSet — PRB allocation of PUCCH
0 (default) | integer from 0 to 274

PRB (0-based) allocation of the PUCCH within the BWP, specified as an integer from 0 to 274. To indicate that no resource blocks are allocated for the PUCCH, use [].

Data Types: double

FrequencyHopping — Frequency hopping configuration for PUCCH
'neither' (default) | 'intraSlot' | 'interSlot'

Frequency hopping configuration for the PUCCH, specified as one of these values.

- 'neither' — Indicates no frequency hopping
- 'intraSlot' — Enables intraslot frequency hopping
- 'interSlot' — Enables interslot frequency hopping

When you enable interslot frequency hopping, the slot number determines the starting resource block for the allocated OFDM symbols. For even-numbered slots, the resource block starts from the minimum resource block allocated in the PRBSet property. For odd-numbered slots, the resource block starts from the resource block provided in the SecondHopStartPRB property.

Data Types: char | string

SecondHopStartPRB — Starting PRB index of second hop relative to BWP
1 (default) | integer from 0 to 274

Starting PRB index of the second hop relative to the BWP, specified as an integer from 0 to 274.

Data Types: double

GroupHopping — Group hopping configuration for PUCCH
'neither' (default) | 'enable' | 'disable'

Group hopping configuration for the PUCCH, specified as one of these values.

- `'neither'` — Disables group hopping and sequence hopping
- `'enable'` — Enables group hopping and disables sequence hopping
- `'disable'` — Disables group hopping and enables sequence hopping

This property is the higher-layer parameter *pucch-GroupHopping*.

Data Types: `char` | `string`

`HoppingID` — Hopping identity of PUCCH
`[]` (default) | integer from 0 to 1023

Hopping identity of the PUCCH, specified as `[]` or an integer from 0 to 1023.

- If the higher-layer parameter *hoppingId* is configured, this property must be in the range from 0 to 1023.
- If the higher-layer parameter *hoppingId* is not configured, this property must be equal to the physical layer cell identity and be in the range from 0 to 1007.

When you specify this property as `[]`, the object sets the PUCCH hopping identity to the physical layer cell identity specified by the `NCellID` property of the `nrCarrierConfig` object.

Data Types: `double`

`SpreadingFactor` — Spreading factor
`2` (default) | `4`

Spreading factor, specified as `2` or `4`.

Data Types: `double`

`OCCI` — Orthogonal cover code index
`0` (default) | integer from 0 to 3

Orthogonal cover code index (OCCI), specified as an integer from 0 to 3. The OCCI value must be less than the `SpreadingFactor` property.

Data Types: `double`

`NID` — PUCCH scrambling identity
`[]` (default) | integer from 0 to 1023

PUCCH scrambling identity, specified as `[]` or an integer from 0 to 1023.

- If the higher-layer parameter *dataScramblingIdentityPUSCH* is configured, this property must be in the range from 0 to 1023.
- If the higher-layer parameter *dataScramblingIdentityPUSCH* is not configured, this property must be equal to the physical layer cell identity and be in the range from 0 to 1007.

When you specify this property as `[]`, the object sets the PUCCH scrambling identity to the physical layer cell identity specified by the `NCellID` property of the `nrCarrierConfig` object.

Data Types: `double`

`RNTI` — Radio network temporary identifier
`1` (default) | integer from 0 to 65,535

Radio network temporary identifier of the user equipment (UE), specified as an integer from 0 to 65,535.

Data Types: `double`

`AdditionalDMRS` — Option to enable additional DM-RS
`false` or `0` (default) | `true` or `1`

Option to enable additional demodulation reference signal (DM-RS), provided by the higher-layer parameter *additionalDMRS*, specified as one of these numeric or logical values.

- `0` (`false`) — Disable additional DM-RS. When the number of the PUCCH OFDM symbols is greater than 9, two DM-RS symbols are present.
- `1` (`true`) — Enable additional DM-RS. When the number of the PUCCH OFDM symbols is greater than 9, four DM-RS symbols are present.

Data Types: `double` | `logical`

Examples

Create PUCCH Format 4 Configuration Object

Create a default PUCCH configuration object for format 4.

```
pucch = nrPUCCH4Config;
```

Specify the modulation scheme as pi/2-BPSK. Enable interslot frequency hopping. Set the starting PRB index of the second hop to 200, the spreading factor to 4, and the orthogonal cover code index to 2. Enable the additional DM-RS configuration flag. Display the object properties.

```
pucch.Modulation = 'pi/2-BPSK';
pucch.FrequencyHopping = 'interSlot';
pucch.SecondHopStartPRB = 200;
pucch.SpreadingFactor = 4;
pucch.OCCI = 2;
pucch.AdditionalDMRS = 1;
disp(pucch)
```

```
  nrPUCCH4Config with properties:

              NSizeBWP: []
             NStartBWP: []
            Modulation: 'pi/2-BPSK'
      SymbolAllocation: [0 14]
                PRBSet: 0
      FrequencyHopping: 'interSlot'
     SecondHopStartPRB: 200
          GroupHopping: 'neither'
             HoppingID: []
       SpreadingFactor: 4
                  OCCI: 2
                   NID: []
                  RNTI: 1
        AdditionalDMRS: 1
```

Version History

Introduced in R2021a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
`nrPUCCH0Config` | `nrPUCCH1Config` | `nrPUCCH2Config` | `nrPUCCH3Config`

Functions
`nrPUCCH` | `nrPUCCHIndices` | `nrPUCCHDMRS` | `nrPUCCHDMRSIndices`

nrPUSCHConfig

PUSCH configuration parameters

Description

The `nrPUSCHConfig` object sets physical uplink shared channel (PUSCH) configuration parameters, as defined in TS 38.211 Sections 6.3.1, 6.4.1.1, and 6.4.1.2 [1].This object bundles all the properties involved in the PUSCH processing chain, including scrambling, symbol modulation, layer mapping, transform precoding, MIMO precoding, and resource element mapping. The object also contains properties to determine the number of resources for the uplink control information (UCI) multiplexing and associated physical reference signals, such as demodulation reference signal (DM-RS) and phase tracking reference signal (PT-RS).

The default `nrPUSCHConfig` object configures a single-layer PUSCH with cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM), mapping type A, QPSK modulation, a resource allocation of 52 resource blocks and 14 OFDM symbols in a slot, and single-symbol DM-RS type 1. This configuration corresponds to a full resource allocation with respect to the default `nrCarrierConfig` object.

Creation

Syntax

```
pusch = nrPUSCHConfig
pusch = nrPUSCHConfig(Name,Value)
```

Description

`pusch = nrPUSCHConfig` creates a PUSCH configuration object with default properties.

`pusch = nrPUSCHConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'NSizeBWP',200,'NumLayers',4` specifies 200 physical resource blocks (PRB) in the bandwidth part (BWP) and 4 transmission layers. Unspecified properties take their default values.

Properties

Channel Configuration

`NSizeBWP` — Number of PRBs in BWP
`[]` (default) | integer from 1 to 275

Number of PRBs in bandwidth part (BWP), specified as an integer from 1 to 275. Use `[]` to set this property to the `NSizeGrid` property of the `nrCarrierConfig` object.

Data Types: `double`

NStartBWP — Starting PRB index of BWP relative to CRB 0
[] (default) | integer from 0 to 2473

Starting PRB index of BWP relative to common resource block 0 (CRB 0), specified as an integer from 0 to 2473. Use [] to set this property to the NStartGrid property of the nrCarrierConfig object.

Data Types: double

Modulation — Modulation scheme
'QPSK' (default) | 'pi/2-BPSK' | '16QAM' | '64QAM' | '256QAM' | string scalar

Modulation scheme, specified as 'QPSK', 'pi/2-BPSK', '16QAM', '64QAM', or '256QAM', a string scalar, or a character array.

Modulation Scheme	Number of Bits Per Symbol
'pi/2-BPSK'	1
'QPSK'	2
'16QAM'	4
'64QAM'	6
'256QAM'	8

Data Types: char | string

NumLayers — Number of transmission layers
1 (default) | 2 | 3 | 4

Number of transmission layers, specified as 1, 2, 3, or 4.

Data Types: double

MappingType — Mapping type
'A' (default) | 'B'

Mapping type of the physical shared channel, specified as 'A' or 'B'.

Data Types: char | string

SymbolAllocation — OFDM symbol allocation
[0 14] (default) | two-element vector of nonnegative integers

OFDM symbol allocation of the physical shared channel, specified as a two-element vector of nonnegative integers. The first element of this property represents the start of symbol allocation (0-based). The second element represents the number of allocated OFDM symbols.

When you set this property to [] or the second element of the vector to 0, no symbol is allocated for the channel.

Data Types: double

PRBSet — PRB allocation
[0:51] (default) | vector of integers from 0 to 274

Physical resource block (PRB) allocation of the PUSCH within the BWP, specified as a vector of integers from 0 to 274.

Data Types: double

TransformPrecoding — Transform precoding
0 or false (default) | 1 or true

Transform precoding, specified as one of these values.

- 0 (false) — Disable transform precoding. The waveform type is cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM).
- 1 (true) — Enable transform precoding. The waveform type is discrete Fourier transform spread orthogonal frequency division multiplexing (DFT-s-OFDM).

Data Types: double | logical

TransmissionScheme — PUSCH transmission scheme
'nonCodebook' (default) | 'codebook'

PUSCH transmission scheme, specified as 'nonCodebook' or 'codebook'.

Data Types: char | string

NumAntennaPorts — Number of antenna ports
1 (default) | 2 | 4

Number of antenna ports, specified as 1, 2, or 4. This value must be greater than or equal to the NumLayers property.

Dependencies

This property is applicable only when TransmissionScheme is set to 'codebook'.

Data Types: double

TPMI — Transmitted precoding matrix indicator
0 (default) | integer from 0 to 27

Transmitted precoding matrix indicator, specified as an integer from 0 to 27.

Dependencies

To enable this property, set the TransmissionScheme property to 'codebook'.

Data Types: double

FrequencyHopping — Frequency hopping
'neither' (default) | 'intraSlot' | 'interSlot'

Frequency hopping for the physical uplink shared channel, specified as 'neither', 'intraSlot', or 'interSlot'.

Data Types: char | string

SecondHopStartPRB — Starting PRB index of second hop
1 (default) | integer from 0 to 274

Starting PRB index of the second hop relative to the BWP, specified as an integer from 0 to 274.

Dependencies

This property is applicable only when `FrequencyHopping` is set to `'intraSlot'`, or `'interSlot'`.

Data Types: `double`

`NID` — PUSCH scrambling identity
`[]` (default) | integer from 0 to 1023

PUSCH scrambling identity, specified as `[]` or an integer from 0 to 1023.

- If the higher layer parameter *dataScramblingIdentityPUSCH* is configured, `NID` must be an integer from 0 to 1023.
- If the higher layer parameter *dataScramblingIdentityPUSCH* is not configured, `NID` must be an integer from 0 to 1007.

When you specify this property as `[]`, the object sets the PUSCH scrambling identity to the physical layer cell identity, specified by the `NCellID` property of the carrier.

Data Types: `double`

`RNTI` — Radio network temporary identifier
`1` (default) | integer from 0 to 65,535

Radio network temporary identifier of the user equipment (UE), specified as an integer from 0 to 65,535.

Data Types: `double`

`NRAPID` — Random access preamble index
`[]` (default) | integer from 0 to 63

Random access preamble index, specified as one of these values.

- `[]` — Use this value to specify that the scrambling initialization does not consider *msgA* on PUSCH.
- Integer from 0 to 63 — Use this value to initialize the scrambling sequence for *msgA* on PUSCH, as defined in Release 16 of TS 38.211 Section 6.3.1.1.

Data Types: `double`

UCI on PUSCH Configuration

`BetaOffsetACK` — Beta offset factor of HARQ-ACK
`20` (default) | positive real-valued scalar

Beta offset factor of the HARQ-ACK, specified as a positive real-valued scalar. This property determines the number of resources for multiplexing HARQ-ACK. The nominal value is one of the entry from the Table 9.3-1 of TS 38.213.

Data Types: `double`

`BetaOffsetCSI1` — Beta offset factor of CSI part 1
`6.25` (default) | positive real-valued scalar

Beta offset factor of the channel state information (CSI) part 1, specified as a positive real-valued scalar. This property determines the number of resources for multiplexing CSI part 1. The nominal value is one of the entry from the Table 9.3-2 of TS 38.213.

Data Types: `double`

`BetaOffsetCSI2` — Beta offset factor of CSI part 2
`6.25` (default) | positive real-valued scalar

Beta offset factor of the CSI part 2, specified as a positive real-valued scalar. This property determines the number of resources for multiplexing CSI part 2. The nominal value is one of the entry from the Table 9.3-2 of TS 38.213.

Data Types: `double`

`UCIScaling` — Scaling factor
`1` (default) | scalar between 0 and 1

Scaling factor to limit the number of the resource elements allocated for the UCI on the PUSCH, specified as a scalar between 0 and 1. The nominal value is 0.5, 0.65, 0.8, or 1.

Data Types: `double`

Reference Signals Configuration

`DMRS` — PUSCH DM-RS configuration parameters
default `nrPUSCHDMRSConfig` object (default) | `nrPUSCHDMRSConfig` object

PUSCH DM-RS configuration parameters, specified as an `nrPUSCHDMRSConfig` configuration object.

`EnablePTRS` — Enable PT-RS
`0` or `false` (default) | `1` or `true`

Enable the PT-RS, specified as one of these values.

- `0` (`false`) — Disable the PT-RS configuration.
- `1` (`true`) — Enable the PT-RS configuration.

Data Types: `double` | `logical`

`PTRS` — PUSCH PT-RS configuration parameters
default `nrPUSCHPTRSConfig` object (default) | `nrPUSCHPTRSConfig` object

PUSCH phase tracking reference signal (PT-RS) configuration, specified as an `nrPUSCHPTRSConfig` configuration object.

Examples

Create PUSCH Configuration Object

Create a physical uplink shared channel configuration object with 'codebook' transmission scheme.

Specify the size of the bandwidth part as 52, scrambling identity as 750, frequency hopping as 'interslot', and number of antenna ports as 2. Enable transform precoding and PT-RS configuration.

```
pusch = nrPUSCHConfig;
pusch.NSizeBWP = 52;
pusch.NID = 750;
pusch.TransmissionScheme = 'codebook';
pusch.FrequencyHopping = 'interslot';
pusch.NumAntennaPorts = 2;
pusch.TransformPrecoding = 1;
pusch.EnablePTRS = 1;
disp(pusch)
```

```
  nrPUSCHConfig with properties:

              NSizeBWP: 52
             NStartBWP: []
            Modulation: 'QPSK'
             NumLayers: 1
           MappingType: 'A'
      SymbolAllocation: [0 14]
                PRBSet: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 ... ]
    TransformPrecoding: 1
    TransmissionScheme: 'codebook'
       NumAntennaPorts: 2
                  TPMI: 0
      FrequencyHopping: 'interSlot'
     SecondHopStartPRB: 1
         BetaOffsetACK: 20
        BetaOffsetCSI1: 6.2500
        BetaOffsetCSI2: 6.2500
            UCIScaling: 1
                   NID: 750
                  RNTI: 1
                NRAPID: []
                  DMRS: [1x1 nrPUSCHDMRSConfig]
            EnablePTRS: 1
                  PTRS: [1x1 nrPUSCHPTRSConfig]
```

Generate PUSCH Indices and PT-RS Indices for Codebook-Based Transmission

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier with 15 kHz subcarrier spacing.

```
carrier = nrCarrierConfig;
```

Create a PUSCH configuration object with codebook-based transmission. Set the number of antenna ports to 4, modulation scheme to pi/2-BPSK, transmitted precoding matrix indicator to 10, and transform precoding to 0. When transform precoding is 0, the waveform type is cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM). Enable phase tracking reference signal (PT-RS).

```
pusch = nrPUSCHConfig;
pusch.TransformPrecoding = 0;
pusch.Modulation = 'pi/2-BPSK';
pusch.TransmissionScheme = 'codebook';
pusch.NumAntennaPorts = 4;
pusch.TPMI = 10;
pusch.EnablePTRS = 1;
```

Generate PUSCH indices and PT-RS indices in subscript form.

```
[ind,info,ptrsInd] = nrPUSCHIndices(carrier,pusch,'IndexStyle','subscript')
```

```
ind = 31096x3 uint32 matrix

    1    1    1
    2    1    1
    3    1    1
    4    1    1
    5    1    1
    6    1    1
    7    1    1
    8    1    1
    9    1    1
   10    1    1
      ⋮

info = struct with fields:
                G: 7774
               Gd: 7774
        NREPerPRB: 156
    DMRSSymbolSet: 2
    PTRSSymbolSet: [0 1 3 4 5 6 7 8 9 10 11 12 13]

ptrsInd = 1352x3 uint32 matrix

    13     1     1
    37     1     1
    61     1     1
    85     1     1
   109     1     1
   133     1     1
   157     1     1
   181     1     1
   205     1     1
   229     1     1
      ⋮
```

Generate PUSCH Symbols and Indices

Create a carrier configuration object with default properties. This object corresponds to 30 kHz of subcarrier spacing and 20 MHz transmission bandwidth.

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 30;
carrier.NSizeGrid = 51;
```

Create a PUSCH configuration object with specified properties. When transform precoding is 1, the waveform type is discrete fourier transform spread orthogonal frequency division multiplexing (DFT-s-OFDM).

```
pusch = nrPUSCHConfig;
pusch.NStartBWP = 10;
```

```
pusch.NSizeBWP = 41;
pusch.Modulation = '16QAM';
pusch.NID = []; % Set NID equal to the NCellID property of carrier.
pusch.PRBSet = 0:5;
pusch.TransformPrecoding = 1;
pusch.FrequencyHopping = 'intraSlot';
pusch.SecondHopStartPRB = 3;
```

Generate PUSCH indices, setting the index orientation with respect to the carrier grid.

```
[ind,info] = nrPUSCHIndices(carrier,pusch,'IndexOrientation','carrier')
```

```
ind = 864x1 uint32 column vector

   121
   122
   123
   124
   125
   126
   127
   128
   129
   130
      ⋮

info = struct with fields:
                G: 3456
               Gd: 864
        NREPerPRB: 144
    DMRSSymbolSet: [2 7]
    PTRSSymbolSet: [1x0 double]
```

Generate PUSCH symbols of data type single.

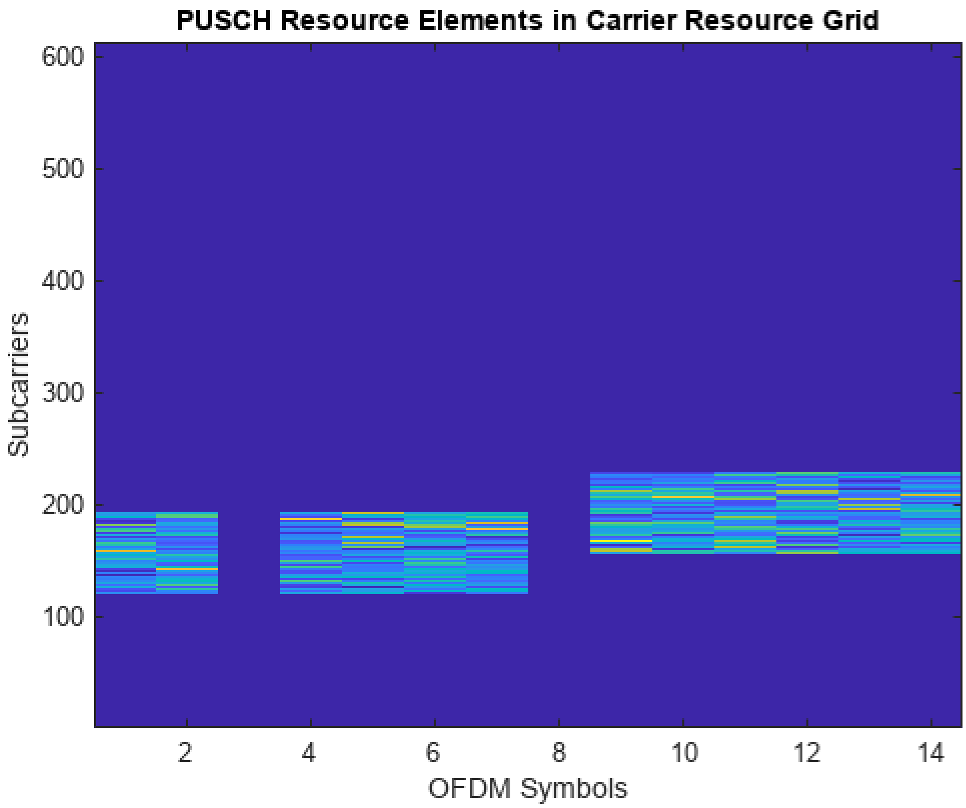
```
numDataBits = info.G;
cws = randi([0 1],numDataBits,1);
sym = nrPUSCH(carrier,pusch,cws,'OutputDataType','single')
```

```
sym = 864x1 single column vector

  -0.7454 + 0.2981i
   0.3406 - 0.2312i
  -0.1153 + 0.2756i
   1.1921 - 0.3658i
  -0.3968 - 0.0277i
  -0.8788 - 0.6493i
  -0.8737 + 0.8318i
  -0.5764 + 0.0269i
  -1.6638 + 0.0482i
  -1.0270 - 0.1347i
      ⋮
```

Plot the generated symbols and indices on the carrier resource grid.

```
grid = complex(zeros([carrier.NSizeGrid*12 carrier.SymbolsPerSlot pusch.NumLayers]));
grid(ind) = sym;
```

```
imagesc(abs(grid(:,:,1)));
axis xy;
xlabel('OFDM Symbols');
ylabel('Subcarriers');
title('PUSCH Resource Elements in Carrier Resource Grid');
```



Version History

Introduced in R2020a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.213. "NR; Physical layer procedures for control." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrPUSCH` | `nrPUSCHDecode` | `nrPUSCHIndices` | `nrPUSCHDMRS` | `nrPUSCHDMRSIndices` | `nrPUSCHPTRS` | `nrPUSCHPTRSIndices` | `nrULSCHMultiplex` | `nrULSCHDemultiplex` | `nrULSCHInfo`

Objects
`nrCarrierConfig` | `nrPUSCHDMRSConfig` | `nrPUSCHPTRSConfig`

nrPUSCHDMRSConfig

PUSCH DM-RS configuration parameters

Description

The `nrPUSCHDMRSConfig` object sets demodulation reference signal (DM-RS) configuration parameters for a physical uplink shared channel (PUSCH), as defined in TS 38.211 Section 6.4.1.1 [1].

The object defines the properties of PUSCH DM-RS symbols and indices generation and the resource elements pattern not used for data in DM-RS symbol locations. The read-only properties of this object provide the DM-RS subcarrier locations within a resource block (RB), code division multiplexing (CDM) groups, and time and frequency weights for DM-RS symbols. By default, the object specifies a single symbol DM-RS at symbol index 2 (0-based) with configuration type 1 and antenna port 0. Use this object when setting the `DMRS` property of the `nrPUSCHConfig` or `nrWavegenPUSCHConfig` objects.

Creation

Syntax

```
dmrs = nrPUSCHDMRSConfig
dmrs = nrPUSCHDMRSConfig(Name,Value)
```

Description

`dmrs = nrPUSCHDMRSConfig` creates a DM-RS configuration object for a PUSCH with default properties.

`dmrs = nrPUSCHDMRSConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'DMRSConfigurationType',1,'DMRSLength',2` specifies a double-symbol DM-RS with configuration type as 1. Unspecified properties take their default values.

Properties

`DMRSConfigurationType` — DM-RS configuration type
1 (default) | 2

DM-RS configuration type, specified as `1` or `2`. This property is the higher-layer parameter *dmrs-Type*.

This property value must be `1` when nrPUSCHDMRSConfig is a property of `nrPUSCHConfig` with `TransformPrecoding` property set to `1`.

Data Types: `double`

`DMRSTypeAPosition` — Position of first DM-RS OFDM symbol
2 (default) | 3

Position of first DM-RS OFDM symbol, provided by higher layer parameter *dmrs-TypeA-Position*, specified as `2` or `3`.

This property is applicable when nrPUSCHDMRSConfig is a property of `nrPUSCHConfig` object with `MappingType` property value set to `'A'`.

Data Types: `double`

`DMRSAdditionalPosition` — Maximum number of DM-RS additional positions
`0` (default) | `1` | `2` | `3`

Maximum number of DM-RS additional positions, specified as `0`, `1`, `2`, or `3`. This property is the higher layer parameter *dmrs-AdditionalPosition*.

This property value must be `0` or `1` when nrPUSCHDMRSConfig is a property of `nrPUSCHConfig` object with `FrequencyHopping` property set to `'intraSlot'`.

Data Types: `double`

`DMRSLength` — Number of consecutive front-loaded DM-RS OFDM symbols
`1` (default) | `2`

Number of consecutive front-loaded DM-RS OFDM symbols, specified as `1` (single-symbol DM-RS) or `2` (double-symbol DM-RS).

This property value must be `1` when nrPUSCHDMRSConfig is a property of `nrPUSCHConfig` object with `FrequencyHopping` property set to `'intraSlot'`.

Data Types: `double`

`CustomSymbolSet` — DM-RS OFDM symbol locations
`[]` (default) | integer from 0 to 13 | vector of nonnegative integers

DM-RS OFDM symbol locations that are 0-based, specified as one of these options.

- Integer from 0 to 13 — For one DM-RS symbol
- Vector of nonnegative integers from 0 to 13 — For multiple DM-RS symbols

Each input symbol location is assumed to be a single-symbol DM-RS within the physical shared channel symbol allocation.

The default value, `[]`, corresponds to the DM-RS symbols locations as per TS 38.211 Table 6.4.1.1.3-3, 6.4.1.1.3-4, or 6.4.1.1.3-6 [1]. Setting this property overrides the corresponding DM-RS symbol locations in these standard lookup tables.

Data Types: `double`

`DMRSPortSet` — DM-RS antenna ports
`[]` (default) | integer scalar | vector of nonnegative integers

DM-RS antenna ports, specified as one of these options.

- Integer from 0 to 11 — For a single antenna port
- Vector of nonnegative integers from 0 to 11 — For multiple antenna ports

Nominal antenna ports supported depend on `DMRSLength` and `DMRSConfigurationType` property values, as shown in this table.

DMRSLength Value	DMRSConfigurationType Value	Nominal Range of Antenna Ports Supported
1	1	[0, 3]
	2	[0, 5]
2	1	[0, 7]
	2	[0, 11]

The default value of [] implies that DM-RS antenna port is equal to 0.

When nrPUSCHDMRSConfig is a property of nrPUSCHConfig object, [] implies that DMRSPortSet is in the range from 0 to NumLayers–1.

Data Types: double

NIDNSCID — DM-RS scrambling identities for CP-OFDM
[] (default) | 1-by-2 integer vector | scalar integer

DM-RS scrambling identities for CP-OFDM (NID^0 and NID^1), specified as one of these options.

- 1-by-2 integer vector of values from 0 to 65,535 — The vector elements define NID^0 and NID^1.
- Scalar integer from 0 to 65,535 — This option specifies equal values for both NID^0 and NID^1.
- [] — Use this option to set the DM-RS scrambling identity to the physical layer cell identity, specified by the NCellID property of the carrier configuration.

Dependencies

This property applies when the TransformPrecoding property of the nrPUSCHConfig object is set to 0.

Data Types: double

NRSID — DM-RS scrambling identity for DFT-s-OFDM
[] (default) | integer from 0 to 1007

DM-RS scrambling identity for DFT-s-OFDM, specified as one of these options.

- Integer from 0 to 1007 — If NRSID is the higher-layer parameter *nPUSCH-Identity*.
- [] — Use this option to set NRSID automatically.
  - If you set the DMRSUplinkTransformPrecodingR16 property to 1 and the Modulation property of the PUSCH configuration object to 'pi/2-BPSK', the object sets NRSID to one of the *NID* values defined by the NIDNSCID property.
  - If you set the DMRSUplinkTransformPrecodingR16 property to 0 or the Modulation property of the PUSCH configuration object to a value that is different than 'pi/2-BPSK', the object sets NRSID to the NCellID property value of the nrCarrierConfig object.

Dependencies

This property applies when the TransformPrecoding property of the nrPUSCHConfig object is set to 1.

Data Types: double

NSCID — DM-RS scrambling initialization for CP-OFDM

0 (default) | 1

DM-RS scrambling initialization for CP-OFDM, specified as 0 or 1.

Dependencies

This property applies when the TransformPrecoding property of the nrPUSCHConfig object is set to 0.

Data Types: double

GroupHopping — Group hopping configuration

0 (default) | 1

Group hopping configuration, specified as one of these options.

- 0 — Group hopping is disabled.
- 1 — Group hopping is enabled.

Dependencies

This property applies when the TransformPrecoding property of the nrPUSCHConfig object is set to 1 and SequenceHopping is set to 0.

Data Types: logical | double

SequenceHopping — Sequence hopping configuration

0 (default) | 1

Sequence hopping configuration, specified as one of these options.

- 0 — SequenceHopping is disabled.
- 1 — SequenceHopping is enabled.

Dependencies

This property applies when the TransformPrecoding property of the nrPUSCHConfig object is set to 1 and GroupHopping is set to 0.

Data Types: logical | double

NumCDMGroupsWithoutData — Number of CDM groups without data

2 (default) | 1 | 3

Number of DM-RS CDM groups without data, specified as 1, 2, or 3.

Each value indicates a different set of CDM group numbers, according to TS 38.214 Section 6.2.2 [2].

- 1 — CDM group number 0
- 2 — CDM group numbers 0 and 1
- 3 — CDM group numbers 0, 1, and 2

When TransformPrecoding property of the nrPUSCHConfig object is set to 1, this property value must be 2.

Data Types: double

DMRSUplinkR16 — Release 16 low PAPR DM-RS sequence
0 (default) | 1

Release 16 low peak-to-average-power ratio (PAPR) DM-RS sequence, specified as one of these logical values.

- 0 (false) — Disable the use of low PAPR DM-RS.
- 1 (true) — Enable the use of low PAPR DM-RS. The DM-RS sequence generation depends on the DM-RS antenna port indices (DMRSPortSet) as well as the scrambling initialization (NSCID).

Dependencies

This property applies when the TransformPrecoding property of the nrPUSCHConfig object is set to 0.

Data Types: double

DMRSUplinkTransformPrecodingR16 — Release 16 low PAPR DM-RS sequence for DFT-s-OFDM
0 (default) | 1

Release 16 low PAPR DM-RS sequence for DFT-s-OFDM, specified as one of these logical values.

- 0 (false) — Disable the use of low PAPR DM-RS for DFT-s-OFDM. The DM-RS sequence generation uses type 1 low PAPR sequences.
- 1 (true) — Enable the use of low PAPR DM-RS for DFT-s-OFDM. The DM-RS sequence generation uses type 2 low PAPR sequences.

Dependencies

This property applies when the TransformPrecoding property of the nrPUSCHConfig object is set to 1 and the Modulation property is set to 'pi/2-BPSK'.

Data Types: double

CDMGroups — CDM group numbers corresponding to each port
0 (default) | integer from 0 to 2 | row vector of integers

This property is read-only.

CDM group numbers corresponding to each DM-RS port, specified as one of these options.

- Integer from 0 to 2 — When DMRSPortSet property is a scalar, specifying single DM-RS antenna port.
- Row vector of integers from 0 to 2 — When DMRSPortSet property is a vector, specifying multiple DM-RS antenna ports. Each element corresponds to a CDM group number for that port.

Value of this property depends on the DMRSConfigurationType according to TS 38.211 Table 6.4.1.1.3-1 or 6.4.1.1.3-2 [1].

Data Types: double

DeltaShifts — Delta shifts corresponding to each CDM group
0 (default) | integer from the set {0, 1, 2, 4} | row vector of integers

This property is read-only.

Delta shifts corresponding to each CDM group, specified as one of these options.

- Integer from the set {0, 1, 2, 4} — When `DMRSPortSet` property is a scalar, specifying single DM-RS antenna port.
- Row vector of integers from the set {0, 1, 2, 4} — When `DMRSPortSet` property is a vector, specifying multiple DM-RS antenna ports. Each element corresponds to the delta shift to be applied for that port.

Value of this property depends on the `DMRSConfigurationType` according to TS 38.211 Table 6.4.1.1.3-1 or 6.4.1.1.3-2 [1].

Data Types: `double`

`FrequencyWeights` — Frequency weights
`[1; 1]` (default) | column vector of integers | matrix of integers

This property is read-only.

Frequency weights for the DM-RS symbols, specified as one of these options.

- Column vector of integers — When `DMRSPortSet` property is a scalar, specifying single DM-RS antenna port.
- Matrix of integers — When `DMRSPortSet` property is a vector, specifying multiple DM-RS antenna ports. Each column corresponds to the weights for that port.

Value of this property depends on the `DMRSConfigurationType` according to TS 38.211 Table 6.4.1.1.3-1 or 6.4.1.1.3-2 [1].

Data Types: `double`

`TimeWeights` — Time weights
`[1; 1]` (default) | column vector of integers | matrix of integers

This property is read-only.

Time weights for to the DM-RS symbols, specified as one of these options.

- Column vector of integers — When `DMRSPortSet` property is a scalar, specifying single DM-RS antenna port.
- Matrix of integers — When `DMRSPortSet` property is a vector, specifying multiple DM-RS antenna ports. Each column corresponds to the weights for that port.

Value of this property depends on the `DMRSConfigurationType` according to TS 38.211 Table 6.4.1.1.3-1 or 6.4.1.1.3-2 [1].

Data Types: `double`

`DMRSSubcarrierLocations` — Subcarrier locations in RB for each port
`[0; 2; 4; 6; 8; 10]` (default) | column vector of integers | matrix of integers

This property is read-only.

Subcarrier locations in an RB for each port, specified as one of these options.

- Column vector of integers — When `DMRSPortSet` property is a scalar, specifying single DM-RS antenna port.

- Matrix of integers — When `DMRSPortSet` property is a vector, specifying multiple DM-RS antenna ports. Each column corresponds to the subcarrier locations for that port.

Data Types: `double`

`CDMLengths` — CDM arrangement for reference signals
`[1 1]` (default) | two-element row vector

This property is read-only.

CDM arrangement for reference signals, specified as the comma-separated pair consisting of `'CDMLengths'` and a two-element row vector of nonnegative integers [*FD TD*]. Array elements *FD* and *TD* specify the length of CDM despreading in the frequency domain (FD-CDM) and time domain (TD-CDM), respectively. A value of `1` for an element specifies no CDM.

Data Types: `double`

Examples

Create PUSCH DM-RS Object

Create a physical uplink shared channel (PUSCH) specific demodulation reference signal (DM-RS) object, `dmrs`.

Specify a single-symbol DMRS with number of DM-RS additional positions as 3, sequence hopping as 1, and having antenna ports as 0 and 4.

