

**5G Toolbox™**

User's Guide



**MATLAB®**

R2020b



# How to Contact MathWorks



Latest news: [www.mathworks.com](http://www.mathworks.com)  
Sales and services: [www.mathworks.com/sales\\_and\\_services](http://www.mathworks.com/sales_and_services)  
User community: [www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)  
Technical support: [www.mathworks.com/support/contact\\_us](http://www.mathworks.com/support/contact_us)



Phone: 508-647-7000



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

*5G Toolbox™ User Guide*

© COPYRIGHT 2018–2020 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

## Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See [www.mathworks.com/trademarks](http://www.mathworks.com/trademarks) for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

## Patents

MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

## 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)

<b>1</b>	<b>Downlink Channels</b>	
	5G NR Downlink Carrier Waveform Generation .....	1-2
	NR PDSCH Resource Allocation and DM-RS and PT-RS Reference Signals .....	1-14
	NR Cell Search and MIB and SIB1 Recovery .....	1-29
	NR PDSCH Throughput .....	1-56
	Downlink Control Processing and Procedures .....	1-75
	NR Channel Estimation Using CSI-RS .....	1-81
	5G NR CSI-RS Measurements .....	1-88
	NR SSB Beam Sweeping .....	1-95
	NR Downlink Transmit-End Beam Refinement Using CSI-RS .....	1-110

<b>2</b>	<b>Uplink Channels</b>	
	5G NR Uplink Carrier Waveform Generation .....	2-2
	NR PUSCH Resource Allocation and DM-RS and PT-RS Reference Signals .....	2-16
	NR PUSCH Throughput .....	2-34
	NR SRS Configuration .....	2-48
	NR Uplink Channel State Information Estimation Using SRS .....	2-68
	5G NR PRACH Configuration .....	2-81
	5G NR PRACH Waveform Generation .....	2-94
	5G NR PRACH Detection Test .....	2-98
	NR UCI Multiplexing on PUSCH .....	2-105

## Signal Reception

### 3

Extract PBCH Symbols and Channel Estimates for Decoding .....	3-2
Deep Learning Data Synthesis for 5G Channel Estimation .....	3-5
NR Phase Noise Modeling and Compensation .....	3-16

## End-To-End Simulation

### 4

Transmission Over MIMO Channel Model with Delay Profile TDL .....	4-2
Plot Path Gains for TDL-E Delay Profile with SISO .....	4-4
Reconstruct Channel Impulse Response Using CDL Channel Path Filters .....	4-6
5G NR CQI Reporting .....	4-8
Visualize CDL Channel Model Characteristics .....	4-17

## System-Level Simulation

### 5

NR PUSCH FDD Scheduling .....	5-2
NR FDD Scheduling Performance Evaluation .....	5-21
NR TDD Symbol Based Scheduling Performance Evaluation .....	5-39
NR Cell Performance Evaluation with Physical Layer Integration .....	5-61
NR Intercell Interference Modeling .....	5-78
Generate and Visualize FTP Application Traffic Pattern .....	5-95

## Test and Measurement

### 6

5G NR-TM and FRC Waveform Generation .....	6-2
App-Based 5G NR-TM and FRC Waveform Generation .....	6-19

<b>5G NR Downlink ACLR Measurement</b> .....	<b>6-22</b>
<b>Modeling and Testing an NR RF Transmitter</b> .....	<b>6-32</b>
<b>Modeling and Testing an NR RF Receiver with LTE Interference</b> .....	<b>6-45</b>
<b>EVM Measurement of 5G NR PDSCH Waveforms</b> .....	<b>6-59</b>

## **Code Generation and Deployment**

# 7

<b>What is C Code Generation from MATLAB?</b> .....	<b>7-2</b>
Using MATLAB Coder .....	<b>7-2</b>
C/C++ Compiler Setup .....	<b>7-2</b>
Functions and System Objects That Support Code Generation .....	<b>7-3</b>



# Downlink Channels

---

## 5G NR Downlink Carrier Waveform Generation

This example shows how to use the 5G NR downlink carrier waveform generator to create a baseband component carrier waveform.

### Introduction

This example shows how to parameterize and generate a 5G New Radio (NR) downlink waveform using `nrWaveformGenerator`. The following channels and signals can be generated:

- PDSCH and its associated DM-RS and PT-RS
- PDCCH and its associated DM-RS
- PBCH and its associated DM-RS
- PSS and SSS
- CSI-RS

This example supports the parameterization and generation of multiple SCS specific carriers and multiple bandwidth parts (BWP). Multiple instances of the PDSCH and PDCCH channels can be generated over the different BWPs. Sets of CORESETs and search space monitoring opportunities can be configured for mapping the PDCCHs. Note that no precoding is applied to the physical channels and signals in this example.

### Waveform and Carrier Configuration

The baseband waveform is parameterized by an `nrDLCarrierConfig` object and a set of additional objects associated with its channels and signals. This section sets the SCS specific carrier bandwidths in resource blocks, the cell ID, and the length of the generated waveform in subframes. You can control SCS carrier bandwidths and guardbands using the `NStartGrid` and `NSizeGrid` parameters.

```

waveconfig = nrDLCarrierConfig(); % Create an instance of the waveform's parameter object
waveconfig.NCellID = 0;          % Cell identity
waveconfig.ChannelBandwidth = 40; % Channel bandwidth (MHz)
waveconfig.FrequencyRange = 'FR1'; % 'FR1' or 'FR2'
waveconfig.NumSubframes = 10;    % Number of 1ms subframes in generated waveform (1,2,4,8 slots)

% Define a set of SCS specific carriers, using the maximum sizes for a
% 40 MHz NR channel. See TS 38.101-1 for more information on defined
% bandwidths and guardband requirements

scscarriers = {nrSCSCarrierConfig(),nrSCSCarrierConfig()};
scscarriers{1}.SubcarrierSpacing = 15;
scscarriers{1}.NSizeGrid = 216;
scscarriers{1}.NStartGrid = 0;

scscarriers{2}.SubcarrierSpacing = 30;
scscarriers{2}.NSizeGrid = 106;
scscarriers{2}.NStartGrid = 1;

```

### SS Burst

In this section you can set the parameters for the SS burst. The numerology of the SS burst can be different from other parts of the waveform. This is specified via the block pattern parameter as specified in TS 38.213 Section 4.1. A bitmap is used to specify which blocks are transmitted in a 5ms half-frame burst. The periodicity in milliseconds and the power of the burst can also be set here.

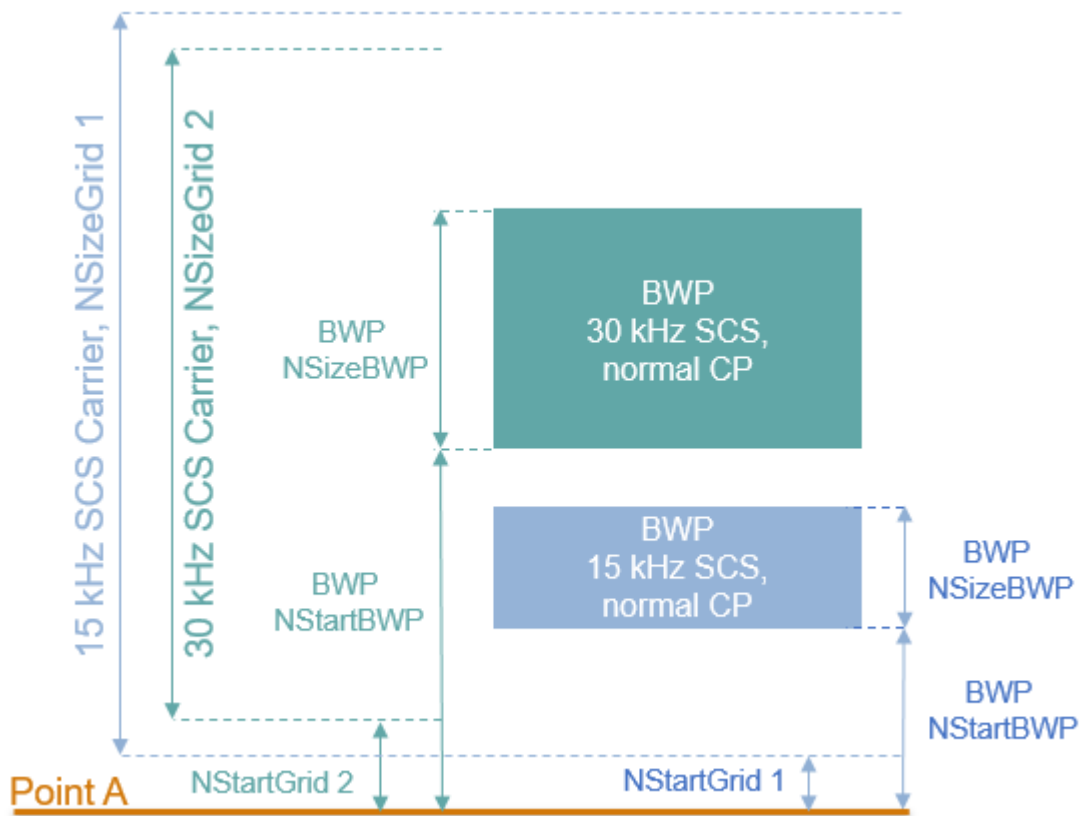


Other SS burst parameters not shown here can also be set. For the full list see the help for `nrWavegenSSBurstConfig`.

```
% SS burst configuration
ssburst = nrWavegenSSBurstConfig();
ssburst.Enable = 1;           % Enable SS Burst
ssburst.Power = 0;           % Power scaling in dB
ssburst.BlockPattern = 'Case B'; % Case B (30kHz) subcarrier spacing
ssburst.TransmittedBlocks = [1 1 1 1]; % Bitmap indicating blocks transmitted in a 5ms half-frame
ssburst.Period = 20;         % SS burst set periodicity in ms (5, 10, 20, 40, 80, 160)
ssburst.NCRBSSB = [];       % Frequency offset of SS burst (CRB), use [] for the wave
```

### Bandwidth Parts

A BWP is formed by a set of contiguous resources sharing a numerology on a given carrier. This example supports the use of multiple BWPs using a cell array. Each entry in the array represents a BWP. For each BWP you can specify the subcarrier spacing (SCS), the cyclic prefix (CP) length and the bandwidth. The `SubcarrierSpacing` parameter maps the BWP to one of the SCS specific carriers defined earlier. The `NStartBWP` parameter controls the location of the BWP in the carrier, relative to point A. This is expressed in terms of the BWP numerology. Different BWPs can overlap with each other.



```
% Bandwidth parts configurations
bwp = {nrWavegenBWPConfig(), nrWavegenBWPConfig()};
bwp{1}.BandwidthPartID = 1; % Bandwidth part ID
```

```

bwp{1}.SubcarrierSpacing = 15;    % BWP subcarrier spacing
bwp{1}.CyclicPrefix = 'Normal';  % BWP cyclic prefix for 15 kHz
bwp{1}.NSizeBWP = 25;           % Size of BWP
bwp{1}.NStartBWP = 12;          % Position of BWP, relative to point A (i.e. CRB)

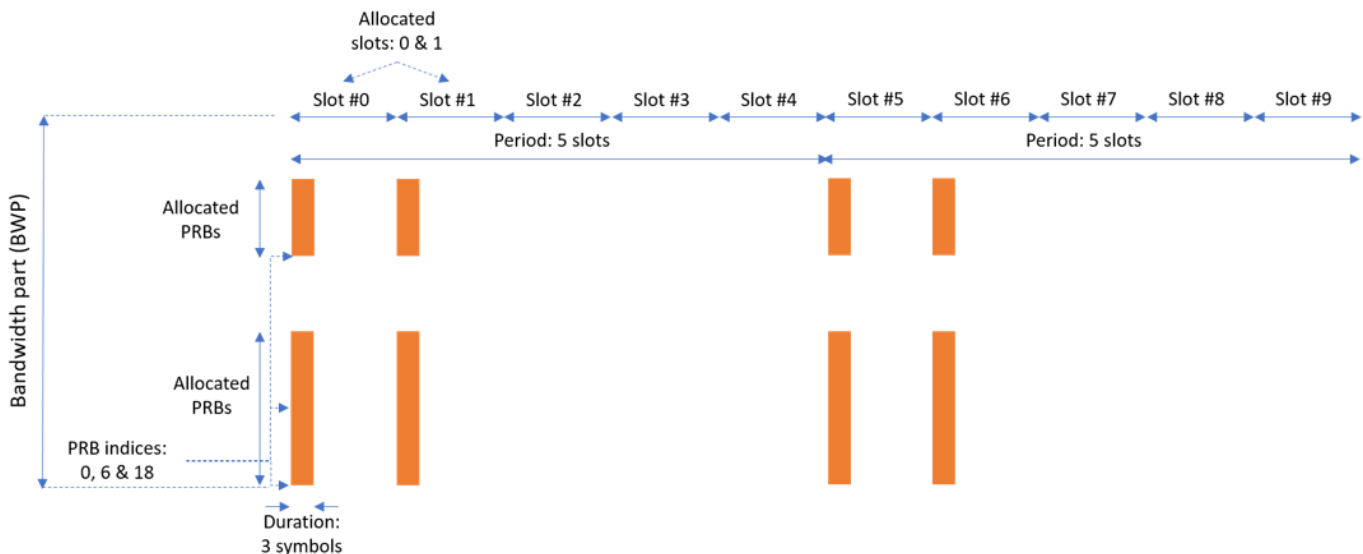
bwp{2}.BandwidthPartID = 2;      % Bandwidth part ID
bwp{2}.SubcarrierSpacing = 30;   % BWP subcarrier spacing
bwp{2}.CyclicPrefix = 'Normal';  % BWP cyclic prefix for 30 kHz
bwp{2}.NSizeBWP = 50;           % Size of BWP
bwp{2}.NStartBWP = 51;          % Position of BWP, relative to point A (i.e. CRB)
    
```

**CORESET and Search Space Configuration**

The parameters in this section specify the control resource set (CORESET) and the PDCCH search space configuration. The CORESET and search spaces specify the possible locations (in time and frequency) of the control channel transmissions for a given numerology. The generator supports multiple CORESETs and search spaces. The following parameters can be specified:

- The OFDM symbols which specify the first symbol of each CORESET monitoring opportunity in a slot
- The duration of the block of allocated slots within a period
- Periodicity of the allocation pattern
- The CORESET duration in symbols, either 1, 2 or 3
- A bitmap defining the allocated physical resource blocks (PRB) of the CORESET. Note that the CORESET frequency allocation is defined in blocks of 6 PRBs, aligned in common resource block (CRB) numbering, i.e. relative to point A. Each bit in the bitmap selects all 6 PRBs in the CRB aligned block that contains it.
- CCE-to-REG mapping which can be 'interleaved' or 'noninterleaved'
- Resource-element group (REG) bundle size (L), either (2,6) or (3,6), based on CORESET duration
- Interleaver size, either 2, 3, or 6
- Shift index, a scalar value in range 0...274

The figure below shows the meaning of some of the CORESET parameters.



```

% CORESET and search space configurations
coresets = {nrCORESETConfig()};
coresets{1}.CORESETID = 1;
coresets{1}.Duration = 3;
coresets{1}.FrequencyResources = [1 1 0 1];
coresets{1}.CCEREGMapping = 'noninterleaved';
coresets{1}.REGBundleSize = 3;
coresets{1}.InterleaverSize = 2;
coresets{1}.ShiftIndex = waveconfig.NCellID;

searchspaces = {nrSearchSpaceConfig()};
searchspaces{1}.SearchSpaceID = 1;
searchspaces{1}.CORESETID = 1;
searchspaces{1}.SearchSpaceType = 'ue';
searchspaces{1}.SlotPeriodAndOffset = [5,0];
searchspaces{1}.Duration = 2;
searchspaces{1}.StartSymbolWithinSlot = 0;
searchspaces{1}.NumCandidates = [8 8 4 2 0];

```

### PDCCH Instances Configuration

This section specifies the parameters for the set of PDCCH instances in the waveform. Each element in the structure array defines a PDCCH sequence instance. The following parameters can be set:

- Enable/disable the PDCCH sequence
- Specify the BWP carrying the PDCCH
- PDCCH instance power in dB
- Enable/disable DCI channel coding
- Allocated search spaces within the CORESET monitoring occasion sequence
- Search space (and CORESET) which carries the PDCCH instances
- Periodicity of the allocation. If this is set to empty it indicates no repetition
- The aggregation level (AL) of the PDCCH (number of control channel elements (CCEs))
- The allocated candidate which specifies the CCE used for the transmission of the PDCCH
- RNTI
- Scrambling NID for this PDCCH and its associated DM-RS
- DM-RS power boosting
- DCI message payload size
- DCI message data source. You can use an array of bits or one of the following standard PN sequences: 'PN9-ITU', 'PN9', 'PN11', 'PN15', 'PN23'. The seed for the generator can be specified using a cell array in the form {'PN9', seed}. If no seed is specified, the generator is initialized with all ones

```

pdcch = {nrWavegenPDCCHConfig()};
pdcch{1}.Enable = 1 ;
pdcch{1}.Power = 1.1;
pdcch{1}.Coding = 1;
pdcch{1}.BandwidthPartID = 1;
pdcch{1}.SearchSpaceID = 1;
pdcch{1}.SlotAllocation = 0;
pdcch{1}.Period = 5;
pdcch{1}.AggregationLevel = 8;
pdcch{1}.AllocatedCandidate = 1;

```

```

pdcch{1}.RNTI = 0; % RNTI
pdcch{1}.DMRSScramblingID = 1; % PDCCH and DM-RS scrambling NID
pdcch{1}.DMRSPower = 0; % Additional power boosting in dB
pdcch{1}.DataBlockSize = 20; % DCI payload size
pdcch{1}.DataSource = 'PN9'; % DCI data source

```

### PDSCH Instances Configuration

This section specifies the set of PDSCH instances in the waveform. Each element in the cell array of `nrWavegenPDSCHConfig` defines a PDSCH sequence instance. This example defines two PDSCH sequence instances.

#### General Parameters

Set these parameters for each PDSCH sequence instance:

- Enable or disable this PDSCH sequence
- Specify the BWP carrying the PDSCH. The PDSCH will use the SCS specified for this BWP
- Power scaling in dB
- Enable or disable DL-SCH transport channel coding
- Transport block data source. You can use an array of bits or one of the following standard PN sequences: 'PN9-ITU', 'PN9', 'PN11', 'PN15', 'PN23'. The seed for the generator can be specified using a cell array in the form {'PN9', seed}. If no seed is specified, the generator is initialized with all ones
- Target code rate used to calculate the transport block sizes
- Overhead parameter
- Symbol modulation
- Number of layers
- Redundancy version (RV) sequence
- Enable or disable the interleaving of the virtual to physical resource block mapping. If this parameter is not specified, the direct, non-interleaved mapping is considered
- Bundle size for the interleaved map, specified by the higher layer parameter `vrB-ToPRB-Interleaver`. If this parameter is not specified, the bundle size is set to 2

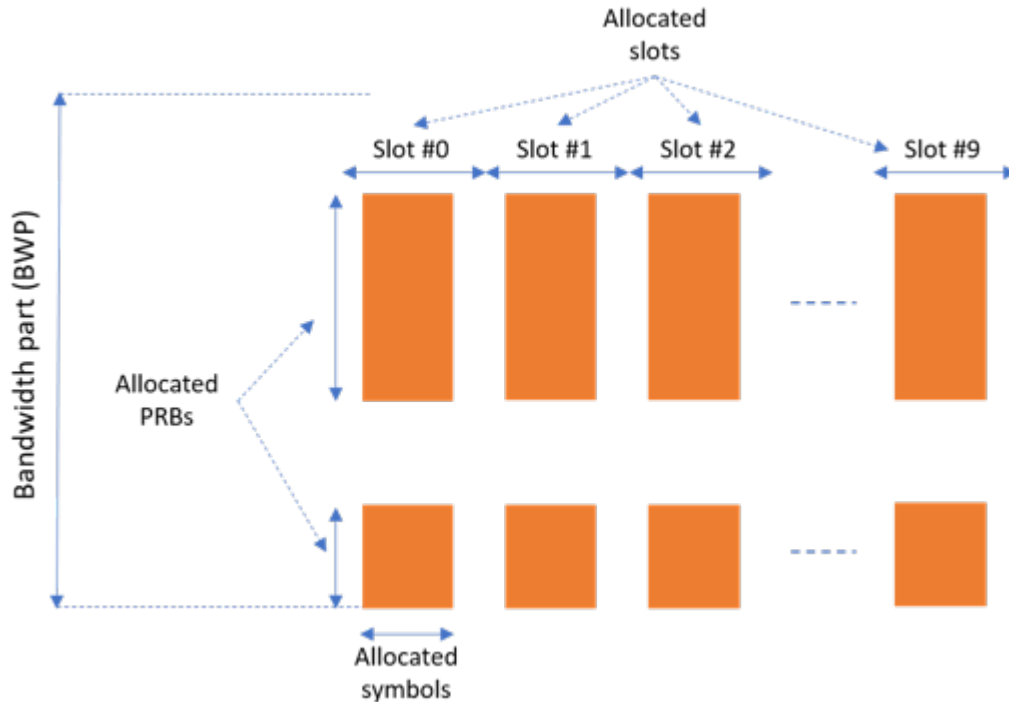
```

pdsch = {nrWavegenPDSCHConfig()};
pdsch{1}.Enable = 1; % Enable PDSCH sequence
pdsch{1}.BandwidthPartID = 1; % Bandwidth part of PDSCH transmission
pdsch{1}.Power = 0; % Power scaling in dB
pdsch{1}.Coding = 1; % Enable DL-SCH transport channel coding
pdsch{1}.DataSource = 'PN9'; % Channel data source
pdsch{1}.TargetCodeRate = 0.4785; % Code rate used to calculate transport block sizes
pdsch{1}.XOverhead = 0; % Rate matching overhead
pdsch{1}.Modulation = 'QPSK'; % 'QPSK', '16QAM', '64QAM', '256QAM'
pdsch{1}.NumLayers = 2; % Number of PDSCH layers
pdsch{1}.RVSequence = [0,2,3,1]; % RV sequence to be applied cyclically across the PDSCH
pdsch{1}.VRBToPRBInterleaving = 0; % Disable interleaved resource mapping
pdsch{1}.VRBBundleSize = 2; % vrB-ToPRB-Interleaver parameter

```

#### Allocation

The following diagram represents some of the parameters used in the PDSCH allocation.



You can set the following parameters to control the PDSCH allocation. Note that these parameters are relative to the BWP. The specified PDSCH allocation will avoid the locations used for the SS burst.

- Symbols in a slot allocated to each PDSCH instance
- Slots in a frame used for the sequence of PDSCH
- Period of the allocation in slots. If this is empty it indicates no repetition
- The allocated PRBs are relative to the BWP
- RNTI. This value is used to link the PDSCH to an instance of the PDCCH
- NID for scrambling the PDSCH bits

```
pdsch{1}.SymbolAllocation = [2,9]; % First symbol and length
pdsch{1}.SlotAllocation = 0:9; % Allocated slot indices for PDSCH sequence
pdsch{1}.Period = 15; % Allocation period in slots
pdsch{1}.PRBSet = [0:5, 10:20]; % PRB allocation
pdsch{1}.RNTI = 0; % RNTI
pdsch{1}.NID = 1; % Scrambling for data part
```

CORESETs and sets of PRB can be specified for rate matching around, if required

- The PDSCH can be rate matched around one or more CORESETs
- The PDSCH can be rate matched around other resource allocations

```
pdsch{1}.ReservedCORESET = 1; % Rate matching pattern, defined by CORESET IDs
pdsch{1}.ReservedPRB{1}.PRBSet = []; % Rate matching pattern, defined by set of PRB (RRC 'bitr
pdsch{1}.ReservedPRB{1}.SymbolSet = [];
pdsch{1}.ReservedPRB{1}.Period = [];
```

### PDSCH DM-RS Configuration

Set the DM-RS parameters

```
% Antenna port and DM-RS configuration (TS 38.211 section 7.4.1.1)
pdsch{1}.MappingType = 'A';           % PDSCH mapping type ('A'(slot-wise),'B'(non slot-wise))
pdsch{1}.DMRSPower = 0;                % Additional power boosting in dB

pdsch{1}.DMRS.DMRSPortSet = [] ;       % DM-RS antenna ports used ([] gives port numbers 0:N)
pdsch{1}.DMRS.DMRSTypeAPosition = 2;   % Mapping type A only. First DM-RS symbol position (2)
pdsch{1}.DMRS.DMRSLength = 1;         % Number of front-loaded DM-RS symbols (1(single symbol))
pdsch{1}.DMRS.DMRSAdditionalPosition = 0; % Additional DM-RS symbol positions (max range 0...3)
pdsch{1}.DMRS.DMRSConfigurationType = 2; % DM-RS configuration type (1,2)
pdsch{1}.DMRS.NumCDMGroupsWithoutData = 1; % CDM groups without data (max range 1...3)
pdsch{1}.DMRS.NIDNSCID = 1;           % Scrambling identity (0...65535)
pdsch{1}.DMRS.NSCID = 0;              % Scrambling initialization (0,1)
```

### PDSCH PT-RS Configuration

Set the PT-RS parameters

```
% PT-RS configuration (TS 38.211 section 7.4.1.2)
pdsch{1}.EnablePTRS = 0;               % Enable or disable the PT-RS (1 or 0)
pdsch{1}.PTRSPower = 0;                % Additional PT-RS power boosting in dB

pdsch{1}.PTRS.TimeDensity = 1;         % Time density (L_PT-RS) of PT-RS (1,2,4)
pdsch{1}.PTRS.FrequencyDensity = 2;   % Frequency density (K_PT-RS) of PT-RS (2,4)
pdsch{1}.PTRS.REOffset = '00';        % PT-RS resource element offset ('00','01','10','11')
pdsch{1}.PTRS.PTRSPortSet = 0;        % PT-RS antenna ports must be a subset of DM-RS ports

% When PT-RS is enabled, the DM-RS ports must be in range 0 to 3 for DM-RS
% configuration type 1 and in range 0 to 5 for DM-RS configuration type 2.
% Nominally the antenna port of PT-RS is the lowest DM-RS port number.
```

### Specifying Multiple PDSCH Instances

A second PDSCH sequence instance is specified next using the second BWP.

```
pdsch{2} = pdsch{1};
pdsch{2}.Enable = 1;
pdsch{2}.BandwidthPartID = 2;          % PDSCH mapped to 2nd BWP
pdsch{2}.SymbolAllocation = [0,12];
pdsch{2}.SlotAllocation = [2:4,6:20];
pdsch{2}.PRBSet = [25:30, 35:38];     % PRB allocation, relative to BWP
```

### CSI-RS

This section configures channel state information reference signals (CSI-RS) in the waveform. Each element in the cell array represents a set of CSI-RS resources associated with a BWP.

#### General Parameters

Set these parameters for a set of CSI-RS resources:

- Enable or disable this set of CSI-RS resources
- Specify the BWP carrying this set of CSI-RS resources. The CSI-RS resource(s) configuration will use the SCS specified for this BWP

- Specify the power scaling in dB. Providing a scalar defines the power scaling for a single CSI-RS resource or all configured CSI-RS resources. Providing a vector defines a separate power level for each of the CSI-RS resources.

```
csirs = {nrWavegenCSIRSConfig()};
csirs{1}.Enable = 0;
csirs{1}.BandwidthPartID = 1;
csirs{1}.Power = 3;    % in dB
```

### CSI-RS configuration

You can configure the following parameters for one or more zero-power (ZP) or non-zero-power (NZP) CSI-RS resource configurations.

- Type of CSI-RS resource(s) ('nzp','zp')
- Row number corresponds to CSI-RS resource(s) as defined in TS 38.211 Table 7.4.1.5.3-1 (1...18)
- Frequency density of CSI-RS resource(s) ('one','three','dot5even','dot5odd')
- Subcarrier locations of CSI-RS resource(s) within a resource block (RB)
- Number of RBs allocated to CSI-RS resource(s) (1...275)
- Starting RB index of CSI-RS resource(s) allocation relative to the carrier resource grid (0...274)
- OFDM symbol locations of CSI-RS resource(s) within a slot
- The period and offset of slots (0-based) of CSI-RS resource(s). This parameter can be a vector or a cell array of vectors. In the latter case, each cell corresponds to an individual CSI-RS resource. In case of a vector, the same set of slots is used for all CSI-RS resources
- Scrambling identity corresponds to CSI-RS resource(s) for pseudo-random sequence generation (0...1023)

```
csirs{1}.CSIRSType = {'nzp','zp'};
csirs{1}.RowNumber = [3 5];
csirs{1}.Density = {'one','one'};
csirs{1}.SubcarrierLocations = {6,4};
csirs{1}.NumRB = 25;
csirs{1}.RBOffset = 12;
csirs{1}.SymbolLocations = {13,9};
csirs{1}.CSIRSPeriod = {[5 0],[5 0]};
csirs{1}.NID = 5;
```

### Specifying Multiple CSI-RS Instances

A set of CSI-RS resources associated with the second BWP.

```
csirs{2} = nrWavegenCSIRSConfig();
csirs{2}.Enable = 0;
csirs{2}.BandwidthPartID = 2;
csirs{2}.Power = 3; % in dB
csirs{2}.CSIRSType = {'nzp','nzp'};
csirs{2}.RowNumber = [1 1];
csirs{2}.Density = {'three','three'};
csirs{2}.SubcarrierLocations = {0,0};
csirs{2}.NumRB = 50;
csirs{2}.RBOffset = 50;
csirs{2}.SymbolLocations = {6,10};
csirs{2}.CSIRSPeriod = {[10,1],[10,1]};
csirs{2}.NID = 0;
```

## Waveform Generation

This section assigns all the channel and signal parameters into the main carrier configuration object `nrDLCarrierConfig`, then generates and plots the waveform.

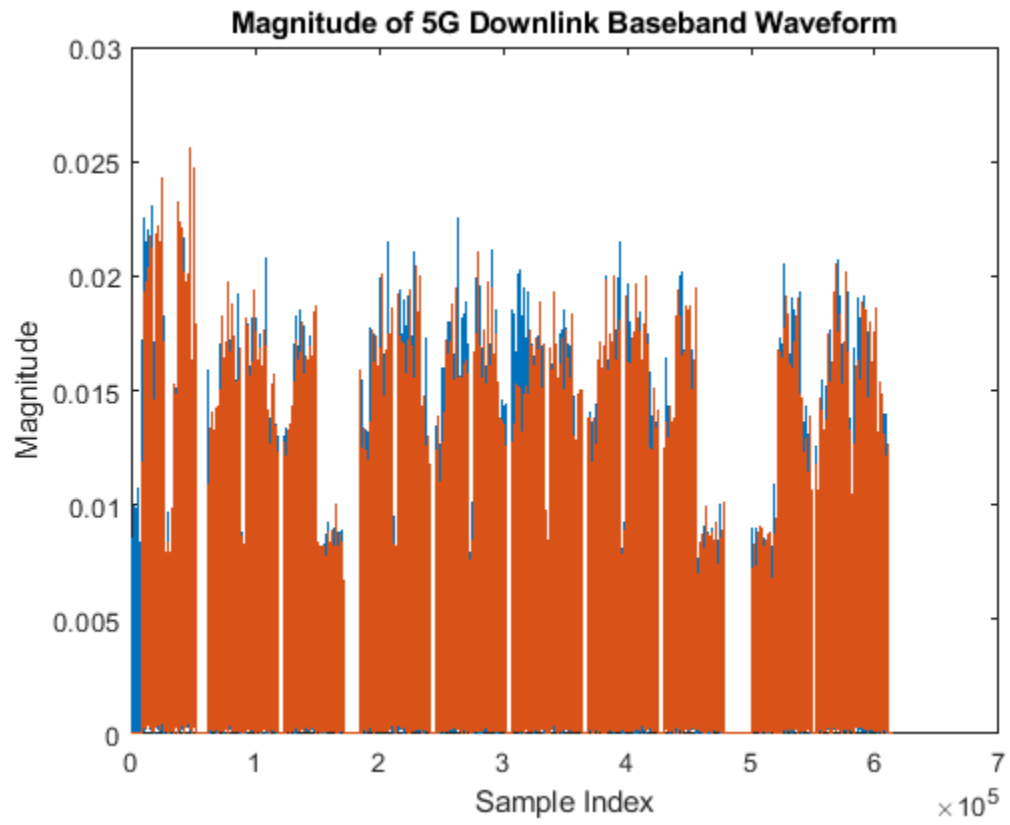
```
waveconfig.SSBurst = ssburst;
waveconfig.SCSCarriers = scscarriers;
waveconfig.BandwidthParts = bwp;
waveconfig.CORESET = coresets;
waveconfig.SearchSpaces = searchspaces;
waveconfig.PDCCH = pdcch;
waveconfig.PDSCH = pdsch;
waveconfig.CSIRS = csirs;

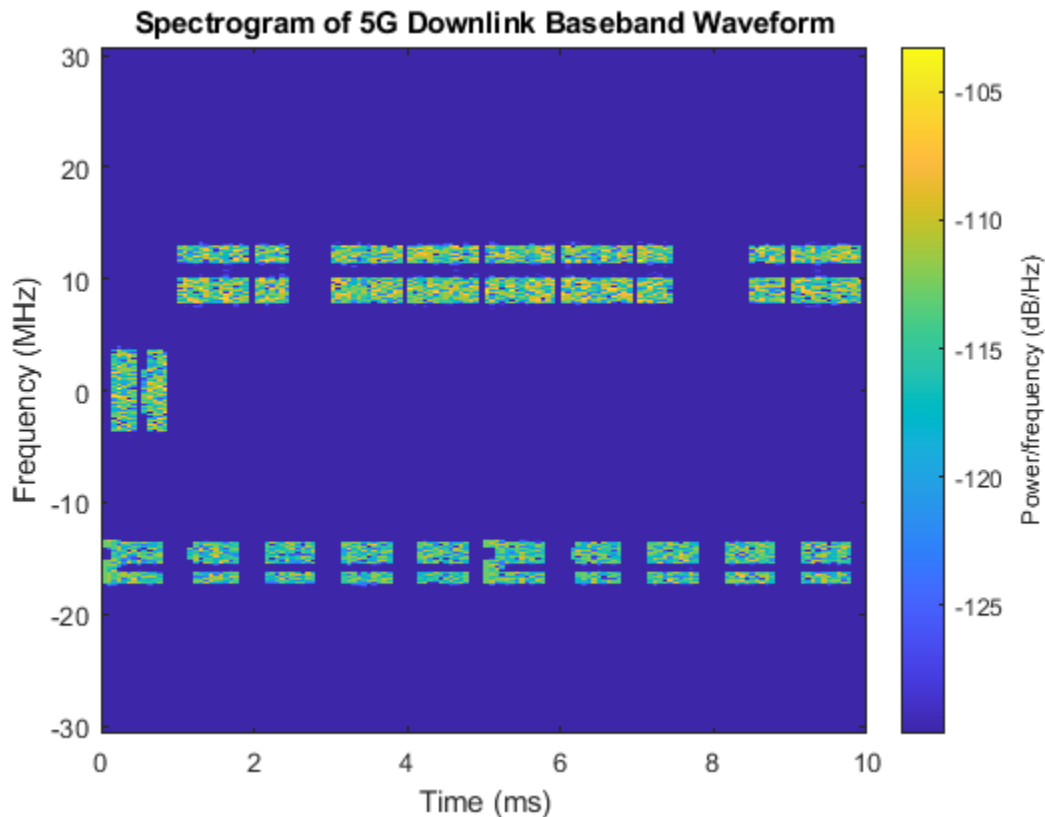
[waveform,info] = nrWaveformGenerator(waveconfig);

% Plot the magnitude of the baseband waveform for the set of antenna ports defined
figure;
plot(abs(waveform));
title('Magnitude of 5G Downlink Baseband Waveform');
xlabel('Sample Index');
ylabel('Magnitude');

% Plot spectrogram of waveform for first antenna port
samplerate = info.ResourceGrids(1).Info.SampleRate;
nfft = info.ResourceGrids(1).Info.Nfft;
figure;
spectrogram(waveform(:,1),ones(nfft,1),0,nfft,'centered',samplerate,'yaxis','MinThreshold',-130)
title('Spectrogram of 5G Downlink Baseband Waveform');
```







The waveform generator function returns the time domain waveform and a structure `info` which contains the underlying resource element grid and a breakdown of the resources used by all the PDSCH and PDCCH instances in the waveform.

The `ResourceGrids` field is structure array which contains the following fields:

- The resource grid corresponding to each BWP
- The resource grid of the overall bandwidth containing the channels and signals in each BWP
- An info structure with information corresponding to each BWP. The contents of this info structure for the first BWP are shown below.

```
disp('Modulation information associated with BWP 1:')
disp(info.ResourceGrids(1).Info)
```

```
Modulation information associated with BWP 1:
      Nfft: 4096
      SampleRate: 61440000
      CyclicPrefixLengths: [1x14 double]
      SymbolLengths: [1x14 double]
      Windowing: 0
      SymbolPhases: [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
      SymbolsPerSlot: 14
      SlotsPerSubframe: 1
      SlotsPerFrame: 10
      k0: 0
```

Note that the generated resource grid is a 3D matrix where the different planes represent the antenna ports. For the different physical channels and signals the lowest port is mapped to the first plane of the grid.

## **See Also**

### **Functions**

nrPBCH | nrPBCHDMRS | nrPDCCH | nrPDSCH | nrPSS | nrSSS

## **More About**

- “5G NR Uplink Carrier Waveform Generation” on page 2-2

## NR PDSCH Resource Allocation and DM-RS and PT-RS Reference Signals

This example shows the time-frequency aspects of the new radio (NR) physical downlink shared channel (PDSCH), the associated demodulation reference signal (DM-RS), and phase tracking reference signal (PT-RS). The example shows how PDSCH resource allocation affects the time-frequency structure of DM-RS and PT-RS.

### Introduction

In 5G NR, PDSCH is the physical downlink channel that carries user data. DM-RS and PT-RS are the reference signals associated with PDSCH. These signals are generated within the PDSCH allocation, as defined in TS 38.211 Sections 7.4.1.1 and 7.4.1.2 [1] on page 1-0 . DM-RS is used for channel estimation as part of coherent demodulation of PDSCH. To compensate for the common phase error (CPE), 3GPP 5G NR introduced PT-RS. Phase noise produced in local oscillators introduces a significant degradation at mmWave frequencies. It produces CPE and inter-carrier interference (ICI). CPE leads to an identical rotation of a received symbol in each subcarrier. ICI leads to loss of orthogonality between the subcarriers. PT-RS is used mainly to estimate and minimize the effect of CPE on system performance.

The 5G Toolbox™ provides the functions for physical (PHY) layer modeling with varying levels of granularity. The levels of granularity range from PHY channel level functions that perform the transport and physical channel processing to individual channel processing stage functions performing cyclic redundancy check (CRC) coding, code block segmentation, low density parity check (LDPC) channel coding, and so on. The toolbox offers the reference signals functionality associated with the PDSCH as functions `nrPDSCHDMRS`, `nrPDSCHDMRSIndices`, `nrPDSCHPTRS`, and `nrPDSCHPTRSIndices`.

### PDSCH

PDSCH is the physical channel that carries the user data. The resources allocated for PDSCH are within the bandwidth part (BWP) of the carrier, as defined in TS 38.214 Section 5.1.2 [2] on page 1-0 . The resources in the time domain for PDSCH transmission are scheduled by downlink control information (DCI) in the field *Time domain resource assignment*. This field indicates the slot offset  $K_0$ , starting symbol  $S$ , the allocation length  $L$ , and the mapping type of PDSCH. The valid combinations of  $S$  and  $L$  are shown in Table 1. For mapping type A, value of  $S$  is 3 only when the DM-RS type A position is set to 3.

PDSCH Mapping Type	Normal Cyclic Prefix			Extended Cyclic Prefix		
	$S$	$L$	$S+L$	$S$	$L$	$S+L$
Type A	{0,1,2,3}	{3,...,14}	{3,...,14}	{0,1,2,3}	{3,...,12}	{3,...,12}
Type B	{0,...,12}	{2,4,7}	{2,...,14}	{0,...,10}	{2,4,6}	{2,...,12}

Table 1: Valid  $S$  and  $L$  Combinations

The resources in the frequency domain for PDSCH transmission are scheduled by a DCI in the field *Frequency domain resource assignment*. This field indicates whether the resource allocation of

resource blocks (RBs) is contiguous or noncontiguous, based on the allocation type. The RBs allocated are within the BWP.

The 5G Toolbox™ provides the `nrCarrierConfig` and `nrPDSCHConfig` objects to set the parameters related to the PDSCH within the BWP.

```
% Setup the carrier with 15 kHz subcarrier spacing and 10 MHz bandwidth
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 15;
carrier.CyclicPrefix = 'normal';
carrier.NSizeGrid = 52;
carrier.NStartGrid = 0;

% Configure the physical downlink shared channel parameters
pdsch = nrPDSCHConfig;
pdsch.NSizeBWP = []; % Empty implies that the value is equal to NSizeGrid
pdsch.NStartBWP = []; % Empty implies that the value is equal to NStartGrid
pdsch.PRBSets = 0:51; % Allocate the complete carrier
pdsch.SymbolAllocation = [0 14]; % Symbol allocation [S L]
pdsch.MappingType = 'A'; % PDSCH mapping type ('A' or 'B')
```

## DM-RS

DM-RS is used to estimate the radio channel. The signal is present only in the RBs allocated for the PDSCH. The DM-RS structure is designed to support different deployment scenarios and use cases. A front-loaded design supports low-latency transmissions, twelve orthogonal antenna ports for MIMO transmissions, and up to four reference signal transmission instances in a slot to support high-speed scenarios. The front-loaded reference signals indicate that the signal occurs early in the transmission. The DM-RS is present in each RB allocated for PDSCH.

### Parameters That Control Time Domain Resources

The parameters that control DM-RS OFDM symbol locations are:

- PDSCH symbol allocation
- Mapping type
- DM-RS type A position
- DM-RS length
- DM-RS additional position

The symbol allocation of PDSCH indicates the OFDM symbol locations used by the PDSCH transmission in a slot. DM-RS symbol locations lie within the PDSCH symbol allocation. The positions of DM-RS OFDM symbols depend on the mapping type. The mapping type of PDSCH is either slot-wise (type A) or non-slot-wise (type B). The positions of any additional DM-RS symbols are defined by a set of tables, as specified in TS 38.211 Section 7.4.1.1.2 [1] on page 1-0 . For the purpose of indexing the tables, the specification defines the term  $l_d$  indicating the duration of OFDM symbols to be accounted for, depending on the mapping type.

For mapping type A, the DM-RS OFDM symbol locations are defined relative to the first OFDM symbol of the slot (symbol #0). The location of first DM-RS OFDM symbol ( $l_0$ ) is provided by the DM-RS type A position, which is either 2 or 3. For any additional DM-RS, the duration of OFDM symbols ( $l_d$ ) is the number of OFDM symbols between the first OFDM symbol of the slot (symbol #0) and the last OFDM symbol of the allocated PDSCH resources. Note that  $l_d$  may differ from the number of

OFDM symbols allocated for PDSCH, when the first OFDM symbol of PDSCH is other than symbol #0.

For mapping type B, the DM-RS OFDM symbol locations are defined relative to the first OFDM symbol of allocated PDSCH resources. The location of first DM-RS OFDM symbol ( $l_0$ ) is always 0, meaning that the first DM-RS OFDM symbol location is the first OFDM symbol location of the allocated PDSCH resources. For any additional DM-RS, the duration of OFDM symbols ( $l_d$ ) is the duration of the allocated PDSCH resources.

Figure 1 illustrates the DM-RS symbol locations depending on the mapping type for an RB within a slot, having single-symbol DM-RS. The figure shows a configuration with PDSCH occupying the OFDM symbols from 1 to 10 (0-based) with  $l_d$  equal to 11 for mapping type A, and from 3 to 9 (0-based) with  $l_d$  equal to 7 for mapping type B respectively.

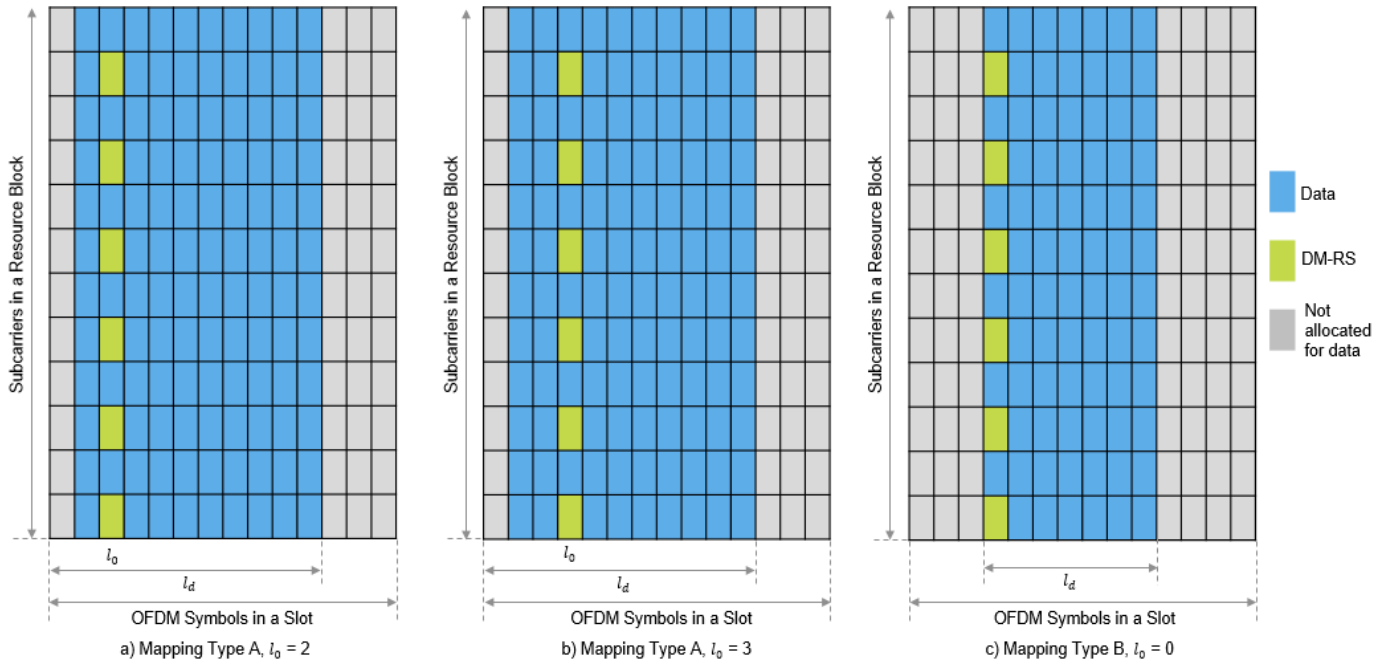


Figure 1: DM-RS Symbol Locations Based on Mapping Type

The maximum number of DM-RS OFDM symbols used by a UE is configured by RRC signaling (*dmrs-AdditionalPosition* and *maxLength*). The *maxLength* RRC parameter configures the length of DM-RS symbol, single symbol DM-RS or double symbol DM-RS. For double-symbol DM-RS, the actual selection is signaled in the DCI format 1\_1 message. Figure 2 illustrates the single-symbol and double-symbol DM-RS locations.

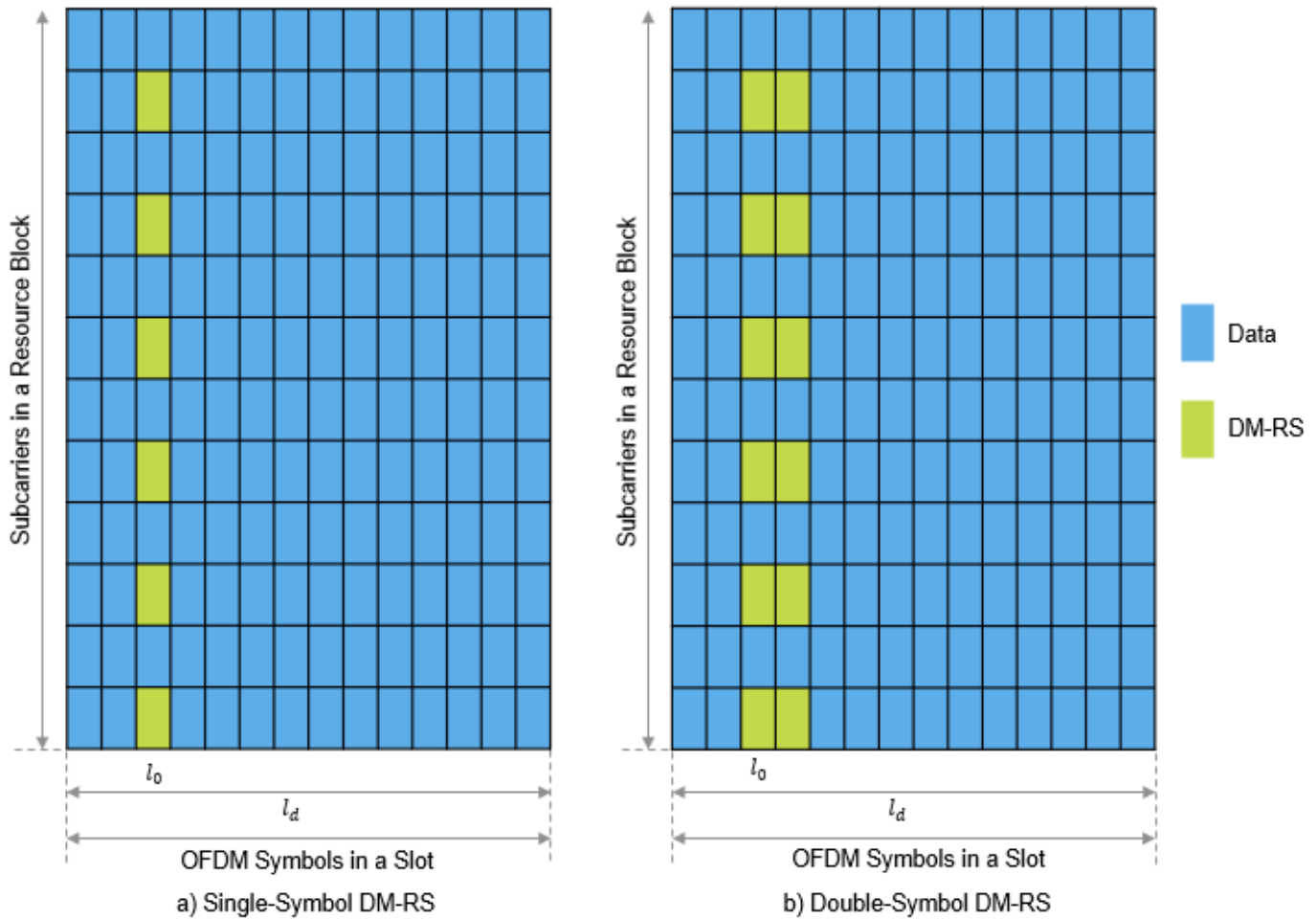


Figure 2: DM-RS Symbol Locations Based on Number of Front-Loaded DM-RS Symbols

The higher-layer parameter *dmrs-AdditionalPosition* defines the maximum number of additional single- or double-symbol DM-RS transmitted. The number of additional positions is in the range of 0 to 3 and depends on the mapping type, DM-RS length, and PDSCH symbol allocation. The DM-RS symbol locations are given by TS 38.211 Tables 7.4.1.1.2-3, and 7.4.1.1.2-4. Figure 3 illustrates the DM-RS additional positions in combination with single-symbol and double-symbol DM-RS.

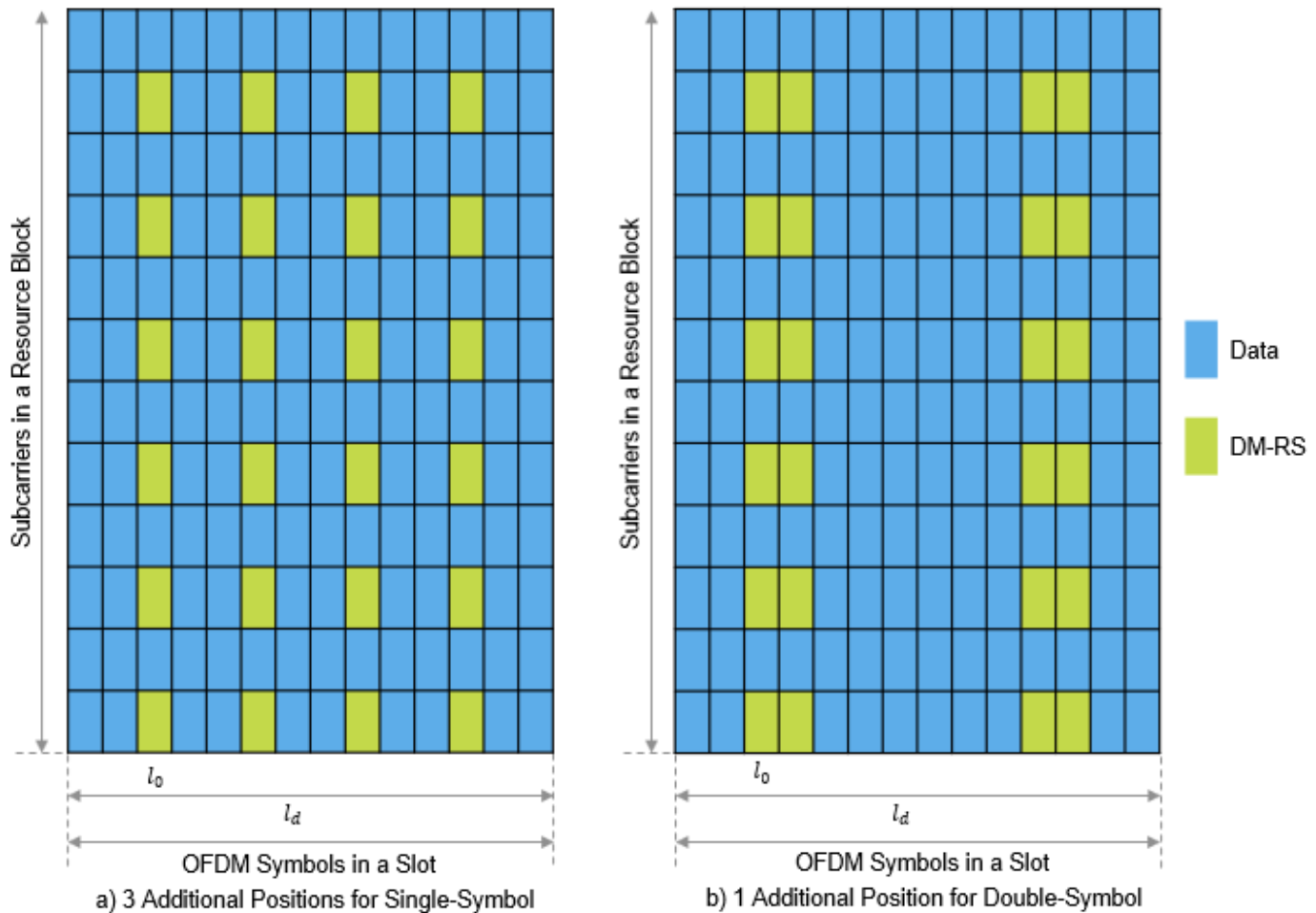


Figure 3: DM-RS Symbol Locations Based on Number of Additional DM-RS Positions

```
% Set the parameters that control the time resources of DM-RS
pdsch.DMRS.DMRSTypeAPosition = 2; % 2 or 3
pdsch.DMRS.DMRSLength = 1; % 1 or 2
pdsch.DMRS.DMRSAdditionalPosition = 1; % 0...3
```

### Parameters That Control Frequency Domain Resources

The parameters that control the subcarrier locations of DM-RS are:

- DM-RS configuration type
- DM-RS antenna ports

The configuration type indicates the frequency density of DM-RS and is signaled by RRC message *dms-Type*. Configuration type 1 defines six subcarriers per physical resource block (PRB) per antenna port, comprising alternate subcarriers. Configuration type 2 defines four subcarriers per PRB per antenna port, consisting of two groups of two consecutive subcarriers. Figure 4 indicates the DM-RS subcarrier locations based on configuration type.



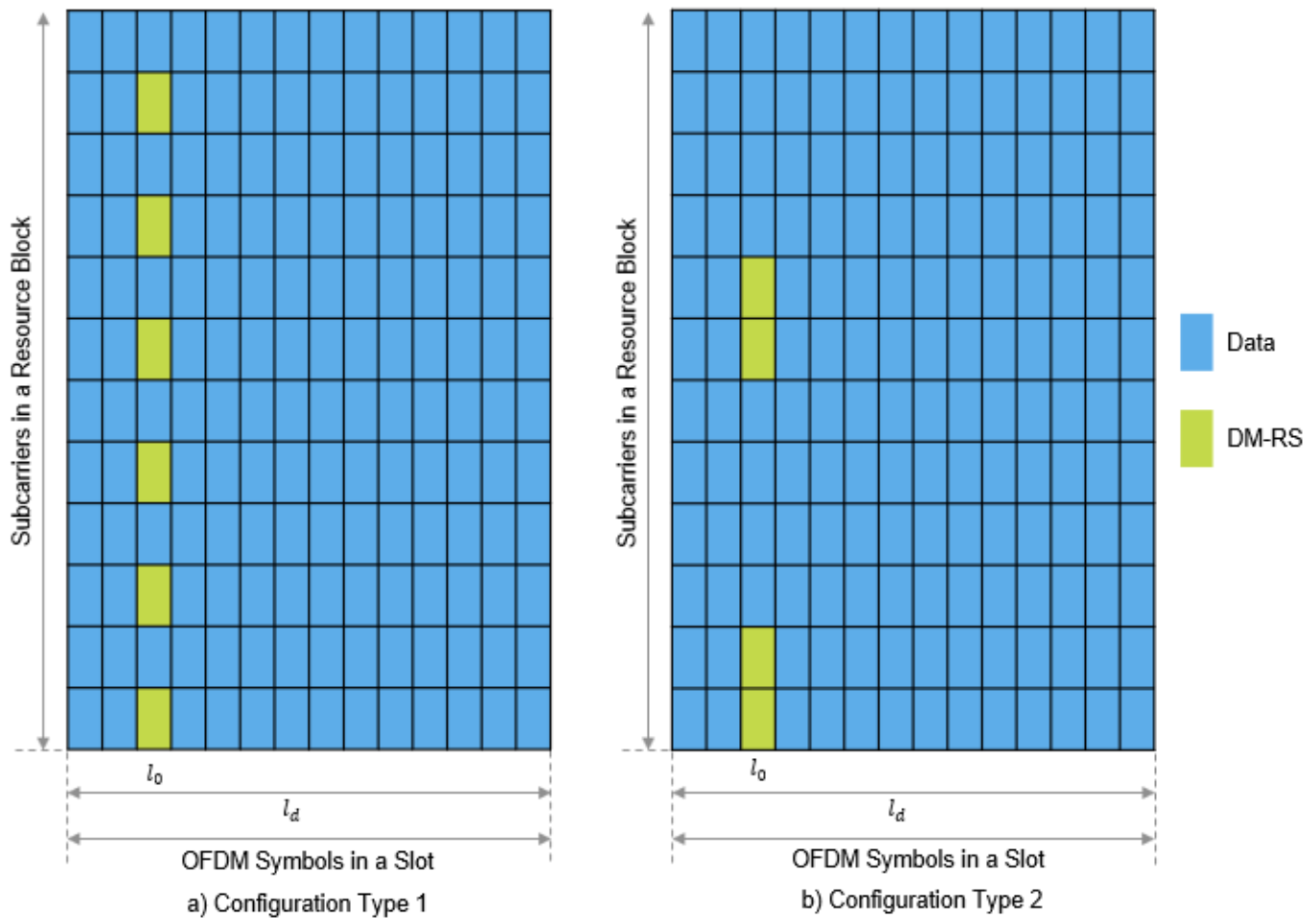


Figure 4: DM-RS Subcarrier Locations Based on DM-RS Configuration Type

Different delta shifts are applied to the sets of subcarriers used, depending on the associated antenna port or code division multiplexing (CDM) group. For configuration type 1, there are two possible CDM groups/shifts across eight possible antenna ports ( $p=0\dots7$ ). Figure 5 illustrates the different shifts associated for DM-RS subcarrier locations with the DM-RS configuration type set to 1. Notice that the resource elements (REs) corresponding to the DM-RS subcarrier locations of lower CDM group (i.e. antenna port 0) are blocked for data transmission in the antenna ports of higher CDM group (i.e. antenna port 2).

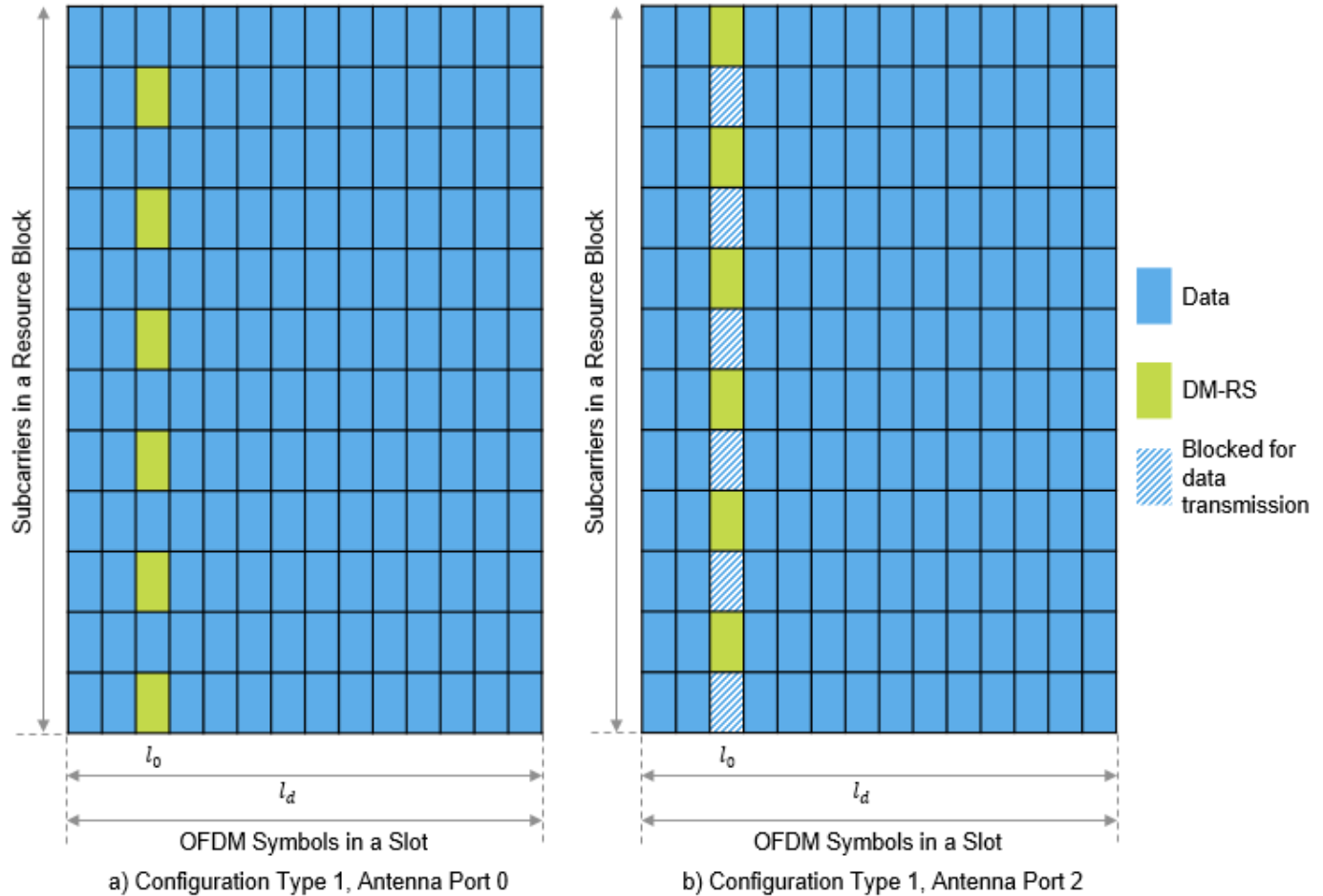


Figure 5: DM-RS Subcarrier Locations Based on DM-RS Antenna Ports for Configuration Type 1

For configuration type 2, there are three possible CDM groups/shifts across twelve antenna ports ( $p=0...11$ ). Figure 6 illustrates the different shifts associated with DM-RS subcarrier locations in DM-RS configuration type 2. For full configuration details, see TS 38.211 Section 7.4.1.1. Notice that the REs corresponding to the DM-RS subcarrier locations of lower CDM groups are blocked for data transmission in the antenna ports of higher CDM groups.

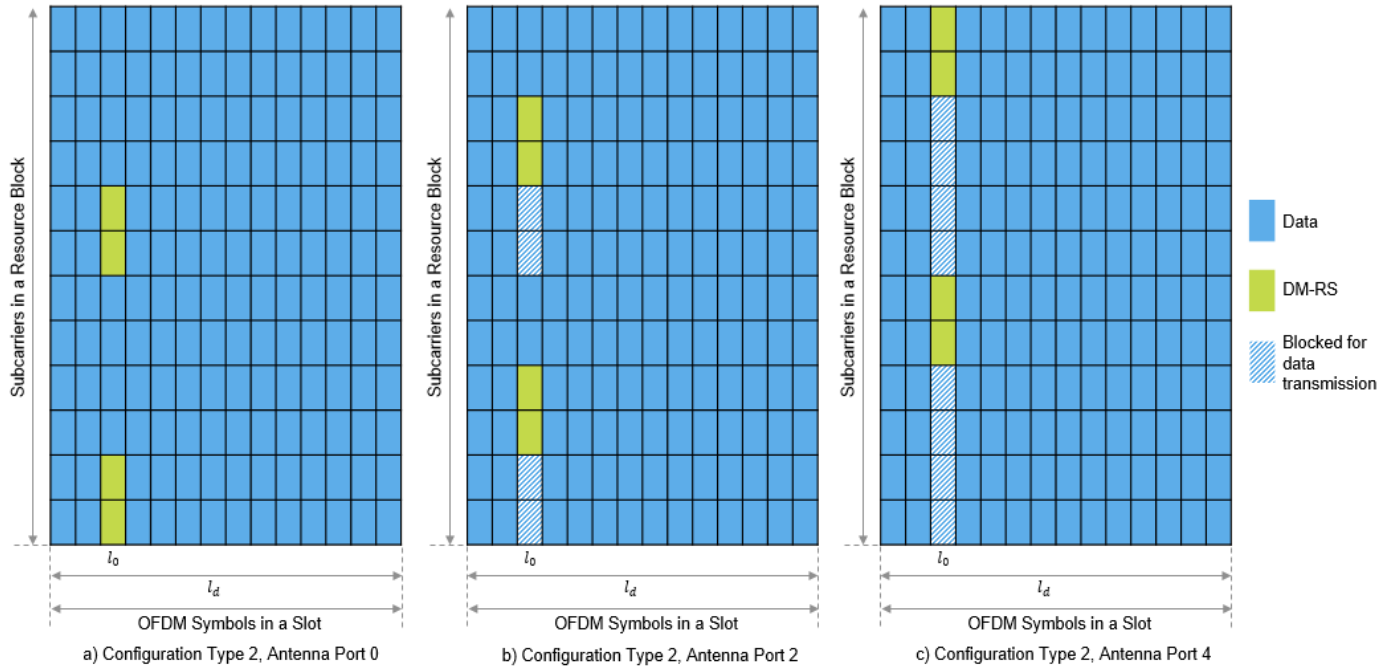


Figure 6: DM-RS Subcarrier Locations Based on DM-RS Antenna Ports for Configuration Type 2

```

% Set the parameters that control the frequency resources of DM-RS
pdsch.DMRS.DMRSConfigurationType = 1; % 1 or 2
pdsch.DMRS.DMRSPortSet = 0;

% Set the parameter that controls the number of REs available for data
% transmission in a DM-RS carrying OFDM symbol. This value is nominally
% greater than the maximum configured CDM group number.
pdsch.DMRS.NumCDMGroupsWithoutData = 1; % 1 corresponds to CDM group number 0

% The read-only properties DeltaShifts and DMRSSubcarrierLocations of DMRS
% property of pdsch object provides the values of delta shift(s) and DM-RS
% subcarrier locations in an RB for each antenna port configured.
pdsch.DMRS.DeltaShifts

ans = 0

pdsch.DMRS.DMRSSubcarrierLocations

ans = 6x1

    0
    2
    4
    6
    8
   10

```

## Sequence Generation

The pseudorandom sequence used for DM-RS is  $2^{31} - 1$  length gold sequence. The sequence is generated across all the common resource blocks (CRBs) and is transmitted only in the RBs allocated

for data because it is not required to estimate the channel outside the frequency region in which data is not transmitted. Generating the reference signal sequence across all the CRBs ensures that the same underlying pseudorandom sequence is used for multiple UEs on overlapping time-frequency resources in the case of a multi-user MIMO. The parameters that control the sequence generation are:

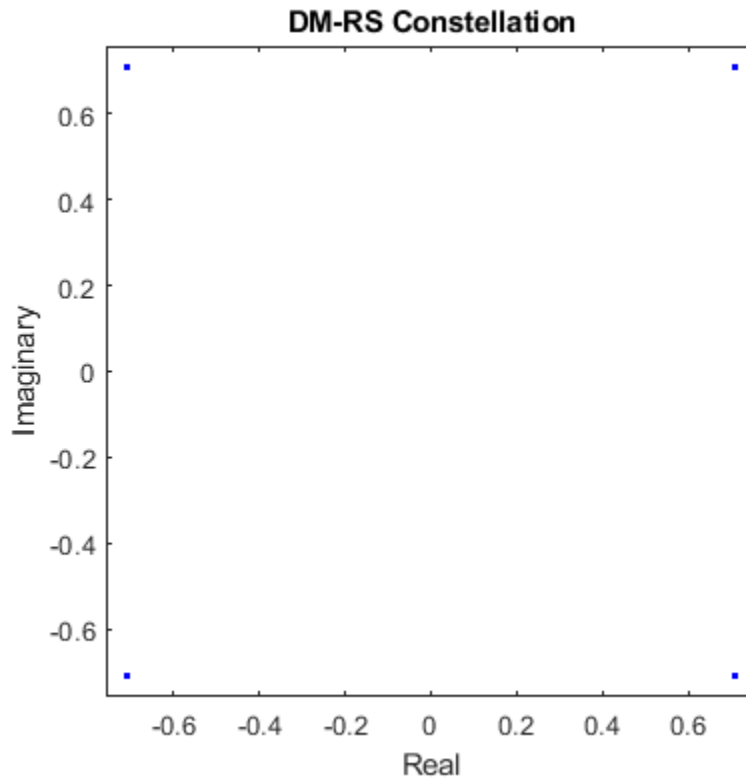
- DM-RS scrambling identity ( $N_{ID}^{n_{SCID}}$ )
- DM-RS scrambling initialization ( $n_{SCID}$ )
- Number of OFDM symbols in a slot
- Slot number in a radio frame
- DM-RS symbol locations
- PRBs allocation

The CyclicPrefix property of the carrier object controls the number of OFDM symbols in a slot. The NSlot property of the carrier object controls the slot number.

```
% Set the parameters that only control the DM-RS sequence generation
pdsch.DMRS.NIDNSCID = 1; % Use empty to set it to NCellID of the carrier
pdsch.DMRS.NSCID = 0;    % 0 or 1
```

```
% Generate DM-RS symbols
pdsch.NumLayers = numel(pdsch.DMRS.DMRSPortSet);
dmrsSymbols = nrPDSCHDMRS(carrier,pdsch);
```

```
% Plot the constellation
scatterplot(dmrsSymbols)
title('DM-RS Constellation')
xlabel('Real')
ylabel('Imaginary')
```



```
% The read-only properties TimeWeights and FrequencyWeights of DMRS
% property of pdsch object provides the values of time and frequency
% weights applied to the DM-RS symbols.
```

```
pdsch.DMRS.TimeWeights
```

```
ans = 2×1
```

```
1
1
```

```
pdsch.DMRS.FrequencyWeights
```

```
ans = 2×1
```

```
1
1
```

```
% Generate DM-RS indices
```

```
dmrsIndices = nrPDSCHDMRSIndices(carrier,pdsch);
```

```
% Map the DM-RS symbols to the grid with the help of DM-RS indices
```

```
grid = zeros([12*carrier.NSizeGrid carrier.SymbolsPerSlot pdsch.NumLayers]);
```

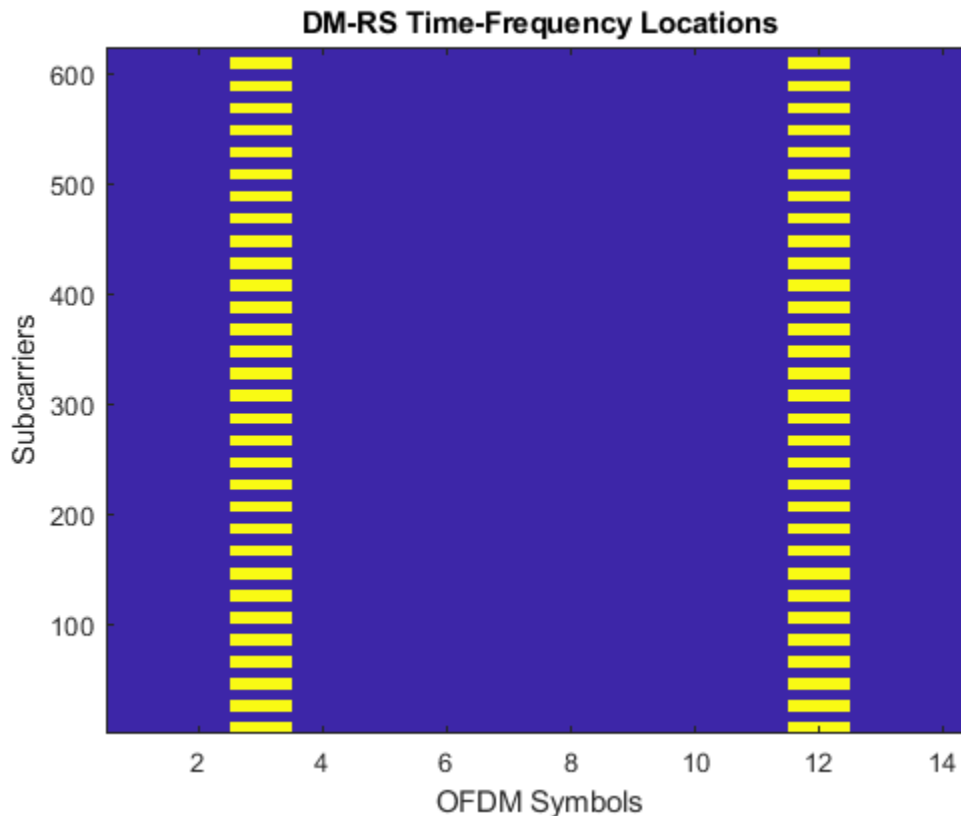
```
grid(dmrsIndices) = dmrsSymbols;
```

```
figure
```

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

```
axis xy;
```

```
xlabel('OFDM Symbols');
ylabel('Subcarriers');
title('DM-RS Time-Frequency Locations');
```



### PT-RS

PT-RS is the phase tracking reference signal. PT-RS is used mainly to estimate and minimize the effect of CPE on system performance. Due to the phase noise properties, PT-RS signal has a low density in the frequency domain and a high density in the time domain. PT-RS always occurs in combination with DM-RS and only when the network has configured PT-RS to be present.

### Parameters That Control Time Domain Resources

PT-RS is configured through the higher layer parameter *DMRS-DownlinkConfig* for downlink. The parameters that control the time resources of PT-RS are:

- DM-RS symbol locations
- Time density of PT-RS ( $L_{PT-RS}$ )

$L_{PT-RS}$  depends on the scheduled modulation and coding scheme. Value of  $L_{PT-RS}$  must be from the set {1, 2, 4}. For the parameters that control DM-RS symbol locations, refer to Parameters that Control DM-RS Time Resources on page 1-0 .

The PT-RS symbol locations in a slot start from the first OFDM symbol in the shared channel allocation and hop every  $L_{PT-RS}$  symbols, if no DM-RS symbol is present in this interval. In the case where a DM-RS symbol or symbols are present in between or at the hop interval, the hop starts from

the last DM-RS symbol location to provide the next PT-RS symbol. Figure 7 shows the PT-RS symbol locations in an RB for single slot with time-density set to 4 and DM-RS symbol locations set to 2 and 11 (0-based).

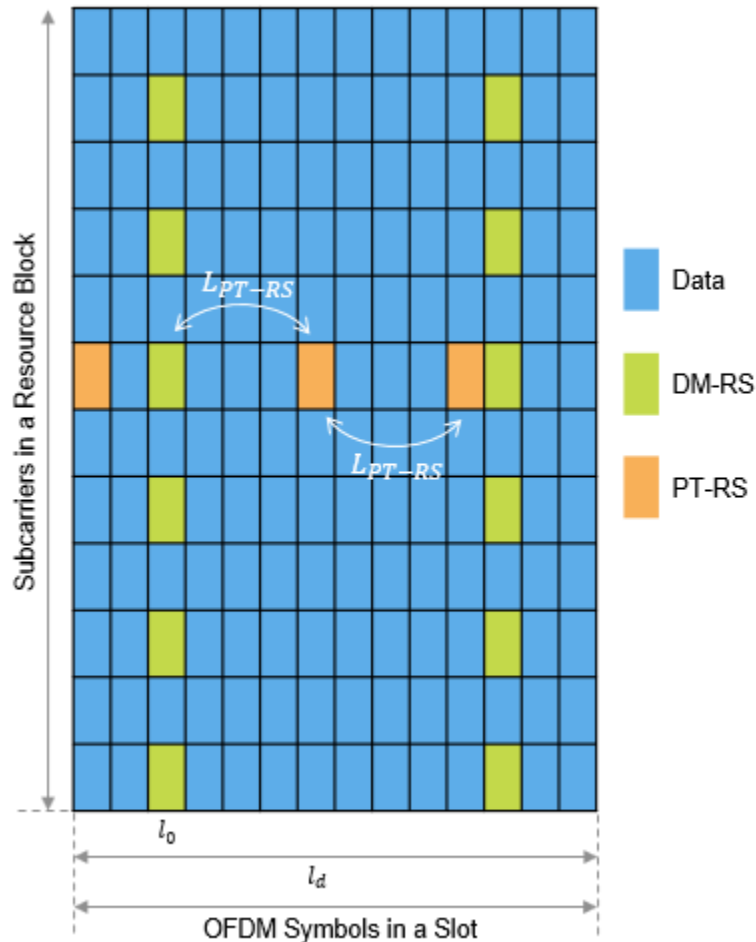


Figure 7: PT-RS Symbol Locations

```
% Set the EnablePTRS property in pdsch to 1
pdsch.EnablePTRS = 1;
```

```
% Set the parameters that control the time resources of PT-RS
pdsch.PTRS.TimeDensity = 4;
```

### Parameters That Control Frequency Domain Resources

PT-RS occupies only one subcarrier in an RB for one OFDM symbol. The parameters that control the frequency resources of PT-RS are:

- PRB allocation
- DM-RS configuration type
- Frequency density of PT-RS ( $K_{PT-RS}$ )
- Radio network temporary identifier ( $n_{RNTI}$ )

- Resource element offset
- PT-RS antenna port

$K_{PT-RS}$  depends on the scheduled bandwidth. The value of  $K_{PT-RS}$  is either 2 or 4, which indicates whether PT-RS is present in every two RBs or every four RBs.

The starting RB at which PT-RS is present ( $k_{ref}^{RB}$ ), depends on  $K_{PT-RS}$ ,  $n_{RNTI}$ , and the number of RBs ( $N_{RB}$ ) allocated for PDSCH. For the purpose of mapping PT-RS, all the RBs of PDSCH are numbered in increasing order from 0 to  $N_{RB} - 1$ . The subcarrier location of PT-RS ( $k_{ref}^{RE}$ ) within a resource block depends on the DM-RS configuration type, resource element (RE) offset, and PT-RS antenna port. The PT-RS antenna port must be a subset of DM-RS antenna ports. The PT-RS subcarrier location always aligns with one of the DM-RS subcarrier locations in an RB.

PT-RS in an RB occupies the same subcarrier locations in all the OFDM symbols where PT-RS is present.

```
% Set the parameters that affect the PT-RS subcarrier locations
```

```
pdsch.RNTI = 1;
pdsch.PTRS.FrequencyDensity = 2; % 2 or 4
pdsch.PTRS.REOffset = '10'; % '00', '01', '10', '11'
pdsch.PTRS.PTRSPortSet = min(pdsch.DMRS.DMRSPortSet);
```

```
% Set the other parameters that control PT-RS subcarrier locations
```

```
pdsch.DMRS.DMRSConfigurationType = 1;
pdsch.DMRS.DMRSPortSet = 0;
```

### Sequence Generation

The sequence used to generate PT-RS is the same pseudorandom sequence used for DM-RS sequence generation. The values of the PT-RS sequence depend on the position of the first DM-RS symbol. For more details, refer to DM-RS sequence generation on page 1-0 .

```
% Set the parameters that control the PT-RS sequence generation
```

```
pdsch.DMRS.NIDNSCID = 1; % Use empty to set it to NCellID of the carrier
pdsch.DMRS.NSCID = 0; % 0 or 1
```

```
% Generate PT-RS symbols
```

```
carrier.NSizeGrid = 4;
pdsch.PRBSet = 0:carrier.NSizeGrid-1;
pdsch.NumLayers = numel(pdsch.DMRS.DMRSPortSet);
ptrsSymbols = nrPDSCHPTRS(carrier,pdsch);
```

```
% Generate PT-RS indices
```

```
ptrsIndices = nrPDSCHPTRSIndices(carrier,pdsch);
```

Get DM-RS symbols, RE indices of PDSCH, and DM-RS.

```
% PDSCH indices, DM-RS symbols and indices
```

```
[pdschIndices, pdschInfo] = nrPDSCHIndices(carrier,pdsch);
dmrsIndices = nrPDSCHDMRSIndices(carrier,pdsch);
dmrsSymbols = nrPDSCHDMRS(carrier,pdsch);
```

Map PDSCH, DM-RS, and PT-RS RE indices to the grid with scaled values to visualize the respective locations on the grid.

```
chpLevel = struct;
chpLevel.PDSCH = 0.4;
```

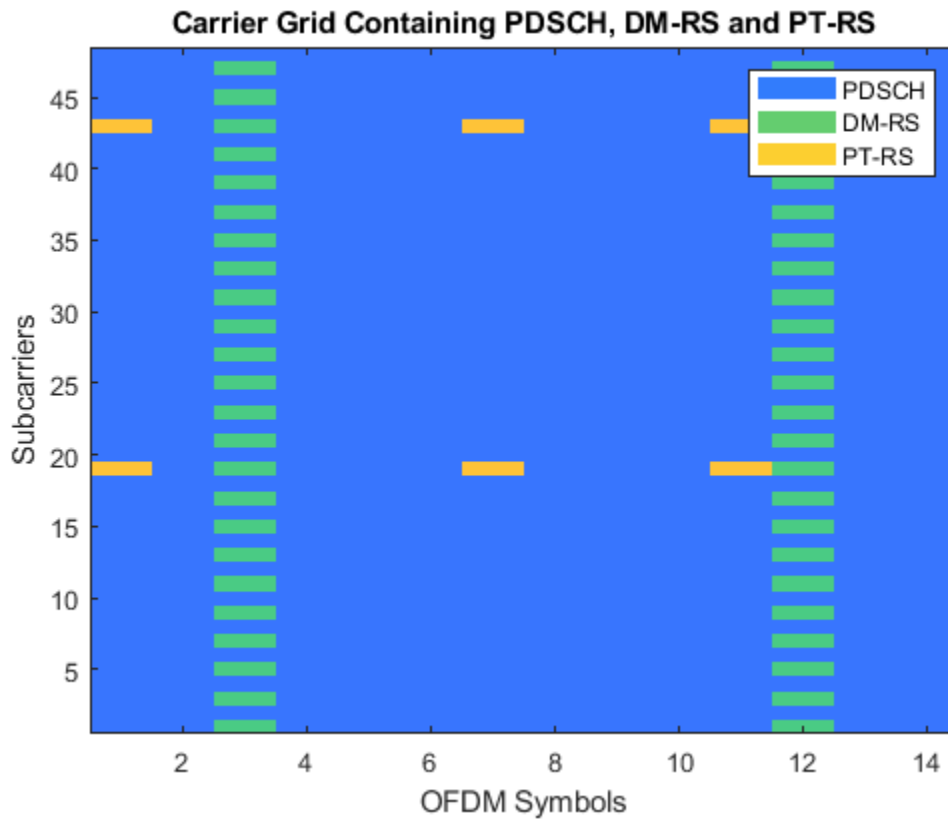


```

chpLevel.DMRS = 1;
chpLevel.PTRS = 1.4;
nSlotSymb = carrier.SymbolsPerSlot;
grid = complex(zeros(carrier.NSizeGrid*12,nSlotSymb,pdsch.NumLayers));
grid(pdschIndices) = chpLevel.PDSCH;
grid(dmrsIndices) = chpLevel.DMRS*dmrsSymbols;
grid(ptrsIndices) = chpLevel.PTRS*ptrsSymbols;
map = parula(64);
cscaling = 40;
im = image(cscaling*abs(grid(:,:,1)));
colormap(im.Parent,map);

% Add legend to the image
chpval = struct2cell(chpLevel);
clevels = cscaling*[chpval{:}];
N = length(clevels);
L = line(ones(N),ones(N), 'LineWidth',8); % Generate lines
% Index the color map and associated the selected colors with the lines
set(L,{'color'},mat2cell(map( min(1+clevels,length(map) ),:),ones(1,N),3)); % Set the colors acco
% Create legend
fnames = {'PDSCH','DM-RS','PT-RS'};
legend(fnames{:});
axis xy
title('Carrier Grid Containing PDSCH, DM-RS and PT-RS')
xlabel('OFDM Symbols')
ylabel('Subcarriers')

```



In the preceding figure, PT-RS is located at the start of the OFDM symbol in the PDSCH allocation. The symbols are present at every  $L_{PT-RS}$  hop interval from each other or from DM-RS symbols. PT-RS symbols in the frequency domain are located at subcarrier 19 (first RB) and at subcarrier 43 (third RB) of each OFDM symbol where PT-RS is present. The difference in consecutive subcarrier locations of PT-RS is 24, which is the number of subcarriers in an RB (12) times the frequency density of PT-RS (2).

### Further Exploration

You can try changing the parameters that affect the time and frequency resources of reference signals and observe the variations in the RE positions for the respective signals.

Try performing channel estimation and phase tracking by using the reference signals. Compute the throughput by following the steps outlined in “NR PDSCH Throughput” on page 1-56.

This example shows how to generate the DM-RS and PT-RS sequences, and how to map the sequences to the OFDM carrier resource grid. It highlights the properties that control the time-frequency structure of reference signals.

### References

- 1 3GPP TS 38.211. "NR; Physical channels and modulation (Release 15)." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
- 2 3GPP TS 38.214. "NR; Physical layer procedures for data (Release 15)." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
- 3 3GPP TS 38.212. "NR; Multiplexing and channel coding (Release 15)." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

### See Also

#### Functions

`nrPDSCHDMRS` | `nrPDSCHDMRSIndices` | `nrPDSCHIndices` | `nrPDSCHPTRS` | `nrPDSCHPTRSIndices`

#### Objects

`nrCarrierConfig` | `nrPDSCHConfig`

### Related Examples

- “NR PUSCH Resource Allocation and DM-RS and PT-RS Reference Signals” on page 2-16

## NR Cell Search and MIB and SIB1 Recovery

This example demonstrates how to use 5G Toolbox™ to synchronize, demodulate, and decode a live gNodeB signal. The example decodes the master information block (MIB) and the first of the system information blocks (SIB1). Decoding MIB and SIB1 requires a comprehensive receiver, capable of demodulating and decoding the majority of the downlink channels and signals.

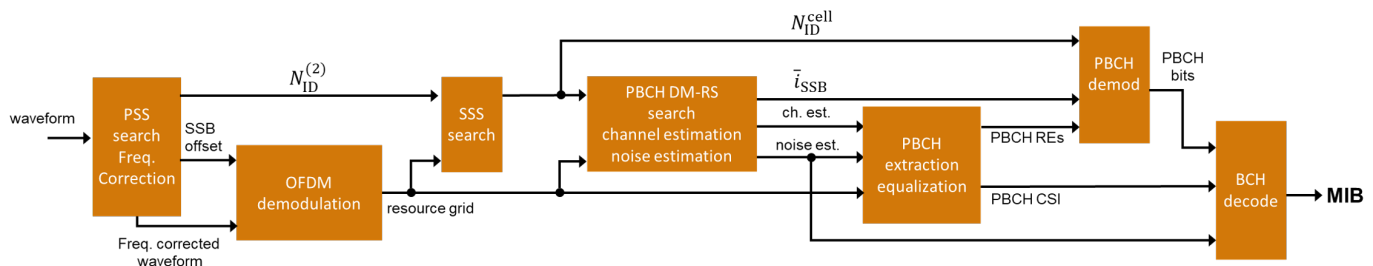
### Introduction

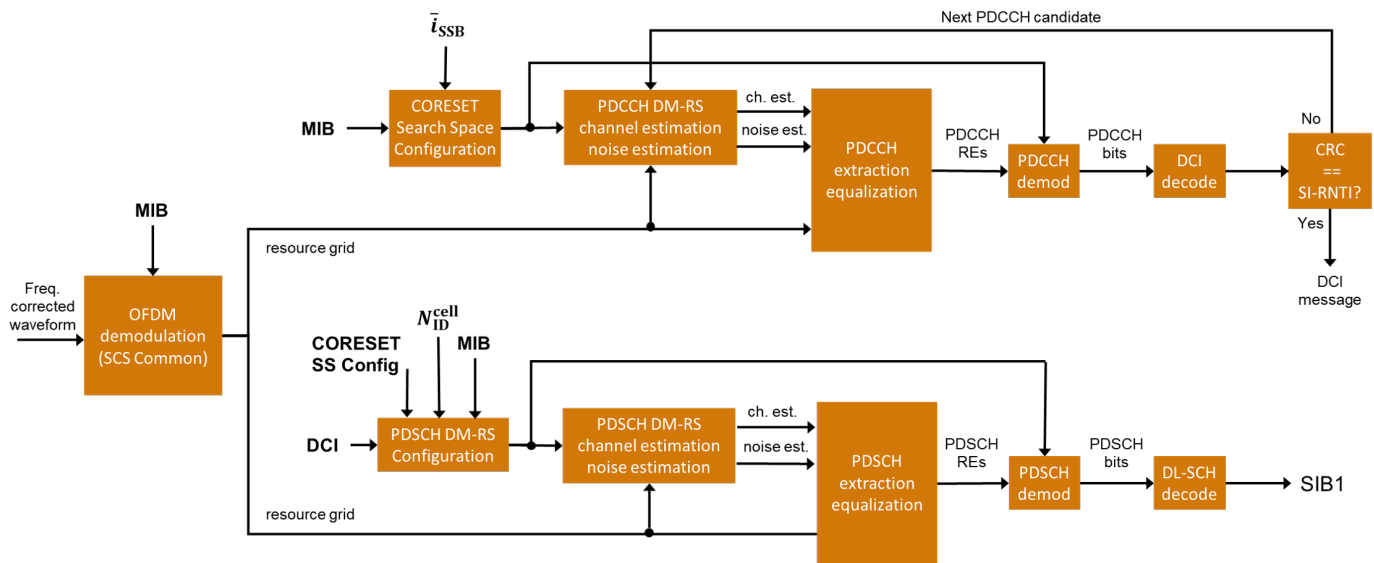
Before user equipment (UE) can communicate with the network, it must perform cell search and selection procedures and obtain initial system information. The first steps in that process are acquiring frame synchronization, finding out the cell identity and decoding the MIB and SIB1. This example shows how to perform these steps by using 5G Toolbox.

You can use this example with a captured waveform of I/Q samples or generate a local waveform containing a synchronization signal (SS) burst and SIB1 using `nrWaveformGenerator`. For locally generated waveforms, the example performs these steps:

- *Waveform generation:* Configure and generate a synchronization signal burst carrying the MIB, CORESET0, PDCCH and PDSCH carrying SIB1 by using the downlink waveform generator from 5G Toolbox. The transmitter can enhance the SNR of one SS block, but it does not perform beamforming. For more information on SSB beamforming, see “NR SSB Beam Sweeping” on page 1-95.
- *AWGN:* Apply additive white Gaussian noise (AWGN) to the waveform.
- *Receiver:* Apply various synchronization and demodulation processes to the received waveform to establish the system frame number, cell identity and SSB, and decode the MIB. These provide the information required for blind decoding of downlink control information (DCI) in a PDCCH. The receiver uses DCI to configure the PDSCH demodulator, decode DL-SCH and finally recover the SIB1.

These figures show the processing steps inside the receiver.





## Receiver Configuration

To synchronize and demodulate the received waveform, this information is needed:

- The waveform sample rate to demodulate the received waveform.
- The carrier center frequency to apply symbol phase compensation to the received waveform.
- The minimum channel bandwidth to determine CORESET0 frequency resources. TS 38.101-1 Table 5.3.5-1 [ 1 ] describes the channel bandwidths for each NR band.
- The SS block pattern (Case A...E) to determine the subcarrier spacing of the SS/PBCH blocks. UE searches for SS block patterns based on the NR operating band. For more information, see TS 38.104 Tables 5.4.3.3-1 and 5.4.3.3-2 [ 2 ].
- The number of SS/PBCH blocks in a burst ( $L_{max}$ ) to calculate parameters of the PBCH DM-RS sequences and PBCH descrambling. These parameters depend on the SS/PBCH block index as described in TS 38.211 Sections 7.3.3.1 and 7.4.1.4.1 [ 3 ]. TS 38.213 Section 4.1 [ 5 ] describes the set of SS/PBCH blocks in a burst in each case. UE knows the value of  $L_{max}$  based on the SS block pattern and the NR operating band.

```
loadFromFile = 0; % Set to 1 to load a captured waveform
```

```
if loadFromFile
    % Load captured waveform
    rx = load('capturedWaveformSIB1.mat');
    rxWaveform = rx.waveform;

    % Configure receiver sample rate (samples/second)
    rxSampleRate = rx.sampleRate;

    % Symbol phase compensation frequency. Specify the carrier center
    % frequency or set to 0 to disable symbol phase compensation
    fPhaseComp = rx.fPhaseComp; % Carrier center frequency (Hz)

    % Set the minimum channel bandwidth for the NR band required to
    % configure CORESET0 in FR1 (See TS 38.101-1 Table 5.3.5-1)
    minChannelBW = rx.minChannelBW; % 5, 10, 40 MHz
```

```

% Configure necessary burst parameters at the receiver. The SSB pattern
% can be 'Case A','Case B','Case C' for FR1 or 'Case D','Case E' for
% FR2. The maximum number of blocks L_max can be 4 or 8 for FR1 and 64
% for FR2.
refBurst.BlockPattern = rx.ssbBlockPattern;
refBurst.L_max = rx.L_max;
else
% Generate waveform containing SS burst and SIB1
% Configure the cell identity
config = struct();
config.NCellID = 102;

% Configure an SS burst
config.BlockPattern = 'Case B';           % FR1: 'Case A','Case B','Case C'. FR2: 'Case D','Case E'
config.TransmittedBlocks = ones(1,8);    % Bitmap of SS blocks transmitted
config.SubcarrierSpacingCommon = 15;     % SIB1 subcarrier spacing in kHz (15 or 30 for FR1. 0 for FR2)
config.EnableSIB1 = 1;                   % Set to 0 to disable SIB1

% Set the minimum channel bandwidth for the NR band required to
% configure CORESET0 in FR1 (See TS 38.101-1 Table 5.3.5-1)
config.MinChannelBW = 5; % 5, 10, 40 MHz

% Configure and generate a waveform containing an SS burst and SIB1
wavegenConfig = hSIB1WaveformConfiguration(config);
[txWaveform,waveInfo] = nrWaveformGenerator(wavegenConfig);
txOfdmInfo = waveInfo.ResourceGrids(1).Info;

% Introduce a beamforming gain by boosting the SNR of one SSB and
% associated SIB1 PDCCH and PDSCH
ssbIdx = 0; % Index of the SSB to boost (0-based)
boost = 6; % SNR boost in dB
txWaveform = hSIB1Boost(txWaveform,wavegenConfig,waveInfo,ssbIdx,boost);

% Add white Gaussian noise to the waveform
rng('default'); % Reset the random number generator
SNRdB = 20; % SNR for AWGN
rxWaveform = awgn(txWaveform,SNRdB-boost,-10*log10(double(txOfdmInfo.Nfft)));

% Configure receiver
% Sample rate
rxSampleRate = txOfdmInfo.SampleRate;

% Symbol phase compensation frequency (Hz). The function
% nrWaveformGenerator does not apply symbol phase compensation to the
% generated waveform.
fPhaseComp = 0; % Carrier center frequency (Hz)

% Minimum channel bandwidth (MHz)
minChannelBW = config.MinChannelBW;

% Configure necessary burst parameters at the receiver
refBurst.BlockPattern = config.BlockPattern;
refBurst.L_max = numel(config.TransmittedBlocks);
end

% Get OFDM information from configured burst and receiver parameters
nrbSSB = 20;

```

```
scsSSB = hSSBurstSubcarrierSpacing(refBurst.BlockPattern);
rxOfdmInfo = nrOFDMInfo(nrbSSB,scsSSB,'SampleRate',rxSampleRate);
```

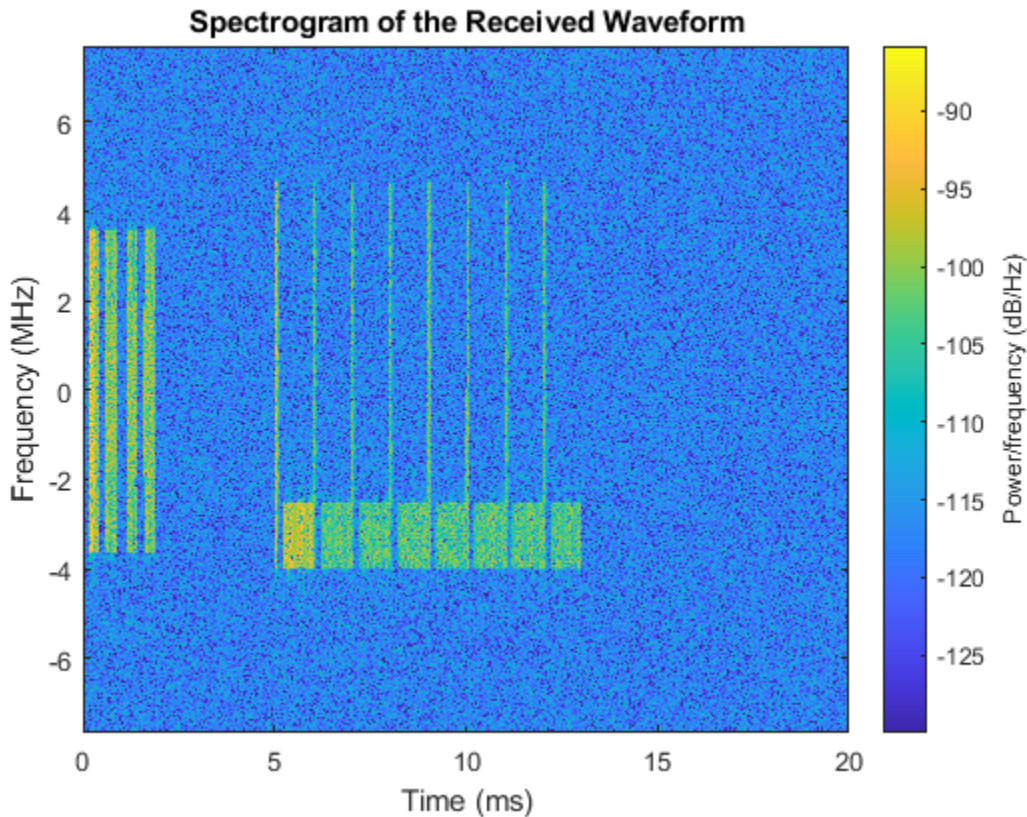
```
% Display spectrogram of received waveform
```

```
figure;
```

```
nfft = rxOfdmInfo.Nfft;
```

```
spectrogram(rxWaveform(:,1),ones(nfft,1),0,nfft,'centered',rxSampleRate,'yaxis','MinThreshold',-125);
```

```
title('Spectrogram of the Received Waveform')
```



### PSS Search and Frequency Offset Correction

The receiver performs PSS search and coarse frequency offset estimation following these steps:

- Frequency shift the received waveform with a candidate frequency offset. Candidate offsets are spaced half subcarrier apart. Use `searchBW` to control the frequency offset search bandwidth.
- Correlate the frequency-shifted received waveform with each of the three possible PSS sequences (NID2) and extract the strongest correlation peak. The reference PSS sequences are centered in frequency. Therefore, the strongest correlation peak provides a measure of coarse frequency offset with respect to the center frequency of the carrier. The peak also indicates which of the three PSS (NID2) has been detected in the received waveform and the time instant of the best channel conditions.
- Estimate frequency offsets below half subcarrier by correlating the cyclic prefix of each OFDM symbol in the SSB with the corresponding useful parts of the OFDM symbols. The phase of this correlation is proportional to the frequency offset in the waveform.

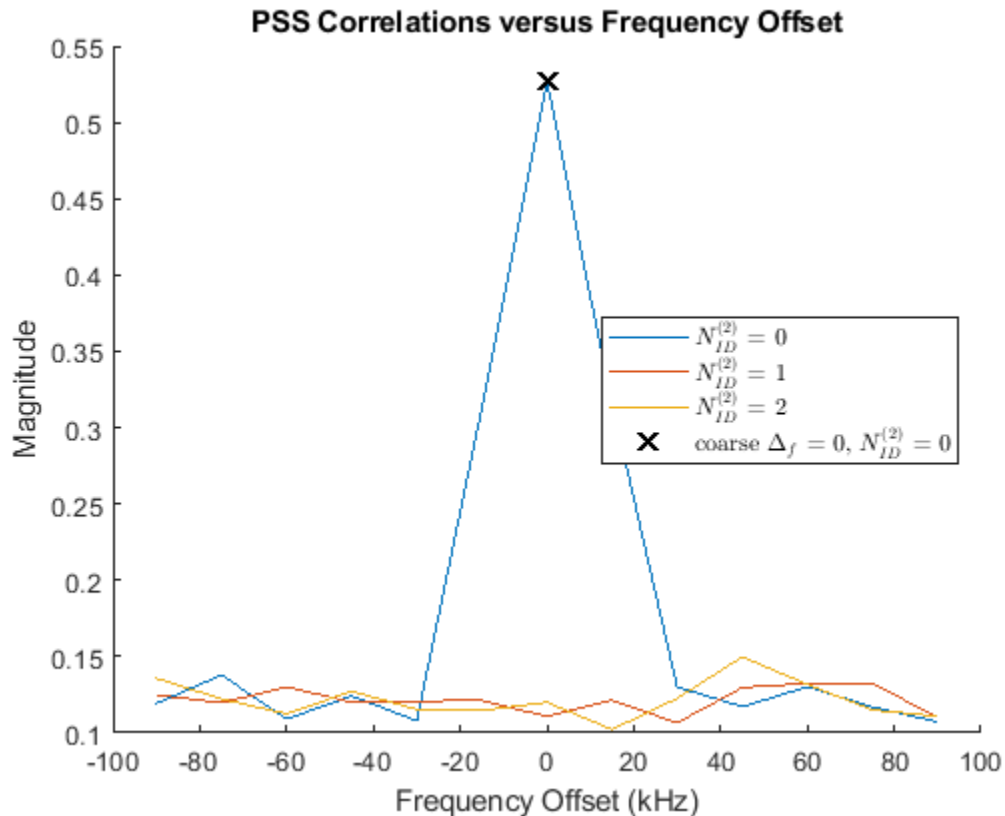
```

disp(' -- Frequency correction and timing estimation --')

% Specify the frequency offset search bandwidth in kHz
searchBW = 6*scsSSB;
[rxWaveform,freqOffset,NID2] = hSSBurstFrequencyCorrect(rxWaveform,refBurst.BlockPattern,rxSampleRate);
disp([' Frequency offset: ' num2str(freqOffset,'%0f') ' Hz'])

-- Frequency correction and timing estimation --
Frequency offset: 65 Hz

```



### Time Synchronization and OFDM Demodulation

The receiver estimates the timing offset to the strongest SS block by using the reference PSS sequence detected in the frequency offset search process. After frequency offset correction, the receiver can assume that the center frequencies of the reference PSS and received waveform are aligned. Finally, the receiver OFDM demodulates the synchronized waveform and extracts the SS block.

```

% Create a reference grid for timing estimation using detected PSS. The PSS
% is placed in the second OFDM symbol of the reference grid to avoid the
% special CP length of the first OFDM symbol.
refGrid = zeros([nrSSB*12 2]);
refGrid(nrPSSIndices,2) = nrPSS(NID2); % Second OFDM symbol for correct CP length

% Timing estimation. This is the timing offset to the OFDM symbol prior to
% the detected SSB due to the content of the reference grid
nSlot = 0;
timingOffset = nrTimingEstimate(rxWaveform,nrbSSB,scsSSB,nSlot,refGrid,'SampleRate',rxSampleRate);

```

```

% Synchronization, OFDM demodulation, and extraction of strongest SS block
rxGrid = nrOFDMDemodulate(rxWaveform(1+timingOffset:end,:),nrbSSB,scsSSB,nSlot,'SampleRate',rxSampleRate);
rxGrid = rxGrid(:,2:5,:);

srRatio = rxSampleRate/(scsSSB*1e3*rxOfdmInfo.Nfft);
firstSymbolLength = rxOfdmInfo.SymbolLengths(1)*srRatio;
str = sprintf(' Time offset to synchronization block: %.0f samples (%%.%.0ff ms) \n',floor(log10(1e3*srRatio)));
fprintf(str, timingOffset+firstSymbolLength, (timingOffset+firstSymbolLength)/rxSampleRate*1e3);

    Time offset to synchronization block: 2200 samples (0.1432 ms)

```

### SSS Search

The receiver extracts the resource elements associated to the SSS from the received grid and correlates them with each possible SSS sequence generated locally. The indices of the strongest PSS and SSS sequences combined give the physical layer cell identity, which is required for PBCH DM-RS and PBCH processing.

```

% Extract the received SSS symbols from the SS/PBCH block
sssIndices = nrSSSIndices;
sssRx = nrExtractResources(sssIndices,rxGrid);

% Correlate received SSS symbols with each possible SSS sequence
sssEst = zeros(1,336);
for NID1 = 0:335

    ncellid = (3*NID1) + NID2;
    sssRef = nrSSS(ncellid);
    sssEst(NID1+1) = sum(abs(mean(sssRx .* conj(sssRef),1)).^2);

end

% Plot SSS correlations
figure;
stem(0:335,sssEst,'o');
title('SSS Correlations (Frequency Domain)');
xlabel('$N_{ID}^{(1)}$', 'Interpreter', 'latex');
ylabel('Magnitude');
axis([-1 336 0 max(sssEst)*1.1]);

% Determine NID1 by finding the strongest correlation
NID1 = find(sssEst==max(sssEst)) - 1;

% Plot selected NID1
hold on;
plot(NID1,max(sssEst),'kx','LineWidth',2,'MarkerSize',8);
legend(["correlations" "$N_{ID}^{(1)}$ = " + num2str(NID1)], 'Interpreter', 'latex');

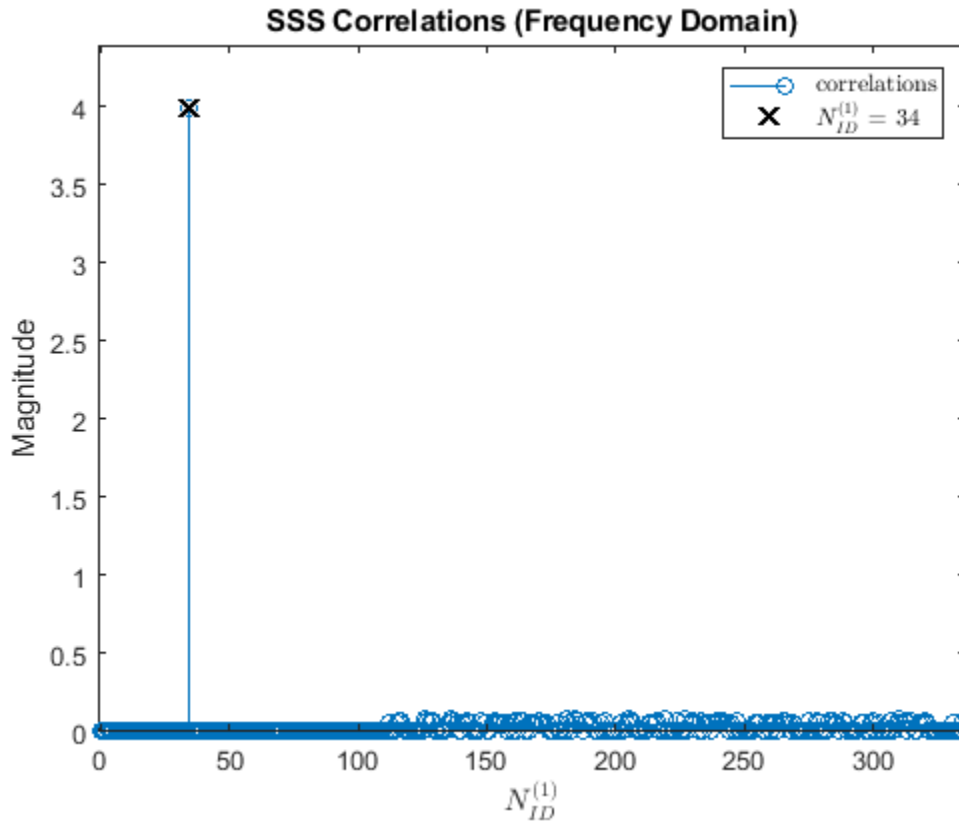
% Form overall cell identity from estimated NID1 and NID2
ncellid = (3*NID1) + NID2;

disp([' Cell identity: ' num2str(ncellid)])

    Cell identity: 102

```





### PBCH DM-RS search

In a process similar to SSS search, the receiver constructs each possible PBCH DM-RS sequence and performs channel and noise estimation. The index of the PBCH DM-RS with the best SNR determines the LSBs of the SS/PBCH block index required for PBCH scrambling initialization.

```
% Calculate PBCH DM-RS indices
dmrsIndices = nrPBCHDMRSIndices(ncellid);

% Perform channel estimation using DM-RS symbols for each possible DM-RS
% sequence and estimate the SNR
dmrsEst = zeros(1,8);
for ibar_SSB = 0:7

    refGrid = zeros([240 4]);
    refGrid(dmrsIndices) = nrPBCHDMRS(ncellid,ibar_SSB);
    [hest,nest] = nrChannelEstimate(rxGrid,refGrid,'AveragingWindow',[0 1]);
    dmrsEst(ibar_SSB+1) = 10*log10(mean(abs(hest(:).^2)) / nest);

end

% Plot PBCH DM-RS SNRs
figure;
stem(0:7,dmrsEst,'o');
title('PBCH DM-RS SNR Estimates');
xlabel('$\overline{i}_{SSB}$','Interpreter','latex');
xticks(0:7);
```

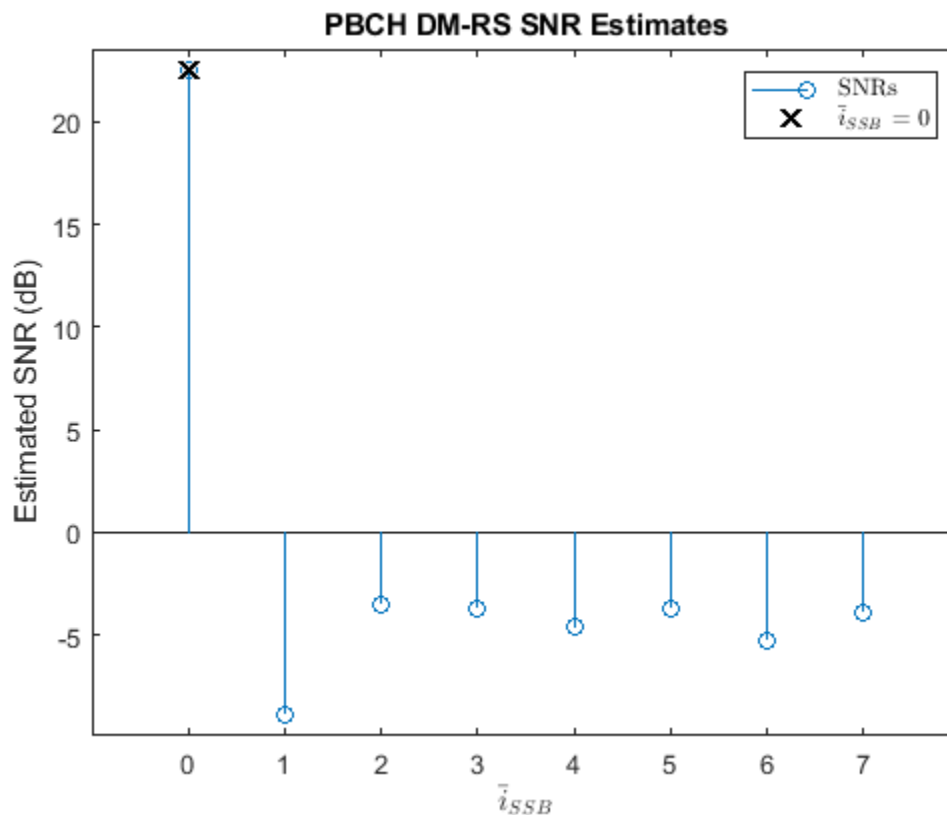
```

ylabel('Estimated SNR (dB)');
axis([-1 8 min(dmrsEst)-1 max(dmrsEst)+1]);

% Record ibar_SSB for the highest SNR
ibar_SSB = find(dmrsEst==max(dmrsEst)) - 1;

% Plot selected ibar_SSB
hold on;
plot(ibar_SSB,max(dmrsEst),'kx','LineWidth',2,'MarkerSize',8);
legend(["SNRs" "\overline{i}_{SSB} = " + num2str(ibar_SSB)],'Interpreter','latex');

```



### Channel Estimation using PBCH DM-RS and SSS

The receiver estimates the channel for the entire SS/PBCH block using the SSS and PBCH DM-RS detected in previous steps. An estimate of the additive noise on the PBCH DM-RS / SSS is also performed.

```

refGrid = zeros([nrSSB*12 4]);
refGrid(dmrsIndices) = nrPBCHDMRS(ncellid,ibar_SSB);
refGrid(sssIndices) = nrSSS(ncellid);
[hest,nest,hestInfo] = nrChannelEstimate(rxGrid,refGrid,'AveragingWindow',[0 1]);

```

### PBCH Demodulation

The receiver uses the cell identity to determine and extract the resource elements associated with the PBCH from the received grid. In addition, the receiver uses the channel and noise estimates to perform MMSE equalization. The equalized PBCH symbols are then demodulated and descrambled to give bit estimates for the coded BCH block.

```

disp(' -- PBCH demodulation and BCH decoding -- ')

% Extract the received PBCH symbols from the SS/PBCH block
[pbchIndices,pbchIndicesInfo] = nrPBCHIndices(ncellid);
pbchRx = nrExtractResources(pbchIndices,rxGrid);

% Configure 'v' for PBCH scrambling according to TS 38.211 Section 7.3.3.1
% 'v' is also the 2 LSBs of the SS/PBCH block index for L_max=4, or the 3
% LSBs for L_max=8 or 64.
if refBurst.L_max == 4
    v = mod(ibar_SSB,4);
else
    v = ibar_SSB;
end
ssbIndex = v;

% PBCH equalization and CSI calculation
pbchHest = nrExtractResources(pbchIndices,hest);
[pbchEq,csi] = nrEqualizeMMSE(pbchRx,pbchHest,hest);
Qm = pbchIndicesInfo.G / pbchIndicesInfo.Gd;
csi = repmat(csi.',Qm,1);
csi = reshape(csi,[],1);

% Plot received PBCH constellation after equalization
figure;
plot(pbchEq,'o');
xlabel('In-Phase'); ylabel('Quadrature')
title('Equalized PBCH Constellation');
m = max(abs([real(pbchEq(:)); imag(pbchEq(:))])) * 1.1;
axis([-m m -m m]);

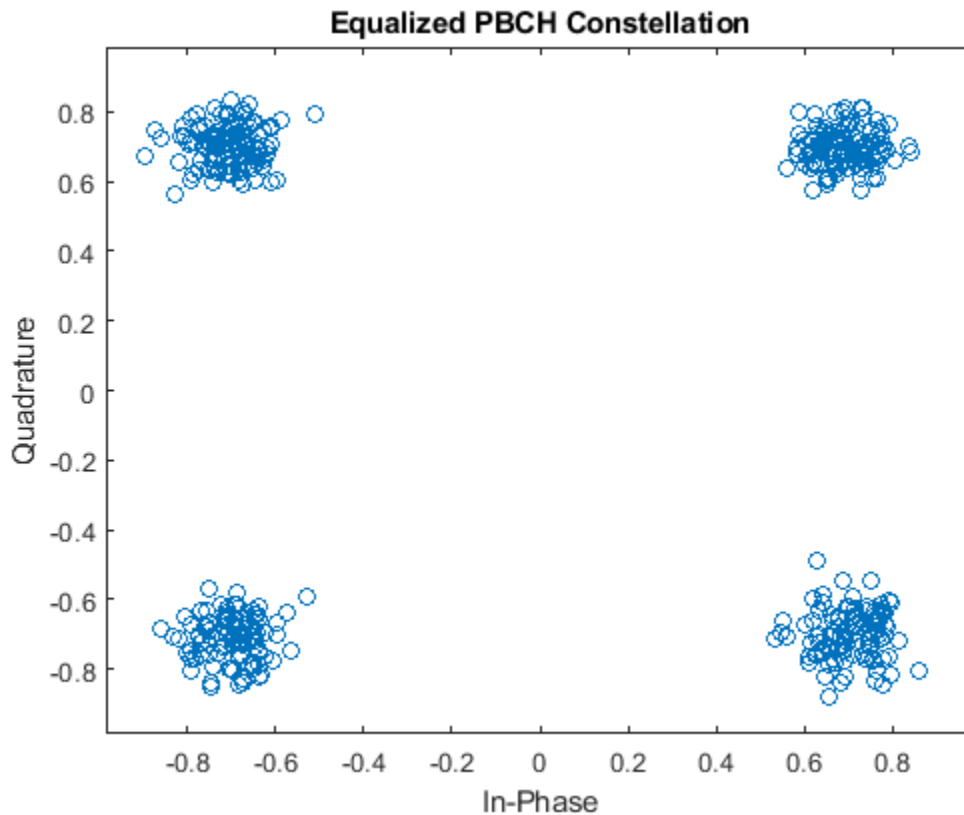
% PBCH demodulation
pbchBits = nrPBCHDecode(pbchEq,ncellid,v,hest);

% Calculate RMS PBCH EVM
pbchRef = nrPBCH(pbchBits<0,ncellid,v);
evm = comm.EVM;
pbchEVMrms = evm(pbchRef,pbchEq);

% Display calculated EVM
disp([' PBCH RMS EVM: ' num2str(pbchEVMrms,'%0.3f') '%']);

-- PBCH demodulation and BCH decoding --
PBCH RMS EVM: 8.740%

```



### BCH Decoding

The receiver weights BCH bit estimates with channel state information (CSI) from the MMSE equalizer and decodes the BCH. BCH decoding consists of rate recovery, polar decoding, CRC decoding, descrambling, and separating the 24 BCH transport block bits from the 8 additional timing-related payload bits.

```
% Apply CSI
pbchBits = pbchBits .* csi;

% Perform BCH decoding including rate recovery, polar decoding, and CRC
% decoding. PBCH descrambling and separation of the BCH transport block
% bits 'trblk' from 8 additional payload bits A...A+7 is also performed:
%   A ... A+3: 4 LSBs of System Frame Number
%       A+4: half frame number
%   A+5 ... A+7: for L_max=64, 3 MSBs of the SS/PBCH block index
%               for L_max=4 or 8, A+5 is the MSB of subcarrier offset k_SSB
polarListLength = 8;
[~,crcBCH,trblk,sfn4lsb,nHalfFrame,msbidxoffset] = ...
    nrBCHDecode(pbchBits,polarListLength,refBurst.L_max,ncellid);

% Display the BCH CRC
disp([' BCH CRC: ' num2str(crcBCH)]);

% Stop processing MIB and SIB1 if BCH was received with errors
if crcBCH
    disp(' BCH CRC is not zero.');
```

```

    return
end

% Use 'msbidxoffset' value to set bits of 'k_SSB' or 'ssbIndex', depending
% on the number of SS/PBCH blocks in the burst
if (refBurst.L_max==64)
    ssbIndex = ssbIndex + (bi2de(msbidxoffset.','left-msb') * 8);
    k_SSB = 0;
else
    k_SSB = msbidxoffset * 16;
end

% Displaying the SSB index
disp([' SSB index: ' num2str(ssbIndex)]);

    BCH CRC: 0
    SSB index: 0

```

### MIB Parsing

The example parses the 24 decoded BCH transport block bits into a structure which represents the MIB message fields. This process includes reconstituting the 10-bit system frame number (SFN) NFrame from the 6 MSBs in the MIB and the 4 LSBs in the PBCH payload bits. It also includes incorporating the MSB of the subcarrier offset  $k\_SSB$  from the PBCH payload bits in the case of  $L\_max=4$  or 8 SS/PBCH blocks per burst.

```

% Create set of subcarrier spacings signaled by the 7th bit of the decoded
% MIB, the set is different for FR1 (L_max=4 or 8) and FR2 (L_max=64)
if (refBurst.L_max==64)
    commonSCSs = [60 120];
else
    commonSCSs = [15 30];
end

% Create a structure of MIB fields from the decoded MIB bits. The BCH
% transport block 'trblk' is the RRC message BCCH-BCH-Message, consisting
% of a leading 0 bit then 23 bits corresponding to the MIB
mib.NFrame = bi2de([trblk(2:7); sfn4lsb] .','left-msb');
mib.SubcarrierSpacingCommon = commonSCSs(trblk(8) + 1);
mib.k_SSB = k_SSB + bi2de(trblk(9:12).','left-msb');
mib.DMRSTypeAPosition = 2 + trblk(13);
mib.PDCCHConfigSIB1 = bi2de(trblk(14:21).','left-msb');
mib.CellBarred = trblk(22);
mib.IntraFreqReselection = trblk(23);

% Display the MIB structure
disp(' BCH/MIB Content:')
disp(mib);

% Check if a CORESET for Type0-PDCCH common search space (CSS) is present,
% according to TS 38.213 Section 4.1
if ~isCORESET0Present(refBurst.BlockPattern,mib.k_SSB)
    fprintf('CORESET0 is not present (k_SSB > k_SSB_max).\n');
    return
end

    BCH/MIB Content:
        NFrame: 0

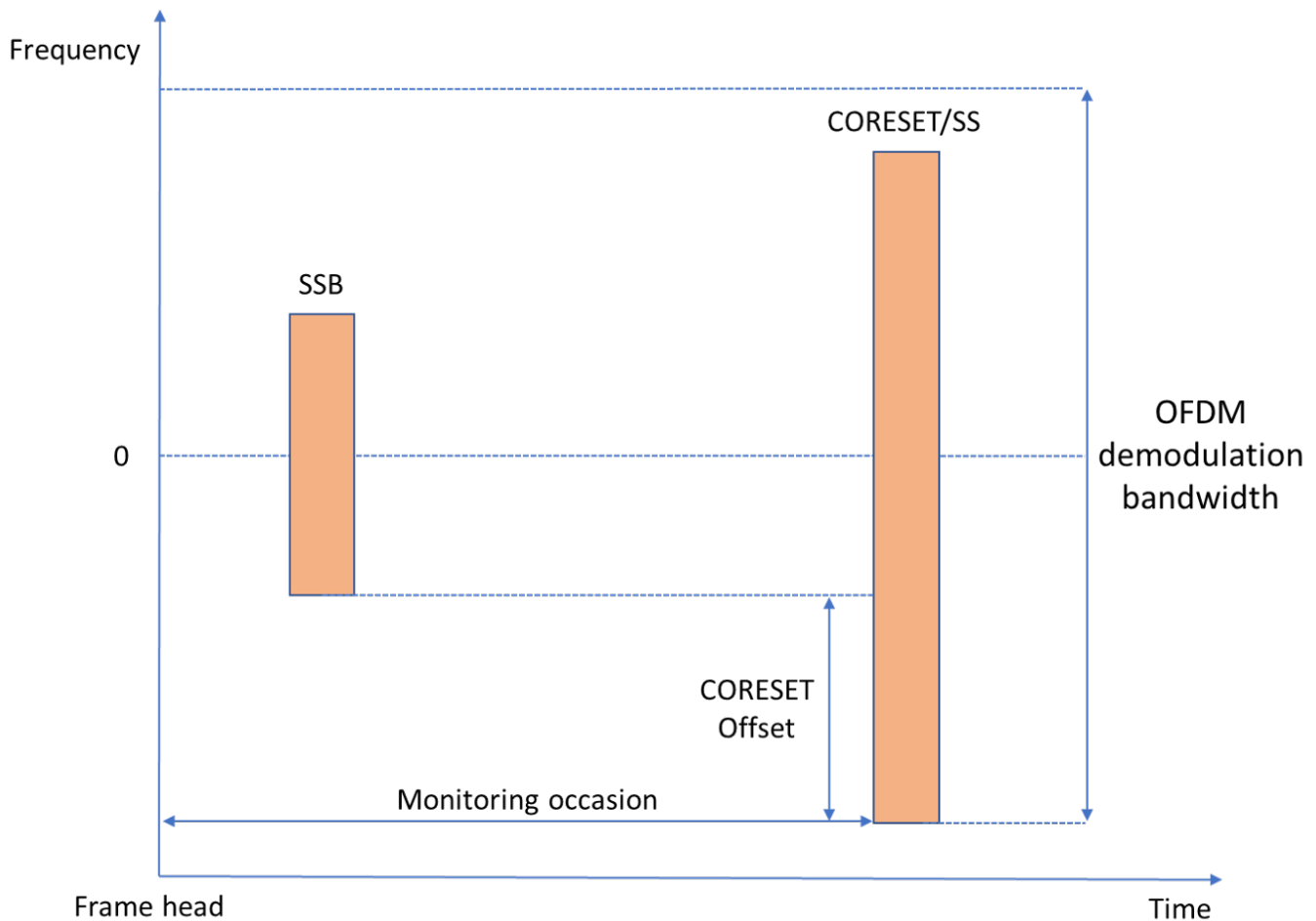
```

```

SubcarrierSpacingCommon: 15
    k_SSB: 0
  DMRSTypeAPosition: 3
    PDCCHConfigSIB1: 4
      CellBarred: 0
    IntraFreqReselection: 0
  
```

### OFDM Demodulation on Full Bandwidth

Once the MIB is recovered, the receiver uses common subcarrier spacing and a bandwidth supporting CORESET0 to OFDM demodulate the frame containing the detected SS block. The receiver determines the CORESET0 frequency resources in common numerology through an offset from the location of the SSB detected and a bandwidth specified in TS 38.213 Section 13 Tables 13-1 through 13-10 [ 5 ]. The frequency correction process aligned the center of the OFDM resource grid with the center frequency of the SS burst. However, these centers are not necessarily aligned with the center frequency of CORESET0. This figure shows the relationship between the SSB, CORESET0 frequency resources and associated PDCCH monitoring occasions.



Unlike the SS burst, control and data channels must be aligned in frequency with their common resource block (CRB) raster. The value of KSSB in the MIB signals the frequency offset of the SSB from that CRB raster. As the frequency correction process centered the SSB in frequency, apply a

```

frequency shift determined by k_SSB to align data and control channels with their CRB before OFDM
demodulation

if (refBurst.L_max==64)
    scsKSSB = mib.SubcarrierSpacingCommon;
else
    scsKSSB = 15;
end
k_SSB = mib.k_SSB;
fShift = k_SSB*scsKSSB*1e3;
rxWaveform = rxWaveform.*exp(1i*2*pi*fShift*(0:length(rxWaveform)-1)/rxSampleRate);

% Adjust timing offset to the frame origin
frameOffset = hTimingOffsetToFrame(refBurst,timingOffset,ssbIndex,rxSampleRate);
rxWaveform = rxWaveform(1+frameOffset:end,:);

% Determine the OFDM demodulation bandwidth using CORESET0 bandwidth
msbIdx = floor(mib.PDCCHConfigSIB1/16); % 4 MSB of PDCCHConfigSIB1 in MIB
scsCommon = mib.SubcarrierSpacingCommon;
scsPair = [scsSSB scsCommon];
[csetNRB,~,csetFreqOffset] = hCORESET0Resources(msbIdx,scsPair,minChannelBW,k_SSB);

% Minimum bandwidth in RB that includes CORESET0 in received waveform.
c0 = (csetFreqOffset+10*scsSSB/scsCommon); % CORESET frequency offset from carrier center
nrb = 2*max(c0,csetNRB-c0); % Minimum number of RB to cover CORESET0

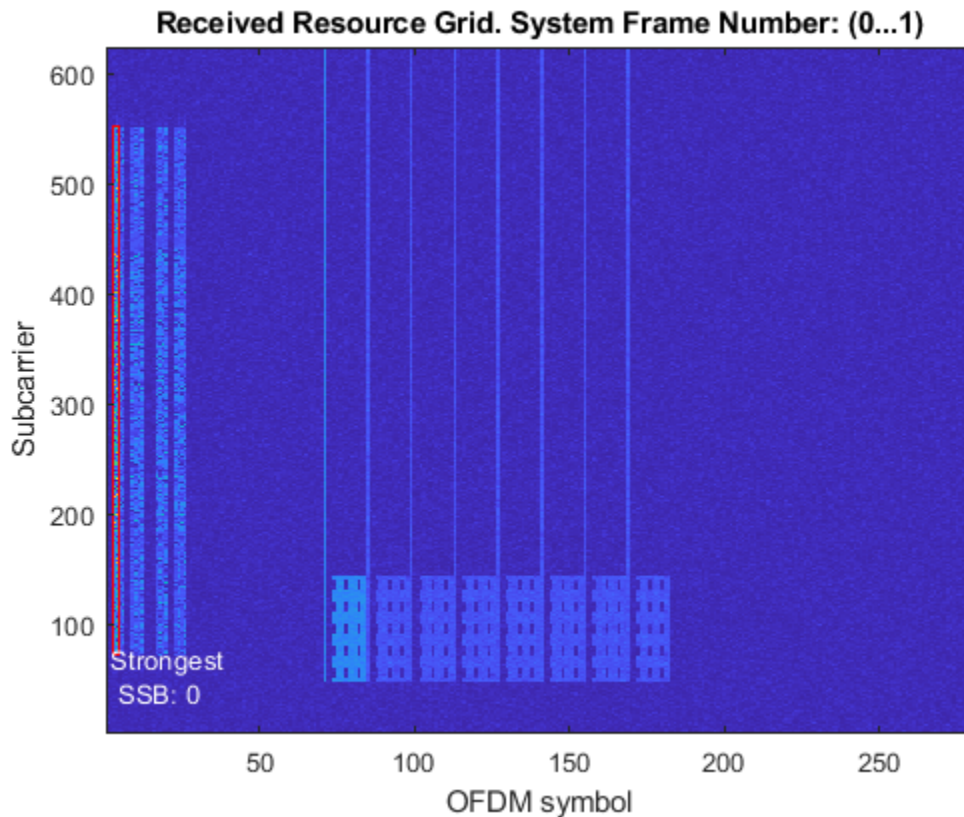
if rxSampleRate < nrb*12*scsCommon*1e3
    disp(['SIB1 recovery cannot continue. CORESET0 resources are beyond '...
        'the frequency limits of the received waveform for the sampling rate configured.']);
    return;
end

% OFDM demodulate received waveform with common subcarrier spacing
nSlot = 0;
rxGrid = nrOFDMDemodulate(rxWaveform, nrb, scsCommon, nSlot,...
    'SampleRate',rxSampleRate,'CarrierFrequency',fPhaseComp);

% Display OFDM resource grid and highlight strongest SS block
figure;
imagesc(abs(rxGrid(:,:,1))); axis xy
xlabel('OFDM symbol'); ylabel('Subcarrier');
numFrames = floor(length(rxWaveform)/rxSampleRate/10e-3);
sfns = sprintf('%d...%d',mib.NFrame, mib.NFrame+numFrames-1);
title(['Received Resource Grid. System Frame Number: ' sfns]);

ssbOffset = timingOffset - frameOffset; % Timing offset to SSB measured from frame head
highlightSSBlock(ssbOffset,ssbIndex,nrb,scsPair,rxSampleRate)

```



### Demodulation of PDCCH and Downlink Control Information Decoding

To blindly search for system information DCI messages in CORESET/SS, the receiver performs these steps:

- Determination of PDCCH monitoring occasions and extraction of the OFDM resource grid containing control information.
- Configuration of CORESET0, Search spaces and PDCCH.
- Blind search for Format 1\_0 DCI messages.