```
dmrs = nrPUSCHDMRSConfig;
dmrs.DMRSLength = 1;
dmrs.DMRSAdditionalPosition = 3;
dmrs.SequenceHopping = 1;
dmrs.DMRSPortSet = [0 4];
```

View the corresponding `dmrs` properties.

```
disp(dmrs)
```

```
  nrPUSCHDMRSConfig with properties:

             DMRSConfigurationType: 1
                 DMRSTypeAPosition: 2
            DMRSAdditionalPosition: 3
                        DMRSLength: 1
                   CustomSymbolSet: []
                       DMRSPortSet: [0 4]
                          NIDNSCID: []
                             NSCID: 0
                      GroupHopping: 0
                   SequenceHopping: 1
                             NRSID: []
           NumCDMGroupsWithoutData: 2
                     DMRSUplinkR16: 0
    DMRSUplinkTransformPrecodingR16: 0

   Read-only properties:
                         CDMGroups: [0 0]
```

```
                DeltaShifts: [0 0]
           FrequencyWeights: [2x2 double]
                TimeWeights: [2x2 double]
    DMRSSubcarrierLocations: [6x2 double]
                 CDMLengths: [1 1]
```

Generate PUSCH DM-RS Symbols for CP-OFDM

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier with 15 kHz subcarrier spacing.

```
carrier = nrCarrierConfig;
```

Create a physical uplink shared channel (PUSCH) configuration object with specified properties. When transform precoding is set to 0, the waveform type is cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM).

```
pusch = nrPUSCHConfig;
pusch.TransformPrecoding = 0;
pusch.TransmissionScheme = 'codebook';
pusch.NumAntennaPorts = 4;
pusch.TPMI = 0;
```

Configure PUSCH demodulation reference signal (DM-RS) with specified parameters.

```
pusch.DMRS.DMRSAdditionalPosition = 1;
pusch.DMRS.DMRSTypeAPosition = 2;
pusch.DMRS.DMRSPortSet = 2;
pusch.DMRS.NIDNSCID = 10;
pusch.DMRS.NSCID = 1;
```

Generate DM-RS symbols associated with PUSCH of `single` data type.

```
sym = nrPUSCHDMRS(carrier,pusch,'OutputDataType','single')
```

```
sym = 624x4 single matrix

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

Generate PUSCH DM-RS Indices for Codebook-Based Transmission

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier with 15 kHz subcarrier spacing.

```
carrier = nrCarrierConfig;
```

Create a physical uplink shared channel (PUSCH) configuration object with specified properties. When transform precoding is set to 0, the waveform type is cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM).

```
pusch = nrPUSCHConfig;
pusch.TransformPrecoding = 0;
pusch.TransmissionScheme = 'codebook';
pusch.NumAntennaPorts = 4;
pusch.TPMI = 0;
```

Configure PUSCH demodulation reference signal (DM-RS) object with specified parameters.

```
pusch.DMRS.DMRSAdditionalPosition = 2;
pusch.DMRS.DMRSTypeAPosition = 2;
pusch.DMRS.DMRSPortSet = 3;
pusch.DMRS.NIDNSCID = 15;
pusch.DMRS.NSCID = 1;
```

Generate DM-RS indices associated to PUSCH of subscript indexing form.

```
ind = nrPUSCHDMRSIndices(carrier,pusch,'IndexStyle','subscript')
```

```
ind = 3744x3 uint32 matrix

     2     3     1
     4     3     1
     6     3     1
     8     3     1
    10     3     1
    12     3     1
    14     3     1
    16     3     1
    18     3     1
    20     3     1
      ⋮
```

Generate PUSCH DM-RS Symbols and Indices

Create a carrier configuration with 30 kHz subcarrier spacing and 5 MHz transmission bandwidth.

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 30;
carrier.NSizeGrid = 11;
```

Create a physical uplink shared channel (PUSCH) configuration object with specified properties. When transform precoding is set to 1, the waveform type is discrete fourier transform spread orthogonal frequency division multiplexing (DFT-s-OFDM).

```
pusch = nrPUSCHConfig;
pusch.NSizeBWP = 9;
pusch.NStartBWP = 1;
pusch.PRBSet = 0:3;
pusch.TransformPrecoding = 1;
pusch.FrequencyHopping = 'intraSlot';
pusch.SecondHopStartPRB = 5;
```

Create a PUSCH demodulation reference signal (DM-RS) object with specified properties.

```
dmrs = nrPUSCHDMRSConfig;
dmrs.DMRSAdditionalPosition = 1;
dmrs.DMRSTypeAPosition = 2;
dmrs.DMRSPortSet = 3;
dmrs.GroupHopping = 1;
dmrs.SequenceHopping = 0;
dmrs.NRSID = 10;
```

Assign the PUSCH DM-RS configuration object to DMRS property of PUSCH configuration object.

```
pusch.DMRS = dmrs;
```

Generate PUSCH DM-RS symbols and indices for the specified carrier, PUSCH configuration, and output formatting `name-value` pair argument.

```
sym = nrPUSCHDMRS(carrier,pusch,'OutputDataType','single')
```

```
sym = 96x1 single column vector

  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
      ⋮
```

```
ind = nrPUSCHDMRSIndices(carrier,pusch,'IndexBase','0based','IndexOrientation','bwp')
```

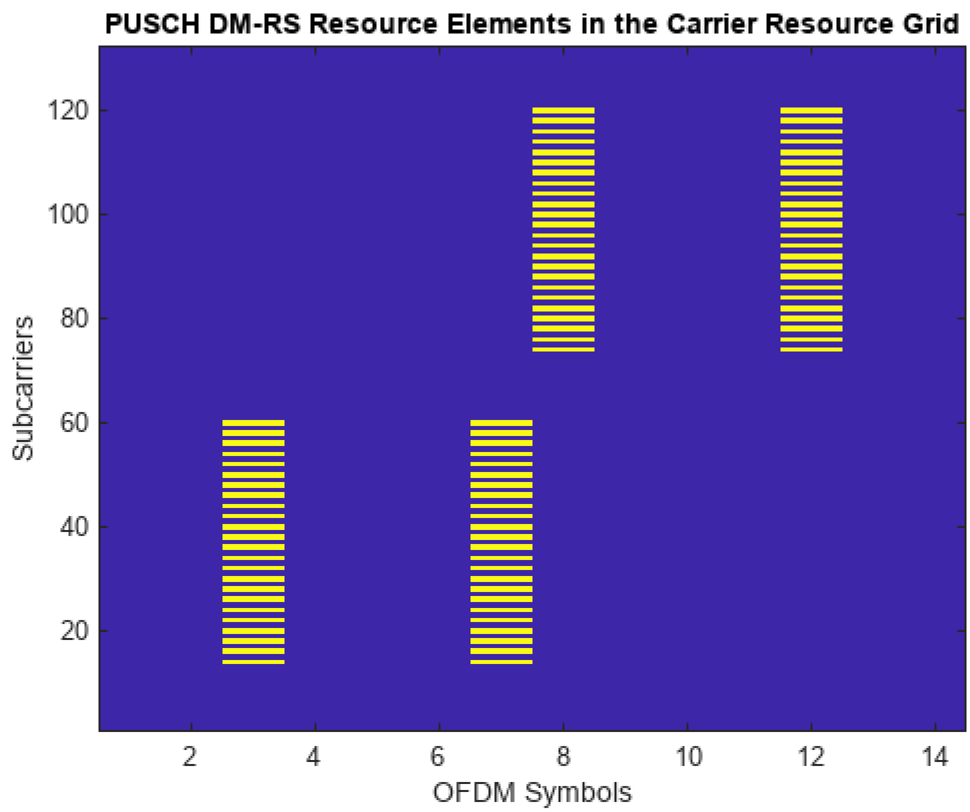
```
ind = 96x1 uint32 column vector

   217
   219
   221
   223
   225
   227
   229
   231
   233
   235
     ⋮
```

Create a bandwidth part (BWP) grid, and then map the DM-RS symbols on the grid.

```
bwp = complex(zeros([pusch.NSizeBWP*12 carrier.SymbolsPerSlot pusch.NumLayers]));
bwp(ind+1) = sym; % Map the DM-RS symbols
```

Map the BWP to the carrier resource grid, and then display the carrier grid.

```
grid = complex(zeros([carrier.NSizeGrid*12 carrier.SymbolsPerSlot pusch.NumLayers])); % Create ca
offset = pusch.NStartBWP-carrier.NStartGrid; % BWP start location in the carrier grid
grid(offset*12+1:(offset+pusch.NSizeBWP)*12,:,:) = bwp;
imagesc(abs(grid(:,:,1)));
axis xy;
xlabel('OFDM Symbols');
ylabel('Subcarriers');
title('PUSCH DM-RS Resource Elements in the Carrier Resource Grid');
```



Version History

Introduced in R2020a

Release 16 updates for low PAPR sequences and DM-RS scrambling identities

- The `DMRSUplinkR16` and `DMRSUplinkTransformPrecodingR16` properties enable low peak-to-average-power (PAPR) PUSCH DM-RS sequences, as defined in Release 16 of TS 38.211 Section 6.4.1.1.

- The NIDNSCID property supports dynamic ID selection.
- The default value of the NRSID property depends on the DMRSUplinkTransformPrecodingR16 object property and the Modulation property of the PUSCH configuration object.

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*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
nrPUSCH | nrPUSCHDMRSIndices | nrPUSCHDMRS

Objects
nrCarrierConfig | nrPUSCHConfig | nrPUSCHPTRSConfig | nrWavegenPUSCHConfig

nrPUSCHPTRSConfig

PUSCH PT-RS configuration parameters

Description

The `nrPUSCHPTRSConfig` object sets phase tracking reference signal (PT-RS) configuration parameters for the physical uplink shared channel (PUSCH), as defined in TS 38.211 Section 6.4.1.2 [1]. This object bundles all the properties involved in PUSCH PT-RS symbols and indices generation. By default, the object defines the PT-RS with a frequency density of 2 and time density of 1. Use this object when setting the `PTRS` property of the `nrPUSCHConfig` or `nrWavegenPUSCHConfig` objects.

Creation

Syntax

```
ptrs = nrPUSCHPTRSConfig
ptrs = nrPUSCHPTRSConfig(Name,Value)
```

Description

`ptrs = nrPUSCHPTRSConfig` creates a PUSCH-specific PT-RS configuration object with default properties.

`ptrs = nrPUSCHPTRSConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'TimeDensity',2,'FrequencyDensity',4` sets the time density to 2 and frequency density to 4. Unspecified properties take their default values.

Properties

`TimeDensity` — PT-RS time density
1 (default) | 2 | 4

PT-RS time density, specified as `1`, `2`, or `4`. This property is the higher layer parameter *timeDensity*.

Data Types: `double`

`FrequencyDensity` — PT-RS frequency density
2 (default) | 4

PT-RS frequency density, specified as `2` or `4`. This property is the higher layer parameter *frequencyDensity*.

Dependencies

This property applies only when `nrPUSCHPTRSConfig` is a property of `nrPUSCHConfig` with `TransformPrecoding` set to `0`.

Data Types: `double`

`NumPTRSSamples` — Number of PT-RS samples
2 (default) | 4

Number of PT-RS samples per PT-RS group, specified as `2` or `4`. This property is the higher layer parameter *sampleDensity*.

Dependencies

This property applies only when `nrPUSCHPTRSConfig` is a property of `nrPUSCHConfig` with `TransformPrecoding` set to `1`.

Data Types: `double`

`NumPTRSGroups` — Number of PT-RS groups
2 (default) | 4 | 8

Number of PT-RS groups, specified as `2`, `4`, or `8`. This property is the higher layer parameter *sampleDensity*.

When this property is set to `8`, the number of PT-RS samples set by the `NumPTRSSamples` property must be set to 4.

Dependencies

This property applies only when `nrPUSCHPTRSConfig` is a property of `nrPUSCHConfig` with `TransformPrecoding` set to `1`.

Data Types: `double`

`REOffset` — Resource element offset
`'00'` (default) | `'01'` | `'10'` | `'11'`

Resource element offset, specified as `'00'`, `'01'`,`'10'`, or `'11'`. This property is the higher layer parameter *resourceElementOffset*.

Dependencies

This property applies only when `nrPUSCHPTRSConfig` is a property of `nrPUSCHConfig` with `TransformPrecoding` set to `0`.

Data Types: `char` | `string`

`PTRSPortSet` — PT-RS antenna port set
`[]` (default) | nonnegative integer | two-element vector of nonnegative integers

PT-RS antenna port set, specified as a two-element vector of nonnegative integers. Specify `[]` to set this property to the lowest value in the `DMRSPortSet` property of `nrPUSCHDMRSConfig` object. This usage of `[]` value is applicable only when `nrPUSCHDMRSConfig` object is used as a property of `nrPUSCHConfig` object.

Dependencies

This property applies only when `nrPUSCHPTRSConfig` is a property of `nrPUSCHConfig` with `TransformPrecoding` set to `0`.

Data Types: `double`

`NID` — PT-RS scrambling identity
`[]` (default) | integer from 0 to 1007

PT-RS scrambling identity, specified as an integer from 0 to 1007. Specify `[]` to set this property equal to the `NRSID` property of `nrPUSCHDMRSConfig` object.

Dependencies

This property applies only when `nrPUSCHPTRSConfig` is a property of `nrPUSCHConfig` with `TransformPrecoding` set to `1`.

Data Types: `double`

Examples

Create PUSCH PT-RS Object

Create a default PUSCH configuration object. Enable the PT-RS configuration and transform precoding for a DFT-s-OFDM waveform.

```
pusch = nrPUSCHConfig;
pusch.EnablePTRS = 1;
pusch.TransformPrecoding = 1;
```

Create a default PT-RS configuration object for the PUSCH. Set number of PT-RS samples to 4, number of PT-RS groups to 8, and PT-RS scrambling identity to 750.

```
ptrs = nrPUSCHPTRSConfig;
ptrs.NumPTRSSamples = 4;
ptrs.NumPTRSGroups = 8;
ptrs.NID = 750;
```

Assign the PUSCH PT-RS configuration object to the PTRS property of PUSCH configuration object.

```
pusch.PTRS = ptrs;
```

Display the properties for PUSCH PT-RS configuration object and PUSCH configuration object, respectively.

```
disp(pusch)
```

```
  nrPUSCHConfig with properties:

              NSizeBWP: []
             NStartBWP: []
            Modulation: 'QPSK'
             NumLayers: 1
           MappingType: 'A'
      SymbolAllocation: [0 14]
                PRBSet: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 ... ]
    TransformPrecoding: 1
    TransmissionScheme: 'nonCodebook'
       NumAntennaPorts: 1
                  TPMI: 0
      FrequencyHopping: 'neither'
     SecondHopStartPRB: 1
         BetaOffsetACK: 20
        BetaOffsetCSI1: 6.2500
        BetaOffsetCSI2: 6.2500
            UCIScaling: 1
```

```
                NID: []
               RNTI: 1
             NRAPID: []
               DMRS: [1x1 nrPUSCHDMRSConfig]
         EnablePTRS: 1
               PTRS: [1x1 nrPUSCHPTRSConfig]
```

```
disp(pusch.PTRS)
```

```
  nrPUSCHPTRSConfig with properties:

       TimeDensity: 1
    NumPTRSSamples: 4
     NumPTRSGroups: 8
               NID: 750
```

Generate PUSCH PT-RS Indices for Codebook-Based Transmission

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier with 15 kHz subcarrier spacing.

```
carrier = nrCarrierConfig;
```

Create a PUSCH configuration object with codebook-based transmission and enable the PT-RS configuration. Set the number of antenna ports to 4 and transform precoding to 0. When transform precoding is 0, the waveform type is cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM).

```
pusch = nrPUSCHConfig;
pusch.TransformPrecoding = 0;
pusch.TransmissionScheme = 'codebook';
pusch.NumAntennaPorts = 4;
pusch.EnablePTRS = 1;
```

Create a PUSCH phase tracking reference signal (PT-RS) configuration object with specified properties.

```
ptrs = nrPUSCHPTRSConfig;
ptrs.TimeDensity = 2;
ptrs.FrequencyDensity = 4;
ptrs.REOffset = '11';
```

Assign the PUSCH PT-RS configuration object to PTRS property of PUSCH configuration object.

```
pusch.PTRS = ptrs;
```

Generate PUSCH PT-RS indices in subscript form

```
ind = nrPUSCHPTRSIndices(carrier,pusch,'IndexStyle','subscript')
```

```
ind = 312x3 uint32 matrix

    21     1     1
    69     1     1
   117     1     1
   165     1     1
```

```
   213     1     1
   261     1     1
   309     1     1
   357     1     1
   405     1     1
   453     1     1
      ⋮
```

Generate PUSCH PT-RS Symbols and Indices

Create a carrier configuration object with 30 kHz subcarrier spacing and 5 MHz transmission bandwidth.

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 30;
carrier.NSizeGrid = 11;
```

Create a PUSCH configuration object with intraslot frequency hopping and enable the PT-RS configuration. Set the transform precoding to 1, starting physical resource blocks (PRB) index of the second hop to 3 and PRB set to 0:5. When transform precoding is 1, the waveform type is discrete fourier transform spread orthogonal frequency division multiplexing (DFT-s-OFDM).

```
pusch = nrPUSCHConfig;
pusch.PRBSet = 0:5;
pusch.TransformPrecoding = 1;
pusch.FrequencyHopping = 'intraSlot';
pusch.SecondHopStartPRB = 3;
pusch.EnablePTRS = 1;
```

Create a PUSCH phase tracking reference signal (PT-RS) configuration object with specified properties.

```
ptrs = nrPUSCHPTRSConfig;
ptrs.TimeDensity = 2;
ptrs.NumPTRSSamples = 4;
ptrs.NumPTRSGroups = 8;
ptrs.NID = 750;
```

Assign the PUSCH PT-RS configuration object to PTRS property of PUSCH configuration object.

```
pusch.PTRS = ptrs;
```

Generate PUSCH PT-RS symbols of data type single.

```
sym = nrPUSCHPTRS(carrier,pusch,'OutputDataType','single')
```

```
sym = 192x1 single column vector

   0.7071 + 0.7071i
  -0.7071 + 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
```

```
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
      ⋮
```

Generate PUSCH PT-RS indices in subscript form.

```
ind = nrPUSCHPTRSIndices(carrier,pusch,'IndexStyle','subscript')
```

```
ind = 192x3 uint32 matrix

     1     1     1
     2     1     1
     3     1     1
     4     1     1
    12     1     1
    13     1     1
    14     1     1
    15     1     1
    21     1     1
    22     1     1
      ⋮
```

Version History

Introduced in R2020a

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

nrPUSCH | nrPUSCHDecode | nrPUSCHIndices | nrPUSCHPTRS | nrPUSCHPTRSIndices

Objects

nrCarrierConfig | nrPUSCHDMRSConfig | nrPUSCHConfig | nrWavegenPUSCHConfig

nrSCSCarrierConfig

SCS carrier configuration parameters for 5G waveform generation

Description

The `nrSCSCarrierConfig` object sets subcarrier spacing (SCS) carrier configuration parameters for a specific OFDM numerology. Use this object to set the `SCSCarriers` property of the `nrDLCarrierConfig` object or the `SCSCarriers` property of the `nrULCarrierConfig` object when configuring 5G waveform generation.

This object defines the carrier SCS, bandwidth, and offset parameters from point A, which is the center of subcarrier 0 in the common resource block 0 (CRB 0). By default, this object specifies a 10 MHz carrier corresponding to 52 resource blocks (RBs) and 15 kHz SCS.

Creation

Syntax

```
carrier = nrSCSCarrierConfig
carrier = nrSCSCarrierConfig(Name,Value)
```

Description

`carrier = nrSCSCarrierConfig` creates a default SCS carrier configuration object for 5G waveform generation.

`carrier = nrSCSCarrierConfig(Name,Value)` sets properties on page 3-156 using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'SubcarrierSpacing',30,'NSizeGrid',273` specifies a 100 MHz carrier corresponding to 273 resource blocks (RBs) and 30 kHz SCS.

Properties

`SubcarrierSpacing` — Subcarrier spacing in kHz
15 (default) | 30 | 60 | 120 | 240

Subcarrier spacing in kHz, for all channels and reference signals of the carrier, specified as `15`, `30`, `60`, `120`, or `240`.

Data Types: `double`

`NSizeGrid` — Number of RBs in carrier resource grid
52 (default) | integer from 1 to 275

Number of RBs in the carrier resource grid, specified as an integer from 1 to 275. The default value of `52` corresponds to the maximum number of RBs of a 10 MHz carrier with 15 kHz SCS.

Data Types: `double`

`NStartGrid` — Start of carrier resource grid relative to CRB 0
`0` (default) | integer from 0 to 2199

Start of carrier resource grid relative to CRB 0, specified as an integer from 0 to 2199. This property is the higher-layer parameter *offsetToCarrier*.

Data Types: `double`

Examples

Configure SCS Carriers with Mixed Numerologies

Create a default SCS carrier configuration object, which configures a 10 MHz carrier with 15 kHz SCS.

```
scs1 = nrSCSCarrierConfig;
```

Create an SCS carrier configuration object, which configures a 100 MHz carrier with 30 kHz SCS.

```
scs2 = nrSCSCarrierConfig('SubcarrierSpacing',30,'NSizeGrid',273);
```

Create a downlink carrier configuration object, specifying the two SCS carrier configurations.

```
cfgDL = nrDLCarrierConfig('SCSCarriers',{scs1,scs2});
```

Version History

Introduced in R2020b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrWaveformGenerator`

Objects
`nrWavegenBWPConfig` | `nrDLCarrierConfig` | `nrULCarrierConfig`

nrSearchSpaceConfig

Search space set configuration parameters

Description

The `nrSearchSpaceConfig` object sets search space set configuration parameters for the physical downlink control channel (PDCCH), as defined in TS 38.213 Section 10 [1]. Use this object when setting the `SearchSpace` property of the `nrPDCCHConfig` or `nrDLCarrierConfig` objects.

Creation

Syntax

```
cfgSS = nrSearchSpaceConfig
cfgSS = nrSearchSpaceConfig(Name,Value)
```

Description

`cfgSS = nrSearchSpaceConfig` creates a search space set configuration object with default properties.

`cfgSS = nrSearchSpaceConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'Duration',3,'NumCandidates',[5 5 3 2 1]` configures the search space set over three consecutive slots with the specified number of candidates at each aggregation level. Unspecified properties take their default values.

Properties

`SearchSpaceID` — Search space set ID
`1` (default) | nonnegative integer

Search space set ID, specified as a nonnegative integer.

Data Types: `double`

`Label` — Name of search space set configuration
`'SearchSpace1'` (default) | character array | string scalar

Name of the search space set configuration, specified as a character array or string scalar. Use this property to set a description to the search space set configuration.

Data Types: `char` | `string`

`CORESETID` — Associated CORESET ID for search space
`1` (default) | integer from 0 to 11

Associated CORESET ID for the search space, specified as an integer from 0 to 11. When this object and `nrCORESETConfig` object specify the `SearchSpace` and `CORESET` properties, respectively, of the same `nrPDCCHConfig` object, the `CORESETID` properties of these objects must match.

Data Types: `double`

`SearchSpaceType` — Search space type
`'ue'` (default) | `'common'`

Search space type, specified as `'ue'` or `'common'`.

Data Types: `char` | `string`

`StartSymbolWithinSlot` — First symbol of CORESET location
`0` (default) | integer from 0 to 13

First symbol of CORESET location in each monitored slot, specified as an integer from 0 to 13. Values from 0 to 11 apply only to extended cyclic prefix. When setting this property, the CORESET must fit within a single slot in terms of the associated CORESET duration.

Data Types: `double`

`SlotPeriodAndOffset` — Slot period and offset for PDCCH monitoring
`[1 0]` (default) | 1-by-2 integer vector

Slot period and offset for PDCCH monitoring, specified as a 1-by-2 integer vector. The first vector element specifies the period. The period must be a positive integer greater than or equal to the search space duration specified by the `Duration` property. The second vector element specifies the offset with respect to the period. The offset must be a nonnegative integer less than the period (the first vector element).

Data Types: `double`

`Duration` — Search space duration in slots
`1` (default) | integer from 0 to 2559

Search space duration in slots, specified as an integer from 0 to 2559. This property specifies the number of consecutive slots that the search space lasts within each period. The value of this property must be less than or equal to the slot period specified by the `SlotPeriodAndOffset` property.

Data Types: `double`

`NumCandidates` — Number of candidates per aggregation level
`[8 8 4 2 1]` (default) | 1-by-5 integer vector

Number of candidates per aggregation level, specified as a 1-by-5 integer vector. For each aggregation level, you can specify 0, 1, 2, 3, 4, 5, 6, or 8 candidates. The vector element values correspond to the number of candidates for aggregation levels *AL1*, *Al2*, *AL4*, *Al8*, and *AL16*, respectively.

Data Types: `double`

Examples

Generate PDCCH DM-RS Symbols for All Candidates and Aggregation Levels

Configure the carrier with default configuration parameters.

```
carrier = nrCarrierConfig;
```

Configure the CORESET with 6 frequency resources, a duration of 3 OFDM symbols, and a REG bundle size of 3.

```
crst = nrCORESETConfig;
crst.FrequencyResources = ones(1,6);
crst.Duration = 3;
crst.REGBundleSize = 3;
```

Configure the search space set for the PDCCH with the specified number of candidates at each aggregation level.

```
cfgSS = nrSearchSpaceConfig;
cfgSS.NumCandidates = [5 5 3 2 1];
```

Configure the PDCCH with the specified bandwidth part, CORESET, and search space set.

```
pdcch = nrPDCCHConfig;
pdcch.NStartBWP = 6;
pdcch.NSizeBWP = 36;
pdcch.CORESET = crst;
pdcch.SearchSpace = cfgSS;
```

Generate PDCCH DM-RS symbols for all candidates and aggregation levels.

```
[~,allDMRS] = nrPDCCHSpace(carrier,pdcch)
```

```
allDMRS=5×1 cell array
    { 18x5 double}
    { 36x5 double}
    { 72x3 double}
    {144x2 double}
    {288x1 double}
```

Verify that the number of generated candidates for the PDCCH DM-RS symbols at each aggregation level matches the number of candidates specified by the search space set.

```
numCandidates = [...
    size(allDMRS{1},2) ...
    size(allDMRS{2},2) ...
    size(allDMRS{3},2) ...
    size(allDMRS{4},2) ...
    size(allDMRS{5},2)];
isequaln(cfgSS.NumCandidates,numCandidates)
```

```
ans = logical
   1
```

Generate PDCCH DM-RS Indices for All Candidates and Aggregation Levels

Configure a carrier grid of 60 resource blocks (RBs), where the starting RB index relative to the common resource block 0 (CRB 0) is 3.

```
carrier = nrCarrierConfig;
carrier.NStartGrid = 3;
carrier.NSizeGrid = 60;
```

Configure noninterleaved CORESET with 6 frequency resources and a duration of 3 OFDM symbols.

```
crst = nrCORESETConfig;
crst.FrequencyResources = ones(1,6);
crst.Duration = 3;
crst.CCEREGMapping = 'noninterleaved';
```

Configure the search space set for the PDCCH with the specified number of candidates at each aggregation level.

```
cfgSS = nrSearchSpaceConfig;
cfgSS.NumCandidates = [5 5 3 2 1];
```

Configure the PDCCH with the specified bandwidth part, CORESET, and search space set.

```
pdcch = nrPDCCHConfig;
pdcch.NStartBWP = 5;
pdcch.NSizeBWP = 48;
pdcch.CORESET = crst;
pdcch.SearchSpace = cfgSS;
```

Generate PDCCH DM-RS resource element indices for all candidates and aggregation levels using 1-based, subscript indexing form relative to the BWP grid.

```
[~,~,allDMRSInd] = nrPDCCHSpace(carrier,pdcch, ...
    'IndexOrientation','bwp','IndexStyle','subscript')
```

```
allDMRSInd=5×1 cell array
    { 18x3x5 uint32}
    { 36x3x5 uint32}
    { 72x3x3 uint32}
    {144x3x2 uint32}
    {288x3   uint32}
```

Verify that the number of generated candidates for PDCCH DM-RS indices at each aggregation level matches the number of candidates specified by the search space set.

```
numCandidates = [...
    size(allDMRSInd{1},3) ...
    size(allDMRSInd{2},3) ...
    size(allDMRSInd{3},3) ...
    size(allDMRSInd{4},3) ...
    size(allDMRSInd{5},3)];
isequaln(cfgSS.NumCandidates,numCandidates)
```

```
ans = logical
   1
```

Version History

Introduced in R2020a

References

[1] 3GPP TS 38.213. “NR; Physical layer procedures for control.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`nrPDCCH` | `nrPDCCHResources` | `nrPDCCHSpace`

Objects

`nrCORESETConfig` | `nrPDCCHConfig` | `nrDLCarrierConfig` | `nrCarrierConfig`

nrSRSConfig

SRS configuration parameters

Description

The `nrSRSConfig` object sets sounding reference signal (SRS) configuration parameters, as defined in TS 38.211 Section 6.4.1.4 [1].

The default `nrSRSConfig` object specifies a single-port, single-symbol, narrowband configuration without frequency hopping and places the SRS at the end of the slot.

Creation

Syntax

```
srs = nrSRSConfig
srs = nrSRSConfig(Name,Value)
```

Description

`srs = nrSRSConfig` creates an SRS configuration object with default properties.

`srs = nrSRSConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'NumSRSPorts',2,'NumSRSSymbols',4` specifies a two-port SRS transmission of 4 OFDM symbols. Unspecified properties take their default values.

Properties

Configurable SRS Properties

`NumSRSPorts` — Number of SRS antenna ports
1 (default) | 2 | 4

Number of SRS antenna ports, specified as `1`, `2`, or `4`.

Data Types: `double`

`NumSRSSymbols` — Number of OFDM symbols
1 (default) | 2 | 4 | 8 | 12

Number of OFDM symbols allocated to the SRS in a slot, specified as `1`, `2`, `4`, `8`, or `12`. Valid property values depend on the `SRSPositioning` property.

- If the `SRSPositioning` property is set to `false`, specify this property as `1`, `2`, or `4`.
- If the `SRSPositioning` property is set to `true`, specify this property as `1`, `2`, `4`, `8`, or `12`. Use the `SubcarrierOffsetTable` property to obtain valid configurations of this property and the `KTC` property.

Data Types: `double`

`SymbolStart` — 0-based index of first OFDM symbol
`13` (default) | integer from 0 to 13

0-based index of first OFDM symbol in the SRS within a slot, specified as one of these options:

- Integer from 0 to 13 — Use this option for normal cyclic prefix.
- Integer from 0 to 11 — Use this option for extended cyclic prefix.

For the SRS symbols and index generation, set the cyclic prefix of the carrier by using the `CyclicPrefix` property of the `nrCarrierConfig` object.

Data Types: `double`

`KTC` — Transmission comb number
`2` (default) | `4` | `8`

Transmission comb number, in subcarriers, specified as `2`, `4`, or `8`. The object allocates the SRS sequence every `KTC` number of subcarriers. Valid property values depend on the `SRSPositioning` property.