The receiver determines the PDCCH monitoring occasions through a slot and OFDM symbol offset from the location of the SS block detected, as described in TS 38.213 Tables 13-11 and 13-12 [ 5 ].

```

msbIdx = floor(mib.PDCCHConfigSIB1/16); % 4 MSB of PDCCHConfigSIB1 in MIB index Tables 13-1 to 13-11
[csetNRB,csetDuration,csetOffset,csetPattern] = hCORESET0Resources(msbIdx,scsPair,minChannelBW,k_scs);
lsbIdx = mod(mib.PDCCHConfigSIB1,16);
[ssSlot,ssFirstSym,isOccasion] = hPDCCH0MonitoringOccasions(lsbIdx,ssbIndex,scsPair,csetPattern,ssbIndex);

% PDCCH monitoring occasions associated to different SS blocks can be in
% different frames. If there are no monitoring occasions in this frame,
% there must be one in the next one.
slotsPerFrame = 10*scsCommon/15;
if ~isOccasion
    [ssSlot,ssFirstSym,isOccasion] = hPDCCH0MonitoringOccasions(lsbIdx,ssbIndex,scsPair,csetPattern,ssbIndex);
    ssSlot = ssSlot+slotsPerFrame;
end

```



```

% For FR1, UE monitors PDCCH in the Type0-PDCCH CSS over two consecutive
% slots for CORESET pattern 1
if csetPattern == 1
    monSlotsPerPeriod = 2;
else
    monSlotsPerPeriod = 1;
end

% Calculate 1-based subscripts of the subcarriers and OFDM symbols for the
% slots containing the PDCCH0 associated to the detected SS block in this
% and subsequent 2-frame blocks
csetSubcarriers = 12*(nrb-20*scsSSB/scsCommon)/2 - csetOffset*12 + (1:csetNRB*12);
numRxSym = size(rxGrid,2);
symbolsPerSlot = 14;
numRxSlots = ceil(numRxSym/symbolsPerSlot);
monSlots = ssSlot + (0:monSlotsPerPeriod-1)' + (0:2*slotsPerFrame:(numRxSlots-ssSlot-1));
monSlots = monSlots(:)';
monSymbols = monSlots*symbolsPerSlot + (1:symbolsPerSlot)';
monSymbols = monSymbols(:)';
% Remove monitoring symbols exceeding waveform limits
monSymbols(monSymbols > numRxSym) = [];

% Check if search space is beyond end of waveform
if isempty(monSymbols)
    disp('Search space slot is beyond end of waveform. ');
    return;
end

% Extract slots containing strongest PDCCH from the received grid
rxMonSlotGrid = rxGrid(csetSubcarriers,monSymbols,:);

Configure CORESET, search space, and other PDCCH parameters. CORESET resources and search
spaces are configured according to TS 38.213 Section 13 Tables 13-1 through 13-15 [ 5 ]. CCE-to-REG
interleaved mapping parameters (REGBundleSize = 6, InterleaverSize = 2, and ShiftIndex = NCellID)
are described in TS 38.211 Section 7.3.2.2 [ 3 ]. For CORESET 0, the BWP is the size of the CORESET
as described in TS 38.212 Section 7.3.1.0 [ 4 ]. The PDCCH scrambling parameters are nRNTI = 0
and nID = NCellID as described in TS 38.211 Section 7.3.2.3 [ 3 ].

pdccch = hPDCCH0Configuration(ssbIndex,mib,scsPair,ncellid,minChannelBW);

% Configure the carrier to span the BWP (CORESET0)
c0Carrier = nrCarrierConfig;
c0Carrier.SubcarrierSpacing = mib.SubcarrierSpacingCommon;
c0Carrier.NStartGrid = pdccch.NStartBWP;
c0Carrier.NSizeGrid = pdccch.NSizeBWP;
c0Carrier.NSlot = pdccch.SearchSpace.SlotPeriodAndOffset(2);
c0Carrier.NFrame = mib.NFrame;
c0Carrier.NCellID = ncellid;

Search for DCI messages. UE decodes the received PDCCH symbols blindly by monitoring all PDCCH
candidates for every aggregation level using the SI-RNTI to identify the right candidate (or instance).

% Specify DCI message with Format 1_0 scrambled with SI-RNTI (TS 38.212
% Section 7.3.1.2.1)
dcispec1_0 = hSystemInformationDCIFieldsSize(pdccch.NSizeBWP);
numDCIBits = sum(structfun(@(x)x,dcispec1_0));

```

```

disp(' -- Downlink control information message search in PDCCH -- ');

siRNTI = 65535; % TS 38.321 Table 7.1-1
dciCRC = true;
mSlot = 0;
% Loop over all monitoring slots
while (mSlot < length(monSlots)) && dciCRC ~= 0
    c0Carrier.NSlot = monSlots(mSlot+1);

    if monSlotsPerPeriod==2
        if mod(mSlot,2)
            pdcch.SearchSpace.SlotPeriodAndOffset(2) = monSlots(2);
        else
            pdcch.SearchSpace.SlotPeriodAndOffset(2) = monSlots(1);
        end
    end

    % Get PDCCH candidates according to TS 38.213 Section 10.1
    [pdcchInd,pdcchDmrsSym,pdcchDmrsInd] = nrPDCCHSpace(c0Carrier,pdcch);
    rxSlotGrid = rxMonSlotGrid(:,(1:symbolsPerSlot) + symbolsPerSlot*mSlot,:);
    rxSlotGrid = rxSlotGrid/max(abs(rxSlotGrid(:))); % Normalization of received RE magnitude

    % Loop over all supported aggregation levels
    aLev = 1;
    while (aLev <= 5) && dciCRC ~= 0
        % Loop over all candidates at each aggregation level in SS
        cIdx = 1;
        numCandidatesAL = pdcch.SearchSpace.NumCandidates(aLev);
        while (cIdx <= numCandidatesAL) && dciCRC ~= 0
            % Channel estimation using PDCCH DM-RS
            [hest,nVar,pdcchHestInfo] = nrChannelEstimate(rxSlotGrid,pdcchDmrsInd{aLev}(:,cIdx),nVar);

            % Equalization and demodulation of PDCCH symbols
            [pdcchRxSym,pdcchHest] = nrExtractResources(pdcchInd{aLev}(:,cIdx),rxSlotGrid,hest);
            pdcchEqSym = nrEqualizeMMSE(pdcchRxSym,pdcchHest,nVar);
            dcicw = nrPDCCHDecode(pdcchEqSym,pdcch.DMRSScramblingID,pdcch.RNTI,nVar);

            % DCI message decoding
            polarListLength = 8;
            [dcibits,dciCRC] = nrDCIDecode(dciw,numDCIBits,polarListLength,siRNTI);

            if dciCRC == 0
                disp([' Decoded PDCCH candidate #' num2str(cIdx) ' at aggregation level ' num2str(aLev)]);
            end
            cIdx = cIdx + 1;
        end
        aLev = aLev+1;
    end
    mSlot = mSlot+1;
end
cIdx = cIdx-1;
aLev = aLev-1;
mSlot = mSlot-1;
monSymbols = monSymbols(mSlot*symbolsPerSlot + (1:symbolsPerSlot));

% Calculate RMS PDCCH EVM
pdcchRef = nrPDCCH(double(dciw<0),pdcch.DMRSScramblingID,pdcch.RNTI);
evm = comm.EVM;

```

```

pdccchEVMrms = evm(pdccchRef,pdccchEqSym);

% Display calculated EVM
disp([' PDCCH RMS EVM: ' num2str(pdccchEVMrms,'%0.3f') '%']);
disp([' PDCCH CRC: ' num2str(dciCRC)]);

% Highlight CORESET0/SS corresponding to strongest SSB
bounding_box = @(y,x,h,w)rectangle('Position',[x+0.5 y-0.5 w h],'EdgeColor','r');
bounding_box(csetSubcarriers(1),monSymbols(1)+ssFirstSym-1,csetNRB*12,csetDuration);
str = sprintf('CORESET0/SS');
text(monSymbols(1)+ssFirstSym-7,csetSubcarriers(1)-20,0,str,'FontSize',10,'Color','w')

if dciCRC
    disp(' DCI decoding failed. ');
    return
end

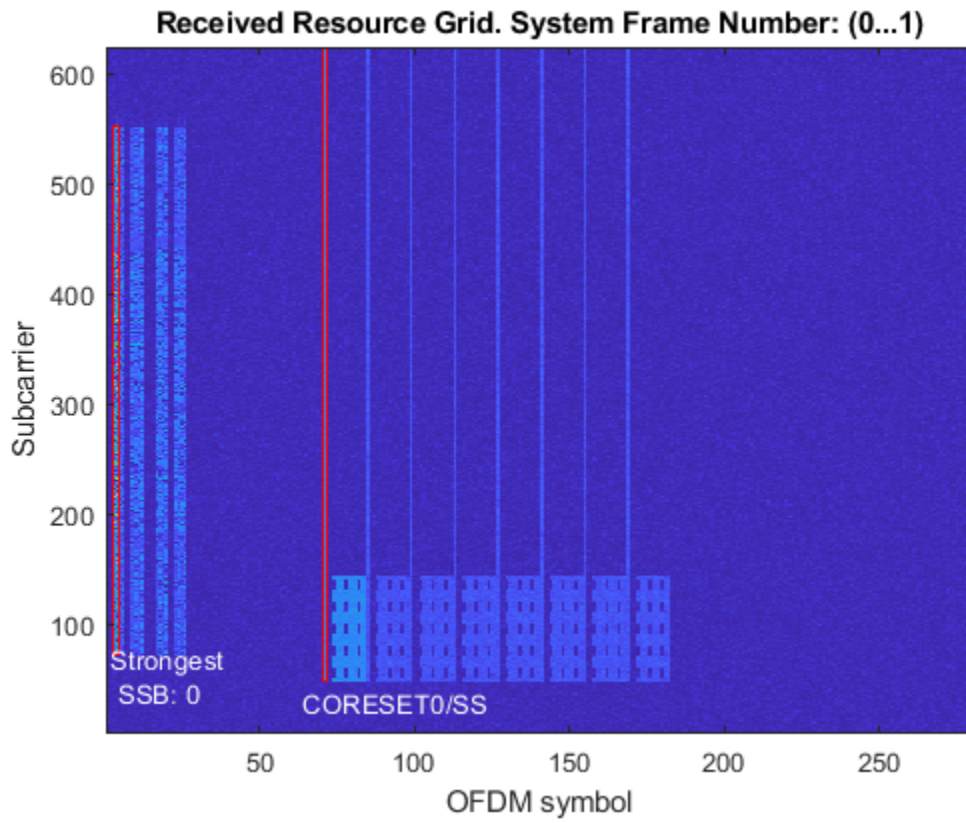
% Plot received PDCCH constellation after equalization
figure;
plot(pdccchEqSym,'o');
xlabel('In-Phase'); ylabel('Quadrature');
title('Equalized PDCCH Constellation');
m = max(abs([real(pdccchEqSym(:)); imag(pdccchEqSym(:))])) * 1.1;
axis([-m m -m m]);

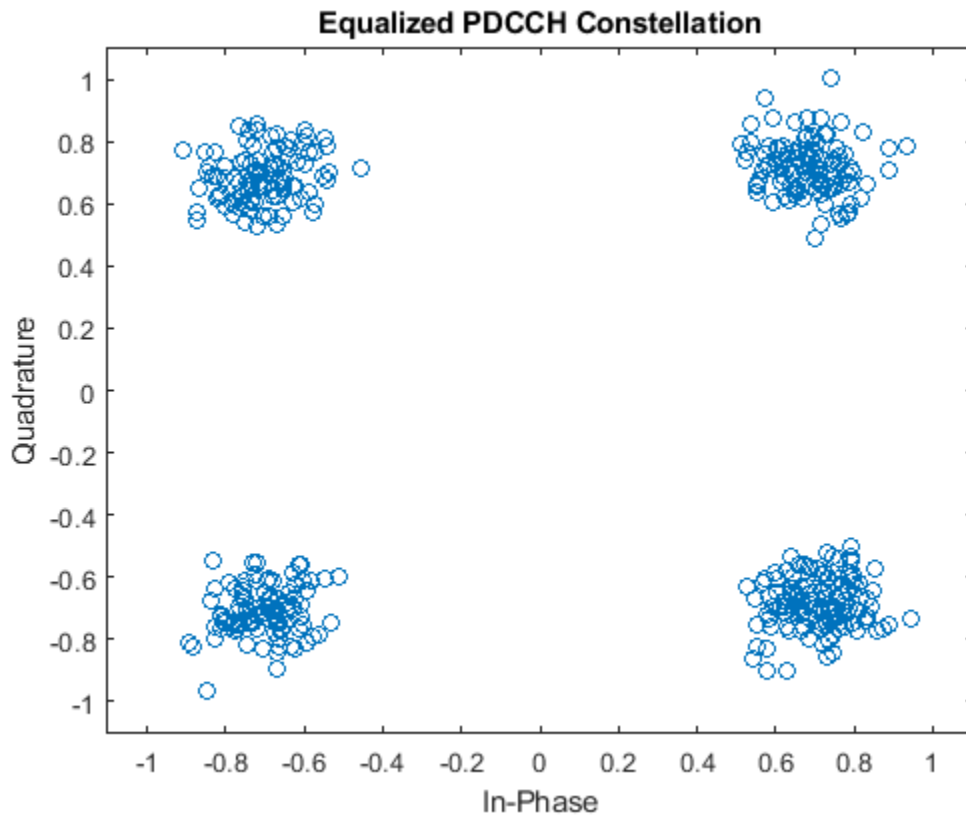
% Display the OFDM grid of the slot containing strongest PDCCH
figure;
imagesc(abs(rxSlotGrid(:,:,1))); axis xy
xlabel('OFDM symbol');
ylabel('subcarrier');
title('Slot Containing Strongest PDCCH');

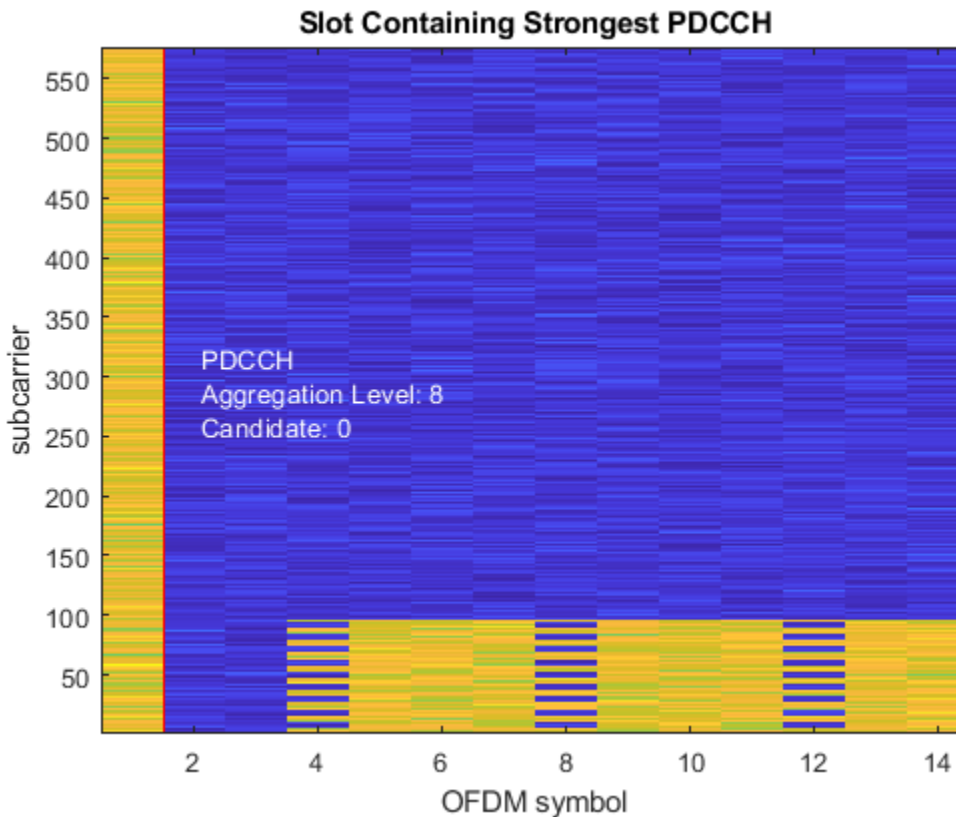
% Highlight PDCCH in resource grid
subsPdcch = nrPDCCHSpace(c0Carrier,pdccch,'IndexStyle','Subs');
subsPdcch = double(subsPdcch{aLev}(:,:,cIdx));
x = min(subsPdcch(:,2))-1; X = max(subsPdcch(:,2))-x;
y = min(subsPdcch(:,1)); Y = max(subsPdcch(:,1))-y;
bounding_box(y,x,Y,X);
str = sprintf(' PDCCH \n Aggregation Level: %d\n Candidate: %d',2^(aLev-1),cIdx-1);
text(x+X+1,y+Y/2,0,str,'FontSize',10,'Color','w')

-- Downlink control information message search in PDCCH --
Decoded PDCCH candidate #1 at aggregation level 8
PDCCH RMS EVM: 11.870%
PDCCH CRC: 0

```







### Demodulation of PDSCH, Decoding of DL-SCH and SIB1 Extraction

To recover the first system information block, the receiver performs these steps:

- Determination of PDSCH configuration using cell ID, MIB, and DCI
- Channel estimation, equalization and demodulation of PDSCH symbols
- Decoding of DL-SCH and SIB1 extraction

```
disp(' -- PDSCH demodulation and DL-SCH decoding -- ')
```

```
% Build DCI message structure
dci = hDCI(dcispec1_0,dcibits);
```

```
% Get PDSCH configuration from cell ID, MIB, and DCI
[psch,K_0] = hSIB1PDSCHConfiguration(dci,pcch.NSizeBWP,mib.DMRSTypeAPosition,csetPattern);
```

```
% For CORESET pattern 2, the gNodeB can allocate PDSCH in the next slot,
% which is indicated by the slot offset K_0 signaled by DCI. For more
% information, see TS 38.214 Table 5.1.2.1.1-4.
```

```
c0Carrier.NSlot = c0Carrier.NSlot+K_0;
symbolOffset = symbolsPerSlot*(mSlot+K_0);
monSymbols = monSymbols+symbolOffset;
rxSlotGrid = rxGrid(csetSubcarriers,monSymbols,:);
rxSlotGrid = rxSlotGrid/max(abs(rxSlotGrid(:))); % Normalization of received RE magnitude
if K_0 > 0
    % Display the OFDM grid of the slot containing associated PDSCH
    figure;
```

```

    imagesc(abs(rxSlotGrid(:,:,1))); axis xy
    xlabel('OFDM symbol');
    ylabel('subcarrier');
    title('Slot Containing PDSCH (Slot Offset K_0 = 1)');
end

```

```

% PDSCH channel estimation and equalization using PDSCH DM-RS
pdschDmrsIndices = nrPDSCHDMRSIndices(c0Carrier,pdsch);
pdschDmrsSymbols = nrPDSCHDMRS(c0Carrier,pdsch);

```

```
-- PDSCH demodulation and DL-SCH decoding --
```

To compensate for the negative effects of a carrier frequency mismatch in symbol phase compensation and channel estimation, the receiver OFDM demodulates the waveform with a set of carrier frequencies over a search bandwidth around `fPhaseComp`. The search finishes when DL-SCH decoding succeeds or the last frequency has been reached. The minimum search bandwidths that produce equal symbol phase compensation are 1920, 3840, 7680, and 15360 kHz for common subcarrier spacings 15, 30, 60, and 120 kHz, respectively. Increase the search bandwidth up to these values when SIB1 decoding fails and the equalized PDSCH symbols result in a heavily distorted and rotated constellation.

```

mu = log2(scsCommon/15);
bw = 2^mu*100; % Search bandwidth (kHz)
freqStep = 2^mu; % Frequency step (kHz)
freqSearch = -bw/2:freqStep:bw/2-freqStep;
[~,fSearchIdx] = sort(abs(freqSearch)); % Sort frequencies from center
freqSearch = freqSearch(fSearchIdx);

for fpc = fPhaseComp + 1e3*freqSearch

    % OFDM demodulate received waveform
    nSlot = 0;
    rxGrid = nrOFDMDemodulate(rxWaveform, nrb, scsCommon, nSlot,...
        'SampleRate',rxSampleRate,'CarrierFrequency',fpc);

    % Extract monitoring slot from the received grid
    rxSlotGrid = rxGrid(csetSubcarriers,monSymbols,:);
    rxSlotGrid = rxSlotGrid/max(abs(rxSlotGrid(:))); % Normalization of received RE magnitude

    % Channel estimation and equalization of PDSCH symbols
    [hest,nVar,pdschHestInfo] = nrChannelEstimate(rxSlotGrid,pdschDmrsIndices,pdschDmrsSymbols);
    [pdschIndices,pdschIndicesInfo] = nrPDSCHIndices(c0Carrier,pdsch);
    [pdschRxSym,pdschHest] = nrExtractResources(pdschIndices,rxSlotGrid,hest);
    pdschEqSym = nrEqualizeMMSE(pdschRxSym,pdschHest,nVar);

    % PDSCH demodulation
    cw = nrPDSCHDecode(c0Carrier,pdsch,pdschEqSym,nVar);

    % Initialize DL-SCH decoder
    decodedLSCH = nrDLSCHDecoder;

    % Target code rate and transport block size
    Xoh_PDSCH = 0; % TS 38.214 Section 5.1.3.2
    tcr = hMCS(dci.ModCoding);
    NREPerPRB = pdschIndicesInfo.NREPerPRB;
    tbsLength = nrTBS(pdsch.Modulation,pdsch.NumLayers,length(pdsch.PRBSets),NREPerPRB,tcr,Xoh_PD
    decodedLSCH.TransportBlockLength = tbsLength;
    decodedLSCH.TargetCodeRate = tcr;

```

```
% Decode DL-SCH
[sib1bits,sib1CRC] = decodeDLSCH(cw,pdsch.Modulation,pdsch.NumLayers,dci.RV);

if sib1CRC == 0
    break;
end

end

% Highlight PDSCH in resource grid
subsPdsch = double(nrPDSCHIndices(c0Carrier,pdsch,'IndexStyle','subscript'));
x = min(subsPdsch(:,2))-1; X = max(subsPdsch(:,2))-x;
y = min(subsPdsch(:,1)); Y = max(subsPdsch(:,1))-y;
bounding_box(y,x,Y,X);
str = sprintf('PDSCH (SIB1) \n Modulation: %s\n Code rate: %.2f',pdsch.Modulation,tcR);
text(x+4,y+Y+60,0, str,'FontSize',10,'Color','w')

% Plot received PDSCH constellation after equalization
figure;
plot(pdschEqSym,'o');
xlabel('In-Phase'); ylabel('Quadrature')
title('Equalized PDSCH Constellation');
m = max(abs([real(pdschEqSym(:)); imag(pdschEqSym(:))])) * 1.1;
axis([-m m -m m]);

% Calculate RMS PDSCH EVM, including normalization of PDSCH symbols for any
% offset between DM-RS and PDSCH power
pdschRef = nrPDSCH(c0Carrier,pdsch,double(cw{1}<0));
evm = comm.EVM;
pdschEVMrms = evm(pdschRef,pdschEqSym/sqrt(var(pdschEqSym)));

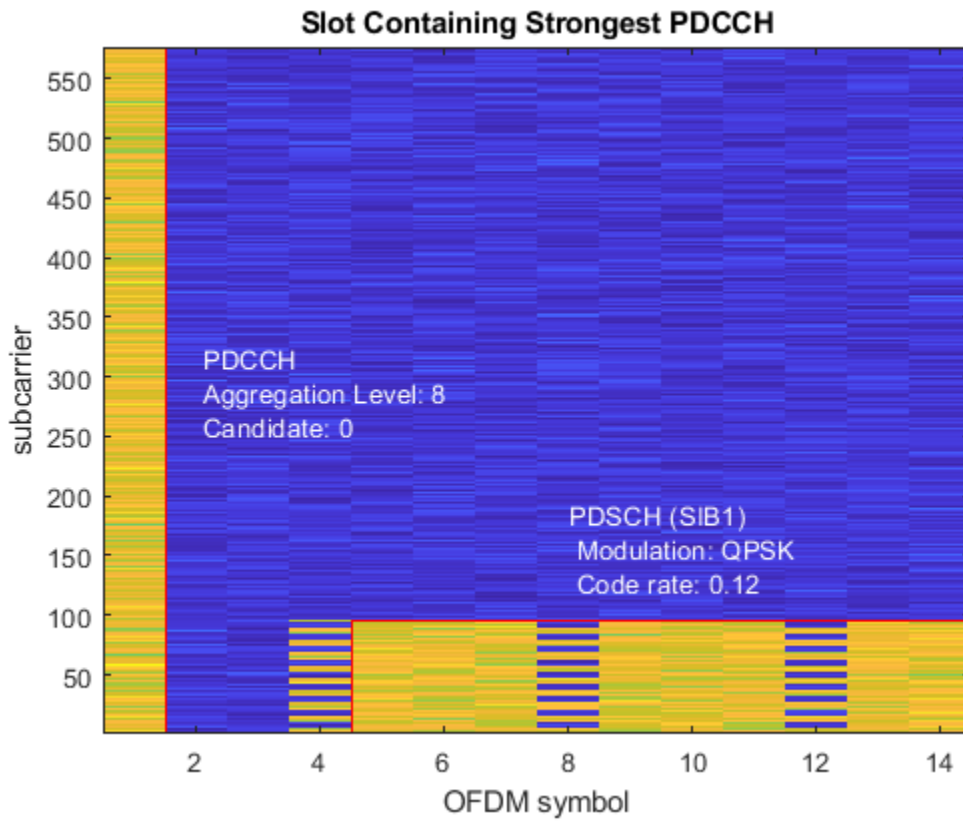
% Display PDSCH EVM and DL-SCH CRC
disp([' PDSCH RMS EVM: ' num2str(pdschEVMrms,'%0.3f') '%']);
disp([' PDSCH CRC: ' num2str(sib1CRC)]);

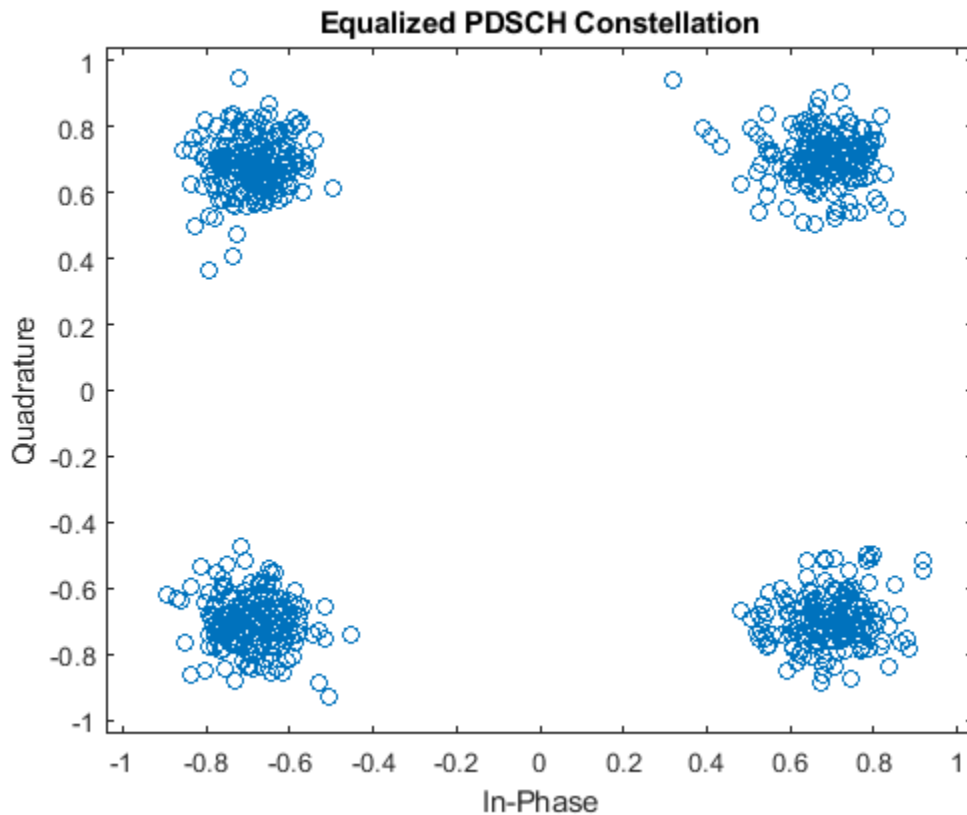
if sib1CRC == 0
    disp(' SIB1 decoding succeeded. ');
else
    disp(' SIB1 decoding failed. ');
end

end

PDSCH RMS EVM: 11.154%
PDSCH CRC: 0
SIB1 decoding succeeded.
```







## Appendix

This example uses these helper functions:

- hCORESET0Resources.m
- hMCS.m
- hPDCCH0Configuration.m
- hPDCCH0MonitoringOccasions.m
- hSIB1PDSCHConfiguration.m
- hPDSCHTimeAllocationTables.m
- hSIB1WaveformConfiguration.m
- hSIB1Boost.m
- hSSBurstFrequencyCorrect.m
- hSSBurstStartSymbols.m
- hSSBurstSubcarrierSpacing.m
- hSystemInformationDCIFieldsSize.m

## 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 38.104. "NR; Base Station (BS) radio transmission and reception." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network.*
- 3 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network.*
- 4 3GPP TS 38.212. "NR; Multiplexing and channel coding." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network.*
- 5 3GPP TS 38.213. "NR; Physical layer procedures for control." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network.*
- 6 3GPP TS 38.214. "NR; Physical layer procedures for data." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network.*
- 7 3GPP TS 38.321. "NR; Medium Access Control (MAC) protocol specification." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network.*

### Local functions

```
function present = isCORESET0Present(ssbBlockPattern,kSSB)
```

```
    switch ssbBlockPattern
        case {'Case A','Case B','Case C'} % FR1
            kssb_max = 23;
        case {'Case D','Case E'} % FR2
            kssb_max = 11;
    end
    if (kSSB <= kssb_max)
        present = true;
    else
        present = false;
    end
end
```

```
end
```

```
function dci = hDCI(dcispec,dcibits)
```

```
    % Parse DCI message into a structure of DCI message fields
    fieldsizes = structfun(@(x)x,dcispec);
    fieldbits2dec = @(x,y)bin2dec(char(x(y(1):y(2)) + '0'));
    fieldbitranges = [[0; cumsum(fieldsizes(1:end-1))]+1 cumsum(fieldsizes)];
    fieldbitranges = num2cell(fieldbitranges,2);
    values = cellfun(@(x)fieldbits2dec(dcibits.',x),fieldbitranges,'UniformOutput',false);
    dci = cell2struct(values,fieldnames(dcispec));
```

```
end
```

```
function timingOffset = hTimingOffsetToFrame(burst,offset,ssbIdx,rxSampleRate)
```

```
    % As the symbol lengths are measured in FFT samples, scale the symbol
    % lengths to account for the receiver sample rate. Non-integer delays
    % are approximated at the end of the process.
    scs = hSSBurstSubcarrierSpacing(burst.BlockPattern);
    ofdmInfo = nrOFDMInfo(1,scs,'SampleRate',rxSampleRate); % smallest FFT size for SCS-SR
    srRatio = rxSampleRate/(scs*1e3*ofdmInfo.Nfft);
    symbolLengths = ofdmInfo.SymbolLengths*srRatio;

    % Adjust timing offset to the start of the SS block. This step removes
    % the extra offset introduced in the reference grid during PSS search,
```

```

% which contained the PSS in the second OFDM symbol.
offset = offset + symbolLengths(1);

% Timing offset is adjusted so that the received grid starts at the
% frame head i.e. adjust the timing offset for the difference between
% the first symbol of the strongest SSB, and the start of the frame
burstStartSymbols = hSSBurstStartSymbols(burst.BlockPattern,burst.L_max); % Start symbols in
ssbFirstSym = burstStartSymbols(ssbIdx+1); % 0-based

% Adjust for whole subframes
symbolsPerSubframe = length(symbolLengths);
subframeOffset = floor(ssbFirstSym/symbolsPerSubframe);
samplesPerSubframe = sum(symbolLengths);
timingOffset = offset - (subframeOffset*samplesPerSubframe);

% Adjust for remaining OFDM symbols and round offset if not integer
symbolOffset = mod(ssbFirstSym,symbolsPerSubframe);
timingOffset = round(timingOffset - sum(symbolLengths(1:symbolOffset)));

% Apply modulo 20 ms periodicity in case offset is negative
if timingOffset < 0
    timingOffset = mod(timingOffset,samplesPerSubframe*20);
end

end

function highlightSSBlock(ssbOffset,ssbIndex,nrb,scs,rxSampleRate)

scsSSB = scs(1);
scsCommon = scs(2);

% As the symbol lengths are measured in FFT samples, scale the symbol
% lengths to account for the receiver sample rate.
ofdmInfo = nrOFDMInfo(20,scsSSB,'SampleRate',rxSampleRate);
srRatio = rxSampleRate/(scsSSB*1e3*ofdmInfo.Nfft);
symbolLengths = ofdmInfo.SymbolLengths*srRatio;

ssbOffset = ssbOffset + round(symbolLengths(1)) ;

% Determine frequency origin of the SSB in common numerology
bounding_box = @(y,x,h,w)rectangle('Position',[x+0.5 y-0.5 w h],'EdgeColor','r');
scsRatio = scsSSB/scsCommon;
ssbFreqOrig = 12*(nrb-20*scsRatio)/2+1;

% Determine time origin of the SSB in common numerology
rxCommonOfdmInfo = nrOFDMInfo(nrb,scsCommon,'SampleRate',rxSampleRate);
symLen = cumsum(rxCommonOfdmInfo.SymbolLengths*srRatio);
symbolsPerSubframe = length(symLen);
ssbSubframeOffset = floor(ssbOffset/symLen(end))*symbolsPerSubframe;
ssbTimeOrig = ssbSubframeOffset + find(symLen - mod(ssbOffset,symLen(end))>=0,1,'first');
ssbTimeOrig = round(ssbTimeOrig);

bounding_box(ssbFreqOrig,ssbTimeOrig,240*scsRatio,4*scsCommon/scsSSB);

str = sprintf('Strongest \n SSB: %d',ssbIndex);
text(ssbTimeOrig,ssbFreqOrig-20,0, str,'FontSize',10,'Color','w')

```

end

## **See Also**

### **More About**

- “Synchronization Signal Blocks and Bursts”
- “NR SSB Beam Sweeping” on page 1-95
- “NR Downlink Transmit-End Beam Refinement Using CSI-RS” on page 1-110

## NR PDSCH Throughput

This example shows how to measure the physical downlink shared channel (PDSCH) throughput of a 5G New Radio (NR) link, as defined by the 3GPP NR standard. The example implements the PDSCH and downlink shared channel (DL-SCH). The transmitter model includes PDSCH demodulation reference signals (DM-RS), PDSCH phase tracking reference signals (PT-RS), and synchronization signal (SS) bursts. The example supports both clustered delay line (CDL) and tapped delay line (TDL) propagation channels. You can perform perfect or practical synchronization and channel estimation. To reduce the total simulation time, you can execute the SNR points in the SNR loop in parallel by using the Parallel Computing Toolbox™.

### Introduction

This example measures the PDSCH throughput of a 5G link, as defined by the 3GPP NR standard [ 1 ], [ 2 ], [ 3 ], [ 4 ].

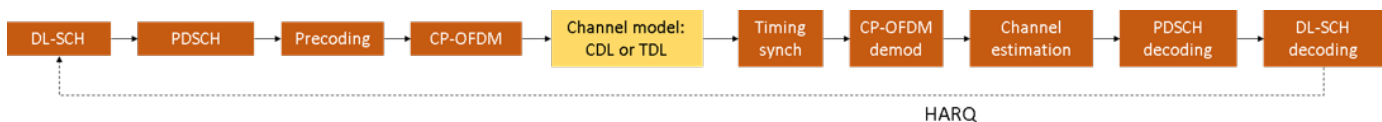
The example models these 5G NR features:

- DL-SCH transport channel coding
- PDSCH, PDSCH DM-RS, and PDSCH PT-RS generation
- SS burst generation (PSS/SSS/PBCH/PBCH DM-RS)
- Variable subcarrier spacing and frame numerologies ( $2^n * 15\text{kHz}$ ) for normal and extended cyclic prefix
- TDL and CDL propagation channel models

Other features of the simulation are:

- PDSCH precoding using SVD
- CP-OFDM modulation
- Slot wise and non slot wise PDSCH and DM-RS mapping
- SS burst generation (cases A-E, SS/PBCH block bitmap control)
- Perfect or practical synchronization and channel estimation
- HARQ operation with 16 processes
- The example uses a single bandwidth part across the whole carrier

The figure shows the implemented processing chain. For clarity, the DM-RS, PT-RS, and SS burst generation are omitted.



This example supports operation with one or two codewords, dependent on the number of layers selected for the transmission. A single precoding matrix for the whole PDSCH allocation is determined using SVD by averaging the channel estimate across all allocated PDSCH PRBs. Therefore, for large PDSCH allocations, i.e. occupying a wide bandwidth, the single precoding matrix may not be well matched to the channel across all frequencies, resulting in performance degradation. There is no beamforming on the SS/PBCH blocks in the SS burst.

To reduce the total simulation time, you can use the Parallel Computing Toolbox to execute the SNR points of the SNR loop in parallel.

### Simulation Length and SNR Points

Set the length of the simulation in terms of the number of 10ms frames. A large number of NFrames should be used to produce meaningful throughput results. Set the SNR points to simulate. The SNR for each layer is defined per RE, and it includes the effect of signal and noise across all antennas.

```
simParameters = []; % Clear simParameters variable
simParameters.NFrames = 2; % Number of 10ms frames
simParameters.SNRIn = [-5 0 5]; % SNR range (dB)
```

### Channel Estimator Configuration

The logical variable `perfectChannelEstimator` controls channel estimation and synchronization behavior. When set to `true`, perfect channel estimation and synchronization is used. Otherwise, practical channel estimation and synchronization is used, based on the values of the received PDSCH DM-RS.

```
perfectChannelEstimator = true;
```

### Carrier and PDSCH Configuration

Set the key parameters of the simulation. These include:

- The bandwidth in resource blocks (12 subcarriers per resource block).
- Subcarrier spacing: 15, 30, 60, 120, 240 (kHz)
- Cyclic prefix length: normal or extended
- Cell ID
- Number of transmit and receive antennas

A substructure containing the DL-SCH and PDSCH parameters is also specified. This includes:

- Target code rate
- Allocated resource blocks (PRBSet)
- Modulation scheme: 'QPSK', '16QAM', '64QAM', '256QAM'
- Number of layers
- PDSCH mapping type
- DM-RS configuration parameters
- PT-RS configuration parameters

Other simulation wide parameters are:

- Propagation channel model: 'TDL' or 'CDL'
- SS burst configuration parameters. Note that the SS burst generation can be disabled by setting the `SSBTransmitted` field to [0 0 0 0].

```
% Set waveform type and PDSCH numerology (SCS and CP type)
simParameters.NRB = 51; % Bandwidth in number of resource blocks (51RBs at 30kHz)
simParameters.SubcarrierSpacing = 30; % 15, 30, 60, 120, 240 (kHz)
simParameters.CyclicPrefix = 'Normal'; % 'Normal' or 'Extended'
```

```

simParameters.NCellID = 1; % Cell identity

% DL-SCH/PDSCH parameters
simParameters.PDSCH.PRBSet = 0:simParameters.NRB-1; % PDSCH PRB allocation
simParameters.PDSCH.SymbolSet = 0:13; % PDSCH symbol allocation in each slot
simParameters.PDSCH.EnableHARQ = true; % Enable/disable HARQ, if disabled, single trans

simParameters.PDSCH.NLayers = 2; % Number of PDSCH layers
simParameters.NTxAnts = 8; % Number of PDSCH transmission antennas (1,2,4,8)
if simParameters.PDSCH.NLayers > 4 % Multicodeword transmission
    simParameters.NumCW = 2; % Number of codewords
    simParameters.PDSCH.TargetCodeRate = [490 490]./1024; % Code rate used to calculate transport block
    simParameters.PDSCH.Modulation = {'16QAM', '16QAM'}; % 'QPSK', '16QAM', '64QAM', '256QAM'
    simParameters.NRxAnts = 8; % Number of UE receive antennas (even or odd)
else
    simParameters.NumCW = 1; % Number of codewords
    simParameters.PDSCH.TargetCodeRate = 490/1024; % Code rate used to calculate transport block
    simParameters.PDSCH.Modulation = '16QAM'; % 'QPSK', '16QAM', '64QAM', '256QAM'
    simParameters.NRxAnts = 2; % Number of UE receive antennas (1 or even number)
end

% DM-RS and antenna port configuration (TS 38.211 Section 7.4.1.1)
simParameters.PDSCH.PortSet = 0:simParameters.PDSCH.NLayers-1; % DM-RS ports to use for the layer
simParameters.PDSCH.PDSCHMappingType = 'A'; % PDSCH mapping type ('A'(slot-wise), 'B'(non slot-wise))
simParameters.PDSCH.DMRSTypeAPosition = 2; % Mapping type A only. First DM-RS symbol position
simParameters.PDSCH.DMRSLength = 1; % Number of front-loaded DM-RS symbols (1(single), 2(double))
simParameters.PDSCH.DMRSAdditionalPosition = 0; % Additional DM-RS symbol positions (max range 0..13)
simParameters.PDSCH.DMRSConfigurationType = 2; % DM-RS configuration type (1,2)
simParameters.PDSCH.NumCDMGroupsWithoutData = 1; % Number of CDM groups without data
simParameters.PDSCH.NIDNSCID = 1; % Scrambling identity (0..65535)
simParameters.PDSCH.NSCID = 0; % Scrambling initialization (0,1)
% Reserved PRB patterns (for CORESETs, forward compatibility etc)
simParameters.PDSCH.Reserved.Symbols = []; % Reserved PDSCH symbols
simParameters.PDSCH.Reserved.PRBS = []; % Reserved PDSCH PRBs
simParameters.PDSCH.Reserved.Period = []; % Periodicity of reserved resources
% PDSCH resource block mapping (TS 38.211 Section 7.3.1.6)
simParameters.PDSCH.VRBToPRBInterleaving = 0; % Disable interleaved resource mapping
simParameters.PDSCH.VRBBundleSize = 4;
% PDSCH PRB bundling (TS 38.214 Section 5.1.2.3)
simParameters.PDSCH.PRGBundleSize = []; % 2, 4, or [] to signify "wideband"
% PT-RS configuration (TS 38.211 Section 7.4.1.2)
simParameters.PDSCH.EnablePTRS = 0; % Enable or disable PT-RS (1 or 0)
simParameters.PDSCH.PTRSTimeDensity = 1; % PT-RS time density (L_PT-RS) (1, 2, 4)
simParameters.PDSCH.PTRSFrequencyDensity = 2; % PT-RS frequency density (K_PT-RS) (2 or 4)
simParameters.PDSCH.PTRSREOffset = '00'; % PT-RS resource element offset ('00', '01', '10', '11')
simParameters.PDSCH.PTRSPortSet = []; % PT-RS antenna port, subset of DM-RS port set.

% Define the propagation channel type
simParameters.ChannelType = 'CDL'; % 'CDL' or 'TDL'

% SS burst configuration
simParameters.SSBurst.BlockPattern = 'Case B'; % 30kHz subcarrier spacing
simParameters.SSBurst.SSBTransmitted = [0 1 0 1]; % Bitmap indicating blocks transmitted in the burst
simParameters.SSBurst.SSBPeriodicity = 20; % SS burst set periodicity in ms (5, 10, 20, 40)

validateNLayers(simParameters);

Create carrier configuration object carrier and PDSCH configuration structure pdsch.

```



```

carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = simParameters.SubcarrierSpacing;
carrier.CyclicPrefix = simParameters.CyclicPrefix;
carrier.NSizeGrid = simParameters.NRB;
carrier.NCellID = simParameters.NCellID;
pdsch = simParameters.PDSCH;

% Specify additional required fields for PDSCH
pdsch.RNTI = 1;

Xoh_PDSCH = 6*simParameters.PDSCH.EnablePTRS;      % Set Xoh-PDSCH overhead to 0 when PT-RS is di

```

### Propagation Channel Model Configuration

Create the channel model object. Both CDL and TDL channel models are supported [ 5 ].

```

snrIn = simParameters.SNRIn;
nTxAnts = simParameters.NTxAnts;
nRxAnts = simParameters.NRxAnts;
channelType = simParameters.ChannelType;

if strcmpi(channelType,'CDL')
    channel = nrCDLChannel; % CDL channel object

    % Use CDL-C model (Urban macrocell model)
    channel.DelayProfile = 'CDL-C';
    channel.DelaySpread = 300e-9;

    % Turn the overall number of antennas into a specific antenna panel
    % array geometry. The number of antennas configured is updated when
    % nTxAnts is not one of (1,2,4,8,16,32,64,128,256,512,1024) or nRxAnts
    % is not 1 or even.
    [channel.TransmitAntennaArray.Size, channel.ReceiveAntennaArray.Size] = ...
        hArrayGeometry(nTxAnts,nRxAnts);
    nTxAnts = prod(channel.TransmitAntennaArray.Size);
    nRxAnts = prod(channel.ReceiveAntennaArray.Size);
    simParameters.NTxAnts = nTxAnts;
    simParameters.NRxAnts = nRxAnts;

elseif strcmpi(channelType,'TDL')
    channel = nrTDLChannel; % TDL channel object
    % Set the channel geometry
    channel.DelayProfile = 'TDL-C';
    channel.DelaySpread = 300e-9;
    channel.NumTransmitAntennas = nTxAnts;
    channel.NumReceiveAntennas = nRxAnts;
else
    error('ChannelType parameter field must be either CDL or TDL.');
```

The sample rate for the channel model is set using the value returned from nrOFDMInfo.

```

waveformInfo = nrOFDMInfo(carrier);
channel.SampleRate = waveformInfo.SampleRate;
```

Get the maximum number of delayed samples by a channel multipath component. This is calculated from the channel path with the largest delay and the implementation delay of the channel filter. This is required later to flush the channel filter to obtain the received signal.

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

### Reserve PDSCH Resources Corresponding to SS burst

This section shows how to reserve resources for the transmission of the SS burst.

```
% Create SS burst information structure
ssburst = simParameters.SSBurst;
ssburst.NCellID = carrier.NCellID;
ssburst.SampleRate = waveformInfo.SampleRate;

ssbInfo = hSSBurstInfo(ssburst);

% Map the occupied subcarriers and transmitted symbols of the SS burst
% (defined in the SS burst numerology) to PDSCH PRBs and symbols in the
% data numerology
[mappedPRB,mappedSymbols] = mapNumerology(ssbInfo.OccupiedSubcarriers,ssbInfo.OccupiedSymbols,ss

% Configure the PDSCH to reserve these resources so that the PDSCH
% transmission does not overlap the SS burst
reservation.Symbols = mappedSymbols;
reservation.PRB = mappedPRB;
reservation.Period = ssburst.SSBPeriodicity * (carrier.SubcarrierSpacing/15); % Period in slots
pdsch.Reserved(end+1) = reservation;
```

### Processing Loop

To determine the throughput at each SNR point, analyze the PDSCH data per transmission instance using the following steps:

- *Update current HARQ process.* Check the CRC of the previous transmission for the given HARQ process. Determine whether a retransmission is required. If retransmission is not required, generate new data.
- *Resource grid generation.* Perform channel coding by calling the nrDLSCH System object. The object operates on the input transport block and keeps an internal copy of the transport block in case a retransmission is required. Modulate the coded bits on the PDSCH by using the nrPDSCH function. Then apply precoding to the resulting signal.
- *Waveform generation.* OFDM modulate the generated grid.
- *Noisy channel modeling.* Pass the waveform through a CDL or TDL fading channel. Add AWGN. The SNR is defined per RE at each UE antenna. For an SNR of 0 dB the signal and noise contribute equally to the energy per PDSCH RE per receive antenna.
- *Perform synchronization and OFDM demodulation.* For perfect synchronization, reconstruct the channel impulse response to synchronize the received waveform. For practical synchronization, correlate the received waveform with the PDSCH DM-RS. Then OFDM modulate the synchronized signal.
- *Perform channel estimation.* For perfect channel estimation, reconstruct the channel impulse response and perform OFDM demodulation. For practical channel estimation, use the PDSCH DM-RS.
- *Perform equalization and CPE compensation.* MMSE equalize the estimated channel. Estimate the common phase error (CPE) by using the PT-RS symbols, then correct the error in each OFDM symbol within the range of reference PT-RS OFDM symbols.
- *Precoding matrix calculation.* Generate the precoding matrix  $W$  for the next transmission by using singular value decomposition (SVD). A single matrix is obtained for the full allocation by averaging

the channel conditions. Therefore, for a channel with frequency selectivity,  $W$  could be less accurate for larger allocated bandwidths.

- *Decode the PDSCH.* To obtain an estimate of the received codewords, demodulate and descramble the recovered PDSCH symbols for all transmit and receive antenna pairs, along with a noise estimate, by using the `nrPDSCHDecode` function.
- *Decode DL-SCH and store the block CRC error for a HARQ process.* Pass the vector of decoded soft bits to the `nrDLSCHDecoder` System object. The object decodes the codeword and returns the block CRC error used to determine the throughput of the system.

```
% Array to store the maximum throughput for all SNR points
maxThroughput = zeros(length(snrIn),1);
% Array to store the simulation throughput for all SNR points
simThroughput = zeros(length(snrIn),1);

% Set up Redundancy Version (RV) sequence, number of HARQ processes and
% the sequence in which the HARQ processes are used
if pdsch.EnableHARQ
    % In the final report of RAN WG1 meeting #91 (R1-1719301), it was
    % observed in R1-1717405 that if performance is the priority, [0 2 3 1]
    % should be used. If self-decodability is the priority, it should be
    % taken into account that the upper limit of the code rate at which
    % each RV is self-decodable is in the following order: 0>3>>2>1
    rvSeq = [0 2 3 1];
else
    % HARQ disabled - single transmission with RV=0, no retransmissions
    rvSeq = 0;
end

% Create DLSCH encoder system object
encodeDLSCH = nrDLSCH;
encodeDLSCH.MultipleHARQProcesses = true;
encodeDLSCH.TargetCodeRate = pdsch.TargetCodeRate;

% Create DLSCH decoder system object
% Use layered belief propagation for LDPC decoding, with half the number of
% iterations as compared to the default for belief propagation decoding
decodeDLSCH = nrDLSCHDecoder;
decodeDLSCH.MultipleHARQProcesses = true;
decodeDLSCH.TargetCodeRate = pdsch.TargetCodeRate;
decodeDLSCH.LDPCDecodingAlgorithm = 'Layered belief propagation';
decodeDLSCH.MaximumLDPCIterationCount = 6;

% The temporary variables 'carrier_init', 'pdsch_init', 'ssburst_init' and
% 'decodeDLSCH_init' are used to create the temporary variables 'carrier',
% 'pdsch', 'ssburst' and 'decodeDLSCH' within the SNR loop to create
% independent instances in case of parallel simulation
carrier_init = carrier;
pdsch_init = pdsch;
ssburst_init = ssburst;
decodeDLSCH_init = clone(decodeDLSCH);

% Get the values of 'NFrames' and 'NumCW' to avoid overhead in case of
% parallel simulation
NFrames = simParameters.NFrames;
NumCW = simParameters.NumCW;
```

```
for snrIdx = 1:numel(snrIn) % comment out for parallel computing
% parfor snrIdx = 1:numel(snrIn) % uncomment for parallel computing
% To reduce the total simulation time, you can execute this loop in
% parallel by using the Parallel Computing Toolbox. Comment out the 'for'
% statement and uncomment the 'parfor' statement. If the Parallel Computing
% Toolbox is not installed, 'parfor' defaults to normal 'for' statement

    % Set the random number generator settings to default values
    rng('default');

    % Initialize variables for this SNR point, required for initialization
    % of variables when using the Parallel Computing Toolbox
    carrier = carrier_init;
    pdsch = pdsch_init;
    ssburst = ssburst_init;
    decodeDLSCH = clone(decodeDLSCH_init);
    pathFilters = [];
    ssbWaveform = [];

    SNRdB = snrIn(snrIdx);
    fprintf('\nSimulating transmission scheme 1 (%dx%d) and SCS=%dkHz with %s channel at %gdB SNR\n',
        nTxAnts,nRxAnts,carrier.SubcarrierSpacing, ...
        channelType,SNRdB,NFrames);

    % Initialize variables used in the simulation and analysis
    bitTput = []; % Number of successfully received bits per transmission
    txedTrBlkSizes = []; % Number of transmitted info bits per transmission

    % Specify the order in which we cycle through the HARQ processes
    NHARQProcesses = 16;
    harqSequence = 1:NHARQProcesses;

    % Initialize the state of all HARQ processes
    harqProcesses = hNewHARQProcesses(NHARQProcesses,rvSeq,NumCW);
    harqProcCntr = 0; % HARQ process counter

    % Reset the channel so that each SNR point will experience the same
    % channel realization
    reset(channel);

    % Total number of slots in the simulation period
    NSlots = NFrames * carrier.SlotsPerFrame;

    % Create 'gnb' structure
    gnb = struct();
    gnb.NRB = carrier.NSizeGrid;
    gnb.CyclicPrefix = carrier.CyclicPrefix;
    gnb.SubcarrierSpacing = carrier.SubcarrierSpacing;

    % Index to the start of the current set of SS burst samples to be
    % transmitted
    ssbSampleIndex = 1;

    % Obtain a precoding matrix (wtx) to be used in the transmission of the
    % first transport block
    estChannelGrid = getInitialChannelEstimate(carrier,nTxAnts,channel);
    newWtx = getPrecodingMatrix(carrier,pdsch,estChannelGrid);
```

```

% Timing offset, updated in every slot for perfect synchronization and
% when the correlation is strong for practical synchronization
offset = 0;

% Loop over the entire waveform length
for nslot = 0:NSlots-1

    % Update the carrier and PDSCH slot numbers to account for a new
    % PDSCH transmission
    carrier.NSlot = nslot;
    pdsch.NSlot = nslot;

    % Generate a new SS burst when necessary
    if (ssbSampleIndex==1)
        nSubframe = carrier.NSlot / carrier.SlotsPerSubframe;
        ssburst.NFrame = floor(nSubframe / 10);
        ssburst.NHalfFrame = mod(nSubframe / 5,2);
        [ssbWaveform,~,ssbInfo] = hSSBurst(ssburst);
    end

    % Get HARQ process index for the current PDSCH from HARQ index table
    harqProcIdx = harqSequence(mod(harqProcCntr,length(harqSequence))+1);

    % Update current HARQ process information (this updates the RV
    % depending on CRC pass or fail in the previous transmission for
    % this HARQ process)
    harqProcesses(harqProcIdx) = hUpdateHARQProcess(harqProcesses(harqProcIdx),NumCW);

    % Calculate the transport block sizes for the codewords in the slot
    [pdschIndices,dmrsIndices,dmrsSymbols,ptrsIndices,ptrsSymbols,pdschIndicesInfo] = hPDSCH
    trBlkSizes = nrTBS(pdsch.Modulation,pdsch.NLayers,numel(pdsch.PRBSets),pdschIndicesInfo.NF

    % HARQ: check CRC from previous transmission per codeword, i.e. is
    % a retransmission required?
    for cwIdx = 1:NumCW
        NDI = false;
        if harqProcesses(harqProcIdx).blkerr(cwIdx) % Errored
            if (harqProcesses(harqProcIdx).RVIdx(cwIdx)==1) % end of rvSeq
                resetSoftBuffer(decodedDLSCH,cwIdx-1,harqProcIdx-1);
                NDI = true;
            end
        else % No error
            NDI = true;
        end
        if NDI
            trBlk = randi([0 1],trBlkSizes(cwIdx),1);
            setTransportBlock(encodeDLSCH,trBlk,cwIdx-1,harqProcIdx-1);
        end
    end

    % Encode the DL-SCH transport blocks
    codedTrBlock = encodeDLSCH(pdsch.Modulation,pdsch.NLayers,...
        pdschIndicesInfo.G,harqProcesses(harqProcIdx).RV,harqProcIdx-1);

    % Get wtx (precoding matrix) calculated in previous slot
    wtx = newWtx;

    % PDSCH modulation and precoding

```

```

pdschSymbols = nrPDSCH(codedTrBlock,pdsch.Modulation,pdsch.NLayers,carrier.NCellID,pdsch
pdschGrid = nrResourceGrid(carrier,nTxAnts);
[pdschSymbols,pdschAntIndices] = hPRGPrecode(size(pdschGrid),carrier.NStartGrid,pdschSym

% PDSCH mapping in grid associated with PDSCH transmission period
pdschGrid(pdschAntIndices) = pdschSymbols;

% PDSCH DM-RS precoding and mapping
[dmrsAntSymbols,dmrsAntIndices] = hPRGPrecode(size(pdschGrid),carrier.NStartGrid,dmrsSym
pdschGrid(dmrsAntIndices) = dmrsAntSymbols;

% PDSCH PT-RS precoding and mapping
[ptrsAntSymbols,ptrsAntIndices] = hPRGPrecode(size(pdschGrid),carrier.NStartGrid,ptrsSym
pdschGrid(ptrsAntIndices) = ptrsAntSymbols;

% OFDM modulation of associated resource elements
txWaveform = nrOFDMModulate(carrier, pdschGrid);

% Add the appropriate portion of SS burst waveform to the
% transmitted waveform
Nt = size(txWaveform,1);
txWaveform = txWaveform + ssbWaveform(ssbSampleIndex + (0:Nt-1),:);
ssbSampleIndex = mod(ssbSampleIndex + Nt,size(ssbWaveform,1));

% Pass data through channel model. Append zeros at the end of the
% transmitted waveform to flush channel content. These zeros take
% into account any delay introduced in the channel. This is a mix
% of multipath delay and implementation delay. This value may
% change depending on the sampling rate, delay profile and delay
% spread
txWaveform = [txWaveform; zeros(maxChDelay, size(txWaveform,2))];
[rxWaveform,pathGains,sampleTimes] = channel(txWaveform);

% Add AWGN to the received time domain waveform
% Normalize noise power by the IFFT size used in OFDM modulation,
% as the OFDM modulator applies this normalization to the
% transmitted waveform. Also normalize by the number of receive
% antennas, as the channel model applies this normalization to the
% received waveform by default
SNR = 10^(SNRdB/20); % Calculate linear noise gain
N0 = 1/(sqrt(2.0*nRxAnts*double(waveformInfo.Nfft))*SNR);
noise = N0*complex(randn(size(rxWaveform)),randn(size(rxWaveform)));
rxWaveform = rxWaveform + noise;

if (perfectChannelEstimator)
    % Perfect synchronization. Use information provided by the channel
    % to find the strongest multipath component
    pathFilters = getPathFilters(channel); % get path filters for perfect channel estimat
    [offset,mag] = nrPerfectTimingEstimate(pathGains,pathFilters);
else
    % Practical synchronization. Correlate the received waveform
    % with the PDSCH DM-RS to give timing offset estimate 't' and
    % correlation magnitude 'mag'. The function
    % hSkipWeakTimingOffset is used to update the receiver timing
    % offset. If the correlation peak in 'mag' is weak, the current
    % timing estimate 't' is ignored and the previous estimate
    % 'offset' is used
    [t,mag] = nrTimingEstimate(carrier,rxWaveform,dmrsIndices,dmrsSymbols); %#ok<UNRCH>

```

```

        offset = hSkipWeakTimingOffset(offset,t,mag);
    end
    rxWaveform = rxWaveform(1+offset:end, :);

    % Perform OFDM demodulation on the received data to recreate the
    % resource grid, including padding in the event that practical
    % synchronization results in an incomplete slot being demodulated
    rxGrid = nrOFDMDemodulate(carrier, rxWaveform);
    [K,L,R] = size(rxGrid);
    if (L < carrier.SymbolsPerSlot)
        rxGrid = cat(2,rxGrid,zeros(K,carrier.SymbolsPerSlot-L,R));
    end

    if (perfectChannelEstimator)
        % Perfect channel estimation, using the value of the path gains
        % provided by the channel. This channel estimate does not
        % include the effect of transmitter precoding
        estChannelGrid = nrPerfectChannelEstimate(carrier,pathGains,pathFilters,offset,sampleRate);

        % Get perfect noise estimate (from the noise realization)
        noiseGrid = nrOFDMDemodulate(carrier,noise(1+offset:end, :));
        noiseEst = var(noiseGrid(:));

        % Get precoding matrix for next slot
        newWtx = getPrecodingMatrix(carrier,pdsch,estChannelGrid);

        % Get PDSCH resource elements from the received grid and
        % channel estimate
        [pdschRx,pdschHest,~,pdschHestIndices] = nrExtractResources(pdschIndices,rxGrid,estChannelGrid);

        % Apply precoding to channel estimate
        pdschHest = hPRGPrecode(size(estChannelGrid),carrier.NStartGrid,pdschHest,pdschHestIndices);
    else
        % Practical channel estimation between the received grid and
        % each transmission layer, using the PDSCH DM-RS for each
        % layer. This channel estimate includes the effect of
        % transmitter precoding
        [estChannelGrid,noiseEst] = nrChannelEstimate(carrier,rxGrid,dmrsIndices,dmrsSymbols);

        % Get PDSCH resource elements from the received grid and
        % channel estimate
        [pdschRx,pdschHest] = nrExtractResources(pdschIndices,rxGrid,estChannelGrid);

        % Remove precoding from estChannelGrid prior to precoding
        % matrix calculation
        estChannelGridPorts = precodingChannelEstimate(carrier,estChannelGrid,conj(wtx));

        % Get precoding matrix for next slot
        newWtx = getPrecodingMatrix(carrier,pdsch,estChannelGridPorts);
    end

    % Equalization
    [pdschEq,csi] = nrEqualizeMMSE(pdschRx,pdschHest,noiseEst);

    % Common phase error (CPE) compensation
    if ~isempty(ptrsIndices)
        % Initialize temporary grid to store equalized symbols
        tempGrid = nrResourceGrid(carrier,pdsch.NLayers);
    end

```

```

% Extract PT-RS symbols from received grid and estimated
% channel grid
[ptrsRx,ptrsHest,~,~,ptrsHestIndices,ptrsLayerIndices] = nrExtractResources(ptrsIndices,ptrsGrid);

if (perfectChannelEstimator)
    % Apply precoding to channel estimate
    ptrsHest = hPRGPrecode(size(estChannelGrid),carrier.NStartGrid,ptrsHest,ptrsHestIndices);
end

% Equalize PT-RS symbols and map them to tempGrid
ptrsEq = nrEqualizeMMSE(ptrsRx,ptrsHest,noiseEst);
tempGrid(ptrsLayerIndices) = ptrsEq;

% Estimate the residual channel at the PT-RS locations in
% tempGrid
cpe = nrChannelEstimate(tempGrid,ptrsIndices,ptrsSymbols);

% Sum estimates across subcarriers, receive antennas, and
% layers. Then, get the CPE by taking the angle of the
% resultant sum
cpe = angle(sum(cpe,[1 3 4]));

% Map the equalized PDSCH symbols to tempGrid
tempGrid(pdschIndices) = pdschEq;

% Correct CPE in each OFDM symbol within the range of reference
% PT-RS OFDM symbols
symLoc = pdschIndicesInfo.PTRSSymbolSet(1)+1:pdschIndicesInfo.PTRSSymbolSet(end)+1;
tempGrid(:,symLoc,:) = tempGrid(:,symLoc,:).*exp(-1i*cpe(symLoc));

% Extract PDSCH symbols
pdschEq = tempGrid(pdschIndices);
end

% Decode PDSCH physical channel
[dlschLLRs,rxSymbols] = nrPDSCHDecode(pdschEq,pdsch.Modulation,carrier.NCellID,pdsch.RNTI);

% Scale LLRs by CSI
csi = nrLayerDemap(csi); % CSI layer demapping
for cwIdx = 1:NumCW
    Qm = length(dlschLLRs{cwIdx})/length(rxSymbols{cwIdx}); % bits per symbol
    csi{cwIdx} = repmat(csi{cwIdx}.',Qm,1); % expand by each bit per symbol
    dlschLLRs{cwIdx} = dlschLLRs{cwIdx} .* csi{cwIdx}(:); % scale
end

% Decode the DL-SCH transport channel
decodeDLSCH.TransportBlockLength = trBlkSizes;
[decbits,harqProcesses(harqProcIdx).blkerr] = decodeDLSCH(dlschLLRs,pdsch.Modulation,pdsch.RNTI);

% Store values to calculate throughput (only for active PDSCH instances)
if(any(trBlkSizes ~= 0))
    bitTput = [bitTput trBlkSizes.*(1-harqProcesses(harqProcIdx).blkerr)];
    txdTrBlkSizes = [txdTrBlkSizes trBlkSizes];
end

% Update HARQ process counter
harqProcCntr = harqProcCntr + 1;

```



```

% Display transport block error information per codeword managed by current HARQ process
fprintf('\n(%3.2f%%) HARQ Proc %d: ',100*(nslot+1)/NSlots,harqProcIdx);
estrings = {'passed','failed'};
rvi = harqProcesses(harqProcIdx).RVIdx;
for cw=1:length(rvi)
    cwrvi = rvi(cw);
    % Create a report on the RV state given position in RV sequence and decoding error
    if cwrvi == 1
        ts = sprintf('Initial transmission (RV=%d)',rvSeq(cwrvi));
    else
        ts = sprintf('Retransmission #%d (RV=%d)',cwrvi-1,rvSeq(cwrvi));
    end
    fprintf('CW%d:%s %s. ',cw-1,ts,estrings{1+harqProcesses(harqProcIdx).blkerr(cw)});
end

end

% Calculate maximum and simulated throughput
maxThroughput(snrIdx) = sum(txedTrBlkSizes); % Max possible throughput
simThroughput(snrIdx) = sum(bitTput,2);      % Simulated throughput

% Display the results dynamically in the command window
fprintf(['\n\nThroughput(Mbps) for ', num2str(NFrames) ' frame(s) '],...
        '= %.4f\n'], 1e-6*simThroughput(snrIdx)/(NFrames*10e-3));
fprintf(['Throughput(%%) for ', num2str(NFrames) ' frame(s) = %.4f\n'],...
        simThroughput(snrIdx)*100/maxThroughput(snrIdx));

end

```

Simulating transmission scheme 1 (8x2) and SCS=30kHz with CDL channel at -5dB SNR for 2 10ms frames

```

(2.50%) HARQ Proc 1: CW0:Initial transmission (RV=0) failed.
(5.00%) HARQ Proc 2: CW0:Initial transmission (RV=0) failed.
(7.50%) HARQ Proc 3: CW0:Initial transmission (RV=0) failed.
(10.00%) HARQ Proc 4: CW0:Initial transmission (RV=0) failed.
(12.50%) HARQ Proc 5: CW0:Initial transmission (RV=0) failed.
(15.00%) HARQ Proc 6: CW0:Initial transmission (RV=0) failed.
(17.50%) HARQ Proc 7: CW0:Initial transmission (RV=0) failed.
(20.00%) HARQ Proc 8: CW0:Initial transmission (RV=0) failed.
(22.50%) HARQ Proc 9: CW0:Initial transmission (RV=0) failed.
(25.00%) HARQ Proc 10: CW0:Initial transmission (RV=0) failed.
(27.50%) HARQ Proc 11: CW0:Initial transmission (RV=0) failed.
(30.00%) HARQ Proc 12: CW0:Initial transmission (RV=0) failed.
(32.50%) HARQ Proc 13: CW0:Initial transmission (RV=0) failed.
(35.00%) HARQ Proc 14: CW0:Initial transmission (RV=0) failed.
(37.50%) HARQ Proc 15: CW0:Initial transmission (RV=0) failed.
(40.00%) HARQ Proc 16: CW0:Initial transmission (RV=0) failed.
(42.50%) HARQ Proc 1: CW0:Retransmission #1 (RV=2) passed.
(45.00%) HARQ Proc 2: CW0:Retransmission #1 (RV=2) passed.
(47.50%) HARQ Proc 3: CW0:Retransmission #1 (RV=2) passed.
(50.00%) HARQ Proc 4: CW0:Retransmission #1 (RV=2) passed.
(52.50%) HARQ Proc 5: CW0:Retransmission #1 (RV=2) passed.
(55.00%) HARQ Proc 6: CW0:Retransmission #1 (RV=2) passed.
(57.50%) HARQ Proc 7: CW0:Retransmission #1 (RV=2) passed.
(60.00%) HARQ Proc 8: CW0:Retransmission #1 (RV=2) passed.
(62.50%) HARQ Proc 9: CW0:Retransmission #1 (RV=2) passed.

```

```
(65.00%) HARQ Proc 10: CW0:Retransmission #1 (RV=2) passed.
(67.50%) HARQ Proc 11: CW0:Retransmission #1 (RV=2) passed.
(70.00%) HARQ Proc 12: CW0:Retransmission #1 (RV=2) passed.
(72.50%) HARQ Proc 13: CW0:Retransmission #1 (RV=2) passed.
(75.00%) HARQ Proc 14: CW0:Retransmission #1 (RV=2) passed.
(77.50%) HARQ Proc 15: CW0:Retransmission #1 (RV=2) passed.
(80.00%) HARQ Proc 16: CW0:Retransmission #1 (RV=2) passed.
(82.50%) HARQ Proc 1: CW0:Initial transmission (RV=0) failed.
(85.00%) HARQ Proc 2: CW0:Initial transmission (RV=0) failed.
(87.50%) HARQ Proc 3: CW0:Initial transmission (RV=0) failed.
(90.00%) HARQ Proc 4: CW0:Initial transmission (RV=0) failed.
(92.50%) HARQ Proc 5: CW0:Initial transmission (RV=0) failed.
(95.00%) HARQ Proc 6: CW0:Initial transmission (RV=0) failed.
(97.50%) HARQ Proc 7: CW0:Initial transmission (RV=0) failed.
(100.00%) HARQ Proc 8: CW0:Initial transmission (RV=0) failed.
```

Throughput(Mbps) for 2 frame(s) = 24.1728

Throughput(%) for 2 frame(s) = 40.0000

Simulating transmission scheme 1 (8x2) and SCS=30kHz with CDL channel at 0dB SNR for 2 10ms frame

```
(2.50%) HARQ Proc 1: CW0:Initial transmission (RV=0) passed.
(5.00%) HARQ Proc 2: CW0:Initial transmission (RV=0) passed.
(7.50%) HARQ Proc 3: CW0:Initial transmission (RV=0) passed.
(10.00%) HARQ Proc 4: CW0:Initial transmission (RV=0) passed.
(12.50%) HARQ Proc 5: CW0:Initial transmission (RV=0) passed.
(15.00%) HARQ Proc 6: CW0:Initial transmission (RV=0) passed.
(17.50%) HARQ Proc 7: CW0:Initial transmission (RV=0) passed.
(20.00%) HARQ Proc 8: CW0:Initial transmission (RV=0) passed.
(22.50%) HARQ Proc 9: CW0:Initial transmission (RV=0) passed.
(25.00%) HARQ Proc 10: CW0:Initial transmission (RV=0) passed.
(27.50%) HARQ Proc 11: CW0:Initial transmission (RV=0) passed.
(30.00%) HARQ Proc 12: CW0:Initial transmission (RV=0) passed.
(32.50%) HARQ Proc 13: CW0:Initial transmission (RV=0) passed.
(35.00%) HARQ Proc 14: CW0:Initial transmission (RV=0) passed.
(37.50%) HARQ Proc 15: CW0:Initial transmission (RV=0) passed.
(40.00%) HARQ Proc 16: CW0:Initial transmission (RV=0) passed.
(42.50%) HARQ Proc 1: CW0:Initial transmission (RV=0) passed.
(45.00%) HARQ Proc 2: CW0:Initial transmission (RV=0) passed.
(47.50%) HARQ Proc 3: CW0:Initial transmission (RV=0) passed.
(50.00%) HARQ Proc 4: CW0:Initial transmission (RV=0) passed.
(52.50%) HARQ Proc 5: CW0:Initial transmission (RV=0) passed.
(55.00%) HARQ Proc 6: CW0:Initial transmission (RV=0) passed.
(57.50%) HARQ Proc 7: CW0:Initial transmission (RV=0) passed.
(60.00%) HARQ Proc 8: CW0:Initial transmission (RV=0) passed.
(62.50%) HARQ Proc 9: CW0:Initial transmission (RV=0) passed.
(65.00%) HARQ Proc 10: CW0:Initial transmission (RV=0) passed.
(67.50%) HARQ Proc 11: CW0:Initial transmission (RV=0) passed.
(70.00%) HARQ Proc 12: CW0:Initial transmission (RV=0) passed.
(72.50%) HARQ Proc 13: CW0:Initial transmission (RV=0) passed.
(75.00%) HARQ Proc 14: CW0:Initial transmission (RV=0) passed.
(77.50%) HARQ Proc 15: CW0:Initial transmission (RV=0) passed.
(80.00%) HARQ Proc 16: CW0:Initial transmission (RV=0) passed.
(82.50%) HARQ Proc 1: CW0:Initial transmission (RV=0) passed.
(85.00%) HARQ Proc 2: CW0:Initial transmission (RV=0) passed.
(87.50%) HARQ Proc 3: CW0:Initial transmission (RV=0) passed.
(90.00%) HARQ Proc 4: CW0:Initial transmission (RV=0) passed.
(92.50%) HARQ Proc 5: CW0:Initial transmission (RV=0) passed.
```

(95.00%) HARQ Proc 6: CW0:Initial transmission (RV=0) passed.  
 (97.50%) HARQ Proc 7: CW0:Initial transmission (RV=0) passed.  
 (100.00%) HARQ Proc 8: CW0:Initial transmission (RV=0) passed.

Throughput(Mbps) for 2 frame(s) = 60.4320

Throughput(%) for 2 frame(s) = 100.0000

Simulating transmission scheme 1 (8x2) and SCS=30kHz with CDL channel at 5dB SNR for 2 10ms frame

(2.50%) HARQ Proc 1: CW0:Initial transmission (RV=0) passed.  
 (5.00%) HARQ Proc 2: CW0:Initial transmission (RV=0) passed.  
 (7.50%) HARQ Proc 3: CW0:Initial transmission (RV=0) passed.  
 (10.00%) HARQ Proc 4: CW0:Initial transmission (RV=0) passed.  
 (12.50%) HARQ Proc 5: CW0:Initial transmission (RV=0) passed.  
 (15.00%) HARQ Proc 6: CW0:Initial transmission (RV=0) passed.  
 (17.50%) HARQ Proc 7: CW0:Initial transmission (RV=0) passed.  
 (20.00%) HARQ Proc 8: CW0:Initial transmission (RV=0) passed.  
 (22.50%) HARQ Proc 9: CW0:Initial transmission (RV=0) passed.  
 (25.00%) HARQ Proc 10: CW0:Initial transmission (RV=0) passed.  
 (27.50%) HARQ Proc 11: CW0:Initial transmission (RV=0) passed.  
 (30.00%) HARQ Proc 12: CW0:Initial transmission (RV=0) passed.  
 (32.50%) HARQ Proc 13: CW0:Initial transmission (RV=0) passed.  
 (35.00%) HARQ Proc 14: CW0:Initial transmission (RV=0) passed.  
 (37.50%) HARQ Proc 15: CW0:Initial transmission (RV=0) passed.  
 (40.00%) HARQ Proc 16: CW0:Initial transmission (RV=0) passed.  
 (42.50%) HARQ Proc 1: CW0:Initial transmission (RV=0) passed.  
 (45.00%) HARQ Proc 2: CW0:Initial transmission (RV=0) passed.  
 (47.50%) HARQ Proc 3: CW0:Initial transmission (RV=0) passed.  
 (50.00%) HARQ Proc 4: CW0:Initial transmission (RV=0) passed.  
 (52.50%) HARQ Proc 5: CW0:Initial transmission (RV=0) passed.  
 (55.00%) HARQ Proc 6: CW0:Initial transmission (RV=0) passed.  
 (57.50%) HARQ Proc 7: CW0:Initial transmission (RV=0) passed.  
 (60.00%) HARQ Proc 8: CW0:Initial transmission (RV=0) passed.  
 (62.50%) HARQ Proc 9: CW0:Initial transmission (RV=0) passed.  
 (65.00%) HARQ Proc 10: CW0:Initial transmission (RV=0) passed.  
 (67.50%) HARQ Proc 11: CW0:Initial transmission (RV=0) passed.  
 (70.00%) HARQ Proc 12: CW0:Initial transmission (RV=0) passed.  
 (72.50%) HARQ Proc 13: CW0:Initial transmission (RV=0) passed.  
 (75.00%) HARQ Proc 14: CW0:Initial transmission (RV=0) passed.  
 (77.50%) HARQ Proc 15: CW0:Initial transmission (RV=0) passed.  
 (80.00%) HARQ Proc 16: CW0:Initial transmission (RV=0) passed.  
 (82.50%) HARQ Proc 1: CW0:Initial transmission (RV=0) passed.  
 (85.00%) HARQ Proc 2: CW0:Initial transmission (RV=0) passed.  
 (87.50%) HARQ Proc 3: CW0:Initial transmission (RV=0) passed.  
 (90.00%) HARQ Proc 4: CW0:Initial transmission (RV=0) passed.  
 (92.50%) HARQ Proc 5: CW0:Initial transmission (RV=0) passed.  
 (95.00%) HARQ Proc 6: CW0:Initial transmission (RV=0) passed.  
 (97.50%) HARQ Proc 7: CW0:Initial transmission (RV=0) passed.  
 (100.00%) HARQ Proc 8: CW0:Initial transmission (RV=0) passed.

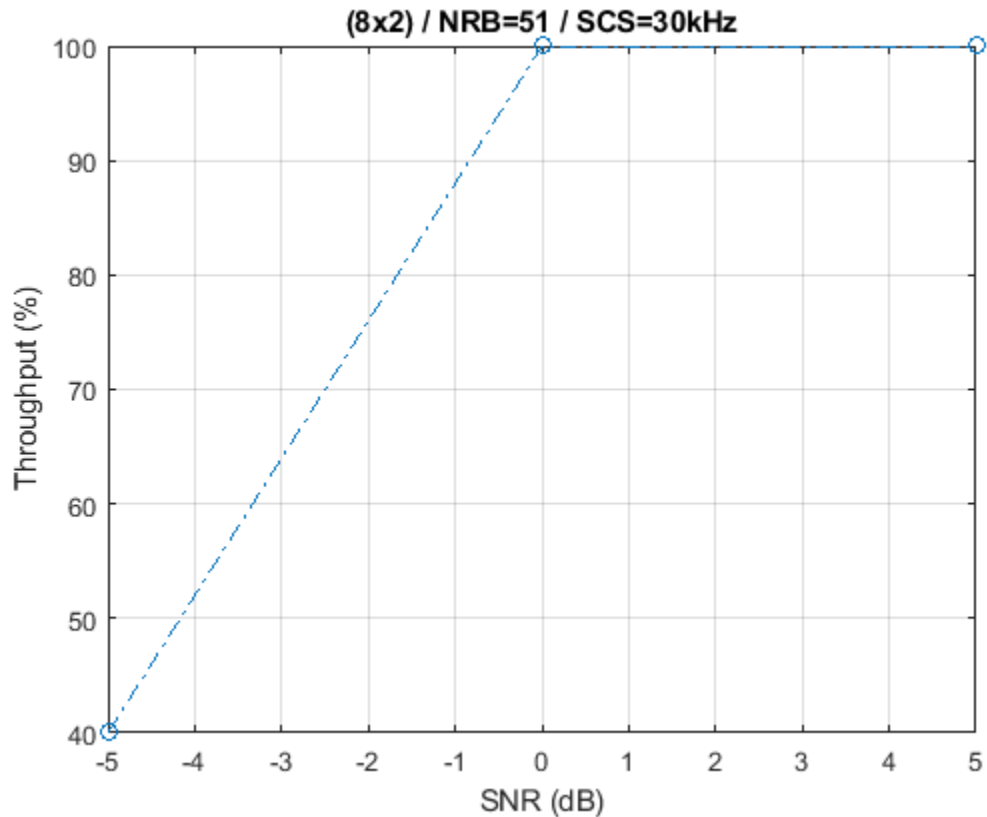
Throughput(Mbps) for 2 frame(s) = 60.4320

Throughput(%) for 2 frame(s) = 100.0000

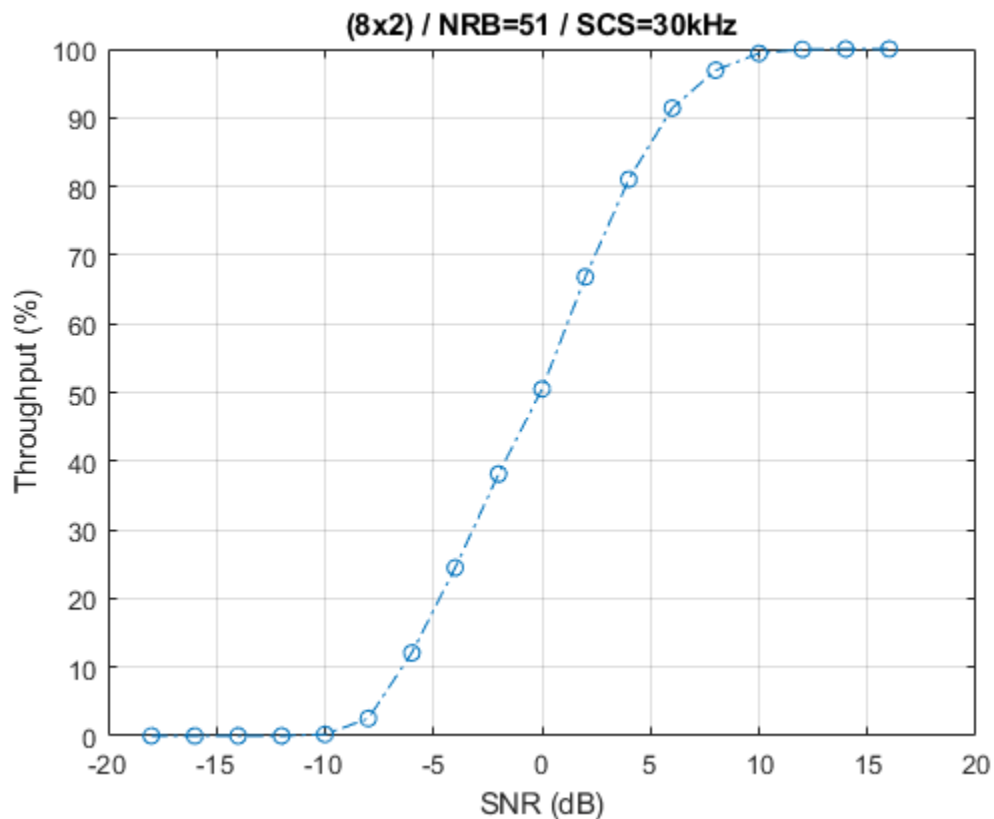
## Results

Display the measured throughput. This is calculated as the percentage of the maximum possible throughput of the link given the available resources for data transmission.

```
figure;  
plot(snrIn,simThroughput*100./maxThroughput,'o-.')  
xlabel('SNR (dB)'); ylabel('Throughput (%)'); grid on;  
title(sprintf('%dx%d) / NRB=%d / SCS=%dkHz',...  
            nTxAnts,nRxAnts,carrier_init.NSizeGrid,carrier_init.SubcarrierSpacing));  
  
% Bundle key parameters and results into a combined structure for recording  
simResults.simParameters = simParameters;  
simResults.simThroughput = simThroughput;
```



The figure below shows throughput results obtained simulating 10000 subframes (NFrames = 1000, SNRIn = -18:2:16).



## Appendix

This example uses the following helper functions:

- hArrayGeometry.m
- hNewHARQProcesses.m
- hPDSCHResources.m
- hPRGPrecode.m
- hSkipWeakTimingOffset.m
- hSSBurst.m
- hUpdateHARQProcess.m

## Selected Bibliography

- 1 3GPP TS 38.211. "NR; Physical channels and modulation (Release 15)." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.
- 2 3GPP TS 38.212. "NR; Multiplexing and channel coding (Release 15)." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.
- 3 3GPP TS 38.213. "NR; Physical layer procedures for control (Release 15)." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.
- 4 3GPP TS 38.214. "NR; Physical layer procedures for data (Release 15)." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.

- 5 3GPP TR 38.901. "Study on channel model for frequencies from 0.5 to 100 GHz (Release 15)."  
3rd Generation Partnership Project; Technical Specification Group Radio Access Network.

### Local Functions

```
function validateNLayers(simParameters)
% Validate the number of layers
    if length(simParameters.PDSCH.PortSet)~= simParameters.PDSCH.NLayers
        error('The number of elements of PortSet (%d) must be the same as the number of layers (%d)',
            length(simParameters.PDSCH.PortSet), simParameters.PDSCH.NLayers);
    end

    if simParameters.PDSCH.NLayers > min(simParameters.NTxAnts,simParameters.NRxAnts)
        error('The number of layers (%d) must satisfy NLayers <= min(NTxAnts,NRxAnts) = min(%d,%d)',
            simParameters.PDSCH.NLayers,simParameters.NTxAnts,simParameters.NRxAnts,min(simParameters.NTxAnts,simParameters.NRxAnts));
    end
end

function estChannelGrid = getInitialChannelEstimate(carrier,nTxAnts,channel)
% Obtain channel estimate before first transmission. This can be used to
% obtain a precoding matrix for the first slot.

    ofdmInfo = nrOFDMInfo(carrier);

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

    % Temporary waveform (only needed for the sizes)
    tmpWaveform = zeros((ofdmInfo.SampleRate/1000/carrier.SlotsPerSubframe)+maxChDelay,nTxAnts);

    % Filter through channel
    [~,pathGains,sampleTimes] = channel(tmpWaveform);

    % Perfect timing synch
    pathFilters = getPathFilters(channel);
    offset = nrPerfectTimingEstimate(pathGains,pathFilters);

    % Perfect channel estimate
    estChannelGrid = nrPerfectChannelEstimate(carrier,pathGains,pathFilters,offset,sampleTimes);
end

function wtx = getPrecodingMatrix(carrier,pdsch,hestGrid)
% Calculate precoding matrices for all PRGs in the carrier that overlap
% with the PDSCH allocation

    % Maximum CRB addressed by carrier grid
    maxCRB = carrier.NStartGrid + carrier.NSizeGrid - 1;

    % PRG size
    if (isfield(pdsch,'PRGBundleSize') && ~isempty(pdsch.PRGBundleSize))
        Pd_BWP = pdsch.PRGBundleSize;
    else
        Pd_BWP = maxCRB + 1;
    end

    % PRG numbers (1-based) for each RB in the carrier grid
    NPRG = ceil((maxCRB + 1) / Pd_BWP);
```

```

prgset = repmat((1:NPRG),Pd_BWP,1);
prgset = prgset(carrier.NStartGrid + (1:carrier.NSizeGrid).');

[~,~,R,P] = size(hestGrid);
wtx = zeros([pdsch.NLayers P NPRG]);
for i = 1:NPRG

    % Subcarrier indices within current PRG and within the PDSCH
    % allocation
    thisPRG = find(prgset==i) - 1;
    thisPRG = intersect(thisPRG,pdsch.PRBSets(:) + carrier.NStartGrid,'rows');
    prgSc = (1:12)' + 12*thisPRG';
    prgSc = prgSc(:);

    if (~isempty(prgSc))

        % Average channel estimate in PRG
        estAllocGrid = hestGrid(prgSc, :, :, :);
        Hest = permute(mean(reshape(estAllocGrid, [], R, P)), [2 3 1]);

        % SVD decomposition
        [~,~,V] = svd(Hest);
        wtx(:, :, i) = V(:, 1:pdsch.NLayers).';

    end

end

wtx = wtx / sqrt(pdsch.NLayers); % Normalize by NLayers

end

function estChannelGrid = precodeChannelEstimate(carrier,estChannelGrid,W)
% Apply precoding matrix W to the last dimension of the channel estimate

[K,L,R,P] = size(estChannelGrid);
estChannelGrid = reshape(estChannelGrid,[K*L R P]);
estChannelGrid = hPRGPrecoder([K L R P],carrier.NStartGrid,estChannelGrid,reshape(1:numel(estChannelGrid),[K L R P]));
estChannelGrid = reshape(estChannelGrid,K,L,R,[]);

end

function [mappedPRB,mappedSymbols] = mapNumerology(subcarriers,symbols,nrbs,nrbs,fs,ft)
% Map the SSBurst numerology to PDSCH numerology. The outputs are:
% - mappedPRB: 0-based PRB indices for carrier resource grid (arranged in a column)
% - mappedSymbols: 0-based OFDM symbol indices in a slot for carrier resource grid (arranged in a column)
% carrier resource grid is sized using gnb.NRB, gnb.CyclicPrefix, spanning 1 slot
% The input parameters are:
% - subcarriers: 1-based row subscripts for SSB resource grid (arranged in a column)
% - symbols: 1-based column subscripts for SSB resource grid (arranged in an N-by-4 matrix, 4 :
% SSB resource grid is sized using ssbInfo.NRB, normal CP, spanning 5 subframes
% - nrbs: source (SSB) NRB
% - nrbs: target (carrier) NRB
% - fs: source (SSB) SCS
% - ft: target (carrier) SCS

mappedPRB = unique(fix((subcarriers-(nrbs*6) - 1)*fs/(ft*12) + nrbs/2),'stable');

```

```
symbols = symbols.';
symbols = symbols(:).' - 1;

if (ft < fs)
    % If ft/fs < 1, reduction
    mappedSymbols = unique(fix(symbols*ft/fs), 'stable');
else
    % Else, repetition by ft/fs
    mappedSymbols = reshape((0:(ft/fs-1))' + symbols(:)'*ft/fs,1,[]);
end

end
```

## See Also

### Objects

[nrCDLChannel](#) | [nrTDLChannel](#)

### Functions

[nrDLSCHInfo](#) | [nrPDSCH](#) | [nrPDSCHDecode](#)



## Downlink Control Processing and Procedures

This example describes the blind search decoding of the physical downlink control channel (PDCCH) for 5G New Radio communications system. Building on the tutorial “Modeling Downlink Control Information”, this example introduces the concepts of control-resource set (CORESET) and search spaces, their generic specification and shows how a PDCCH instance is mapped to one of several candidates within a search space. To recover the transmitted control information at the receiver, the example performs a blind search over the set of candidates.

### System Parameters

Set system parameters corresponding to the carrier, CORESET, search space set, and PDCCH instance respectively.

```
rng(111); % Set RNG state for repeatability

% Carrier configuration
carrier = nrCarrierConfig;
carrier.NCellID = 2; % Cell identity
carrier.SubcarrierSpacing = 30; % Carrier/BWP Subcarrier spacing
carrier.CyclicPrefix = 'normal'; % Cyclic prefix
carrier.NSlot = 0; % Slot counter
carrier.NFrame = 0; % Frame counter
carrier.NStartGrid = 10; % Carrier offset
carrier.NSizeGrid = 48; % Size of carrier in RB

% CORESET configuration
coreset = nrCORESETConfig;
coreset.CORESETID = 1; % CORESET ID (0...11)
coreset.FrequencyResources = ones(1,4); % 6 RB sized
coreset.Duration = 1; % CORESET symbol duration (1,2,3)
coreset.CCEREGMapping = 'interleaved'; % CORESET Mapping
coreset.REGBundleSize = 2; % L (2,6) or (3,6)
coreset.InterleaverSize = 2; % R (2,3,6)
coreset.ShiftIndex = carrier.NCellID; % default to NCellID

% Search space configuration
ss = nrSearchSpaceConfig;
ss.CORESETID = 1; % Associated CORESET ID (0...11)
ss.SearchSpaceType = 'ue'; % 'ue', 'common'
ss.StartSymbolWithinSlot = 0; % Starting symbol in slot
ss.SlotPeriodAndOffset = [1 0]; % Search space period and offset
ss.Duration = 1; % Search space duration in slots
ss.NumCandidates = [4 2 1 0 0]; % For (1,2,4,8,16) levels respectively

% PDCCH configuration
pdcch = nrPDCCHConfig;
pdcch.NStartBWP = 10; % BWP offset wrt CRB 0
pdcch.NSizeBWP = 48; % Size of BWP in resource blocks
pdcch.CORESET = coreset; % Associated CORESET
pdcch.SearchSpace = ss; % Associated SearchSpace
pdcch.RNTI = 1; % C-RNTI
pdcch.DMRSScramblingID = []; % Use carrier.NCellID instead
pdcch.AggregationLevel = 4; % Number of CCEs in PDCCH (1,2,4,8,16)
pdcch.AllocatedCandidate = 1; % 1-based scalar
```

This example assumes single slot processing, using a single bandwidth part with a single PDCCH transmission for an associated CORESET and search space set.

For more information on waveform generation with multiple physical channels, see the “5G NR Downlink Carrier Waveform Generation” on page 1-2 example.

### PDCCH Bit Capacity

The bit capacity for a PDCCH instance is determined based on the number of control-channel elements (CCE) configured for the PDCCH. A CCE consists of six resource-element groups (REGs) where a REG equals one resource block (RB) during one OFDM symbol.

```
% Number of bits for PDCCH resources and actual indices
[ind,dmrs,dmrsInd] = nrPDCCHResources(carrier,pcch);
E = 2*numel(ind);
```

### DCI Encoding

The `nrDCIEncode` function encodes the DCI message bits based on a downlink format. DCI encoding includes the stages of CRC attachment, polar encoding and rate matching the codeword to the PDCCH bit capacity `E`.

```
K = 64; % Number of DCI message bits
dciBits = randi([0 1],K,1,'int8');

dciCW = nrDCIEncode(dciBits,pcch.RNTI,E);
```

### PDCCH Symbol Generation and Mapping

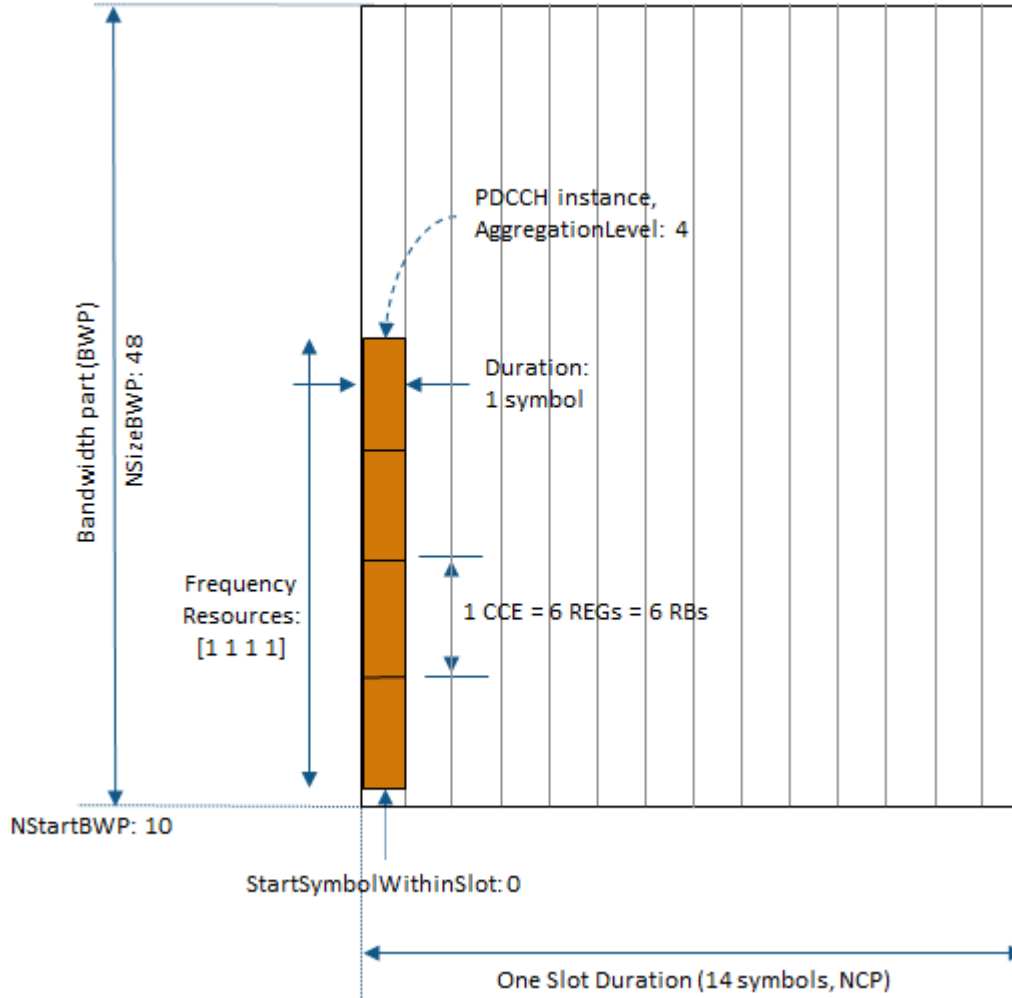
The `nrPDCCH` function maps the encoded DCI bits onto the physical downlink control channel (PDCCH). The function returns the scrambled, QPSK-modulated symbols. The scrambling accounts for the user-specific parameters.

```
if isempty(pcch.DMRSScramblingID)
    nID = carrier.NCellID;
else
    nID = pcch.DMRSScramblingID;
end
sym = nrPDCCH(dciCW,nID,pcch.RNTI);
```

The PDCCH symbols are then mapped to the resource elements corresponding to the allocated candidate within an OFDM grid. The resource grid also contains PDSCH and PBCH symbols, and other reference signal elements. For simplicity, this example only additionally maps the PDCCH DM-RS symbols to the grid.

```
carrierGrid = nrResourceGrid(carrier);
carrierGrid(ind) = sym; % PDCCH symbols
carrierGrid(dmrsInd) = dmrs; % PDCCH DM-RS
```

For a resource grid spanning the whole bandwidth part and a single slot, this figure shows some of the CORESET, search space set and PDCCH instance parameters for the selected example configuration.



## OFDM Modulation

OFDM modulate the carrier grid. Specify no windowing for the slot-based processing.

```
[wave,winfo] = nrOFDMModulate(carrier,carrierGrid,'Windowing',0);
```

## Fading Channel

Transmit the generated waveform over a TDL fading channel with delay profile A and a delay spread of 30 ns.

```
channel = nrTDLChannel;
channel.DelayProfile = 'TDL-A';
channel.DelaySpread = 30e-9;
channel.NumTransmitAntennas = 1;
channel.NumReceiveAntennas = 1;
channel.SampleRate = winfo.SampleRate;

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

```
txWave = [wave; zeros(maxChDelay, size(wave,2))];
rxWave = channel(txWave);
```

### Noise Addition

Add white Gaussian noise with the specified level to the received signal, taking into account the coding rate, QPSK modulation, and sampling rate.

```
EbNo = 6; % in dB
bps = 2; % bits per symbol, 2 for QPSK
EsNo = EbNo + 10*log10(bps);
snrdB = EsNo + 10*log10(K/E);
noiseVar = 10.^(-snrdB/10); % assumes unit signal power
N0 = sqrt(noiseVar)/sqrt(2*winfo.Nfft);
noise = N0 * complex(randn(size(rxWave)),randn(size(rxWave)));
rxWaveN = rxWave + noise;
```

### Blind PDCCH and DCI Decoding

The UE does not have information about the detailed control channel structure. Therefore, the UE decodes the received PDCCH symbols blindly by monitoring a set of PDCCH candidates for each slot using the UE's RNTI to identify the right candidate (or instance).

Monitoring a candidate implies attempting to decode a set of resource elements corresponding to the candidate by checking if the returned checksum is zero for the known RNTI (UE). Use the `nrPDCCHSpace` function to determine all candidates specified by the search space set in terms of the PDCCH resource element indices, corresponding DM-RS symbols and indices.

For each candidate, the front-end recovery includes

- timing offset estimation based on the DM-RS symbols using the function `nrTimingEstimate`,
- OFDM demodulation using the function `nrOFDMDemodulate`,
- channel estimation based on the DM-RS symbols using the function `nrChannelEstimate`, and
- MMSE equalization using the function `nrEqualizeMMSE`

to yield the equalized candidate PDCCH symbols.

The equalized symbols per candidate are demodulated with known user-specific parameters and channel noise variance using the `nrPDCCHDecode` function.

For an instance of the received PDCCH codeword, the `nrDCIDeCode` function includes the stages of rate recovery, polar decoding, and CRC decoding. If the output mask value is zero, the PDCCH is decoded successfully and the UE can process the DCI message.

In this example, the receiver assumes knowledge of the DCI format and the DCI payload size  $K$ . In practice, even these would be searched for in an outer loop over all supported formats with respective bit lengths per format.

```
listLen = 8; % polar decoding list length
% Get all possible candidates
[allInd,allDMRS,allDMRSInd] = nrPDCCHSpace(carrier,pcch);
% Loop over all supported aggregation levels
decoded = false;
```

```

for aIdx = 1:5

% Loop over all candidates at each aggregation level
for cIdx = 1:pdccch.SearchSpace.NumCandidates(aIdx)

% Get candidate
cSymIdx = allInd{aIdx}(:,cIdx);
cDMRS = allDMRS{aIdx}(:,cIdx);
cDMRSInd = allDMRSInd{aIdx}(:,cIdx);

% Timing estimate
offset = nrTimingEstimate(carrier,rxWaveN,cDMRSInd,cDMRS);
if offset > maxChDelay
    offset = 0;
end
rxWaveS = rxWaveN(1+offset:end,:);

% OFDM demodulate the carrier
rxCarrGrid = nrOFDMDemodulate(carrier,rxWaveS);

% Channel estimate
[hest,nVar] = nrChannelEstimate(carrier,rxCarrGrid,cDMRSInd,cDMRS);
[rxSym,pcchHest] = nrExtractResources(cSymIdx,rxCarrGrid,hest);

% Equalization
[pcchEq,csi] = nrEqualizeMMSE(rxSym,pcchHest,nVar);

% Demodulate
rxCW = nrPDCCHDecode(pcchEq,nID,pcch.RNTI,nVar);

% Apply CSI
csiRep = repmat(csi.',2,1);
scalRxCW = rxCW.*csiRep(:);

% Decode
[decDCIBits,errFlag] = nrDCIDecode(scalRxCW,K,listLen,pcch.RNTI);

if isequal(errFlag,0)
    disp(['Decoded candidate #' num2str(cIdx) ...
         ' at aggregation level ' num2str(2^(aIdx-1)) ...
         ' in slot'])
    decoded = true;
    if isequal(decDCIBits,dciBits)
        disp(' Recovered DCI bits with no errors');
    else
        disp(' Recovered DCI bits with errors');
    end
    break;
end
end
% Dont loop over other aggregation levels if RNTI matched
if decoded
    break;
end
end

Decoded candidate #1 at aggregation level 4 in slot
Recovered DCI bits with no errors

```

For the chosen system parameters, the decoded information matches the transmitted information bits.

The example searched over all candidates within a single search space set as specified by the `ss` configuration parameter. Search over multiple search space sets would require another external loop over all the sets defined.

### **Selected References**

- 1 3GPP TS 38.211. "NR; Physical channels and modulation (Release 15)." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.
- 2 3GPP TS 38.212. "NR; Multiplexing and channel coding (Release 15)." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.
- 3 3GPP TS 38.213. "NR; Physical layer procedures for control (Release 15)." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.

### **See Also**

#### **Functions**

`nrDCIDecode` | `nrDCIEncode` | `nrPDCCH` | `nrPDCCHDecode` | `nrPDCCHResources` | `nrPDCCHSpace`

## NR Channel Estimation Using CSI-RS

This example shows how to generate channel state information reference signal (CSI-RS) symbols and indices for a given carrier and CSI-RS resource configuration, as defined in TS 38.211 Section 7.4.1.5. The example shows how to map the generated symbols to the carrier resource grid, performs channel estimation at the receiver side, and compares the estimated channel against the actual channel.

### Introduction

CSI-RS is a downlink-specific (DL) reference signal. The NR standard defines zero-power (ZP) and non-zero-power (NZP) CSI-RSs.

The user equipment (UE) processes utilize NZP-CSI-RSs:

- L1-Reference signal received power (RSRP) measurements for mobility and beam management
- DL CSI acquisition
- Interference measurement
- Time and frequency tracking

ZP-CSI-RS is used for DL CSI acquisition and interference measurement. It also masks certain resource elements (REs) to make them unavailable for PDSCH transmission. As the name ZP indicates, nothing is transmitted in those REs.

This example shows how to use CSI-RS to perform channel estimation which forms the basis of CSI acquisition.

### Initialize Configuration Objects

Create a carrier configuration object representing a 5 MHz carrier with subcarrier spacing of 15 kHz.

```
carrier = nrCarrierConfig;
carrier.NSizeGrid = 25;
carrier.SubcarrierSpacing = 15;
carrier.NSlot = 1;
carrier.NFrame = 0

carrier =
  nrCarrierConfig with properties:

      NCellID: 1
  SubcarrierSpacing: 15
      CyclicPrefix: 'normal'
      NSizeGrid: 25
      NStartGrid: 0
      NSlot: 1
      NFrame: 0

  Read-only properties:
      SymbolsPerSlot: 14
      SlotsPerSubframe: 1
      SlotsPerFrame: 10
```

Create a CSI-RS configuration object representing two CSI-RS resources, NZP with row number 3 and ZP with row number 5.

```
csirs = nrCSIRSConfig;
csirs.CSIRSType = {'nzp', 'zp'};
csirs.CSIRSPeriod = {[5 1], [5 1]};
csirs.Density = {'one', 'one'};
csirs.RowNumber = [3 5];
csirs.SymbolLocations = {1,6};
csirs.SubcarrierLocations = {6,4};
csirs.NumRB = 25

csirs =
  nrCSIRSConfig with properties:
      CSIRSType: {'nzp'  'zp'}
  CSIRSPeriod: {[5 1]  [5 1]}
    RowNumber: [3 5]
      Density: {'one'  'one'}
 SymbolLocations: {[1]  [6]}
SubcarrierLocations: {[6]  [4]}
        NumRB: 25
      RBOffset: 0
         NID: 0

  Read-only properties:
    NumCSIRSPorts: [2 4]
      CDMType: {'FD-CDM2'  'FD-CDM2'}
```

Consider the power scaling of CSI-RS in dB.

```
powerCSIRS = 0;
disp(['CSI-RS power scaling: ' num2str(powerCSIRS) ' dB']);
```

```
CSI-RS power scaling: 0 dB
```

### Generate CSI-RS Symbols and Indices

Generate CSI-RS symbols for the specified carrier and CSI-RS configuration parameters. Apply power scaling.

```
sym = nrCSIRS(carrier, csirs);
csirsSym = sym*db2mag(powerCSIRS);
```

The variable `csirsSym` is a column vector containing CSI-RS symbols.

Generate CSI-RS indices for the specified carrier and CSI-RS configuration parameters.

```
csirsInd = nrCSIRSIndices(carrier, csirs);
```

The variable `csirsInd` is also a column vector of same size as that of `csirsSym`.

When you configure both ZP and NZP resources, the generation of ZP signals takes priority over the generation of NZP signals.

### Initialize Carrier Grid

Initialize the carrier resource grid for one slot.

```
ports = max(csirs.NumCSIRSPorts); % Number of antenna ports
txGrid = nrResourceGrid(carrier, ports);
```



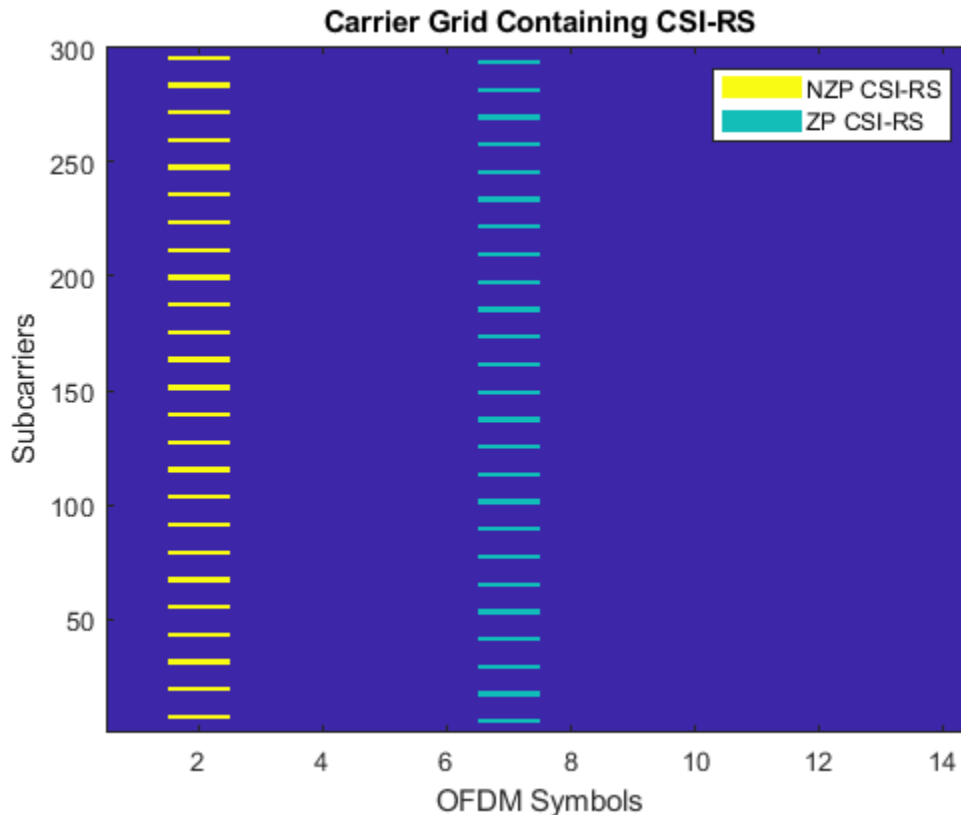
### Map CSI-RS Symbols Onto Carrier Grid

Perform resource element mapping.

```
txGrid(csirsInd) = csirsSym;
```

Plot the locations of the CSI-RS (both ZP and NZP) in the grid.

```
plotGrid(size(txGrid),csirsInd,csirsSym);
```



### Perform OFDM Modulation

Perform OFDM modulation to generate the time-domain waveform.

```
[txWaveform,ofdmInfo] = nrOFDMModulate(carrier,txGrid);
```

### Pass Time-Domain Waveform Through Channel and Add AWGN Noise

Configure the number of receiving antennas.

```
R = 4;
```

Configure the channel.

```
channel = nrTDLChannel;  
channel.NumTransmitAntennas = ports;  
channel.NumReceiveAntennas = R;  
channel.DelayProfile = 'TDL-C';
```

```

channel.MaximumDopplerShift = 10;
channel.DelaySpread = 1e-8

channel =
  nrTDLChannel with properties:

        DelayProfile: 'TDL-C'
        DelaySpread: 1.0000e-08
  MaximumDopplerShift: 10
        SampleRate: 30720000
        MIMOCorrelation: 'Low'
        Polarization: 'Co-Polar'
  TransmissionDirection: 'Downlink'
  NumTransmitAntennas: 4
  NumReceiveAntennas: 4
  NormalizePathGains: true
        InitialTime: 0
        NumSinusoids: 48
        RandomStream: 'mt19937ar with seed'
        Seed: 73
  NormalizeChannelOutputs: true

```

Based on the configured channel, append zeros to the transmitted waveform to account for the channel delay.

```

chInfo = info(channel);
maxChDelay = ceil(max(chInfo.PathDelays*channel.SampleRate)) + chInfo.ChannelFilterDelay;
txWaveform = [txWaveform; zeros(maxChDelay,size(txWaveform,2))];

```

Pass waveform through channel.

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

To produce the actual propagation channel  $H_{\text{actual}}$ , perform perfect channel estimation.

```

pathFilters = getPathFilters(channel);
H_actual = nrPerfectChannelEstimate(carrier,pathGains,pathFilters);

```

Add AWGN noise to the waveform.

```

SNRdB = 50;           % in dB
SNR = 10^(SNRdB/20); % Linear value
N0 = 1/(sqrt(2.0*R*double(ofdmInfo.Nfft))*SNR); % Noise variance
rng(0);
noise = N0*complex(randn(size(rxWaveform)),randn(size(rxWaveform)));
rxWaveform = rxWaveform + noise

```

```
rxWaveform = 7690x4 complex
```

```

 0.0000 - 0.0001i  -0.0001 + 0.0000i  -0.0001 - 0.0000i  -0.0001 + 0.0000i
 0.0001 - 0.0001i  -0.0000 + 0.0000i   0.0001 - 0.0000i  -0.0000 - 0.0000i
-0.0001 + 0.0000i   0.0001 + 0.0000i  -0.0001 - 0.0000i  -0.0001 + 0.0000i
 0.0000 - 0.0000i   0.0000 - 0.0000i  -0.0001 + 0.0001i  -0.0000 - 0.0000i
 0.0000 + 0.0001i   0.0001 - 0.0000i   0.0000 - 0.0000i   0.0001 - 0.0000i
-0.0001 + 0.0000i  -0.0000 + 0.0000i  -0.0000 + 0.0000i   0.0001 + 0.0000i
-0.0000 - 0.0001i  -0.0000 - 0.0001i  -0.0001 - 0.0000i  -0.0000 - 0.0000i
 0.0000 + 0.0001i  -0.0001 - 0.0000i  -0.0000 + 0.0000i  -0.0000 - 0.0000i
 0.0002 - 0.0001i   0.0001 + 0.0001i   0.0001 + 0.0000i   0.0001 + 0.0000i

```

```
0.0001 + 0.0000i  0.0000 + 0.0000i  0.0000 - 0.0001i  0.0000 - 0.0001i
:
```

Perform timing synchronization using NZP-CSI-RS. To estimate timing offset, use `nrTimingEstimate` and consider the NZP-CSI-RS as a reference.

```
% Disable ZP-CSI-RS resource, not going to be used for timing and channel
% estimation
csirs.CSIRSPeriod = {[5 1], 'off'};
% Generate reference symbols and apply power scaling
refSym = db2mag(powerCSIRS)*nrCSIRS(carrier,csirs);
% Generate reference indices
refInd = nrCSIRSIndices(carrier,csirs);
offset = nrTimingEstimate(carrier,rxWaveform,refInd,refSym)

offset = 7

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

OFDM demodulate the received time-domain waveform.

```
rxGrid = nrOFDMDemodulate(carrier,rxWaveform); % Of size K-by-L-by-R
```

### Compare Estimated Channel Against Actual Channel

Perform practical channel estimation using NZP-CSI-RS. Make sure the CSI-RS symbols `csirsSym` are of the same CDM type.

```
cdmLen = [2 1]; % Corresponds to CDMType = 'FD-CDM2'
[H_est,nVar] = nrChannelEstimate(carrier,rxGrid,refInd,refSym,'CDMLengths',cdmLen);
estSNR = -10*log10(nVar);
disp(['estimated SNR = ' num2str(estSNR) ' dB'])
```

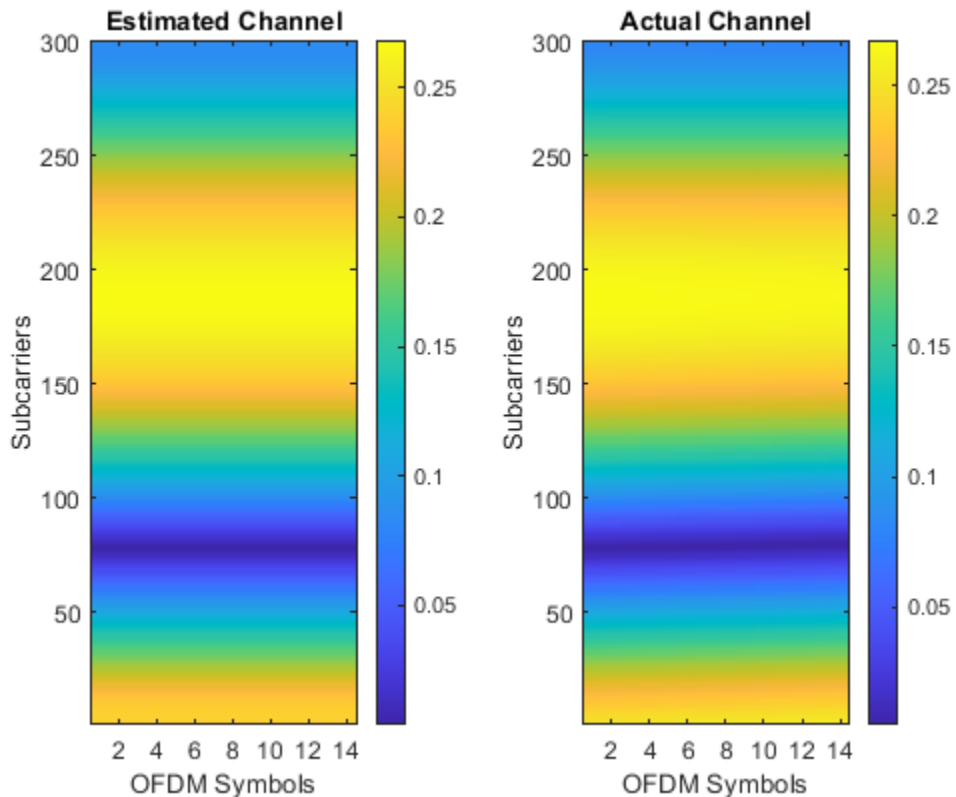
```
estimated SNR = 27.4577 dB
```

Plot the estimated channel and actual channel between first transmitting antenna and first receiving antenna.

```
figure;

% Plot the estimated channel
subplot(1,2,1)
imagesc(abs(H_est(:,:,1,1)));
colorbar;
title('Estimated Channel')
axis xy;
xlabel('OFDM Symbols');
ylabel('Subcarriers');

% Plot the actual channel
subplot(1,2,2)
imagesc(abs(H_actual(:,:,1,1)));
colorbar;
title('Actual Channel')
axis xy;
xlabel('OFDM Symbols');
ylabel('Subcarriers');
```



Calculate the channel estimation error.

```
H_err = (H_est - H_actual(:,:,1:size(H_est,4)));
[minErr,maxErr] = bounds(abs(H_err),'all');
disp(['Absolute value of the channel estimation error is in the range of [' num2str(minErr) ', ' ' num2str(maxErr) '].'])
```

Absolute value of the channel estimation error is in the range of [4.9184e-06, 0.035047]

### Local Functions

```
function plotGrid(gridSize,csirsInd,csirsSym)
% plotGrid(GRIDSIZE,CSIRSIND,CSIRSSYM) plots the carrier grid of size GRIDSIZE
% by populating the grid with CSI-RS symbols using CSIRSIND and CSIRSSYM.

figure()
cmap = colormap(gcf);
chpval = {20,2};
chpscale = 0.25*length(cmap); % Scaling factor

tempSym = csirsSym;
tempSym(tempSym ~= 0) = chpval{1}; % Replacing non-zero-power symbols
tempSym(tempSym == 0) = chpval{2}; % Replacing zero-power symbols
tempGrid = complex(zeros(gridSize));
tempGrid(csirsInd) = tempSym;

image(chpscale*tempGrid(:,:,1)); % Multiplied with scaling factor for better visualization
axis xy;
names = {'N ZP CSI-RS','Z P CSI-RS'};
```

```
clevels = chpscale*[chpval{:}];
N = length(clevels);
L = line(ones(N),ones(N),'LineWidth',8); % Generate lines
% Index the color map and associate the selected colors with the lines
set(L,{'color'},mat2cell(cmap( min(1+clevels,length(cmap) ),:),ones(1,N),3)); % Set the color
% Create legend
legend(names{:});

title('Carrier Grid Containing CSI-RS')
xlabel('OFDM Symbols');
ylabel('Subcarriers');
end
```

## References

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

## See Also

### Functions

nrCSIRS | nrCSIRSIndices

### Objects

nrCSIRSConfig | nrCarrierConfig

## 5G NR CSI-RS Measurements

This example shows the measurement procedure of CSI-RSRP, CSI-RSSI, and CSI-RSRQ for the test environment, as described in TS 38.133 Annex A.4.6.3.3, by using the channel state information reference signal from 5G Toolbox™.

### Introduction

In NR 5G, the three types of CSI-RS based reference signal measurements, as defined in TS 38.215 Sections 5.1.2 and 5.1.4, includes:

- **CSI-RSRP (CSI reference signal received power):** CSI-RSRP is defined as the linear average over the power contributions of the resource elements of the antenna ports, which carry CSI-RS configured for RSRP measurements. This measurement is performed across  $N$  number of resource blocks (measurement bandwidth). For this measurement, CSI-RS transmitted on antenna port(s) 3000 or 3000 and 3001 is used.
- **CSI-RSSI (CSI received signal strength indicator):** CSI-RSSI is defined as the linear average of the total received power observed only in the OFDM symbols, in which CSI-RS is present. This measurement is also performed across  $N$  number of resource blocks (measurement bandwidth). CSI-RSSI includes the power from sources, such as co-channel serving and non-serving cells, adjacent channel interference, and thermal noise. For this measurement, CSI-RS transmitted on antenna port 3000 is used.
- **CSI-RSRQ (CSI reference signal received quality):** CSI-RSRQ is defined as,  $N * \frac{\text{CSI\_RSRP}}{\text{CSI\_RSSI}}$ .

The purposes of these measurements include:

- Cell selection and reselection
- Mobility and handover management
- Beam management (beam adjustment and beam recovery)

This example configures only the CSI-RS from the test environment.

### Initialize Configuration Objects

#### Carrier configuration

Create carrier configuration object occupying 10 MHz bandwidth with 15 kHz subcarrier spacing as per configuration 1 in TS 38.133 Table A.4.6.3.3.1-1.

```
carrier = nrCarrierConfig;
carrier.NSlot = 1;
carrier.NSizeGrid = 52;
```

#### CSI-RS configuration

As per the test environment TS 38.133 Table A.4.6.3.3.2-1, user equipment (UE) is configured with one CSI-RS resource set (CSI-RS 1.2 FDD), consisting of 2 CSI-RS resources.

```
csirs = nrCSIRSConfig;
% CSI-RS resource
csirs.CSIRSType      = {'#0', '#1'};
csirs.CSIRSPeriod    = {[10 1], [10 1]};
csirs.RowNumber      = [1      1]; % Single port (3000) CSI-RS resources
```

```

csirs.Density           = {'three', 'three'};
csirs.SymbolLocations  = {6,      10};
csirs.SubcarrierLocations = {0,      0};
csirs.NumRB            = [52,     52]; % Measurement bandwidth in terms of resource blocks

```

### Generate CSI-RS Symbols and Indices

Generate CSI-RS symbols and indices for the specified carrier and CSI-RS configuration parameters with the output resource format as 'cell'. This output resource format provides a way to identify the outputs uniquely for each CSI-RS resource in a resource set. You can also apply different power levels to each CSI-RS resource.

```

ind = nrCSIRSIndices(carrier,csirs,'OutputResourceFormat','cell');
sym = nrCSIRS(carrier,csirs,'OutputResourceFormat','cell');

```

### Signal and Noise Power Setup

Set up the signal and noise powers as described in TS 38.133 Table A.4.6.3.3.2-2. As per note 2 in TS 38.133 Table A.4.6.3.3.2-2, interference from other cells and noise from other sources is modeled as additive white Gaussian noise (AWGN) of appropriate power  $N_{oc}$ .

```

SINRdB0 = 0; % For CSI-RS #0
SINRdB1 = 3; % For CSI-RS #1
NocdBm = -94.65;
NocdB = NocdBm - 30;
Noc = 10^(NocdB/10);

```

Calculate the power scaling of CSI-RS resources by using SINR values.

```

% Power scaling of CSI-RS resource #0
SINR0 = 10^(SINRdB0/10); % linear Es/Noc
Es0 = SINR0*Noc;

% Power scaling of CSI-RS resource #1
SINR1 = 10^(SINRdB1/10); % linear Es/Noc
Es1 = SINR1*Noc;

```

### Initialize the Carrier Resource Grid and Map CSI-RS Symbols to the Grid

Initialize the carrier resource grid for one slot.

```

ports = max(csirs.NumCSIRSPorts); % Number of antenna ports
txGrid = nrResourceGrid(carrier,ports);

```

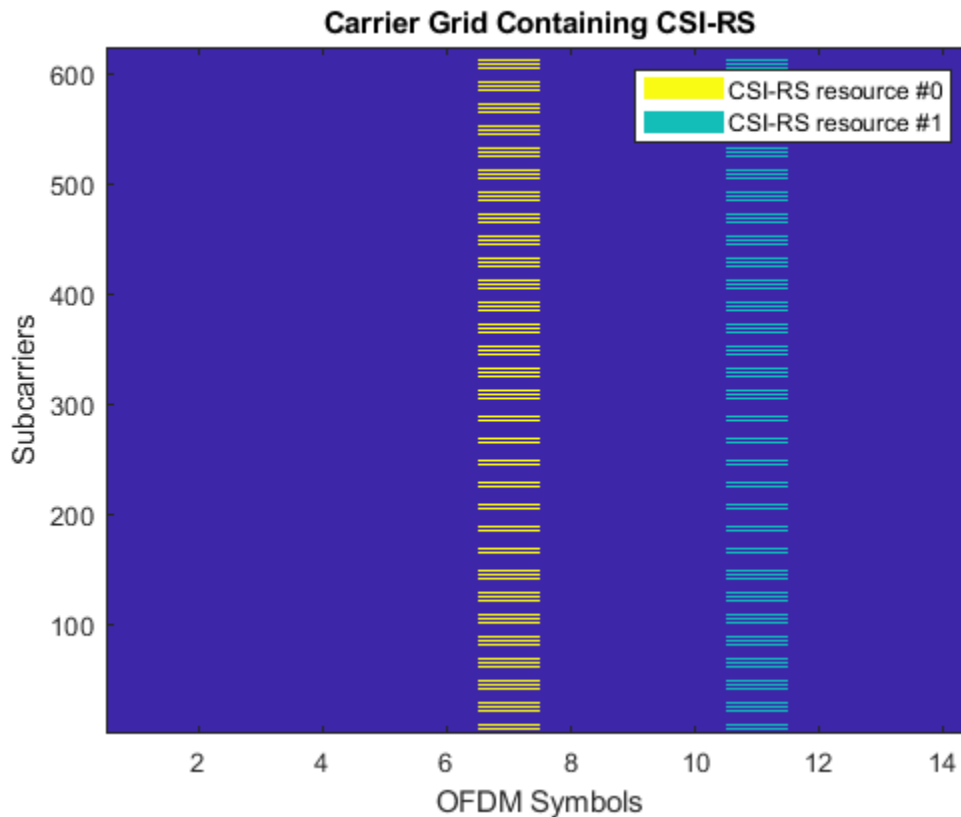
Apply the power scaling values to the CSI-RS resources and map them onto the grid.

```

txGrid(ind{1}) = sqrt(Es0)*sym{1};
txGrid(ind{2}) = sqrt(Es1)*sym{2};

% Plot the carrier grid for two CSI-RS resources
plotGrid(size(txGrid),ind)

```



### Perform OFDM modulation

Perform OFDM modulation to generate the time-domain waveform.

```
[txWaveform,ofdmInfo] = nrOFDMModulate(carrier,txGrid);
```

### Add AWGN to the Transmitted Waveform and Perform OFDM Demodulation

Consider the propagation condition as AWGN, as specified in TS 38.133 Table A.4.6.3.3.2-1.

```
% Generate the noise
rng('default'); % Set RNG state for repeatability
N0 = sqrt(Noc/(2*double(ofdmInfo.Nfft)));
noise = N0*complex(randn(size(txWaveform)),randn(size(txWaveform)));
```

```
% Add AWGN to the transmitted waveform
rxWaveform = txWaveform + noise;
```

Perform OFDM demodulation on the received time-domain waveform to get the received resource element array.

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

### Perform CSI-RSRP, CSI-RSSI, and CSI-RSRQ Measurements

Finally, perform the CSI-RSRP, CSI-RSSI, and CSI-RSRQ measurements on the CSI-RS resources present in the received grid, by using the helper file *hCSIRSMeasurements*.

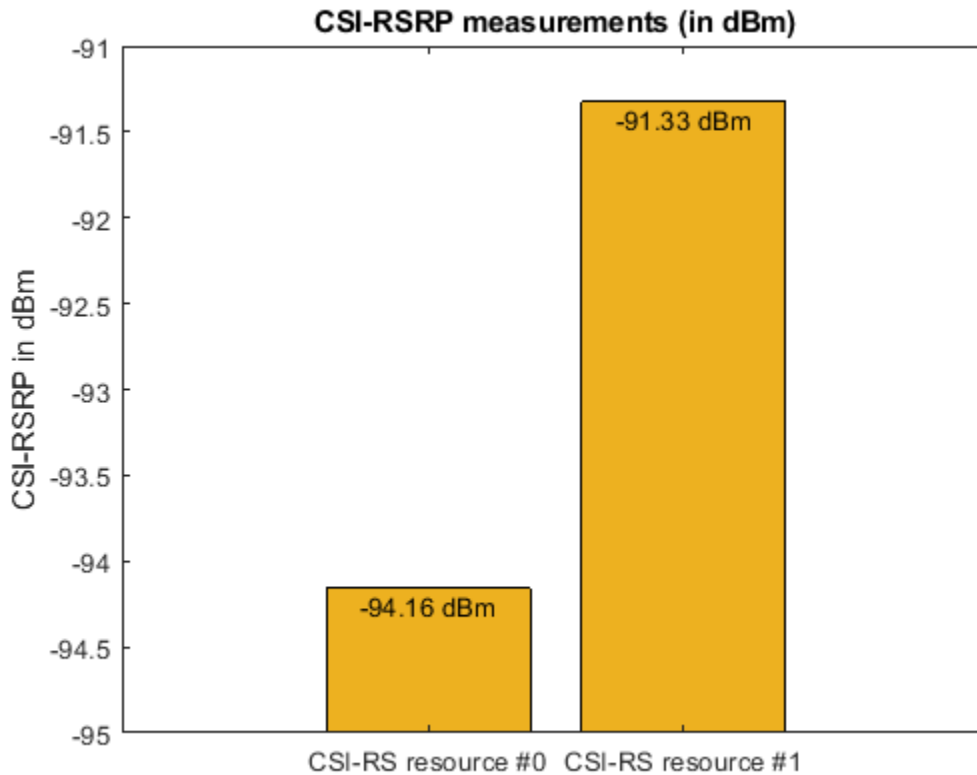


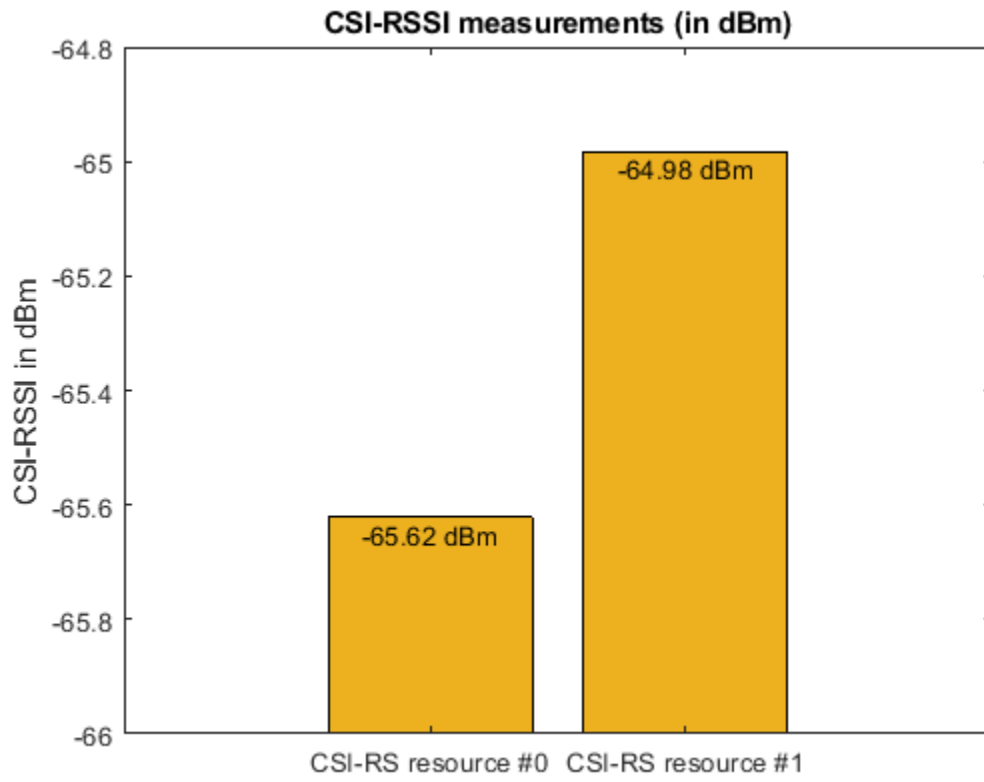
```
meas = hCSIRSMeasurements(carrier,csirs,rxGrid)
```

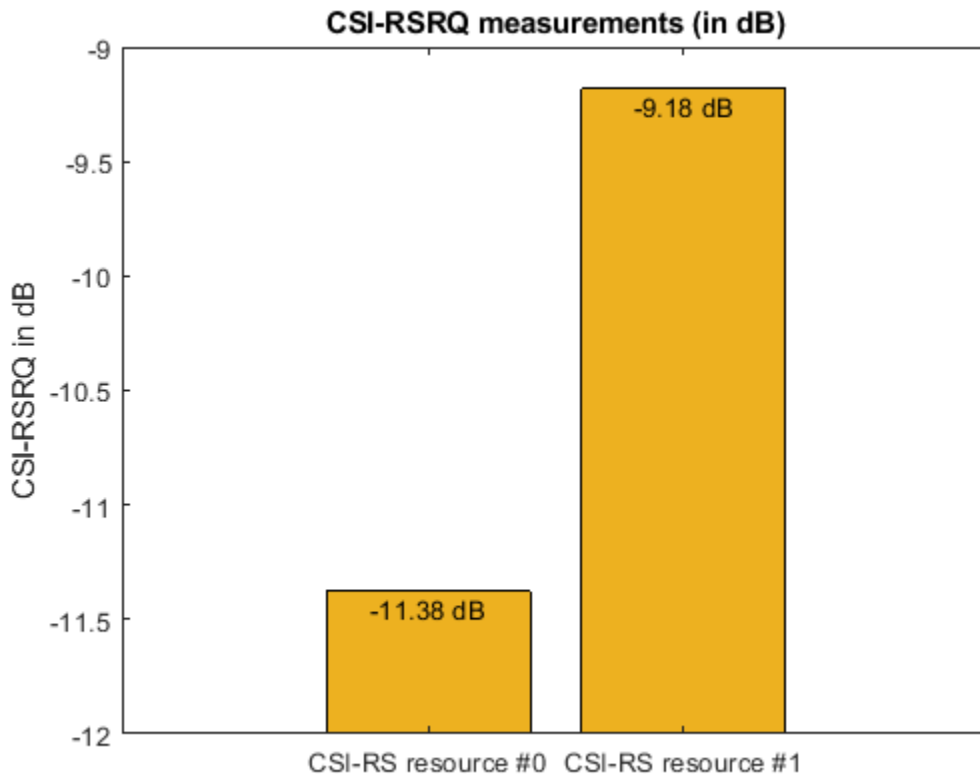
```
meas = struct with fields:
```

```
RSRPPerAntennaPerResource: [3.8372e-13 7.3692e-13]  
RSSIPerAntennaPerResource: [2.7403e-10 3.1752e-10]  
RSRQPerAntennaPerResource: [0.0728 0.1207]  
RSRP: [3.8372e-13 7.3692e-13]  
RSSI: [2.7403e-10 3.1752e-10]  
RSRQ: [0.0728 0.1207]  
RSRPaBm: [-94.1599 -91.3258]  
RSSIdBm: [-65.6220 -64.9823]  
RSRQdB: [-11.3779 -9.1834]
```

```
% Plot RSRPaBm, RSSIdBm and RSRQdB measurements for all CSI-RS resources  
hPlotCSIRSMeasurements(meas)
```







You can compare the measured CSI-RSRP values of two CSI-RS resources represented by the output field RSRPdBm to the standard specified values given in TS 38.133 Table A.4.6.3.3.2-2.

### Local Functions

```
function plotGrid(gridSize,csirsInd)
% plotGrid(GRIDSIZE,CSIRSIND) plots the carrier grid of size GRIDSIZE
% by populating the grid with CSI-RS symbols of multiple resources
% indicated by a cell array of CSI-RS indices CSIRSIND.

figure()
cmap = colormap(gcf);

% Considering the following values for two CSI-RS resources and they need
% to be updated based on the number of CSI-RS resources
names = {'CSI-RS resource #0','CSI-RS resource #1'};
chpval = {20,2};
chpscale = 0.25*length(cmap); % Scaling factor
tempGrid = zeros(gridSize);
tempGrid(csirsInd{1}) = chpval{1};
tempGrid(csirsInd{2}) = chpval{2};

image(chpscale*tempGrid(:,:,1)); % Multiplied with scaling factor for better visualization
axis xy;
clevels = chpscale*[chpval{:}];
N = length(clevels);
L = line(ones(N),ones(N),'LineWidth',8); % Generate lines
```

```
% Index the color map and associate the selected colors with the lines
set(L,{'color'},mat2cell(cmap( min(1+clevels,length(cmap) ),:),ones(1,N),3)); % Set the color
% Create legend
legend(names{:});

title('Carrier Grid Containing CSI-RS')
xlabel('OFDM Symbols');
ylabel('Subcarriers');
end
```

## References

- [1] 3GPP TS 38.133. “NR; Requirements for support of radio resource management.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
- [2] 3GPP TS 38.215. “NR; Physical layer measurements.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

## See Also

### Functions

nrCSIRS | nrCSIRSIndices

### Objects

nrCSIRSConfig | nrCarrierConfig

## Related Examples

- “NR Channel Estimation Using CSI-RS” on page 1-81
- “5G NR CQI Reporting” on page 4-8

## NR SSB Beam Sweeping

This example shows how to employ beam sweeping at both the transmitter (gNB) and receiver (UE) ends of a 5G NR system. Using synchronization signal blocks (SSB), this example illustrates some of the beam management procedures used during initial access. To accomplish beam sweeping, the example uses several components from Phased Array System Toolbox™.

### Introduction

The support of millimeter wave (mmWave) frequencies requires directional links, which led to the specification of beam management procedures for initial access in NR. Beam management is a set of Layer 1 (physical) and Layer 2 (medium access control) procedures to acquire and maintain a set of beam pair links (a beam used at gNB paired with a beam used at UE). Beam management procedures are applied for both downlink and uplink transmission and reception [ 1 ], [ 2 ]. These procedures include:

- Beam sweeping
- Beam measurement
- Beam determination
- Beam reporting
- Beam recovery

This example focuses on initial access procedures for idle users when a connection is established between the user equipment (UE) and access network node (gNB). At the physical layer, using synchronization signal blocks (SSB) transmitted as a burst in the downlink direction (gNB to UE), the example highlights both transmit/receive point (TRP) beam sweeping and UE beam sweeping to establish a beam pair link. Amongst the multiple beam management procedures, TR 38.802 defines this dual-end sweep as procedure P-1 [ 1 ].

Once connected, the same beam pair link can be used for subsequent transmissions. If necessary, the beams are further refined using CSI-RS (for downlink) and SRS (for uplink). In case of beam failure, these pair links can be reestablished. For an example of beam pair refinement, see “NR Downlink Transmit-End Beam Refinement Using CSI-RS” on page 1-110.

This example generates an NR synchronization signal burst, beamforms each of the SSBs within the burst to sweep over both azimuth and elevation directions, transmits this beamformed signal over a spatial scattering channel, and processes this received signal over the multiple receive-end beams. The example measures the reference signal received power (RSRP) for each of the transmit-receive beam pairs (in a dual loop) and determines the beam pair link with the maximum RSRP. This beam pair link thus signifies the best beam-pair at transmit and receive ends for the simulated spatial scenario. This figure shows the main processing steps with the beam management ones highlighted in color.



```
rng(211); % Set RNG state for repeatability
```

### Simulation Parameters

Define system parameters for the example. Modify these parameters to explore their impact on the system.

```

prm.NCellID = 1; % Cell ID
prm.FreqRange = 'FR1'; % Frequency range: 'FR1' or 'FR2'
prm.CenterFreq = 3.5e9; % Hz
prm.SSBBlockPattern = 'Case B'; % Case A/B/C/D/E
prm.SSBTransmitted = [ones(1,8) zeros(1,0)]; % 4/8 or 64 in length

prm.TxArraySize = [8 8]; % Transmit array size, [rows cols]
prm.TxAZlim = [-60 60]; % Transmit azimuthal sweep limits
prm.TxELlim = [-90 0]; % Transmit elevation sweep limits

prm.RxArraySize = [2 2]; % Receive array size, [rows cols]
prm.RxAZlim = [-180 180]; % Receive azimuthal sweep limits
prm.RxELlim = [0 90]; % Receive elevation sweep limits

prm.ElevationSweep = false; % Enable/disable elevation sweep
prm.SNRdB = 30; % SNR, dB
prm.RSRPMode = 'SSSwDMRS'; % {'SSSwDMRS', 'SSSonly'}

```

The example uses these parameters:

- Cell ID for a single-cell scenario with a single BS and UE
- Frequency range, as a string to designate FR1 or FR2 operation
- Center frequency, in Hz, dependent on the frequency range
- Synchronization signal block pattern as one of Case A/B/C for FR1 and Case D/E for FR2. This also selects the subcarrier spacing.
- Transmitted SSBs in the pattern, as a binary vector of length 4 or 8 for FR1 and length 64 for FR2. The number of SSBs transmitted sets the number of beams at both transmit and receive ends.
- Transmit array size, as a two-element row vector specifying the number of antenna elements in the rows and columns of the transmit array, respectively. A uniform rectangular array (URA) is used when both values are greater than one.
- Transmit azimuthal sweep limits in degrees to specify the starting and ending azimuth angles for the sweep
- Transmit elevation sweep limits in degrees to specify the starting and ending elevation angles for the sweep
- Receive array size, as a two-element row vector specifying the number of antenna elements in the rows and columns of the receive array, respectively. A uniform rectangular array (URA) is used when both values are greater than one.
- Receive azimuthal sweep limits in degrees to specify the starting and ending azimuth angles for the sweep
- Receive elevation sweep limits in degrees to specify the starting and ending elevation angles for the sweep
- Enable or disable elevation sweep for both transmit and receive ends. Enable elevation sweep for FR2 and/or URAs
- Signal-to-noise ratio in dB
- Measurement mode for SSB to specify the use of only secondary synchronization signals ('SSSonly') or use of PBCH DM-RS along with secondary synchronization signals ('SSSwDMRS')

```
prm = validateParams(prm);
```

## Synchronization Signal Burst Configuration

Set up the synchronization signal burst parameters by using the specified system parameters. For initial access, set the SSB periodicity to 20 ms.

```
txBurst.BlockPattern = prm.SSBBlockPattern;
txBurst.SSBTransmitted = prm.SSBTransmitted;
txBurst.NCellID = prm.NCellID;
txBurst.SSBPeriodicity = 20;
txBurst.NFrame = 0;
txBurst.Windowing = 0;
txBurst.DisplayBurst = true;

% Assume same subcarrier spacing for carrier as the burst
carrier = nrCarrierConfig('NCellID',prm.NCellID,'NFrame',txBurst.NFrame);
carrier.SubcarrierSpacing = prm.SCS;
carrierInfo = nrOFDMInfo(carrier);
txBurst.SampleRate = carrierInfo.SampleRate;
```

Refer to the tutorial “Synchronization Signal Blocks and Bursts” for more details on synchronization signal blocks and bursts.