- If the `SRSPositioning` property is set to `false`, specify this property as `2` or `4`.
- If the `SRSPositioning` property is set to `true`, specify this property as `2`, `4`, or `8`. Use the `SubcarrierOffsetTable` property to obtain valid configurations of this property and the `NumSRSSymbols` property.

Data Types: `double`

`KBarTC` — Transmission comb offset
`0` (default) | integer from 0 to (`KTC` – 1)

Transmission comb offset, in subcarriers, specified as an integer from 0 to (`KTC` – 1). This property specifies a frequency shift within the comb.

Data Types: `double`

`CyclicShift` — Cyclic shift offset
`0` (default) | integer from 0 to 11

Cyclic shift offset, specified as an integer from 0 to 11. This property determines the cyclic shift applied to the SRS sequence for each antenna port. This property corresponds to parameter n_{SRS}^{cs} in TS 38.211 Section 6.4.1.4.2.

Set the cyclic shift offset in relation to the transmission comb property, `KTC`:

- If you set `KTC` to `2`, set `CyclicShift` to an integer from 0 to 7.
- If you set `KTC` to `4`, set `CyclicShift` to an integer from 0 to 11.
- If you set `KTC` to `8`, set `CyclicShift` to an integer from 0 to 5.

For multiport SRS transmissions, consecutive cyclic shift numbers are used for each port, modulo 6, 8, or 12, depending on the `KTC` property.

Data Types: `double`

FrequencyStart — Frequency-domain offset
0 (default) | integer from 0 to 271

Frequency-domain offset of the SRS, in terms of a physical resource block (PRB) index relative to the carrier, specified as an integer from 0 to 271. FrequencyStart is analogous to parameter n_{shift} from TS 38.211 Section 6.4.1.4.3.

This property, the additional circular frequency-domain offset property NRRC, and the bandwidth configuration parameters in TS 38.211 Table 6.4.1.4.3-1 determine the actual frequency-domain location of the SRS. For more information, see "NR SRS Configuration".

Data Types: double

NRRC — Additional circular frequency-domain offset
0 (default) | integer from 0 to 67

Additional circular frequency-domain offset of the SRS, as a multiple of 4 PRBs, specified as an integer from 0 to 67.

This property, the frequency domain offset property FrequencyStart, and the bandwidth configuration parameters in TS 38.211 Table 6.4.1.4.3-1 determine the actual frequency-domain location of the SRS. For more information, see "NR SRS Configuration".

Data Types: double

CSRS — Row index of bandwidth configuration table
0 (default) | integer from 0 to 63

Row index of the bandwidth configuration table from TS 38.211 Table 6.4.1.4.3-1, specified as an integer from 0 to 63. Use this property with the BSRS property to control the bandwidth allocated to the SRS and the frequency hopping pattern. Increasing the CSRS value increases the SRS bandwidth. The default value of 0 results in a bandwidth of 4 PRBs.

Data Types: double

BSRS — Column index of bandwidth configuration table
0 (default) | integer from 0 to 3

Column index of the bandwidth configuration table from TS 38.211 Table 6.4.1.4.3-1, specified as an integer from 0 to 3. Use this property with the CSRS property to control the bandwidth allocated to the SRS and the frequency hopping pattern. Increasing the BSRS value decreases the SRS bandwidth.

Data Types: double

BHop — Frequency hopping index
0 (default) | integer from 0 to 3

Frequency hopping index, specified as an integer from 0 to 3. Setting this property to a value greater than or equal to the column index of the bandwidth configuration table property, BSRS, disables frequency hopping. Increasing the BHop value decreases the hopping bandwidth.

Data Types: double

Repetition — Repetition factor of OFDM symbols
1 (default) | 2 | 4 | 8 | 12

Repetition factor of OFDM symbols, specified as 1, 2, 4, 8, or 12.

- When frequency hopping is enabled, `Repetition` specifies the number of consecutive OFDM symbols in a slot occupied by the SRS in the same frequency resource. Set `Repetition` such that `Repetition` ≤ `NumSRSSymbols`.
- When frequency hopping is disabled, this property is ignored.

Data Types: `double`

`SRSPeriod` — Slot periodicity and offset
`'on'` (default) | `'off'` | [T_{SRS} T_{offset}]

Slot periodicity and offset, specified as one of these values:

- `'on'` — The SRS is present in all slots.
- `'off'` — The SRS is absent in all slots.
- [T_{SRS} T_{offset}] — The presence of the SRS in a given slot depends on the specified slot periodicity, T_{SRS}, and the offset, T_{offset}, based on TS 38.211 Section 6.4.1.4.4. Specify T_{SRS} as `1`, `2`, `4`, `5`, `8`, `10`, `16`, `20`, `32`, `40`, `64`, `80`, `160`, `320`, `640`, `1280`, or `2560`. Specify T_{offset} as a nonnegative integer such that $T_{offset} < T_{SRS}$.

Data Types: `char` | `string` | `double`

`ResourceType` — Time-domain behavior
`'periodic'` (default) | `'semi-persistent'` | `'aperiodic'`

Time-domain behavior of the SRS, specified as `'periodic'`, `'semi-persistent'`, or `'aperiodic'`. Downlink control information (DCI) triggers aperiodic SRS transmissions. When the resource type is aperiodic, the `SRSPeriod` property determines the periodicity and offset of the DCI-triggering signal. An aperiodic resource type also disables interslot frequency hopping.

Data Types: `char` | `string`

`GroupSeqHopping` — Type of SRS symbol hopping
`'neither'` (default) | `'groupHopping'` | `'sequenceHopping'`

Type of SRS symbol hopping, specified as `'neither'`, `'groupHopping'`, or `'sequenceHopping'`. When either group or sequence hopping is enabled, the group or sequence hopping numbers per OFDM symbol in the SRS transmission are based on a pseudo-random binary sequence (PRBS). Set the scrambling identity for the PRBS by using the `NSRSID` property.

Data Types: `char` | `string`

`NSRSID` — SRS scrambling identity
`0` (default) | integer from 0 to 65,535

SRS scrambling identity, specified as an integer from 0 to 65,535.

- When you set the `GroupSeqHopping` property to `'neither'`, this property determines the group number.
- When you set the `GroupSeqHopping` property to `'groupHopping'` or `'sequenceHopping'`, this property initializes the PRBS.

Data Types: `double`

`SRSPositioning` — SRS for user positioning
`0` or `false` (default) | `1` or `true`

SRS for user positioning, as defined in Release 16 of TS 38.211 Section 6.4.1.4, specified as one of these values.

- `0` (`false`) — Disable SRS for user positioning. This option corresponds to the higher-layer parameter *SRS-Resource*.
- `1` (`true`) — Enable SRS for user positioning. This option corresponds to the higher-layer parameter *SRS-PosResource-r16*.

This property affects the valid range of the `NumSRSSymbols` and `KTC` properties.

Data Types: `logical` | `double`

Nonconfigurable SRS Properties

The object automatically sets these properties based on configurable SRS property values by using the configuration tables from TS 38.211 Section 6.4.1.4.

`NRB` — Number of RBs allocated for SRS transmission
4 (default) | positive integer

This property is read-only.

Number of RBs allocated for SRS transmission, specified as a positive integer. When frequency hopping is enabled, this property denotes the hopping bandwidth or number of RBs over which the SRS signal hops across multiple time slots.

Data Types: `double`

`NRBPerTransmission` — Number of RBs allocated per SRS OFDM symbol
4 (default) | positive integer

This property is read-only.

Number of RBs allocated per SRS OFDM symbol, specified as a positive integer. When frequency hopping is enabled, this property specifies the allocated bandwidth at each SRS OFDM symbol. When frequency hopping is disabled, this property is equal to the `NRB` property.

Data Types: `double`

SRS Lookup Tables

`BandwidthConfigurationTable` — SRS bandwidth configuration table
constant 64-by-9 table (default)

This property is read-only.

SRS bandwidth configuration table corresponding to TS 38.211 Table 6.4.1.4.3-1, specified as a constant 64-by-9 table.

`SubcarrierOffsetTable` — Subcarrier offset configuration table
constant 3-by-6 table (default)

This property is read-only.

Subcarrier offset configuration table corresponding to TS 38.211 Table 6.4.1.4.3-2, specified as a constant 3-by-6 table. Use this property to determine the SRS subcarrier offset for each OFDM symbol and to obtain valid combinations of the number of OFDM symbols and the transmission comb number.

Examples

Generate and Map SRS Symbols to Carrier Grid

Configure the SRS and the carrier with default configuration parameters.

```
carrier = nrCarrierConfig;
srs = nrSRSConfig;
```

Generate SRS symbols and indices using the specified carrier and SRS configuration parameters.

```
srsSym = nrSRS(carrier,srs);
srsInd = nrSRSIndices(carrier,srs);
```

Create a carrier grid corresponding to the number of subcarriers, OFDM symbols, and number of antenna ports specified in the configuration objects.

```
K = carrier.NSizeGrid*12;       % Number of subcarriers
L = carrier.SymbolsPerSlot;     % Number of OFDM symbols per slot
P = srs.NumSRSPorts;  % Number of antenna ports
gridSize = [K L P];
```

Initialize the carrier grid for one slot with all zeros.

```
slotGrid = complex(zeros(gridSize));
```

Map the SRS symbols to the carrier grid using the indices.

```
slotGrid(srsInd) = srsSym;
```

Generate SRS Symbols for Two-Port Transmission

Configure the carrier with default configuration parameters.

```
carrier = nrCarrierConfig;
```

Configure a two-port SRS transmission of 4 OFDM symbols.

```
srs = nrSRSConfig;
srs.NumSRSPorts = 2;
srs.NumSRSSymbols = 4;
```

The SRS must be located in the last six symbols of the slot. Set the time-domain starting position of the SRS to 8 and the bandwidth configuration index to 5.

```
srs.SymbolStart = 8;
srs.CSRS = 5;
```

Generate SRS symbols for the specified carrier and SRS configuration parameters.

```
[sym,info] = nrSRS(carrier,srs);
```

Verify that the symbols vector contains two columns corresponding to the two-port transmission.

```
size(sym)
```

```
ans = 1×2

   480     2
```

Verify the number of SRS symbols per port.

```
isequal(info.SeqLength*srs.NumSRSSymbols,size(sym,1))
```

```
ans = logical
   1
```

Version History

Introduced in R2020a

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrSRSIndices` | `nrSRS`

Objects
`nrCarrierConfig`

Topics
“NR SRS Configuration”

nrULCarrierConfig

5G uplink waveform configuration parameters

Description

The `nrULCarrierConfig` object sets the parameters of a single-component-carrier 5G uplink waveform. Use this object to configure 5G uplink waveform generation when calling the `nrWaveformGenerator` function.

This object defines these aspects of the uplink waveform:

- Frequency range
- Channel bandwidth
- Cell identity
- Waveform duration
- Subcarrier spacing (SCS) carriers
- Bandwidth parts (BWPs)
- Physical uplink shared channel (PUSCH), PUSCH demodulation reference signal (DM-RS), and PUSCH phase-tracking reference signal (PT-RS)
- Physical uplink control channel (PUCCH) and associated DM-RS
- Sounding reference signal (SRS)

Creation

Syntax

```
cfgUL = nrULCarrierConfig
cfgUL = nrULCarrierConfig(Name,Value)
```

Description

`cfgUL = nrULCarrierConfig` creates a default single-component-carrier 5G uplink waveform configuration object.

`cfgUL = nrULCarrierConfig(Name,Value)` sets properties on page 3-170 using one or more name-value arguments. Enclose each property name in quotes. For example, `'FrequencyRange','FR2'` specifies an uplink waveform for frequency range 2 (FR2).

Properties

`Label` — Name of uplink carrier configuration
`'Uplink carrier 1'` (default) | character array | string scalar

Name of the uplink carrier configuration, specified as a character array or string scalar. Use this property to set a description to the uplink carrier configuration.

Data Types: `char` | `string`

`FrequencyRange` — Frequency range
`'FR1'` (default) | `'FR2'`

Frequency range, specified as one of these values.

- `'FR1'` for frequency range 1 (FR1)
- `'FR2'` for frequency range 2 (FR2)

Data Types: `char` | `string`

`ChannelBandwidth` — Channel bandwidth
`50` (default) | `5` | `10` | `15` | `20` | `25` | `30` | `40` | `50` | `60` | `70` | `80` | `90` | `100` | `200` | `400`

Channel bandwidth, in MHz, specified as one of these values.

- `5`, `10`, `15`, `20`, `25`, `30`, `40`, `50`, `60`, `70`, `80`, `90`, or `100` for FR1
- `50`, `100`, `200`, or `400` for FR2

Set the frequency range with the `FrequencyRange` property.

Data Types: `double`

`NCellID` — Physical layer cell identity
`1` (default) | integer from 0 to 1007

Physical layer cell identity, specified as an integer from 0 to 1007.

Data Types: `double`

`NumSubframes` — Waveform duration in subframes
`10` (default) | positive integer

Waveform duration in subframes (multiples of 1 ms), specified as a positive integer. The default value of 10 subframes corresponds to one frame.

Data Types: `double`

`WindowingPercent` — Windowing percentage relative to FFT length
`0` (default) | real scalar from 0 to 50 | five-element row vector | `[]`

Windowing percentage relative to the fast Fourier transform (FFT) length, specified as one of these values.

- Real scalar from 0 to 50 — The object sets the same windowing percentage for all combinations of SCS and cyclic prefix.
- Five-element row vector of the form [*w1 w2 w3 w4 w5*], where each element is a real scalar from 0 to 50 — The object sets individual windowing percentage for the different SCS and cyclic prefix combinations.

  - *w1* specifies the windowing percentage for 15 kHz SCS.
  - *w2* specifies the windowing percentage for 30 kHz SCS.
  - *w3* specifies the windowing percentage for 60 kHz SCS and normal cyclic prefix.

- *w4* specifies the windowing percentage for 60 kHz SCS and extended cyclic prefix.
- *w5* specifies the windowing percentage for 120 kHz SCS.

- [] — The object automatically selects the windowing percentage of each SCS carrier (specified by SCSCarriers) based on the SampleRate property and these additional properties.
  - The NSizeGrid and SubcarrierSpacing properties of the actual SCS carrier.
  - The CyclicPrefix property of the actual BWP specified by the BandwidthParts property.

  For more information, see the 'Windowing' name-value argument description of the nrOFDMModulate function.

This property configures the number of time-domain samples, as a percentage of the FFT length, over which windowing and overlapping of the OFDM symbols take place.

Data Types: double

SampleRate — Sample rate of OFDM-modulated waveform
[] (default) | positive integer scalar

Sample rate of the OFDM-modulated waveform, specified as [] or a positive integer scalar. When you set this value to [], the object sets the sample rate to the minimum value that accommodates all carriers in the waveform without aliasing.

CarrierFrequency — Carrier frequency in Hz
0 (default) | real number

Carrier frequency in Hz, specified as a real number. This property corresponds to f_0, defined in TS 38.211 Section 5.4, and is used for symbol phase compensation before OFDM modulation.

Data Types: double

SCSCarriers — One or more SCS carrier configurations
{nrSCSCarrierConfig} (default) | cell array of nrSCSCarrierConfig objects

One or more SCS carrier configurations, specified as a cell array of nrSCSCarrierConfig objects. Because this property configures the subcarrier spacing and grid size of each numerology, each nrSCSCarrierConfig object in the cell array must have a unique SubcarrierSpacing property value.

BandwidthParts — One or more BWP configurations
{nrWavegenBWPConfig} (default) | cell array of nrWavegenBWPConfig objects

One or more BWP configurations, specified as a cell array of nrWavegenBWPConfig objects. The SubcarrierSpacing properties of these BWP objects must be one of the values defined by the SubcarrierSpacing properties of the carriers specified by the SCSCarriers property.

PUSCH — One or more PUSCH configurations
{nrWavegenPUSCHConfig} (default) | cell array of nrWavegenPUSCHConfig objects

One or more PUSCH configurations, specified as a cell array of nrWavegenPUSCHConfig objects. Use this property to configure different PUSCH and associated DM-RS and PT-RS.

PUCCH — One or more PUCCH configurations
{nrWavegenPUCCH0Config('Enable',0)} (default) | cell array of nrWavegenPUCCH0Config , nrWavegenPUCCH1Config, nrWavegenPUCCH2Config, nrWavegenPUCCH3Config, and nrWavegenPUCCH4Config objects

One or more PUCCH configurations, specified as a cell array of any combination of nrWavegenPUCCH0Config, nrWavegenPUCCH1Config, nrWavegenPUCCH2Config, nrWavegenPUCCH3Config, and nrWavegenPUCCH4Config objects. This property configures different PUCCH and associated DM-RS. By default, the PUCCH is disabled in the uplink configuration.

SRS — One or more SRS configurations
{nrWavegenSRSConfig('Enable',0)} (default) | cell array of nrWavegenSRSConfig objects

One or more SRS configurations, specified as a cell array of nrWavegenSRSConfig objects. Use this property to configure different SRS. By default, the SRS is disabled in the uplink configuration.

Examples

Configure and Generate Single-User 5G Uplink Waveform

Create an SCS carrier configuration object with the default SCS of 15 kHz and 100 resource blocks.

```
carrier = nrSCSCarrierConfig('NSizeGrid',100);
```

Create a customized BWP configuration object for the SCS carrier.

```
bwp = nrWavegenBWPConfig('NStartBWP',carrier.NStartGrid+10);
```

Create a single-user 5G uplink waveform configuration object, specifying the previously defined configurations. In the uplink configuration object, by default, the PUSCH is enabled, while the PUCCH and the SRS are disabled.

```
cfgUL = nrULCarrierConfig( ...
    'FrequencyRange','FR1', ...
    'ChannelBandwidth',40, ...
    'NumSubframes',20, ...
    'SCSCarriers',{carrier}, ...
    'BandwidthParts',{bwp});
```

Generate a 5G uplink waveform using the specified configuration.

```
waveform = nrWaveformGenerator(cfgUL);
```

Configure and Generate Multiuser 5G Uplink Waveform

Create two SCS carrier configuration objects with mixed numerologies and custom numbers of resource blocks.

```
carriers = {
    nrSCSCarrierConfig('SubcarrierSpacing',15,'NStartGrid',10,'NSizeGrid',100), ...
    nrSCSCarrierConfig('SubcarrierSpacing',30,'NStartGrid',0,'NSizeGrid',70)};
```

Create two custom BWP configuration objects, one for each of the carriers.

```
bwp = {
    nrWavegenBWPConfig('BandwidthPartID',0,'SubcarrierSpacing',15,'NStartBWP',30,'NSizeBWP',80),
    nrWavegenBWPConfig('BandwidthPartID',1,'SubcarrierSpacing',30,'NStartBWP',0,'NSizeBWP',60)};
```

Create two PUSCH configuration objects, one for each of the carriers, with mixed modulation schemes.

```
pusch = {
    nrWavegenPUSCHConfig('BandwidthPartID',0,'Modulation','16QAM','SlotAllocation',0:2:9,'PRBSet
    nrWavegenPUSCHConfig('BandwidthPartID',1,'Modulation','QPSK','RNTI',2,'NID',2,'PRBSet',50:59
```

Create a single PUCCH configuration object, only for the second carrier. By default, the PUCCH is enabled in this configuration.

```
pucch = {nrWavegenPUCCH0Config('BandwidthPartID',1,'SlotAllocation',0:9,'PRBSet',2,'DataSourceUC
```

Create two SRS configuration objects, one for each of the carriers. By default, the SRS is enabled in both configurations.

```
srs = {
    nrWavegenSRSConfig('BandwidthPartID',0,'SlotAllocation',1:2:9,'NumSRSPorts',2), ...
    nrWavegenSRSConfig('BandwidthPartID',1,'FrequencyStart',4)};
```

Create a multiuser 5G uplink waveform configuration object, specifying the previously defined configurations.

```
cfgUL = nrULCarrierConfig( ...
    'FrequencyRange','FR1', ...
    'ChannelBandwidth',40, ...
    'NumSubframes',20, ...
    'SCSCarriers',carriers, ...
    'BandwidthParts',bwp, ...
    'PUSCH',pusch, ...
    'PUCCH',pucch, ...
    'SRS',srs);
```

Generate a 5G uplink waveform using the specified configuration.

```
waveform = nrWaveformGenerator(cfgUL);
```

Version History

Introduced in R2021a

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`nrWaveformGenerator`

Objects
nrWavegenBWPConfig | nrSCSCarrierConfig | nrWavegenSRSConfig | nrDLCarrierConfig | nrWavegenPUCCH0Config | nrWavegenPUCCH1Config | nrWavegenPUCCH2Config | nrWavegenPUCCH3Config | nrWavegenPUCCH4Config | nrWavegenPUSCHConfig | nrPUSCHDMRSConfig | nrPUSCHPTRSConfig | nrWavegenSRSConfig

nrWavegenBWPConfig

BWP configuration parameters for 5G waveform generation

Description

The nrWavegenBWPConfig object sets bandwidth part (BWP) configuration parameters in a specific subcarrier spacing (SCS) carrier. Use this object to set the BandwidthParts property of the nrDLCarrierConfig object or the BandwidthParts property of the nrULCarrierConfig object when configuring 5G waveform generation. Specify the SCS carrier with the same numerology by using the SCSCarriers property of the same nrDLCarrierConfig or nrULCarrierConfig object.

This object defines the SCS of the carrier containing the BWP, the size of the BWP, the offset from the common resource block 0 (CRB 0), and the cyclic prefix. For a SCS of 60 kHz, you can specify either normal or extended cyclic prefix.

Creation

Syntax

```
bwp = nrWavegenBWPConfig
bwp = nrWavegenBWPConfig(Name,Value)
```

Description

bwp = nrWavegenBWPConfig creates a default BWP configuration object for 5G waveform generation.

bwp = nrWavegenBWPConfig(Name,Value) specifies properties on page 3-176 using one or more name-value arguments. Enclose each property name in quotes. For example, 'SubcarrierSpacing',30 specifies a SCS of 30 kHz for the carrier containing the BWP.

Properties

BandwidthPartID — ID of BWP configuration
1 (default) | nonnegative integer

ID of the BWP configuration, specified as a nonnegative integer.

Data Types: double

Label — Name of BWP configuration
'BWP1' (default) | character array | string scalar

Name of the BWP configuration, specified as a character array or string scalar. Use this property to set a description to the BWP configuration.

Data Types: char | string

SubcarrierSpacing — Subcarrier spacing in kHz
15 (default) | 30 | 60 | 120

Subcarrier spacing in kHz, for all channels and reference signals of the carrier, specified as 15, 30, 60, or 120.

Data Types: double

CyclicPrefix — Cyclic prefix length
'normal' (default) | 'extended'

Cyclic prefix length, specified as one of these options.

- 'normal' — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- 'extended' — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies for only 60 kHz subcarrier spacing.

Data Types: char | string

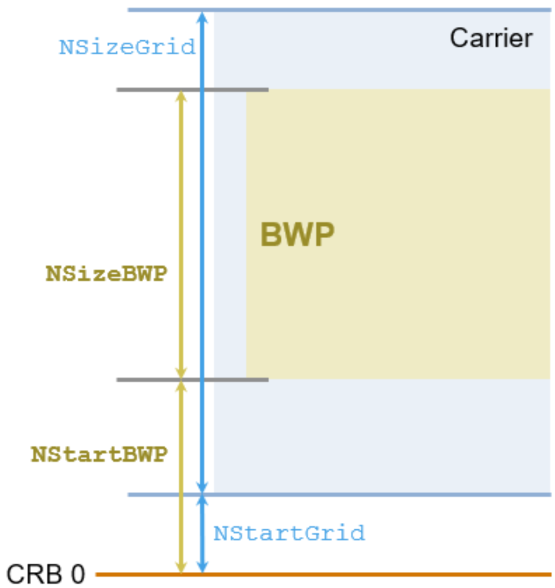
NSizeBWP — Number of RBs in BWP resource grid
52 (default) | integer from 1 to 275

Number of resource blocks (RBs) in the BWP resource grid, specified as an integer from 1 to 275. This property must be less than or equal to the size of the SCS carrier with the same SCS, specified by the SCSCarriers property of the nrDLCarrierConfig or nrULCarrierConfig objects.

Data Types: double

NStartBWP — Start of BWP resource grid relative to CRB 0
0 (default) | nonnegative integer

Start of the BWP resource grid relative to CRB 0, specified as a nonnegative integer. Set this property relative to the SCS carrier such that the property value is in this range: NStartGrid ≤ NStartBWP ≤ (NStartGrid + NSizeGrid – NSizeBWP). NStartGrid and NSizeGrid are properties of the SCS carrier with the same SCS, specified by the SCSCarriers property of the nrDLCarrierConfig or nrULCarrierConfig objects. This figure shows where in the carrier the BWP is located in terms of this property and the NSizeBWP property.



Data Types: `double`

Examples

Configure Bandwidth Part for Default SCS

Create a BWP configuration object for the default SCS carrier of 15 kHz. Specify the number of RBs in the BWP resource grid and the start of the BWP resource grid relative to the CRB 0.

```
bwp = nrWavegenBWPConfig;
bwp.NSizeBWP = 50;
bwp.NStartBWP = 12;
```

Create a downlink carrier configuration object, specifying the previously defined BWP configuration.

```
cfgDL = nrDLCarrierConfig('BandwidthParts',{bwp});
```

Configure Two Bandwidth Parts for Two SCS Carriers

Create a default SCS carrier configuration object, which configures a 10 MHz carrier with 15 kHz SCS.

```
scs1 = nrSCSCarrierConfig;
```

Create an SCS carrier configuration object, which configures a 100 MHz carrier with 30 kHz SCS.

```
scs2 = nrSCSCarrierConfig('SubcarrierSpacing',30,'NSizeGrid',273);
```

Create two BWP configurations, one for each of the SCS carriers.

```
bwp1 = nrWavegenBWPConfig;
bwp2 = nrWavegenBWPConfig('SubcarrierSpacing',scs2.SubcarrierSpacing, ...
    'NSizeBWP',12,'NStartBWP',30);
```

Create a downlink carrier configuration object, specifying the previously defined BWP and corresponding SCS carrier configurations.

```
cfgDL = nrDLCarrierConfig( ...
    'SCSCarriers',{scs1,scs2}, ...
    'BandwidthParts',{bwp1,bwp2});
```

Version History

Introduced in R2020b

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`nrWaveformGenerator`

Objects

`nrDLCarrierConfig` | `nrSCSCarrierConfig` | `nrULCarrierConfig`

nrWavegenCSIRSConfig

CSI-RS configuration parameters for 5G waveform generation

Description

The `nrWavegenCSIRSConfig` object sets channel state information reference signal (CSI-RS) configuration parameters for one or more zero-power (ZP) or non-zero-power (NZP) CSI-RS resources, as defined in TS 38.211 Section 7.4.1.5 [1]. Use this object to set the `CSIRS` property of the `nrDLCarrierConfig` object when configuring 5G downlink waveform generation.

By default, the object defines an NZP-CSI-RS resource configured for two antenna ports with a CDM type of FD-CDM2 and density 1, corresponding to CSI-RS resource defined in row 3 of TS 38.211 Table 7.4.1.5.3-1.

Creation

Syntax

```
csirs = nrWavegenCSIRSConfig
csirs = nrWavegenCSIRSConfig(Name,Value)
```

Description

`csirs = nrWavegenCSIRSConfig` creates a default CSI-RS configuration object for 5G waveform generation.

`csirs = nrWavegenCSIRSConfig(Name,Value)` sets properties on page 3-180 using one or more name-value arguments. Enclose each property name in quotes. For example, `'CSIRSType', {'zp','nzp','zp'}` specifies three CSI-RS resources.

Properties

`Enable` — Enable CSI-RS
`1` (default) | `0`

Enable the CSI-RS in 5G waveform generation, specified as one of these values.

- `1` — Enable the CSI-RS.
- `0` — Disable the CSI-RS.

Data Types: `double` | `logical`

`Label` — Name of CSI-RS configuration
`'CSIRS1'` (default) | character array | string scalar

Name of the CSI-RS configuration, specified as a character array or string scalar. Use this property to set a description to the CSI-RS configuration.

Data Types: char | string

Power — Power scaling of CSI-RS in dB
0 (default) | real scalar

Power scaling of CSI-RS in dB, specified as a real scalar. Use this property to scale the power of the CSI-RS in the generated 5G waveform.

Data Types: double

BandwidthPartID — ID of BWP
1 (default) | nonnegative integer

ID of the bandwidth part (BWP) containing the configured CSI-RS, specified as a nonnegative integer. Use this property to associate this CSI-RS configuration with one of the BWP configurations specified by the BandwidthParts property of the nrDLCarrierConfig object.

Data Types: double

CSIRSType — Type of one or more CSI-RS resource configurations
'nzp' (default) | 'zp' | cell array | string scalar | string array

Type of one or more CSI-RS resource configurations, specified as one of these options.

- 'nzp' — Use this option to specify a single NZP-CSI-RS resource.
- 'zp' — Use this option to specify a single ZP-CSI-RS resource.
- Cell array with elements 'nzp' or 'zp' — Use this option to specify multiple CSI-RS resources.

Alternatively, you can specify this property by using "nzp" and "zp" as string scalars or as elements of a string array.

The number of CSI-RS resource configurations is equal to the number of values provided for this property.

Data Types: cell | string | char

CSIRSPeriod — Slot periodicity and offset of CSI-RS resource
'on' (default) | 'off' | vector of integers | cell array | string scalar | string array

Slot periodicity and offset of the CSI-RS resource, specified as one of these options.

For Single CSI-RS Resource

- 'on' — Use this option to indicate that the resource is present in all slots.
- 'off' — Use this option to indicate that the resource is absent in all slots.
- Vector of integers of the form [*Tcsi-rs Toffset*] — Use this option to specify slot periodicity *Tcsi-rs* and offset *Toffset* for scheduling the CSI-RS resource in specific slots.

  Tcsi-rs is 4, 5, 8, 10, 16, 20, 32, 40, 64, 80, 160, 320, or 640. For a particular value of *Tcsi-rs*, the value of *Toffset* is in the range from 0 to *Tcsi-rs*–1.

For Multiple CSI-RS Resources

- Cell array with elements 'on', 'off', or [*Tcsi-rs Toffset*] — The number of elements in the cell array must be one or equal the number of CSI-RS resources specified by the CSIRSType property. When the cell array contains only one element, the specified value applies to all CSI-RS resources.

Alternatively, you can specify this property by using "on" and "off" as string scalars or as elements of a string array.

This property is the higher-layer parameter *CSI-ResourcePeriodicityAndOffset* or *slotConfig* defined in the *CSI-RS-CellMobility* IE.

Data Types: cell | string | char | double

RowNumber — Row number of CSI-RS resource
3 (default) | integer from 1 to 18 | vector of integers

Row number of CSI-RS resource, as defined in TS 38.211 Table 7.4.1.5.3-1, specified as one of these options.

For Single CSI-RS Resource

- Integer from 1 to 18

For Multiple CSI-RS Resources

- Vector of integers in the range from 1 to 18 — The number of vector elements must equal the number of CSI-RS resources specified by the CSIRSType property.

Data Types: double

Density — Frequency density of CSI-RS resource
'one' (default) | 'three' | 'dot5even' | 'dot5odd' | cell array | string scalar | string array

Frequency density of the CSI-RS resource, as defined in TS 38.211 Table 7.4.1.5.3-1, specified as one of these options.

For Single CSI-RS Resource

- 'one' — This option corresponds to $\rho = 1$ from the specified table.
- 'three' — This option corresponds to $\rho = 3$ from the specified table.
- 'dot5even' — This option corresponds to $\rho = 0.5$ from the specified table with even resource block (RB) allocation regarding the common resource block 0 (CRB 0).
- 'dot5odd' — This option corresponds to $\rho = 0.5$ from the specified table with odd RB allocation regarding CRB 0.

For Multiple CSI-RS Resources

- Cell array of the character vectors 'one', 'three', 'dot5even', or 'dot5odd' — The number of elements in the cell array must equal the number of CSI-RS resources specified by the CSIRSType property.

Alternatively, you can specify this property by using "one", "three", "dot5even", and "dot5odd" as string scalars or as elements of a string array.

The value of ρ is the higher-layer parameter *density* defined in the *CSI-RS-ResourceMapping* IE or the *CSI-RS-CellMobility* IE.

Data Types: cell | string | char

SymbolLocations — Time-domain locations of CSI-RS resource
0 (default) | integer from 0 to 13 | vector of integers | cell array

Time-domain locations of the CSI-RS resource (l_0 and l_1 values in the TS 38.211 Table 7.4.1.5.3-1), specified as one of these options.

For Single CSI-RS Resource

- Integer from 0 to 13 — This option corresponds to the l_0 value in the specified table.
- Vector of integers of the form [l_0 l_1] or [l_0; l_1], where l_0 and l_1 are the corresponding l_0 and l_1 values in the specified table — The l_1 values are required only in table rows 13, 14, 16, and 17. l_0 is an integer from 0 to 13, and l_1 is an integer from 2 to 12.

For Multiple CSI-RS Resources

- Cell array of l_0 values or vectors of the form [l_0 l_1] or [l_0; l_1] — The number of elements in the cell array must equal the number of CSI-RS resources specified by the `CSIRSType` property.