## Channel Configuration

Configure a spatial scattering MIMO channel channel. Specify the locations for the BS and UE as  $[x, y, z]$  coordinates in a cartesian system. Depending on the array sizes specified, employ either uniform linear arrays (ULA) or uniform rectangular arrays (URA). Use isotropic antenna elements for the arrays.

```
c = physconst('LightSpeed'); % Propagation speed
lambda = c/prm.CenterFreq; % Wavelength

prm.posTx = [0;0;0]; % Transmit array position, [x;y;z], meters
prm.posRx = [100;50;0]; % Receive array position, [x;y;z], meters

toRxRange = rangeangle(prm.posTx,prm.posRx);
spLoss = fspl(toRxRange,lambda); % Free space path loss

% Transmit array
if prm.IsTxURA
    % Uniform rectangular array
    arrayTx = phased.URA(prm.TxArraySize,0.5*lambda, ...
        'Element',phased.IsotropicAntennaElement('BackBaffled',true));
else
    % Uniform linear array
    arrayTx = phased.ULA(prm.NumTx, ...
        'ElementSpacing',0.5*lambda, ...
        'Element',phased.IsotropicAntennaElement('BackBaffled',true));
end

% Receive array
if prm.IsRxURA
    % Uniform rectangular array
    arrayRx = phased.URA(prm.RxArraySize,0.5*lambda, ...
        'Element',phased.IsotropicAntennaElement);
else
    % Uniform linear array
    arrayRx = phased.ULA(prm.NumRx, ...
```

```

        'ElementSpacing',0.5*lambda, ...
        'Element',phased.IsotropicAntennaElement);
end

% Scatterer locations
prm.FixedScatMode = true;
if prm.FixedScatMode
    % Fixed single scatterer location
    prm.ScatPos = [50; 80; 0];
else
    % Generate scatterers at random positions
    Nscat = 10; % Number of scatterers
    azRange = -180:180;
    elRange = -90:90;
    randAzOrder = randperm(length(azRange));
    randElOrder = randperm(length(elRange));
    azAngInSph = azRange(randAzOrder(1:Nscat));
    elAngInSph = elRange(randElOrder(1:Nscat));
    r = 20; % radius
    [x,y,z] = sph2cart(deg2rad(azAngInSph),deg2rad(elAngInSph),r);
    prm.ScatPos = [x;y;z] + (prm.posTx + prm.posRx)/2;
end

% Configure channel
channel = phased.ScatteringMIMOChannel;
channel.PropagationSpeed = c;
channel.CarrierFrequency = prm.CenterFreq;
channel.SampleRate = txBurst.SampleRate;
channel.SimulateDirectPath = false;
channel.ChannelResponseOutputPort = true;
channel.Polarization = 'None';
channel.TransmitArray = arrayTx;
channel.TransmitArrayPosition = prm.posTx;
channel.ReceiveArray = arrayRx;
channel.ReceiveArrayPosition = prm.posRx;
channel.ScattererSpecificationSource = 'Property';
channel.ScattererPosition = prm.ScatPos;
channel.ScattererCoefficient = ones(1,size(prm.ScatPos,2));

% Get maximum channel delay
[~,~,tau] = channel(complex(randn(txBurst.SampleRate*1e-3,prm.NumTx), ...
    randn(txBurst.SampleRate*1e-3,prm.NumTx)));
maxChDelay = ceil(max(tau)*txBurst.SampleRate);

```

### Burst Generation

Create the SS burst waveform [ 3 ] by calling the hSSBurst helper function. The generated waveform is not yet beamformed.

```

% Create and display burst information
txBurstInfo = hSSBurstInfo(txBurst);
disp(txBurstInfo);

% Generate burst waveform and grid
[burstWaveform,txBurstGrid] = hSSBurst(txBurst);

SubcarrierSpacing: 30
NCRB_SSB: -20

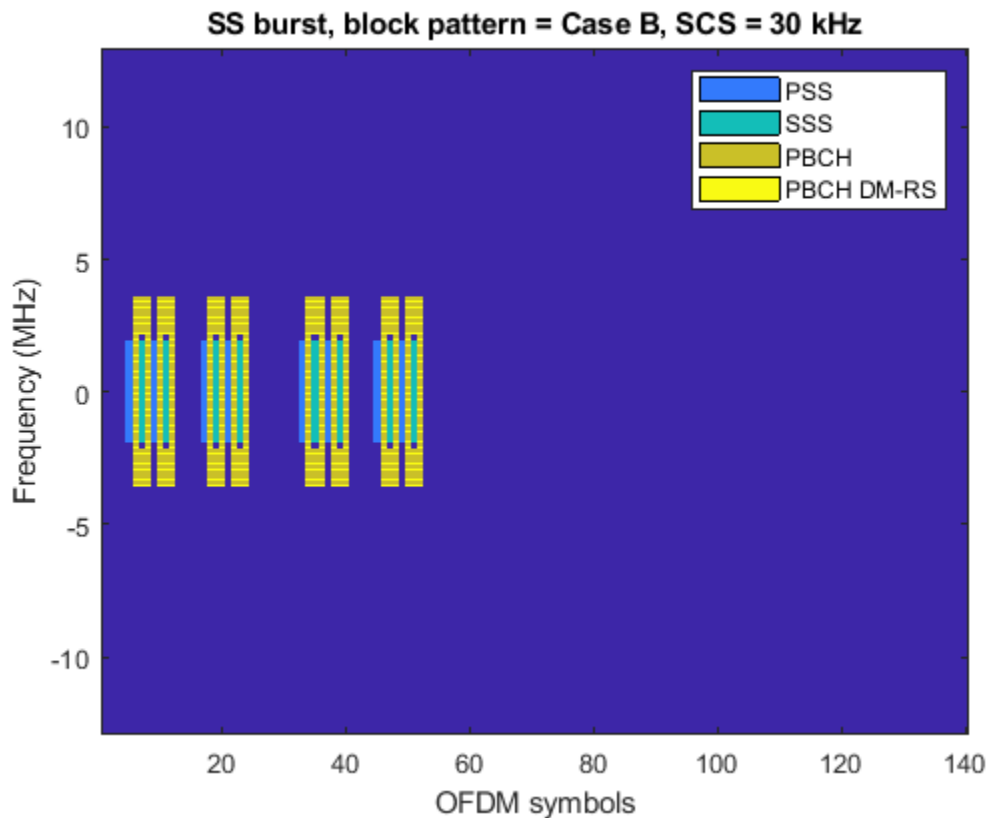
```



```

k_SSB: 0
FrequencyOffsetSSB: 0
MIB: [24x1 double]
L: 8
SSBIndex: [0 1 2 3 4 5 6 7]
i_SSB: [0 1 2 3 4 5 6 7]
ibar_SSB: [0 1 2 3 4 5 6 7]
SampleRate: 30720000
Nfft: 1024
NRB: 72
CyclicPrefix: 'Normal'
OccupiedSubcarriers: [24x1 double]
OccupiedSymbols: [8x4 double]
Windowing: 0

```



### Transmit-End Beam Sweeping

To achieve TRP beam sweeping, beamform each of the SS blocks in the generated burst using analog beamforming. Based on the number of SS blocks in the burst and the sweep ranges specified, determine both the azimuth and elevation directions for the different beams. Then beamform the individual blocks within the burst to each of these directions.

```

% Number of beams at both transmit and receive ends
numBeams = sum(txBurst.SSBTransmitted);

```

```

% Transmit beam angles in azimuth and elevation, equi-spaced

```

```

azBW = beamwidth(arrayTx,prm.CenterFreq,'Cut','Azimuth');
elBW = beamwidth(arrayTx,prm.CenterFreq,'Cut','Elevation');
txBeamAng = hGetBeamSweepAngles(numBeams,prm.TxAZlim,prm.TxELlim, ...
    azBW,elBW,prm.ElevationSweep);

% For evaluating transmit-side steering weights
SteerVecTx = phased.SteeringVector('SensorArray',arrayTx, ...
    'PropagationSpeed',c);

% Apply steering per OFDM symbol for each SSB
carrier.NSizeGrid = txBurstInfo.NRB;
ofdmInfo = nrOFDMInfo(carrier);
gridSymLengths = repmat(ofdmInfo.SymbolLengths,1, ...
    size(txBurstGrid,2)/length(ofdmInfo.SymbolLengths));
% repeat burst over numTx to prepare for steering
strTxWaveform = repmat(burstWaveform,1,prm.NumTx)./sqrt(prm.NumTx);
for ssb = 1:length(txBurstInfo.SSBIndex)

    % Extract SSB waveform from burst
    blockSymbols = txBurstInfo.OccupiedSymbols(ssb,:);
    startSSBInd = sum(gridSymLengths(1:blockSymbols(1)-1))+1;
    endSSBInd = sum(gridSymLengths(1:blockSymbols(4)));
    ssbWaveform = strTxWaveform(startSSBInd:endSSBInd,1);

    % Generate weights for steered direction
    wT = SteerVecTx(prm.CenterFreq,txBeamAng(:,ssb));

    % Apply weights per transmit element to SSB
    strTxWaveform(startSSBInd:endSSBInd,:) = ssbWaveform.*(wT');
end

```

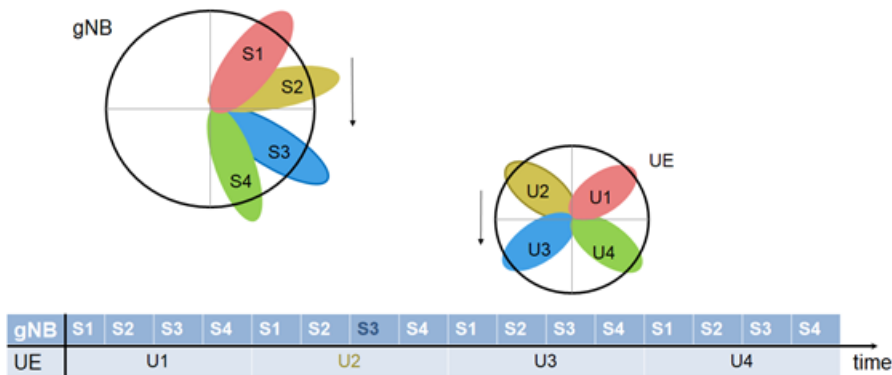
The beamformed burst waveform is then transmitted over the spatially-aware scattering channel.

### Receive-End Beam Sweeping and Measurement

For receive-end beam sweeping, the transmitted beamformed burst waveform is received successively over each receive beam. For  $N$  transmit beams and  $M$  receive beams in procedure P-1, each of the  $N$  beams is transmitted  $M$  times from gNB so that each transmit beam is received over the  $M$  receive beams.

The example assumes both  $N$  and  $M$  to be equal to the number of SSBs in the burst. For simplicity, the example generates only one burst, but to mimic the burst reception over the air  $M$  times, the receiver processes this single burst  $M$  times.

This figure shows a beam-based diagram for the sweeps at both gNB and UE for  $N = M = 4$ , in the azimuthal plane. The diagram shows the time taken for the dual sweep, where each interval at gNB corresponds to an SSB and each interval at the UE corresponds to the SS burst. For the depicted scenario, beams S3 and U2 are highlighted as the selected beam-pair link notionally. The example implements the dual-sweep over a time duration of  $N*M$  time instants.



The receive processing of the transmitted burst includes

- Application of the fading channel and AWGN
- Receive-end beamforming
- Timing correction
- OFDM demodulation
- Extracting the known SSB grid
- Measuring the RSRP based on the specified measurement mode

The processing repeats these steps for each of the receive beams, then selects the best beam-pair based on the complete set of measurements made.

To highlight beam sweeping, the example assumes known SSB information at the receiver. For more details on recovery processing see “NR Cell Search and MIB and SIB1 Recovery” on page 1-29.

For the idle mode SS-RSRP measurement, use either only the secondary synchronization signals (SSS) or the physical broadcast channel (PBCH) demodulation reference signals (DM-RS) in addition to the SSS (Section 5.1.1. of [ 4 ]). Specify this by the RSRPMode parameter of the example. For FR2, the RSRP measurement is based on the combined signal from antenna elements, while the measurement is per antenna element for FR1.

```
% Receive beam angles in azimuth and elevation, equi-spaced
azBW = beamwidth(arrayRx,prm.CenterFreq,'Cut','Azimuth');
elBW = beamwidth(arrayRx,prm.CenterFreq,'Cut','Elevation');
rxBeamAng = hGetBeamSweepAngles(numBeams,prm.RxAZlim,prm.RxELlim, ...
    azBW,elBW,prm.ElevationSweep);

% For evaluating receive-side steering weights
SteerVecRx = phased.SteeringVector('SensorArray',arrayRx, ...
    'PropagationSpeed',c);

% AWGN level
SNR = 10^(prm.SNRdB/20); % Convert to linear gain
N0 = 1/(sqrt(2.0*prm.NumRx*double(ofdmInfo.Nfft))*SNR); % Noise Std. Dev.

% Generate a reference grid for timing correction
% assumes an SSB in first slot
pssRef = nrPSS(carrier.NCellID);
pssInd = nrPSSIndices;
pbchdmrsRef = nrPBCHDMRS(carrier.NCellID,txBurstInfo.ibar_SSB(1));
pbchDMRSInd = nrPBCHDMRSIndices(carrier.NCellID);
```

```

pssGrid = zeros([240 4]);
pssGrid(pssInd) = pssRef;
pssGrid(pbchDMRSInd) = pbchdmrsRef;
refGrid = zeros([12*carrier.NSizeGrid ofdmInfo.SymbolsPerSlot]);
refGrid(txBurstInfo.OccupiedSubcarriers, ...
        txBurstInfo.OccupiedSymbols(1,:)) = pssGrid;

% Loop over all receive beams
rsrp = zeros(numBeams,numBeams);
for rIdx = 1:numBeams

    % Fading channel
    txWave = [strTxWaveform; zeros(maxChDelay,size(strTxWaveform,2))];
    fadWave = channel(txWave);

    % AWGN, after accounting for path loss
    noise = N0*complex(randn(size(fadWave)),randn(size(fadWave)));
    rxWaveform = fadWave*(10^(spLoss/20)) + noise;

    % Generate weights for steered direction
    wR = SteerVecRx(prm.CenterFreq,rxBeamAng(:,rIdx));

    % Apply weights per receive element
    if strcmp(prm.FreqRange, 'FR1')
        strRxWaveform = rxWaveform.*(wR');
    else % for FR2, combine signal from antenna elements
        strRxWaveform = rxWaveform*conj(wR);
    end

    % Correct timing
    offset = nrTimingEstimate(carrier, ...
        strRxWaveform(1:ofdmInfo.SampleRate*1e-3,:),refGrid*wR(1)');
    if offset > maxChDelay
        offset = 0;
    end
    strRxWaveformS = strRxWaveform(1+offset:end,:);

    % OFDM Demodulate
    rxGrid = nrOFDMDemodulate(carrier,strRxWaveformS);

    % Loop over all SSBs in rxGrid (transmit end)
    for tIdx = 1:numBeams
        % Get each SSB grid
        rxSSBGrid = rxGrid(txBurstInfo.OccupiedSubcarriers, ...
            txBurstInfo.OccupiedSymbols(tIdx,:),:);

        % Make measurements, store per receive, transmit beam
        rsrp(rIdx,tIdx) = measureSSB(rxSSBGrid,prm.RSRPMode,txBurst.NCellID);
    end
end
end

```

### Beam Determination

After the dual-end sweep and measurements are complete, determine the best beam-pair link based on the RSRP measurement.

```

[m,i] = max(rsrp,[],'all','linear'); % First occurrence is output
% i is column-down first (for receive), then across columns (for transmit)

```

```

[rxBeamID,txBeamID] = ind2sub([numBeams numBeams],i(1));

% Display the selected beam pair
disp(['Selected Beam pair with RSRP: ' num2str(10*log10(rsrp(rxBeamID, ...
    txBeamID))+30) ' dBm', 13 ' Transmit #' num2str(txBeamID) ...
    ' (Azimuth: ' num2str(txBeamAng(1,txBeamID)) ', Elevation: ' ...
    num2str(txBeamAng(2,txBeamID)) ') ' 13 ' Receive #' num2str(rxBeamID) ...
    ' (Azimuth: ' num2str(rxBeamAng(1,rxBeamID)) ', Elevation: ' ...
    num2str(rxBeamAng(2,rxBeamID)) ') ' ]);

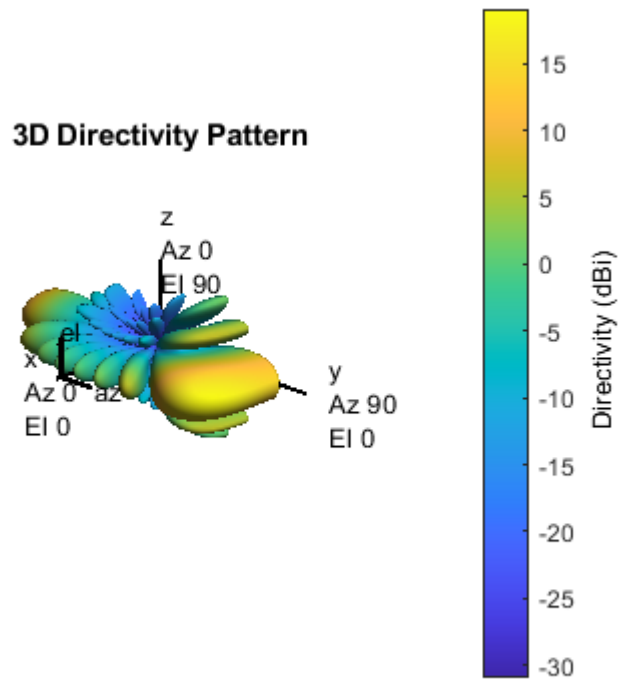
% Display final beam pair patterns
h = figure('Position',figposition([32 55 32 40]),'MenuBar','none');
h.Name = 'Selected Transmit Array Response Pattern';
wT = SteerVecTx(prm.CenterFreq,txBeamAng(:,txBeamID));
pattern(arrayTx,prm.CenterFreq,'PropagationSpeed',c,'Weights',wT);

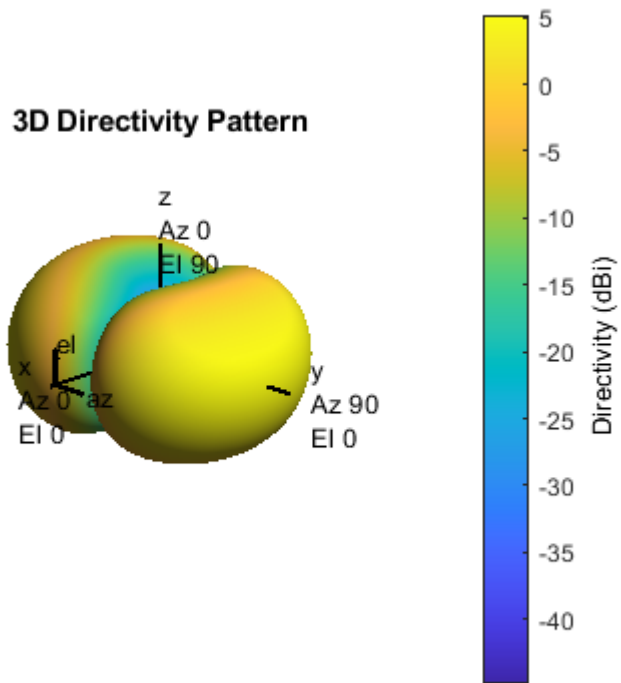
h = figure('Position',figposition([32 55 32 40]),'MenuBar','none');
h.Name = 'Selected Receive Array Response Pattern';
wR = SteerVecRx(prm.CenterFreq,rxBeamAng(:,rxBeamID));
pattern(arrayRx,prm.CenterFreq,'PropagationSpeed',c,'Weights',wR);

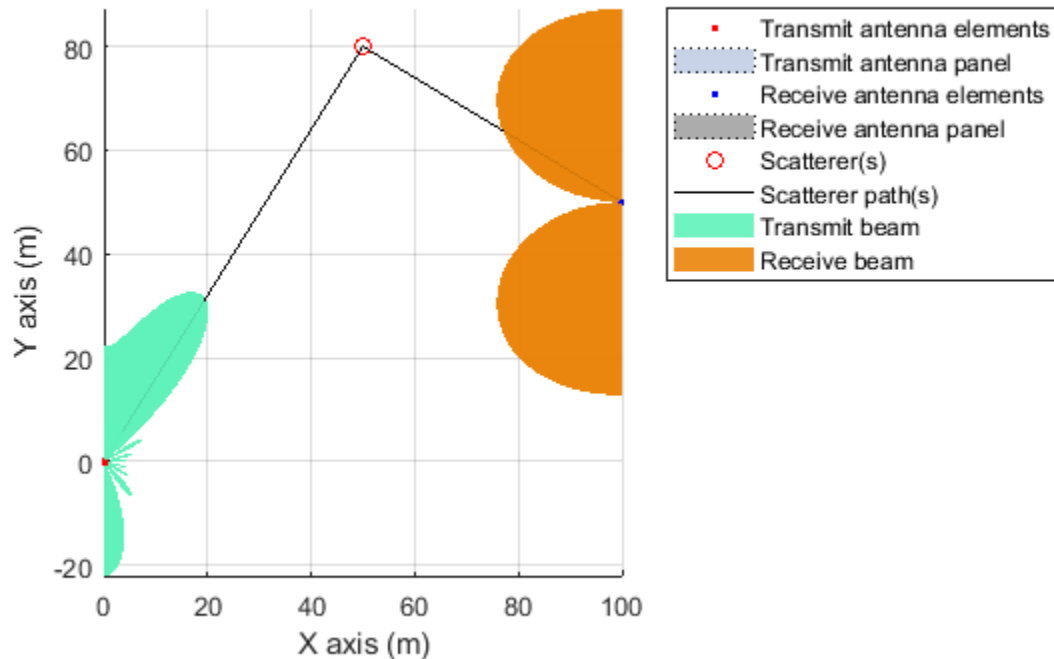
% Plot MIMO scenario with tx, rx, scatterers, and determined beams
prmScene = struct();
prmScene.txArraySize = prm.TxArraySize;
prmScene.rxArraySize = prm.RxArraySize;
prmScene.txElemPos = getElementPosition(arrayTx); % meters
prmScene.rxElemPos = getElementPosition(arrayRx); % meters
prmScene.txArrayPos = prm.posTx;
prmScene.rxArrayPos = prm.posRx;
prmScene.txAzAngles = -90:90;
prmScene.rxAzAngles = [90:180 -179:-90];
prmScene.scatPos = prm.ScatPos;
prmScene.lambda = lambda;
prmScene.arrayScaling = 1;
hPlotSpatialMIMOScene(prmScene,wT,wR);
if ~prm.ElevationSweep
    view(2);
end

```

Selected Beam pair with RSRP: 45.1564 dBm Transmit #8 (Azimuth: 60, Elevation: 0) Receive #6 (







These plots highlight the transmit directivity pattern, receive directivity pattern, and the spatial scene, respectively. The results are dependent on the individual beam directions used for the sweeps. The spatial scene offers a combined view of the transmit and receive arrays and the respective determined beams, along with the scatterers.

### Summary and Further Exploration

This example highlights the P-1 beam management procedure by using synchronization signal blocks for transmit-end and receive-end beam sweeping. By measuring the reference signal received power for SSBs, you can identify the best beam pair link for a selected spatial environment.

The example allows variation on frequency range, SSB block pattern, number of SSBs, transmit and receive array sizes, transmit and receive sweep ranges, and the measuring mode. To see the impact of parameters on the beam selection, experiment with different values.

For an example of the P-2 procedures of transmit-end beam sweeping using CSI-RS signals for the downlink, see “NR Downlink Transmit-End Beam Refinement Using CSI-RS” on page 1-110. You can use these procedures for beam refinement and adjustment in the connected mode, once the initial beam pair links are established [ 5 ], [ 6 ].

### References

- 1 3GPP TR 38.802. "Study on New Radio access technology physical layer aspects." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
- 2 Giordani, M., M. Polese, A. Roy, D. Castor, and M. Zorzi. "A tutorial on beam management for 3GPP NR at mmWave frequencies." *IEEE Comm. Surveys & Tutorials*, vol. 21, No. 1, Q1 2019.



- 3 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
- 4 3GPP TS 38.215. "NR; Physical layer measurements." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
- 5 Giordani, M., M. Polese, A. Roy, D. Castor, and M. Zorzi. "Standalone and non-standalone beam management for 3GPP NR at mmWaves." *IEEE Comm. Mag.*, April 2019, pp. 123-129.
- 6 Onggosanusi, E., S. Md. Rahman, et al. "Modular and high-resolution channel state information and beam management for 5G NR." *IEEE Comm. Mag.*, March 2018, pp. 48-55.

## Local Functions

```
function prm = validateParams(prm)
% Validate user specified parameters and return updated parameters
%
% Only cross-dependent checks are made for parameter consistency.

if strcmpi(prm.FreqRange, 'FR1')
    if prm.CenterFreq > 7.125e9 || prm.CenterFreq < 410e6
        error(['Specified center frequency is outside the FR1 ', ...
            'frequency range (410 MHz - 7.125 GHz).']);
    end
    if strcmpi(prm.SSBLOCKPattern, 'Case D') || ...
        strcmpi(prm.SSBLOCKPattern, 'Case E')
        error(['Invalid SSBLOCKPattern for selected FR1 frequency ' ...
            'range. SSBLOCKPattern must be one of ' 'Case A' or ' ...
            ''Case B' or 'Case C' for FR1.']);
    end
    if ~(length(prm.SSBTransmitted)==4) || ...
        (length(prm.SSBTransmitted)==8)
        error(['SSBTransmitted must be a vector of length 4 or 8', ...
            'for FR1 frequency range.']);
    end
    if (prm.CenterFreq <= 3e9) && (length(prm.SSBTransmitted)~=4)
        error(['SSBTransmitted must be a vector of length 4 for ' ...
            'center frequency less than or equal to 3GHz.']);
    end
    if (prm.CenterFreq > 3e9) && (length(prm.SSBTransmitted)~=8)
        error(['SSBTransmitted must be a vector of length 8 for ', ...
            'center frequency greater than 3GHz and less than ', ...
            'or equal to 7.125GHz.']);
    end
end
else % 'FR2'
    if prm.CenterFreq > 52.6e9 || prm.CenterFreq < 24.25e9
        error(['Specified center frequency is outside the FR2 ', ...
            'frequency range (24.25 GHz - 52.6 GHz).']);
    end
    if ~(strcmpi(prm.SSBLOCKPattern, 'Case D') || ...
        strcmpi(prm.SSBLOCKPattern, 'Case E'))
        error(['Invalid SSBLOCKPattern for selected FR2 frequency ' ...
            'range. SSBLOCKPattern must be either ' 'Case D' or ' ...
            ''Case E' for FR2.']);
    end
    if length(prm.SSBTransmitted)~=64
        error(['SSBTransmitted must be a vector of length 64 for ', ...
            'FR2 frequency range.']);
    end
end
```

```

end

prm.NumTx = prod(prm.TxArraySize);
prm.NumRx = prod(prm.RxArraySize);
if prm.NumTx==1 || prm.NumRx==1
    error(['Number of transmit or receive antenna elements must be', ...
          ' greater than 1.']);
end
prm.IsTxURA = (prm.TxArraySize(1)>1) && (prm.TxArraySize(2)>1);
prm.IsRxURA = (prm.RxArraySize(1)>1) && (prm.RxArraySize(2)>1);

if ~( strcmpi(prm.RSRPMode,'SSSonly') || ...
      strcmpi(prm.RSRPMode,'SSSwDMRS') )
    error(['Invalid RSRP measuring mode. Specify either ', ...
          ''SSSonly'' or ''SSSwDMRS'' as the mode.']);
end

end

% Select SCS based on SSBBlockPattern
switch lower(prm.SSBBlockPattern)
    case 'case a'
        scs = 15;
    case {'case b', 'case c'}
        scs = 30;
    case 'case d'
        scs = 120;
    case 'case e'
        scs = 240;
end
prm.SCS = scs;

end

function rsrp = measureSSB(rxSSBGrid,mode,NCellID)
% Compute the reference signal received power (RSRP) based on SSS, and if
% selected, also PBCH DM-RS.

    sssInd = nrSSSIndices; % SSS indices

    numRx = size(rxSSBGrid,3);
    rsrpSSS = zeros(numRx,1);
    for rxIdx = 1:numRx
        % Extract signals per rx element
        rxSSBGridperRx = rxSSBGrid(:,:,rxIdx);
        rxSSS = rxSSBGridperRx(sssInd);

        % Average power contributions over all REs for RS
        rsrpSSS(rxIdx) = mean(rxSSS.*conj(rxSSS));
    end

    if strcmpi(mode,'SSSwDMRS')
        pbchDMRSInd = nrPBCHDMRSIndices(NCellID); % PBCH DM-RS indices
        rsrpDMRS = zeros(numRx,1);
        for rxIdx = 1:numRx
            % Extract signals per rx element
            rxSSBGridperRx = rxSSBGrid(:,:,rxIdx);
            rxPBCHDMRS = rxSSBGridperRx(pbchDMRSInd);

            % Average power contributions over all REs for RS

```

```
        rsrpDMRS(rxIdx) = mean(rxPBCHDMRS.*conj(rxPBCHDMRS));
    end
end

switch lower(mode)
    case 'ssonly' % Only SSS
        rsrp = max(rsrpSSS); % max over receive elements
    case 'sswdmrs' % Both SSS and PBCH-DMRS
        rsrp = max(rsrpSSS+rsrpDMRS)/2; % max over receive elements
    end
end
```

## See Also

### Objects

phased.ScatteringMIMOChannel | phased.SteeringVector

## Related Examples

- “NR Cell Search and MIB and SIB1 Recovery” on page 1-29
- “NR Downlink Transmit-End Beam Refinement Using CSI-RS” on page 1-110

## NR Downlink Transmit-End Beam Refinement Using CSI-RS

This example demonstrates the downlink transmit-end beam refinement procedure using the channel state information reference signal (CSI-RS) from 5G Toolbox™. The example shows how to transmit multiple CSI-RS resources in different directions in a scattering environment and how to select the optimal transmit beam based on reference signal received power (RSRP) measurements.

### Introduction

In NR 5G, frequency range 2 (FR2) operates at millimeter wave (mmWave) frequencies (24.25 GHz to 52.6 GHz). As the frequency increases, the transmitted signal is prone to high path loss and penetration loss, which affects the link budget. To improve the gain and directionality of the transmission and reception of the signals at higher frequencies, beamforming is essential. Beam management is a set of Layer 1 (physical layer) and Layer 2 (medium access control) procedures to establish and retain an optimal beam pair (transmit beam and a corresponding receive beam) for good connectivity. TR 38.802 Section 6.1.6.1 [1 on page 1-0 ] defines beam management as three procedures:

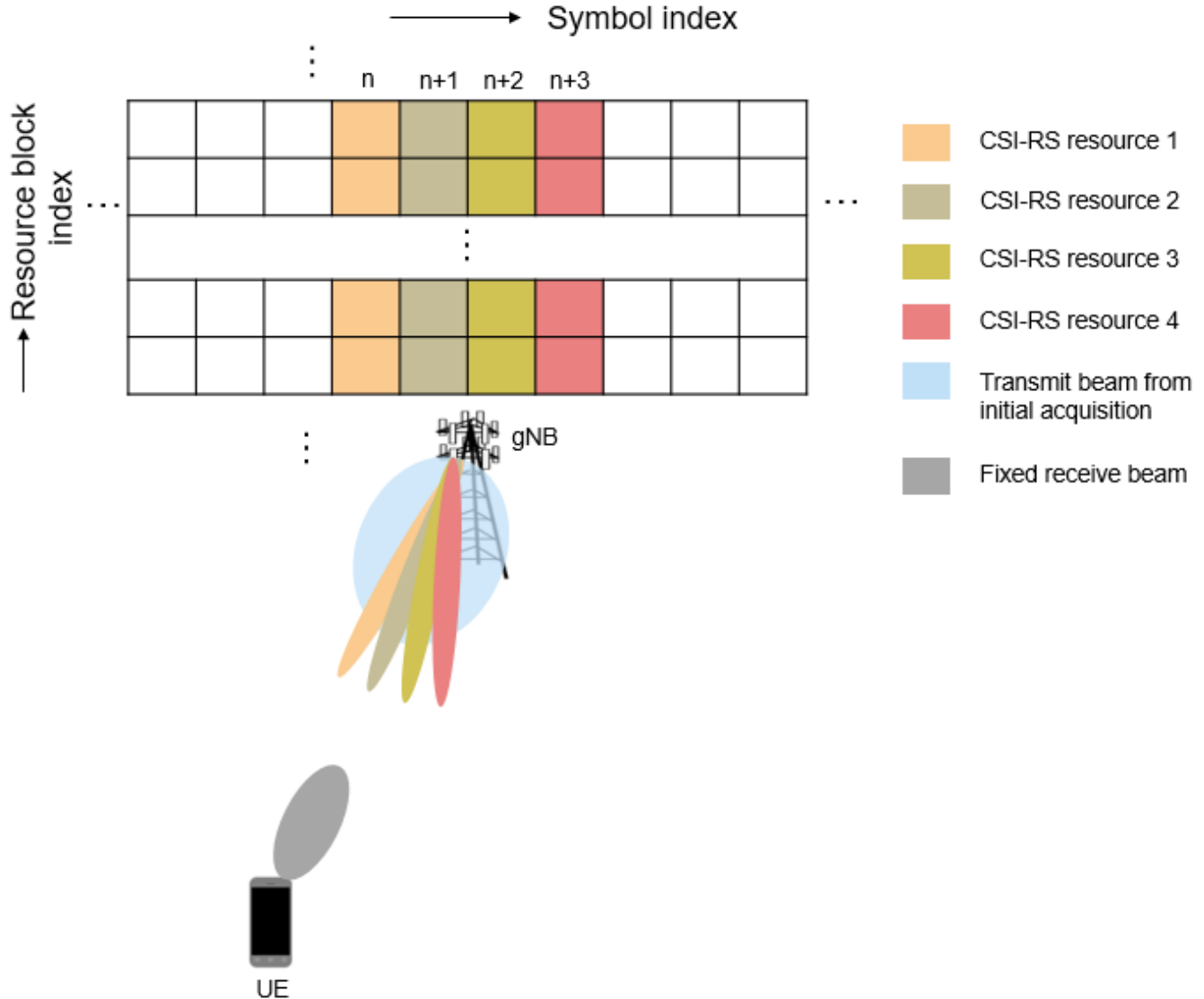
**Procedure 1 (P-1):** This procedure focuses on the initial acquisition based on synchronization signal blocks (SSB). During the initial acquisition, beam sweeping takes place at both transmit and receive ends to select the best beam pair based on the RSRP measurements. In general, the selected beams are wide and may not be an optimal beam pair for the data transmission and reception. For more details on this procedure, see “NR SSB Beam Sweeping” on page 1-95.

**Procedure 2 (P-2):** This procedure focuses on transmit-end beam refinement, where the beam sweeping happens at the transmit end by keeping the receive beam fixed. The procedure is based on non-zero-power CSI-RS (NZP-CSI-RS) for downlink transmit-end beam refinement and sounding reference signal (SRS) for uplink transmit-end beam refinement.

After the initial beam establishment, obtaining a unicast data transmission with high directivity and high gain requires a beam much finer than the SSB beam. Therefore, a set of reference signal resources are configured and transmitted in different directions by using finer beams within the angular range of the beam from the initial acquisition process. Then the user equipment (UE) or the access network node (gNB) measures all these beams by capturing the signals with a fixed receive beam. Finally, the best transmit beam is selected based on the RSRP measurements on all transmit beams.

**Procedure 3 (P-3):** This procedure focuses on receive-end beam adjustment, where the beam sweeping happens at the receive end given the current transmit beam. This process aims to find the best receive beam, which can be a neighbor beam or a refined beam. For this procedure, a set of reference signal resources (NZP-CSI-RS for downlink and SRS for uplink) are transmitted with the same transmit beam and the UE or gNB receives the signal using different beams from different directions covering an angular range. Finally, the best receive beam is selected based on the RSRP measurements on all receive beams.

This example focuses on downlink beam refinement at the transmitter. The example works for both frequency range 1 (FR1) and frequency range 2 (FR2) of NR 5G. This figure depicts the transmit-end beam refinement procedure, considering four NZP-CSI-RS resources transmitted in four different directions.



### Generate CSI-RS Resources

#### Configure Carrier

Create a carrier configuration object representing a 50 MHz carrier with subcarrier spacing of 30 kHz.

```
carrier = nrCarrierConfig;
% Maximum transmission bandwidth configuration for 50 MHz carrier with 30 kHz subcarrier spacing
carrier.NSizeGrid = 133;
carrier.SubcarrierSpacing = 60;
carrier.NSlot = 0;
carrier.NFrame = 0

carrier =
  nrCarrierConfig with properties:
```

```
        NCellID: 1
    SubcarrierSpacing: 60
        CyclicPrefix: 'normal'
        NSizeGrid: 133
        NStartGrid: 0
        NSlot: 0
        NFrame: 0

Read-only properties:
    SymbolsPerSlot: 14
    SlotsPerSubframe: 4
    SlotsPerFrame: 40
```

### Configure CSI-RS

Create a CSI-RS configuration object representing an NZP-CSI-RS resource set with `numNZPRes` number of NZP-CSI-RS resources. For Layer 1 RSRP measurements, configure all the CSI-RS resources in a resource set with the same number of antenna ports (either single-port or dual-port), as specified in TS 38.215 Section 5.1.2 [2 on page 1-0 ] or TS 38.214 Section 5.1.6.1.2 [3 on page 1-0 ]. This example works for single-port CSI-RS.

```
numNZPRes = 12;
csirs = nrCSIRSConfig;
csirs.CSIRSType = repmat({'nzp'},1,numNZPRes);
csirs.CSIRSPeriod = 'on';
csirs.Density = repmat({'one'},1,numNZPRes);
csirs.RowNumber = repmat(2,1,numNZPRes);
csirs.SymbolLocations = {0,1,2,3,4,5,6,7,8,9,10,11};
csirs.SubcarrierLocations = repmat({0},1,numNZPRes);
csirs.NumRB = 25
```

```
csirs =
nrCSIRSConfig with properties:

    CSIRSType: {1x12 cell}
    CSIRSPeriod: 'on'
    RowNumber: [2 2 2 2 2 2 2 2 2 2 2 2]
    Density: {1x12 cell}
    SymbolLocations: {1x12 cell}
    SubcarrierLocations: {1x12 cell}
    NumRB: 25
    RBOffset: 0
    NID: 0

Read-only properties:
    NumCSIRSPorts: [1 1 1 1 1 1 1 1 1 1 1 1]
    CDMAType: {1x12 cell}
```

```
% Validate CSI-RS antenna ports
validateCSIRSPorts(csirs);
```

```
% Get the binary vector to represent the presence of each CSI-RS resource
% in a specified slot
csirsTransmitted = getActiveCSIRSRes(carrier,csirs);
```

Configure the power scaling of all NZP-CSI-RS resources in decibels (dB).

```
powerCSIRS = 0;
```

### Generate CSI-RS Symbols and Indices

Generate CSI-RS symbols and indices by using the `carrier` and `csirs` configuration objects. To distinguish each CSI-RS resource output separately, specify the `OutputResourceFormat`, 'cell' name-value pair.

```
csirsSym = nrCSIRS(carrier,csirs,'OutputResourceFormat','cell')
```

```
csirsSym=1x12 cell array
Columns 1 through 4
```

```
{25x1 double} {25x1 double} {25x1 double} {25x1 double}
```

```
Columns 5 through 8
```

```
{25x1 double} {25x1 double} {25x1 double} {25x1 double}
```

```
Columns 9 through 12
```

```
{25x1 double} {25x1 double} {25x1 double} {25x1 double}
```

```
csirsInd = nrCSIRSIndices(carrier,csirs,'OutputResourceFormat','cell')
```

```
csirsInd=1x12 cell array
Columns 1 through 4
```

```
{25x1 uint32} {25x1 uint32} {25x1 uint32} {25x1 uint32}
```

```
Columns 5 through 8
```

```
{25x1 uint32} {25x1 uint32} {25x1 uint32} {25x1 uint32}
```

```
Columns 9 through 12
```

```
{25x1 uint32} {25x1 uint32} {25x1 uint32} {25x1 uint32}
```

### Configure Antenna Arrays and Scatterers

#### Configure Transmit and Receive Antenna Arrays

Configure the carrier frequency and the signal propagation speed.

```
% Set the carrier frequency
```

```
fc = 3.5e9;
```

```
freqRange = validateFc(fc);
```

```
% Set the propagation speed
```

```
c = physconst('LightSpeed');
```

```
% Calculate wavelength
```

```
lambda = c/fc;
```

Configure the size of transmit and receive antenna arrays as a two-element vector, where the first element represents the number of rows and the second element represents the number of columns in the antenna array.

```
txArySize = [8 8];
rxArySize = [2 2];
```

Calculate the total number of transmit and receive antenna elements.

```
nTx = prod(txArySize);
nRx = prod(rxArySize);
```

Configure the positions of transmit and receive antenna arrays. Then calculate the free space path loss based on the spatial separation between transmit and receive antenna array positions.

```
% Configure antenna array positions
txArrayPos = [0;0;0];
rxArrayPos = [100;50;0];

% Calculate the free space path loss
toRxRange = rangeangle(txArrayPos,rxArrayPos);
spLoss = fspl(toRxRange,lambda);
```

Configure the uniform linear array (ULA) or uniform rectangular array (URA) based on the sizes of antennas arrays.

```
% Initialize the flags to choose between URA and ULA
isTxRectArray = false;
isRxRectArray = false;

% Enable isTxRectArray if both the number of rows and columns of transmit
% antenna array are greater than one
if ~any(txArySize == 1)
    isTxRectArray = true;
end
% Enable isRxRectArray if both the number of rows and columns of receive
% antenna array are greater than one
if ~any(rxArySize == 1)
    isRxRectArray = true;
end

% Configure the transmit and receive antenna elements
txAntenna = phased.IsotropicAntennaElement('BackBaffled',true); % To avoid transmission beyond -
% degrees from the broadside, ba
% the back of the transmit anten
% element by setting the BackBa
% property to true
rxAntenna = phased.IsotropicAntennaElement('BackBaffled',false); % To receive the signal from 360
% set the BackBaffled property t

% Configure transmit antenna array
if isTxRectArray
    % Create a URA System object for signal transmission
    txArray = phased.URA('Element',txAntenna,'Size',txArySize,'ElementSpacing',lambda/2);
else
    % Create a ULA System object for signal transmission
    txArray = phased.ULA('Element',txAntenna,'NumElements',nTx,'ElementSpacing',lambda/2);
end

% Configure receive antenna array
if isRxRectArray
    % Create a URA System object for signal reception
```



```

    rxArray = phased.URA('Element', rxAntenna, 'Size', rxArySize, 'ElementSpacing', lambda/2);
else
    % Create a ULA System object for signal reception
    rxArray = phased.ULA('Element', rxAntenna, 'NumElements', nRx, 'ElementSpacing', lambda/2);
end

```

### Configure Scatterers

```

fixedScatMode = true;
rng(42);
if fixedScatMode
    % Fixed single scatterer location
    numScat = 1;
    scatPos = [60;10;15];
else
    % Generate scatterers at random positions
    numScat = 10; %#ok<UNRCH>
    azRange = -180:180;
    randAzOrder = randperm(length(azRange));
    elRange = -90:90;
    randElOrder = randperm(length(elRange));
    azAngInSph = deg2rad(azRange(randAzOrder(1:numScat)));
    elAngInSph = deg2rad(elRange(randElOrder(1:numScat)));
    r = 20;

    % Transform spherical coordinates to cartesian coordinates
    [x,y,z] = sph2cart(azAngInSph,elAngInSph,r);
    scatPos = [x;y;z] + (txArrayPos + rxArrayPos)/2;
end

```

### Transmit Beamforming and OFDM Modulation

#### Calculate the Steering Vectors

Create the steering vector System object for transmit antenna array.

```
txArrayStv = phased.SteeringVector('SensorArray', txArray, 'PropagationSpeed', c);
```

Calculate the angle of scatterer position with respect to the transmit antenna array.

```
[~,scatAng] = rangeangle(scatPos(:,1),txArrayPos); % Pointing towards the first scatterer position
```

Configure the azimuth and elevation beamwidths of SSB transmit beam from the initial acquisition process (P-1).

```
azTxBeamWidth = 30; % In degrees
elTxBeamWidth = 30; % In degrees
```

Get the SSB transmit beam direction which is aligned (partially or fully) to the position of scatterer, by using the beamwidths in azimuth and elevation planes.

```
ssbTxAng = getInitialBeamDir(scatAng,azTxBeamWidth,elTxBeamWidth);
```

Calculate the beam directions (azimuth and elevation angle pairs) for all active CSI-RS resources within the angular range covered by the SSB transmit beam.

```
% Get the number of transmit beams based on the number of active CSI-RS resources in a slot
numBeams = sum(csirsTransmitted);
```

```

% Get the azimuthal sweep range based on the SSB transmit beam direction
% and its beamwidth in azimuth plane
azSweepRange = [ssbTxAng(1) - azTxBeamWidth/2 ssbTxAng(1) + azTxBeamWidth/2];

% Get the elevation sweep range based on the SSB transmit beam direction
% and its beamwidth in elevation plane
elSweepRange = [ssbTxAng(2) - elTxBeamWidth/2 ssbTxAng(2) + elTxBeamWidth/2];

% Get the azimuth and elevation angle pairs for all NZP-CSI-RS transmit beams
azBW = beamwidth(txArray,fc,'Cut','Azimuth');
elBW = beamwidth(txArray,fc,'Cut','Elevation');
csirsBeamAng = hGetBeamSweepAngles(numBeams,azSweepRange,elSweepRange,azBW,elBW);

```

Calculate the steering vectors for all active CSI-RS resources.

```

wT = zeros(nTx,numBeams);
for beamIdx = 1:numBeams
    tempW = txArrayStv(fc,csirsBeamAng(:,beamIdx));
    wT(:,beamIdx) = tempW;
end

```

### Apply Digital Beamforming

Loop over all NZP-CSI-RS resources and apply the digital beamforming to all the active ones. Digital beamforming is considered to offer frequency selective beamforming within the same OFDM symbol.

```

% Number of CSI-RS antenna ports
ports = csirs.NumCSIRSPorts(1);
% Initialize the beamformed grid
bfGrid = nrResourceGrid(carrier,nTx);
% Get the active NZP-CSI-RS resource indices
activeRes = find(logical(csirsTransmitted));
for resIdx = 1:numNZPRes
    % Initialize the carrier resource grid for one slot and map NZP-CSI-RS symbols onto
    % the grid
    txSlotGrid = nrResourceGrid(carrier,ports);
    txSlotGrid(csirsInd{resIdx}) = db2mag(powerCSIRS)*csirsSym{resIdx};
    reshapedSymb = reshape(txSlotGrid,[],ports);

    % Get the transmit beam index
    beamIdx = find(activeRes == resIdx);

    % Apply the digital beamforming
    if ~isempty(beamIdx)
        bfSymb = reshapedSymb * wT(:,beamIdx)';
        bfGrid = bfGrid + reshape(bfSymb,size(bfGrid));
    end
end

```

### Perform OFDM Modulation

Generate the time-domain waveform by performing the OFDM modulation.

```

% Perform OFDM modulation
[tbfWaveform,ofdmInfo] = nrOFDMModulate(carrier,bfGrid);

% Normalize the beamformed time-domain waveform over the number of transmit
% antennas
tbfWaveform = tbfWaveform/sqrt(nTx);

```

## Scattering MIMO Channel

### Configure the Channel

Configure the scattering-based MIMO propagation channel by using the System object `phased.ScatteringMIMOChannel`.

```
chan = phased.ScatteringMIMOChannel;
chan.PropagationSpeed = c;
chan.CarrierFrequency = fc;
chan.Polarization = 'none';
chan.SpecifyAtmosphere = false;
chan.SampleRate = ofdmInfo.SampleRate;
chan.SimulateDirectPath = false;
chan.ChannelResponseOutputPort = true;

% Configure transmit array parameters
chan.TransmitArray = txArray;
chan.TransmitArrayMotionSource = 'property';
chan.TransmitArrayPosition = txArrayPos;

% Configure receive array parameters
chan.ReceiveArray = rxArray;
chan.ReceiveArrayMotionSource = 'property';
chan.ReceiveArrayPosition = rxArrayPos;

% Configure scatterers
chan.ScattererSpecificationSource = 'property';
chan.ScattererPosition = scatPos;
chan.ScattererCoefficient = ones(1,numScat);

% Get the maximum channel delay by transmitting random signal
[~,~,tau] = chan(complex(randn(chan.SampleRate*1e-3,nTx), ...
    randn(chan.SampleRate*1e-3,nTx)));
maxChDelay = ceil(max(tau)*chan.SampleRate);
```

### Apply the Channel to Transmit Waveform

```
% Append zeros to the transmit waveform to account for channel delay
tbfWaveform = [tbfWaveform; zeros(maxChDelay,nTx)];
% Pass the signal through the channel
fadWave = chan(tbfWaveform);
```

### Apply AWGN to Received Waveform

```
% Configure SNR
SNRdB = 20;           % in dB
SNR = 10^(SNRdB/20); % Linear value

% Calculate the standard deviation for AWGN
N0 = 1/(sqrt(2.0*nRx*double(ofdmInfo.Nfft))*SNR);

% Generate AWGN
noise = N0*complex(randn(size(fadWave)),randn(size(fadWave)));

% Apply AWGN to the waveform after accounting for free space path loss
rxWaveform = fadWave*(10^(spLoss/20)) + noise;
```

## Timing Synchronization

Perform the timing synchronization by cross correlating the received reference symbols with a local copy of NZP-CSI-RS symbols.

```
% Generate reference symbols and indices
refSym = nrCSIRS(carrier,csirs);
refInd = nrCSIRSIndices(carrier,csirs);

% Estimate timing offset
offset = nrTimingEstimate(carrier,rxWaveform,refInd,refSym);
if offset > maxChDelay
    offset = 0;
end

% Correct timing offset
syncTdWaveform = rxWaveform(1+offset:end,:);
```

## OFDM Demodulation and Receive Beamforming

### OFDM Demodulation

OFDM demodulate the synchronized time-domain waveform.

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

### Calculate Steering Vector

Create the steering vector System object for the receive antenna array.

```
rxArrayStv = phased.SteeringVector('SensorArray',rxArray,'PropagationSpeed',c);
```

Calculate the angle of scatterer position with respect to the receive antenna array. Assuming this as receive beam direction from the initial acquisition process using SSB.

```
[~,scatRxAng] = rangeangle(scatPos(:,1),rxArrayPos); % Pointing towards the first scatterer position
```

Configure the azimuth and elevation beamwidths of receive beam from the initial acquisition process (P-1).

```
azRxBeamWidth = 30; % In degrees
elRxBeamWidth = 30; % In degrees
```

Get the initial receive beam direction which is aligned (partially or fully) to the position of scatterer, by using the beamwidths in azimuth and elevation planes from P-1.

```
rxAng = getInitialBeamDir(scatRxAng,azRxBeamWidth,elRxBeamWidth);
```

Calculate the steering vector for the angle of reception.

```
wR = rxArrayStv(fc,rxAng);
```

### Apply Receive Beamforming

To perform digital beamforming at the receiver side, apply the steering weights to rxGrid, with the assumption that there is no other signal present in rxGrid (single UE scenario). Combine the signals from all receive antenna elements in case of FR2, as specified in TS 38.215 Section 5.1.2 [2 on page 1-0 ].

```

temp = rxGrid;
if strcmpi(freqRange, 'FR1')
    % Beamforming without combining
    rbfGrid = reshape(reshape(temp, [], nRx).*wR', size(temp,1), size(temp,2), []);
else % 'FR2'
    % Beamforming with combining
    rbfGrid = reshape(reshape(temp, [], nRx)*conj(wR), size(temp,1), size(temp,2), []);
end

```

### Plot the Scattering MIMO Scenario

Get the element positions of transmit and receive antenna arrays for plotting.

```

% Get the positions of transmit antenna elements (in meters)
txElemPos = getElementPosition(txArray);
% Get the positions of receive antenna elements (in meters)
rxElemPos = getElementPosition(rxArray);

```

Configure the MIMO scene parameters.

```

sceneParams.txElemPos = txElemPos;
sceneParams.rxElemPos = rxElemPos;
sceneParams.txArraySize = txArySize;
sceneParams.rxArraySize = rxArySize;
sceneParams.txArrayPos = txArrayPos;
sceneParams.rxArrayPos = rxArrayPos;
sceneParams.txAzAngles = -90:90;           % Vector of azimuth angles, where the pattern of
                                           % transmit antenna array is calculated
sceneParams.txElAngles = -90:90;          % Vector of elevation angles, where the pattern of
                                           % transmit antenna array is calculated
sceneParams.rxAzAngles = [90:180 -179:-90]; % Vector of azimuth angles, where the pattern of
                                           % receive antenna array is calculated
sceneParams.rxElAngles = -90:90;          % Vector of elevation angles, where the pattern of
                                           % receive antenna array is calculated

sceneParams.scatPos = scatPos;
sceneParams.lambda = lambda;
sceneParams.arrayScaling = 100;           % To enlarge antenna arrays in the plot

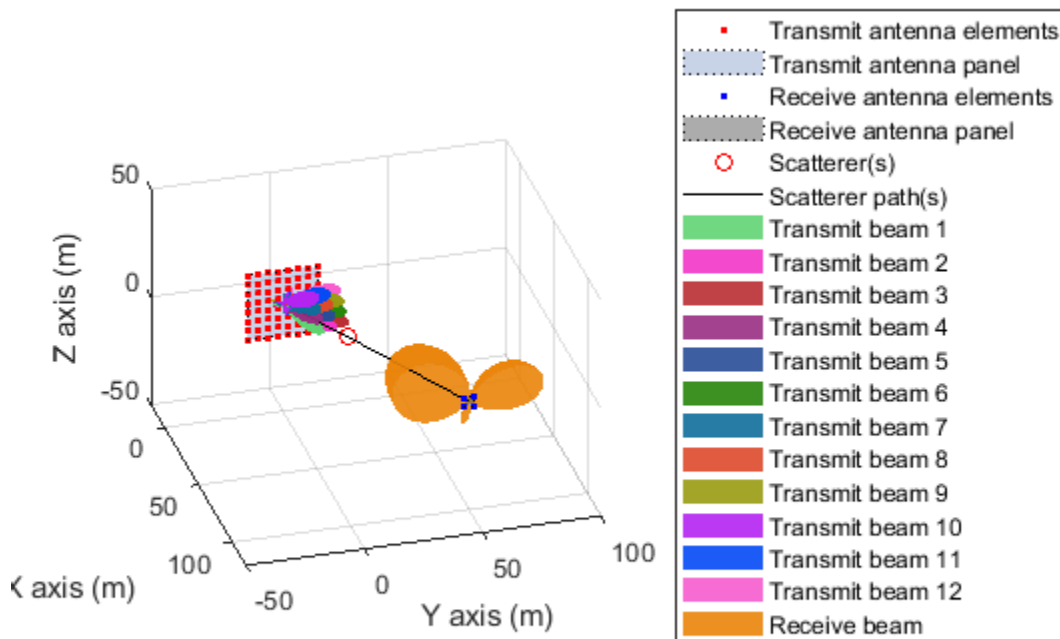
```

Plot the scattering MIMO scenario (including transmit and receive antenna arrays, scatterer positions and their paths, and all the transmit and receive antenna array beam patterns) by using the helper function `hPlotSpatialMIMOScene`.

```

hPlotSpatialMIMOScene(sceneParams, wT, wR);
xlim([-20 120]);
ylim([-50 100]);
zlim([-50 50]);
view([74 29]);

```



### Beam Determination

After the OFDM demodulation, the UE measures the RSRP for all the CSI-RS resources transmitted in different beams, given the current receive beam. Perform these measurements by using the helper function `hCSIRSMeasurements`.

```
% Perform RSRP measurements
meas = hCSIRSMeasurements(carrier,csirs,rbfGrid);
```

```
% Display the measurement quantities for all CSI-RS resources in dBm
RSRPaBm = meas.RSRPaBm;
disp(['RSRP measurements of all CSI-RS resources (in dBm):' 13 num2str(RSRPaBm)]);
```

```
RSRP measurements of all CSI-RS resources (in dBm):
40.1313      31.0269      27.0399      42.9433      34.8756      29.2513      37.7758      31.342
```

Identify the maximum RSRP value from the measurements and find the best corresponding beam.

```
% Get the transmit beam index with maximum RSRP value
[~,maxRSRPIdx] = max(RSRPaBm(logical(csirsTransmitted)));
```

```
% Get the CSI-RS resource index with maximum RSRP value
[~,maxRSRPResIdx] = max(RSRPaBm);
```

Calculate the beamwidth which corresponds to the refined transmit beam.

```
% Get the steering weights corresponding to refined transmit beam
if numBeams == 0
```

```

disp('Refinement has not happened, as NZP-CSI-RS is not transmitted')
else
refBeamWts = wT(:,maxRSRPIdx);
csirsAzBeamWidth = beamwidth(txArray,fc,'PropagationSpeed',c,'Weights',refBeamWts);
csirsElBeamWidth = beamwidth(txArray,fc,'PropagationSpeed',c,'Weights',refBeamWts,'Cut','Elev');
disp(['From initial beam acquisition:' 13 ' Beamwidth of initial SSB beam in azimuth plane is: '...
num2str(azTxBeamWidth) ' degrees' 13 ...
' Beamwidth of initial SSB beam in elevation plane is: '...
num2str(elTxBeamWidth) ' degrees' 13 13 ...
'With transmit-end beam refinement:' 13 ' Refined transmit beam ('...
num2str(maxRSRPIdx) ') corresponds to CSI-RS resource '...
num2str(maxRSRPResIdx) ' is selected in the direction ['...
num2str(csirsBeamAng(1,maxRSRPIdx)) ';' num2str(csirsBeamAng(2,maxRSRPIdx))...
']' 13 ' Beamwidth of refined transmit beam in azimuth plane is: '...
num2str(csirsAzBeamWidth) ' degrees' 13 ...
' Beamwidth of refined transmit beam in elevation plane is: '...
num2str(csirsElBeamWidth) ' degrees']);
end

```

From initial beam acquisition:

```

Beamwidth of initial SSB beam in azimuth plane is: 30 degrees
Beamwidth of initial SSB beam in elevation plane is: 30 degrees

```

With transmit-end beam refinement:

```

Refined transmit beam (4) corresponds to CSI-RS resource 4 is selected in the direction [10;
Beamwidth of refined transmit beam in azimuth plane is: 12.98 degrees
Beamwidth of refined transmit beam in elevation plane is: 13.25 degrees

```

## Summary and Further Exploration

This example highlights the beam refinement procedure (P-2) using NZP-CSI-RS. The procedure identifies a transmit beam that is finer than the beam from the initial acquisition.

You can configure multiple CSI-RS resources, transmit and receive antenna array configurations, and multiple scatterers to see the variations in the selection of the refined beam. You can also configure the azimuth and elevation angle pairs for the signal transmission and reception.

## References

- 1 3GPP TR 38.802. "Study on New Radio access technology physical layer aspects." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
- 2 3GPP TS 38.215. "NR; Physical layer measurements." *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*.

## Local Functions

```

function validateCSIRSPorts(csirs)
% validateCSIRSPorts validates the CSI-RS antenna ports, given the
% CSI-RS configuration object CSIRS.

numPorts = csirs.NumCSIRSPorts;
if any(numPorts > 1)
    error('nr5g:PortsGreaterThan1','CSI-RS resources must be configured for single-port for
end
end

```

```

function csirsTransmitted = getActiveCSIRSRes(carrier,csirs)
% getActiveCSIRSRes returns a binary vector indicating the presence of
% all CSI-RS resources in a specified slot, given the carrier
% configuration object CARRIER and CSI-RS configuration object CSIRS.

% Extract the following properties of carrier
NSlotA      = carrier.NSlot;          % Absolute slot number
NFrameA     = carrier.NFrame;        % Absolute frame number
SlotsPerFrame = carrier.SlotsPerFrame; % Number of slots per frame

% Calculate the appropriate frame number (0...1023) based on the
% absolute slot number
NFrameR = mod(NFrameA + fix(NSlotA/SlotsPerFrame),1024);
% Relative slot number (0...slotsPerFrame-1)
NSlotR = mod(NSlotA,SlotsPerFrame);

% Loop over the number of CSI-RS resources
numCSIRSRes = numel(csirs.CSIRSRes);
csirsTransmitted = zeros(1,numCSIRSRes);
csirs_struct = validateConfig(csirs);
for resIdx = 1:numCSIRSRes
    % Extract the CSI-RS slot periodicity and offset
    if isnumeric(csirs_struct.CSIRSPeriod{resIdx})
        Tcsi_rs = csirs_struct.CSIRSPeriod{resIdx}(1);
        Toffset = csirs_struct.CSIRSPeriod{resIdx}(2);
    else
        if strcmpi(csirs_struct.CSIRSPeriod{resIdx},'on')
            Tcsi_rs = 1;
        else
            Tcsi_rs = 0;
        end
        Toffset = 0;
    end
    % Check for the presence of CSI-RS, based on slot periodicity and offset
    if (Tcsi_rs ~= 0) && (mod(SlotsPerFrame*NFrameR + NSlotR - Toffset, Tcsi_rs) == 0)
        csirsTransmitted(resIdx) = 1;
    end
end
end

function freqRange = validateFc(fc)
% validateFc validates the carrier frequency FC and returns the frequency
% range as either 'FR1' or 'FR2'.

if fc >= 410e6 && fc <= 7.125e9
    freqRange = 'FR1';
elseif fc >= 24.25e9 && fc <= 52.6e9
    freqRange = 'FR2';
else
    error('nr5g:invalidFreq',['Selected carrier frequency is outside '...
        'FR1 (410 MHz to 7.125 GHz) and FR2 (24.25 GHz to 52.6 GHz).']);
end
end

function beamDir = getInitialBeamDir(scatAng,azBeamWidth,elBeamWidth)
% getInitialBeamDir returns the initial beam direction BEAMDIR, given the

```



```

% angle of scatterer position with respect to transmit or receive antenna
% array SCATANG, beamwidth of transmit or receive beam in azimuth plane
% AZBEAMWIDTH, and beamwidth of transmit or receive beam in elevation
% plane ELBEAMWIDTH.

% Azimuth angle boundaries of all transmit/receive beams
azSSBSweep = -180:azBeamWidth:180;
% Elevation angle boundaries of all transmit/receive beams
elSSBSweep = -90:elBeamWidth:90;

% Get the azimuth angle of transmit/receive beam
azIdx1 = find(azSSBSweep <= scatAng(1),1,'last');
azIdx2 = find(azSSBSweep >= scatAng(1),1,'first');
azAng = (azSSBSweep(azIdx1) + azSSBSweep(azIdx2))/2;

% Get the elevation angle of transmit/receive beam
elIdx1 = find(elSSBSweep <= scatAng(2),1,'last');
elIdx2 = find(elSSBSweep >= scatAng(2),1,'first');
elAng = (elSSBSweep(elIdx1) + elSSBSweep(elIdx2))/2;

% Form the azimuth and elevation angle pair (in the form of [az;el])
% for transmit/receive beam
beamDir = [azAng;elAng];
end

```

## See Also

### Objects

nrCSIRSConfig | nrCarrierConfig

## Related Examples

- “NR Cell Search and MIB and SIB1 Recovery” on page 1-29
- “NR SSB Beam Sweeping” on page 1-95



# Uplink Channels

---

## 5G NR Uplink Carrier Waveform Generation

This example implements a 5G NR uplink carrier waveform generator using 5G Toolbox™.

### Introduction

This example shows how to parameterize and generate a 5G New Radio (NR) uplink waveform. The following channels and signals can be generated:

- PUSCH and its associated DM-RS and PT-RS
- PUCCH and its associated DM-RS
- SRS

This example supports the parameterization and generation of multiple bandwidth parts (BWP). Multiple instances of PUSCH, PUCCH and SRS can be generated over the different BWPs. The example allows to configure PUCCH, PUSCH and SRS for a specific UE categorized by RNTI and transmits only PUSCH for that specific RNTI when both PUCCH and PUSCH overlap in a slot.

### Waveform and Carrier Configuration

This section sets the subcarrier spacing (SCS) specific carrier bandwidths in resource blocks, the physical layer cell identity NCellID, and the length of the generated waveform in subframes. You can visualize the generated resource grids by setting the DisplayGrids field to 1. The channel bandwidth and frequency range parameters are used to display the associated minimum guardbands on a schematic diagram of the SCS carrier alignment. The schematic diagram is displayed in one of the output plots of the example.

```

waveconfig = [];
waveconfig.NCellID = 0;           % Cell identity
waveconfig.ChannelBandwidth = 50; % Channel bandwidth (MHz)
waveconfig.FrequencyRange = 'FR1'; % 'FR1' or 'FR2'
waveconfig.NumSubframes = 10;    % Number of 1ms subframes in generated waveform
                                   % (1,2,4,8 slots per 1ms subframe, depending on SCS)
waveconfig.DisplayGrids = 1;     % Display the resource grids after signal generation

% Define a set of SCS specific carriers, using the maximum sizes for a 50
% MHz NR channel. See TS 38.101-1 for more information on defined
% bandwidths
carriers = [];
carriers(1).SubcarrierSpacing = 15;
carriers(1).NRB = 270;
carriers(1).RBStart = 0;

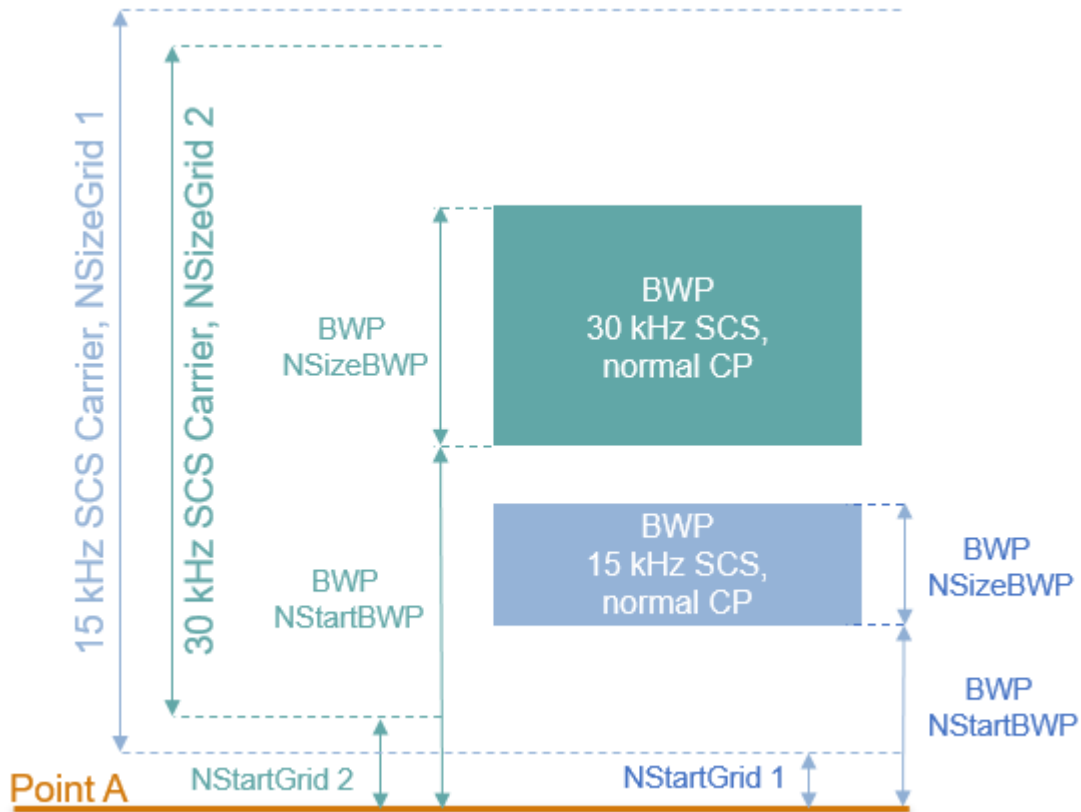
carriers(2).SubcarrierSpacing = 30;
carriers(2).NRB = 133;
carriers(2).RBStart = 1;

```

### Bandwidth Parts

A BWP is formed by a set of contiguous resources sharing a numerology on a given SCS specific carrier. This example supports the use of multiple BWPs using a struct array. Each entry in the array represents a BWP. For each BWP you can specify the subcarrier spacing (SCS), the cyclic prefix (CP) length and the bandwidth. The SubcarrierSpacing parameter maps the BWP to one of the SCS specific carriers defined earlier. The RBOffset parameter controls the location of the BWP in the

carrier. This is expressed in terms of the BWP numerology. Different BWPs can overlap with each other.



```
% Bandwidth parts configurations
bwp = [];
```

```
bwp(1).SubcarrierSpacing = 15;
bwp(1).CyclicPrefix = 'Normal';
bwp(1).NRB = 25;
bwp(1).RBOffset = 10;
```

```
% BWP1 Subcarrier Spacing
% BWP1 cyclic prefix
% Size of BWP1
% Position of BWP1 in carrier
```

```
bwp(2).SubcarrierSpacing = 30;
bwp(2).CyclicPrefix = 'Normal';
bwp(2).NRB = 51;
bwp(2).RBOffset = 40;
```

```
% BWP2 Subcarrier Spacing
% BWP2 cyclic prefix
% Size of BWP2
% Position of BWP2 in carrier
```

### PUCCH Instances Configuration

This section specifies the parameters for the set of PUCCH instances in the waveform. Each element in the structure array defines a PUCCH sequence instance. The following parameters can be set:

- Enable/disable the PUCCH sequence
- Specify the BWP carrying the PUCCH
- PUCCH instance power in dB

- Slots within a period used for PUCCH
- Periodicity of the allocation. Use empty to indicate no repetition
- DM-RS power boosting in dB

```
pucch = [];
pucch(1).Enable = 1;           % Enable PUCCH sequence
pucch(1).BWP = 1;             % Bandwidth part
pucch(1).Power = 0;           % Power scaling in dB
pucch(1).AllocatedSlots = [3 4]; % Allocated slots within a period
pucch(1).AllocatedPeriod = 6; % Allocation slot period (empty implies no repetition)
pucch(1).PowerDMRS = 1;       % Additional power boosting in dB
```

### PUCCH Resource Configuration

This section specifies the PUCCH sequence resource related parameters. The parameters can be categorized into the following sections:

- Enable/Disable the PUCCH dedicated resource. If this is disabled, it uses common resource as per TS 38.213 Section 9.2.1
- Provide the resource index value (0...15), when dedicated resource is disabled and the cyclic prefix of BWP transmitting PUCCH is normal. In this case, the resource and format parameters for the PUCCH transmission are filled up directly based on the resource index. All the other parameters that are provided for resource and format configurations are not considered.