The values of l_0 and l_1 are the higher-layer parameters *firstOFDMSymbolInTimeDomain* and *firstOFDMSymbolInTimeDomain2*, respectively, in the *CSI-RS-ResourceMapping* IE or *CSI-RS-Resource-Mobility* IE.

Data Types: `double`

`SubcarrierLocations` — Frequency-domain locations of CSI-RS resource

`0` (default) | numeric vector | cell array

Frequency-domain locations of the CSI-RS resource (k_i values in the TS 38.211 Table 7.4.1.5.3-1), specified as one of these options.

For Single CSI-RS Resource

- Numeric vector with elements 1, 2, 3, 4, or 6 — The vector elements correspond to the possible lengths of subcarrier locations.

For Multiple CSI-RS Resources

- Cell array of numeric vectors with elements 1, 2, 3, 4, or 6 — The number of elements in the cell array must equal the number of CSI-RS resources specified by the `CSIRSType` property.

Data Types: `double`

`NumRB` — CSI-RS resource bandwidth

`52` (default) | integer from 1 to 275 | vector of integers

CSI-RS resource bandwidth, in terms of the number of allocated RBs, specified as one of these options.

For Single CSI-RS Resource

- Integer from 1 to 275

For Multiple CSI-RS Resources

- Vector of integers in the range from 1 to 275 — The number of vector elements must equal to one or the number of CSI-RS resources specified by the `CSIRSType` property. When the vector contains only one element, the specified value applies to all CSI-RS resources.

This property is the higher-layer parameter *nrOfRBs* in *FrequencyOccupation* IE or *nrOfPRBs* in *CSI-RS-ResourceConfigMobility* IE.

Data Types: `double`

`RBOffset` — Starting RB index of CSI-RS resource allocation
`0` (default) | integer from 0 to 274 | vector of integers

Starting RB index of the CSI-RS resource allocation, relative to the carrier resource grid, specified as one of these options.

For Single CSI-RS Resource

- Integer from 0 to 274

For Multiple CSI-RS Resources

- Vector of integers in the range from 0 to 274 — The number of vector elements must be equal to one or the number of CSI-RS resources specified by the `CSIRSType` property. When the vector contains only one element, the specified value applies to all CSI-RS resources.

Note Specify this property relative to the carrier start, which contains the BWP specified by the `BandwidthPartID` property.

Data Types: `double`

`NID` — Scrambling identity
`0` (default) | integer from 0 to 1023 | vector of integers

Scrambling identity, specified as one of these options.

For Single CSI-RS Resource

- Integer from 0 to 1023

For Multiple CSI-RS Resources

- Vector of integers in the range from 0 to 1023 — The number of vector elements must be one or equal the number of CSI-RS resources specified by the `CSIRSType` property. When the vector contains only one element, the specified value applies to all CSI-RS resources.

This property is the higher-layer parameter *scramblingID* in *NZP-CSI-RS-Resource* IE or *sequenceGenerationConfig* in *CSI-RS-ResourceConfigMobility* IE.

When the `CSIRSType` property defines only ZP resources, this property is hidden.

Data Types: `double`

Examples

Configure CSI-RS for 5G Downlink Waveform Generation

Create two SCS carrier configuration objects with mixed numerologies and a custom number of resource blocks.

```
carriers = {
    nrSCSCarrierConfig('SubcarrierSpacing',15,'NStartGrid',10,'NSizeGrid',100), ...
```

```
    nrSCSCarrierConfig('SubcarrierSpacing',30,'NStartGrid',0,'NSizeGrid',70)};
```

Create two BWP configuration objects, one for each of the SCS carriers.

```
bwp = {
    nrWavegenBWPConfig('BandwidthPartID',1,'SubcarrierSpacing',15,'NStartBWP',10,'NSizeBWP',80),
    nrWavegenBWPConfig('BandwidthPartID',2,'SubcarrierSpacing',30,'NStartBWP',0,'NSizeBWP',60)};
```

Create two CSI-RS configuration objects, one for each of the BWP.

```
 csirs = {
     nrWavegenCSIRSConfig('BandwidthPartID',1,'RowNumber',2,'RBOffset',10), ...
     nrWavegenCSIRSConfig('BandwidthPartID',2,'Density','three','RowNumber',4)};
```

Create a downlink carrier configuration object, specifying the previously defined configurations.

```
cfgDL = nrDLCarrierConfig( ...
    'SCSCarriers',carriers, ...
    'BandwidthParts',bwp, ...
    'CSIRS',csirs);
```

Version History

Introduced in R2020b

`nrWavegenCSIRSConfig` object enables CSI-RS by default

Behavior changed in R2022a

The `Enable` property now defaults to `1` instead of `0`. As a result, the object now enables CSI-RS by default. To preserve the behavior of the `nrDLCarrierConfig` object from previous releases, in which CSI-RS is disabled by default, the `CSRSI` property of the `nrDLCarrierConfig` object now defaults to `{nrWavegenCSIRSConfig('Enable',0)}` instead of `{nrWavegenCSIRSConfig}`.

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`nrWaveformGenerator`

Objects

`nrDLCarrierConfig` | `nrWavegenBWPConfig`

nrWavegenPDCCHConfig

PDCCH configuration parameters for 5G waveform generation

Description

The `nrWavegenPDCCHConfig` object sets physical downlink control channel (PDCCH) configuration parameters, as defined in TS 38.211 Section 7.3.2 [1] and TS 38.213 Section 10 [2]. Use this object to set the `PDCCH` property of the `nrDLCarrierConfig` object when configuring 5G downlink waveform generation.

Creation

Syntax

```
pdcch = nrWavegenPDCCHConfig
pdcch = nrWavegenPDCCHConfig(Name,Value)
```

Description

`pdcch = nrWavegenPDCCHConfig` creates a default PDCCH configuration object for 5G waveform generation.

`pdcch = nrWavegenPDCCHConfig(Name,Value)` specifies properties on page 3-186 using one or more name-value arguments. Enclose each property name in quotes. For example, `'AggregationLevel',2` configures the PDCCH with the specified aggregation level.

Properties

`Enable` — Enable PDCCH
`1` (default) | `0`

Enable the PDCCH in 5G waveform generation, specified as one of these values.

- `1` — Enable the PDCCH.
- `0` — Disable the PDCCH.

Data Types: `double` | `logical`

`Label` — Name of PDCCH configuration
`'PDCCH1'` (default) | character array | string scalar

Name of the PDCCH configuration, specified as a character array or string scalar. Use this property to set a description to the PDCCH configuration.

Data Types: `char` | `string`

`Power` — Power scaling of PDCCH in dB
`0` (default) | real scalar

Power scaling of the PDCCH in dB, specified as a real scalar. Use this property to scale the power of the PDCCH in the generated 5G waveform.

Data Types: `double`

`BandwidthPartID` — ID of BWP
`1` (default) | nonnegative integer

ID of the bandwidth part (BWP) containing the configured PDCCH, specified as a nonnegative integer. Use this property to associate this PDCCH configuration with one of the BWP configurations specified by the `BandwidthParts` property of the `nrDLCarrierConfig` object.

Data Types: `double`

`SearchSpaceID` — ID of search space set
`1` (default) | nonnegative integer

ID of the search space set containing the configured PDCCH, specified as a nonnegative integer. Use this property to associate this PDCCH configuration with one of the search space set configurations specified by the `SearchSpaces` property of the `nrDLCarrierConfig` object.

Data Types: `double`

`AggregationLevel` — PDCCH aggregation level
`8` (default) | `1` | `2` | `4` | `16`

PDCCH aggregation level, specified as `1`, `2`, `4`, `8`, or `16`.

Data Types: `double`

`AllocatedCandidate` — Candidate used for PDCCH instance
`1` (default) | integer from 1 to 8

Candidate used for the PDCCH instance, specified as an integer from 1 to 8. The value of this property is an index from the set of candidates specified for the aggregation level by the `NumCandidates` property of the search space specified by the `SearchSpaceID` property.

This property does not apply when the `CCEOffset` property is set to a nonempty value.

Data Types: `double`

`CCEOffset` — CCE offset used for PDCCH instance
`[]` (default) | positive integer

Control channel element (CCE) offset used for the PDCCH instance, specified as one of these options.

- `[]` — The `AllocatedCandidate` property determines the first CCE used for the PDCCH instance.
- Positive integer — This value explicitly specifies the index of the first CCE used for the PDCCH instance. The value must be a multiple of the `AggregationLevel` property value and less than or equal to the CCE capacity of the associated CORESET. This option overrides the allocation specified by the `AllocatedCandidate` property.

Data Types: `double`

`SlotAllocation` — Slot allocation in PDCCH period
`0` (default) | nonnegative integer | row vector of nonnegative integers

Slot allocation in a PDCCH period, specified as a nonnegative integer or row vector of nonnegative integers. This property specifies the slot positions of the PDCCH by using 0-based indexing and values smaller than the value of the `Period` property. The object ignores slot allocation values larger than the period. This slot allocation must be within the slot allocation of the search space specified by the `SearchSpaceID` property.

Data Types: `double`

`Period` — PDCCH allocation period in slots
`1` (default) | nonnegative integer | `[]`

PDCCH allocation period in slots, specified as a nonnegative integer or `[]`. An empty period indicates no repetition.

Data Types: `double`

`Coding` — Enable DCI encoding
`1` (default) | nonnegative integer

Enable downlink control information (DCI) encoding, specified as one of these values.

- `1` — Enable DCI encoding.
- `0` — Disable DCI encoding.

Data Types: `double` | `logical`

`DataBlockSize` — Length of DCI in bits
`20` (default) | integer from 0 to 140

Length of DCI in bits, specified as an integer from 0 to 140.

Dependencies

To enable this property, set the `Coding` property to `1`.

Data Types: `double`

`DataSource` — Source of DCI contents
`'PN9-ITU'` (default) | `'PN9'` | `'PN11''PN15'` | `'PN23'` | cell array | binary vector

Source of DCI contents, specified as one of these options:

- `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, or `'PN23'`
- Two-element cell array consisting of one of the character vectors from the previous list and a random numeric seed (for example, `{'PN9',7}`)
- Binary vector

If you do not specify a random seed, all shift registers are initialized with an active state.

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

`RNTI` — Radio network temporary identifier
`1` (default) | integer from 0 to 65,535

Radio network temporary identifier of the user equipment (UE), specified as an integer from 0 to 65,535. When you set this property to a value greater than 65,519, the object infers this property value to be `0`.

Data Types: double

DMRSScramblingID — PDCCH DM-RS scrambling identity
2 (default) | integer from 0 to 65,535 | [ ]

PDCCH demodulation reference signal (DM-RS) scrambling identity, specified as an integer from 0 to 65,535 if the higher layer parameter *pdcch-DMRS-ScramblingID* is configured or as [] if *pdcch-DMRS-ScramblingID* is not configured. When you specify this property as [], the object sets the PDCCH DM-RS scrambling identity to the physical layer cell identity specified by the NCellID property of the carrier.

Data Types: double

DMRSPower — Power scaling of PDCCH DM-RS in dB
0 (default) | real scalar

Power scaling of the PDCCH DM-RS in dB, specified as a real scalar. Use this property to scale the power of the PDCCH DM-RS in the generated 5G waveform. This scaling is additional to the PDCCH-wide power scaling specified by the Power property.

Data Types: double

Examples

Configure PDCCH for 5G Downlink Waveform Generation

Create a default CORESET configuration object.

```
coreset = nrCORESETConfig;
```

Create a search space set configuration object, associating the search space set with the previously defined CORESET configuration.

```
searchSpace = nrSearchSpaceConfig('CORESETID',coreset.CORESETID);
```

Create a PDCCH configuration object for 5G waveform generation with the specified property values.

```
pdcch = nrWavegenPDCCHConfig( ...
    'SearchSpaceID',searchSpace.SearchSpaceID, ...
    'AggregationLevel',4, ...
    'AllocatedCandidate',2, ...
    'SlotAllocation',[0 2], ...
    'Period',3);
```

Create a downlink carrier configuration object, specifying the previously defined configurations.

```
cfgDL = nrDLCarrierConfig( ...
    'CORESET',{coreset}, ...
    'SearchSpaces',{searchSpace}, ...
    'PDCCH',{pdcch});
```

Configure Multiple PDCCH for 5G Downlink Waveform Generation

Create two CORESET configuration objects with unique IDs.

```
coreset1 = nrCORESETConfig('CORESETID',1);
coreset2 = nrCORESETConfig('CORESETID',2);
```

Create three search space set configuration objects with unique IDs. Associate each search space set with one of the previously defined CORESET configurations.

```
searchSpace1 = nrSearchSpaceConfig('SearchSpaceID',1,'CORESETID',coreset1.CORESETID);
searchSpace2 = nrSearchSpaceConfig('SearchSpaceID',2,'CORESETID',coreset1.CORESETID);
searchSpace3 = nrSearchSpaceConfig('SearchSpaceID',3,'CORESETID',coreset2.CORESETID);
```

Create four PDCCH configuration objects for 5G waveform generation. Specify a unique UE and one of the search space set configurations for each PDCCH.

```
pdcch1 = nrWavegenPDCCHConfig('RNTI',1,'SearchSpaceID',searchSpace1.SearchSpaceID);
pdcch2 = nrWavegenPDCCHConfig('RNTI',2,'SearchSpaceID',searchSpace2.SearchSpaceID);
pdcch3 = nrWavegenPDCCHConfig('RNTI',3,'SearchSpaceID',searchSpace2.SearchSpaceID);
pdcch4 = nrWavegenPDCCHConfig('RNTI',4,'SearchSpaceID',searchSpace3.SearchSpaceID);
```

Create a downlink carrier configuration object, specifying the previously defined configurations.

```
cfgDL = nrDLCarrierConfig( ...
    'CORESET',{coreset1,coreset2}, ...
    'SearchSpaces',{searchSpace1,searchSpace2,searchSpace3}, ...
    'PDCCH',{pdcch1,pdcch2,pdcch3,pdcch4});
```

Version History

Introduced in R2020b

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.213. “NR; Physical layer procedures for control.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrWaveformGenerator`

Objects
`nrDLCarrierConfig` | `nrSearchSpaceConfig` | `nrCORESETConfig`

nrWavegenPDSCHConfig

PDSCH configuration parameters for 5G waveform generation

Description

The `nrWavegenPDSCHConfig` object sets physical downlink shared channel (PDSCH) configuration parameters, as defined in TS 38.211 Sections 7.3.1, 7.4.1.1, and 7.4.1.2 [1]. Use this object to set the `PDSCH` property of the `nrDLCarrierConfig` object when configuring 5G downlink waveform generation.

This object defines several properties of the PDSCH, including the modulation scheme, layer mapping, target code rate, time-domain and frequency-domain allocation, and virtual resource blocks (VRB) to physical resource blocks (PRBs) interleaving. The object also contains properties of the associated physical reference signals, such as the demodulation reference signal (DM-RS) and the phase tracking reference signal (PT-RS).

The default `nrWavegenPDSCHConfig` object configures a single-layer PDSCH with mapping type A, QPSK modulation, a resource allocation of 52 resource blocks and 14 OFDM symbols in a slot, transmission in all slots, and single-symbol DM-RS type 1. This configuration corresponds to a full resource allocation with respect to the default `nrWavegenBWPConfig` object.

Creation

Syntax

```
pdsch = nrWavegenPDSCHConfig
pdsch = nrWavegenPDSCHConfig(Name,Value)
```

Description

`pdsch = nrWavegenPDSCHConfig` creates a default PDSCH configuration object for 5G waveform generation.

`pdsch = nrWavegenPDSCHConfig(Name,Value)` specifies properties on page 3-191 using one or more name-value arguments. Enclose each property name in quotes. For example, `'NumLayers',7` specifies seven transmission layers.

Properties

Channel Configuration

`Enable` — Enable PDSCH
`1` (default) | `0`

Enable the PDSCH in 5G waveform generation, specified as one of these values.

- `1` — Enable the PDSCH.

- `0` — Disable the PDSCH.

Data Types: `double` | `logical`

`Label` — Name of PDSCH configuration
`'PDSCH1'` (default) | character array | string scalar

Name of the PDSCH configuration, specified as a character array or string scalar. Use this property to set a description to the PDSCH configuration.

Data Types: `char` | `string`

`Power` — Power scaling of PDSCH in dB
`0` (default) | real scalar

Power scaling of the PDSCH in dB, specified as a real scalar. Use this property to scale the power of the PDSCH in the generated 5G waveform.

Data Types: `double`

`BandwidthPartID` — ID of BWP
`1` (default) | nonnegative integer

ID of the bandwidth part (BWP) containing the configured PDSCH, specified as a nonnegative integer. Use this property to associate this PDSCH configuration with one of the BWP configurations specified by the `BandwidthParts` property of the `nrDLCarrierConfig` object.

Data Types: `double`

`Modulation` — Modulation scheme
`'QPSK'` (default) | `'16QAM'` | `'64QAM'` | `'256QAM'` | string scalar | string array | cell array of character vectors

Modulation scheme, specified as `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`, a string scalar, a string array, or a cell array of character vectors. This modulation scheme specifies the modulation type of the codewords and the number of bits used per modulation symbol. For one codeword, specify the modulation scheme as a character vector or string scalar. If two codewords are present (`NumLayers` > 4), the same modulation scheme applies to both codewords or you can specify different modulation schemes for each codeword by using a string array or a cell array of character vectors.

Modulation Scheme	Number of Bits Per Symbol
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

Example: `{'QPSK','16QAM'}` or `["QPSK","16QAM"]` specifies different modulation schemes for two codewords.

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

`NumLayers` — Number of transmission layers
`1` (default) | integer from 1 to 8

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

- For one codeword, specify an integer from 1 to 4.
- For two codewords, specify an integer from 5 to 8.

Data Types: double

MappingType — Mapping type
'A' (default) | 'B'

Mapping type of the physical shared channel, specified as 'A' or 'B'.

Data Types: char | string

ReservedPRB — Reserved PRBs and OFDM symbols pattern in BWP
default nrPDSCHReservedConfig object (default) | cell array of nrPDSCHReservedConfig objects

Reserved PRBs and OFDM symbols pattern in the BWP, specified as a cell array of nrPDSCHReservedConfig objects.

Data Types: cell

ReservedCORESET — CORESET IDs for PDSCH rate matching
[] (default) | integer from 0 to 11 | vector of integers from 0 to 11

Control resource set (CORESET) IDs for PDSCH rate matching, specified as [], an integer from 0 to 11, or a vector of integers from 0 to 11. The integers must match the CORESETID property values of the nrCORESETConfig objects specified by the nrDLCarrierConfig.CORESET property. When you set this property to a value other than [], this property specifies PDSCH rate matching around the denoted CORESET and associated search spaces.

Data Types: double

SymbolAllocation — OFDM symbol allocation
[0 14] (default) | two-element vector of nonnegative integers

OFDM symbol allocation of the physical shared channel, specified as a two-element vector of nonnegative integers. The first element of this property represents the start of symbol allocation (0-based). The second element represents the number of allocated OFDM symbols.

Data Types: double

SlotAllocation — Slot allocation in PDSCH period
[0:9] (default) | nonnegative integer | row vector of nonnegative integers

Slot allocation in a PDSCH period, specified as a nonnegative integer or row vector of nonnegative integers. This property specifies the slot positions of the PDSCH by using 0-based indexing and values smaller than the value of the Period property. The object ignores slot allocation values larger than the period.

Data Types: double

Period — PDSCH allocation period in slots
10 (default) | nonnegative integer | []

PDSCH allocation period in slots, specified as a nonnegative integer or []. An empty period indicates no repetition.

Data Types: double

PRBSet — PRB allocation
[0:51] (default) | vector of integers from 0 to 274

PRB allocation of the PDSCH in the BWP, specified as a vector of integers from 0 to 274.

Data Types: double

VRBToPRBInterleaving — Enable VRB-to-PRB interleaving
0 (default) | 1

Enable VRB-to-PRB interleaving, specified as one of these values.

- 0 — Disable VRB-to-PRB interleaving.
- 1 — Enable VRB-to-PRB interleaving.

Data Types: double | logical

VRBBundleSize — VRB bundle size
2 (default) | 4

VRB bundle size, in terms of the number of PRBs for VRB-to-PRB interleaving, specified as 2 or 4.

Dependencies

To enable this property, set the VRBToPRBInterleaving property to 1.

Data Types: double

NID — PDSCH scrambling identity
[] (default) | integer from 0 to 1023

PDSCH scrambling identity, specified as [] or an integer from 0 to 1023.

- If the higher layer parameter *dataScramblingIdentityPDSCH* is configured, NID must be in the range from 0 to 1023.
- If the higher layer parameter *dataScramblingIdentityPDSCH* is not configured, NID must be in the range from 0 to 1007.

When you specify this property as [], the object sets the PDSCH scrambling identity to the physical layer cell identity, specified by the NCellID property of the carrier.

Data Types: double

RNTI — Radio network temporary identifier
1 (default) | integer from 0 to 65,535

Radio network temporary identifier of the user equipment (UE), specified as an integer from 0 to 65,535.

Data Types: double

Coding — Enable DL-SCH encoding of transport blocks
1 (default) | nonnegative integer

Enable downlink shared channel (DL-SCH) encoding of transport blocks, specified as one of these values.

- `1` — Enable transport block encoding.
- `0` — Disable transport block encoding.

Data Types: `double` | `logical`

`TargetCodeRate` — Target code rate
`0.5137` (default) | numeric scalar | 1-by-2 numeric vector

Target code rate, specified as a numeric scalar or a 1-by-2 numeric vector of values in the range (0, 1). The default value corresponds to 526/1024. If you specify this property as a scalar, the object applies scalar expansion when processing two transport blocks (`NumLayers` > 4). To specify different target code rates for each transport block, specify this property as a vector.

Dependencies

To enable this property, set the `Coding` property to `1`.

Data Types: `double`

`TBScaling` — Codeword scaling factor
`1` (default) | `0.25` | `0.5` | 1-by-2 integer vector

Codeword scaling factor, specified as one of these options.

- For one codeword, specify `0.25`, `0.5`, or `1`.
- For two codewords (`NumLayers` > 4), specify a 1-by-2 integer vector with values `0.25`, `0.5`, or `1`.

Dependencies

To enable this property, set the `Coding` property to `1`.

Data Types: `double`

`XOverhead` — Rate matching overhead
`0` (default) | `6` | `12` | `18`

Rate matching overhead, specified as `0`, `6`, `12`, or `18`.

Data Types: `double`

`RVSequence` — Redundancy version sequence
`[0 2 3 1]` (default) | nonnegative integer | vector of nonnegative integers | two-element cell array

Redundancy version sequence, specified as a nonnegative integer, a vector of nonnegative integers, or a two-element cell array containing unique nonnegative integers. When the sequence is a two-element cell array, the second value only applies to a second codeword (`NumLayers` > 4).

Dependencies

To enable this property, set the `Coding` property to `1`.

Data Types: `double`

`DataSource` — Source of contents for transport blocks
`'PN9-ITU'` (default) | `'PN9'` | `'PN11'` | `'PN15'` | `'PN23'` | two-element cell array | binary-valued vector

Source of contents for the transport blocks, specified as one of these options.

- 'PN9-ITU', 'PN9', 'PN11', 'PN15', or 'PN23'
- Two-element cell array consisting of one of the character vectors from the previous list and a random numeric seed (for example, {'PN9',7})
- Binary-valued vector

If you do not specify a random seed, all shift registers are initialized with an active state.

Data Types: double | cell | string | char

Reference Signals Configuration

DMRS — PDSCH DM-RS configuration parameters
default nrPDSCHDMRSConfig object (default) | nrPDSCHDMRSConfig object

PDSCH DM-RS configuration parameters, specified as an nrPDSCHDMRSConfig object.

DMRSPower — Power scaling of PDSCH DM-RS in dB
0 (default) | real scalar

Power scaling of the PDSCH DM-RS in dB, specified as a real scalar. Use this property to scale the power of the PDSCH DM-RS in the generated 5G waveform. This scaling is additional to the PDSCH-wide power scaling specified by the Power property.

Data Types: double

EnablePTRS — Enable PT-RS
0 or false (default) | 1 or true

Enable the PT-RS, specified as one of these values.

- 0 (false) — Disable the PT-RS configuration.
- 1 (true) — Enable the PT-RS configuration.

Data Types: double | logical

PTRS — PDSCH PT-RS configuration parameters
default nrPDSCHPTRSConfig object (default) | nrPDSCHPTRSConfig object

PDSCH PT-RS configuration, specified as an nrPDSCHPTRSConfig object.

Dependencies

To enable this property, set the EnablePTRS property to 1.

PTRSPower — Power scaling of PDSCH PT-RS in dB
0 (default) | real scalar

Power scaling of the PDSCH PT-RS in dB, specified as a real scalar. Use this property to scale the power of the PDSCH PT-RS in the generated 5G waveform. This scaling is additional to the PDSCH-wide power scaling specified by the Power property.

Dependencies

To enable this property, set the EnablePTRS property to 1.

Data Types: double

Examples

Configure PDSCH for 5G Downlink Waveform Generation

Create a PDSCH configuration object for 5G waveform generation with the specified property values.

```
pdsch = nrWavegenPDSCHConfig( ...
    'BandwidthPartID',0, ...
    'Modulation','16QAM', ...
    'TargetCodeRate',658/1024, ...
    'SymbolAllocation',[0 7], ...
    'SlotAllocation',[0 2], ...
    'Period',3, ...
    'PRBSet',[0:20], ...
    'EnablePTRS',true);
```

Create a downlink carrier configuration object, specifying the previously defined PDSCH configuration.

```
cfg = nrDLCarrierConfig('PDSCH',{pdsch});
```

Configure Multiple PDSCH for 5G Downlink Waveform Generation

Create two SCS carrier configuration objects with mixed numerologies.

```
carrier1 = nrSCSCarrierConfig('SubcarrierSpacing',15);
carrier2 = nrSCSCarrierConfig('SubcarrierSpacing',30);
```

Create two BWP configuration objects, one for each of the SCS carriers.

```
bwp1 = nrWavegenBWPConfig('BandwidthPartID',0,'SubcarrierSpacing',15);
bwp2 = nrWavegenBWPConfig('BandwidthPartID',1,'SubcarrierSpacing',30);
```

Create two PDSCH configuration objects for 5G waveform generation, specifying a unique UE and one of the BWP configurations for each PDSCH.

```
pdsch1 = nrWavegenPDSCHConfig('RNTI',1,'BandwidthPartID',0,'Modulation','QPSK');
pdsch2 = nrWavegenPDSCHConfig('RNTI',2,'BandwidthPartID',1,'Modulation','16QAM');
```

Create a downlink carrier configuration object, specifying the previously defined configurations.

```
cfg = nrDLCarrierConfig( ...
    'SCSCarriers',{carrier1,carrier2}, ...
    'BandwidthParts',{bwp1,bwp2}, ...;
    'PDSCH',{pdsch1,pdsch2});
```

Version History
Introduced in R2020b

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*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrWaveformGenerator`

Objects
`nrDLCarrierConfig`

nrWavegenPUCCH0Config

PUCCH format 0 configuration parameters for 5G waveform generation

Description

The `nrWavegenPUCCH0Config` object sets physical uplink control channel (PUCCH) configuration parameters for format 0, as defined in TS 38.211 Sections 6.3.2.1 to 6.3.2.3 [1]. Use this object to set the `PUCCH` property of the `nrULCarrierConfig` object when you configure 5G uplink waveform generation.

The default `nrWavegenPUCCH0Config` object allocates a PUCCH format 0 in the first resource block (RB) of the bandwidth part (BWP) and the last OFDM symbol in the slot of 14 OFDM symbols. PUCCH format 0 transmission is in all slots and carries a single uplink control information (UCI) bit and no scheduling request (SR).

Creation

Syntax

```
pucch = nrWavegenPUCCH0Config
pucch = nrWavegenPUCCH0Config(Name=Value)
```

Description

`pucch = nrWavegenPUCCH0Config` creates a default PUCCH format 0 configuration object for 5G waveform generation.

`pucch = nrWavegenPUCCH0Config(Name=Value)` sets properties on page 3-199 using one or more name-value arguments. For example, `Period=20` specifies 20 slots for the PUCCH allocation period.

Properties

Enable — Option to enable PUCCH configuration
`true` or `1` (default) | `false` or `0`

Option to enable this PUCCH configuration in 5G waveform generation, specified as one of these numeric or logical values.

- `1` (`true`) — Enable this PUCCH.
- `0` (`false`) — Disable this PUCCH.

Data Types: `double` | `logical`

Label — Name of PUCCH format 0 configuration
`'PUCCH format 0'` (default) | character vector | string scalar

Name of this PUCCH format 0 configuration, specified as a character vector or string scalar. Use this property to set a description to the configuration.

Data Types: char | string

Power — Power scaling of PUCCH in dB
0 (default) | real-valued scalar

Power scaling of the PUCCH in dB, specified as a real-valued scalar. Use this property to scale the power of the configured PUCCH in the generated 5G waveform.

Data Types: double

BandwidthPartID — ID of BWP
1 (default) | nonnegative integer

ID of the BWP containing the configured PUCCH, specified as a nonnegative integer. Use this property to associate this PUCCH configuration with one of the BWP configurations specified by the BandwidthParts property of the nrULCarrierConfig object.

Data Types: double

SymbolAllocation — OFDM symbol allocation of PUCCH
[13 1] (default) | [] | two-element vector of nonnegative integers

OFDM symbol allocation of the PUCCH within a slot, specified as [] or a two-element vector of nonnegative integers. The first element of this property represents the start of symbol allocation. The second element represents the number of consecutively allocated OFDM symbols and must equal 1 or 2. To indicate that no OFDM symbols are allocated for the PUCCH, use [] or specify the second element of the vector as 0.

Data Types: double

SlotAllocation — Slot allocation in PUCCH period
[0:9] (default) | nonnegative integer | row vector of nonnegative integers

Slot allocation in a PUCCH period, specified as a nonnegative integer or a row vector of nonnegative integers. This property specifies the slot positions of the PUCCH by using 0-based indexing and values that are less than the value of the Period property. The object ignores slot allocation values that are greater than the period.

Data Types: double

Period — PUCCH allocation period in slots
10 (default) | nonnegative integer | []

PUCCH allocation period in slots, specified as a nonnegative integer or []. An empty period indicates no repetition.

Data Types: double

PRBSet — PRB allocation
0 (default) | integer from 0 to 274 | []

Physical resource block (PRB) allocation of the PUCCH within the BWP, specified as an integer from 0 to 274 or []. To indicate that no resource blocks are allocated for the PUCCH, use [].

Data Types: double

FrequencyHopping — Frequency hopping configuration for PUCCH
'neither' (default) | 'intraSlot' | 'interSlot'

Frequency hopping configuration for the PUCCH, specified as one of these values.

- 'neither' — Indicates no frequency hopping
- 'intraSlot' — Enables intraslot frequency hopping
- 'interSlot' — Enables interslot frequency hopping

When you enable interslot frequency hopping, the slot number determines the starting resource block for the allocated OFDM symbols. For even-numbered slots, the resource block starts from the minimum resource block allocated in the PRBSet property. For odd-numbered slots, the resource block starts from the resource block provided in the SecondHopStartPRB property.

Data Types: char | string

SecondHopStartPRB — Starting PRB index of second hop relative to BWP
1 (default) | integer from 0 to 274

Starting PRB index of the second hop relative to the BWP, specified as an integer from 0 to 274.

Data Types: double

GroupHopping — Group hopping configuration for PUCCH
'neither' (default) | 'enable' | 'disable'

Group hopping configuration for the PUCCH, specified as one of these values.

- 'neither' — Disables group hopping and sequence hopping
- 'enable' — Enables group hopping and disables sequence hopping
- 'disable' — Disables group hopping and enables sequence hopping

This property is the higher-layer parameter *pucch-GroupHopping*.

Data Types: char | string

HoppingID — Hopping identity of PUCCH
[] (default) | integer from 0 to 1023

Hopping identity of the PUCCH, specified as [] or an integer from 0 to 1023.

- If the higher-layer parameter *hoppingId* is configured, this property must be an integer from 0 to 1023.
- If the higher-layer parameter *hoppingId* is not configured, this property must be equal to the physical layer cell identity and be an integer from 0 to 1007.

When you specify this property as [], the object sets the PUCCH hopping identity to the physical layer cell identity specified by the NCellID property of the nrULCarrierConfig object.

Data Types: double

InitialCyclicShift — Initial cyclic shift
0 (default) | integer from 0 to 11

Initial cyclic shift, specified as an integer from 0 to 11.

This property is the higher-layer parameter *initialCyclicShift*.

Data Types: `double`

`NumUCIBits` — Number of UCI bits
`1` (default) | `0` | `2`

Number of UCI bits, specified as `0`, `1`, or `2`. For no UCI transmission, set this property to `0`.

Data Types: `double`

`DataSourceUCI` — Source of UCI contents
`'PN9-ITU'` (default) | `'PN9'` | `'PN11'` | `'PN15'` | `'PN23'` | two-element cell array | binary-valued vector

Source of UCI contents, specified as one of these options.

- `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, or `'PN23'`
- Two-element cell array consisting of one of the character vectors from the previous list and a random numeric seed (for example, `{'PN9',7}`)
- Binary-valued vector

If you do not specify a random seed, the object initializes all shift registers with an active state.

Dependencies

To enable this property, set the `NumUCIBits` property to a positive integer.

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

`DataSourceSR` — Source of SR content
`0` (default) | `'PN9-ITU'` | `'PN9'` | `'PN11'` | `'PN15'` | `'PN23'` | two-element cell array | binary-valued vector

Source of SR content, specified as one of these options.

- `0` — Indicates no UCI transmission.
- `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, or `'PN23'`
- Two-element cell array consisting of one of the character vectors from the previous list and a random numeric seed (for example, `{'PN9',7}`)
- Binary-valued vector

If you do not specify a random seed, the object initializes all shift registers with an active state.

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

Examples

Configure Multiple PUCCH for 5G Uplink Waveform Generation

Create two SCS carrier configuration objects with mixed numerologies.

```
carrier1 = nrSCSCarrierConfig(SubcarrierSpacing=15);
carrier2 = nrSCSCarrierConfig(SubcarrierSpacing=30);
```

Create two BWP configuration objects, one for each of the SCS carriers.

```
bwp1 = nrWavegenBWPConfig(BandwidthPartID=1,SubcarrierSpacing=15);
bwp2 = nrWavegenBWPConfig(BandwidthPartID=2,SubcarrierSpacing=30);
```

Create a PUCCH format 0 configuration object for the first SCS carrier and a PUCCH format 1 configuration object for the second SCS carrier. For configuring PUCCH format 2, 3, or 4, use the `nrWavegenPUCCH2Config`, `nrWavegenPUCCH3Config`, or `nrWavegenPUCCH4Config` configuration objects, respectively. By default, each PUCCH configuration object enables the PUCCH on their respective carriers.

```
pucch1 = nrWavegenPUCCH0Config( ...
    BandwidthPartID=1,SlotAllocation=0:9,PRBSet=2,DataSourceUCI='PN9');
pucch2 = nrWavegenPUCCH1Config( ...
    BandwidthPartID=2,SlotAllocation=0:9,PRBSet=2,DataSourceUCI='PN9');
```

Create a PUSCH configuration object such that the PUSCH does not overlap with the previously configured PUCCH in the generated waveform.

```
pusch = nrWavegenPUSCHConfig(BandwidthPartID=1, ...
    SymbolAllocation=[0,8],PRBSet=10:51);
```

Create an uplink carrier configuration object for 5G waveform generation, specifying the previously defined configurations. You can disable the PUCCH in either SCS carrier by setting the corresponding `cfg.PUCCH{1}.Enable` or `cfg.PUCCH{2}.Enable` properties, respectively, to `false`.

```
cfg = nrULCarrierConfig( ...
    SCSCarriers={carrier1,carrier2}, ...
    BandwidthParts={bwp1,bwp2}, ...
    PUSCH={pusch}, ...
    PUCCH={pucch1,pucch2});
```

Version History

Introduced in R2021b

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrWaveformGenerator`

Objects
`nrULCarrierConfig` | `nrWavegenPUCCH1Config` | `nrWavegenPUCCH2Config` | `nrWavegenPUCCH3Config` | `nrWavegenPUCCH4Config`

nrWavegenPUCCH1Config

PUCCH format 1 configuration parameters for 5G waveform generation

Description

The `nrWavegenPUCCH1Config` object sets physical uplink control channel (PUCCH) configuration parameters for format 1, as defined in TS 38.211 Sections 6.3.2.1, 6.3.2.2, 6.3.2.4, and 6.4.1.3.1 [1]. Use this object to set the `PUCCH` property of the `nrULCarrierConfig` object when you configure 5G uplink waveform generation.

The default `nrWavegenPUCCH1Config` object allocates a PUCCH format 1 in the first resource block (RB) of the bandwidth part (BWP) and in all OFDM symbols in a slot. PUCCH format 1 transmission is in all slots and carries a single uplink control information (UCI) bit.

Creation

Syntax

```
pucch = nrWavegenPUCCH1Config
pucch = nrWavegenPUCCH1Config(Name=Value)
```

Description

`pucch = nrWavegenPUCCH1Config` creates a default PUCCH format 1 configuration object for 5G waveform generation.

`pucch = nrWavegenPUCCH1Config(Name=Value)` sets properties on page 3-205 using one or more name-value arguments. For example, `Period=20` specifies 20 slots for the PUCCH allocation period.

Properties

`Enable` — Option to enable PUCCH configuration
`true` or `1` (default) | `false` or `0`

Option to enable this PUCCH configuration in 5G waveform generation, specified as one of these numeric or logical values.

- `1` (`true`) — Enable this PUCCH.
- `0` (`false`) — Disable this PUCCH.

Data Types: `double` | `logical`

`Label` — Name of PUCCH format 1 configuration
`'PUCCH format 1'` (default) | character vector | string scalar

Name of this PUCCH format 1 configuration, specified as a character vector or string scalar. Use this property to set a description to the configuration.

Data Types: char | string

Power — Power scaling of PUCCH in dB
0 (default) | real-valued scalar

Power scaling of the PUCCH in dB, specified as a real-valued scalar. Use this property to scale the power of the configured PUCCH in the generated 5G waveform.

Data Types: double

BandwidthPartID — ID of BWP
1 (default) | nonnegative integer

ID of the BWP containing the configured PUCCH, specified as a nonnegative integer. Use this property to associate this PUCCH configuration with one of the BWP configurations specified by the BandwidthParts property of the nrULCarrierConfig object.

Data Types: double

SymbolAllocation — OFDM symbol allocation of PUCCH
[0 14] (default) | [] | two-element vector of nonnegative integers

OFDM symbol allocation of the PUCCH within a slot, specified as [] or a two-element vector of nonnegative integers. The first element of this property represents the start of symbol allocation. The second element represents the number of consecutively allocated OFDM symbols and must be in the range [4, 14]. To indicate that no OFDM symbols are allocated for the PUCCH, use [] or specify the second element of the vector as 0.

Data Types: double

SlotAllocation — Slot allocation in PUCCH period
[0:9] (default) | nonnegative integer | row vector of nonnegative integers

Slot allocation in a PUCCH period, specified as a nonnegative integer or a row vector of nonnegative integers. This property specifies the slot positions of the PUCCH by using 0-based indexing and values that are less than the value of the Period property. The object ignores slot allocation values that are greater than the period.

Data Types: double

Period — PUCCH allocation period in slots
10 (default) | nonnegative integer | []

PUCCH allocation period in slots, specified as a nonnegative integer or []. An empty period indicates no repetition.

Data Types: double

PRBSet — PRB allocation
0 (default) | integer from 0 to 274 | []

Physical resource block (PRB) allocation of the PUCCH within the BWP, specified as an integer from 0 to 274 or []. To indicate that no resource blocks are allocated for the PUCCH, use [].

Data Types: double

FrequencyHopping — Frequency hopping configuration for PUCCH
'neither' (default) | 'intraSlot' | 'interSlot'

Frequency hopping configuration for the PUCCH, specified as one of these values.

- `'neither'` — Indicates no frequency hopping
- `'intraSlot'` — Enables intraslot frequency hopping
- `'interSlot'` — Enables interslot frequency hopping

When you enable interslot frequency hopping, the slot number determines the starting resource block for the allocated OFDM symbols. For even-numbered slots, the resource block starts from the minimum resource block allocated in the `PRBSet` property. For odd-numbered slots, the resource block starts from the resource block provided in the `SecondHopStartPRB` property.

Data Types: `char` | `string`

`SecondHopStartPRB` — Starting PRB index of second hop relative to BWP
`1` (default) | integer from 0 to 274

Starting PRB index of the second hop relative to the BWP, specified as an integer from 0 to 274.

Data Types: `double`

`GroupHopping` — Group hopping configuration for PUCCH
`'neither'` (default) | `'enable'` | `'disable'`

Group hopping configuration for the PUCCH, specified as one of these values.

- `'neither'` — Disables group hopping and sequence hopping
- `'enable'` — Enables group hopping and disables sequence hopping
- `'disable'` — Disables group hopping and enables sequence hopping

This property is the higher-layer parameter *pucch-GroupHopping*.

Data Types: `char` | `string`

`HoppingID` — Hopping identity of PUCCH
`[]` (default) | integer from 0 to 1023

Hopping identity of the PUCCH, specified as `[]` or an integer from 0 to 1023.

- If the higher-layer parameter *hoppingId* is configured, this property must be an integer from 0 to 1023.
- If the higher-layer parameter *hoppingId* is not configured, this property must be equal to the physical layer cell identity and be an integer from 0 to 1007.

When you specify this property as `[]`, the object sets the PUCCH hopping identity to the physical layer cell identity specified by the `NCellID` property of the `nrULCarrierConfig` object.

Data Types: `double`

`InitialCyclicShift` — Initial cyclic shift
`0` (default) | integer from 0 to 11

Initial cyclic shift, specified as an integer from 0 to 11.

This property is the higher-layer parameter *initialCyclicShift*.

Data Types: `double`

OCCI — Orthogonal cover code index
0 (default) | integer from 0 to 6

Orthogonal cover code index (OCCI), specified as an integer from 0 to 6.

- When you disable intraslot frequency hopping, the OCCI value must be less than the floor of half of the number of OFDM symbols allocated for the PUCCH.
- When you enable intraslot frequency hopping, the OCCI value must be less than the floor of one-fourth of the number of OFDM symbols allocated for the PUCCH.

Data Types: double

NumUCIBits — Number of UCI bits
1 (default) | 0 | 2

Number of UCI bits, specified as 0, 1, or 2. For no UCI transmission, set this property to 0.

Data Types: double

DataSourceUCI — Source of UCI contents
'PN9-ITU' (default) | 'PN9' | 'PN11' | 'PN15' | 'PN23' | two-element cell array | binary-valued vector

Source of UCI contents, specified as one of these options.

- 'PN9-ITU', 'PN9', 'PN11', 'PN15', or 'PN23'
- Two-element cell array consisting of one of the character vectors from the previous list and a random numeric seed (for example, {'PN9',7})
- Binary-valued vector

If you do not specify a random seed, the object initializes all shift registers with an active state.

Dependencies

To enable this property, set the NumUCIBits property to a positive integer.

Data Types: double | cell | string | char

DMRSPower — Power scaling of PUCCH DM-RS in dB
0 (default) | real-valued scalar

Power scaling of the PUCCH DM-RS in dB, specified as a real-valued scalar. Use this property to scale the power of the PUCCH DM-RS in the generated 5G waveform. This scaling is additional to the PUCCH-wide power scaling specified by the Power property.

Data Types: double

Examples

Configure Multiple PUCCH for 5G Uplink Waveform Generation

Create two SCS carrier configuration objects with mixed numerologies.

```
carrier1 = nrSCSCarrierConfig(SubcarrierSpacing=15);
carrier2 = nrSCSCarrierConfig(SubcarrierSpacing=30);
```

Create two BWP configuration objects, one for each of the SCS carriers.

```
bwp1 = nrWavegenBWPConfig(BandwidthPartID=1,SubcarrierSpacing=15);
bwp2 = nrWavegenBWPConfig(BandwidthPartID=2,SubcarrierSpacing=30);
```

Create a PUCCH format 0 configuration object for the first SCS carrier and a PUCCH format 1 configuration object for the second SCS carrier. For configuring PUCCH format 2, 3, or 4, use the `nrWavegenPUCCH2Config`, `nrWavegenPUCCH3Config`, or `nrWavegenPUCCH4Config` configuration objects, respectively. By default, each PUCCH configuration object enables the PUCCH on their respective carriers.

```
pucch1 = nrWavegenPUCCH0Config( ...
    BandwidthPartID=1,SlotAllocation=0:9,PRBSet=2,DataSourceUCI='PN9');
pucch2 = nrWavegenPUCCH1Config( ...
    BandwidthPartID=2,SlotAllocation=0:9,PRBSet=2,DataSourceUCI='PN9');
```

Create a PUSCH configuration object such that the PUSCH does not overlap with the previously configured PUCCH in the generated waveform.

```
pusch = nrWavegenPUSCHConfig(BandwidthPartID=1, ...
    SymbolAllocation=[0,8],PRBSet=10:51);
```

Create an uplink carrier configuration object for 5G waveform generation, specifying the previously defined configurations. You can disable the PUCCH in either SCS carrier by setting the corresponding `cfg.PUCCH{1}.Enable` or `cfg.PUCCH{2}.Enable` properties, respectively, to `false`.

```
cfg = nrULCarrierConfig( ...
    SCSCarriers={carrier1,carrier2}, ...
    BandwidthParts={bwp1,bwp2}, ...
    PUSCH={pusch}, ...
    PUCCH={pucch1,pucch2});
```

Version History

Introduced in R2021b

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrWaveformGenerator`

Objects
`nrULCarrierConfig` | `nrWavegenPUCCH0Config` | `nrWavegenPUCCH2Config` | `nrWavegenPUCCH3Config` | `nrWavegenPUCCH4Config`

nrWavegenPUCCH2Config

PUCCH format 2 configuration parameters for 5G waveform generation

Description

The `nrWavegenPUCCH2Config` object sets physical uplink control channel (PUCCH) configuration parameters for format 2, as defined in TS 38.211 Sections 6.3.2.1, 6.3.2.5, and 6.4.1.3.2 [1]. Use this object to set the `PUCCH` property of the `nrULCarrierConfig` object when you configure 5G uplink waveform generation.

The default `nrWavegenPUCCH2Config` object allocates a PUCCH format 2 in the first resource block (RB) of the bandwidth part (BWP) and the last OFDM symbol in the slot of 14 OFDM symbols. PUCCH format 2 transmission is in all slots and carries 10 uplink control information (UCI) bits.

Creation

Syntax

```
pucch = nrWavegenPUCCH2Config
pucch = nrWavegenPUCCH2Config(Name=Value)
```

Description

`pucch = nrWavegenPUCCH2Config` creates a default PUCCH format 2 configuration object for 5G waveform generation.

`pucch = nrWavegenPUCCH2Config(Name=Value)` sets properties on page 3-211 using one or more name-value arguments. For example, `Period=20` specifies 20 slots for the PUCCH allocation period.

Properties

`Enable` — Option to enable PUCCH configuration
`true` or `1` (default) | `false` or `0`

Option to enable this PUCCH configuration in 5G waveform generation, specified as one of these numeric or logical values.

- `1` (`true`) — Enable this PUCCH.
- `0` (`false`) — Disable this PUCCH.

Data Types: `double` | `logical`

`Label` — Name of PUCCH format 2 configuration
`'PUCCH format 2'` (default) | character vector | string scalar

Name of this PUCCH format 2 configuration, specified as a character vector or string scalar. Use this property to set a description to the configuration.

Data Types: `char` | `string`

`Power` — Power scaling of PUCCH in dB

`0` (default) | real-valued scalar

Power scaling of the PUCCH in dB, specified as a real-valued scalar. Use this property to scale the power of the configured PUCCH in the generated 5G waveform.

Data Types: `double`

`BandwidthPartID` — ID of BWP

`1` (default) | nonnegative integer

ID of the BWP containing the configured PUCCH, specified as a nonnegative integer. Use this property to associate this PUCCH configuration with one of the BWP configurations specified by the `BandwidthParts` property of the `nrULCarrierConfig` object.

Data Types: `double`

`SymbolAllocation` — OFDM symbol allocation of PUCCH

`[13 1]` (default) | `[]` | two-element vector of nonnegative integers

OFDM symbol allocation of the PUCCH within a slot, specified as `[]` or a two-element vector of nonnegative integers. The first element of this property represents the start of symbol allocation. The second element represents the number of consecutively allocated OFDM symbols and must equal `1` or `2`. To indicate that no OFDM symbols are allocated for the PUCCH, use `[]` or specify the second element of the vector as `0`.

Data Types: `double`

`SlotAllocation` — Slot allocation in PUCCH period

`[0:9]` (default) | nonnegative integer | row vector of nonnegative integers

Slot allocation in a PUCCH period, specified as a nonnegative integer or a row vector of nonnegative integers. This property specifies the slot positions of the PUCCH by using 0-based indexing and values that are less than the value of the `Period` property. The object ignores slot allocation values that are greater than the period.

Data Types: `double`

`Period` — PUCCH allocation period in slots

`10` (default) | nonnegative integer | `[]`

PUCCH allocation period in slots, specified as a nonnegative integer or `[]`. An empty period indicates no repetition.

Data Types: `double`

`PRBSet` — PRB allocation

`0` (default) | vector of integers from 0 to 274 | `[]`

Physical resource block (PRB) allocation of the PUCCH within the BWP, specified as a vector of integers from 0 to 274 or `[]`. To indicate that no resource blocks are allocated for the PUCCH, use `[]`.

Data Types: `double`

`FrequencyHopping` — Frequency hopping configuration for PUCCH
`'neither'` (default) | `'intraSlot'` | `'interSlot'`

Frequency hopping configuration for the PUCCH, specified as one of these values.

- `'neither'` — Indicates no frequency hopping
- `'intraSlot'` — Enables intraslot frequency hopping
- `'interSlot'` — Enables interslot frequency hopping

When you enable interslot frequency hopping, the slot number determines the starting resource block for the allocated OFDM symbols. For even-numbered slots, the resource block starts from the minimum resource block allocated in the `PRBSet` property. For odd-numbered slots, the resource block starts from the resource block provided in the `SecondHopStartPRB` property.

Data Types: `char` | `string`

`SecondHopStartPRB` — Starting PRB index of second hop relative to BWP
`1` (default) | integer from 0 to 274

Starting PRB index of the second hop relative to the BWP, specified as an integer from 0 to 274.

Data Types: `double`

`NID` — PUCCH scrambling identity
`[]` (default) | integer from 0 to 1023

PUCCH scrambling identity, specified as `[]` or an integer from 0 to 1023.

- If the higher-layer parameter *dataScramblingIdentityPUSCH* is configured, this property must be an integer from 0 to 1023.
- If the higher-layer parameter *dataScramblingIdentityPUSCH* is not configured, this property must be equal to the physical layer cell identity and be an integer from 0 to 1007.

When you specify this property as `[]`, the object sets the PUCCH scrambling identity to the physical layer cell identity specified by the `NCellID` property of the `nrULCarrierConfig` object.

Data Types: `double`

`RNTI` — Radio network temporary identifier
`1` (default) | integer from 0 to 65,535

Radio network temporary identifier of the user equipment (UE), specified as an integer from 0 to 65,535.

Data Types: `double`

`Coding` — PUCCH encoding of UCI
`true` or `1` (default) | `false` or `0`

PUCCH encoding of UCI, specified as one of these numeric or logical values.

- `1` (`true`) — Enable UCI encoding.
- `0` (`false`) — Disable UCI encoding.

Data Types: `double` | `logical`

`NumUCIBits` — Number of UCI bits

`10` (default) | integer from 0 to 1706

Number of UCI bits, specified as an integer from 0 to 1706. For no UCI transmission, set this property to `0`.

Data Types: `double`

`DataSourceUCI` — Source of UCI contents

`'PN9-ITU'` (default) | `'PN9'` | `'PN11'` | `'PN15'` | `'PN23'` | two-element cell array | binary-valued vector

Source of UCI contents, specified as one of these options.

- `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, or `'PN23'`
- Two-element cell array consisting of one of the character vectors from the previous list and a random numeric seed (for example, `{'PN9',7}`)
- Binary-valued vector

If you do not specify a random seed, the object initializes all shift registers with an active state.

Dependencies

To enable this property, set the `NumUCIBits` property to a positive integer.

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

`NID0` — DM-RS scrambling identity

`[]` (default) | integer from 0 to 65,535

Demodulation reference signal (DM-RS) scrambling identity, specified as `[]` or an integer from 0 to 65,535.

- If the higher-layer parameter *scramblingID0* is configured, this property must be an integer from 0 to 65,535.
- If the higher-layer parameter *scramblingID0* is not configured, this property must be equal to the physical layer cell identity and be an integer from 0 to 1007.

When you specify this property as `[]`, the object sets the DM-RS scrambling identity to the physical layer cell identity specified by the `NCellID` property of the `nrULCarrierConfig` object.

Data Types: `double`

`DMRSPower` — Power scaling of PUCCH DM-RS in dB

`0` (default) | real-valued scalar

Power scaling of the PUCCH DM-RS in dB, specified as a real-valued scalar. Use this property to scale the power of the PUCCH DM-RS in the generated 5G waveform. This scaling is additional to the PUCCH-wide power scaling specified by the `Power` property.

Data Types: `double`

Examples

Configure Multiple PUCCH for 5G Uplink Waveform Generation

Create two SCS carrier configuration objects with mixed numerologies.

```
carrier1 = nrSCSCarrierConfig(SubcarrierSpacing=15);
carrier2 = nrSCSCarrierConfig(SubcarrierSpacing=30);
```

Create two BWP configuration objects, one for each of the SCS carriers.

```
bwp1 = nrWavegenBWPConfig(BandwidthPartID=1,SubcarrierSpacing=15);
bwp2 = nrWavegenBWPConfig(BandwidthPartID=2,SubcarrierSpacing=30);
```

Create a PUCCH format 0 configuration object for the first SCS carrier and a PUCCH format 1 configuration object for the second SCS carrier. For configuring PUCCH format 2, 3, or 4, use the `nrWavegenPUCCH2Config`, `nrWavegenPUCCH3Config`, or `nrWavegenPUCCH4Config` configuration objects, respectively. By default, each PUCCH configuration object enables the PUCCH on their respective carriers.

```
pucch1 = nrWavegenPUCCH0Config( ...
    BandwidthPartID=1,SlotAllocation=0:9,PRBSet=2,DataSourceUCI='PN9');
pucch2 = nrWavegenPUCCH1Config( ...
    BandwidthPartID=2,SlotAllocation=0:9,PRBSet=2,DataSourceUCI='PN9');
```

Create a PUSCH configuration object such that the PUSCH does not overlap with the previously configured PUCCH in the generated waveform.

```
pusch = nrWavegenPUSCHConfig(BandwidthPartID=1, ...
    SymbolAllocation=[0,8],PRBSet=10:51);
```

Create an uplink carrier configuration object for 5G waveform generation, specifying the previously defined configurations. You can disable the PUCCH in either SCS carrier by setting the corresponding `cfg.PUCCH{1}.Enable` or `cfg.PUCCH{2}.Enable` properties, respectively, to `false`.

```
cfg = nrULCarrierConfig( ...
    SCSCarriers={carrier1,carrier2}, ...
    BandwidthParts={bwp1,bwp2}, ...
    PUSCH={pusch}, ...
    PUCCH={pucch1,pucch2});
```

Version History

Introduced in R2021b

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions
nrWaveformGenerator

Objects
nrULCarrierConfig | nrWavegenPUCCH0Config | nrWavegenPUCCH1Config |
nrWavegenPUCCH3Config | nrWavegenPUCCH4Config

nrWavegenPUCCH3Config

PUCCH format 3 configuration parameters for 5G waveform generation

Description

The `nrWavegenPUCCH3Config` object sets physical uplink control channel (PUCCH) configuration parameters for format 3, as defined in TS 38.211 Sections 6.3.2.1, 6.3.2.2, 6.3.2.6, and 6.4.1.3.3 [1]. Use this object to set the `PUCCH` property of the `nrULCarrierConfig` object when you configure 5G uplink waveform generation.

The default `nrWavegenPUCCH3Config` object allocates a PUCCH format 3 in the first resource block (RB) of the bandwidth part (BWP) and in all OFDM symbols in a slot. PUCCH format 3 transmission is in all slots and carries 10 uplink control information (UCI) part 1 bits and no UCI part 2 bits.

Creation

Syntax

```
pucch = nrWavegenPUCCH3Config
pucch = nrWavegenPUCCH3Config(Name=Value)
```

Description

`pucch = nrWavegenPUCCH3Config` creates a default PUCCH format 3 configuration object for 5G waveform generation.

`pucch = nrWavegenPUCCH3Config(Name=Value)` sets properties on page 3-217 using one or more name-value arguments. For example, `Period=20` specifies 20 slots for the PUCCH allocation period.

Properties

Channel Configuration

`Enable` — Option to enable PUCCH configuration
`true` or `1` (default) | `false` or `0`

Option to enable this PUCCH configuration in 5G waveform generation, specified as one of these numeric or logical values.

- `1` (`true`) — Enable this PUCCH.
- `0` (`false`) — Disable this PUCCH.

Data Types: `double` | `logical`

`Label` — Name of PUCCH format 3 configuration
`'PUCCH format 3'` (default) | character vector | string scalar

Name of this PUCCH format 3 configuration, specified as a character vector or string scalar. Use this property to set a description to the configuration.

Data Types: char | string

Power — Power scaling of PUCCH in dB
0 (default) | real-valued scalar

Power scaling of the PUCCH in dB, specified as a real-valued scalar. Use this property to scale the power of the configured PUCCH in the generated 5G waveform.

Data Types: double

BandwidthPartID — ID of BWP
1 (default) | nonnegative integer

ID of the BWP containing the configured PUCCH, specified as a nonnegative integer. Use this property to associate this PUCCH configuration with one of the BWP configurations specified by the BandwidthParts property of the nrULCarrierConfig object.

Data Types: double

Modulation — Modulation scheme
'QPSK' (default) | 'pi/2-BPSK'

Modulation scheme, specified as 'QPSK' or 'pi/2-BPSK'.

Modulation Scheme	Number of Bits Per Symbol
'pi/2-BPSK'	1
'QPSK'	2

Data Types: char | string

SymbolAllocation — OFDM symbol allocation of PUCCH
[0 14] (default) | [] | two-element vector of nonnegative integers

OFDM symbol allocation of the PUCCH within a slot, specified as [] or a two-element vector of nonnegative integers. The first element of this property represents the start of symbol allocation. The second element represents the number of consecutively allocated OFDM symbols and must be in the range [4, 14]. To indicate that no OFDM symbols are allocated for the PUCCH, use [] or specify the second element of the vector as 0.

Data Types: double

SlotAllocation — Slot allocation in PUCCH period
[0:9] (default) | nonnegative integer | row vector of nonnegative integers

Slot allocation in a PUCCH period, specified as a nonnegative integer or a row vector of nonnegative integers. This property specifies the slot positions of the PUCCH by using 0-based indexing and values that are less than the value of the Period property. The object ignores slot allocation values that are greater than the period.

Data Types: double

Period — PUCCH allocation period in slots
10 (default) | nonnegative integer | []

PUCCH allocation period in slots, specified as a nonnegative integer or `[]`. An empty period indicates no repetition.

Data Types: `double`

`PRBSet` — PRB allocation
`0` (default) | vector of integers from 0 to 274 | `[]`

Physical resource block (PRB) allocation of the PUCCH within the BWP, specified as a vector of integers from 0 to 274 or `[]`. To indicate that no resource blocks are allocated for the PUCCH, use `[]`.

Data Types: `double`

`FrequencyHopping` — Frequency hopping configuration for PUCCH
`'neither'` (default) | `'intraSlot'` | `'interSlot'`

Frequency hopping configuration for the PUCCH, specified as one of these values.

- `'neither'` — Indicates no frequency hopping
- `'intraSlot'` — Enables intraslot frequency hopping
- `'interSlot'` — Enables interslot frequency hopping

When you enable interslot frequency hopping, the slot number determines the starting resource block for the allocated OFDM symbols. For even-numbered slots, the resource block starts from the minimum resource block allocated in the `PRBSet` property. For odd-numbered slots, the resource block starts from the resource block provided in the `SecondHopStartPRB` property.

Data Types: `char` | `string`

`SecondHopStartPRB` — Starting PRB index of second hop relative to BWP
`1` (default) | integer from 0 to 274

Starting PRB index of the second hop relative to the BWP, specified as an integer from 0 to 274.

Data Types: `double`

`GroupHopping` — Group hopping configuration for PUCCH
`'neither'` (default) | `'enable'` | `'disable'`

Group hopping configuration for the PUCCH, specified as one of these values.

- `'neither'` — Disables group hopping and sequence hopping
- `'enable'` — Enables group hopping and disables sequence hopping
- `'disable'` — Disables group hopping and enables sequence hopping

This property is the higher-layer parameter *pucch-GroupHopping*.

Data Types: `char` | `string`

`HoppingID` — Hopping identity of PUCCH
`[]` (default) | integer from 0 to 1023

Hopping identity of the PUCCH, specified as `[]` or an integer from 0 to 1023.

- If the higher-layer parameter *hoppingId* is configured, this property must be an integer from 0 to 1023.

- If the higher-layer parameter *hoppingId* is not configured, this property must be equal to the physical layer cell identity and be an integer from 0 to 1007.

When you specify this property as `[]`, the object sets the PUCCH hopping identity to the physical layer cell identity specified by the `NCellID` property of the `nrULCarrierConfig` object.

Data Types: `double`

`NID` — PUCCH scrambling identity
`[]` (default) | integer from 0 to 1023

PUCCH scrambling identity, specified as `[]` or an integer from 0 to 1023.

- If the higher-layer parameter *dataScramblingIdentityPUSCH* is configured, this property must be an integer from 0 to 1023.
- If the higher-layer parameter *dataScramblingIdentityPUSCH* is not configured, this property must be equal to the physical layer cell identity and be an integer from 0 to 1007.

When you specify this property as `[]`, the object sets the PUCCH scrambling identity to the physical layer cell identity specified by the `NCellID` property of the `nrULCarrierConfig` object.

Data Types: `double`

`RNTI` — Radio network temporary identifier
`1` (default) | integer from 0 to 65,535

Radio network temporary identifier of the user equipment (UE), specified as an integer from 0 to 65,535.

Data Types: `double`

`Coding` — PUCCH encoding of UCI
`true` or `1` (default) | `false` or `0`

PUCCH encoding of UCI, specified as one of these numeric or logical values.

- `1` (`true`) — Enable UCI encoding.
- `0` (`false`) — Disable UCI encoding.

Data Types: `double` | `logical`

UCI and DM-RS Configuration

`TargetCodeRate` — Target code rate
`0.15` (default) | numeric scalar between 0 and 1

Target code rate, specified as a numeric scalar between 0 and 1.

Dependencies

To enable this property, set the `Coding` property to `1` and the `NumUCIBits` and `NumUCI2Bits` properties to a positive integer.

Data Types: `double`

`NumUCIBits` — Number of UCI part 1 bits
`10` (default) | integer from 0 to 1706

Number of UCI part 1 bits, specified as an integer from 0 to 1706. For no UCI transmission, set this property to `0`.

Data Types: `double`

`DataSourceUCI` — Source of UCI part 1 contents
`'PN9-ITU'` (default) | `'PN9'` | `'PN11'` | `'PN15'` | `'PN23'` | two-element cell array | binary-valued vector

Source of UCI part 1 contents, specified as one of these options.

- `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, or `'PN23'`
- Two-element cell array consisting of one of the character vectors from the previous list and a random numeric seed (for example, `{'PN9',7}`)
- Binary-valued vector

If you do not specify a random seed, the object initializes all shift registers with an active state.

Dependencies

To enable this property, set the `NumUCIBits` property to a positive integer.

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

`NumUCI2Bits` — Number of UCI part 2 bits
`0` (default) | integer from 0 to 1706

Number of UCI part 2 bits, specified as an integer from 0 to 1706. For no UCI transmission, set this property to `0`.

Data Types: `double`

`DataSourceUCI2` — Source of UCI part 2 contents
`'PN9-ITU'` (default) | `'PN9'` | `'PN11'` | `'PN15'` | `'PN23'` | two-element cell array | binary-valued vector

Source of UCI part 2 contents, specified as one of these options.

- `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, or `'PN23'`
- Two-element cell array consisting of one of the character vectors from the previous list and a random numeric seed (for example, `{'PN9',7}`)
- Binary-valued vector

If you do not specify a random seed, the object initializes all shift registers with an active state.

Dependencies

To enable this property, set the `NumUCI2Bits` property to a positive integer.

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

`AdditionalDMRS` — Option to enable additional DM-RS
`false` or `0` (default) | `true` or `1`

Option to enable additional demodulation reference signal (DM-RS), provided by the higher-layer parameter *additionalDMRS*, specified as one of these numeric or logical values.

- `0` (`false`) — Disable additional DM-RS. When the number of the PUCCH OFDM symbols is greater than 9, two DM-RS symbols are present.
- `1` (`true`) — Enable additional DM-RS. When the number of the PUCCH OFDM symbols is greater than 9, four DM-RS symbols are present.

Data Types: `double` | `logical`

`DMRSPower` — Power scaling of PUCCH DM-RS in dB
`0` (default) | real-valued scalar

Power scaling of the PUCCH DM-RS in dB, specified as a real-valued scalar. Use this property to scale the power of the PUCCH DM-RS in the generated 5G waveform. This scaling is additional to the PUCCH-wide power scaling specified by the `Power` property.