When the dedicated resource is enabled or when the dedicated resource is disabled with the cyclic prefix of BWP transmitting PUCCH is extended, the following resource parameters need to be provided:

- Specify the index of first PRB prior to frequency hopping or for no frequency hopping within the BWP
- Specify the index of first PRB after frequency hopping within the BWP
- Intra-slot frequency hopping configuration ('enabled','disabled')
- Group hopping configuration ('neither','enable','disable')

and the following format specific parameters need to be provided:

- PUCCH format configuration in the resource (0...4)
- Starting symbol index allocated for PUCCH transmission
- Number of OFDM symbols allocated for PUCCH transmission. For PUCCH formats 1, 3 and 4, the number of OFDM symbols allocated are in range 4 to 14, and for formats 0 and 2, it is either 1 or 2
- Initial cyclic shift for formats 0 and 1. The value is in range 0 to 11
- Modulation scheme for formats 3 and 4 ('QPSK','pi/2-BPSK')
- Number of resource blocks allocated for format 2 and 3. The nominal value is one of the set {1,2,3,4,5,6,8,9,10,12,15,16}
- Spreading factor for format 4. The value is either 2 or 4
- Orthogonal cover code index for formats 1 and 4. For format 1, the value is in range 0 to 6. For format 4, the value is less than spreading factor and greater than or equal to 0
- Indicate the presence of additional DM-RS for formats 3 and 4. The value is either 0 or 1

Scrambling identities to be used for different formats

- RNTI for formats 2/3/4. It is used for sequence generation. It is in range 0 to 65535
- Scrambling identity (NID) for PUCCH formats 2/3/4. It is in range 0 to 1023. Use empty ([]) to use physical layer cell identity. It is used in sequence generation. This parameter is provided by higher-layer parameter *dataScramblingIdentityPUSCH*
- PUCCH hopping identity for formats 0/1/3/4. Use empty ([]) to use physical layer cell identity. The value is used in sequence generation for format 0, both sequence and DM-RS generation for format 1 and only for DM-RS generation for formats 3 and 4
- DM-RS scrambling NID for PUCCH format 2. It is in range 0 to 65535. Use empty ([]) to use physical layer cell identity

Irrespective of dedicated resource configuration, the following parameters are to be provided for slot repetitions:

- Specify the number of slot repetitions for formats 1,3,4 (2 or 4 or 8). For no slot repetition, the value can be specified as 1
- Specify the inter-slot frequency hopping for formats 1,3,4 ('enabled','disabled'). If this is enabled and the number of slot repetitions is more than one, then intra-slot frequency hopping is disabled
- Specify the maximum code rate. The nominal value is one of the set {0.08, 0.15, 0.25, 0.35, 0.45, 0.6, 0.8}

*% Dedicated resource parameters*

```
pucch(1).DedicatedResource = 1;           % Enable/disable the dedicated resource configuration
% Provide the resource index value when dedicated resource is disabled. The
% PUCCH resource is configured based on the resource index value, as per
% the table 9.2.1-1 of Section 9.2.1, TS 38.213.
pucch(1).ResourceIndex = 0;              % Resource index for PUCCH dedicated resource (0...1)
```

```
% When dedicated resource is enabled or when the dedicated resource is
% disabled with the cyclic prefix of BWP transmitting PUCCH is extended,
% the resource index value is ignored and the parameters specified below
% for the resource and format configurations are considered.
```

*% Resource parameters*

```
pucch(1).StartPRB = 0;                    % Index of first PRB prior to frequency hopping or f
pucch(1).SecondHopPRB = 1;               % Index of first PRB after frequency hopping
pucch(1).IntraSlotFreqHopping = 'enabled'; % Indication for intra-slot frequency hopping ('enab
pucch(1).GroupHopping = 'enable';        % Group hopping configuration ('enable','disable','ne
```

*% Format specific parameters*

```
pucch(1).PUCCHFormat = 3;                % PUCCH format 0/1/2/3/4
pucch(1).StartSymbol = 3;                % Starting symbol index
pucch(1).NrOfSymbols = 11;               % Number of OFDM symbols allocated for PUCCH
pucch(1).InitialCS = 3;                  % Initial cyclic shift for format 0 and 1
pucch(1).OCCI = 0;                       % Orthogonal cover code index for format 1 and 4
pucch(1).Modulation = 'QPSK';            % Modulation for format 3/4 ('pi/2-BPSK','QPSK')
pucch(1).NrOfRB = 9;                     % Number of resource blocks for format 2/3
pucch(1).SpreadingFactor = 4;            % Spreading factor for format 4, value is either 2 or
pucch(1).AdditionalDMRS = 1;             % Additional DM-RS (0/1) for format 3/4
```

*% Scrambling identities of PUCCH and PUCCH DM-RS*

```
pucch(1).RNTI = 0;                       % RNTI (0...65535) for formats 2/3/4
pucch(1).NID = 1;                        % PUCCH scrambling identity (0...1023) for formats 2
pucch(1).HoppingId = 1;                  % PUCCH hopping identity (0...1023) for formats 0/1/3
pucch(1).NIDDMRS = 1;                    % DM-RS scrambling identity (0...65535) for PUCCH fo
```

```

% Multi-slot configuration parameters
pucch(1).NrOfSlots = 1; % Number of slots for PUCCH repetition (1/2/4/8). One
pucch(1).InterSlotFreqHopping = 'disabled'; % Indication for inter-slot frequency hopping ('enab

% Code rate - This parameter is used when there is multiplexing of UCI part
% 1 (HARQ-ACK, SR, CSI part 1) and UCI part 2 (CSI part 2) to get the rate
% matching lengths of each UCI part
pucch(1).MaxCodeRate = 0.15; % Maximum code rate (0.08, 0.15, 0.25, 0.35, 0.45, 0

```

### UCI payload configuration

Configure the UCI payload based on the format configuration

- Enable or disable the UCI coding for formats 2/3/4
- Number of HARQ-ACK bits. For formats 0 and 1, value can be at most 2. Set the value to 0, for no HARQ-ACK transmission
- Number of SR bits. For formats 0 and 1, value can be at most 1. Set the value to 0, for no SR transmission
- Number of CSI part 1 bits for formats 2/3/4. Set value to 0, for no CSI part 1 transmission
- Number of CSI part 2 bits for formats 3/4. Set value to 0, for no CSI part 2 transmission. The value is ignored when there are no CSI part 1 bits

Note that the generator in the example transmits UCI information on PUSCH whenever there is a overlap between PUCCH and PUSCH for a specific RNTI in a BWP. The parameters to be configured for UCI transmission on PUSCH are provided in the section UCI on PUSCH. It requires the lengths of UCI and UL-SCH to be transmitted on PUSCH.

```

pucch(1).EnableCoding = 1; % Enable UCI coding
pucch(1).LenACK = 5; % Number of HARQ-ACK bits
pucch(1).LenSR = 5; % Number of SR bits
pucch(1).LenCSI1 = 10; % Number of CSI part 1 bits (for formats 2/3/4)
pucch(1).LenCSI2 = 10; % Number of CSI part 2 bits (for formats 3/4)

pucch(1).DataSource = 'PN9'; % UCI data source

% UCI message data source. You can use one of the following standard PN
% sequences: 'PN9-ITU', 'PN9', 'PN11', 'PN15', 'PN23'. The seed for the
% generator can be specified using a cell array in the form |{'PN9',seed}|.
% If no seed is specified, the generator is initialized with all ones

```

### Specifying Multiple PUCCH Instances

A second PUCCH sequence instance is specified next using the second BWP.

```

% PUCCH sequence instance specific to second BWP
pucch(2) = pucch(1);
pucch(2).BWP = 2;
pucch(2).StartSymbol = 10;
pucch(2).NrOfSymbols = 2;
pucch(2).PUCCHFormat = 2;
pucch(2).AllocatedSlots = 0:2;
pucch(2).AllocatedPeriod = [];
pucch(2).RNTI = 10;

```



## PUSCH Instances Configuration

This section specifies the set of PUSCH instances in the waveform using a struct array. This example defines two PUSCH sequence instances.

### General Parameters

The following parameters are set for each instance:

- Enable/disable this PUSCH sequence
- Specify the BWP this PUSCH maps to. The PUSCH will use the SCS specified for this BWP
- Power scaling in dB
- Enable/disable the UL-SCH transport coding
- Scrambling identity (NID) for PUSCH bits. It is in range 0 to 1023. Use empty ([]) to use physical layer cell identity
- RNTI
- Transform precoding (0,1). The value of 1, enables the transform precoding and the resultant waveform is DFT-s-OFDM. When the value is 0, the resultant waveform is CP-OFDM
- Target code rate used to calculate the transport block sizes.
- Overhead parameter. It is used to calculate the length of transport block size. It is one of the set {0, 6, 12, 18}
- Transmission scheme ('codebook', 'nonCodebook'). When the transmission scheme is 'codebook', the MIMO precoding is enabled and a precoding matrix is selected based on the number of layers, number of antenna ports and the transmitted precoding matrix indicator. When the transmission is set to 'nonCodebook', an identity matrix is used, leading to no MIMO precoding
- Modulation scheme ('pi/2-BPSK', 'QPSK', '16QAM', '64QAM', '256QAM'). Nominally, the modulation scheme 'pi/2-BPSK' is used when transform precoding is enabled
- Number of layers (1...4). The number of layers is restricted to a maximum of 4 in uplink as there is only one code word transmission. Nominally, the number of layers is set to 1 when transform precoding is enabled. This value is ignored, when PortSet field is specified
- Number of antenna ports (1,2,4). It is used when codebook transmission is enabled. The number of antenna ports must be greater than or equal to number of DM-RS ports configured
- Transmitted precoding matrix indicator (0...27). It depends on the number of layers and the number of antenna ports
- Redundancy version (RV) sequence
- Intra-slot frequency hopping ('enabled', 'disabled')
- Resource block offset for second hop. It is used when frequency (Intra-slot/Inter-slot) hopping is enabled
- Inter-slot frequency hopping ('enabled', 'disabled'). If this is enabled, intra-slot frequency hopping is disabled, the starting position of resource block in the allocated PRB of PUSCH in the bandwidth part depends on the whether the slot is even-numbered or odd-numbered
- Transport block data source. You can use one of the following standard PN sequences: 'PN9-ITU', 'PN9', 'PN11', 'PN15', 'PN23'. The seed for the generator can be specified using a cell array in the form { 'PN9' , seed }. If no seed is specified, the generator is initialized with all ones

```
pusch = [];
pusch(1).Enable = 1;                                % Enable PUSCH config
```

```

pusch(1).BWP = 1; % Bandwidth part
pusch(1).Power = 0; % Power scaling in dB
pusch(1).EnableCoding = 1; % Enable the UL-SCH transport coding
pusch(1).NID = 1; % Scrambling for data part (0...1023)
pusch(1).RNTI = 0; % RNTI
pusch(1).TransformPrecoding = 0; % Transform precoding flag (0 or 1)
pusch(1).TargetCodeRate = 0.47; % Code rate used to calculate transport block sizes
pusch(1).Xoh_PUSCH = 0; % Overhead. It is one of the set {0,6,12,18}

% Transmission settings
pusch(1).TxScheme = 'codebook'; % Transmission scheme ('codebook','nonCodebook')
pusch(1).Modulation = 'QPSK'; % 'pi/2-BPSK','QPSK','16QAM','64QAM','256QAM'
pusch(1).NLayers = 2; % Number of PUSCH layers (1...4)
pusch(1).NAntennaPorts = 4; % Number of antenna ports (1,2,4). It must not be less than 2
pusch(1).TPMI = 0; % Transmitted precoding matrix indicator (0...27)
pusch(1).RVSequence = [0 2 3 1]; % RV sequence to be applied cyclically across the PUSCH
pusch(1).IntraSlotFreqHopping = 'disabled'; % Intra-slot frequency hopping ('enabled','disabled')
pusch(1).RBOffset = 10; % Resource block offset for second hop

% Multi-slot transmission
pusch(1).InterSlotFreqHopping = 'enabled'; % Inter-slot frequency hopping ('enabled','disabled')

% Data source
pusch(1).DataSource = 'PN9'; % Transport block data source

```

### Allocation

You can set the following parameters to control the PUSCH allocation.

- PUSCH mapping type. It can be either 'A' or 'B'.
- Symbols in a slot where the PUSCH is mapped to. It needs to be a contiguous allocation. For PUSCH mapping type 'A', the start symbol within a slot must be zero and the length can be from 4 to 14 (for normal CP) and up to 12 (for extended CP). For PUSCH mapping type 'B', the start symbol can be from any symbol in the slot
- Slots in a frame used for the PUSCH
- Period of the allocation in slots. If this is empty it indicates no repetition
- The allocated PRBs are relative to the BWP

```

pusch(1).PUSCHMappingType = 'A'; % PUSCH mapping type ('A'(slot-wise),'B'(non slot-wise))
pusch(1).AllocatedSymbols = 0:13; % Range of symbols in a slot
pusch(1).AllocatedSlots = [0 1]; % Allocated slots indices
pusch(1).AllocatedPeriod = 5; % Allocation period in slots (empty implies no repetition)
pusch(1).AllocatedPRB = 0:10; % PRB allocation

```

### DM-RS Configuration

Set the DM-RS parameters

```

% DM-RS configuration (TS 38.211 section 6.4.1.1)
pusch(1).DMRSConfigurationType = 1; % DM-RS configuration type (1,2)
pusch(1).NumCDMGroupsWithoutData = 2; % Number of DM-RS CDM groups without data. The value can be 1 or 2
pusch(1).PortSet = [0 2]; % DM-RS antenna ports to use for the layers, when field is empty, all ports are used
pusch(1).DMRSTypeAPosition = 2; % Mapping type A only. First DM-RS symbol position (2,3)
pusch(1).DMRSLength = 1; % Number of front-loaded DM-RS symbols (1(single symbol),2,3)
pusch(1).DMRSAdditionalPosition = 2; % Additional DM-RS symbol positions (max range 0...3)
pusch(1).NIDNSCID = 1; % Scrambling identity for CP-OFDM (0...65535). Use empty for random scrambling

```

```

pusch(1).NSCID = 0; % Scrambling initialization for CP-OFDM (0,1)
pusch(1).NRSID = 0; % Scrambling identity for DFT-s-OFDM DM-RS (0...1007). Us
pusch(1).PowerDMRS = 0; % Additional power boosting in dB
pusch(1).GroupHopping = 'enable'; % {'enable','disable','neither'}. This parameter is used

```

The parameter GroupHopping is used in DM-RS sequence generation when transform precoding is enabled. This can be set to

- 'enable' to indicate the presence of group hopping. It is configured by higher-layer parameter *sequenceGroupHopping*
- 'disable' to indicate the presence of sequence hopping. It is configured by higher-layer parameter *sequenceHopping*
- 'neither' to indicate both group hopping and sequence hopping are not present

Note: The number of DM-RS CDM groups without data depends on the configuration type. The maximum number of DM-RS CDM groups can be 2 for DM-RS configuration type 1 and it can be 3 for DM-RS configuration type 2.

### PT-RS Configuration

Set the PT-RS parameters

```

% PT-RS configuration (TS 38.211 section 6.4.1.2)
pusch(1).EnablePTRS = 0; % Enable or disable the PT-RS (1 or 0)
pusch(1).PTRSTimeDensity = 1; % Time density (L_PT-RS) of PT-RS (1,2,4)
pusch(1).PTRSFrequencyDensity = 2; % Frequency density (K_PT-RS) of PT-RS for CP-OFDM (2,4)
pusch(1).PTRSNumSamples = 2; % Number of PT-RS samples (NGroupSamp) for DFT-s-OFDM (2,4)
pusch(1).PTRSNumGroups = 2; % Number of PT-RS groups (NPTRSGroup) for DFT-s-OFDM (2,4,8)
pusch(1).PTRSREOffset = '00'; % PT-RS resource element offset for CP-OFDM ('00','01','10',
pusch(1).PTRSPortSet = 0; % PT-RS antenna ports must be a subset of DM-RS ports for CP
pusch(1).PTRSNID = 0; % PT-RS scrambling identity for DFT-s-OFDM (0...1007)
pusch(1).PowerPTRS = 0; % Additional PT-RS power boosting in dB for CP-OFDM

```

```

% When PT-RS is enabled for CP-OFDM, the DM-RS ports must be in range from
% 0 to 3 for DM-RS configuration type 1, and in the range from 0 to 5 for
% DM-RS configuration type 2.
% When PT-RS is enabled for DFT-s-OFDM and the number of PT-RS groups is
% set to 8, the number of PT-RS samples must be set to 4.

```

### UCI on PUSCH

The following parameters must be set to transmit UCI on PUSCH in overlapping slots:

- Disable UL-SCH transmission on the overlapping slots of PUSCH (1/0). When set to 1, UL-SCH transmission is disabled on PUSCH. The example considers there is UL-SCH transmission all the time on PUSCH. A provision is provided to disable the UL-SCH transmission on the overlapping slots of PUSCH and PUCCH
- BetaOffsetACK, BetaOffsetCSI1 and BetaOffsetCSI2 can be set from the tables 9.3-1, 9.3-2 TS 38.213 Section 9.3
- ScalingFactor is provided by higher layer parameter *scaling*, as per TS 38.212, Section 6.3.2.4. The possible value is one of the set {0.5, 0.65, 0.8, 1}. This is used to limit the number of resource elements assigned to UCI on PUSCH

```

pusch(1).DisableULSCH = 1; % Disable UL-SCH on overlapping slots of PUSCH and PUCCH
pusch(1).BetaOffsetACK = 1; % Power factor of HARQ-ACK

```

```
pusch(1).BetaOffsetCSI1 = 2;           % Power factor of CSI part 1
pusch(1).BetaOffsetCSI2 = 2;           % Power factor of CSI part 2
pusch(1).ScalingFactor = 1;            % Scaling factor (0.5, 0.65, 0.8, 1)
```

### Specifying Multiple PUSCH Instances

A second PUSCH sequence instance is specified next using the second BWP.

```
pusch(2) = pusch(1);
pusch(2).Enable = 1;
pusch(2).BWP = 2;
pusch(2).AllocatedSymbols = 0:11;
pusch(2).AllocatedSlots = [5 6 7 8];
pusch(2).AllocatedPRB = 5:10;
pusch(2).AllocatedPeriod = 10;
pusch(2).TransformPrecoding = 1;
pusch(2).IntraSlotFreqHopping = 'disabled';
pusch(2).GroupHopping = 'neither';
pusch(2).NLayers = 1;
pusch(2).PortSet = 1;
pusch(2).RNTI = 0;
```

### SRS Instances Configuration

This section specifies the parameters for the set of SRS instances in the waveform. Each element in the structure array defines an SRS sequence instance. This example defines two SRS sequence instances that are disabled. The following parameters can be set:

- Enable/Disable this SRS sequence
- BWP carrying the SRS
- Number of SRS antenna ports (1,2,4).
- Number of OFDM symbols allocated for SRS transmission (1,2,4)
- Starting OFDM symbol of the SRS transmission within a slot. It must be (8...13) for normal CP and (6...11) for extended CP
- Slots within a period used for SRS transmission
- Periodicity of the allocation. Use empty to indicate no repetition
- Starting position of the SRS sequence in the BWP in RBs
- Additional frequency offset from the starting position in 4-PRB blocks
- Bandwidth and frequency hopping configuration. The occupied bandwidth depends on the parameters CSRS, BSRS, and BHop. Set BHop < BSRS to enable frequency hopping.
- Transmission comb to specify the SRS frequency density in subcarriers (2,4)
- Offset of the transmission comb in subcarriers
- Cyclic shift rotating the low-PAPR base sequence. The maximum number of cyclic shifts, 8 or 12, depends on the transmission comb number, 2 or 4. For 4 SRS antenna ports, the subcarrier set allocated to the SRS in the first and third antenna ports depends on the cyclic shift.
- Number of repeated SRS symbols within a slot. It disables frequency hopping in blocks of Repetition symbols. Set Repetition = 1 for no repetition.
- Group or sequence hopping. It can be 'neither', 'groupHopping' or 'sequenceHopping'
- Scrambling identity. It initializes the pseudo-random binary sequence when group or sequence hopping are enabled.

```

srs = struct();
srs(1).Enable = 0; % Enable SRS config
srs(1).BWP = 1; % BWP Index
srs(1).NumSRSPorts = 1; % Number of SRS ports (1,2,4)
srs(1).NumSRSSymbols = 4; % Number of SRS symbols in a slot (1,2,4)
srs(1).SymbolStart = 10; % Time-domain position of the SRS in the slot. (8...13) for 1
srs(1).AllocatedSlots = 2; % Allocated slots indices
srs(1).AllocatedPeriod = 5; % Allocation period in slots (empty implies no repetition)
srs(1).FreqStart = 0; % Frequency position of the SRS in BWP in RBs
srs(1).NRRC = 0; % Additional offset from FreqStart specified in blocks of 4 RBs
srs(1).CSRS = 13; % Bandwidth configuration C_SRS (0...63). It controls the all
srs(1).BSRS = 2; % Bandwidth configuration B_SRS (0...3). It controls the all
srs(1).BHop = 1; % Frequency hopping configuration (0...3). Set BHop < BSRS to
srs(1).KTC = 2; % Comb number (2,4). It indicates the allocation of the SRS o
srs(1).KBarTC = 0; % Subcarrier offset of the SRS sequence (0...KTC-1)
srs(1).CyclicShift = 0; % Cyclic shift number (0...NCSmax-1). NCSmax = 8 for KTC = 2
srs(1).Repetition = 1; % Repetition factor (1,2,4). It indicates the number of equa
srs(1).GroupSeqHopping = 'neither'; % Group or sequence hopping ('neither', 'groupHopping', 'sequ
srs(1).NSRSID = 0; % Scrambling identity (0...1023)

```

### Specifying Multiple SRS Instances

A second SRS sequence instance is specified next using the second BWP.

```

srs(2) = srs(1);
srs(2).Enable = 0;
srs(2).BWP = 2;
srs(2).NumSRSSymbols = 2;
srs(2).SymbolStart = 12;
srs(2).AllocatedSlots = [5 6 7 8];
srs(2).AllocatedPeriod = 10;
srs(2).BSRS = 0;
srs(2).BHop = 0;

```

### Waveform Generation

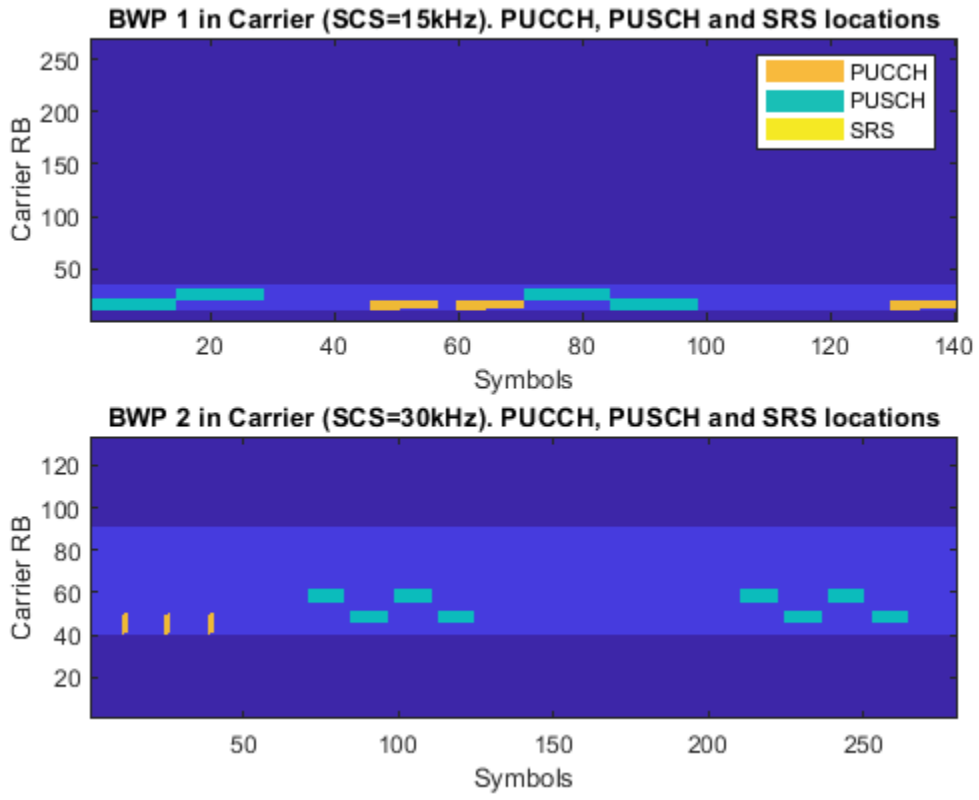
This section collects all the parameters into the carrier configuration and generates the waveform.

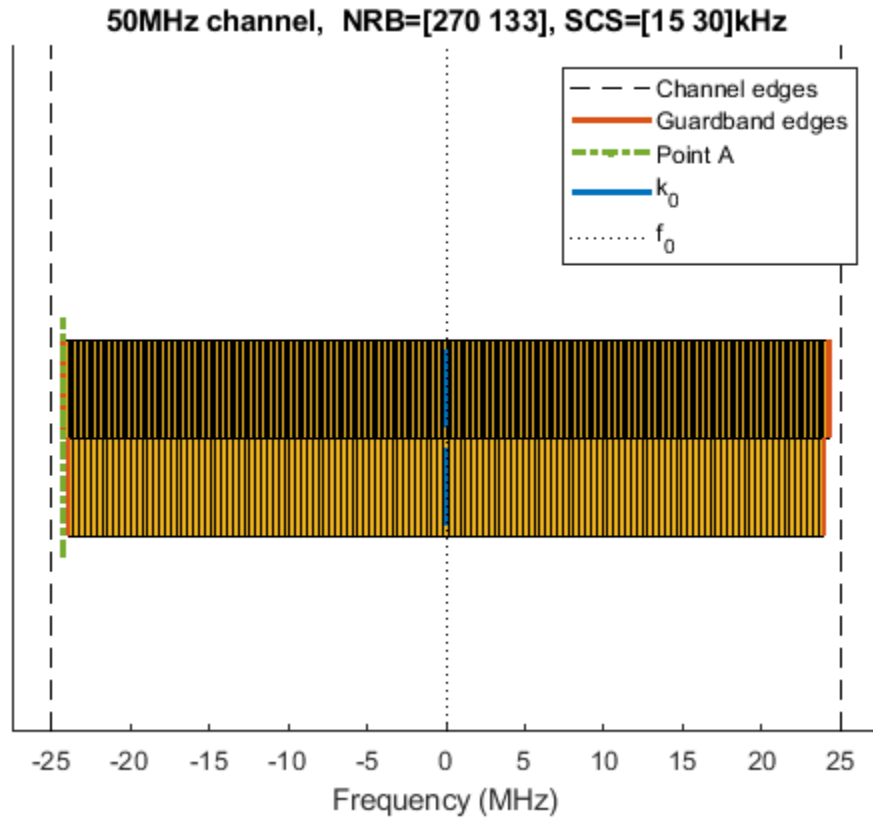
```

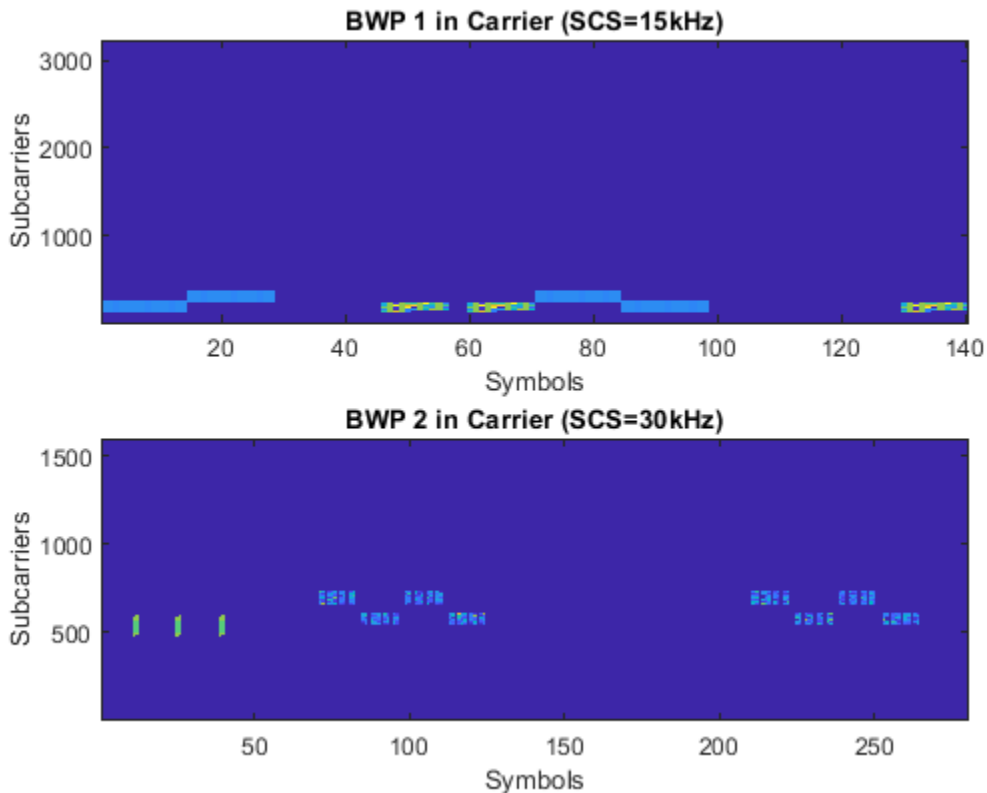
% Collect together channel oriented parameter sets into a single
% configuration
waveconfig.Carriers = carriers;
waveconfig.BWP = bwp;
waveconfig.PUCCH = pucch;
waveconfig.PUSCH = pusch;
waveconfig.SRS = srs;

% Generate complex baseband waveform
[waveform,bwpset] = hNRUplinkWaveformGenerator(waveconfig);

```







The waveform generator also plots the SCS carrier alignment and the resource grids for the bandwidth parts (this is controlled by the field `DisplayGrids` in the carrier configuration). The following plots are generated:

- Resource grid showing the location of the components (PUCCH, PUSCH and SRS) in each BWP. This does not plot the power of the signals, just their location in the grid
- Schematic diagram of SCS carrier alignment with the associated guardbands
- Generated waveform in the frequency domain for each BWP. This includes the PUCCH, PUSCH and SRS instances

The waveform generator function returns the time domain waveform and a struct array `bwpsset`, which contains the following fields:

- The resource grid corresponding to this BWP
- The resource grid of the overall bandwidth containing the channels and signals in this BWP
- An info structure with information corresponding to the BWP. The contents of this info structure for the first BWP are shown below:

```
disp('Information associated to BWP 1:')
disp(bwpsset(1).Info)
```

```
Information associated to BWP 1:
      Nfft: 4096
      SampleRate: 61440000
      CyclicPrefixLengths: [1x14 double]
```



```
SymbolLengths: [1x14 double]
Windowing: 144
SymbolPhases: [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
SymbolsPerSlot: 14
SlotsPerSubframe: 1
SlotsPerFrame: 10
NSubcarriers: 3240
SubcarrierSpacing: 15
SymbolsPerSubframe: 14
SamplesPerSubframe: 61440
SubframePeriod: 1.0000e-03
k0: 0
SamplingRate: 61440000
```

Note that the generated resource grid is a 3D matrix where the different planes represent the antenna ports. For the different physical channels and signals the lowest port is mapped to the first plane of the grid.

## See Also

### Functions

nrPUCCH0 | nrPUCCH1 | nrPUCCH2 | nrPUCCH3 | nrPUCCH4 | nrPUSCH | nrULSCH

## More About

- “5G NR Downlink Carrier Waveform Generation” on page 1-2

## NR PUSCH Resource Allocation and DM-RS and PT-RS Reference Signals

This example shows the time-frequency aspects of the new radio (NR) physical uplink shared channel (PUSCH), the associated demodulation reference signal (DM-RS), and phase tracking reference signal (PT-RS). The example shows how PUSCH resource allocation affects the time-frequency structure of DM-RS and PT-RS.

### Introduction

In 5G NR, PUSCH is the physical uplink channel that carries user data. DM-RS and PT-RS are the reference signals associated with PUSCH. DM-RS is used for channel estimation as part of coherent demodulation of PUSCH. To compensate for the common phase error (CPE), 3GPP 5G NR introduced PT-RS. Phase noise produced in local oscillators introduces a significant degradation at mmWave frequencies. It produces CPE and inter-carrier interference (ICI). CPE leads to an identical rotation of a received symbol in each subcarrier. ICI leads to a loss of orthogonality between the subcarriers. PT-RS is used mainly to estimate and minimize the effect of CPE on system performance.

The time-frequency structure of reference signals depends on the type of waveform configured for PUSCH, as defined in TS 38.211 Sections 6.4.1.1 and 6.4.1.2 [1] on page 2-0 . When transform precoding is disabled, the waveform configured is cyclic-prefix-orthogonal frequency division multiplexing (CP-OFDM). When transform precoding is enabled, the waveform configured is discrete-fourier-transform-spread orthogonal frequency division multiplexing (DFT-s-OFDM).

The 5G Toolbox™ provides the functions for physical (PHY) layer modeling with varying levels of granularity. The levels of granularity range from PHY channel level functions that perform the transport and physical channel processing to individual channel processing stage functions performing cyclic redundancy check (CRC) coding, code block segmentation, low density parity check (LDPC) channel coding, and so on. The toolbox provides reference signals functionality associated with the PUSCH as functions `nrPUSCHDMRS`, `nrPUSCHDMRSIndices`, `nrPUSCHPTRS`, and `nrPUSCHPTRSIndices`.

### PUSCH

PUSCH is the physical channel that carries the user data. The resources allocated for PUSCH are within the bandwidth part (BWP) of the carrier, as defined in TS 38.214 Section 6.1.2 [2] on page 2-0 . The resources in time domain for PUSCH transmission are scheduled by downlink control information (DCI) in the field *Time domain resource assignment*. This field indicates the slot offset  $K_0$ , starting symbol  $S$ , the allocation length  $L$ , and the mapping type of PUSCH. The valid combinations of  $S$  and  $L$  are shown in Table 1.

PUSCH Mapping Type	Normal Cyclic Prefix			Extended Cyclic Prefix		
	$S$	$L$	$S+L$	$S$	$L$	$S+L$
Type A	0	{4,...,14}	{4,...,14}	0	{4,...,12}	{4,...,12}
Type B	{0,...,13}	{1,...,14}	{1,...,14}	{0,...,11}	{1,...,12}	{1,...,12}

Table 1: Valid  $S$  and  $L$  Combinations

The resources in the frequency domain for PUSCH transmission are scheduled by a DCI in the field *Frequency domain resource assignment*. This field indicates whether the resource allocation of resource blocks (RBs) is contiguous or noncontiguous, based on the allocation type. The RBs allocated are within the BWP.

The 5G Toolbox™ provides the `nrCarrierConfig` and `nrPUSCHConfig` objects to set the parameters related to the PUSCH within the BWP.

```
% Setup the carrier with 15 kHz subcarrier spacing and 10 MHz bandwidth
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 15;
carrier.CyclicPrefix = 'normal';
carrier.NSizeGrid = 52;
carrier.NStartGrid = 0;

% Configure the physical uplink shared channel parameters
pusch = nrPUSCHConfig;
pusch.NSizeBWP = []; % Empty implies that the value is equal to NSizeGrid
pusch.NStartBWP = []; % Empty implies that the value is equal to NStartGrid
pusch.PRBSets = 0:25; % Allocate half of the carrier bandwidth
pusch.SymbolAllocation = [0 14]; % Symbol allocation [S L]
pusch.MappingType = 'A'; % PUSCH mapping type ('A' or 'B')
pusch.TransmissionScheme = 'nonCodebook'; % ('codebook' or 'nonCodebook')
% The following parameters are applicable when TransmissionScheme is set
% to 'codebook'
pusch.NumAntennaPorts = 4;
pusch.TPMI = 0;
```

### DM-RS for CP-OFDM

DM-RS is used to estimate the radio channel. DM-RS is present only in the RBs scheduled for PUSCH. The DM-RS structure is designed to support different deployment scenarios and use cases.

### Parameters That Control Time Resources

The parameters that control the time resources of DM-RS are:

- PUSCH symbol allocation
- Mapping type
- Intra-slot frequency hopping
- DM-RS type A position
- DM-RS length
- DM-RS additional position

Symbol allocation of PUSCH indicates the OFDM symbol locations allocated for the PUSCH transmission in a slot. The mapping type indicates the first DM-RS OFDM symbol location and the duration of OFDM symbols ( $l_d$ ). For mapping type A,  $l_d$  is the duration between the first OFDM symbol of the slot and the last OFDM symbol of the allocated PUSCH resources. For mapping type B,  $l_d$  is the duration of the allocated PUSCH resources. When intra-slot frequency hopping is enabled,  $l_d$  is the duration per hop. The DM-RS symbols are present in each hop when intra-slot frequency hopping is enabled. When intra-slot frequency hopping is enabled, DM-RS is single-symbol with the maximum number of additional positions either 0 or 1. The DM-RS symbol locations is given by TS 38.211 Tables 6.4.1.1.3-3, 6.4.1.1.3-4, and 6.4.1.1.3-6. Figure 1 shows the DM-RS symbol locations for PUSCH occupying 14 symbols with PUSCH mapping type A, intra-slot frequency hopping enabled,

and number of DM-RS additional positions as 1. The figure shows DM-RS is present in each hop. The locations of DM-RS symbols in each hop depends on the number of OFDM symbols allocated for PUSCH in each hop.

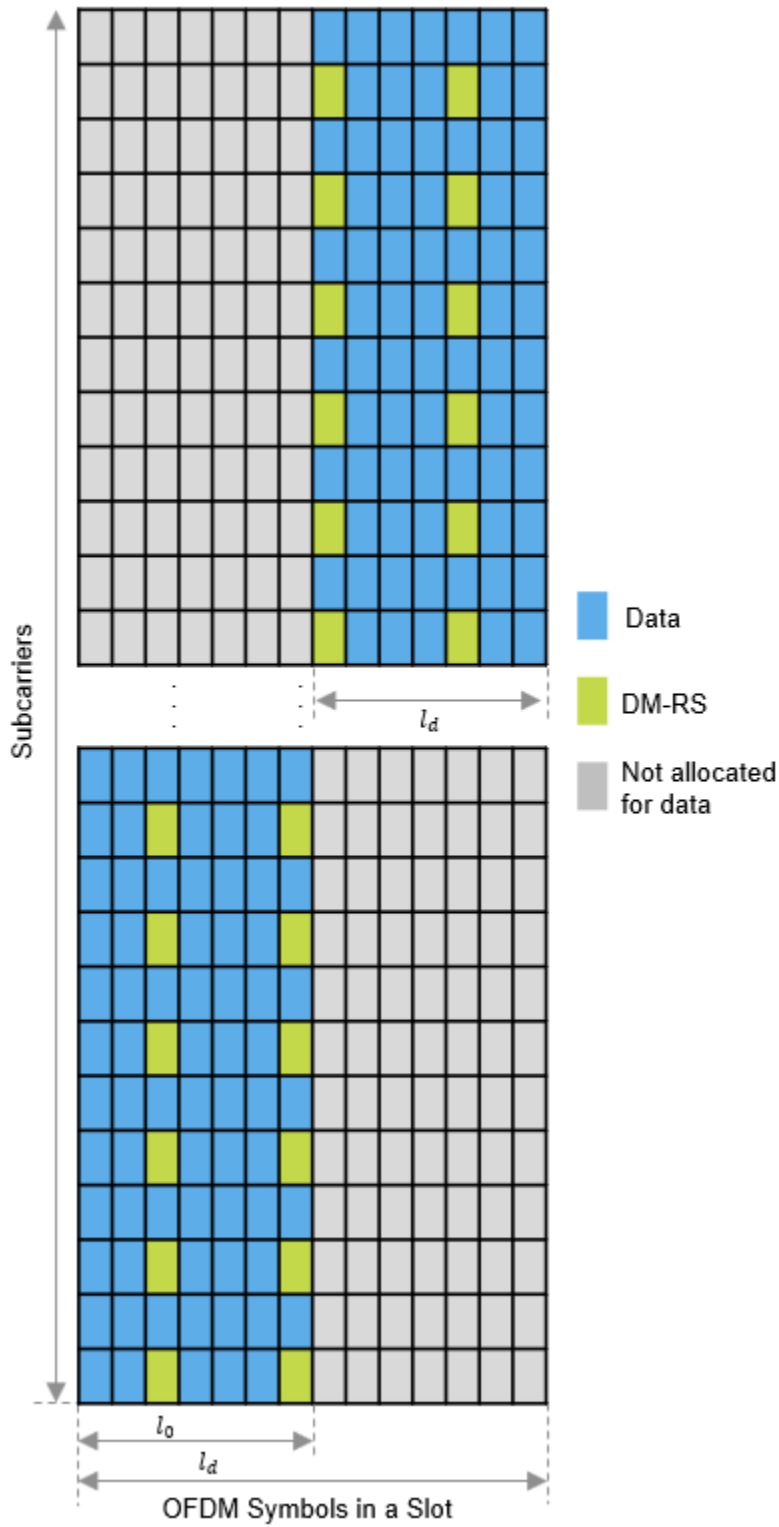


Figure 1: DM-RS Symbol Locations Based on Intra-Slot Frequency Hopping

For details on other DM-RS parameters, see “NR PDSCH Resource Allocation and DM-RS and PT-RS Reference Signals” on page 1-14.

```
% Assign intra-slot frequency hopping for PUSCH
pusch.FrequencyHopping = 'intraSlot'; % 'neither', 'intraSlot', 'interSlot'
pusch.SecondHopStartPRB = 26;

% Set the parameters that control the time resources of DM-RS
pusch.DMRS.DMRSTypeAPosition = 2; % 2 or 3
pusch.DMRS.DMRSLength = 1; % 1 or 2 (single-symbol or double-symbol)
pusch.DMRS.DMRSAdditionalPosition = 1; % 0...3 (Number of additional DM-RS positions)
```

### Parameters That Control Frequency Resources

The parameters that control the frequency resources of DM-RS are:

- DM-RS configuration type
- DM-RS antenna ports

The configuration type indicates the frequency density of DM-RS and is signaled by the RRC message *dms-Type*. Configuration type 1 defines six subcarriers per physical resource block (PRB) per antenna port, comprising alternate subcarriers. Configuration type 2 defines four subcarriers per PRB per antenna port, consisting of two groups of two consecutive subcarriers. Different delta shifts are applied to the sets of subcarriers used, depending on the associated antenna port or code division multiplexing (CDM) group. For configuration type 1, there are two possible CDM groups/shifts across eight possible antenna ports ( $p=0\dots7$ ). For configuration type 2, there are three possible CDM groups/shifts across twelve antenna ports ( $p=0\dots11$ ). For more details, see “NR PDSCH Resource Allocation and DM-RS and PT-RS Reference Signals” on page 1-14.

In the case of codebook-based PUSCH processing, the union of DM-RS subcarrier locations present in each layer are projected to all the antenna ports.

```
% Set the parameters that control the frequency resources of DM-RS
pusch.DMRS.DMRSConfigurationType = 1; % 1 or 2
pusch.DMRS.DMRSPortSet = 0;

% The read-only properties DeltaShifts and DMRSSubcarrierLocations of DMRS
% property of pusch object provides the values of delta shift(s) and DM-RS
% subcarrier locations in an RB for each antenna port configured.
pusch.DMRS.DeltaShifts

ans = 0

pusch.DMRS.DMRSSubcarrierLocations

ans = 6x1

    0
    2
    4
    6
    8
   10
```

## Sequence Generation

The pseudorandom sequence used for DM-RS is  $2^{31} - 1$  length gold sequence. The sequence is generated across all the common resource blocks (CRBs) and is transmitted only in the RBs allocated for data because the sequence is not required to estimate the channel outside the frequency region in which data is not transmitted. Generating the reference signal sequence across all the CRBs ensures that the same underlying pseudorandom sequence is used for multiple UEs on overlapping time-frequency resources in the case of a multi-user MIMO. The parameters that control the sequence generation are:

- DM-RS scrambling identity ( $N_{ID}^{n_{SCID}}$ )
- DM-RS scrambling initialization ( $n_{SCID}$ )
- Number of OFDM symbols in a slot
- Slot number in a radio frame
- DM-RS symbol locations
- PRBs allocation

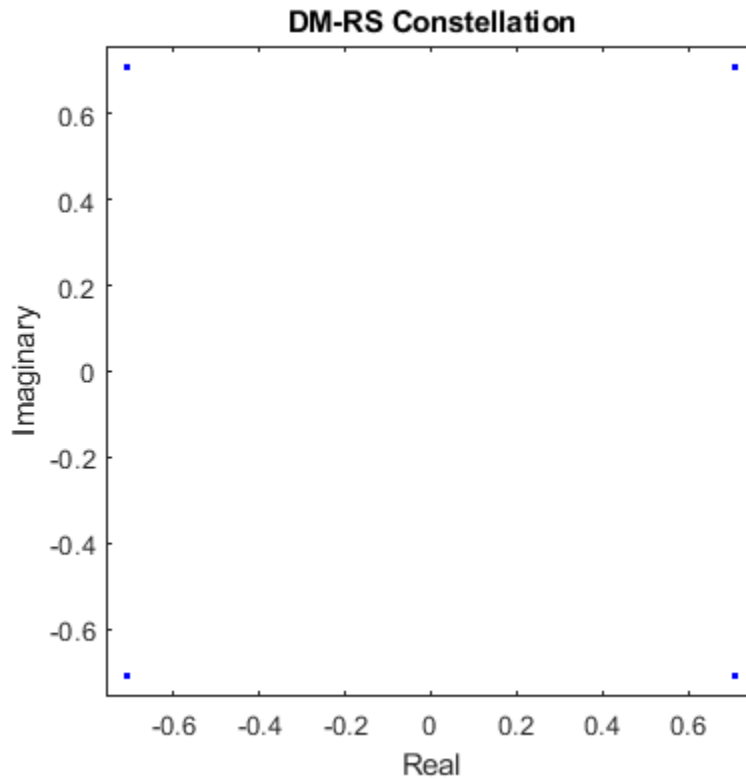
The CyclicPrefix property of the carrier object controls the number of OFDM symbols in a slot. The NSlot property of the carrier object controls the slot number.

In the case of codebook-based PUSCH processing, the sequence is multiplied with a precoder matrix, which depends on the number of layers, number of antenna ports, and the transmitted precoder matrix indicator (TPMI).

```
% Set the parameters that only control the DM-RS sequence generation
pusch.DMRS.NIDNSCID = 1; % Use empty to set it to NCellID of the carrier
pusch.DMRS.NSCID = 0;    % 0 or 1
```

```
% Generate DM-RS symbols
pusch.NumLayers = numel(pusch.DMRS.DMRSPortSet);
dmrsSymbols = nrPUSCHDMRS(carrier,pusch);
```

```
% Plot the constellation
scatterplot(dmrsSymbols)
title('DM-RS Constellation')
xlabel('Real')
ylabel('Imaginary')
```



```
% The read-only properties TimeWeights and FrequencyWeights of DMRS
% property of pusch object provides the values of time and frequency
% weights applied to the DM-RS symbols.
```

```
pusch.DMRS.TimeWeights
```

```
ans = 2x1
```

```
1
1
```

```
pusch.DMRS.FrequencyWeights
```

```
ans = 2x1
```

```
1
1
```

```
% Generate DM-RS indices
```

```
dmrsIndices = nrPUSCHDMRSIndices(carrier, pusch);
```

```
% Map the DM-RS symbols to the grid with the help of DM-RS indices
```

```
if strcmpi(pusch.TransmissionScheme, 'codebook')
```

```
    nports = pusch.NumAntennaPorts;
```

```
else
```

```
    nports = pusch.NumLayers;
```

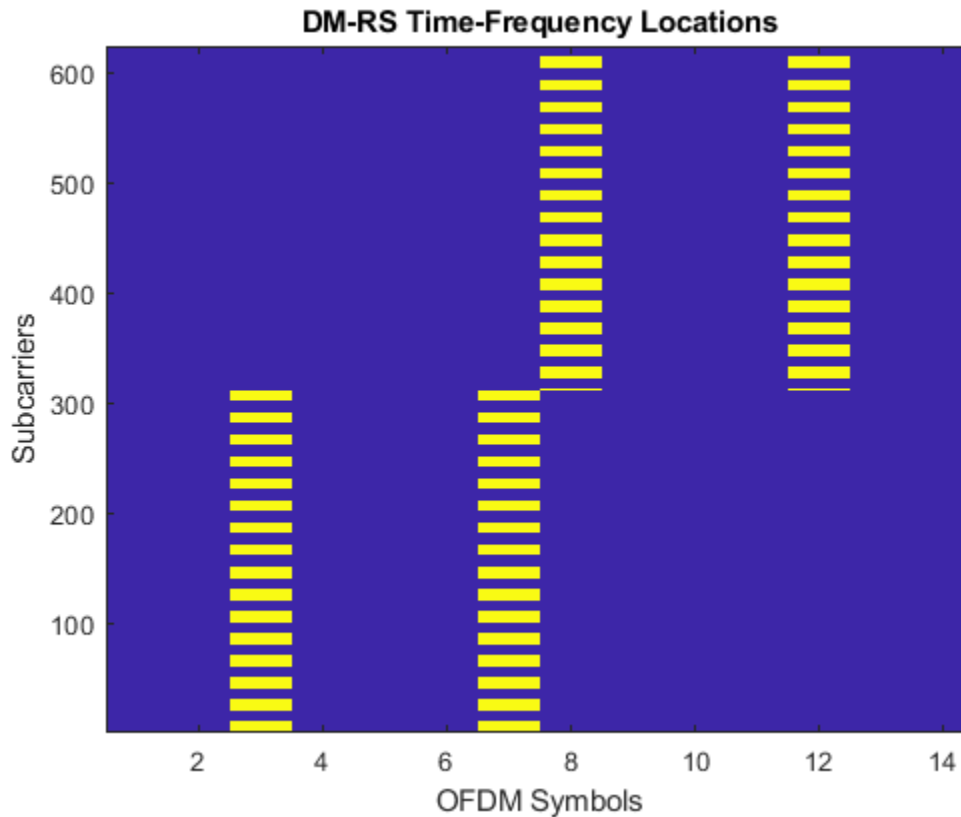
```
end
```



```

grid = zeros([12*carrier.NSizeGrid carrier.SymbolsPerSlot nports]);
grid(dmrsIndices) = dmrsSymbols;
figure
imagesc(abs(grid(:,:,1)));
axis xy;
xlabel('OFDM Symbols');
ylabel('Subcarriers');
title('DM-RS Time-Frequency Locations');

```



### PT-RS for CP-OFDM

PT-RS is the phase tracking reference signal. PT-RS is used mainly to estimate and minimize the effect of CPE on system performance. Due to the phase noise properties, the PT-RS signal has low density in the frequency domain and high density in the time domain. PT-RS always occurs in combination with DM-RS and only when the network has configured PT-RS to be present.

### Parameters That Control Time Resources

PT-RS is configured through the higher layer parameter *DMRS-UplinkConfig* for uplink. The parameters that control the time resources of PT-RS are:

- DM-RS symbol locations
- Time density of PT-RS ( $L_{PT-RS}$ )

$L_{PT-RS}$  depends on the scheduled modulation and coding scheme. The value must be one of {1, 2, 4}. For the parameters that control DM-RS symbol locations, refer to Parameters that Control DM-RS Time Resources (CP-OFDM) on page 2-0 .

```
% Set the EnablePTRS property in pusch to 1
pusch.EnablePTRS = 1;

% Set the parameters that control the time resources of PT-RS
pusch.PTRS.TimeDensity = 2;
```

### Parameters That Control Frequency Resources

PT-RS occupies only one subcarrier in an RB for one OFDM symbol. The parameters that control the frequency resources of PT-RS are:

- PRB allocation
- DM-RS configuration type
- Frequency density of PT-RS ( $K_{PT-RS}$ )
- Radio network temporary identifier ( $n_{RNTI}$ )
- Resource element offset
- PT-RS antenna ports

$K_{PT-RS}$  depends on the scheduled bandwidth. The value is either 2 or 4. The value indicates whether PT-RS is present in every two RBs or every four RBs.

For more details, see “NR PDSCH Resource Allocation and DM-RS and PT-RS Reference Signals” on page 1-14.

```
% Set the parameters that control PT-RS subcarrier locations
pusch.RNTI = 1;
pusch.DMRS.DMRSConfigurationType = 1;
pusch.DMRS.DMRSPortSet = 0;
% Set the PT-RS parameters
pusch.PTRS.FrequencyDensity = 2; % 2 or 4
pusch.PTRS.REOffset = '10'; % '00', '01', '10', '11'
pusch.PTRS.PTRSPortSet = min(pusch.DMRS.DMRSPortSet);
```

### Sequence Generation

The sequence used for generating PT-RS is the same pseudorandom sequence used for the DM-RS sequence generation. In the absence of intra-slot frequency hopping, the values of PT-RS sequence depend on the first DM-RS symbol position. In the presence of intra-slot frequency hopping, the values of PT-RS sequence depend on first DM-RS symbol positions in each hop. For more details, refer the section DM-RS Sequence Generation (CP-OFDM) on page 2-0 .

In the case of codebook-based PUSCH processing, the sequence is multiplied with a precoder matrix, which depends on the number of layers, number of antenna ports, and the transmitted precoder matrix indicator (TPMI).

```
% Set the parameters that control the PT-RS sequence generation
pusch.DMRS.NIDNSCID = 1; % Use empty to set it to NCellID of the carrier
pusch.DMRS.NSCID = 0; % 0 or 1
```

Generate the resource element (RE) indices of PUSCH, DM-RS, and PT-RS. Also, generate DM-RS and PT-RS symbols.

```
% Control the resource elements available for data in DM-RS OFDM symbol
% locations
```

```

pusch.DMRS.NumCDMGroupsWithoutData = 1;

% PUSCH, DM-RS and PT-RS indices
pusch.NumLayers = numel(pusch.DMRS.DMRSPortSet);
[puschIndices, puschInfo] = nrPUSCHIndices(carrier, pusch);
dmrsIndices = nrPUSCHDMRSIndices(carrier, pusch);
ptrsIndices = nrPUSCHPTRSIndices(carrier, pusch);

% DM-RS and PT-RS symbols
dmrsSymbols = nrPUSCHDMRS(carrier, pusch);
ptrsSymbols = nrPUSCHPTRS(carrier, pusch);

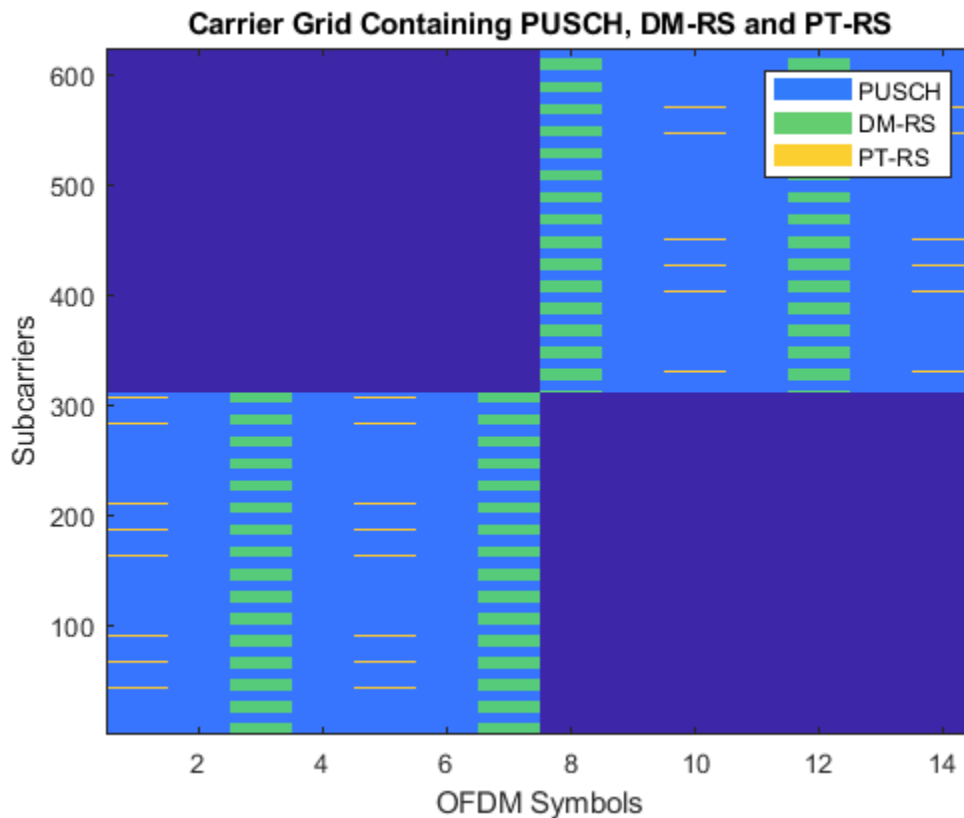
```

Map PUSCH, DM-RS, and PT-RS RE indices to the grid with scaled values to visualize the respective locations on the grid.

```

chpLevel = struct;
chpLevel.PUSCH = 0.4;
chpLevel.DMRS = 1;
chpLevel.PTRS = 1.4;
gridCPOFDM = complex(zeros([carrier.NSizeGrid*12 carrier.SymbolsPerSlot nports]));
gridCPOFDM(puschIndices) = chpLevel.PUSCH;
dmrsFactor = chpLevel.DMRS*(1/(max(abs(dmrsSymbols))));
gridCPOFDM(dmrsIndices) = dmrsFactor*dmrsSymbols;
ptrsFactor = chpLevel.PTRS*(1/(max(abs(ptrsSymbols))));
gridCPOFDM(ptrsIndices) = ptrsFactor*ptrsSymbols;
plotGrid(gridCPOFDM, 1, chpLevel)

```



In the preceding figure, PT-RS is located from the start of the OFDM symbol in the physical uplink shared channel allocation. The symbols are present at every  $L_{\text{PT-RS}}$  hop interval from each other or from DM-RS symbols. The difference in consecutive subcarrier locations of PT-RS is 24, which is the number of subcarriers in an RB (12) times the frequency density of PT-RS (2).

### **DM-RS for DFT-s-OFDM**

DFT-s-OFDM supports only single layer transmission and is primarily used for low coverage scenarios. The time-frequency resources of DM-RS in DFT-s-OFDM are structured in a way to achieve low cubic metric and high power amplifier efficiency. The transmission of a reference signal frequency multiplexed with other uplink data transmissions highly impacts the power amplifier efficiency due to the increased cubic metric. The reference signals are time-multiplexed with uplink transmissions, thereby blocking all the resource elements for data transmission in the OFDM symbols carrying DM-RS.

### **Parameters That Control Time Resources**

The parameters that control the time resources of DM-RS in DFT-s-OFDM are:

- PUSCH symbol allocation
- Mapping type
- Intra-slot frequency hopping
- DM-RS type A position
- DM-RS length
- DM-RS additional position

These parameters are the same parameters that control the time resources of DM-RS in CP-OFDM. For more details, refer to Parameters that Control DM-RS Time Resources (CP-OFDM) on page 2-0 .

```
% Set the TransformPrecoding property in pusch to 1
pusch.TransformPrecoding = 1;
```

```
% Parameters that control the time resources
pusch.DMRS.DMRSTypeAPosition = 2;
pusch.DMRS.DMRSLength = 1;
pusch.DMRS.DMRSAdditionalPosition = 0;
```

### **Parameters That Control Frequency Resources**

The parameters that control the frequency resources of DM-RS in DFT-s-OFDM are:

- DM-RS configuration type
- DM-RS antenna port

These two parameters are the same as the parameters of CP-OFDM. The DM-RS configuration type is always set to 1. The DM-RS antenna port is nominally a scalar with value 0.

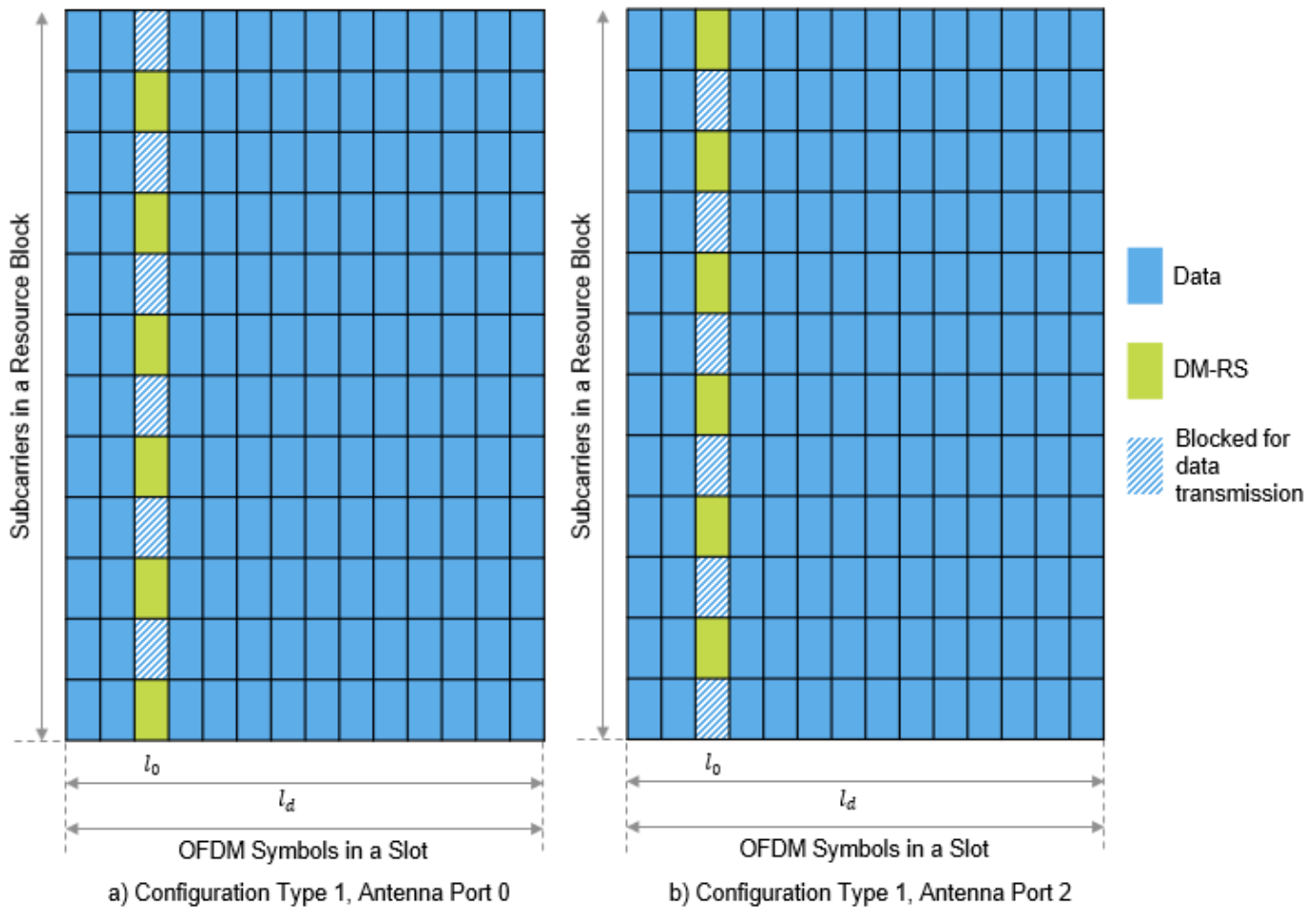


Figure 2: DM-RS Subcarrier Locations for DFT-s-OFDM

There is no need to support multi-user MIMO situations because DFT-s-OFDM is for coverage-limited scenarios. With no MIMO situations, the reference signal is generated only for the transmitted PRBs rather than CRBs as in OFDM. Due to the single layer and single configuration type allowed in DFT-s-OFDM, the number of subcarrier locations used for DM-RS in an RB is constant. Figure 2 illustrates the DM-RS subcarrier locations in DFT-s-OFDM for mapping type A with OFDM symbols allocated for PUSCH spanning over the complete slot.

```
% Set the DM-RS antenna port
pusch.DMRS.DMRSPortSet = 0;
```

### Sequence Generation

The DM-RS sequence is the ZadoffChu sequence in DFT-s-OFDM. The orthogonal sequences are generated with different cyclic shifts for a group number and sequence number. The parameters that control the sequence generation are:

- PRB allocation
- Group hopping
- Sequence hopping

- DM-RS scrambling identity ( $N_{ID}^{RS}$ )
- DM-RS symbol locations

```

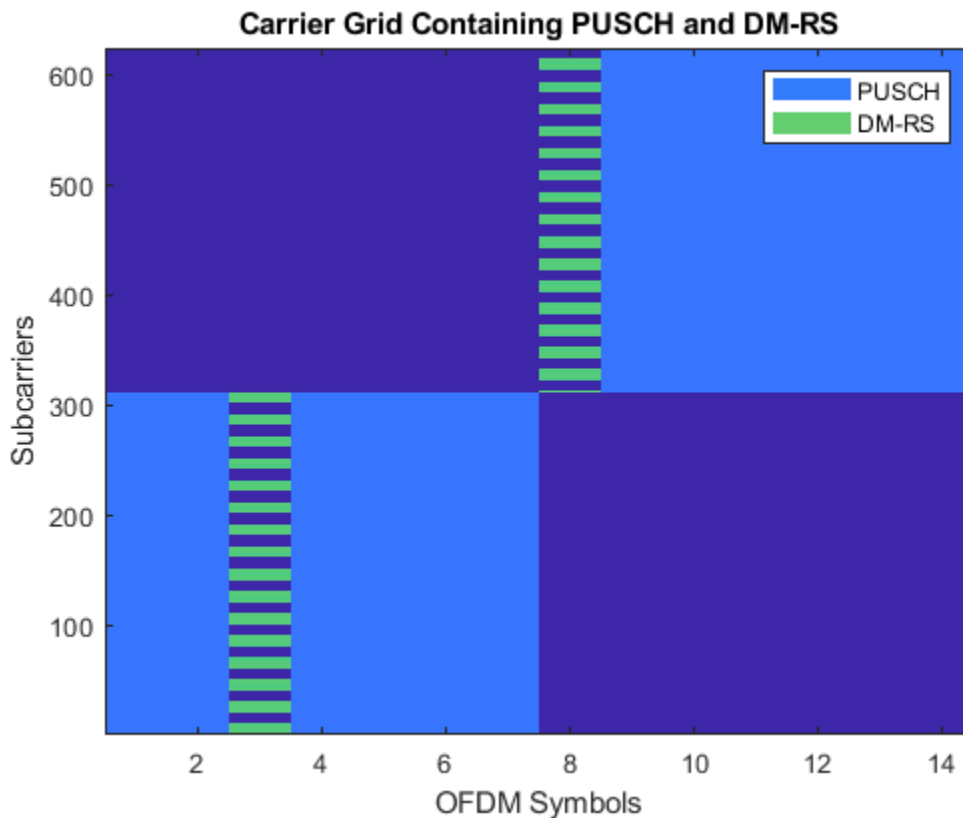
% Parameters that control the sequence generation
pusch.DMRS.SequenceHopping = 0; % Sequence hopping (0 or 1)
pusch.DMRS.GroupHopping = 1;    % Group hopping (0 or 1)
pusch.DMRS.NRSID = 1;           % Use empty to set it to NCellID of carrier

% Generate the DM-RS symbols and indices
pusch.NumLayers = numel(pusch.DMRS.DMRSPortSet);
dmrsSymbols = nrPUSCHDMRS(carrier,pusch);
dmrsIndices = nrPUSCHDMRSIndices(carrier,pusch);
dmrsFactor = chpLevel.DMRS*(1/(max(abs(dmrsSymbols))));
% Map DM-RS onto the grid
grid = complex(zeros([12*carrier.NSizeGrid carrier.SymbolsPerSlot nports]));
grid(dmrsIndices) = dmrsFactor*dmrsSymbols;

% Generate PUSCH indices and map onto the grid
puschIndices = nrPUSCHIndices(carrier,pusch);
grid(puschIndices) = chpLevel.PUSCH;

% Plot the grid
titleText = 'Carrier Grid Containing PUSCH and DM-RS';
plotGrid(grid,1,struct('PUSCH',chpLevel.PUSCH,'DMRS',chpLevel.DMRS),titleText,{'PUSCH','DM-RS'})

```



The subcarrier locations in the OFDM symbols occupying DM-RS are not allocated for PUSCH.

## PT-RS for DFT-s-OFDM

PT-RS in DFT-s-OFDM is inserted with data in the transform precoding stage.

### Parameters That Control Time Resources

The parameters that control the time resources of PT-RS in DFT-s-OFDM are same as the parameters that control the time resources of PT-RS in CP-OFDM. The value of  $L_{PT-RS}$  is either 1 or 2 in DFT-s-OFDM. For more details, refer to Parameters that Control PT-RS Time Resources (CP-OFDM) on page 2-0 .

```
% Generate a grid with shared channel allocation for an RB in a single slot
% with complete symbol allocation of 14 symbols for a single layer
```

```
% Set the carrier resource grid with one RB
carrier.NSizeGrid = 1;
```

```
% Configure PUSCH with DFT-s-OFDM and no frequency hopping
pusch.TransformPrecoding = 1;
pusch.FrequencyHopping = 'neither';
```

```
% Set the parameter that control PT-RS time resources
pusch.EnablePTRS = 1;
pusch.PTRS.TimeDensity = 2;
```

### Parameters That Control Frequency Resources

The PT-RS pattern in the frequency domain is quite different from CP-OFDM. The PT-RS samples are inserted as chunks or groups ( $N_{\text{group}}^{\text{PT-RS}}$ ). Each group consists of a finite number of samples ( $N_{\text{samp}}^{\text{group}}$ ) in the scheduled bandwidth for each OFDM symbol where PT-RS is present.

The parameters that control the frequency resources of PT-RS in DFT-s-OFDM are:

- PRB allocation
- Number of PT-RS samples in a group ( $N_{\text{samp}}^{\text{group}}$ )
- Number of PT-RS groups ( $N_{\text{group}}^{\text{PT-RS}}$ )

The valid combinations of PT-RS sample density ( $[N_{\text{samp}}^{\text{group}} N_{\text{group}}^{\text{PT-RS}}]$ ) are {[2 2], [2 4], [4 2], [4 4], [4 8]}. The number of PT-RS samples in an OFDM symbol is fixed in DFT-s-OFDM, based on the number of PT-RS samples in all the PT-RS groups. This number is different from CP-OFDM in which the number of PT-RS samples increase based on the number of RBs in PUSCH.

Figure 3 shows the subcarrier locations of PT-RS symbols for an RB with the number of PT-RS samples set to 2 and the number of PT-RS groups set to 2 for an OFDM symbol carrying PT-RS.