Data Types: `double`

Examples

Configure Multiple PUCCH for 5G Uplink Waveform Generation

Create two SCS carrier configuration objects with mixed numerologies.

```
carrier1 = nrSCSCarrierConfig(SubcarrierSpacing=15);
carrier2 = nrSCSCarrierConfig(SubcarrierSpacing=30);
```

Create two BWP configuration objects, one for each of the SCS carriers.

```
bwp1 = nrWavegenBWPConfig(BandwidthPartID=1,SubcarrierSpacing=15);
bwp2 = nrWavegenBWPConfig(BandwidthPartID=2,SubcarrierSpacing=30);
```

Create a PUCCH format 0 configuration object for the first SCS carrier and a PUCCH format 1 configuration object for the second SCS carrier. For configuring PUCCH format 2, 3, or 4, use the `nrWavegenPUCCH2Config`, `nrWavegenPUCCH3Config`, or `nrWavegenPUCCH4Config` configuration objects, respectively. By default, each PUCCH configuration object enables the PUCCH on their respective carriers.

```
pucch1 = nrWavegenPUCCH0Config( ...
    BandwidthPartID=1,SlotAllocation=0:9,PRBSet=2,DataSourceUCI='PN9');
pucch2 = nrWavegenPUCCH1Config( ...
    BandwidthPartID=2,SlotAllocation=0:9,PRBSet=2,DataSourceUCI='PN9');
```

Create a PUSCH configuration object such that the PUSCH does not overlap with the previously configured PUCCH in the generated waveform.

```
pusch = nrWavegenPUSCHConfig(BandwidthPartID=1, ...
    SymbolAllocation=[0,8],PRBSet=10:51);
```

Create an uplink carrier configuration object for 5G waveform generation, specifying the previously defined configurations. You can disable the PUCCH in either SCS carrier by setting the corresponding `cfg.PUCCH{1}.Enable` or `cfg.PUCCH{2}.Enable` properties, respectively, to `false`.

```
cfg = nrULCarrierConfig( ...
    SCSCarriers={carrier1,carrier2}, ...
    BandwidthParts={bwp1,bwp2}, ...
    PUSCH={pusch}, ...
    PUCCH={pucch1,pucch2});
```

Version History

Introduced in R2021b

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`nrWaveformGenerator`

Objects

`nrULCarrierConfig` | `nrWavegenPUCCH0Config` | `nrWavegenPUCCH1Config` | `nrWavegenPUCCH2Config` | `nrWavegenPUCCH4Config`

nrWavegenPUCCH4Config

PUCCH format 4 configuration parameters for 5G waveform generation

Description

The `nrWavegenPUCCH4Config` object sets physical uplink control channel (PUCCH) configuration parameters for format 4, as defined in TS 38.211 Sections 6.3.2.1, 6.3.2.2, 6.3.2.6, and 6.4.1.3.3 [1]. Use this object to set the `PUCCH` property of the `nrULCarrierConfig` object when you configure 5G uplink waveform generation.

The default `nrWavegenPUCCH4Config` object allocates a PUCCH format 4 in the first resource block (RB) of the bandwidth part (BWP) and in all OFDM symbols in a slot. PUCCH format 4 transmission is in all slots and carries 10 uplink control information (UCI) part 1 bits and no UCI part 2 bits.

Creation

Syntax

```
pucch = nrWavegenPUCCH4Config
pucch = nrWavegenPUCCH4Config(Name=Value)
```

Description

`pucch = nrWavegenPUCCH4Config` creates a default PUCCH format 4 configuration object for 5G waveform generation.

`pucch = nrWavegenPUCCH4Config(Name=Value)` sets properties on page 3-217 using one or more name-value arguments. For example, `Period=20` specifies 20 slots for the PUCCH allocation period.

Properties

Channel Configuration

`Enable` — Option to enable PUCCH configuration
`true` or `1` (default) | `false` or `0`

Option to enable this PUCCH configuration in 5G waveform generation, specified as one of these numeric or logical values.

- `1` (`true`) — Enable this PUCCH.
- `0` (`false`) — Disable this PUCCH.

Data Types: `double` | `logical`

`Label` — Name of PUCCH format 4 configuration
`'PUCCH format 4'` (default) | character vector | string scalar

Name of this PUCCH format 4 configuration, specified as a character vector or string scalar. Use this property to set a description to the configuration.

Data Types: `char` | `string`

`Power` — Power scaling of PUCCH in dB
`0` (default) | real-valued scalar

Power scaling of the PUCCH in dB, specified as a real-valued scalar. Use this property to scale the power of the configured PUCCH in the generated 5G waveform.

Data Types: `double`

`BandwidthPartID` — ID of BWP
`1` (default) | nonnegative integer

ID of the BWP containing the configured PUCCH, specified as a nonnegative integer. Use this property to associate this PUCCH configuration with one of the BWP configurations specified by the `BandwidthParts` property of the `nrULCarrierConfig` object.

Data Types: `double`

`Modulation` — Modulation scheme
`'QPSK'` (default) | `'pi/2-BPSK'`

Modulation scheme, specified as `'QPSK'` or `'pi/2-BPSK'`.

Modulation Scheme	Number of Bits Per Symbol
`'pi/2-BPSK'`	1
`'QPSK'`	2

Data Types: `char` | `string`

`SymbolAllocation` — OFDM symbol allocation of PUCCH
`[0 14]` (default) | `[]` | two-element vector of nonnegative integers

OFDM symbol allocation of the PUCCH within a slot, specified as `[]` or a two-element vector of nonnegative integers. The first element of this property represents the start of symbol allocation. The second element represents the number of consecutively allocated OFDM symbols and must be in the range [4, 14]. To indicate that no OFDM symbols are allocated for the PUCCH, use `[]` or specify the second element of the vector as `0`.

Data Types: `double`

`SlotAllocation` — Slot allocation in PUCCH period
`[0:9]` (default) | nonnegative integer | row vector of nonnegative integers

Slot allocation in a PUCCH period, specified as a nonnegative integer or a row vector of nonnegative integers. This property specifies the slot positions of the PUCCH by using 0-based indexing and values that are less than the value of the `Period` property. The object ignores slot allocation values that are greater than the period.

Data Types: `double`

`Period` — PUCCH allocation period in slots
`10` (default) | nonnegative integer | `[]`

PUCCH allocation period in slots, specified as a nonnegative integer or []. An empty period indicates no repetition.

Data Types: double

PRBSet — PRB allocation
0 (default) | integer from 0 to 274 | []

Physical resource block (PRB) allocation of the PUCCH within the BWP, specified as an integer from 0 to 274 or []. To indicate that no resource blocks are allocated for the PUCCH, use [].

Data Types: double

FrequencyHopping — Frequency hopping configuration for PUCCH
'neither' (default) | 'intraSlot' | 'interSlot'

Frequency hopping configuration for the PUCCH, specified as one of these values.

- 'neither' — Indicates no frequency hopping
- 'intraSlot' — Enables intraslot frequency hopping
- 'interSlot' — Enables interslot frequency hopping

When you enable interslot frequency hopping, the slot number determines the starting resource block for the allocated OFDM symbols. For even-numbered slots, the resource block starts from the minimum resource block allocated in the PRBSet property. For odd-numbered slots, the resource block starts from the resource block provided in the SecondHopStartPRB property.

Data Types: char | string

SecondHopStartPRB — Starting PRB index of second hop relative to BWP
1 (default) | integer from 0 to 274

Starting PRB index of the second hop relative to the BWP, specified as an integer from 0 to 274.

Data Types: double

GroupHopping — Group hopping configuration for PUCCH
'neither' (default) | 'enable' | 'disable'

Group hopping configuration for the PUCCH, specified as one of these values.

- 'neither' — Disables group hopping and sequence hopping
- 'enable' — Enables group hopping and disables sequence hopping
- 'disable' — Disables group hopping and enables sequence hopping

This property is the higher-layer parameter *pucch-GroupHopping*.

Data Types: char | string

HoppingID — Hopping identity of PUCCH
[] (default) | integer from 0 to 1023

Hopping identity of the PUCCH, specified as [] or an integer from 0 to 1023.

- If the higher-layer parameter *hoppingId* is configured, this property must be an integer from 0 to 1023.

- If the higher-layer parameter *hoppingId* is not configured, this property must be equal to the physical layer cell identity and be an integer from 0 to 1007.

When you specify this property as `[]`, the object sets the PUCCH hopping identity to the physical layer cell identity specified by the `NCellID` property of the `nrULCarrierConfig` object.

Data Types: `double`

`OCCI` — Orthogonal cover code index
`0` (default) | integer from 0 to 3

Orthogonal cover code index (OCCI), specified as an integer from 0 to 3. The OCCI value must be less than the `SpreadingFactor` property.

Data Types: `double`

`NID` — PUCCH scrambling identity
`[]` (default) | integer from 0 to 1023

PUCCH scrambling identity, specified as `[]` or an integer from 0 to 1023.

- If the higher-layer parameter *dataScramblingIdentityPUSCH* is configured, this property must be an integer from 0 to 1023.
- If the higher-layer parameter *dataScramblingIdentityPUSCH* is not configured, this property must be equal to the physical layer cell identity and be an integer from 0 to 1007.

When you specify this property as `[]`, the object sets the PUCCH scrambling identity to the physical layer cell identity specified by the `NCellID` property of the `nrULCarrierConfig` object.

Data Types: `double`

`RNTI` — Radio network temporary identifier
`1` (default) | integer from 0 to 65,535

Radio network temporary identifier of the user equipment (UE), specified as an integer from 0 to 65,535.

Data Types: `double`

`Coding` — PUCCH encoding of UCI
`true` or `1` (default) | `false` or `0`

PUCCH encoding of UCI, specified as one of these numeric or logical values.

- `1` (`true`) — Enable UCI encoding.
- `0` (`false`) — Disable UCI encoding.

Data Types: `double` | `logical`

UCI and DM-RS Configuration

`TargetCodeRate` — Target code rate
`0.15` (default) | numeric scalar between 0 and 1

Target code rate, specified as a numeric scalar between 0 and 1.

Dependencies

To enable this property, set the `Coding` property to `1` and the `NumUCIBits` and `NumUCI2Bits` properties to a positive integer.

Data Types: `double`

`NumUCIBits` — Number of UCI part 1 bits
`10` (default) | integer from 0 to 1706

Number of UCI part 1 bits, specified as an integer from 0 to 1706. For no UCI transmission, set this property to `0`.

Data Types: `double`

`DataSourceUCI` — Source of UCI part 1 contents
`'PN9-ITU'` (default) | `'PN9'` | `'PN11'` | `'PN15'` | `'PN23'` | two-element cell array | binary-valued vector

Source of UCI part 1 contents, specified as one of these options.

- `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, or `'PN23'`
- Two-element cell array consisting of one of the character vectors from the previous list and a random numeric seed (for example, `{'PN9',7}`)
- Binary-valued vector

If you do not specify a random seed, the object initializes all shift registers with an active state.

Dependencies

To enable this property, set the `NumUCIBits` property to a positive integer.

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

`NumUCI2Bits` — Number of UCI part 2 bits
`0` (default) | integer from 0 to 1706

Number of UCI part 2 bits, specified as an integer from 0 to 1706. For no UCI transmission, set this property to `0`.

Data Types: `double`

`DataSourceUCI2` — Source of UCI part 2 contents
`'PN9-ITU'` (default) | `'PN9'` | `'PN11'` | `'PN15'` | `'PN23'` | two-element cell array | binary-valued vector

Source of UCI part 2 contents, specified as one of these options.

- `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, or `'PN23'`
- Two-element cell array consisting of one of the character vectors from the previous list and a random numeric seed (for example, `{'PN9',7}`)
- Binary-valued vector

If you do not specify a random seed, the object initializes all shift registers with an active state.

Dependencies

To enable this property, set the `NumUCI2Bits` property to a positive integer.

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

`AdditionalDMRS` — Option to enable additional DM-RS
`false` or `0` (default) | `true` or `1`

Option to enable additional demodulation reference signal (DM-RS), provided by the higher-layer parameter *additionalDMRS*, specified as one of these numeric or logical values.

- `0` (`false`) — Disable additional DM-RS. When the number of the PUCCH OFDM symbols is greater than 9, two DM-RS symbols are present.
- `1` (`true`) — Enable additional DM-RS. When the number of the PUCCH OFDM symbols is greater than 9, four DM-RS symbols are present.

Data Types: `double` | `logical`

`DMRSPower` — Power scaling of PUCCH DM-RS in dB
`0` (default) | real-valued scalar

Power scaling of the PUCCH DM-RS in dB, specified as a real-valued scalar. Use this property to scale the power of the PUCCH DM-RS in the generated 5G waveform. This scaling is additional to the PUCCH-wide power scaling specified by the `Power` property.

Data Types: `double`

Examples

Configure Multiple PUCCH for 5G Uplink Waveform Generation

Create two SCS carrier configuration objects with mixed numerologies.

```
carrier1 = nrSCSCarrierConfig(SubcarrierSpacing=15);
carrier2 = nrSCSCarrierConfig(SubcarrierSpacing=30);
```

Create two BWP configuration objects, one for each of the SCS carriers.

```
bwp1 = nrWavegenBWPConfig(BandwidthPartID=1,SubcarrierSpacing=15);
bwp2 = nrWavegenBWPConfig(BandwidthPartID=2,SubcarrierSpacing=30);
```

Create a PUCCH format 0 configuration object for the first SCS carrier and a PUCCH format 1 configuration object for the second SCS carrier. For configuring PUCCH format 2, 3, or 4, use the `nrWavegenPUCCH2Config`, `nrWavegenPUCCH3Config`, or `nrWavegenPUCCH4Config` configuration objects, respectively. By default, each PUCCH configuration object enables the PUCCH on their respective carriers.

```
pucch1 = nrWavegenPUCCH0Config( ...
    BandwidthPartID=1,SlotAllocation=0:9,PRBSet=2,DataSourceUCI='PN9');
pucch2 = nrWavegenPUCCH1Config( ...
    BandwidthPartID=2,SlotAllocation=0:9,PRBSet=2,DataSourceUCI='PN9');
```

Create a PUSCH configuration object such that the PUSCH does not overlap with the previously configured PUCCH in the generated waveform.

```
pusch = nrWavegenPUSCHConfig(BandwidthPartID=1, ...
    SymbolAllocation=[0,8],PRBSet=10:51);
```

Create an uplink carrier configuration object for 5G waveform generation, specifying the previously defined configurations. You can disable the PUCCH in either SCS carrier by setting the corresponding `cfg.PUCCH{1}.Enable` or `cfg.PUCCH{2}.Enable` properties, respectively, to `false`.

```
cfg = nrULCarrierConfig( ...
    SCSCarriers={carrier1,carrier2}, ...
    BandwidthParts={bwp1,bwp2}, ...
    PUSCH={pusch}, ...
    PUCCH={pucch1,pucch2});
```

Version History

Introduced in R2021b

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`nrWaveformGenerator`

Objects

`nrULCarrierConfig` | `nrWavegenPUCCH0Config` | `nrWavegenPUCCH1Config` | `nrWavegenPUCCH2Config` | `nrWavegenPUCCH3Config`

nrWavegenPUSCHConfig

PUSCH configuration parameters for 5G waveform generation

Description

The `nrWavegenPUSCHConfig` object sets physical uplink shared channel (PUSCH) configuration parameters, as defined in TS 38.211 Sections 6.3.1, 6.4.1.1, and 6.4.1.2 [1]. Use this object to set the `PUSCH` property of the `nrULCarrierConfig` object when configuring 5G uplink waveform generation.

This object defines several properties of the PUSCH, including the modulation scheme, layer mapping, transform precoding, target code rate, and time-domain and frequency-domain allocation. The object also contains properties of the associated physical reference signals, such as the demodulation reference signal (DM-RS) and the phase tracking reference signal (PT-RS).

The default `nrWavegenPUSCHConfig` object configures a single-layer PUSCH with cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM), mapping type A, QPSK modulation, a resource allocation of 52 resource blocks and 14 OFDM symbols in a slot, transmission in all slots, and a single-symbol DM-RS type 1. The default object also disables frequency hopping, transform precoding, the PT-RS, and the uplink control information (UCI). This configuration corresponds to a full resource allocation with respect to the default `nrWavegenBWPConfig` object.

Creation

Syntax

```
pusch = nrWavegenPUSCHConfig
pusch = nrWavegenPUSCHConfig(Name,Value)
```

Description

`pusch = nrWavegenPUSCHConfig` creates a default PUSCH configuration object for 5G waveform generation.

`pusch = nrWavegenPUSCHConfig(Name,Value)` specifies properties on page 3-231 using one or more name-value arguments. Enclose each property in quotes. For example, `'NumLayers',4` specifies 4 transmission layers.

Properties

Channel Configuration

`Enable` — Enable PUSCH
`1` or `true` (default) | `0` or `false`

Enable the PUSCH in 5G waveform generation, specified as one of these values.

- 1 (true) — Enable the PUSCH.
- 0 (false) — Disable the PUSCH.

Data Types: double | logical

Label — Name of PUSCH configuration
'PUSCH1' (default) | character array | string scalar

Name of the PUSCH configuration, specified as a character array or string scalar. Use this property to set a description to the PUSCH configuration.

Data Types: char | string

Power — Power scaling of PUSCH in dB
0 (default) | real-valued scalar

Power scaling of the PUSCH in dB, specified as a real-valued scalar. Use this property to scale the power of the PUSCH in the generated 5G waveform.

Data Types: double

BandwidthPartID — ID of BWP
1 (default) | nonnegative integer

ID of the bandwidth part (BWP) containing the configured PUSCH, specified as a nonnegative integer. Use this property to associate this PUSCH configuration with one of the BWP configurations specified by the BandwidthParts property of the nrULCarrierConfig object.

Data Types: double

Modulation — Modulation scheme
'QPSK' (default) | 'pi/2-BPSK' | '16QAM' | '64QAM' | '256QAM' | string scalar

Modulation scheme, specified as 'QPSK', 'pi/2-BPSK', '16QAM', '64QAM', or '256QAM', a string scalar, or a character array.

Modulation Scheme	Number of Bits Per Symbol
'pi/2-BPSK'	1
'QPSK'	2
'16QAM'	4
'64QAM'	6
'256QAM'	8

Data Types: char | string

NumLayers — Number of transmission layers
1 (default) | 2 | 3 | 4

Number of transmission layers, specified as 1, 2, 3, or 4.

Data Types: double

MappingType — Mapping type
'A' (default) | 'B'

Mapping type of the physical shared channel, specified as 'A' or 'B'.

Data Types: char | string

SymbolAllocation — OFDM symbol allocation
[0 14] (default) | two-element vector of nonnegative integers

OFDM symbol allocation of the physical shared channel, specified as a two-element vector of nonnegative integers. The first element of this property represents the start of symbol allocation (0-based). The second element represents the number of allocated OFDM symbols.

When you set this property to [] or the second element of the vector to 0, no symbol is allocated for the channel.

Data Types: double

SlotAllocation — Slot allocation in PUSCH period
[0:9] (default) | nonnegative integer | row vector of nonnegative integers

Slot allocation in a PUSCH period, specified as a nonnegative integer or a row vector of nonnegative integers. This property specifies the slot positions of the PUSCH by using 0-based indexing and values that are less than the value of the Period property. The object ignores slot allocation values that are greater than the period.

Data Types: double

Period — PUSCH allocation period in slots
10 (default) | nonnegative integer | []

PUSCH allocation period in slots, specified as a nonnegative integer or []. An empty period indicates no repetition.

Data Types: double

PRBSet — PRB allocation
[0:51] (default) | vector of integers from 0 to 274

Physical resource block (PRB) allocation of the PUSCH within the BWP, specified as a vector of integers from 0 to 274.

Data Types: double

TransformPrecoding — Transform precoding
0 or false (default) | 1 or true

Transform precoding, specified as one of these values.

- 0 (false) — Disable transform precoding. The waveform type is cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM).
- 1 (true) — Enable transform precoding. The waveform type is discrete Fourier transform spread orthogonal frequency division multiplexing (DFT-s-OFDM).

Data Types: double | logical

TransmissionScheme — PUSCH transmission scheme
'nonCodebook' (default) | 'codebook'

PUSCH transmission scheme, specified as `'nonCodebook'` or `'codebook'`.

Data Types: `char` | `string`

`TPMI` — Transmitted precoding matrix indicator
`0` (default) | integer from 0 to 27

Transmitted precoding matrix indicator, specified as an integer from 0 to 27.

Dependencies

To enable this property, set the `TransmissionScheme` property to `'codebook'`.

Data Types: `double`

`FrequencyHopping` — Frequency hopping
`'neither'` (default) | `'intraSlot'` | `'interSlot'`

Frequency hopping for the physical uplink shared channel, specified as `'neither'`, `'intraSlot'`, or `'interSlot'`.

Data Types: `char` | `string`

`SecondHopStartPRB` — Starting PRB index of second hop
`1` (default) | integer from 0 to 274

Starting PRB index of the second hop relative to the BWP, specified as an integer from 0 to 274.

Dependencies

This property is applicable only when `FrequencyHopping` is set to `'intraSlot'`, or `'interSlot'`.

Data Types: `double`

`NID` — PUSCH scrambling identity
`[]` (default) | integer from 0 to 1023

PUSCH scrambling identity, specified as `[]` or an integer from 0 to 1023.

- If the higher layer parameter *dataScramblingIdentityPUSCH* is configured, `NID` must be an integer from 0 to 1023.
- If the higher layer parameter *dataScramblingIdentityPUSCH* is not configured, `NID` must be an integer from 0 to 1007.

When you specify this property as `[]`, the object sets the PUSCH scrambling identity to the physical layer cell identity, specified by the `NCellID` property of the carrier.

Data Types: `double`

`RNTI` — Radio network temporary identifier
`1` (default) | integer from 0 to 65,535

Radio network temporary identifier of the user equipment (UE), specified as an integer from 0 to 65,535.

Data Types: `double`

`NRAPID` — Random access preamble index
`[]` (default) | integer from 0 to 63

Random access preamble index, specified as one of these values.

- `[]` — Use this value to specify that the scrambling initialization does not consider *msgA* on PUSCH.
- Integer from 0 to 63 — Use this value to initialize the scrambling sequence for *msgA* on PUSCH, as defined in Release 16 of TS 38.211 Section 6.3.1.1.

Data Types: `double`

`Coding` — UL-SCH encoding of transport blocks
`1` or `true` (default) | `0` or `false`

Uplink shared channel (UL-SCH) encoding of the transport blocks, specified as one of these values.

- `1` (`true`) — Enable transport block encoding.
- `0` (`false`) — Disable transport block encoding.

Data Types: `double` | `logical`

`TargetCodeRate` — Target code rate
`0.5137` (default) | numeric scalar between 0 and 1

Target code rate, specified as a numeric scalar between 0 and 1. The default value corresponds to 526/1024.

Dependencies

To enable this property, set the `Coding` property to `1` (`true`).

Data Types: `double`

`XOverhead` — Rate matching overhead
`0` (default) | `6` | `12` | `18`

Rate matching overhead, specified as `0`, `6`, `12`, or `18`.

Data Types: `double`

`RVSequence` — Redundancy version sequence
`[0 2 3 1]` (default) | nonnegative integer | vector of nonnegative integers

Redundancy version sequence, specified as a nonnegative integer or a vector of nonnegative integers.

Dependencies

To enable this property, set the `Coding` property to `1` (`true`).

Data Types: `double`

`DataSource` — Source of contents for transport blocks
`'PN9-ITU'` (default) | `'PN9'` | `'PN11'` | `'PN15'` | `'PN23'` | two-element cell array | binary-valued vector

Source of contents for the transport blocks, specified as one of these options.

- `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, or `'PN23'`

- Two-element cell array consisting of one of the character vectors from the previous list and a random numeric seed (for example, {'PN9',7})
- Binary-valued vector

If you do not specify a random seed, all shift registers are initialized with an active state.

Data Types: double | cell | string | char

UCI on PUSCH Configuration

EnableACK — HARQ-ACK for UCI on PUSCH

0 or false (default) | 1 or true

Hybrid automatic repeat request acknowledgment (HARQ-ACK) for UCI on the PUSCH, specified as one of these values.

- 0 (false) — Disable HARQ-ACK transmission.
- 1 (true) — Enable HARQ-ACK transmission.

Dependencies

To enable this property, set the Coding property to 1 (true).

Data Types: logical | double

NumACKBits — Number of HARQ-ACK bits in UCI on PUSCH

10 (default) | integer from 0 to 1706

Number of HARQ-ACK bits in the UCI on the PUSCH, specified as an integer from 0 to 1706.

Dependencies

To enable this property, set the Coding and EnableACK properties to 1 (true).

Data Types: double

BetaOffsetACK — Beta offset factor of HARQ-ACK

20 (default) | positive real-valued scalar

Beta offset factor of the HARQ-ACK, specified as a positive real-valued scalar. This property determines the number of resources for multiplexing HARQ-ACK. The nominal value is one of the entries from the Table 9.3-1 of TS 38.213.

Dependencies

To enable this property, set the Coding and EnableACK properties to 1 (true) and the NumACKBits property to a positive value.

Data Types: double

DataSourceACK — Source of HARQ-ACK contents

'PN9-ITU' (default) | 'PN9' | 'PN11' | 'PN15' | 'PN23' | two-element cell array | binary-valued vector

Source of HARQ-ACK contents, specified as one of these options.

- 'PN9-ITU', 'PN9', 'PN11', 'PN15', or 'PN23'

- Two-element cell array consisting of one of the character vectors from the previous list and a random numeric seed (for example, `{'PN9',7}`)
- Binary-valued vector

If you do not specify a random seed, all shift registers are initialized with an active state.

Dependencies

To enable this property, set the `Coding` and `EnableACK` properties to `1` (`true`) and the `NumACKBits` property to a positive value.

Data Types: `double`

`EnableCSI1` — CSI part 1 for UCI on PUSCH

`0` or `false` (default) | `1` or `true`

Channel-state information (CSI) part 1 for the UCI on the PUSCH, specified as one of these options.

- `0` (`false`) — Disable the CSI part 1 transmission on the PUSCH.
- `1` (`true`) — Enable the CSI part 1 transmission on the PUSCH.

Dependencies

To enable this property, set the `Coding` property to `1` (`true`).

Data Types: `logical` | `double`

`NumCSI1Bits` — Number of CSI part 1 bits for UCI on PUSCH

`10` (default) | integer from 0 to 1706

Number of CSI part 1 bits for the UCI on the PUSCH, specified as an integer from 0 to 1706.

Dependencies

To enable this property, set the `Coding` and `EnableCSI1` properties to `1` (`true`).

Data Types: `double`

`BetaOffsetCSI1` — Beta offset factor of CSI part 1

`6.25` (default) | positive real-valued scalar

Beta offset factor of CSI part 1, specified as a positive real-valued scalar. This property determines the number of resources for multiplexing CSI part 1. The nominal value is one of the entries from the Table 9.3-2 of TS 38.213.

Dependencies

To enable this property, set the `Coding` and `EnableCSI1` properties to `1` (`true`) and the `NumCSI1Bits` property to a positive value.

Data Types: `double`

`DataSourceCSI1` — Source of CSI part 1 contents

`'PN9-ITU'` (default) | `'PN9'` | `'PN11'` | `'PN15'` | `'PN23'` | two-element cell array | binary-valued vector

Source of CSI part 1 contents, specified as one of these options.

- 'PN9-ITU', 'PN9', 'PN11', 'PN15', or 'PN23'
- Two-element cell array consisting of one of the character vectors from the previous list and a random numeric seed (for example, {'PN9',7}).
- Binary-valued vector

If you do not specify a random seed, all shift registers are initialized with an active state.

Dependencies

To enable this property, set the Coding and EnableCSI1 properties to 1 (true) and the NumCSI1Bits property to a positive value.

Data Types: double

EnableCSI2 — CSI part 2 for UCI on PUSCH

0 or false (default) | 1 or true

CSI part 2 for the UCI on the PUSCH, specified as one of these values.

- 0 (false) — Disable the CSI part 2 transmission on the PUSCH.
- 1 (true) — Enable the CSI part 2 transmission on the PUSCH.

Dependencies

To enable this property, set the Coding and EnableCSI1 properties to 1 (true) and the NumCSI1Bits property to a positive value.

Data Types: logical | double

NumCSI2Bits — Number of CSI part 2 bits for UCI on PUSCH

10 (default) | integer from 0 to 1706

Number of CSI part 2 bits for the UCI on the PUSCH, specified as an integer from 0 to 1706.

Dependencies

To enable this property, set the Coding, EnableCSI1, and EnableCSI2 properties to 1 (true) and the NumCSI1Bits property to a positive value.

Data Types: double

BetaOffsetCSI2 — Beta offset factor of CSI part 2

6.25 (default) | positive real-valued scalar

Beta offset factor of the CSI part 2, specified as a positive real-valued scalar. This property determines the number of resources for multiplexing CSI part 2. The nominal value is one of the entries from the Table 9.3-2 of TS 38.213.

Dependencies

To enable this property, set the Coding, EnableCSI1, and EnableCSI2 properties to 1 (true) and the NumCSI1Bits and NumCSI2Bits properties to a positive value.

Data Types: double

DataSourceCSI2 — Source of CSI part 2 contents

'PN9-ITU' (default) | 'PN9' | 'PN11' | 'PN15' | 'PN23' | two-element cell array | binary-valued vector

Source of CSI part 2 contents, specified as one of these options.

- 'PN9-ITU', 'PN9', 'PN11', 'PN15', or 'PN23'
- Two-element cell array consisting of one of the character vectors from the previous list and a random numeric seed (for example, {'PN9',7}).
- Binary-valued vector

If you do not specify a random seed, all shift registers are initialized with an active state.

Dependencies

To enable this property, set the Coding, EnableCSI1, and EnableCSI2 properties to 1 (true) and the NumCSI1Bits and NumCSI2Bits properties to a positive value.

Data Types: double

EnableCGUCI — CG-UCI transmission on PUSCH
0 or false (default) | 1 or true

Configured grant transmissions in UCI (CG-UCI) on the PUSCH, specified as one of these values.

- 0 (false) — Disable CG-UCI transmission on the PUSCH.
- 1 (true) — Enable CG-UCI transmission on the PUSCH.

Dependencies

To enable this property, set the Coding property to 1 (true).

Data Types: logical | double

NumCGUCIBits — Number of CG-UCI bits on PUSCH
7 (default) | 0 | integer from 7 to 1706

Number of CG-UCI bits on PUSCH, specified as 0 or an integer from 7 to 1706. Setting this property to 0 disables the CG-UCI.

Dependencies

To enable this property, set the Coding and EnableCGUCI properties to 1 (true).

Data Types: double

BetaOffsetCGUCI — Beta offset factor of CG-UCI
20 (default) | positive real-valued scalar

Beta offset factor of the CG-UCI, specified as a positive real-valued scalar. This property determines the number of resources for multiplexing CG-UCI.

Dependencies

To enable this property, set the Coding and EnableCGUCI properties to 1 (true) and the NumCGUCIBits property to a positive value. If the EnableACK property is also set to 1 (true) and the NumACKBits property is set to a positive value, the object uses the BetaOffsetACK property value as the beta offset factor for the CG-UCI instead.

Data Types: double

DataSourceCGUCI — Source of CG-UCI contents
'PN9-ITU' (default) | 'PN9' | 'PN11' | 'PN15' | 'PN23' | two-element cell array | binary-valued vector

Source of CG-UCI, specified as one of these options.

- 'PN9-ITU', 'PN9', 'PN11', 'PN15', or 'PN23'
- Two-element cell array consisting of one of the character vectors from the previous list and a random numeric seed (for example, {'PN9',7}).
- Binary-valued vector

If you do not specify a random seed, all shift registers are initialized with an active state.

Dependencies

To enable this property, set the Coding and EnableCGUCI properties to 1 (true) and the NumCGUCIBits property to a positive value.

Data Types: double

EnableULSCH — UL-SCH for UCI on PUSCH
1 or true (default) | 0 or false

UL-SCH transmission on the slots used for the UCI on the PUSCH, specified as one of these values.

- 1 (true) — Enable UL-SCH and UCI multiplexing on the PUSCH.
- 0 (false) — Disable UL-SCH and UCI multiplexing on the PUSCH.

Dependencies

To enable this property, set the Coding property to 1 (true) and one of these property combinations:

- Set the EnableACK property to 1 (true) and the NumACKBits property to a positive value.
- Set the EnableCSI1 property to 1 (true) and the NumCSI1Bits property to a positive value.
- Set the EnableCGUCI property to 1 (true) and the NumCGUCIBits property to a positive value.

Data Types: logical | double

UCIScaling — Scaling factor
1 (default) | scalar between 0 and 1

Scaling factor to limit the number of the resource elements allocated for the UCI on the PUSCH, specified as a scalar between 0 and 1. The nominal value is 0.5, 0.65, 0.8, or 1.

Data Types: double

Reference Signals Configuration

DMRS — PUSCH DM-RS configuration parameters
default nrPUSCHDMRSConfig object (default) | nrPUSCHDMRSConfig object

PUSCH DM-RS configuration parameters, specified as an nrPUSCHDMRSConfig configuration object.