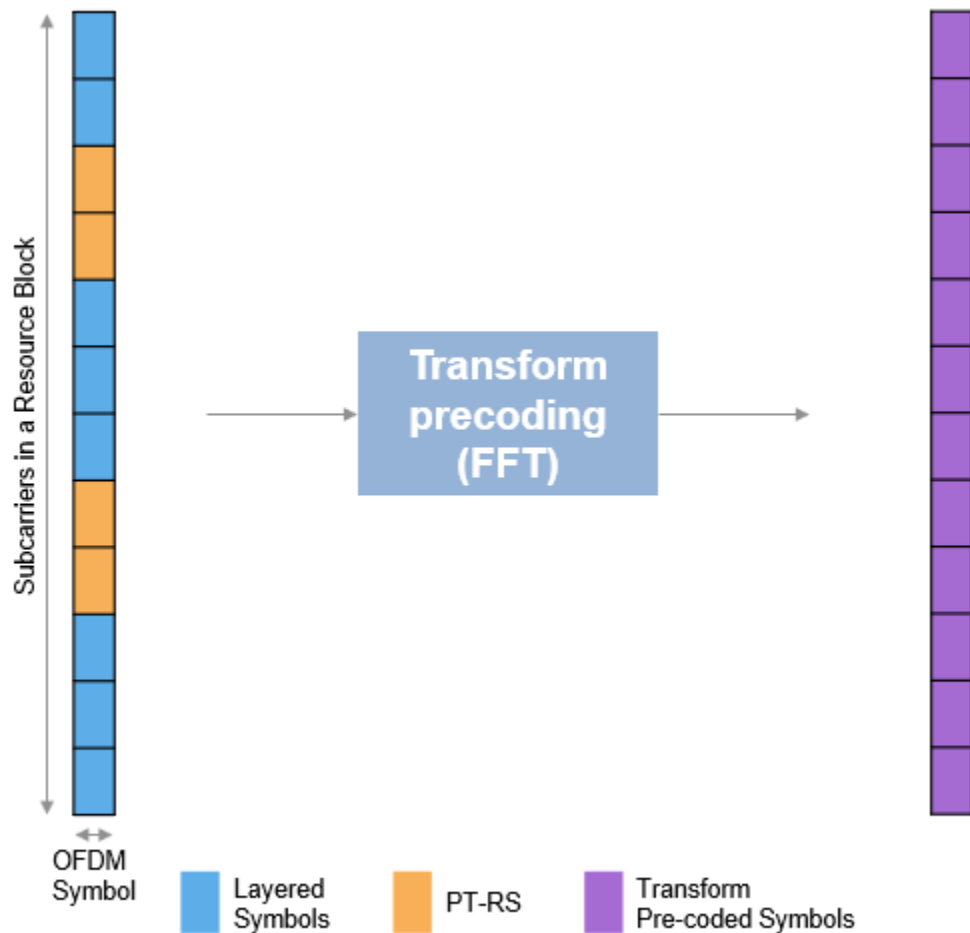


Figure 3: PT-RS Insertion in DFT-s-OFDM

PT-RS sample density [2 2] implies that there are two PT-RS groups in a scheduled bandwidth with two symbols each.

PT-RS is inserted with layered symbols at the input of transform precoding. After transform precoding, both layered symbols and PT-RS are treated as data. Therefore, the PT-RS is not visible in the grid directly.

```
% Set the parameters that control PT-RS frequency resources
pusch.PRBSets = 0:carrier.NSizeGrid-1;
pusch.PTRS.NumPTRSSamples = 2; % 2, 4
pusch.PTRS.NumPTRSGroups = 2; % 2, 4, 8
```

### Sequence Generation

The PT-RS sequence in DFT-s-OFDM is a modified  $\pi/2$ -BPSK sequence. The parameters that control the sequence generation are:

- Starting OFDM symbol of PUSCH allocation
- Number of OFDM symbols in a slot



- Slot number in a radio frame
- PT-RS scrambling identity ( $N_{ID}$ )
- PT-RS subcarrier locations

```
% Set the parameters that control PT-RS sequence generation
```

```
pusch.DMRS.NRSID = 1;
```

```
pusch.PTRS.NID = 10; % Use empty to set it to NRSID of DMRS configuration
```

Generate PUSCH and PT-RS RE indices.

```
% PUSCH, and PT-RS indices
```

```
[puschIndices, puschInfoDFTsOFDM] = nrPUSCHIndices(carrier, pusch);
```

```
ptrsIndices = nrPUSCHPTRSIndices(carrier, pusch);
```

Set PUSCH and PT-RS resource elements to constant values.

```
% Insert PT-RS along with the PUSCH data
```

```
GdPTRS = size(reshape(ptrsIndices, [], pusch.NumLayers), 1);
```

```
dataWithPTRS = chpLevel.PUSCH*ones(puschInfoDFTsOFDM.Gd+GdPTRS, 1);
```

```
dataWithPTRS(ptrsIndices(:, 1)) = chpLevel.PTRS;
```

Plot PT-RS projections onto the grid.

```
gridDFTsOFDM = zeros(numel(pusch.PRBSets)*12, carrier.SymbolsPerSlot);
```

```
% Map the grid with data and reference signals
```

```
gridDFTsOFDM(:, puschInfoDFTsOFDM.DMRSSymbolSet+1) = chpLevel.DMRS;
```

```
gridDFTsOFDM(~(gridDFTsOFDM==chpLevel.DMRS)) = dataWithPTRS;
```

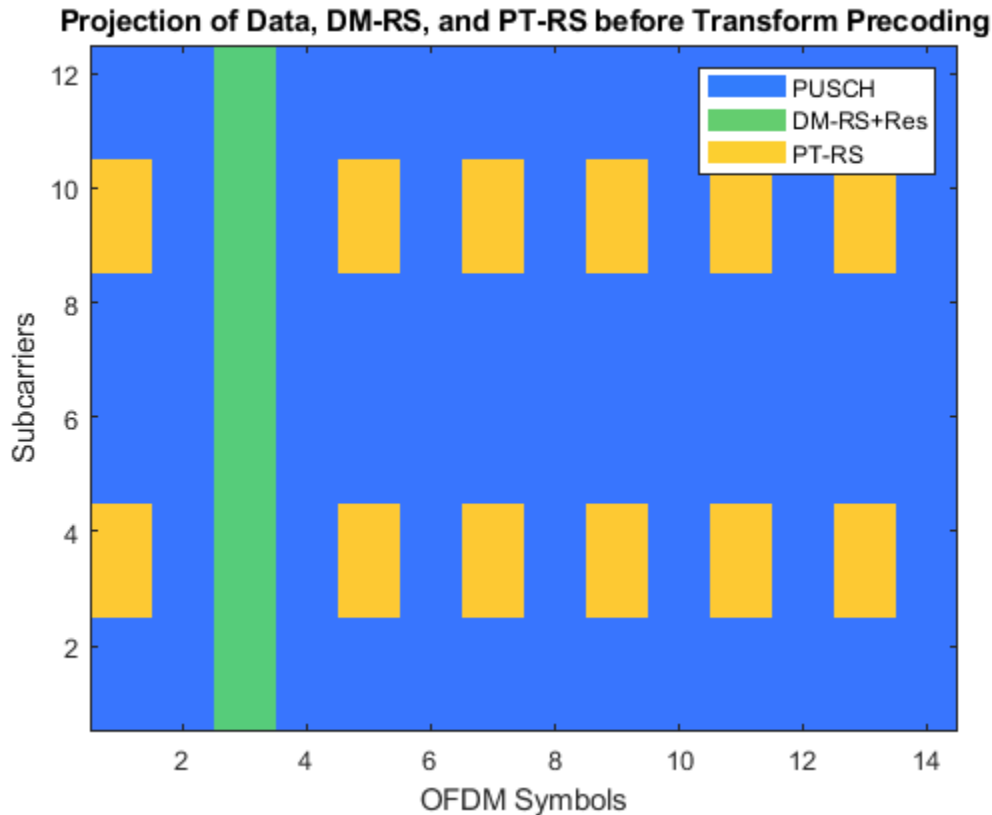
```
% Plot the projections of data, DM-RS and PT-RS on grid before transform
```

```
% precoding
```

```
fNames = {'PUSCH', 'DM-RS+Res', 'PT-RS'};
```

```
titleText = 'Projection of Data, DM-RS, and PT-RS before Transform Precoding';
```

```
plotGrid(gridDFTsOFDM, 1, chpLevel, titleText, fNames)
```



### Further Exploration

You can try changing the parameters that affect the time and frequency resources of reference signals and observe the variations in the RE positions for the respective signals.

Try changing the number of antenna ports configured for DM-RS and PT-RS, then observe the variations of reference signals and data across the ports. For example, try to configure DM-RS for two antenna ports 0 and 2, configuration type 1, and PT-RS for antenna port 0. Generate the PUSCH indices, DM-RS signal (indices and symbols), and PT-RS signal (indices and symbols). Map them to a grid and visualize the grid for both the ports.

Try performing channel estimation and phase tracking using the PT-RS symbols and indices. Compute the throughput by following the steps outlined in “NR PUSCH Throughput” on page 2-34.

This example shows how to generate the DM-RS and PT-RS sequences and how to map the sequences to the OFDM carrier resource grid. It highlights the properties that control the time-frequency structure of reference signals for different waveforms. For example, the time-frequency pattern for reference signals in CP-OFDM and DFT-s-OFDM and the variation in the sequences generated for reference signals in different waveforms.

### References

- 1 3GPP TS 38.211. "NR; Physical channels and modulation (Release 15)." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
- 2 3GPP TS 38.214. "NR; Physical layer procedures for data (Release 15)." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

- 3 3GPP TS 38.212. "NR; Multiplexing and channel coding (Release 15)." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network.*

### Local Functions

```
function plotGrid(grid,nLayer,chnLevel,titleText,names)
% plotGrid Display resource grid GRID of the layer number NLAYER with the
% legend containing physical channel and associated reference signals, at
% different power levels CHPLEVEL with title TITLETEXT. Legend is created
% using a cell array of character vectors NAMES.

    if nargin < 4
        titleText = 'Carrier Grid Containing PUSCH, DM-RS and PT-RS';
    end

    if nargin < 5
        names = {'PUSCH', 'DM-RS', 'PT-RS'};
    end

    map = parula(64);
    cscaling = 40;
    im = image(1:size(grid,2),1:size(grid,1),cscaling*abs(grid(:,:,nLayer)));
    colormap(im.Parent,map);

    % Add legend to the image
    chpval = struct2cell(chnLevel);
    clevels = cscaling*[chpval{:}];
    N = length(clevels);
    L = line(ones(N),ones(N), 'LineWidth',8); % Generate lines
    % Index the color map and associated the selected colors with the lines
    set(L,{'color'},mat2cell(map( min(1+cleveles,length(map) ),:),ones(1,N),3)); % Set the colors
    % Create legend
    legend(names{:});
    axis xy;
    ylabel('Subcarriers');
    xlabel('OFDM Symbols');
    title(titleText);
end
```

### See Also

#### Functions

nrPUSCHDMRS | nrPUSCHDMRSIndices | nrPUSCHIndices | nrPUSCHPTRS |  
nrPUSCHPTRSIndices

#### Objects

nrCarrierConfig | nrPUSCHConfig

### Related Examples

- “NR PDSCH Resource Allocation and DM-RS and PT-RS Reference Signals” on page 1-14

## NR PUSCH Throughput

This example shows how to measure the physical uplink shared channel (PUSCH) throughput of a 5G New Radio (NR) link, as defined by the 3GPP NR standard. The example implements PUSCH and uplink transport channel (UL-SCH). The transmitter model includes PUSCH demodulation reference symbols (DM-RS). The example supports both clustered delay line (CDL) and tapped delay line (TDL) propagation channels. You can perform perfect or practical synchronization and channel estimation. To reduce the total simulation time, you can execute the SNR points in the SNR loop in parallel by using the Parallel Computing Toolbox™.

### Introduction

This example measures the PUSCH throughput of a 5G link, as defined by the 3GPP NR standard [ 1 ], [ 2 ], [ 3 ], [ 4 ].

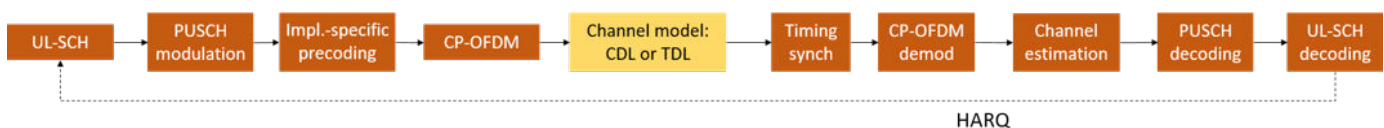
The following 5G NR features are modeled:

- UL-SCH transport channel coding
- PUSCH and PUSCH DM-RS generation
- Variable subcarrier spacing and frame numerologies ( $2^n * 15\text{kHz}$ )
- Normal and extended cyclic prefix
- TDL and CDL propagation channel models

Other features of the simulation are:

- Codebook and non-codebook based PUSCH transmission schemes
- Optional PUSCH transform precoding
- Slot wise and non slot wise PUSCH and DM-RS mapping
- Perfect or practical synchronization and channel estimation
- HARQ operation with 16 processes

The figure below shows the processing chain implemented. For clarity, the DM-RS generation has been omitted.



Note that this example does not include closed-loop adaptation of the MIMO precoding according to channel conditions. The PUSCH MIMO precoding used in the example is as follows:

- For codebook based transmission, the MIMO precoding matrix used inside the PUSCH modulation can be selected using the TPMI parameter.
- The implementation-specific MIMO precoding matrix (for non-codebook based transmission, or MIMO precoding between transmission antenna ports and antennas for codebook based transmission) is an identity matrix.

To reduce the total simulation time, you can use the Parallel Computing Toolbox to execute the SNR points of the SNR loop in parallel.

## Simulation Length and SNR Points

Set the length of the simulation in terms of the number of 10ms frames. A large number of NFrames should be used to produce meaningful throughput results. Set the SNR points to simulate. The SNR is defined per RE and applies to each receive antenna.

```
simParameters = []; % Clear simParameters variable
simParameters.NFrames = 2; % Number of 10ms frames
simParameters.SNRIn = [-5 0 5]; % SNR range (dB)
```

The variable `displaySimulationInformation` controls the display of simulation information such as the HARQ process ID used for each subframe. In case of CRC error, the value of the index to the RV sequence is also displayed.

```
displaySimulationInformation = true;
```

## Channel Estimator Configuration

The logical variable `perfectChannelEstimator` controls channel estimation and synchronization behavior. When set to `true`, perfect channel estimation and synchronization is used. Otherwise, practical channel estimation and synchronization is used, based on the values of the received PUSCH DM-RS.

```
perfectChannelEstimator = true;
```

## Carrier and PUSCH Configuration

Set the key parameters of the simulation. These include:

- The bandwidth in resource blocks (12 subcarriers per resource block)
- Subcarrier spacing: 15, 30, 60, 120, 240 (kHz)
- Cyclic prefix length: normal or extended
- Cell ID
- Number of transmit and receive antennas

A substructure containing the UL-SCH and PUSCH parameters is also specified. This includes:

- Target code rate
- Allocated resource blocks (PRBSet)
- Modulation scheme: 'pi/2-BPSK', 'QPSK', '16QAM', '64QAM', '256QAM'
- Number of layers
- Transform precoding (enable/disable)
- PUSCH transmission scheme and MIMO precoding matrix indication (TPMI)
- Number of antenna ports
- PUSCH mapping type
- DM-RS configuration parameters

Other simulation wide parameters are:

- Propagation channel model: 'TDL' or 'CDL'

Note that if transform precoding is enabled, the number of layers should be set to 1.

```

% Bandwidth, numerology (SCS and CP type) and other general parameters
simParameters.NRB = 52; % Bandwidth in number of resource blocks (52RBs at 15kHz)
simParameters.SubcarrierSpacing = 15; % 15, 30, 60, 120, 240 (kHz)
simParameters.CyclicPrefix = 'Normal'; % 'Normal' or 'Extended'
simParameters.NCellID = 0; % Cell identity
simParameters.NTxAnts = 1; % Number of transmit antennas
simParameters.NRxAnts = 2; % Number of receive antennas

% UL-SCH/PUSCH parameters
simParameters.PUSCH.TargetCodeRate = 193 / 1024; % Code rate used to calculate transport block size
simParameters.PUSCH.PRBSets = (0:simParameters.NRB-1); % PUSCH PRB allocation
simParameters.PUSCH.SymbolSet = 0:13; % PUSCH symbol allocation in each slot
simParameters.PUSCH.NohPRB = 0; % Additional RE overhead per PRB
simParameters.PUSCH.EnableHARQ = true; % Enable/disable HARQ, if disabled, single transmission
simParameters.PUSCH.Modulation = 'QPSK'; % 'pi/2-BPSK', 'QPSK', '16QAM', '64QAM', '256QAM'
simParameters.PUSCH.NLayers = 1; % Number of PUSCH layers
simParameters.PUSCH.RNTI = 1; % Radio Network Temporary Identifier
simParameters.PUSCH.TransformPrecoding = false; % Enable/disable transform precoding
simParameters.PUSCH.TxScheme = 'nonCodebook'; % Transmission scheme ('nonCodebook','codebook')
simParameters.PUSCH.NAntennaPorts = 1; % Number of antenna ports for codebook based precoding
simParameters.PUSCH.TPMI = 0; % Precoding matrix indicator for codebook based precoding
% PUSCH DM-RS configuration
simParameters.PUSCH.PUSCHMappingType = 'A'; % PUSCH mapping type ('A'(slot-wise),'B'(non slot-wise))
simParameters.PUSCH.DMRSTypeAPosition = 2; % Mapping type A only. First DM-RS symbol position
simParameters.PUSCH.DMRSLength = 1; % Number of front-loaded DM-RS symbols (1(single),2(dual))
simParameters.PUSCH.DMRSAdditionalPosition = 1; % Additional DM-RS symbol positions (max range 0..13)
simParameters.PUSCH.DMRSConfigurationType = 1; % DM-RS configuration type (1,2)
simParameters.PUSCH.NumCDMGroupsWithoutData = 2; % CDM groups without data
simParameters.PUSCH.NIDNSCID = 0; % Scrambling identity (0..65535)
simParameters.PUSCH.NSCID = 0; % Scrambling initialization (0,1)
simParameters.PUSCH.NRSID = 0; % Scrambling ID for low-PAPR sequences (0..100)
simParameters.PUSCH.GroupHopping = 'Disable'; % Hopping type ('Enable','Disable')

% Define the propagation channel type
simParameters.ChannelType = 'TDL'; % 'CDL' or 'TDL'

```

Create carrier configuration object `carrier` and PUSCH configuration structure `pusch`.

```

carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = simParameters.SubcarrierSpacing;
carrier.CyclicPrefix = simParameters.CyclicPrefix;
carrier.NSizeGrid = simParameters.NRB;
carrier.NCellID = simParameters.NCellID;
pusch = simParameters.PUSCH;

```

For key simulation parameters, define local variables for convenience.

```

snrIn = simParameters.SNRIn;
nTxAnts = simParameters.NTxAnts;
nRxAnts = simParameters.NRxAnts;
channelType = simParameters.ChannelType;

```

### Propagation Channel Model Configuration

Create the channel model object. Both CDL and TDL channel models are supported [ 5 ].

```

if strcmpi(channelType, 'CDL')

```

```

channel = nrCDLChannel;
channel.DelayProfile = 'CDL-A';
[txsize,rxsize] = hArrayGeometry(nTxAnts,nRxAnts,'uplink');
channel.TransmitAntennaArray.Size = txsize;
channel.ReceiveAntennaArray.Size = rxsize;

else

channel = nrTDLChannel;
channel.DelayProfile = 'TDL-A';
channel.NumTransmitAntennas = nTxAnts;
channel.NumReceiveAntennas = nRxAnts;

end

channel.DelaySpread = 30e-9; % in seconds
channel.MaximumDopplerShift = 10; % in Hz

```

The sample rate for the channel model is set using the value returned from nrOFDMInfo.

```

waveformInfo = nrOFDMInfo(carrier);
channel.SampleRate = waveformInfo.SampleRate;

```

Get the maximum number of delayed samples by a channel multipath component. This is calculated from the channel path with the largest delay and the implementation delay of the channel filter. This is required later to flush the channel filter to obtain the received signal.

```

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

```

### Processing Loop

To determine the throughput at each SNR point, the PUSCH data is analyzed per transmission instance using the following steps:

- *Update current HARQ process.* Check the CRC of the previous transmission for the given HARQ process. Determine whether a retransmission is required. If that is not the case generate new data.
- *Generate resource grid.* Channel coding is performed by nrULSCH. It operates on the input transport block provided. Internally, it keeps a copy of the transport block in case a retransmission is required. The coded bits are modulated by nrPUSCH. Implementation-specific MIMO precoding is applied to the resulting signal. Note that if TxScheme='codebook', codebook based MIMO precoding will already have been applied inside nrPUSCH and the implementation-specific MIMO precoding is an additional stage of MIMO precoding.
- *Generate waveform.* The generated grid is then OFDM modulated.
- *Model noisy channel.* The waveform is passed through a CDL or TDL fading channel. AWGN is added. The SNR for each layer is defined per RE and per receive antenna.
- *Perform synchronization and OFDM demodulation.* For perfect synchronization, the channel impulse response is reconstructed and used to synchronize the received waveform. For practical synchronization, the received waveform is correlated with the PUSCH DM-RS. The synchronized signal is then OFDM demodulated.
- *Perform channel estimation.* If perfect channel estimation is used, the channel impulse response is reconstructed and OFDM demodulated to provide a channel estimate. For practical channel estimation, the PUSCH DM-RS is used.

- *Extract PUSCH and perform equalization.* The resource elements corresponding to the PUSCH allocation are extracted from the received OFDM resource grid and the channel estimate using `nrExtractResources`. The received PUSCH resource elements are then MMSE equalized using `nrEqualizeMMSE`.
- *Decode the PUSCH.* The equalized PUSCH symbols, along with a noise estimate, are demodulated and descrambled by `nrPUSCHDecode` to obtain an estimate of the received codewords.
- *Decode the Uplink Shared Channel (UL-SCH) and store the block CRC error for a HARQ process.* The vector of decoded soft bits is passed to `nrULSCHDecoder` which decodes the codeword and returns the block CRC error used to determine the throughput of the system.

```

% Array to store the maximum throughput for all SNR points
maxThroughput = zeros(length(snrIn),1);
% Array to store the simulation throughput for all SNR points
simThroughput = zeros(length(snrIn),1);

% Set up Redundancy Version (RV) sequence, number of HARQ processes and
% the sequence in which the HARQ processes are used
if pusch.EnableHARQ
    % From PUSCH demodulation requirements in RAN WG4 meeting #88bis
    % (R4-1814062)
    rvSeq = [0 2 3 1];
else
    % HARQ disabled - single transmission with RV=0, no retransmissions
    rvSeq = 0;
end

% Create UL-SCH encoder System object
encodeULSCH = nrULSCH;
encodeULSCH.MultipleHARQProcesses = true;
encodeULSCH.TargetCodeRate = pusch.TargetCodeRate;

% Create UL-SCH decoder System object
% Use layered belief propagation for LDPC decoding, with half the number of
% iterations as compared to the default for belief propagation decoding
decodeULSCH = nrULSCHDecoder;
decodeULSCH.MultipleHARQProcesses = true;
decodeULSCH.TargetCodeRate = pusch.TargetCodeRate;
decodeULSCH.LDPCDecodingAlgorithm = 'Layered belief propagation';
decodeULSCH.MaximumLDPCIterationCount = 6;

% The temporary variables 'carrier_init', 'pusch_init' and
% 'decodeULSCH_init' are used to create the temporary variables 'carrier',
% 'pusch' and 'decodeULSCH' within the SNR loop to create independent
% instances in case of parallel simulation
carrier_init = carrier;
pusch_init = pusch;
decodeULSCH_init = clone(decodeULSCH);

% Get the values of 'NFrames' to avoid overhead in case of parallel
% simulation
NFrames = simParameters.NFrames;

for snrIdx = 1:numel(snrIn) % comment out for parallel computing
% parfor snrIdx = 1:numel(snrIn) % uncomment for parallel computing
% To reduce the total simulation time, you can execute this loop in
% parallel by using the Parallel Computing Toolbox. Comment out the 'for'

```



```

% statement and uncomment the 'parfor' statement. If the Parallel Computing
% Toolbox is not installed, 'parfor' defaults to normal 'for' statement.
% Because parfor-loop iterations are executed in parallel in a
% nondeterministic order, the simulation information displayed for each SNR
% point can be intertwined. To switch off simulation information display,
% set the 'displaySimulationInformation' variable above to false

    % Reset the random number generator and channel, so that each SNR point
    % will experience the same noise and channel realizations
    rng('default');
    reset(channel);

    % Initialize variables for this SNR point, required for initialization
    % of variables when using the Parallel Computing Toolbox
    carrier = carrier_init;
    pusch = pusch_init;
    pathFilters = [];

    % Specify the order in which we cycle through the HARQ processes
    NHARQProcesses = 16;
    harqSequence = 1:NHARQProcesses;

    % Initialize the state of all HARQ processes and reset the UL-SCH
    % decoder
    harqProcesses = hNewHARQProcesses(NHARQProcesses,rvSeq,1);
    harqProcCntr = 0; % HARQ process counter
    decodeULSCH = clone(decodeULSCH_init);

    SNRdB = snrIn(snrIdx);
    fprintf('\nSimulating %s-based transmission scheme (%dx%d) and SCS=%dkHz with %s channel at %s\n',
        harqSequence{1}, harqSequence{2}, harqSequence{3}, harqSequence{4}, harqSequence{5}, harqSequence{6},
        harqSequence{7}, harqSequence{8}, harqSequence{9}, harqSequence{10}, harqSequence{11}, harqSequence{12},
        harqSequence{13}, harqSequence{14}, harqSequence{15}, harqSequence{16}, SNRdB, harqSequence{17}, harqSequence{18}, harqSequence{19}, harqSequence{20});

    % Total number of slots in the simulation period
    NSlots = NFrames * carrier.SlotsPerFrame;

    % Create 'ue' structure
    ue = struct();
    ue.NRB = carrier.NSizeGrid;
    ue.CyclicPrefix = carrier.CyclicPrefix;
    ue.SubcarrierSpacing = carrier.SubcarrierSpacing;
    ue.NCellID = carrier.NCellID;

    % Timing offset, updated in every slot for perfect synchronization and
    % when the correlation is strong for practical synchronization
    offset = 0;

    % Loop over the entire waveform length
    for nslot = 0:NSlots-1

        % Update the carrier and PUSCH slot numbers to account for a new
        % PUSCH transmission
        carrier.NSlot = nslot;
        pusch.NSlot = nslot;

        % Calculate the transport block size for this slot
        [puschIndices,dmrsIndices,dmrsSymbols,puschIndicesInfo] = hPUSCHResources(ue,pusch);
        MRB = numel(pusch.PRBSet);
        TBS = nrTBS(pusch.Modulation,pusch.NLayers,MRB,puschIndicesInfo.NREPerPRB,pusch.TargetCo

```

```

% Get HARQ process index for the current PUSCH from the HARQ index
% table
harqProcIdx = harqSequence(mod(harqProcCntr,length(harqSequence))+1);

% Update current HARQ process information (this updates the RV
% depending on CRC pass or fail in the previous transmission for
% this HARQ process)
harqProcesses(harqProcIdx) = hUpdateHARQProcess(harqProcesses(harqProcIdx),1);

% HARQ: check CRC from previous transmission, i.e. is a
% retransmission required?
NDI = false;
if harqProcesses(harqProcIdx).blkerr % errored
    if (harqProcesses(harqProcIdx).RVIdx==1) % end of rvSeq
        resetSoftBuffer(decodeULSCH,harqProcIdx-1);
        NDI = true;
    end
else % no error
    NDI = true;
end
if NDI
    trBlk = randi([0 1],TBS,1);
    setTransportBlock(encodeULSCH,trBlk,harqProcIdx-1);
end

% UL-SCH encoding
codedTrBlock = encodeULSCH(pusch.Modulation,pusch.NLayers,puschIndicesInfo.G,harqProcesses

% PUSCH modulation, including codebook based MIMO precoding if
% TxScheme = 'codebook'
puschSymbols = nrPUSCH(codedTrBlock,pusch.Modulation,pusch.NLayers,carrier.NCellID,pusch

% Create resource grid associated with PUSCH transmission period
puschGrid = nrResourceGrid(carrier,nTxAnts);

% Implementation-specific PUSCH MIMO precoding and mapping. This
% MIMO precoding step is in addition to any codebook based
% MIMO precoding done during PUSCH modulation above
if (strcmpi(pusch.TxScheme,'codebook'))
    % codebook based MIMO precoding, F precodes between PUSCH
    % transmit antenna ports and transmit antennas
    F = eye(pusch.NAntennaPorts,nTxAnts);
else
    % non-codebook based MIMO precoding, F precodes between PUSCH
    % layers and transmit antennas
    F = eye(pusch.NLayers,nTxAnts);
end
[~,puschAntIndices] = nrExtractResources(puschIndices,puschGrid);
puschGrid(puschAntIndices) = puschSymbols * F;

% Implementation-specific PUSCH DM-RS MIMO precoding and mapping.
% The DM-RS creation in hPUSCHResources above includes codebook
% based MIMO precoding if applicable
for p = 1:size(dmrsSymbols,2)
    [~,dmrsAntIndices] = nrExtractResources(dmrsIndices(:,p),puschGrid);
    puschGrid(dmrsAntIndices) = puschGrid(dmrsAntIndices) + dmrsSymbols(:,p) * F(p,:);
end

```

```

% OFDM modulation
txWaveform = nrOFDMModulate(carrier,puschGrid);

% Pass data through channel model. Append zeros at the end of the
% transmitted waveform to flush channel content. These zeros take
% into account any delay introduced in the channel. This is a mix
% of multipath delay and implementation delay. This value may
% change depending on the sampling rate, delay profile and delay
% spread
txWaveform = [txWaveform; zeros(maxChDelay,size(txWaveform,2))]; %#ok<AGROW>
[rxWaveform,pathGains,sampleTimes] = channel(txWaveform);

% Add AWGN to the received time domain waveform
% Normalize noise power by the IFFT size used in OFDM modulation,
% as the OFDM modulator applies this normalization to the
% transmitted waveform. Also normalize by the number of receive
% antennas, as the default behaviour of the channel model is to
% apply this normalization to the received waveform
SNR = 10^(SNRdB/20);
N0 = 1/(sqrt(2.0*nRxAnts*double(waveformInfo.Nfft))*SNR);
noise = N0*complex(randn(size(rxWaveform)),randn(size(rxWaveform)));
rxWaveform = rxWaveform + noise;

if (perfectChannelEstimator)
    % Perfect synchronization. Use information provided by the
    % channel to find the strongest multipath component
    pathFilters = getPathFilters(channel);
    [offset,mag] = nrPerfectTimingEstimate(pathGains,pathFilters);
else
    % Practical synchronization. Correlate the received waveform
    % with the PUSCH DM-RS to give timing offset estimate 't' and
    % correlation magnitude 'mag'. The function
    % hSkipWeakTimingOffset is used to update the receiver timing
    % offset. If the correlation peak in 'mag' is weak, the current
    % timing estimate 't' is ignored and the previous estimate
    % 'offset' is used
    [t,mag] = nrTimingEstimate(carrier,rxWaveform,dmrsIndices,dmrsSymbols); %#ok<UNRCH>
    offset = hSkipWeakTimingOffset(offset,t,mag);
end
rxWaveform = rxWaveform(1+offset:end,:);

% Perform OFDM demodulation on the received data to recreate the
% resource grid, including padding in the event that practical
% synchronization results in an incomplete slot being demodulated
rxGrid = nrOFDMDemodulate(carrier,rxWaveform);
[K,L,R] = size(rxGrid);
if (L < carrier.SymbolsPerSlot)
    rxGrid = cat(2,rxGrid,zeros(K,carrier.SymbolsPerSlot-L,R));
end

if (perfectChannelEstimator)
    % Perfect channel estimation, use the value of the path gains
    % provided by the channel
    estChannelGrid = nrPerfectChannelEstimate(carrier,pathGains,pathFilters,offset,sampleTimes);
else
    % Get perfect noise estimate (from the noise realization)
    noiseGrid = nrOFDMDemodulate(carrier,noise(1+offset:end,:));
    noiseEst = var(noiseGrid(:));
end

```

```

% Apply MIMO deprecoding to estChannelGrid to give an estimate
% per transmission layer
K = size(estChannelGrid,1);
estChannelGrid = reshape(estChannelGrid,K*carrier.SymbolsPerSlot*nRxAnts,nTxAnts);
estChannelGrid = estChannelGrid * F.';
if (strcmpi(pusch.TxScheme,'codebook'))
    W = nrPUSCHCodebook(pusch.NLayers,pusch.NAntennaPorts,pusch.TPMI,pusch.TransformPrecoding);
    estChannelGrid = estChannelGrid * W.';
end
estChannelGrid = reshape(estChannelGrid,K,carrier.SymbolsPerSlot,nRxAnts,[]);
else
% Practical channel estimation between the received grid and
% each transmission layer, using the PUSCH DM-RS for each layer
[~,dmrsLayerIndices,dmrsLayerSymbols] = hPUSCHResources(ue,setfield(pusch,'TxScheme','codebook'));
[estChannelGrid,noiseEst] = nrChannelEstimate(carrier,rxGrid,dmrsLayerIndices,dmrsLayerSymbols);
end

% Get PUSCH resource elements from the received grid
[puschRx,puschHest] = nrExtractResources(puschIndices,rxGrid,estChannelGrid);

% Equalization
[puschEq,csi] = nrEqualizeMMSE(puschRx,puschHest,noiseEst);

% Decode PUSCH physical channel
[ulschLLRs,rxSymbols] = nrPUSCHDecode(puschEq,pusch.Modulation,carrier.NCellID,pusch.RNTI);

% Apply channel state information (CSI) produced by the equalizer,
% including the effect of transform precoding if enabled
if (pusch.TransformPrecoding)
    MSC = MRB * 12;
    csi = nrTransformDeprecode(csi,MRB) / sqrt(MSC);
    csi = repmat(csi(1:MSC:end).',1,MSC).';
    csi = reshape(csi,size(rxSymbols));
end
csi = nrLayerDemap(csi);
Qm = length(ulschLLRs) / length(rxSymbols);
csi = reshape(repmat(csi{1}.',Qm,1),[],1);
ulschLLRs = ulschLLRs .* csi;

% Decode the UL-SCH transport channel
decodeULSCH.TransportBlockLength = TBS;
[decbits,harqProcesses(harqProcIdx).blkerr] = decodeULSCH(ulschLLRs,pusch.Modulation,pusch.RNTI);

% Store values to calculate throughput
simThroughput(snrIdx) = simThroughput(snrIdx) + (~harqProcesses(harqProcIdx).blkerr * TBS);
maxThroughput(snrIdx) = maxThroughput(snrIdx) + TBS;

% Display transport block error information
if (displaySimulationInformation)
    fprintf('\n(%3.2f%) HARQ Proc %d: ',100*(nslot+1)/NSlots,harqProcIdx);
    estrings = {'passed','failed'};
    rvi = harqProcesses(harqProcIdx).RVIdx;
    if rvi == 1
        ts = sprintf('Initial transmission (RV=%d)',rvSeq(rvi));
    else
        ts = sprintf('Retransmission #%d (RV=%d)',rvi-1,rvSeq(rvi));
    end
end

```

```

        fprintf('%s %s. ',ts,eststrings{1+harqProcesses(harqProcIdx).blkerr});
    end

    % Update HARQ process counter
    harqProcCntr = harqProcCntr + 1;

end

% Display the results dynamically in the command window
if (displaySimulationInformation)
    fprintf('\n');
end
fprintf(['\nThroughput(Mbps) for ' num2str(NFrames) ' frame(s) ',' = %.4f\n'], 1e-6*simThroughput(snrIdx)*NFrames);
fprintf(['Throughput(%) for ' num2str(NFrames) ' frame(s) = %.4f\n'],simThroughput(snrIdx)*100);

end

```

Simulating nonCodebook-based transmission scheme (1x2) and SCS=15kHz with TDL channel at -5dB SNR

```

(5.00%) HARQ Proc 1: Initial transmission (RV=0) failed.
(10.00%) HARQ Proc 2: Initial transmission (RV=0) failed.
(15.00%) HARQ Proc 3: Initial transmission (RV=0) failed.
(20.00%) HARQ Proc 4: Initial transmission (RV=0) failed.
(25.00%) HARQ Proc 5: Initial transmission (RV=0) failed.
(30.00%) HARQ Proc 6: Initial transmission (RV=0) failed.
(35.00%) HARQ Proc 7: Initial transmission (RV=0) failed.
(40.00%) HARQ Proc 8: Initial transmission (RV=0) failed.
(45.00%) HARQ Proc 9: Initial transmission (RV=0) failed.
(50.00%) HARQ Proc 10: Initial transmission (RV=0) failed.
(55.00%) HARQ Proc 11: Initial transmission (RV=0) failed.
(60.00%) HARQ Proc 12: Initial transmission (RV=0) failed.
(65.00%) HARQ Proc 13: Initial transmission (RV=0) failed.
(70.00%) HARQ Proc 14: Initial transmission (RV=0) failed.
(75.00%) HARQ Proc 15: Initial transmission (RV=0) failed.
(80.00%) HARQ Proc 16: Initial transmission (RV=0) failed.
(85.00%) HARQ Proc 1: Retransmission #1 (RV=2) passed.
(90.00%) HARQ Proc 2: Retransmission #1 (RV=2) passed.
(95.00%) HARQ Proc 3: Retransmission #1 (RV=2) passed.
(100.00%) HARQ Proc 4: Retransmission #1 (RV=2) passed.

```

Throughput(Mbps) for 2 frame(s) = 0.5712

Throughput(%) for 2 frame(s) = 20.0000

Simulating nonCodebook-based transmission scheme (1x2) and SCS=15kHz with TDL channel at 0dB SNR

```

(5.00%) HARQ Proc 1: Initial transmission (RV=0) passed.
(10.00%) HARQ Proc 2: Initial transmission (RV=0) passed.
(15.00%) HARQ Proc 3: Initial transmission (RV=0) passed.
(20.00%) HARQ Proc 4: Initial transmission (RV=0) passed.
(25.00%) HARQ Proc 5: Initial transmission (RV=0) passed.
(30.00%) HARQ Proc 6: Initial transmission (RV=0) passed.
(35.00%) HARQ Proc 7: Initial transmission (RV=0) passed.
(40.00%) HARQ Proc 8: Initial transmission (RV=0) passed.
(45.00%) HARQ Proc 9: Initial transmission (RV=0) passed.
(50.00%) HARQ Proc 10: Initial transmission (RV=0) passed.
(55.00%) HARQ Proc 11: Initial transmission (RV=0) passed.
(60.00%) HARQ Proc 12: Initial transmission (RV=0) passed.

```

```
(65.00%) HARQ Proc 13: Initial transmission (RV=0) passed.
(70.00%) HARQ Proc 14: Initial transmission (RV=0) passed.
(75.00%) HARQ Proc 15: Initial transmission (RV=0) passed.
(80.00%) HARQ Proc 16: Initial transmission (RV=0) passed.
(85.00%) HARQ Proc 1: Initial transmission (RV=0) passed.
(90.00%) HARQ Proc 2: Initial transmission (RV=0) passed.
(95.00%) HARQ Proc 3: Initial transmission (RV=0) passed.
(100.00%) HARQ Proc 4: Initial transmission (RV=0) passed.
```

```
Throughput(Mbps) for 2 frame(s) = 2.8560
Throughput(%) for 2 frame(s) = 100.0000
```

Simulating nonCodebook-based transmission scheme (1x2) and SCS=15kHz with TDL channel at 5dB SNR

```
(5.00%) HARQ Proc 1: Initial transmission (RV=0) passed.
(10.00%) HARQ Proc 2: Initial transmission (RV=0) passed.
(15.00%) HARQ Proc 3: Initial transmission (RV=0) passed.
(20.00%) HARQ Proc 4: Initial transmission (RV=0) passed.
(25.00%) HARQ Proc 5: Initial transmission (RV=0) passed.
(30.00%) HARQ Proc 6: Initial transmission (RV=0) passed.
(35.00%) HARQ Proc 7: Initial transmission (RV=0) passed.
(40.00%) HARQ Proc 8: Initial transmission (RV=0) passed.
(45.00%) HARQ Proc 9: Initial transmission (RV=0) passed.
(50.00%) HARQ Proc 10: Initial transmission (RV=0) passed.
(55.00%) HARQ Proc 11: Initial transmission (RV=0) passed.
(60.00%) HARQ Proc 12: Initial transmission (RV=0) passed.
(65.00%) HARQ Proc 13: Initial transmission (RV=0) passed.
(70.00%) HARQ Proc 14: Initial transmission (RV=0) passed.
(75.00%) HARQ Proc 15: Initial transmission (RV=0) passed.
(80.00%) HARQ Proc 16: Initial transmission (RV=0) passed.
(85.00%) HARQ Proc 1: Initial transmission (RV=0) passed.
(90.00%) HARQ Proc 2: Initial transmission (RV=0) passed.
(95.00%) HARQ Proc 3: Initial transmission (RV=0) passed.
(100.00%) HARQ Proc 4: Initial transmission (RV=0) passed.
```

```
Throughput(Mbps) for 2 frame(s) = 2.8560
Throughput(%) for 2 frame(s) = 100.0000
```

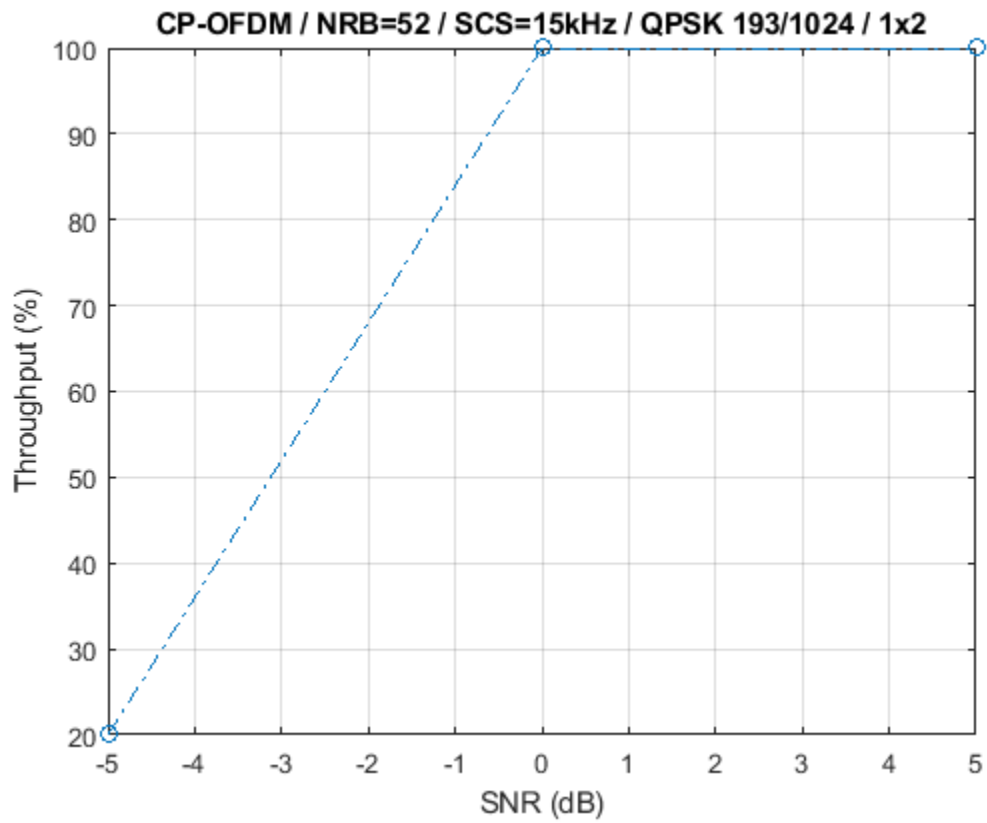
## Results

Display the measured throughput. This is calculated as the percentage of the maximum possible throughput of the link given the available resources for data transmission.

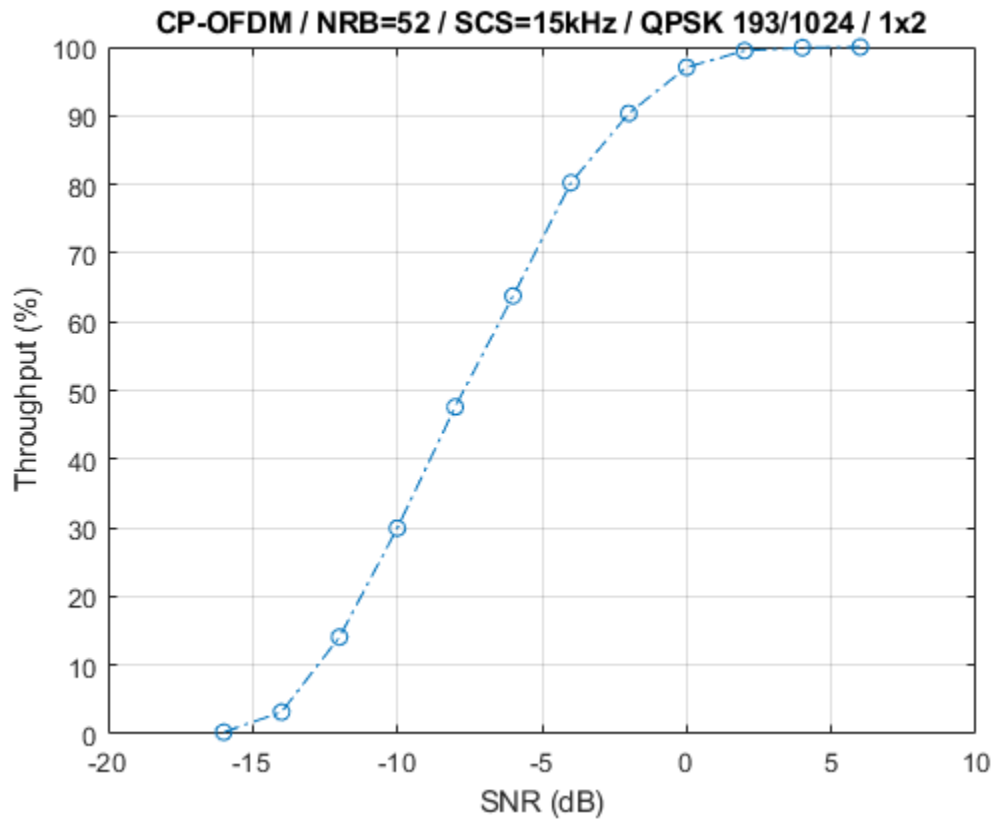
```
figure;
plot(snrIn,simThroughput*100./maxThroughput,'o-.')
xlabel('SNR (dB)'); ylabel('Throughput (%)'); grid on;
if (pusch_init.TransformPrecoding)
    ofdmType = 'DFT-s-OFDM';
else
    ofdmType = 'CP-OFDM';
end
title(sprintf('%s / NRB=%d / SCS=%dkHz / %s %d/1024 / %dx%d', ...
    ofdmType,carrier_init.NSizeGrid,carrier_init.SubcarrierSpacing, ...
    pusch_init.Modulation, ...
    round(pusch_init.TargetCodeRate*1024),nTxAnts,nRxAnts));

simResults.simParameters = simParameters;
```

```
simResults.simThroughput = simThroughput;  
simResults.maxThroughput = maxThroughput;
```



The figure below shows throughput results obtained simulating 10000 subframes (NFrames = 1000, SNRIn = -16:2:6).



## Appendix

This example uses the following helper functions:

- hArrayGeometry.m
- hNewHARQProcesses.m
- hPUSCHResources.m
- hSkipWeakTimingOffset.m
- hUpdateHARQProcess.m

## Selected Bibliography

- 1 3GPP TS 38.211. "NR; Physical channels and modulation (Release 15)." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.
- 2 3GPP TS 38.212. "NR; Multiplexing and channel coding (Release 15)." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.
- 3 3GPP TS 38.213. "NR; Physical layer procedures for control (Release 15)." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.
- 4 3GPP TS 38.214. "NR; Physical layer procedures for data (Release 15)." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.



- 5 3GPP TR 38.901. "Study on channel model for frequencies from 0.5 to 100 GHz (Release 15)."  
3rd Generation Partnership Project; Technical Specification Group Radio Access Network.

## See Also

### Objects

[nrCDLChannel](#) | [nrTDLChannel](#) | [nrULSCH](#) | [nrULSCHDecoder](#)

### Functions

[nrPUSCH](#) | [nrPUSCHDecode](#) | [nrULSCHInfo](#)

## NR SRS Configuration

This example shows how to generate sounding reference signals (SRS), as defined in TS 38.211 Section 6.4.1.4 [1 on page 2-0 ], including SRS configuration, symbol and indices generation, OFDM resource grid mapping, and SRS waveform generation. The example demonstrates how to select the appropriate parameters to position the SRS in frequency and to setup:

- *Full-band transmissions*: Generate an SRS spanning all the available bandwidth.
- *Frequency-hopping transmissions*: Generate periodic and aperiodic SRS transmissions with different frequency-hopping patterns.
- *Multi-user transmissions*: Generate orthogonal SRS using time, frequency, and cyclic shifts.

### Introduction

Sounding reference signals are uplink physical signals employed by user equipment (UE) for uplink channel sounding, including channel quality estimation and synchronization. Unlike demodulation reference signals (DM-RS), SRS are not associated to any physical uplink channels and they support uplink channel-dependent scheduling and link adaptation. SRS assist in:

- Codebook-based closed-loop spatial multiplexing.
- Control uplink transmit timing.
- Reciprocity-based downlink precoding in multi-user MIMO setups.
- Quasi co-location of physical channels and reference signals.

### Frequency Positioning

This section shows how to configure the SRS position in frequency domain and determine the bandwidth allocated to the SRS.

Configure a 15 MHz bandwidth carrier with 15 kHz subcarrier spacing (SCS).

```
carrier = nrCarrierConfig;
carrier.NSizeGrid = 79; % Bandwidth in RB
carrier.SubcarrierSpacing = 15 kHz;
```

The bandwidth configuration parameters CSRS and BSRS control the bandwidth allocated to the SRS, which normally increases with CSRS and decreases with BSRS. Use these interactive controls to configure the SRS bandwidth.

```
srs = nrSRSConfig;
srs.CSRS = 10; % Bandwidth configuration C_SRS (0...63)
srs.BSRS = 1; % Bandwidth configuration B_SRS (0...3)
```

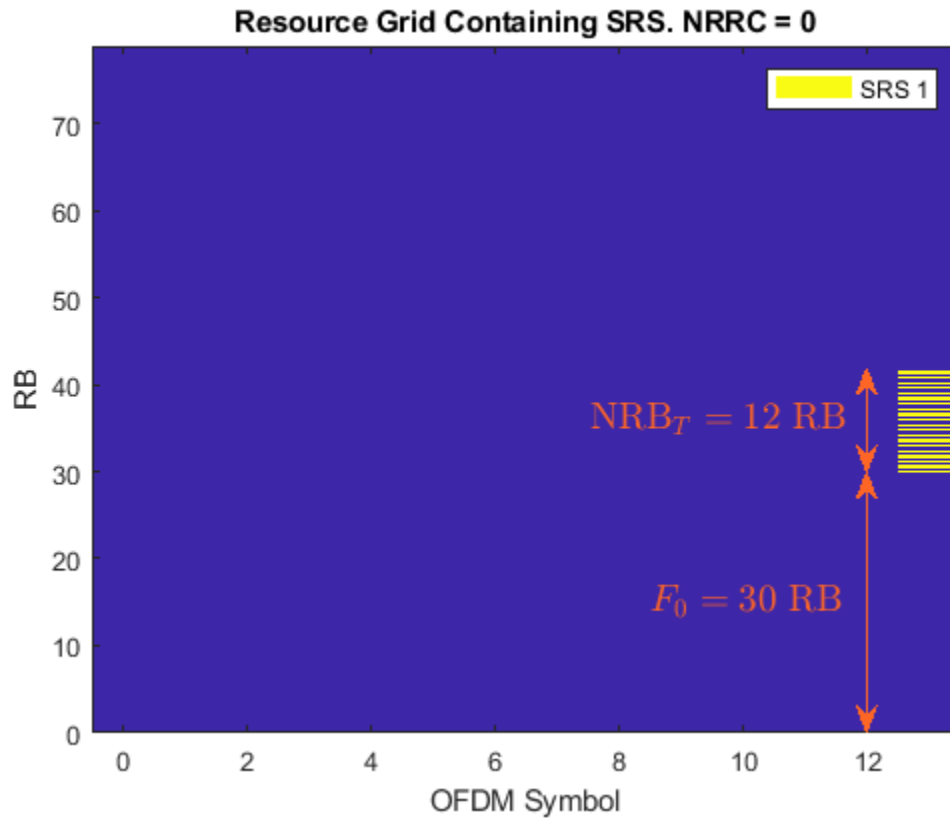
To modify the frequency position of the SRS, change the values of FrequencyStart and NRRC. FrequencyStart specifies the frequency origin of the SRS in RBs with respect to the carrier origin when NRRC = 0.

```
srs.FrequencyStart = 30; % Frequency position of the SRS in carrier in RB
srs.NRRC = 0; % Frequency domain position in blocks of 4 PRB
```

```

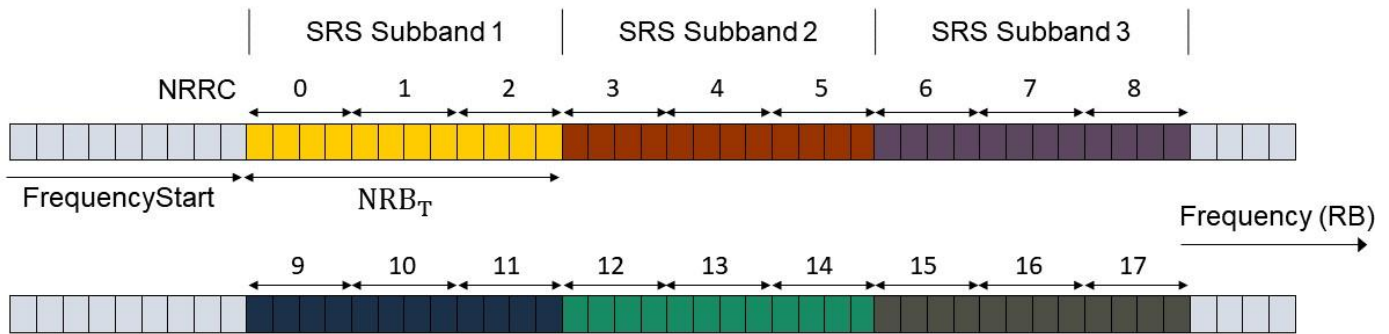
hSRSGrid(carrier,srs,1,true); % Create and display a single-slot resource grid containing SRS
title(['Resource Grid Containing SRS. NRRC = ' num2str(srs.NRRC)]);
hSRSAnnotations(carrier,srs);

```



This figure displays a single-slot OFDM resource grid containing an SRS. For  $CSRS = 10$  and  $BSRS = 1$ , the frequency position of the SRS ( $F_0$ ) shifts by  $NRB_T$  when  $NRRC$  is in the range (3...5) and by  $2NRB_T$  when  $NRRC$  is in the range (6...8). The SRS returns to the initial position (FrequencyStart) when  $NRRC$  is 9.  $NRB_T$  is the number of resource blocks (RBs) allocated to the SRS per transmission.

This figure illustrates the concepts introduced above for  $CSRS = 10$  and  $BSRS = 1$ .



$F_0$  for NRRC = (0, 1, 2, 9, 10, 11, 18 ...)

$F_0$  for NRRC = (3, 4, 5, 12, 13, 14, 21, ...)

NRRC is an additional frequency offset specified as a number of 4 RBs and it corresponds to the higher layer parameter *freqDomainPosition* (see TS 38.331 Section 6.3.2 SRS-Config [2 on page 2-0]). For values of NRRC between  $k \cdot (\text{NRB}_T/4)$  and  $(k + 1) \cdot (\text{NRB}_T/4) - 1$ , the frequency position of the SRS is shifted by  $k \cdot \text{NRB}_T$ , where  $k$  is an integer. TS 38.211 Section 6.4.1.4 refers to  $\text{NRB}_T$  as  $m_{\text{SRS},b}$  with  $b = \text{BSRS}$ . For more information, see the *nrBPerTransmission* property of the *nrSRSConfig* configuration object.

This equation determines the origin of the SRS in frequency:

$$F_0 = \text{FrequencyStart} + \text{NRB}_T \cdot \text{mod}\left(\left\lfloor 4 \frac{\text{NRRC}}{\text{NRB}_T} \right\rfloor, \text{NSB}\right)$$

NSB denotes the number of SRS subbands (frequency bands of size  $\text{NRB}_T$ ) where the SRS can be positioned through the parameter NRRC. To calculate NSB, you can use the SRS bandwidth configuration table (see TS 38.211 Table 6.4.1.4.3-1). You can also access this table through the *BandwidthConfigurationTable* property of the *nrSRSConfig* object.

```
csrs = srs.CSRS;
disp(nrSRSConfig.BandwidthConfigurationTable(csrs+(0:2) + (csrs==0),:));
```

C_SRS	m_SRS_0	N_0	m_SRS_1	N_1	m_SRS_2	N_2	m_SRS_3	N_3
9	32	1	16	2	8	2	4	2
10	36	1	12	3	4	3	4	1
11	40	1	20	2	4	5	4	1

The first column contains possible values of the parameter CSRS. For CSRS = 10 and BSRS = 1, the number of unique SRS subbands is  $\text{NSB} = N_0 \cdots N_{\text{BSRS}} = 3$ , where  $N_0 = 1$  and  $N_1 = 3$ .

```
% Calculate and display the number of SRS subbands configurable by NRRC
NSBTable = hSRSNumberOfSubbandsOrHoppingPatterns(srs);

fprintf('Number of SRS subbands (NRRC < %d): %d', NSBTable*srs.NRBPerTransmission/4,NSBTable);

Number of SRS subbands (NRRC < 9): 3
```

```
% Calculate the frequency origin of the first SRS symbol
f0 = hSRSEFrequencyOrigin(srs);
fprintf('The frequency origin of the SRS is F0 = %d RB.', f0);
```

The frequency origin of the SRS is F0 = 30 RB.

Confirm the frequency position of the SRS using the info output of nrSRSIndices

```
[~,info] = nrSRSIndices(carrier,srs);
display(info.PRBSets(1))
```

30

For a given value of CSRS, the frequency band where the SRS can be allocated using NRRC is  $\text{FrequencyStart} + (0 \dots \text{NSB} \cdot \text{NRB}_T - 1)$ . For  $\text{CSRS} = 10$ , the SRS wraps around when  $\text{NRRC} = \text{NSB} \cdot \text{NRB}_T / 4 = N_0 \dots N_{B_{\text{SRS}}} \cdot m_{\text{SRS},b} / 4 = m_{\text{SRS},0} / 4 = 9$ , which results into the same frequency position as that of  $\text{NRRC} = 0$ .

```
fprintf('The SRS frequency range is limited to the range (%d,%d) RB for the current value of CSRS
```

The SRS frequency range is limited to the range (30,66) RB for the current value of CSRS (10).

When intra-slot frequency hopping is enabled, the calculation of FrequencyOrigin is only valid for the first SRS symbol in the slot. For frequency-hopping SRS, use the outputs info.PRBSets or info.SubcarrierOffset of the nrSRSIndices function to determine the frequency position of subsequent symbols.

```
fprintf('The frequency origin of the SRS is F0 = %d RB.',info.PRBSets(1));
```

The frequency origin of the SRS is F0 = 30 RB.

### Full-Bandwidth Configuration

This section shows how to configure and generate a full-band SRS transmission by calculating the appropriate SRS bandwidth parameters for a carrier.

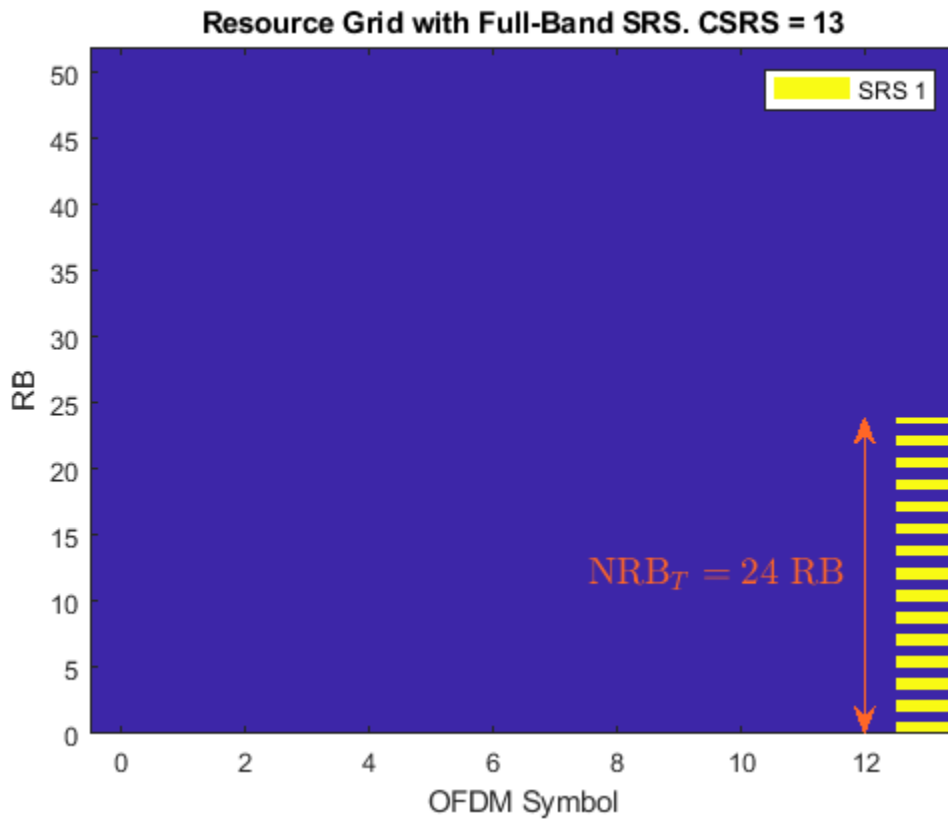
Configure a 10 MHz bandwidth carrier with 15 kHz SCS.

```
carrier = nrCarrierConfig;
carrier.NSizeGrid = ;
carrier.SubcarrierSpacing = ;
```

The bandwidth configuration parameters CSRS, BSRS, and BHop control the bandwidth allocated to the SRS. To configure a full-band SRS, set  $\text{CSRS} = 14$  and  $\text{BSRS} = 0$ .

```
srs = nrSRSConfig;
srs.CSRS = 13  ; % Bandwidth configuration C_SRS (0...63)
srs.BSRS = 1  ; % Bandwidth configuration B_SRS (0...3)
```

```
hSRSGrid(carrier,srs, 1, true); % Create and display a single-slot resource grid containing SRS
title(['Resource Grid with Full-Band SRS. CSRS = ' num2str(srs.CSRS)]);
hSRSAnnotations(carrier,srs);
```



For an SRS frequency allocation, you can find the appropriate values of CSRS, BSRS, and BHop in the SRS bandwidth configuration table (see TS 38.211 Table 6.4.1.4.3-1). Alternatively, `nrSRSConfig.BandwidthConfigurationTable` provides an easy way to access and display this table.

```
% To display relevant rows of the bandwidth configuration table, calculate the value of CSRS for
[csrs,bsrs] = hSRSSBandwidthConfiguration(srs,carrier.NSizeGrid);
```

```
% Display bandwidth configuration table
disp(nrSRSConfig.BandwidthConfigurationTable(csrs+(0:2) + 1*(csrs==0),:));
```

C_SRS	m_SRS_0	N_0	m_SRS_1	N_1	m_SRS_2	N_2	m_SRS_3	N_3
13	48	1	24	2	12	2	4	3
14	52	1	4	13	4	1	4	1
15	56	1	28	2	4	7	4	1

The columns labeled as `m_SRS_b` with `b = 0...3` contain the number of RBs allocated to the SRS for the parameter `b = BSRS` and non-hopping configurations. The row corresponding to `C_SRS = 14` contains a number of RBs `m_SRS_0 = 52`, which is the closest value to the carrier bandwidth. Therefore, the parameters `CSRS = 14` and `BSRS = 0` configure a full-band SRS transmission for the current carrier configuration. This example uses the bandwidth configuration table to calculate the values of CSRS and BSRS that maximize the SRS bandwidth within the carrier.

```
fprintf('For a full-band SRS in a carrier bandwidth of %d RB, set CSRS = %d and BSRS = %d.',carrier.NSizeGrid,csrs,bsrs);
```

For a full-band SRS in a carrier bandwidth of 52 RB, set CSRS = 14 and BSRS = 0.

You can use the SRS read-only property `NRBPerTransmission` to confirm that the generated SRS fits into the carrier bandwidth.

```
fprintf('The SRS bandwidth (%d RB) is lower than or equal to the carrier bandwidth (%d RB).',srs
```

```
The SRS bandwidth (24 RB) is lower than or equal to the carrier bandwidth (52 RB).
```

### Frequency-Hopping Configuration

You can configure intra-slot and inter-slot frequency hopping for multi-symbol and multi-slot SRS transmissions, respectively. The instantaneous bandwidth per OFDM symbol is constant across SRS OFDM symbols and is smaller than the bandwidth over which the SRS hops.

Configure a 15 MHz bandwidth carrier with 15 kHz SCS.

```
carrier = nrCarrierConfig;
carrier.NSizeGrid = 79;
carrier.SubcarrierSpacing = 15 kHz;
```

Create a four-symbol SRS located at the end of the slot. Select the repetition factor to indicate the number of equal consecutive SRS transmissions (OFDM symbols) in a slot. For frequency-hopping configurations, `Repetition` must be lower than the number of SRS symbols.

```
srs = nrSRSConfig;
srs.NumSRSSymbols = 4;
srs.Repetition = 1;
srs.SymbolStart = 10; % Time-domain position of the SRS in the slot. (
```

Downlink control information (DCI) can trigger aperiodic SRS transmissions using the higher layer parameter `resourceType` (see TS 38.331 Section 6.3.2 SRS-Config). As the frequency hopping pattern is reset for aperiodic SRS resource types at each slot, select `periodic` or `semi-persistent` SRS resource types to enable inter-slot frequency hopping or `aperiodic` to disable it. You can configure the slot-wise periodicity and offset of the SRS transmissions by using the property `SRSPeriod`. For aperiodic resource types, the parameter `SRSPeriod` controls the periodicity and offset of DCI signals triggering aperiodic SRS transmissions.

```
srs.ResourceType = periodic;
srs.SRSPeriod = [2 0]; % Periodicity in slots (1,2,4,5,8,10,...)
srs.SRSPeriod(2) = 0; % Offset in slots must be smaller than the periodi
```

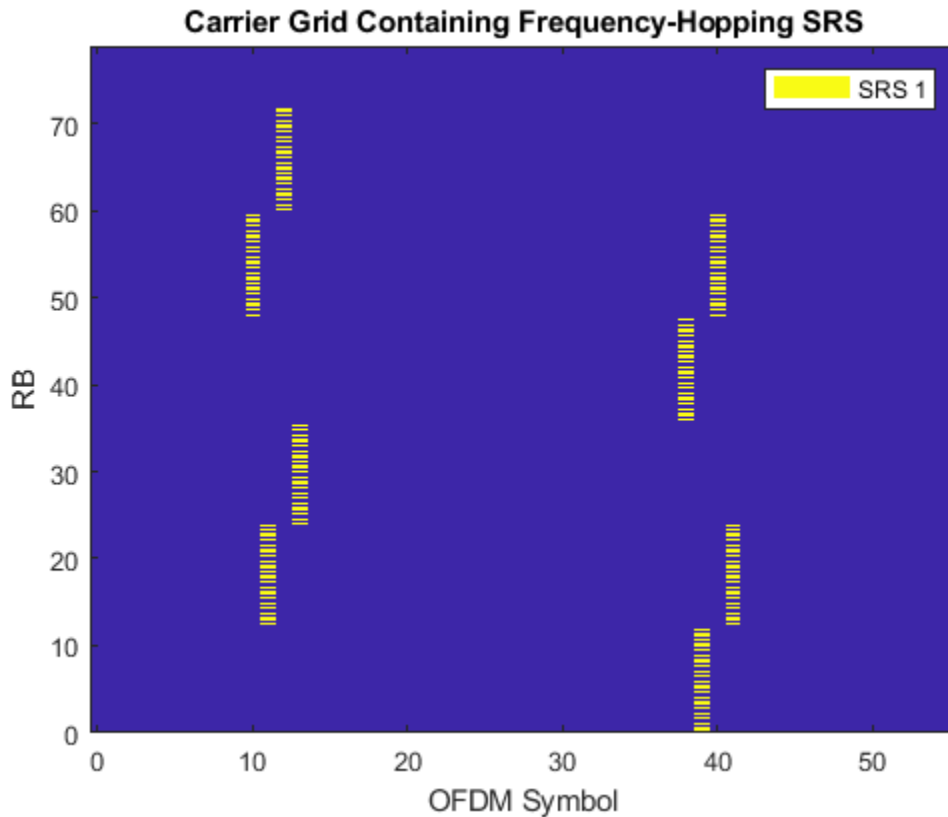
Use these interactive controls to select a hopping configuration and observe the changes to the OFDM resource grid.

```
srs.CSRS = 19; % Bandwidth configuration C_SRS (0...63)
srs.BSRS = 2; % Bandwidth configuration B_SRS (0...3)
srs.BHop = 0; % Frequency hopping configuration (0...3). Set BHop >= BSRS
srs.NRRC = 14; % Frequency domain position in blocks of 4 PRB (0...67)
```

```

% Create and display a multi-slot resource grid containing SRS
duration = 2*srs.SRSPeriod(1); % Transmission length in slots
hSRSGrid(carrier,srs, duration, true);
title('Carrier Grid Containing Frequency-Hopping SRS')

```



The bandwidth over which the SRS symbols hop increases with the value of CSRS and decreases with BHop (until BHop = BSRS disables hopping). Increasing BSRS reduces the allocated bandwidth per OFDM symbol (property NRBPPerTransmission) and can reduce the intra-slot frequency hopping as well. To disable frequency hopping, set BHop  $\geq$  BSRS. For non-hopping configurations, the roles of CSRS and BSRS are analogous, as the allocated bandwidth increases with CSRS and decreases with BSRS.