DMRSPower — Power scaling of PUSCH DM-RS in dB
0 (default) | real-valued scalar

Power scaling of the PUSCH DM-RS in dB, specified as a real-valued scalar. Use this property to scale the power of the PUSCH DM-RS in the generated 5G waveform. This scaling is additional to the PUSCH-wide power scaling specified by the `Power` property.

Data Types: `double`

EnablePTRS — Enable PT-RS
`0` or `false` (default) | `1` or `true`

Enable the PT-RS, specified as one of these values.

- `0` (`false`) — Disable the PT-RS configuration.
- `1` (`true`) — Enable the PT-RS configuration.

Data Types: `double` | `logical`

PTRS — PUSCH PT-RS configuration parameters
default `nrPUSCHPTRSConfig` object (default) | `nrPUSCHPTRSConfig` object

PUSCH PT-RS configuration, specified as an `nrPUSCHPTRSConfig` configuration object. This property relates to the PT-RS configuration and contains all properties of the specified `nrPUSCHPTRSConfig` object.

Dependencies

To enable this property, set the `EnablePTRS` property to `1`.

PTRSPower — Power scaling of PUSCH PT-RS in dB
`0` (default) | real-valued scalar

Power scaling of the PUSCH PT-RS in dB, specified as a real-valued scalar. Use this property to scale the power of the PUSCH PT-RS in the generated 5G waveform. This scaling is additional to the PUSCH-wide power scaling specified by the `Power` property. The object ignores this property value when you set the `TransformPrecoding` property to `1` (`true`).

Dependencies

To enable this property, set the `EnablePTRS` property to `1` (`true`).

Data Types: `double`

Examples

Configure PUSCH for 5G Uplink Waveform Generation

Create a PUSCH configuration object for 5G waveform generation with the specified property values.

```
pusch = nrWavegenPUSCHConfig( ...
    'BandwidthPartID',0, ...
    'Modulation','16QAM', ...
    'TargetCodeRate',658/1024, ...
    'SymbolAllocation',[0 7], ...
    'SlotAllocation',[0 2], ...
    'Period',3, ...
    'PRBSet',[0:20], ...
    'EnablePTRS',true);
```

Create an uplink carrier configuration object, specifying the previously defined PUSCH configuration.

```
cfg = nrULCarrierConfig('PUSCH',{pusch});
```

Configure Multiple PUSCH for 5G Uplink Waveform Generation

Create two SCS carrier configuration objects with mixed numerologies.

```
carrier1 = nrSCSCarrierConfig('SubcarrierSpacing',15);
carrier2 = nrSCSCarrierConfig('SubcarrierSpacing',30);
```

Create two BWP configuration objects, one for each of the SCS carriers.

```
bwp1 = nrWavegenBWPConfig('BandwidthPartID',0,'SubcarrierSpacing',15);
bwp2 = nrWavegenBWPConfig('BandwidthPartID',1,'SubcarrierSpacing',30);
```

Create two PUSCH configuration objects for 5G waveform generation, specifying a unique UE and one of the BWP configurations for each PUSCH.

```
pusch1 = nrWavegenPUSCHConfig('RNTI',1,'BandwidthPartID',0,'Modulation','QPSK');
pusch2 = nrWavegenPUSCHConfig('RNTI',2,'BandwidthPartID',1,'Modulation','16QAM');
```

Create an uplink carrier configuration object, specifying the previously defined configurations.

```
cfg = nrULCarrierConfig( ...
    'SCSCarriers',{carrier1,carrier2}, ...
    'BandwidthParts',{bwp1,bwp2}, ...
    'PUSCH',{pusch1,pusch2});
```

Version History

Introduced in R2021a

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*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrWaveformGenerator`

Objects
`nrULCarrierConfig`

nrWavegenSRSConfig

SRS configuration parameters for 5G waveform generation

Description

The `nrWavegenSRSConfig` object sets sounding reference signal (SRS) configuration parameters, as defined in TS 38.211 Section 6.4.1.4 [1]. Use this object to set the `SRS` property of the `nrULCarrierConfig` object when configuring 5G uplink waveform generation.

The default `nrWavegenSRSConfig` object specifies a single-port, single-symbol, narrowband SRS configuration without frequency hopping (`BHop` ≥ `BSRS`) and places the SRS at the end of the slot.

Creation

Syntax

```
srs = nrWavegenSRSConfig
srs = nrWavegenSRSConfig(Name,Value)
```

Description

`srs = nrWavegenSRSConfig` creates a default SRS configuration object for 5G waveform generation.

`srs = nrWavegenSRSConfig(Name,Value)` specifies properties on page 3-244 using one or more name-value arguments. Enclose each property in quotes. For example, `'NumSRSPorts',2,'NumSRSSymbols',4` specifies a two-port SRS transmission of 4 OFDM symbols.

Properties

`Enable` — Enable SRS
`1` or `true` (default) | `0` or `false`

Enable the SRS in 5G waveform generation, specified as one of these values.

- `1` (`true`) — Enable the SRS.
- `0` (`false`) — Disable the SRS.

Data Types: `double` | `logical`

`Label` — Name of SRS configuration
`'SRS1'` (default) | character array | string scalar

Name of the SRS configuration, specified as a character array or string scalar. Use this property to set a description to the SRS configuration.

Data Types: `char` | `string`

Power — Power scaling of SRS in dB
0 (default) | real scalar

Power scaling of the SRS in dB, specified as a real scalar. Use this property to scale the power of the SRS in the generated 5G waveform.

Data Types: double

BandwidthPartID — ID of BWP
1 (default) | nonnegative integer

ID of the bandwidth part (BWP) containing the configured SRS, specified as a nonnegative integer. Use this property to associate this SRS configuration with one of the BWP configurations specified by the BandwidthParts property of the nrULCarrierConfig object.

Data Types: double

NumSRSPorts — Number of SRS antenna ports
1 (default) | 2 | 4

Number of SRS antenna ports, specified as 1, 2, or 4.

Data Types: double

NumSRSSymbols — Number of OFDM symbols
1 (default) | 2 | 4 | 8 | 12

Number of OFDM symbols allocated to the SRS in a slot, specified as 1, 2, 4, 8, or 12. Valid property values depend on the SRSPositioning property.

- If you set the SRSPositioning property to 0 (false), specify this property as 1, 2, or 4.
- If you set the SRSPositioning property to 1 (true), specify this property as 1, 2, 4, 8, or 12. For valid configurations of this property and the KTC property, see TS 38.211 Table 6.4.1.4.3-2. Alternatively, type nrSRSConfig.SubcarrierOffsetTable at the command line to display this table.

Data Types: double

SymbolStart — 0-based index of first OFDM symbol
13 (default) | integer from 0 to 13

0-based index of the first OFDM symbol in the SRS within a slot, specified as one of these options:

- Integer from 0 to 13 — Use this option for normal cyclic prefix.
- Integer from 0 to 11 — Use this option for extended cyclic prefix.

For the SRS symbols and index generation, set the cyclic prefix of the carrier by using the CyclicPrefix property of the BWP configuration object specified by the BandwidthPartID property.

Data Types: double

SlotAllocation — Slot allocation in SRS period
[0:9] (default) | nonnegative integer | row vector of nonnegative integers

Slot allocation in an SRS period, specified as a nonnegative integer or a row vector of nonnegative integers. This property specifies the slot positions of the SRS by using 0-based indexing and values that are less than the value of the `Period` property. The object ignores slot allocation values that are greater than the period. Each element of the vector corresponds to an SRS resource.

Data Types: `char`

`Period` — SRS allocation period in slots
`10` (default) | nonnegative integer | `[]`

SRS allocation period in slots, specified as a nonnegative integer or `[]`. An empty period indicates aperiodic SRS resource type (no repetition), as defined in TS 38.211 Section 6.4.1.4.3.

Data Types: `double`

`KTC` — Transmission comb number
`2` (default) | `4` | `8`

Transmission comb number, in subcarriers, specified as `2`, `4`, or `8`. The object allocates the SRS sequence every `KTC` number of subcarriers. Valid property values depend on the `SRSPositioning` property.

- If you set the `SRSPositioning` property to `0` (`false`), specify this property as `2` or `4`.
- If you set the `SRSPositioning` property to `1` (`true`), specify this property as `2`, `4`, or `8`. For valid configurations of this property and the `NumSRSSymbols` property, see TS 38.211 Table 6.4.1.4.3-2. Alternatively, type `nrSRSConfig.SubcarrierOffsetTable` at the command line to display this table.

Data Types: `double`

`KBarTC` — Transmission comb offset
`0` (default) | integer from 0 to (`KTC` – 1)

Transmission comb offset, in subcarriers, specified as an integer from 0 to (`KTC` – 1). This property specifies a frequency shift within the comb.

Data Types: `double`

`CyclicShift` — Cyclic shift offset
`0` (default) | integer from 0 to 11

Cyclic shift offset, specified as an integer from 0 to 11. This property determines the cyclic shift applied to the SRS sequence for each antenna port. This property corresponds to parameter n_{SRS}^{cs} in TS 38.211 Section 6.4.1.4.2.

Set the cyclic shift offset in relation to the transmission comb property, `KTC`:

- If you set `KTC` to `2`, set `CyclicShift` to an integer from 0 to 7.
- If you set `KTC` to `4`, set `CyclicShift` to an integer from 0 to 11.
- If you set `KTC` to `8`, set `CyclicShift` to an integer from 0 to 5.

For multiport SRS transmissions, consecutive cyclic shift numbers are used for each port, modulo 6, 8, or 12, depending on the `KTC` property.

Data Types: `double`

`FrequencyStart` — Frequency-domain offset
`0` (default) | integer from 0 to 271

Frequency-domain offset of the SRS, in terms of a physical resource block (PRB) index relative to the carrier, specified as an integer from 0 to 271. `FrequencyStart` is analogous to parameter n_{shift} from TS 38.211 Section 6.4.1.4.3.

This property, the additional circular frequency-domain offset property `NRRC`, and the bandwidth configuration parameters in TS 38.211 Table 6.4.1.4.3-1 determine the actual frequency-domain location of the SRS. For more information, see "NR SRS Configuration".

Data Types: `double`

NRRC — Additional circular frequency-domain offset
`0` (default) | integer from 0 to 67

Additional circular frequency-domain offset of the SRS, as a multiple of 4 PRBs, specified as an integer from 0 to 67.

This property, the frequency domain offset property `FrequencyStart`, and the bandwidth configuration parameters in TS 38.211 Table 6.4.1.4.3-1 determine the actual frequency-domain location of the SRS. For more information, see "NR SRS Configuration".

Data Types: `double`

CSRS — Row index of bandwidth configuration table
`0` (default) | integer from 0 to 63

Row index of the bandwidth configuration table from TS 38.211 Table 6.4.1.4.3-1, specified as an integer from 0 to 63. Use this property with the `BSRS` property to control the bandwidth allocated to the SRS and the frequency hopping pattern. Increasing the `CSRS` value increases the SRS bandwidth. The default value of `0` results in a bandwidth of 4 PRBs.

Data Types: `double`

BSRS — Column index of bandwidth configuration table
`0` (default) | integer from 0 to 3

Column index of the bandwidth configuration table from TS 38.211 Table 6.4.1.4.3-1, specified as an integer from 0 to 3. Use this property with the `CSRS` property to control the bandwidth allocated to the SRS and the frequency hopping pattern. Increasing the `BSRS` value decreases the SRS bandwidth.

Data Types: `double`

BHop — Frequency hopping index
`0` (default) | integer from 0 to 3

Frequency hopping index, specified as an integer from 0 to 3. Setting this property to a value greater than or equal to the column index of the bandwidth configuration table property, `BSRS`, disables frequency hopping. Increasing the `BHop` value decreases the hopping bandwidth.

Data Types: `double`

`Repetition` — Repetition factor of OFDM symbols
`1` (default) | `2` | `4` | `8` | `12`

Repetition factor of OFDM symbols, specified as `1`, `2`, `4`, `8`, or `12`.

- When frequency hopping is enabled, `Repetition` specifies the number of consecutive OFDM symbols in a slot occupied by the SRS in the same frequency resource. Set `Repetition` such that `Repetition` ≤ `NumSRSSymbols`.
- When frequency hopping is disabled, this property is ignored.

Data Types: `double`

`GroupSeqHopping` — Type of SRS symbol hopping
`'neither'` (default) | `'groupHopping'` | `'sequenceHopping'`

Type of SRS symbol hopping, specified as `'neither'`, `'groupHopping'`, or `'sequenceHopping'`. When either group or sequence hopping is enabled, the group or sequence hopping numbers per OFDM symbol in the SRS transmission are based on a pseudo-random binary sequence (PRBS). Set the scrambling identity for the PRBS by using the `NSRSID` property.

Data Types: `char` | `string`

`NSRSID` — SRS scrambling identity
`0` (default) | integer from 0 to 65,535

SRS scrambling identity, specified as an integer from 0 to 65,535.

- When you set the `GroupSeqHopping` property to `'neither'`, this property determines the group number.
- When you set the `GroupSeqHopping` property to `'groupHopping'` or `'sequenceHopping'`, this property initializes the PRBS.

Data Types: `double`

`SRSPositioning` — SRS for user positioning
`0` or `false` (default) | `1` or `true`

SRS for user positioning, as defined in Release 16 of TS 38.211 Section 6.4.1.4, specified as one of these values.

- `0` (`false`) — Disable SRS for user positioning. This option corresponds to the higher-layer parameter *SRS-Resource*.
- `1` (`true`) — Enable SRS for user positioning. This option corresponds to the higher-layer parameter *SRS-PosResource-r16*.

This property affects the valid range of the `NumSRSSymbols` and `KTC` properties.

Data Types: `logical` | `double`

Examples

Configure SRS for 5G Uplink Waveform Generation

Create two SCS carrier configuration objects with mixed numerologies and custom numbers of resource blocks.

```
carriers = {
    nrSCSCarrierConfig('SubcarrierSpacing',15,'NStartGrid',10,'NSizeGrid',100), ...
    nrSCSCarrierConfig('SubcarrierSpacing',30,'NStartGrid',0,'NSizeGrid',70)};
```

Create two BWP configuration objects, one for each of the SCS carriers.

```
bwp = {
    nrWavegenBWPConfig('BandwidthPartID',1,'SubcarrierSpacing',15,'NStartBWP',10,'NSizeBWP',80),
    nrWavegenBWPConfig('BandwidthPartID',2,'SubcarrierSpacing',30,'NStartBWP',0,'NSizeBWP',60)};
```

Create two SRS configuration objects, one for each of the carriers, with the specified properties. In the first SRS configuration, frequency hopping is enabled. In the second SRS configuration, frequency hopping is disabled.

```
srs = {
    nrWavegenSRSConfig('BandwidthPartID',1,'NumSRSPorts',2,'NumSRSSymbols',4,'SymbolStart',8,'CSI
    nrWavegenSRSConfig('BandwidthPartID',2,'FrequencyStart',4)};
```

Create a PUSCH configuration object such that the PUSCH does not overlap with the previously configured SRS in the generated waveform.

```
pusch = {
    nrWavegenPUSCHConfig('BandwidthPartID',1,'SymbolAllocation',[0 8],'PRBSet',(10:51))};
```

Create an uplink carrier configuration object, specifying the previously defined configurations.

```
cfg = nrULCarrierConfig( ...
    'SCSCarriers',carriers, ...
    'BandwidthParts',bwp, ...
    'SRS',srs, ...
    'PUSCH',pusch);
```

Version History

Introduced in R2021a

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`nrWaveformGenerator`

Objects

`nrULCarrierConfig`

nrWavegenSSBurstConfig

SS burst configuration parameters for 5G waveform generation

Description

The `nrWavegenSSBurstConfig` object sets synchronization signal (SS) burst configuration parameters. Use this object to set the `SSBurst` property of the `nrDLCarrierConfig` object when configuring 5G downlink waveform generation.

This object defines the subcarrier spacing (SCS), time-domain and frequency-domain allocations, power, and payload of the SS burst.

The default `nrWavegenSSBurstConfig` object configures the SS burst with four active SS blocks and a periodicity of 20 ms, corresponding to the initial cell selection. The default configuration also specifies the SS burst to carry the master information block (MIB) and places the SS burst in the center of a carrier with 15 kHz subcarrier spacing (block pattern Case A). To update the frequency location of the SS burst, set the `NCRBSSB` and `KSSB` object properties to a nonempty value.

Creation

Syntax

```
ssb = nrWavegenSSBurstConfig
ssb = nrWavegenSSBurstConfig(Name,Value)
```

Description

`ssb = nrWavegenSSBurstConfig` creates a default SS burst configuration object for 5G waveform generation.

`ssb = nrWavegenSSBurstConfig(Name,Value)` sets properties on page 3-250 using one or more name-value arguments. Enclose each property name in quotes. For example, `'BlockPattern','Case B'` specifies block pattern Case B.

Properties

`Enable` — Enable SS burst
`1` (default) | `0`

Enable the SS burst in 5G waveform generation, specified as one of these values.

- `1` — Enable the SS burst.
- `0` — Disable the SS burst.

Data Types: `double` | `logical`

`Power` — Power scaling of SS burst in dB
`0` (default) | real scalar | row vector of real numbers

Power scaling of the SS burst in dB, specified as one of these values.

- Real scalar — The object scales the power of the SS burst in the generated 5G waveform by applying the same power level to each SS block in the burst.
- Row vector of real numbers — The object scales the power of the SS burst in the generated 5G waveform by applying different power levels to the SS blocks in the burst. The vector must have the same length as the block transmission bitmap, specified by the `TransmittedBlocks` property. Use this option to model the power level effect of beam sweeping on different blocks in the burst.

Data Types: `double`

`BlockPattern` — Block pattern of SS burst
`'Case A'` (default) | `'Case B'` | `'Case C'` | `'Case D'` | `'Case E'`

Block pattern of the SS burst, specified as one of these values corresponding to the patterns from TS 38.213 Section 4.1 [1].

- `'Case A'` — Use this value for frequency range 1 (FR1) and 15 kHz SCS.
- `'Case B'` or `'Case C'` — Use either of these values for FR1 and 30 kHz SCS.
- `'Case D'` — Use this value for frequency range 2 (FR2) and 120 kHz SCS.
- `'Case E'` — Use this value for FR2 and 240 kHz SCS.

Data Types: `char` | `string`

`TransmittedBlocks` — Block transmission bitmap
`[1 1 1 1]` (default) | 4-bit, 8-bit, or 64-bit binary vector

Block transmission bitmap in a 5 ms half-frame burst, specified as a 4-bit or 8-bit binary vector for FR1 or a 64-bit binary vector for FR2. This vector specifies which SS blocks are active in the SS burst.

Data Types: `double`

`Period` — Period of SS burst in ms
`20` (default) | `5` | `10` | `40` | `80` | `160`

Period of the SS burst in ms, specified as `5`, `10`, `20`, `40`, `80`, or `160`.

Data Types: `double`

`NCRBSSB` — Frequency offset from point A
`[]` (default) | integer from 0 to 2199

Frequency offset from point A, specified as `[]` or an integer from 0 to 2199. Point A is the center of subcarrier 0 in the common resource block 0 (CRB 0). This property specifies the frequency offset of the SS burst in resource blocks (RBs) relative to point A.

- For block pattern Case A, Case B and Case C, the unit of this property is expressed in terms of 15 kHz SCS.
- For block pattern Case D and Case E, the unit of this property is expressed in terms of 60 kHz SCS.

When `NCRBSSB` is an empty vector, `[]`, the SS burst is positioned in the center of the carrier with the SCS corresponding to the block pattern specified by the `BlockPattern` property.

Data Types: `double`

KSSB — Subcarrier offset
`0` (default) | integer from 0 to 23

Subcarrier offset, specified as one these options.

- For FR1:
  - If `SubcarrierSpacingCommon` is `15`, specify this property as an integer from 0 to 11.
  - If `SubcarrierSpacingCommon` is `30`, specify this property as an integer from 0 to 23.

  In both cases, units are in terms of 15 kHz SCS. For block pattern Case B, `KSSB` must be even.
- For FR2, specify this property as an integer from 0 to 11. For block pattern Case D, `KSSB` must be even. For block pattern Case E, `KSSB` must be a multiple of 4. Units are in terms of SCS equal to the `SubcarrierSpacingCommon` property value.

The object increases the frequency offset of the SS burst from point A by `KSSB` subcarriers.

Dependencies

To enable this property, set the `NCRBSSB` property to a value other than `[]`.

Data Types: `double`

`DataSource` — Source of SS burst payload
`'MIB'` (default) | `'PN9-ITU'` | `'PN9'` | `'PN11''PN15'` | `'PN23'` | two-element cell array | 24-bit binary vector

Source of the SS burst payload, specified as one of these options.

- `'MIB'`, `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, or `'PN23'`
- Two-element cell array consisting of one of the character vectors from the previous list and a random numeric seed (for example, `{'PN9',7}`)
- 24-bit binary vector

If you do not specify a random seed, all shift registers are initialized with an active state.

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

`DMRSTypeAPosition` — Position of first DM-RS symbol
`2` (default) | `3`

Position of the first demodulation reference signal (DM-RS) symbol in the physical downlink shared channel (PDSCH) system information block type 1 (SIB1), specified as `2` or `3`.

Dependencies

To enable this property, set the `DataSource` property to `'MIB'`.

Data Types: `double`

`CellBarred` — Cell barring
`0` (default) | `1`

Cell barring, specified as `0` or `1`. When `CellBarred` is set to `1`, the cell enables the user equipment (UE) to camp on the cell.

Dependencies

To enable this property, set the `DataSource` property to `'MIB'`.

Data Types: `double` | `logical`

`IntraFreqReselection` — Enable intrafrequency reselection
`0` (default) | `1`

Enable intrafrequency reselection, specified as one of these values.

- `0` — Disable intrafrequency reselection.
- `1` — Enable intrafrequency reselection of the same frequency cells.

Dependencies

To enable this property, set the `DataSource` property to `'MIB'`.

Data Types: `double` | `logical`

`PDCCHConfigSIB1` — Configuration type of PDCCH SIB1
`0` (default) | integer from 0 to 255

Configuration type of the physical downlink control channel (PDCCH) SIB1, specified as an integer from 0 to 255.

Dependencies

To enable this property, set the `DataSource` property to `'MIB'`.

Data Types: `double`

`SubcarrierSpacingCommon` — SIB1 SCS in kHz
`15` (default) | `30` | `60` | `120`

SIB1 SCS in kHz, specified as one of these values.

- `15` or `30` for FR1
- `60` or `120` for FR2

Dependencies

To enable this property, either set the `DataSource` property to `'MIB'` or set the `NCRBSSB` property to a value other than `[]` and the `BlockPattern` property to `'Case D'` or `'Case E'`.

Data Types: `double`

Examples

Configure SS Burst for 5G Downlink Waveform Generation

Create a downlink carrier configuration object with the specified property values.

```
cfgDL = nrDLCarrierConfig('FrequencyRange','FR2','ChannelBandwidth',100);
cfgDL.SCSCarriers{1} = nrSCSCarrierConfig('SubcarrierSpacing',120);
cfgDL.BandwidthParts{1} = nrWavegenBWPConfig('SubcarrierSpacing',120);
```

Create an SS burst configuration object for block pattern Case D, corresponding to 120 kHz SCS, and a block transmission bitmap for FR2.

```
ssb = nrWavegenSSBurstConfig('BlockPattern','Case D','TransmittedBlocks',ones(1,64));
```

Specify the SS burst frequency offset to be one third of the carrier size.

```
ssb.NCRBSSB = round(cfgDL.SCSCarriers{1}.NSizeGrid/3);
```

Specify the subcarrier offset of the SS burst, taking into account the specified block pattern. For block pattern Case D, the subcarrier offset value must be even.

```
ssb.KSSB = 2*4;
```

Specify the SS burst configuration for the downlink carrier configuration.

```
cfgDL.SSBurst = ssb;
```

Version History

Introduced in R2020b

Specify different power levels for each SS block in a burst

The `Power` property enables you to specify a different power level for each SS block in a burst. You can use this feature to model the power level effect of beam sweeping on different blocks in the burst.

Subcarrier offset range in SS burst configuration has changed

Behavior changed in R2021a

Starting in R2021a, the range of the `KSSB` property for FR1 depends on the value of the `SubcarrierSpacingCommon` property.

References

[1] 3GPP TS 38.213. “NR; Physical layer procedures for control.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`nrWaveformGenerator`

Objects

`nrDLCarrierConfig`

pcapReader

PCAP file reader of protocol packets

Description

The `pcapReader` object reads and decodes Ethernet and enhanced common public radio interface (eCPRI) protocol packets based on specific criteria. You can also plug in custom protocol decoders.

Creation

Syntax

```
pcap = pcapReader(filename)
pcap = pcapReader(filename,OutputTimestampFormat='seconds')
```

Description

`pcap = pcapReader(filename)` creates a PCAP file reader object to read protocol packets from the input PCAP file.

`pcap = pcapReader(filename,OutputTimestampFormat='seconds')` sets OutputTimestampFormat property to `seconds`.

Input Arguments

`filename` — Name of PCAP file
character vector | string scalar

Name of a PCAP file including the extension, specified as a character vector or a string scalar.

Example: `"ethernetSamplePackets.pcap"`

Data Types: `char` | `string`

Properties

`OutputTimestampFormat` — Output format for packet timestamp
`'microseconds'` (default) | `'seconds'` | `'datetime'`

Output format for the packet timestamp, specified as `'microseconds'`, `'seconds'`, or `'datetime'`. This value specifies the timestamp of the decoded protocol packet.

Data Types: `char` | `string`

`MajorVersion` — Major version of PCAP file format
nonnegative scalar

This property is read-only.

Major version of the PCAP file format, returned as a nonnegative scalar.

Data Types: `double`

`MinorVersion` — Minor version of PCAP file format
nonnegative scalar

This property is read-only.

Minor version of the PCAP file format, returned as a nonnegative scalar.

Data Types: `double`

`SnapLength` — Maximum length of packet in PCAP file
nonnegative scalar

This property is read-only.

Maximum length of the packet in the PCAP file, returned as a nonnegative scalar.

Data Types: `double`

`LinkType` — Link type in PCAP global header
nonnegative scalar

This property is read-only.

Link type in the PCAP global header, returned as a nonnegative scalar. For more information about this property, see Tcpdump/Libpcap Public Repository [1].

Data Types: `double`

`LinkName` — Name of link type given by PCAP file reader object
character vector

This property is read-only.

Name of the link type given by the PCAP file reader object, returned as a character vector.

Data Types: `char`

`NanoSecResolution` — Flag to indicate whether PCAP file has nanosecond resolution for packet timestamp
`1` or `true` | `0` or `false`

This property is read-only.

Flag to indicate whether the PCAP file has nanosecond resolution for the packet timestamp, returned as `1` (`true`) or `0` (`false`).

Data Types: `logical`

Object Functions

Specific to This Object

addLinkTypeDecoder	Add custom link layer protocol decoder to PCAP file reader
addUpperLayerDecoder	Add custom upper-layer protocol decoder to PCAP file reader
read	Read next protocol packet from PCAP file

readAll	Read all protocol packets from current position to end of PCAP file
reset	Reset position of PCAP file reader to first protocol packet of PCAP file

Examples

Read Ethernet Packets from PCAP File

Create a PCAP file reader object, specifying the name of a PCAP file.

```
pcapReaderObj = pcapReader('ethernetSamplePackets.pcap');
```

Read all of the packets from the PCAP file to the MATLAB® workspace.

```
decodedPackets = readAll(pcapReaderObj)
```

```
decodedPackets=1×35 struct array with fields:
    SNo
    Timestamp
    LinkType
    Protocol
    PacketLength
    Packet
    RawBytes
```

Read Ethernet Packets in Streaming Mode from PCAP File

Create a PCAP file reader object, specifying the name of a PCAP file and an output format for the packet timestamp.

```
pcapReaderObj = pcapReader('ethernetSamplePackets.pcap', ...
    OutputTimestampFormat='datetime');
```

Create a filter for the Ethernet source address and Ethernet type.

```
filterString = ['eth.SourceAddress == 44FB5A9710AC && ' ...
    'eth.Type == 2048'];
```

In streaming mode, read the Ethernet packets that match the specified filter to the MATLAB workspace.

```
for packetCount = 1:3
    ethPacket = read(pcapReaderObj,filterString)
end
```

```
ethPacket = struct with fields:
             SNo: 1
       Timestamp: 08-Feb-2021 03:27:18.043900
        LinkType: 1
        Protocol: 'eth'
    PacketLength: 171
          Packet: [1x1 struct]
        RawBytes: [1x0 double]
```

```
    TimestampSec: 1.6128e+09


ethPacket = struct with fields:
             SNo: 4
       Timestamp: 08-Feb-2021 03:27:19.098190
        LinkType: 1
        Protocol: 'eth'
    PacketLength: 120
          Packet: [1x1 struct]
        RawBytes: [1x0 double]
    TimestampSec: 1.6128e+09


ethPacket = struct with fields:
             SNo: 5
       Timestamp: 08-Feb-2021 03:27:20.145857
        LinkType: 1
        Protocol: 'eth'
    PacketLength: 171
          Packet: [1x1 struct]
        RawBytes: [1x0 double]
    TimestampSec: 1.6128e+09
```

Read eCPRI Packets from PCAP File

Create a PCAP file reader object, specifying the name of a PCAP file.

```
pcapReaderObj = pcapReader('ethernetSamplePackets.pcap');
```

Create a filter for the eCPRI packets, specifying the eCPRI message types.

```
filterString = ['ecpri.MessageType == IQData || ecpri.MessageType == BitSequence ' ...
    '|| ecpri.MessageType == RemoteReset'];
```

Read the eCPRI packets that match the specified filters to the MATLAB workspace.

```
ecpriFilteredFirstPacket = read(pcapReaderObj,filterString)
```

```
ecpriFilteredFirstPacket = struct with fields:
             SNo: 21
       Timestamp: 1.6128e+15
        LinkType: 1
        Protocol: 'eth'
    PacketLength: 64
          Packet: [1x1 struct]
        RawBytes: [1x0 double]
```

```
ecpriFilteredSecondPacket = read(pcapReaderObj,filterString)
```

```
ecpriFilteredSecondPacket = struct with fields:
             SNo: 22
       Timestamp: 1.6128e+15
        LinkType: 1
```

```
        Protocol: 'eth'
    PacketLength: 64
          Packet: [1x1 struct]
        RawBytes: [1x0 double]
```

```
ecpriFilteredRemainingPackets = readAll(pcapReaderObj,filterString)
```

```
ecpriFilteredRemainingPackets=1×5 struct array with fields:
    SNo
    Timestamp
    LinkType
    Protocol
    PacketLength
    Packet
    RawBytes
```

Reset the position of the PCAP file reader to the first packet of the PCAP file.

```
reset(pcapReaderObj);
```

Create a new filter on the same PCAP file, specifying the message type as in-phase and quadrature (IQ) data.

```
filterString = 'ecpri.MessageType == IQData';
```

Read the eCPRI packets that match the specified filter to the MATLAB workspace.

```
ecpriFilteredPackets = readAll(pcapReaderObj,filterString)
```

```
ecpriFilteredPackets = struct with fields:
             SNo: 21
       Timestamp: 1.6128e+15
        LinkType: 1
        Protocol: 'eth'
    PacketLength: 64
          Packet: [1x1 struct]
        RawBytes: [1x0 double]
```

Version History

Introduced in R2021b

References

[1] Group, The Tcpdump. "Tcpdump/Libpcap Public Repository." Accessed May 20, 2020. https://www.tcpdump.org.

[2] "Development/LibpcapFileFormat - The Wireshark Wiki." Accessed May 20, 2020. https://www.wireshark.org.

[3] "Common Public Radio Interface: eCPRI Interface Specification V1.2 " Accessed June 22, 2021.

See Also

Objects
pcapWriter | pcapngWriter | nrPCAPWriter

pcapWriter

PCAP file writer of protocol packets

Description

The `pcapWriter` object writes generated and recovered protocol packets to a packet capture (PCAP) file (`.pcap`).

You can write these packet types to a PCAP file:

- Generated and recovered 5G NR protocol packets
- Generated and recovered WLAN protocol packets (requires WLAN Toolbox)
- Generated and recovered Bluetooth low energy (LE) link layer (LL) packets (requires Bluetooth Toolbox)

Creation

Syntax

```
pcapObj = pcapWriter
pcapObj = pcapWriter(Name,Value)
```

Description

`pcapObj = pcapWriter` creates a default PCAP file writer object.

`pcapObj = pcapWriter(Name,Value)` sets properties on page 3-261 using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'ByteOrder','big-endian'` specifies the byte order as big-endian.

Properties

Note The `pcapWriter` object does not overwrite the existing PCAP or PCAPNG file. Each time when you create this object, specify a unique PCAP or PCAPNG file name.

`FileName` — Name of the PCAP file
`'capture'` (default) | character row vector | string scalar

Name of the PCAP file, specified as a character row vector or a string scalar.

Data Types: `char` | `string`

`ByteOrder` — Byte order
`'little-endian'` (default) | `'big-endian'`

Byte order, specified as `'little-endian'` or `'big-endian'`.

Data Types: char | string

Object Functions

Specific to This Object

write	Write protocol packet data to PCAP or PCAPNG file
writeGlobalHeader	Write global header to PCAP file

Examples

Write 5G NR Packet to PCAP File

Create a PCAP file writer object, specifying the name of the PCAP file. 5G NR packets do not have a valid link type. As per Tcpdump, if a valid link type is not present, specify the link type of SLL packet.

```
pcapObj = pcapWriter('FileName','sample');
linkType = 113;                                        % Link type of SLL packet
timestamp = 300;                                       % Timestamp
```

Write a global header to the PCAP file.

```
writeGlobalHeader(pcapObj,linkType);
```

The 5G New Radio (NR) packets are not directly supported by Wireshark. To enable Wireshark to parse 5G NR packets, add encapsulation and metadata to the 5G NR packet.

```
payload = [59; 205];                                   % MAC subPDU (contains truncated buffer s
radioType = 1;                                         % Frequency division duplexing
linkDir = 0;                                           % Uplink packet
rntiType = 3;                                          % Cell-RNTI
startString = [109; 97; 99; 45; 110; 114];             % Tag to indicate the start of NR MAC sig
payloadTag = 1;                                        % Payload tag for NR packets
signature = [startString; radioType; linkDir; rntiType];
macNRInfoPacket = [signature; payloadTag; payload];
```

Construct a user datagram protocol (UDP) header.

```
udpPacketLength = 8 + length(macNRInfoPacket);         % Length of header (8 bytes) and payload
udpHeader = [163; 76;                                  % Source port number
    39; 15;                                            % Destination port number
    fix(udpPacketLength/256); mod(udpPacketLength,256); % Total length of UDP packet
    0; 0];                                             % Checksum
```

Construct an IPv4 header.

```
ipPacketLength = 20 + udpPacketLength;                 % Length of header (20 bytes) and payloa
ipHeader = [69;                                        % Version of IP protocol and priority or
    0;                                                 % Type of service
    fix(ipPacketLength/256); mod(ipPacketLength,256);  % Total length of the IPv4 packet
    0; 1;                                              % Identification
    0; 0;                                              % Flags and fragmentation offset
    64;                                                % Time to live in seconds
    17;                                                % UDP protocol number
    0; 0;                                              % Header checksum
```

```
    127; 0; 0; 1;                                         % Source IP address
    127; 0; 0; 1];                                        % Destination IP address
```

Construct an SLL header.

```
sllHeader = [0; 0;                                        % Packet type
    3; 4;                                                 % Address resolution protocol hardware (A
    0; 0;                                                 % Link layer address length
    0; 0; 0; 0; 0; 0; 0; 0;                               % Link layer address
    8; 0];                                                % Protocol type
```

Construct 5G NR packet by adding encapsulation and metadata.

```
packet = [sllHeader; ipHeader; udpHeader; macNRInfoPacket];
```

Write the 5G NR packet to the PCAP file.

```
write(pcapObj,packet,timestamp);
```

Version History

Introduced in R2020b

References

[1] Group, The Tcpdump. “Tcpdump/Libpcap Public Repository.” Accessed May 20, 2020. https://www.tcpdump.org.

[2] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
`pcapngWriter` | `nrPCAPWriter` | `pcapReader`

pcapngWriter

PCAPNG file writer of protocol packets

Description

The `pcapngWriter` object writes generated and recovered protocol packets to a packet capture next generation (PCAPNG) file (`.pcapng`).

You can write these packet types to a PCAPNG file:

- Generated and recovered 5G NR protocol packets
- Generated and recovered WLAN protocol packets (requires WLAN Toolbox)
- Generated and recovered Bluetooth low energy (LE) link layer (LL) packets (requires Bluetooth Toolbox)

Creation

Syntax

```
pcapngObj = pcapngWriter
pcapngObj = pcapngWriter(Name,Value)
```

Description

`pcapngObj = pcapngWriter` creates a default PCAPNG file writer object.

`pcapngObj = pcapngWriter(Name,Value)` sets properties on page 3-264 using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'ByteOrder','big-endian'` specifies the byte order as big-endian.

Properties

Note The `pcapngWriter` object does not overwrite the existing PCAP or PCAPNG file. Each time when you create this object, specify a unique PCAP or PCAPNG file name.

`FileName` — Name of the PCAPNG file
`'capture'` (default) | character row vector | string scalar

Name of the PCAPNG file, specified as a character row vector or a string scalar.

Data Types: `char` | `string`

`ByteOrder` — Byte order
`'little-endian'` (default) | `'big-endian'`

Byte order, specified as `'little-endian'` or `'big-endian'`.

Data Types: char | string

FileComment — Comment for PCAPNG file
'' (default) | character vector | string scalar

Comment for the PCAPNG file, specified as a character vector or a string scalar.

Data Types: char | string

Object Functions

Specific to This Object

write	Write protocol packet data to PCAP or PCAPNG file
writeCustomBlock	Write custom block to PCAPNG file
writeInterfaceDescriptionBlock	Write interface description block to PCAPNG file

Examples

Write 5G NR Packet to PCAPNG File

Create a PCAPNG file writer object, specifying the name of the PCAPNG file.

```
pcapngObj = pcapngWriter('FileName','sample');
```

Write the interface block for 5G New Radio (NR). 5G NR packets do not have a valid link type. As per Tcpdump, if a valid link type is not present, specify the link type of SLL packet.

```
interface = '5GNR';                                          % Interface name
linkType = 113;                                              % Link type of SLL packet
timestamp = 300;                                             % Timestamp
interfaceID = writeInterfaceDescriptionBlock(pcapngObj,linkType,interface);
```

5G NR packets are not directly supported by Wireshark. To enable Wireshark to parse 5G NR packets, add encapsulation and metadata to the 5G NR packet.

```
payload = [59; 205];                                         % MAC subPDU (contains truncated buffe
radioType = 1;                                               % Frequency division duplexing
linkDir   = 0;                                               % Uplink packet
rntiType = 3;                                                % Cell-RNTI
startString = [109; 97; 99; 45; 110; 114];                   % Tag to indicate start of NR MAC signa
payloadTag = 1;                                              % Payload tag for NR MAC packets
signature = [startString; radioType; linkDir; rntiType];
macNRInfoPacket = [signature; payloadTag; payload];
```

Construct a user datagram protocol (UDP) header.

```
udpPacketLength = 8 + length(macNRInfoPacket);               % Length of header (8 bytes) and paylo
udpHeader = [163; 76;                                        % Source port number
    39; 15;                                                  % Destination port number
    fix(udpPacketLength/256); mod(udpPacketLength,256);      % Total length of UDP packet
    0; 0];                                                   % Checksum
```

Construct an IPv4 header.

```
ipPacketLength = 20 + udpPacketLength;               % Length of header (20 bytes) and paylo
ipHeader = [69;                                      % Version of IP protocol and priority o
    0;                                               % Type of service
    fix(ipPacketLength/256);mod(ipPacketLength,256); % Total length of the IPv4 packet
    0; 1;                                            % Identification
    0; 0;                                            % Flags and fragmentation offset
    64;                                              % Time to live in seconds
    17;                                              % Protocol number
    0; 0;                                            % Header checksum
    127; 0; 0; 1;                                    % Source IP address
    127; 0; 0; 1];                                   % Destination IP address
```

Construct an SLL header.

```
sllHeader = [0; 0;                                   % Packet type
    3; 4;                                            % Address resolution protocol hardware
    0; 0;                                            % Link layer address length
    0; 0; 0; 0; 0; 0; 0; 0;                          % Link layer address
    8; 0];                                           % Protocol type
```

Construct an 5G NR packet by adding encapsulation and metadata.

```
packet = [sllHeader; ipHeader; udpHeader; macNRInfoPacket];
```

Write the 5G NR packet to the PCAPNG file.

```
packetComment = 'This is NR MAC packet';             % Packet comment
write(pcapngObj,packet,timestamp,interfaceID,'PacketComment',packetComment);
```

Version History

Introduced in R2020b

References

[1] Tuexen, M. “PCAP Next Generation (Pcapng) Capture File Format.” 2020. https://www.ietf.org/.

[2] Group, The Tcpdump. “Tcpdump/Libpcap Public Repository.” Accessed May 20, 2020. https://www.tcpdump.org.

[3] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`pcapWriter` | `nrPCAPWriter` | `pcapReader`

Apps

5G Waveform Generator

Create, impair, visualize, and export 5G NR waveforms

Description

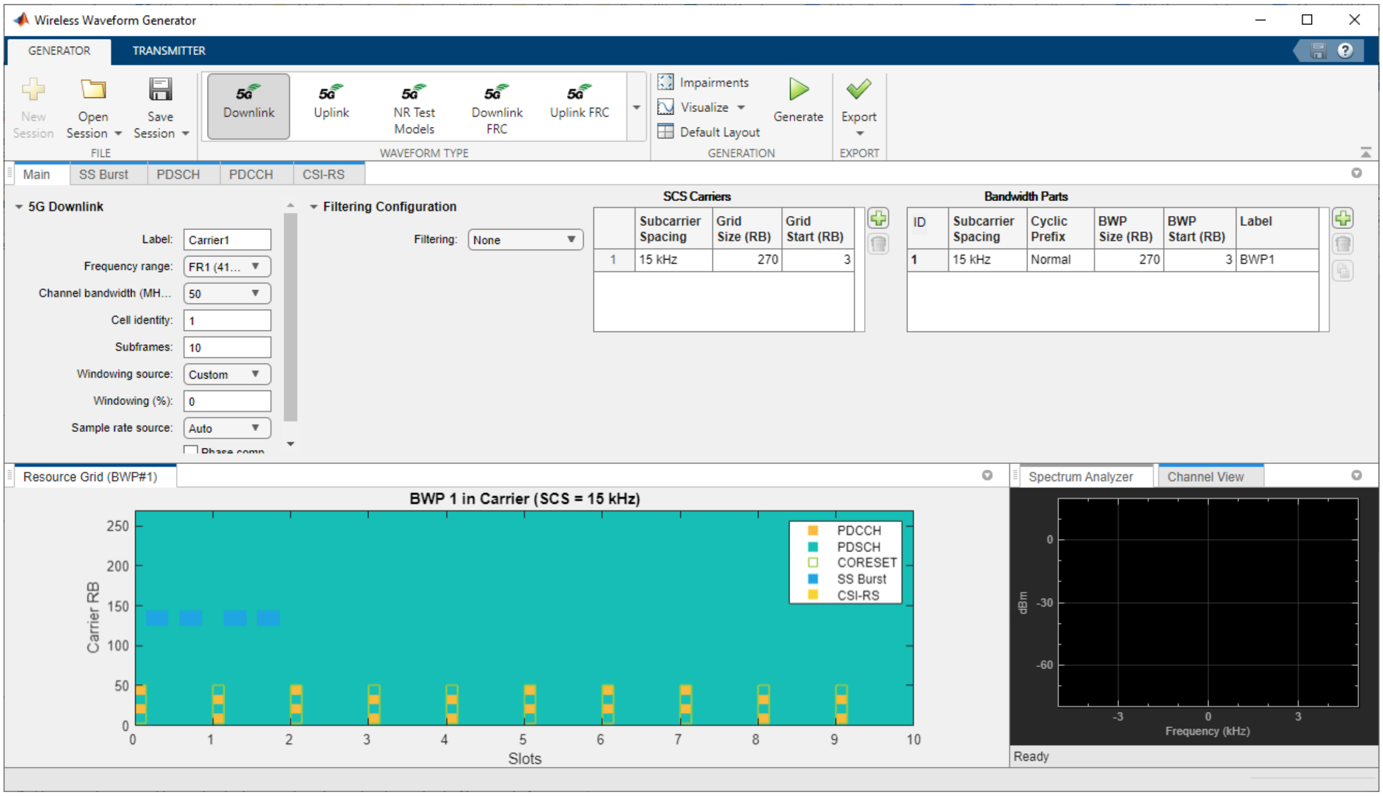
The **5G Waveform Generator** app enables you to create, impair, visualize, and export 5G NR waveforms.

The app provides these capabilities by using the **Wireless Waveform Generator** app configured for 5G NR waveform generation. Using the app, you can:

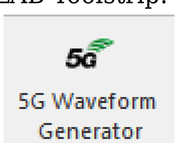
- Generate NR uplink and downlink carrier waveforms.
- Generate NR test models (NR-TM) for FR1, as defined in TS 38.141-1 Section 4.9.2 [1].
- Generate NR-TM for FR2, as defined in TS 38.141-2 Section 4.9.2 [2].
- Generate NR downlink fixed reference channel (FRC) waveforms, as defined in TS 38.101-1 Annex A.3 [3].
- Generate NR uplink FRC waveforms, as defined in TS 38.104 Annex A [4].
- Export the NR waveform to your workspace or to a `.mat` or a `.bb` file.
- Export NR waveform generation parameters to a runnable MATLAB script or a Simulink® block.
  - Use the exported script to generate your waveform without the app from the command line.
  - Use the exported block as a waveform source in a Simulink model. For more information, see Waveform From Wireless Waveform Generator App.
- Visualize the NR waveform in spectrum analyzer, channel view, OFDM grid, resource element (RE) mapping (only downlink and uplink), and complementary cumulative distribution function (CCDF) plots. The app instantly updates the OFDM grid and the channel view plots to reflect the current waveform configuration. The OFDM grid also highlights the conflicts across channels and signals.
- Distort the NR waveform by adding RF impairments, such as AWGN, phase offset, frequency offset, DC offset, IQ imbalance, and memoryless cubic nonlinearity.
- Generate an NR waveform that you can transmit using a connected radio or lab test instrument.
  - To transmit a waveform by using an SDR, connect one of the supported SDRs (ADALM-Pluto, USRP™, USRP embedded series, and Xilinx® Zynq-based radios) to your computer and have the associated add-on installed. For more information, see "Transmit Using SDR".
  - To transmit a waveform by using lab test instrument, connect one of the instruments supported by the `rfsiggen` function to your computer. For more information, see "Quick-Control RF Signal Generator Requirements" (Instrument Control Toolbox). This feature requires "Instrument Control Toolbox".
  - To transmit your waveforms over the air at full radio device rates, use the Wireless Testbench™ software and connect a supported radio to your computer. For a list of radios that support full device rates, see "Supported Radio Devices" (Wireless Testbench). This feature requires "Wireless Testbench". For an example, see "Transmit App-Generated Wireless Waveform Using Radio Transmitters" on page 4-5.

To create, impair, visualize, and export waveforms other than NR waveforms, you must reconfigure the app. For a full list of features, see the **Wireless Waveform Generator** app.

For more information, see “Create Waveforms Using Wireless Waveform Generator App”.



Open the 5G Waveform Generator App

MATLAB Toolstrip: On the **Apps** tab, under **Signal Processing and Communications**, click the app icon. 

MATLAB Command Prompt: Enter `nrWaveformGenerator`. This command opens the **Wireless Waveform Generator** app configured for 5G waveform generation.

Examples

App-Based 5G Waveform Generation

This example shows how to generate standard-compliant NR uplink and downlink carrier waveforms, NR test models (NR-TM), and NR uplink and downlink fixed reference channel (FRC) waveforms by using the **5G Waveform Generator** app. The example also discusses waveform exporting and transferring options available in the app.

Open 5G Waveform Generator App

On the **Apps** tab of the MATLAB® toolstrip, under **Signal Processing and Communications**, click the **5G Waveform Generator** app icon. This app opens the **Wireless Waveform Generator** app configured for 5G waveform generation.

Select 5G NR Waveform

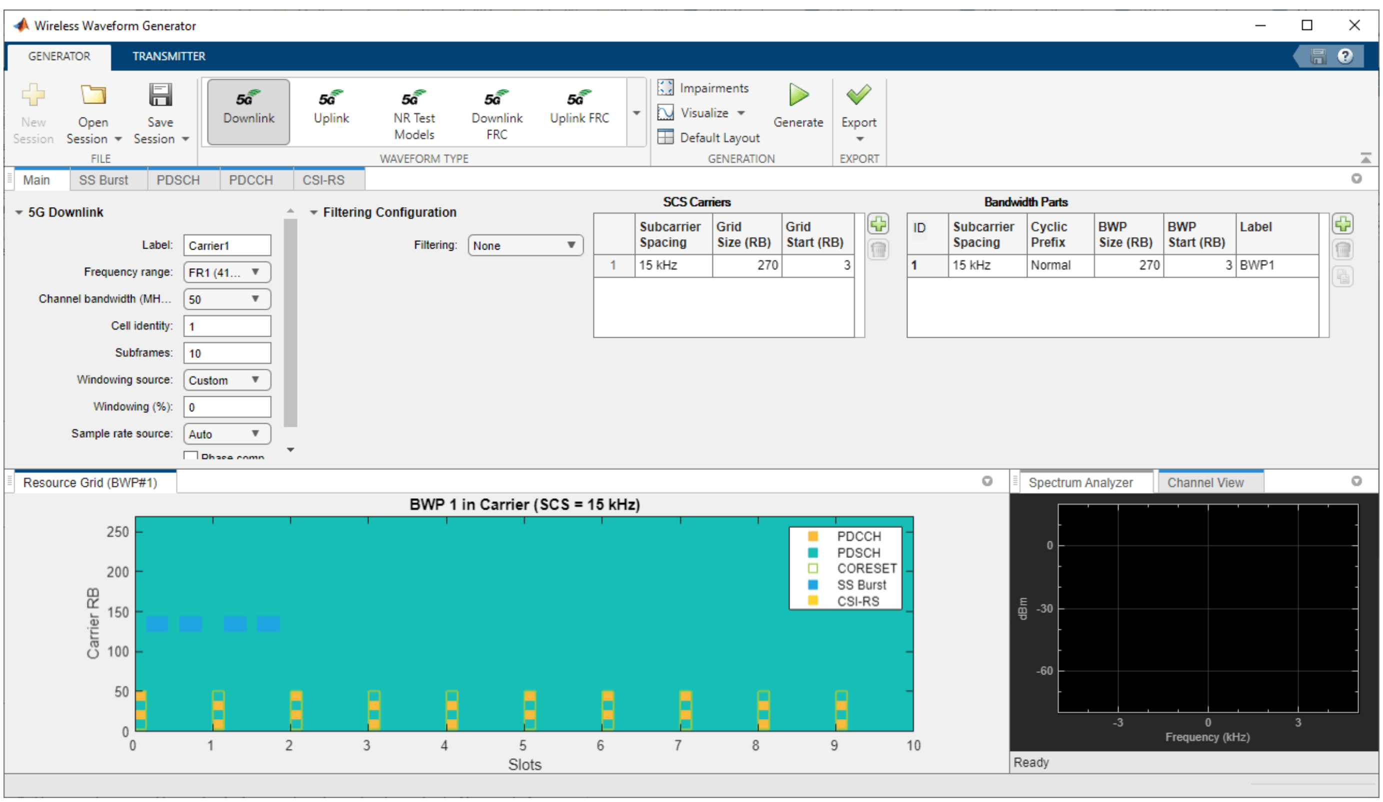
In the **Waveform Type** section on the app toolstrip, click the waveform you want to generate. Select one of these waveforms.

- `5G Downlink`
- `5G Uplink`
- `5G Test Models`
- `5G Downlink FRC`
- `5G Uplink FRC`

Generate 5G NR Waveform

Depending on the selected waveform, the app presents specific tabs where you can set the parameters of the selected waveform. On the app toolstrip, in the **Generation** section, you can add impairments and set visualization tools applicable for the selected waveform. To visualize the waveform on the selected visualization tools, click **Generate**.

For example, this figure shows the visualization results of a 5G NR downlink waveform using default parameters.



Export Generated Waveform to MATLAB Workspace or File

To export the generated waveform, on the app toolstrip, in the **Export** section, select **Export to Workspace** or **Export to File**. You can export the waveform as a structure to the MATLAB workspace or a MAT-file (`.mat`). You can also export the waveform to a baseband file (`.bb`).

Export Waveform Configuration Parameters to MATLAB Script

To export waveform configuration parameters as a MATLAB script, on the app toolstrip, in the **Export** section, select **Export to MATLAB Script**. You can run the exported MATLAB script to generate the waveform without the app.

Export Waveform Configuration Parameters to Simulink

To export waveform configuration parameters as a Simulink block, on the app toolstrip, in the **Export** section, select **Export to Simulink**. You can use the exported block to generate the waveform in a Simulink model without the app.

Transmit 5G NR Waveform

To transmit the generated waveform using a connected radio or lab test instrument, on the app toolstrip, click on the **Transmitter** tab.

- To transmit your waveforms over the air at full radio device rates, use the Wireless Testbench™ software and connect a supported radio to your computer. For a list of radios that support full device rates, see “Supported Radio Devices” (Wireless Testbench). This feature requires “Wireless Testbench”.
- To transmit a waveform by using an SDR, connect one of the supported SDRs (ADALM-Pluto, USRP™, USRP embedded series, and Xilinx® Zynq-based radios) to your computer and have the associated add-on installed. For more information, see “Transmit Using SDR”.
- To transmit a waveform by using a lab test instrument, connect one of the instruments supported by the `rfsiggen` (Instrument Control Toolbox) function to your computer. For more information, see “Quick-Control RF Signal Generator Requirements” (Instrument Control Toolbox). This feature requires “Instrument Control Toolbox”.

Transmit App-Generated Wireless Waveform Using Radio Transmitters

This example shows how to use the NI™ USRP™ N310, USRP N320, USRP N321, and USRP X310 radio transmitters available in the **Wireless Waveform Generator** app to transmit an app-generated waveform over the air (requires Wireless Testbench™). These radio transmitters enable you to transmit up to 2 GB of contiguous data over the air at full radio device rate.

Introduction

The Wireless Waveform Generator app is an interactive tool for creating, impairing, visualizing, and transmitting waveforms. Using the USRP N310, USRP N320, USRP N321, and USRP X310 radio transmitters available in the app, you can transmit your generated waveform repeatedly over the air. You can also export the waveform generation and transmission parameters to a runnable MATLAB® script. This example shows how to configure these radio transmitters.

Although this example shows how to transmit an OFDM waveform, the same process applies for all waveform types that you can generate with the app.

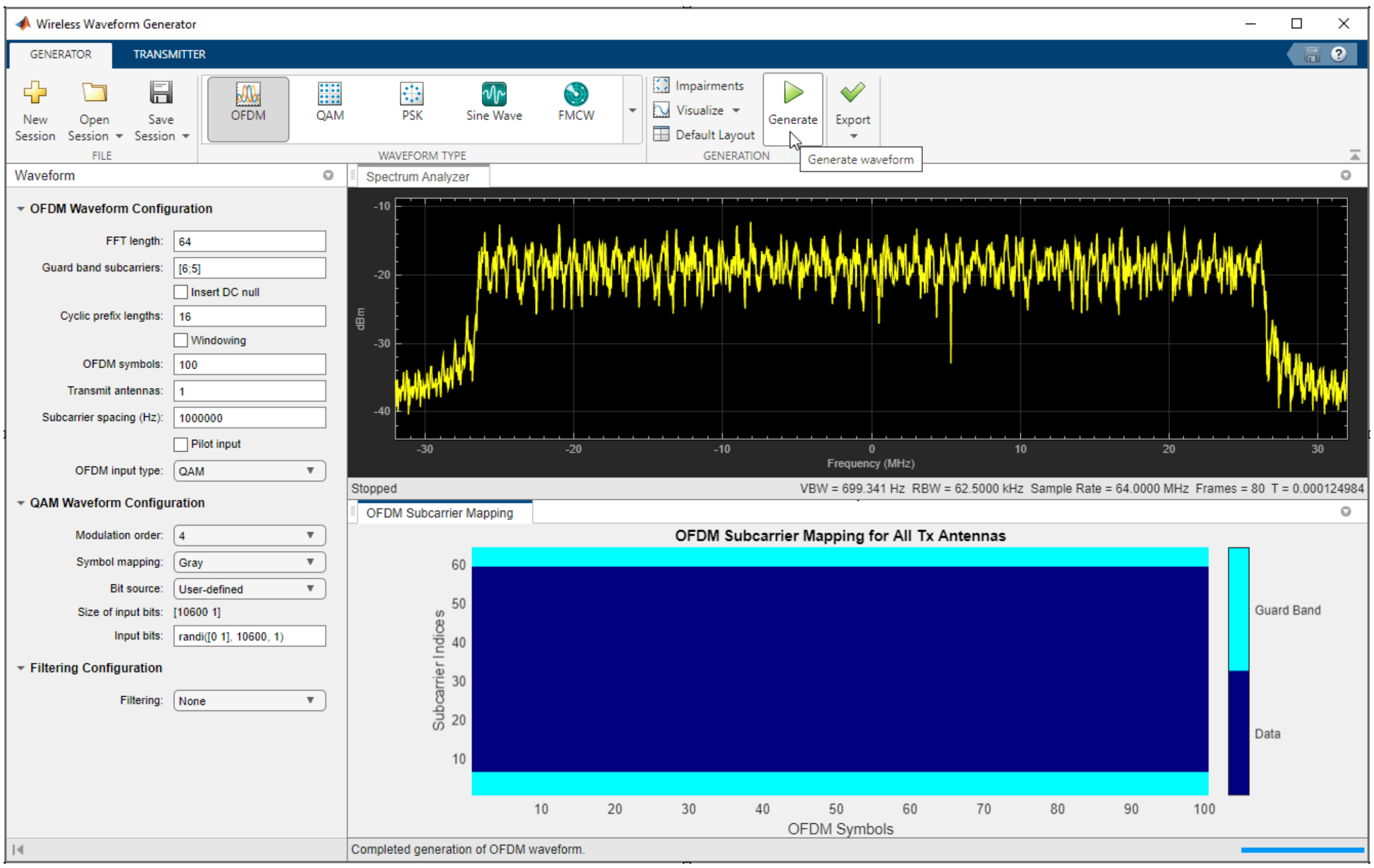
Set Up for Radio Transmission

To use the radio transmitters in the app, you need to install the Wireless Testbench Support Package for NI USRP Radios add-on and set up your radio outside the app. For more information, see “Connect and Set Up NI USRP Radios” (Wireless Testbench).

Generate Waveform for Transmission

Open the **Wireless Waveform Generator** app by clicking the app icon on the **Apps** tab, under **Signal Processing and Communications**. Alternatively, enter `wirelessWaveformGenerator` at the MATLAB command prompt.

In the **Waveform Type** section, select an OFDM waveform by clicking **OFDM**. In the leftmost pane of the app, adjust any configuration parameters for the selected waveform. Then generate the configuration by clicking **Generate** in the app toolstrip.



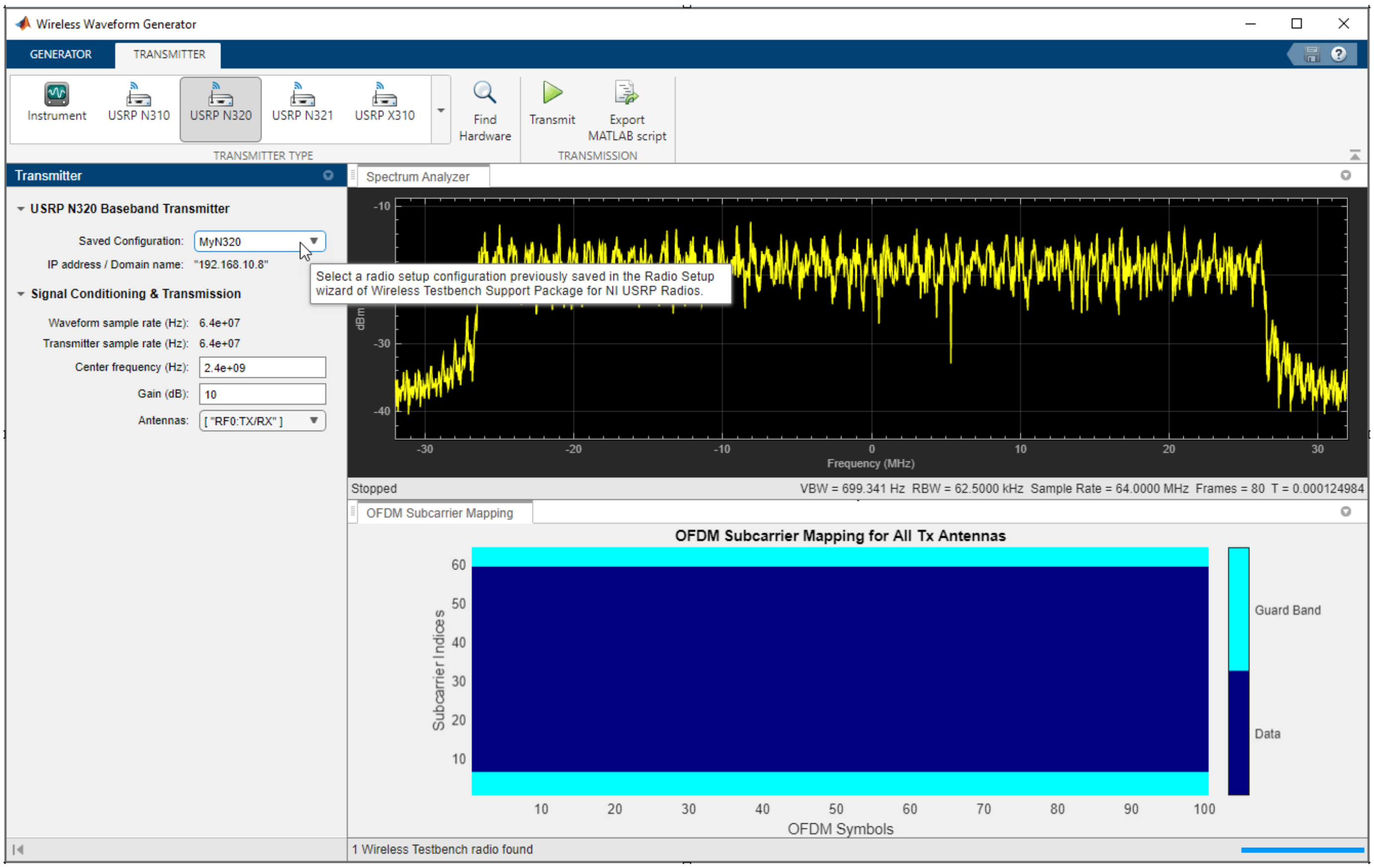
Configure Radio Transmitter

Select the **Transmitter** tab from the app toolstrip. In the transmitter gallery, select the USRP N310, USRP N320, USRP N321, or USRP X310 radio transmitter.

In the leftmost pane of the app, select the name of a radio setup configuration that you saved using the Radio Setup wizard. For more information, see “Connect and Set Up NI USRP Radios” (Wireless Testbench).

Set the center frequency, gain, and antennas configuration parameters. The app automatically sets the waveform sample rate based on the waveform that you generated earlier. The radio transmitter uses onboard data buffering to ensure contiguous data transmission at up to the full hardware sample rate. If necessary, to achieve the specified sample rate, the radio uses a Farrow rate converter. Use this list as a reference when setting the sample rate:

- **USRP N310** — 120,945 Hz to 76.8 MHz, or one of: 122.88 MHz, 125 MHz, or 153.6 MHz
- **USRP N320** — 196,851 Hz to 125 MHz, or one of: 200 MHz, 245.76 MHz or 250 MHz
- **USRP N321** — 196,851 Hz to 125 MHz, or one of: 200 MHz, 245.76 MHz or 250 MHz
- **USRP X310** — 181,418 Hz to 100 MHz, or one of: 184.32 MHz or 200 MHz



Transmit Waveform

To transmit the waveform continuously, click **Transmit**. To end the continuous transmission, click **Stop transmission**. To export the waveform generation and transmission parameters to a runnable MATLAB script, click **Export MATLAB script**.

Version History

Introduced in R2020a

References

[1] 3GPP TS 38.141-1. "NR; Base Station (BS) conformance testing Part 1: Conducted conformance testing." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.141-2. "NR; Base Station (BS) conformance testing Part 2: Radiated conformance testing." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 38.101-1. "NR; User Equipment (UE) radio transmission and reception; Part 1: Range 1 Standalone." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[4] 3GPP TS 38.104. "NR; Base Station (BS) radio transmission and reception." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Apps
Wireless Waveform Generator

Blocks
Waveform From Wireless Waveform Generator App

Topics
"Create Waveforms Using Wireless Waveform Generator App"

5

Blocks

Waveform From Wireless Waveform Generator App

Wireless waveform source exported to Simulink
Library: None

Description

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

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

For an overview of the waveform types that you can export to Simulink using the 5G Toolbox software, see the **5G Waveform Generator** app.

To generate a block:

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

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

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

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

Limitations

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

Ports

Output

`wf` — Time-domain wireless waveform
complex matrix

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

Data Types: `double`

Parameters

Read-Only Waveform Parameters

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

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

This parameter is read-only.

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

`Waveform length` — Waveform length
positive integer

This parameter is read-only.

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

Simulation Parameters

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

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

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

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

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

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

Version History

Introduced in R2021b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also

Apps
5G Waveform Generator