You can use the bandwidth configuration table (TS 38.211 Table 6.4.1.4.3-1) to calculate the number of different frequency-hopping patterns configurable by the SRS parameter NRRC as  $N_{\text{FHP}} = N_{\text{BHop}+1} \cdots N_{\text{BSRS}}$ . The frequency-hopping patterns repeat when  $\text{NRRC} > N_{\text{FHP}} \cdot \text{NRB}_T/4$ .

```
N = hSRSNumberOfSubbandsOrHoppingPatterns(srs);
```

```

if srs.BHop < srs.BSRS && srs(1).NRB > 16 % Number of unique frequency-hopping patterns
    fprintf('Number of unique frequency-hopping patterns (configurable by NRRC < %d): %d.',N*srs
else % Number of unique SRS subbands
    fprintf('Number of unique SRS subbands (configurable by NRRC < %d): %d.',N*srs.NRBPerTransm
end

```

```
Number of unique frequency-hopping patterns (configurable by NRRC < 18): 6.
```



Set the bandwidth configurations parameters as  $CSRS = 20$ ,  $BSRS = 2$ , and  $BHop = 1$ . The condition  $BHop < BSRS$  does not guarantee frequency hopping. However, the opposite is true:  $BHop \geq BSRS$  always disables frequency hopping. You can determine when an SRS bandwidth configuration ( $CSRS, BSRS, BHop$ ) produces frequency hopping using the  $N_b$  parameter from the bandwidth configuration table and evaluating the condition  $NFHP = N_{BHop} + 1 \cdots N_{BSRS} > 1$ .

```
isFreqHoppingConfiguration = hSRSNumberOfSubbandsOrHoppingPatterns(srs) > 1;
disp(['Frequency hopping is' repmat(' not',1,~isFreqHoppingConfiguration) ' enabled.'])
```

Frequency hopping is enabled.

In frequency-hopping cases, the read-only property `NRB` specifies the bandwidth over which the SRS hops and `NRBPerTransmission` specifies the instantaneous bandwidth allocated per SRS OFDM symbol.

```
t = table([srs.NRB; srs.NRBPerTransmission], 'RowNames', {'Freq-Hopping bandwidth', 'Instantaneous
disp(t)
```

	NRB
Freq-Hopping bandwidth	72
Instantaneous bandwidth	12

## Multi-User Configurations

This section explains how to configure multiple SRS transmissions suitable for multi-user setups. You can use time-domain, frequency-domain, and sequence-domain parameters to create sets of orthogonal (interference-free) SRS transmissions.

### Time-Domain Orthogonal SRS

You can create time-domain orthogonal SRS transmissions in multiple ways, for example:

- Configure different slot periodicity and offset for different SRS by using the property `SRSPeriod`.
- Configure different symbol-wise time domain allocations for different SRS by using the property `SymbolStart`.

Configure a 10 MHz bandwidth carrier with 15 kHz SCS.

```
carrier = nrCarrierConfig;
carrier.NSizeGrid = 52;
carrier.SubcarrierSpacing = 15 kHz;
```

Specify the slot periodicity and offset to create orthogonal SRS transmissions.

#### First SRS configuration:

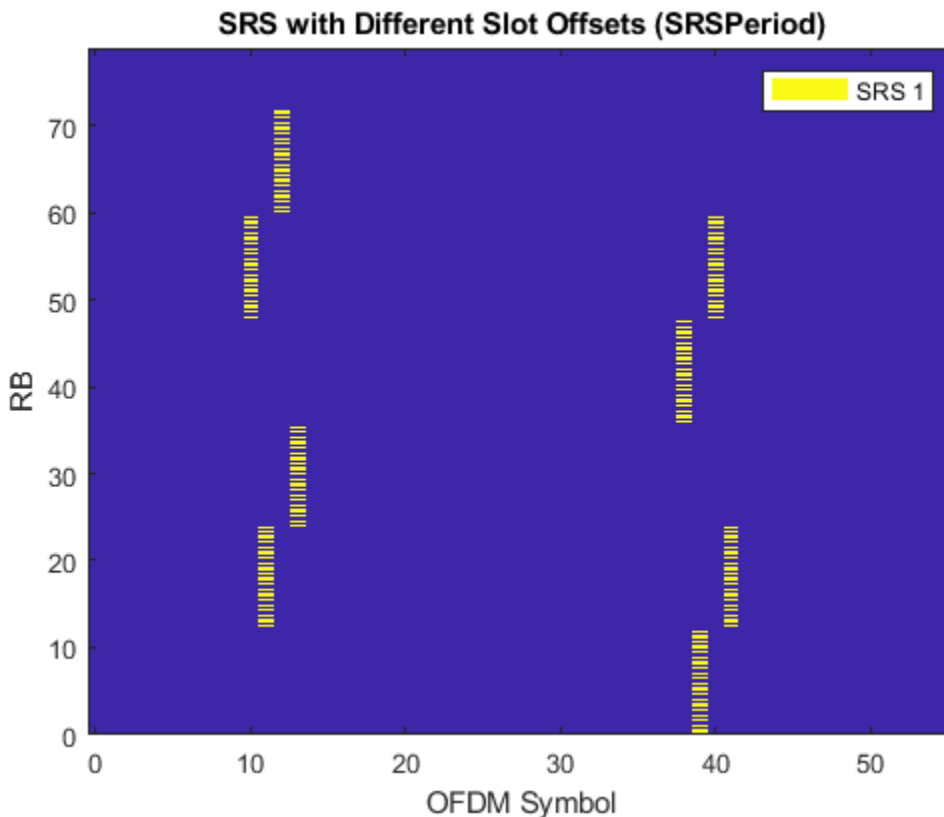
```
srs = nrSRSConfig;
srs(1).SRSPeriod = [1 0];
srs(1).SRSPeriod(1) = 4; % Slot periodicity and offset
srs(1).SRSPeriod(2) = 0; % Slot offset of first SRS
```

#### Second SRS configuration:

```
srs(2) = srs(1); % Create a copy of the configured SRS
srs(2).SRSPeriod(1) = 2; % Periodicity in slots
srs(2).SRSPeriod(2) = 1; % Offset in slots
```

This figure displays an OFDM resource grid containing the configured SRS transmissions.

```
hSRSGrid(carrier,srs,srs(1).SRSPeriod(1)*2); % Generate a multi-slot resource grid containing SRS
title('SRS with Different Slot Offsets (SRSPeriod)')
```



Specify the number of SRS symbols and location in the slot to create orthogonal SRS transmissions.

**First SRS configuration:**

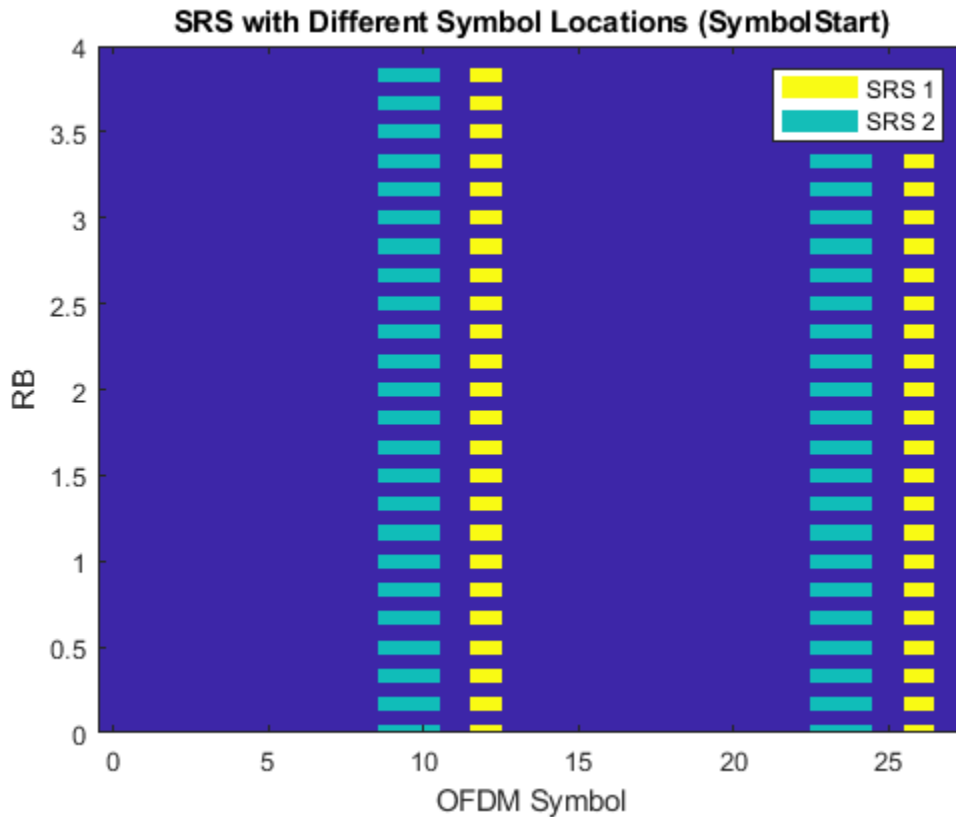
```
srs = nrSRSConfig;
srs(1).NumSRSSymbols = 1; % Number of SRS symbols in a slot (1,2,4)
srs(1).SymbolStart = 12; % Time-domain position of the SRS in the slot
```

**Second SRS configuration:**

```
srs(2) = srs(1); % Create a copy of the configured SRS
srs(2).NumSRSSymbols = 2;
srs(2).SymbolStart = 9;
```

This figure displays the OFDM resource grid containing the SRS transmissions.

```
hSRSGrid(carrier,srs, 2, true); % Generate and display a 2-slot resource grid containing SRS
title('SRS with Different Symbol Locations (SymbolStart)');
ylim([0 srs(1).NRBPerTransmission]);
```



### Frequency-Domain Orthogonal SRS

You can create frequency-domain orthogonal SRS transmissions in multiple ways:

- Configure different comb offsets for different SRS by using the property `KBarTC`.
- Configure different frequency-hopping patterns using `CSRS`, `BSRS`, `BHop` and `NRRC`.

Configure a 10 MHz bandwidth carrier with 15 kHz SCS.

```
carrier = nrCarrierConfig;
carrier.NSizeGrid = 52;
carrier.SubcarrierSpacing = 15 kHz;
```

Specify the comb numbers and offsets to create orthogonal SRS transmissions.

### First SRS configuration:

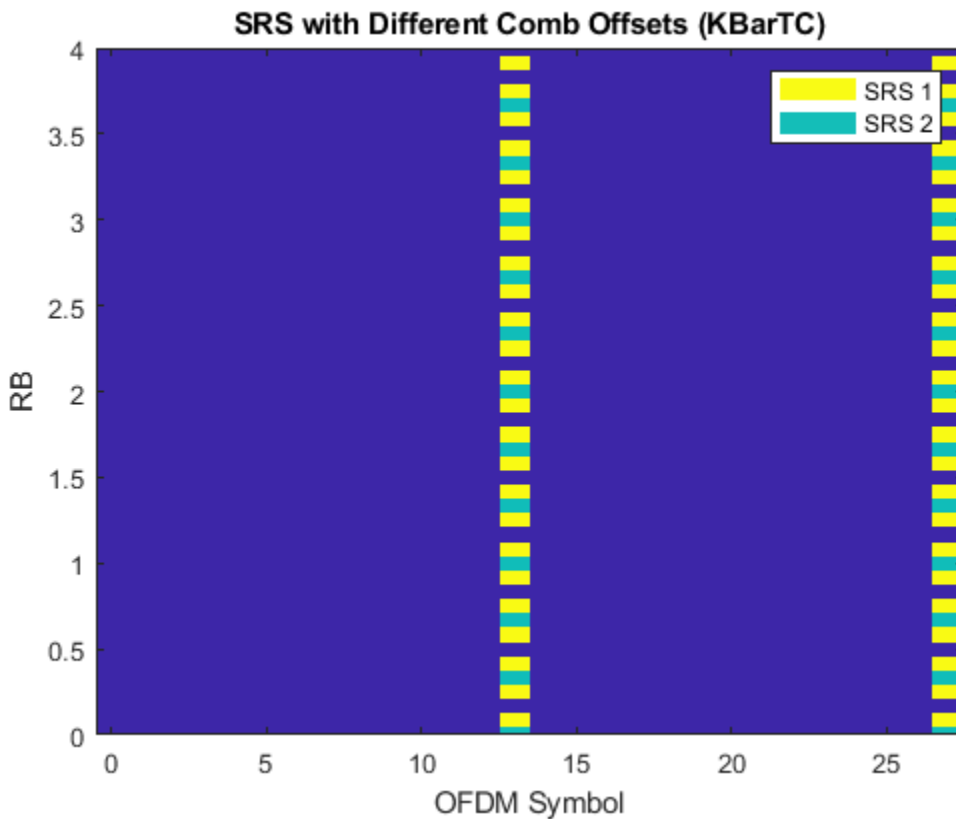
```
srs = nrSRSSConfig;
srs(1).KTC = 2; % Comb number (2,4). It indicates the allocation of the S
srs(1).KBarTC = 1; % Comb offset (0...KTC-1)
```

**Second SRS configuration:**

```
srs(2) = srs(1); % Create a copy of the configured SRS
srs(2).KTC = 4;
srs(2).KBarTC = 0;
```

This figure displays the OFDM resource grid containing the SRS transmissions.

```
hSRSGrid(carrier,srs,2,true);
title('SRS with Different Comb Offsets (KBarTC)')
ylim([0 srs(1).NRBPerTransmission]);
```








Create frequency-domain orthogonal SRS configurations using different frequency-hopping patterns.

```
srs = nrSRSSConfig;
srs.NumSRSSymbols = 4; % Number of SRS symbols in a slot
srs.SymbolStart = 8; % Allocate SRS in OFDM symbols 10:13
```

Select the desired bandwidth configuration parameters, resource type and repetition factor.

```

srs.CSRS = 10 ; % Bandwidth configuration C_SRS (0...63)
srs.BSRS = 2 ; % Bandwidth configuration B_SRS (0...3)
srs.BHop = 0 ; % Frequency hopping configuration (0...3). Set BHop >= BSRS
srs.ResourceType = periodic ; % ('periodic', 'semi-persistent', 'aperiodic')
srs.Repetition = 1 ; % Number of equal consecutive SRS transmissions (1,2,4)

```

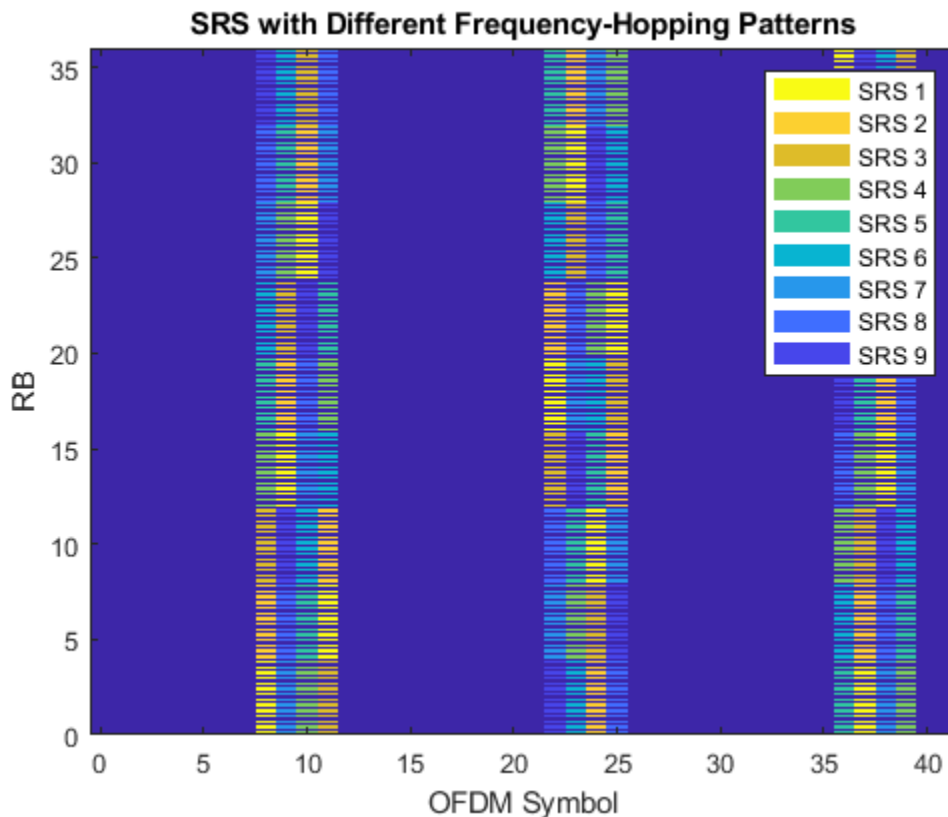
This example calculates the number of orthogonal SRS sequences that can be configured by NRRC and creates frequency non-overlapping SRS configurations. The figure displays the OFDM resource grid containing the SRS transmissions.

```

NRRC = num2cell(hNRRCSet(srs));
srs(2:length(NRRC)) = srs(1); % Create N-1 copies of the configured SRS
[srs(:).NRRC] = deal(NRRC{:}); % Assign the appropriate NRRC to each SRS configuration

hSRSGrid(carrier,srs, 3, true); % Generate and display a 3-slot resource grid containing SRS
title('SRS with Different Frequency-Hopping Patterns');
ylim([0 srs(1).NRBPerTransmission*hSRSNumberofSubbandsOrHoppingPatterns(srs(1))]);

```



```

N = hSRSNumberofSubbandsOrHoppingPatterns(srs(1));
if srs(1).BHop < srs(1).BSRS % Frequency-hopping cases
    fprintf('Number of unique frequency-hopping patterns (configurable by NRRC < %d): %d.', N*srs(1).NRBPerTransmission, N);
else

```

```
    fprintf('Number of unique subbands (configurable by NRRC < %d): %d.', N*srs(1).NRBPerTransm.
end
```

Number of unique frequency-hopping patterns (configurable by NRRC < 9): 9.

Note that the SRS transmissions are never overlapping regardless of the Repetition factor and ResourceType (aperiodic disables inter-slot frequency hopping). Disable frequency hopping by setting BHop >= BSRS and observe that the different SRS are still allocated to different subbands.

### Cyclic-Shift Orthogonal SRS

This section generates multiple SRS allocated to the same time and frequency resources (OFDM symbols and subcarriers) but different time-domain cyclic shifts. Due to the properties of Zadoff-Chu sequences, this configuration produces orthogonal SRS. To demonstrate the orthogonality among the configured SRS, this section performs CP-OFDM modulation and calculates the cross-correlation of the time-domain waveforms.

```
% Configure a 10 MHz bandwidth carrier with 15 kHz SCS.
```

```
carrier = nrCarrierConfig;
carrier.NSizeGrid = 52;
carrier.SubcarrierSpacing = 15;
```

```
% Create a full-band SRS
```

```
srs = nrSRSConfig;
srs.CSRS = hSRSSBandwidthConfiguration(srs,carrier.NSizeGrid);
```

```
% All SRS share the same physical resources, but they are configured with
% different cyclic shifts.
```

```
for i = 1:8
    srs(i) = srs(1);
    srs(i).CyclicShift = i-1;
end
```

```
% Create a resource grid containing SRS
```

```
numSlots = 1; % Number of slots to generate
for ich = length(srs):-1:1
    slotGrid{ich} = hSRSGrid(carrier,srs(ich),numSlots);
end
```

```
% Get OFDM modulation related information
```

```
ofdmInfo = nrOFDMInfo(carrier);
```

```
% OFDM modulation
```

```
nsrs = length(srs); % Number of SRS waveforms
numSamples = numSlots*ofdmInfo.SampleRate/1000/carrier.SlotsPerSubframe;
txWaveform = zeros(numSamples,nsrs);
for i = 1:nsrs
    txWaveform(:,i) = nrOFDMModulate(carrier,slotGrid{i});
end
```

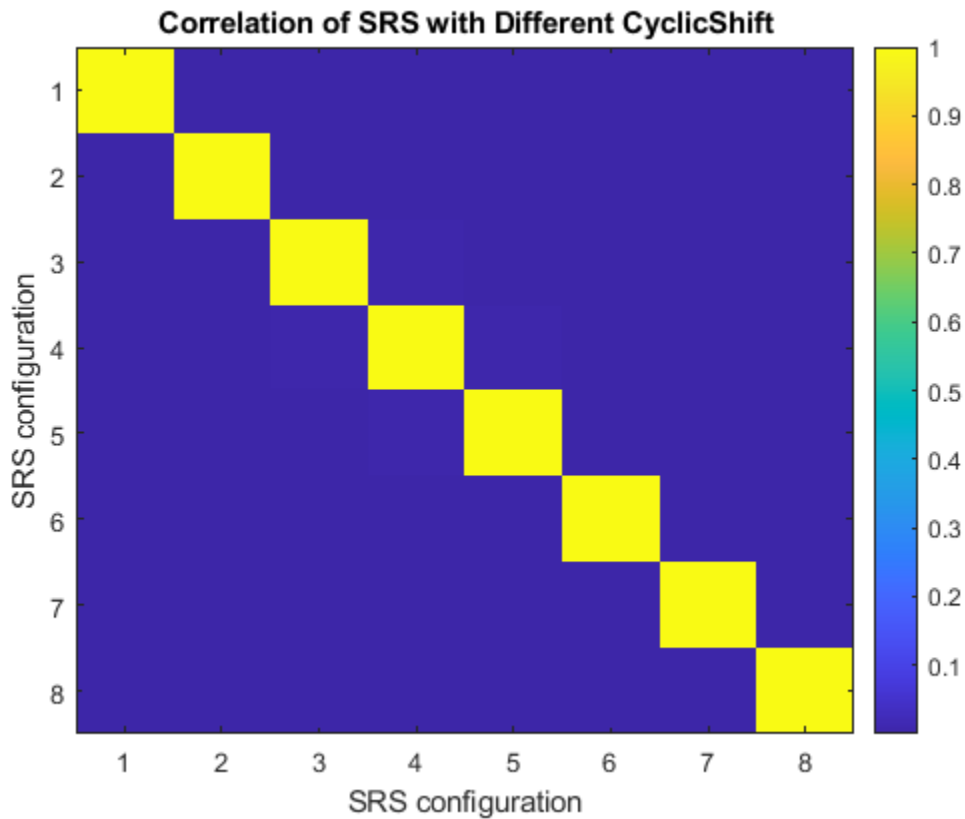
```
% Cross correlation of SRS waveforms generated with different cyclic shifts
```

```
C = txWaveform'*txWaveform;
srsCorr = C./diag(C);
```

This figure shows the time-domain cross-correlation of the SRS waveforms with different cyclic shifts.

```
imagesc(abs(srsCorr))
title('Correlation of SRS with Different CyclicShift')
```

```
xlabel('SRS configuration'); ylabel('SRS configuration');
colorbar
```



The low correlations among SRS waveforms generated using different time-domain cyclic shifts show their orthogonality.

```
disp('Absolute value of correlation matrix: '); disp(' '); disp(abs(srsCorr));
```

Absolute value of correlation matrix:

Columns 1 through 7

1.0000	0.0027	0.0021	0.0008	0.0003	0.0010	0.0007
0.0027	1.0000	0.0030	0.0006	0.0006	0.0001	0.0006
0.0021	0.0030	1.0000	0.0041	0.0007	0.0008	0.0003
0.0008	0.0006	0.0041	1.0000	0.0040	0.0004	0.0009
0.0003	0.0006	0.0007	0.0040	1.0000	0.0039	0.0002
0.0010	0.0001	0.0008	0.0004	0.0039	1.0000	0.0039
0.0007	0.0006	0.0003	0.0009	0.0002	0.0039	1.0000
0.0038	0.0005	0.0008	0.0000	0.0009	0.0003	0.0039

Column 8

```
0.0038
0.0005
0.0008
0.0000
0.0009
```

```
0.0003
0.0039
1.0000
```

### Summary and Further Exploration

This example describes how to generate and map SRS sequences into an OFDM carrier resource grid, and how to generate the corresponding waveform for multiple carrier and SRS configurations. The example highlights the relationship between SRS configuration parameters and their effects on both the resource grid and SRS waveform properties. For example:

- The bandwidth allocated to the SRS generally increases with CSRS and decreases with BSRS. For frequency-hopping configurations, the instantaneous bandwidth and hopping bandwidth decrease with BSRS and BHop, respectively.
- The frequency position of the SRS depends on the FrequencyStart and NRRC parameters. NRRC allows to select different SRS subbands and frequency-hopping patterns.
- BHop < BSRS generally produces frequency hopping, but it is not guaranteed. However, BHop >= BSRS is sufficient to disable frequency hopping.
- Inter-slot frequency hopping is active only for periodic and semi-persistent resource types.
- Time-, frequency-, and sequence-related parameters can be used to create orthogonal SRS transmissions.

The example also demonstrates how to extract useful information from both the SRS bandwidth configuration table and read-only properties, such as, the number of unique SRS subbands and frequency-hopping patterns configurable by NRRC.

The SRS sequence generation for multiport configurations employs the orthogonalization mechanism presented in Cyclic-Shift Orthogonal SRS on page 2-0 . To verify the orthogonality across multiple ports, configure a multi-port SRS, generate the symbols and indices, map the symbols into a resource grid, generate the SRS waveforms, and compute the cross-correlation between waveforms generated for different ports.

### References

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

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

### Local functions

This example uses these local functions:

```
function [csrs,bsrs] = hSRSSBandwidthConfiguration(srs, NRB)
% [CSRS,BSRS] = hBandwidthConfiguration(SRS, NRB) returns the SRS
% bandwidth configuration parameters CSRS and BSRS required to transmit an
% SRS in a bandwidth specified by NRB. The function calculates CSRS and
% BSRS considering the SRS properties FrequencyStart and NRRC, so the
% available bandwidth NRB is reduced by the frequency origin of the SRS.
% For frequency hopping cases, the value of BSRS returned is equal to that
% of the BSRS property in the input SRS configuration object.

f0 = hSRSSFrequencyOrigin(srs);
```



```

NRB = NRB - f0;
if NRB < 4
    error('The available bandwidth is not sufficient to allocate an SRS transmission. Increase bandwidth');
end

% For frequency hopping configurations
if srs.BHop >= srs.BSRS && nargout == 2 % No frequency hopping and BSRS is requested
    BSRS = 0:3;
else
    BSRS = 0;
end

% Initialize gap between SRS frequency allocation and carrier bandwidth
NRBGap = NRB;

% Find the appropriate CSRS for each BSRS that minimizes the gap only
% in non-hopping cases. For freq. hopping, find the value of CSRS only.
for b = BSRS
    % NRB allocated to the SRS for BSRS = b and all CSRS
    srsNRBb = srs.BandwidthConfigurationTable{:,2*(b+1)};
    mSRSbMax = max( srsNRBb( srsNRBb <= NRB ) );

    % Calculate gap between SRS allocation and carrier bandwidth
    gap = NRB - mSRSbMax;
    if gap < NRBGap
        csrs = srs.BandwidthConfigurationTable{ srsNRBb == mSRSbMax ,1};
        bsrs = b;
        NRBGap = gap;
    end
end
csrs = csrs(1);

if srs.BHop < bsrs
    bsrs = srs.BSRS;
end
end

function out = hSRSNumberOfSubbandsOrHoppingPatterns(srs)
% N = hSRSNumberOfSubbandsOrHoppingPatterns(SRS) returns the number of
% unique SRS subbands or frequency-hopping patterns N configurable by the
% SRS property NRRC. An SRS subband is the frequency band allocated for SRS
% in a given OFDM symbol (See SRS property NRBPerTransmission). N is
% calculated using TS 38.211 Table 6.4.1.4.3-1 for the bandwidth
% configuration parameters CSRS, BSRS, and BHop specified in the SRS
% configuration object SRS.

    bwct = nrSRSConfig.BandwidthConfigurationTable{:, :};
    if srs.BHop < srs.BSRS % Number of unique frequency-hopping patterns
        b0 = srs.BHop+1;
    else % Number of unique SRS subbands
        b0 = 0;
    end
    out = prod(bwct(srs.CSRS+1,2*(b0:srs.BSRS)+3));
end

function [Grid,dispGrid] = hSRSGrid(carrier,srs,Duration,displayGrid,chplevels)
% [GRID,DISPGRID] = hSRSGrid(CARRIER,SRS,DURATION,DISPLAYGRID,CHPLEVELS)
% returns a multi-slot OFDM resource grid GRID containing a set of sounding

```

```

% reference signals in a carrier, as specified by the configuration objects
% CARRIER and SRS. This function also returns a scaled version of the grid
% used for display purposes. The optional input DURATION (Default 1)
% specifies the number of slots of the generated grid. The resource grid
% can be displayed using the optional input DISPLAYGRID (Default false).
% CHPLEVELS specifies the channel power levels for display purposes only
% and it must be of the same size as SRS.

numSRS = length(srs);
if nargin < 5
    chplevels = 1:-1/numSRS:1/numSRS;
    if nargin < 4
        displayGrid = false;
        if nargin < 3
            Duration = 1;
        end
    end
end
end

SymbolsPerSlot = carrier.SymbolsPerSlot;
emptySlotGrid = nrResourceGrid(carrier,max([srs(:).NumSRSPorts])); % Initialize slot grid

% Create the SRS symbols and indices and populate the grid with the SRS symbols
Grid = repmat(emptySlotGrid,1,Duration);
dispGrid = repmat(emptySlotGrid,1,Duration); % Frame-size grid for display
for ns = 0:Duration-1
    carrier.NSlot = ns;
    slotGrid = emptySlotGrid;
    dispSlotGrid = emptySlotGrid; % Slot-size grid for display
    for ich = 1:numSRS
        srsIndices = nrSRSIndices(carrier,srs(ich));
        srsSymbols = nrSRS(carrier,srs(ich));
        slotGrid(srsIndices) = srsSymbols;
        dispSlotGrid(srsIndices) = chplevels(ich)*srsSymbols; % Scale the SRS for display on
    end
    OFDMSymIdx = ns*SymbolsPerSlot + (1:SymbolsPerSlot);
    Grid(:,OFDMSymIdx) = slotGrid;
    dispGrid(:,OFDMSymIdx) = dispSlotGrid;
end

if displayGrid
    plotGrid(dispGrid,chplevels,"SRS " + (1:numSRS)');
end
end

function varargout = plotGrid(Grid,chplevels,leg)
% plotGrid(GRID, CHPLEVEL,LEG) displays a resource grid GRID containing
% channels or signals at different power levels CHPLEVEL and create a
% legend for these using a cell array of character vector LEG

if nargin < 3
    leg = {'SRS'};
    if nargin < 2
        chplevels = 1;
    end
end
end

cmap = colormap(gcf);

```

```

chpscale = length(cmap); % Scaling factor

h = figure;
image(0:size(Grid,2)-1,(0:size(Grid,1)-1)/12,chpscale*abs(Grid(:,:,1))); % Multiplied with s
axis xy;

title('Carrier Grid Containing SRS')
xlabel('OFDM Symbol'); ylabel('RB');

clevels = chpscale*chplevels(:);
N = length(clevels);
L = line(ones(N),ones(N), 'LineWidth',8); % Generate lines

% Index the color map and associate the selected colors with the lines
set(L,{'color'},mat2cell(cmap( min(1+fix(clevels),length(cmap) ),:),ones(1,N),3)); % Set the

% Create legend
legend(leg(:));

if nargout > 0
    varargout = {h};
end
end

function hSRSAnnotations(carrier,srs)
% hSRSAnnotations(carrier,srs) adds annotation to the current figure
% indicating the frequency origin of the SRS and the number of RB used per
% OFDM symbol for the configuration objects CARRIER and SRS.

% Calculate the frequency origin of the first SRS symbol
f0 = hSRSFrequencyOrigin(srs);

hold on;
hfig =(gcf);
set(hfig,'Units','Normalized');
Sym0 = srs.SymbolStart-0.5;
if isnumeric(srs.SRSPeriod)
    Sym0 = srs.SRSPeriod(2)*carrier.SymbolsPerSlot + srs.SymbolStart-0.5;
end

IP = get(gca,'Position');

% Y-coordinate in the current axes of the SRS freq position f0
Yf0 = f0/carrier.NSizeGrid*IP(4)+IP(2);
Xc = Sym0/((carrier.NSlot+1)*carrier.SymbolsPerSlot)*IP(3)+IP(1);

% Add annotation to the figure including f0 in RB
if f0/carrier.NSizeGrid > 0.08 % Only plot f0 when there is enough space in the y-axis
    % Create doublearrow for F0
    YMin = IP(2);
    annotation(hfig,'doublearrow',Xc*[1 1], [YMin Yf0],...
        'Color',[1 0.4 0.15]);

    % Text for F0
    str = sprintf('$$F_0 = %d $$ RB', f0);
    Ystr = f0/carrier.NSizeGrid/2;
    text(gca,(Xc-IP(1))/IP(3)-0.25, Ystr,str,...
        'Color',[1 0.4 0.15],'FontSize',14, ...

```

```

        'Units','Normalized','Interpreter','latex');
end

% Create doublearrow from f0 and spanning the SRS bandwidth
Yf1 = Yf0 + srs.NRBPerTransmission/carrier.NSizeGrid*IP(4);
annotation(hfig,'doublearrow',Xc*[1 1], [Yf0 Yf1],...
    'Color',[1 0.4 0.15]);

% Text for NRBT
str = sprintf('\textrm{NRB}_T = %d $$ RB', srs.NRBPerTransmission);
Ystr = (f0 + 0.5*srs.NRBPerTransmission)/carrier.NSizeGrid;
text(gca,(Xc-IP(1))/IP(3)-0.32, Ystr ,str,...
    'Color',[1 0.4 0.15],'FontSize',14, ...
    'Units','Normalized','Interpreter','latex');
end

function f0 = hSRSSFrequencyOrigin(srs)
% Calculate the frequency origin of the first SRS symbol in a slot

    NSBTable = hSRSNumberOfSubbandsOrHoppingPatterns(srs);
    NRBT = srs.NRBPerTransmission;

    % Origin of the SRS in frequency in RB
    f0 = srs.FrequencyStart + NRBT*mod(floor(4*srs.NRRC/NRBT),NSBTable);
end

function [NRRC,NRB] = hNRRCSet(srs)
% Calculate the values of NRRC that generate a unique set of orthogonal SRS in frequency
    if srs.BHop < srs.BSRS % Frequency-hopping cases
        NRB = srs(1).NRB; % Hopping bandwidth
    else
        NRB = nrSRSConfig.BandwidthConfigurationTable{srs(1).CSRS+1,2};
    end

    % Number of frequency-hopping patterns or SRS subbands depending on the values of BSRS and B
    N = hSRSNumberOfSubbandsOrHoppingPatterns(srs);

    NRRC = NRB/4*(0:N-1)/N;
end

```

## References

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

## See Also

### Functions

nrSRS | nrSRSIndices

### Objects

nrSRSConfig

**Related Examples**

- “NR Uplink Channel State Information Estimation Using SRS” on page 2-68

## NR Uplink Channel State Information Estimation Using SRS

This example shows how to use sounding reference signals (SRS) for synchronization, channel estimation, and uplink channel state information (CSI) estimation.

### Introduction

Sounding reference signals are uplink physical signals used by user equipment (UE) for uplink channel sounding, including synchronization and CSI estimation. CSI comprises channel quality indicator (CQI), rank indicator (RI), and precoder matrix indicator (PMI). This example demonstrates how to use SRS to select the appropriate PMI under frequency-selective time-varying noisy channels. Uplink codebook-based transmissions use PMIs, as defined in TS 38.211 Section 6.3.1.5 [ 1 ].

This example performs a simulation including:

- SRS configuration and transmission
- Perfect and practical synchronization and channel estimation
- Signal-to-noise ratio (SNR) estimation
- PMI selection
- PMI selection performance assessment

### Simulation Length and SNR

Set the length of the simulation in terms of the number of 10 ms frames. Set the SNR points to simulate. The SNR is defined per RE and applies to each receive antenna.

```
numFrames = 1; % 10 ms frames
snr = 20; % SNR in dB
```

### UE and SRS Configuration

Set the key parameters of the simulation. These include:

- The bandwidth in resource blocks (12 subcarriers per resource block)
- Subcarrier spacing: 15, 30, 60, 120, 240 (kHz)
- Cyclic prefix length: normal or extended
- Number of transmit and receive antennas: 1, 2 or 4.
- Number of layers. It must be lower than or equal to the number of transmit and receive antennas.

The SRS parameters specified include:

- Number of SRS antenna ports: 1,2,4
- Number of OFDM symbols allocated for SRS per slot: 1,2,4
- Starting OFDM symbol of the SRS transmission within a slot. It must be (8...13) for normal CP and (6...11) for extended CP.
- Starting position of the SRS in frequency specified in RBs
- Bandwidth and frequency hopping configuration CSRS, BSRS, and BHop. Set BHop  $\geq$  BSRS to disable frequency hopping.
- Transmission comb to specify the SRS frequency density in subcarriers: 2,4

- Number of repeated SRS symbols within a slot. It disables frequency hopping in blocks of Repetition symbols. Set Repetition = 1 for no repetition.
- Periodicity and offset of the SRS in slots.
- Resource type can be 'periodic', 'semi-persistent', and 'aperiodic'. The frequency hopping pattern is reset for aperiodic SRS resource types in every slot.

```

% Create UE/carrier configuration
ue = nrCarrierConfig;
ue.NSizeGrid = 52;           % Bandwidth in number of resource blocks (52RBs at 15kHz SCS for 10MHz)
ue.SubcarrierSpacing = 15;  % 15, 30, 60, 120, 240 (kHz)
ue.CyclicPrefix = 'Normal'; % 'Normal' or 'Extended'

nTxAnts = 2; % Number of transmit antennas (1,2,4)
nRxAnts = 2; % Number of receive antennas
nLayers = min(nTxAnts,nRxAnts);

% Configure a periodic multi-port SRS and enable frequency hopping
srs = nrSRSConfig;
srs.NumSRSSymbols = 4;      % Number of OFDM symbols allocated per slot (1,2,4)
srs.SymbolStart = 8;       % Starting OFDM symbol within a slot
srs.NumSRSPorts = nTxAnts; % Number of SRS antenna ports (1,2,4).
srs.FrequencyStart = 0;    % Frequency position of the SRS in BWP in RBs
srs.NRRC = 0;              % Additional offset from FreqStart specified in blocks of 4 PRBs
srs.CSRS = 14;             % Bandwidth configuration C_SRS (0...63). It controls the allocated
srs.BSRS = 0;              % Bandwidth configuration B_SRS (0...3). It controls the allocated
srs.BHop = 0;              % Frequency hopping configuration (0...3). Set BHop < BSRS to enable
srs.KTC = 2;               % Comb number (2,4). Frequency density in subcarriers
srs.Repetition = 2;        % Repetition (1,2,4). It disables frequency hopping in blocks of
srs.SRSPeriod = [2 0];    % Periodicity and offset in slots. SRSPeriod(2) must be < SRSPer
srs.ResourceType = 'periodic'; % Resource type ('periodic', 'semi-persistent','aperiodic'). Use

```

### Synchronization, Channel Estimation and CSI Measurement Configuration

This example performs synchronization and channel estimation in SRS candidate slots. Timing and channel estimates are updated only in slots containing SRS transmissions. In frequency-hopping SRS setups, channel estimates are only updated in those resource blocks containing SRS symbols. When there is no SRS transmission, timing and channel estimates from previous slots are held and used for CSI acquisition. Similarly, noise power estimates are updated only in SRS candidate slots.

The logical variable `practicalSynchronization` controls channel synchronization behavior. When set to `true`, the example performs practical synchronization based on the values of the received SRS. When set to `false`, the example performs perfect synchronization. Synchronization is performed only in slots where the SRS is transmitted to keep perfect and practical channel estimates synchronized.

```
practicalSynchronization = true;
```

This example estimates CSI by splitting the carrier bandwidth into a number of subbands. Specify the size of the frequency subbands in RB

```
csiSubbandSize = 4; % Number of RBs per subband
```

### Propagation Channel Model Configuration

Create a TDL channel model object and specify its propagation characteristics. Select the channel delay spread and maximum Doppler shift to create a time-varying and frequency-selective channel within the simulation duration and carrier bandwidth.

```

channel = nrTDLChannel;
channel.DelayProfile = 'TDL-C';
channel.DelaySpread = 40e-9;
channel.MaximumDopplerShift = 30;
channel.NumTransmitAntennas = nTxAnts;
channel.NumReceiveAntennas = nRxAnts;
channel.Seed = 5;

% Set channel sample rate
ofdmInfo = nrOFDMInfo(ue);
channel.SampleRate = ofdmInfo.SampleRate;

```

Get the maximum number of delayed samples by a channel multipath component. This is calculated from the channel path with the largest delay and the implementation delay of the channel filter. This is required later to flush the channel filter to obtain the received signal.

```

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

% Reset random generator for reproducibility
rng('default');

```

### Processing Loop

Measure the CSI per slot. The CSI is obtained using the following steps:

- *Generate resource grid.* Use SRS symbols and indices to create a resource element (RE) grid.
- *Generate waveform.* The generated grid is then OFDM modulated.
- *Model noisy channel.* The waveform is passed through a TDL fading channel. AWGN is added. The SNR for each layer is defined per RE and per receive antenna.
- *Perform synchronization and OFDM demodulation.* For perfect synchronization, the channel impulse response is reconstructed and used to synchronize the received waveform. For practical synchronization, the received waveform is correlated with the SRS. The synchronized signal is then OFDM demodulated.
- *Perform channel estimation.* For perfect channel estimation, the channel impulse response is reconstructed and OFDM demodulated to provide a channel estimate. For practical channel estimation, the transmitted SRS is used.
- *PMI selection.* The SRS-based channel estimate is used to select the best PMI in each CSI estimation subband. The PMI selection criterion maximizes the average signal-to-interference-plus-noise-ratio (SINR) after precoding.
- *PMI selection SINR loss.* The SINR loss is calculated by comparing the SINR after precoding with the estimated and ideal PMIs. Ideal PMIs are selected using perfect channel estimates.

```

% Number of slots to simulate
numSlots = numFrames*ue.SlotsPerFrame;

% Total number of subcarriers and symbols per slot
K = ue.NSizeGrid * 12;
L = ue.SymbolsPerSlot;

% Initialize arrays storing channel estimates
allTxGrid = zeros([K L*numSlots nTxAnts]);

```



```

slotGridSize = [K L nRxAnts nTxAnts];
hEst = zeros(slotGridSize);
hestInterp = zeros(slotGridSize);
hEstUpdate = zeros(slotGridSize);

totalGridSize = [K L*numSlots nRxAnts nTxAnts];
allHest = zeros(totalGridSize);
allHestPerfect = zeros(totalGridSize);
allHestInterp = zeros(totalGridSize);

% Initialize noise power estimate
nvar = 0;

% Calculate the number of CSI subbands for the carrier
numCSISubbands = ceil(ue.NSizeGrid/csiSubbandSize);

% Initialize SINR per subband, slot, and PMI
maxPMI = hMaxPUSCHPrecodingMatrixIndicator(nLayers,nTxAnts);
sinrSubband = zeros([numCSISubbands numSlots maxPMI+1]);

% Initialize PMI matrix and SINR loss
pmi = NaN(numCSISubbands,numSlots);
pmiPerfect = pmi;
loss = zeros(size(pmi));

% Initialize timing estimation offset
offset = chInfo.ChannelFilterDelay;

% Calculate SRS CDM lengths
cdmLengths = hSRSCDMLengths(srs);

% OFDM symbols used for CSI acquisition
csiSelectSymbols = srs.SymbolStart + (1:srs.NumSRSSymbols);

for nSlot = 0:numSlots-1

    % Update slot counter
    ue.NSlot = nSlot;

    % Generate SRS and map to slot grid
    [srsIndices,srsIndInfo] = nrSRSIndices(ue,srs);
    srsSymbols = nrSRS(ue,srs);

    % Create a slot-wise resource grid empty grid and map SRS symbols
    txGrid = nrResourceGrid(ue,nTxAnts);
    txGrid(srsIndices) = srsSymbols;

    % Determine if the slot contains SRS
    isSRSSlot= ~isempty(srsSymbols);

    % OFDM Modulation
    [txWaveform,waveformInfo] = nrOFDMModulate(ue,txGrid);

    txWaveform = [txWaveform; zeros(maxChDelay, size(txWaveform,2))]; % required later to flush

    % Transmission through channel
    [rxWaveform,pathGains] = channel(txWaveform);

```

```

% Add AWGN to the received time domain waveform
% Normalize noise power to take account of sampling rate, which is
% a function of the IFFT size used in OFDM modulation. The SNR
% is defined per RE for each receive antenna (TS 38.101-4).
SNR = 10^(snr/20); % Calculate linear noise gain
N0 = 1/(sqrt(2.0*nRxAnts*double(waveformInfo.Nfft))*SNR);
noise = N0*complex(randn(size(rxWaveform)),randn(size(rxWaveform)));

rxWaveform = rxWaveform + noise;

% Perform timing offset estimation
pathFilters = getPathFilters(channel);

% Timing estimation is only performed in the slots where the SRS is
% transmitted to keep the perfect and practical channel estimation
% synchronized.
if isSRSSlot
    if practicalSynchronization
        % Practical synchronization. Correlate the received waveform
        % with the SRS to give timing offset estimate
        offset = nrTimingEstimate(ue,rxWaveform,srsIndices,srsSymbols);
    else
        offset = nrPerfectTimingEstimate(pathGains,pathFilters);
    end
end

% Perform OFDM demodulation
rxGrid = nrOFDMDemodulate(ue,rxWaveform(1+offset:end,:));

% Perform practical channel estimation
% Update channel estimates only in the symbols and RBs containing SRS
% in this slot and hold the estimates from the previous slot in other
% locations. nvar is not updated when there is no SRS transmission. Use
% a time-averaging window that covers all the SRS symbols transmitted.
if isSRSSlot % this slot contains an SRS transmission
    [hEst,nvar] = nrChannelEstimate(ue,rxGrid,srsIndices,srsSymbols,'AveragingWindow',[0 7],
    % Use channel estimate from previous slot for OFDM symbols before the first SRS symbol
    hestInterp = repmat(hestInterp(:,end,:,:),1,ue.SymbolsPerSlot);

    % Update channel estimate in OFDM symbols and RB where the SRS is
    % present and hold all channel estimates until the end of the slot
    firstSymbol = srs.SymbolStart+1;
    lastSymbol = srs.SymbolStart + srs.NumSRSSymbols;
    hEstUpdate(:,firstSymbol:lastSymbol,:,:) = hEst(:,firstSymbol:lastSymbol,:,:);
    hEstUpdate(:,lastSymbol:L,:,:) = repmat(hEst(:,lastSymbol,:,:),1,ue.SymbolsPerSlot-lastS

    idxHEstUpdate = hEstUpdate ~= 0; % Indices of updated channel estimates
    hestInterp(idxHEstUpdate) = hEstUpdate(idxHEstUpdate);
else % Hold previous channel estimates if this slot does not contain SRS
    hestInterp = repmat(hestInterp(:,end,:,:),1,ue.SymbolsPerSlot);
end

% PMI Selection
% Select the precoder matrix indicators for a number of layers using
% the interpolated channel estimates in the OFDM symbols specified by
% csiSelectSymbols. The PMIs are estimated per CSI subband
[pmi(:,nSlot+1),sinrSubband(:,nSlot+1,:),subbandIndices] = hPMISelect(nLayers, hestInterp(:,

```

```

% PMI selection SINR loss
% Calculate the performance loss as a ratio of the SINR after precoding
% with PMIs selected using a practical channel estimate and the SINR
% after precoding with PMIs selected using a perfect channel estimate.

% Calculate perfect channel estimate for perfect PMI selection
hEstPerfect = nrPerfectChannelEstimate(ue,pathGains,pathFilters,offset);

% Perfect noise estimate from noise realization
noiseGrid = nrOFDMDemodulate(ue,noise(1+offset:end,:));
nvarPerfect = var(noiseGrid(:));

[loss(:,nSlot+1),pmiPerfect(:,nSlot+1)] = hPMISelectionSINRLoss(pmi(:,nSlot+1), nLayers, hEstPerfect);

% Save a copy of all transmitted OFDM grids and channel estimates for
% display purposes
thisSlot = nSlot*L + (1:L); % Symbols of the current slot
allTxGrid(:,thisSlot,:) = txGrid;
allHest(:,thisSlot,:,:) = hEst;
allHestInterp(:,thisSlot,:,:) = hestInterp;
allHestPerfect(:,thisSlot,:,:) = hEstPerfect;

end

```

## Results

This section displays these results for all configured frames:

- Transmitted OFDM grid containing SRS
- Perfect and practical channel estimates, and channel estimation error. The error is calculated as the absolute value of the difference between the perfect and practical channel estimates.
- Selected PMI using perfect and practical channel estimates, and the PMI absolute error.
- Average SINR per subband after precoding with best estimated PMI
- SINR performance loss

```

% Create x-axis and y-axis vectors
symbols = 0:(ue.NSlot+1)*ue.SymbolsPerSlot-1;
slots = 0:ue.NSlot;
subcarriers = 1:ue.NSizeGrid*12;
resourceBlocks = 1:ue.NSizeGrid;

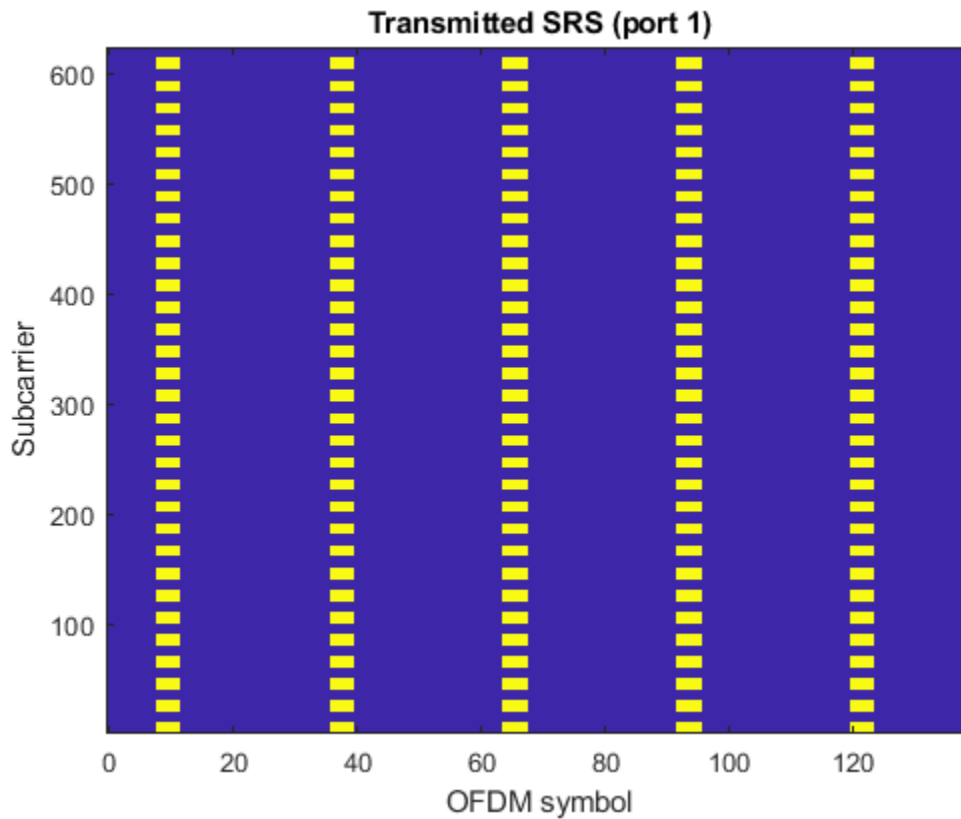
```

Display the transmitted OFDM grid containing SRS

```

figure
imagesc(symbols,subcarriers,abs(allTxGrid(:,:,1,1)));
xlabel('OFDM symbol'); ylabel('Subcarrier'); axis xy;
title('Transmitted SRS (port 1)');

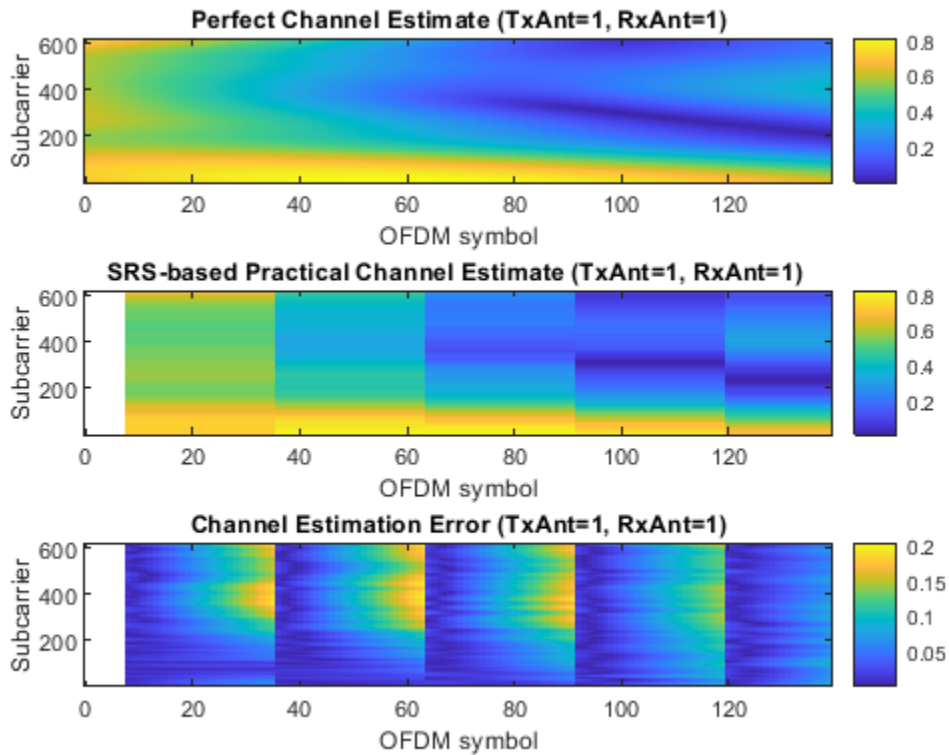
```



Display perfect and practical channel estimates, and channel estimation error per slot and RB. The channel estimation error is defined as the absolute value of the difference between the perfect and practical channel estimates.

```
% Remove first OFDM symbols not containing SRS to improve visualization of
% channel estimation error
hEstInterp = allHestInterp;
idx = 1:(srs.SRSPeriod(2)*ue.SymbolsPerSlot + srs.SymbolStart);
hEstInterp(:,idx, :, :) = NaN;
hEstInterp((srs.NRB*12+1):end, :, :, :) = NaN;

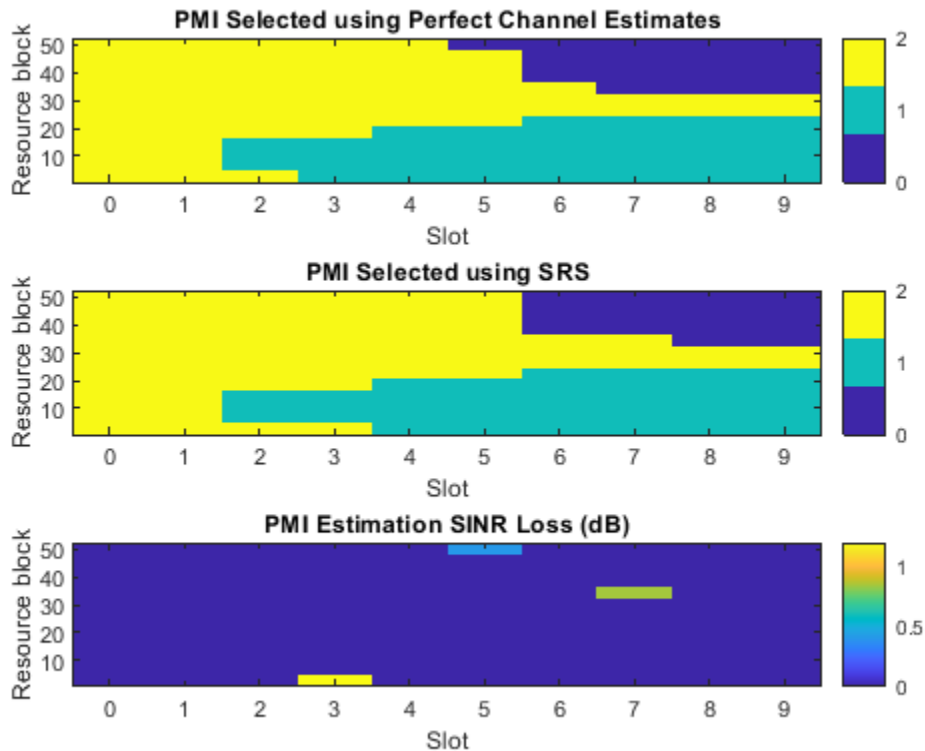
hChannelEstimationPlot(symbols, subcarriers, allHestPerfect, hEstInterp);
```



Display the selected PMI using perfect and practical channel estimates, and the PMI selection SINR loss per slot and RB. The SINR loss is defined as a ratio of the SINR after precoding with estimated and perfect PMIs. Estimated PMIs are obtained using a practical channel estimate and perfect PMIs are selected using a perfect channel estimate.

```
% First expand loss from subbands into RBs for display purposes
pmiRB = hExpandSubbandToRB(pmi, csiSubbandSize, ue.NSizeGrid);
pmiPerfectRB = hExpandSubbandToRB(pmiPerfect, csiSubbandSize, ue.NSizeGrid);
lossRB = hExpandSubbandToRB(loss, csiSubbandSize, ue.NSizeGrid);

hPMIPlot(slots, resourceBlocks, pmiRB, pmiPerfectRB, lossRB);
```



Next, compare the PMIs obtained using both perfect and practical channel estimates. This indicates the ratio of correct to total PMIs and the location in the resource grid where errors have occurred.

```

numLayers = min(size(allHestInterp,[3 4]));
if numLayers ~= 1
    pmiErr = sum( abs(pmi - pmiPerfect) > 0, [1 2])./ sum( ~isnan(pmi), [1 2]);

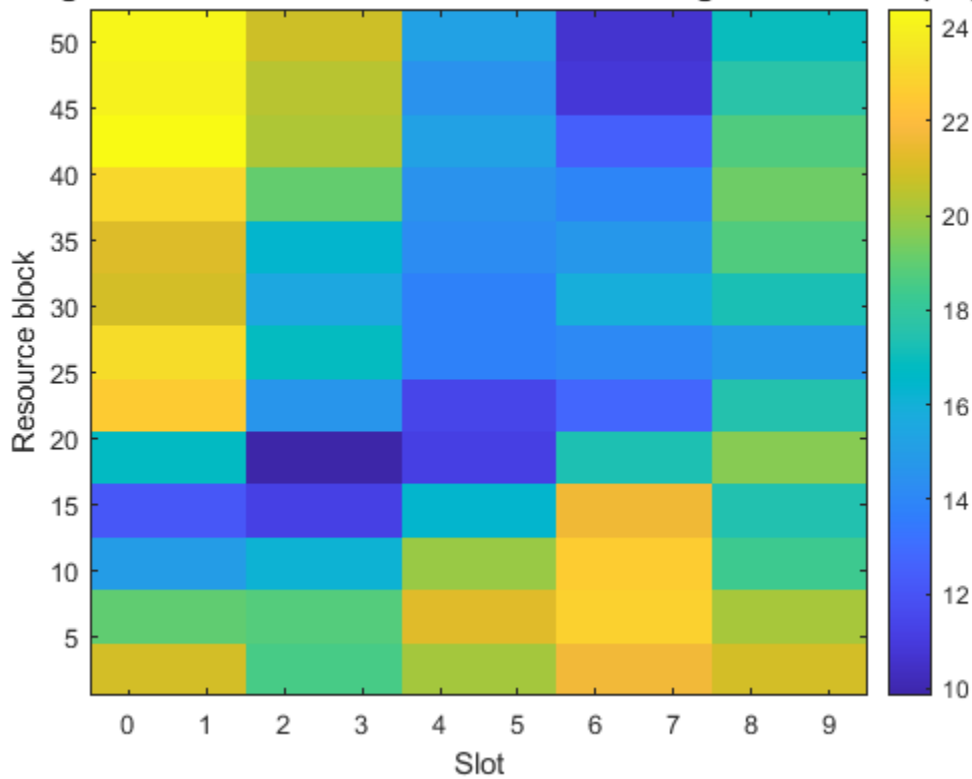
    TotPMIEst = sum(~isnan(pmi),[1 2]);
    fprintf('Number of estimated PMI: %d \n', TotPMIEst);
    fprintf('    Number of wrong PMI: %d \n', ceil(pmiErr*TotPMIEst));
    fprintf('    Relative error: %.1f (%) \n', pmiErr*100);
else
    fprintf('For a single layer, PMI is always 0.\n');
end

Number of estimated PMI: 130
    Number of wrong PMI: 3
    Relative error: 2.3 (%)
    
```

Display the SINR per slot and RB obtained after precoding with the PMI that maximizes the SINR per subband.

```

hBestSINRPlot(slots,resourceBlocks,sinrSubband,pmi,csiSubbandSize);
    
```

**Average SINR Per Subband and Slot After Precoding with Best PMI (dB)**

### Summary and Further Exploration

This example shows how to use SRS for synchronization, channel estimation and PMI selection commonly employed in codebook-based uplink transmission modes. The example also evaluates the channel estimation and PMI selection performance loss using the SINR after precoding.

You can investigate the performance of the channel estimation and PMI selection in more complex settings. Increase the SRS periodicity and observe how the channel estimates aging introduces a delay and worsens the SINR after precoding. In addition, you can investigate the performance under frequency hopping conditions by setting the SRS parameters  $B_{Hop} < B_{SRS}$ . In this setup, channel estimates aging is not uniform in frequency.

### Appendix

This example uses the following helper functions:

- hMaxPUSCHPrecodingMatrixIndicator.m
- hPMISelect.m
- hPMISelectionSINRLoss.m
- hPrecodedSINR.m
- hSINRPerSubband.m
- hSRSCDMLengths.m

## References

- 1 3GPP TS 38.211. "NR; Physical channels and modulation (Release 15)." 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 (Release 15)." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.

## Local functions

```
% Displays perfect and practical channel estimates and the channel
% estimation error for the first transmit and receive ports. The channel
% estimation error is defined as the absolute value of the difference
% between the perfect and practical channel estimates.
function hChannelEstimationPlot(symbols,subcarriers,allHestPerfect,allHestInterp)
```

```
    figure
    subplot(311)
    imagesc(symbols, subcarriers, abs(allHestPerfect(:,:,1,1)));
    axis xy; xlabel('OFDM symbol'); ylabel('Subcarrier');
    colorbar;
    title('Perfect Channel Estimate (TxAnt=1, RxAnt=1)');
```

```
    subplot(312)
    imagesc(symbols, subcarriers, abs(allHestInterp(:,:,1,1)), ...
            'AlphaData',~isnan(allHestInterp(:,:,1,1)))
    axis xy; xlabel('OFDM symbol'); ylabel('Subcarrier');
    colorbar;
    title('SRS-based Practical Channel Estimate (TxAnt=1, RxAnt=1) ');
```

```
% Display channel estimation error, defined as the difference between the
% SRS-based and perfect channel estimates
```

```
    subplot(313)
    hestErr = abs(allHestInterp - allHestPerfect);
    imagesc(symbols, subcarriers, hestErr(:,:,1,1),...
            'AlphaData',~isnan(hestErr(:,:,1,1)));
    axis xy; xlabel('OFDM symbol'); ylabel('Subcarrier');
    colorbar;
    title('Channel Estimation Error (TxAnt=1, RxAnt=1)');
```

```
end
```

```
% Displays the PMI evolution and PMI estimation SINR loss over time and
% frequency. The SINR loss is defined as a ratio of the SINR after
% precoding with estimated and perfect PMIs. Estimated PMIs are
% obtained using a practical channel estimate and perfect PMIs are
% selected using a perfect channel estimate.
```

```
function hPMIPlot(slots,resourceBlocks,pmiRB,pmiPerfectRB,lossRB)
```

```
    figure
    subplot(311)
    imagesc(slots,resourceBlocks,pmiPerfectRB,'AlphaData',~isnan(pmiPerfectRB)); axis xy;
    c = caxis;
    cm = colormap;
    colormap( cm(1:floor(size(cm,1)/(c(2)-c(1)) -1):end,:) ); % Adjust colormap to PMI discrete v
    colorbar
    xlabel('Slot'); ylabel('Resource block'), title('PMI Selected using Perfect Channel Estimates
```



```

subplot(312)
imagesc(slots,resourceBlocks,pmiRB,'AlphaData',~isnan(pmiRB)); axis xy;
colorbar,
xlabel('Slot'); ylabel('Resource block'), title('PMI Selected using SRS')

subplot(313)
imagesc(slots,resourceBlocks,lossRB,'AlphaData',~isnan(lossRB));
colormap(gca,cm)
xlabel('Slot'); ylabel('Resource block'); axis xy; colorbar;
title('PMI Estimation SINR Loss (dB)')

end

% Displays the SINR per resource block obtained after precoding with the
% PMI that maximizes the SINR per subband.
function hBestSINRPlot(slots,resourceBlocks,sinrSubband,pmi,csiBandSize)

    % Display SINR after precoding with best PMI
    bestSINRPerSubband = nan(size(sinrSubband,[1 2]));

    % Get SINR per subband and slot using best PMI
    [sb,nslot] = find(~isnan(pmi));
    for i = 1:length(sb)
        bestSINRPerSubband(sb(i),nslot(i)) = sinrSubband(sb(i),nslot(i),pmi(sb(i),nslot(i))+1);
    end

    % First expand SINR from subbands into RBs for display purposes
    bestSINRPerRB = hExpandSubbandToRB(bestSINRPerSubband, csiBandSize, length(resourceBlocks));

    figure
    sinrdb = 10*log10(abs(bestSINRPerRB));
    imagesc(slots,resourceBlocks,sinrdb,'AlphaData',~isnan(sinrdb));
    axis xy; colorbar;
    xlabel('Slot');
    ylabel('Resource block')
    title('Average SINR Per Subband and Slot After Precoding with Best PMI (dB)')

end

% Expands a 2D matrix of values per subband in the first dimension into a
% matrix of values per resource block.
function rbValues = hExpandSubbandToRB(subbandValues, bandSize, NRB)

    lastBandSize = mod(NRB,bandSize);
    lastBandSize = lastBandSize + bandSize*(lastBandSize==0);

    rbValues = [kron(subbandValues(1:end-1,:),ones(bandSize,1));...
                subbandValues(end,:).*ones(lastBandSize,1)];

end

```

## See Also

### Functions

nrSRS | nrSRSIndices

### **Objects**

nrSRSSConfig

### **Related Examples**

- “NR SRS Configuration” on page 2-48

## 5G NR PRACH Configuration

This example shows how to configure the 5G New Radio (NR) physical random access channel (PRACH), as defined in TS 38.211 Sections 5.3.2 and 6.3.3 [ 1 on page 2-0 ]. You can learn about PRACH time resources, their relation to PRACH preambles, and learn how to generate PRACH preambles without the need to look up configuration tables. This example also shows how to map PRACH symbols to the resource grid, and how to generate a time-domain waveform for a single PRACH preamble.

### Configure Carrier and PRACH

#### Supported Combinations of Subcarrier Spacing

Table 6.3.3.2-1 in TS 38.211 lists the supported combinations of subcarrier spacing for the PRACH and the physical uplink shared channel (PUSCH) during initial access. You can access this table directly from the PRACH configuration object.

```
disp(nrPRACHConfig.Tables.SupportedSCSCCombinations)
```

LRA	PRACHSubcarrierSpacing	PUSCHSubcarrierSpacing	NRBAallocation	kbar
839	1.25	15	6	7
839	1.25	30	3	1
839	1.25	60	2	133
839	5	15	24	12
839	5	30	12	10
839	5	60	6	7
139	15	15	12	2
139	15	30	6	2
139	15	60	3	2
139	30	15	24	2
139	30	30	12	2
139	30	60	6	2
139	60	60	12	2
139	60	120	6	2
139	120	60	24	2
139	120	120	12	2

The system information block 1 (SIB1) contains the radio resource control (RRC) information element *UplinkConfigCommonSIB* (TS 38.331 Section 6.3.2 [ 2 on page 2-0 ]) that defines the subcarrier spacing for the PUSCH. The user equipment (UE) needs this information to transmit the PRACH preamble during the random-access procedure.

### Carrier Configuration

Because the PUSCH is not defined at the PRACH preamble transmission, to configure the PUSCH subcarrier spacing and the frequency-domain dimensions of the resource grid, use the `nrCarrierConfig` object.

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 15; % Subcarrier spacing in kHz (15, 30, 60, 120)
```

Because the PRACH preamble is modulated with respect to the carrier, changing the carrier subcarrier spacing leads to a different PRACH waveform. To see how a different carrier affects the generated waveform, check the Information associated with PRACH OFDM modulation

output for several carriers in the Generate Waveform for Single PRACH Preamble on page 2-0 section, .

### PRACH Configuration

You can configure PRACH parameters by setting property values of the `nrPRACHConfig` object. According to TS 38.211, not all PRACH parameter combinations are valid. For more information on how the properties of `nrPRACHConfig` reflect these limitations, see `nrPRACHConfig`.

```
prach = nrPRACHConfig;
prach.FrequencyRange = 'FR1';           % Frequency range ('FR1', 'FR2')
prach.DuplexMode = 'FDD';               % Duplex mode ('FDD', 'TDD', 'SUL')
prach.ConfigurationIndex = 27;          % Configuration index (0..255). This value is automatic
prach.SubcarrierSpacing = 15;           % Subcarrier spacing in kHz (1.25, 5, 15, 30, 60, 120)
prach.SequenceIndex = 0;                % Logical root sequence index (0..837)
prach.PreambleIndex = 0;                % Scalar preamble index within the cell (0..63)
prach.RestrictedSet = 'UnrestrictedSet'; % Type of restricted set ('UnrestrictedSet', 'Restricted')
prach.ZeroCorrelationZone = 0;          % Cyclic shift configuration index (0..15)
prach.RBOffset = 0;                     % Starting resource block index of the initial uplink block
prach.FrequencyStart = 0;                % Frequency offset of lowest PRACH transmission occasion
prach.FrequencyIndex = 0;                % Index of the PRACH transmission occasions in frequency
prach.TimeIndex = 0;                    % Index of the PRACH transmission occasions in time domain
                                        % For formats B2 and B3, this value is automatically up
prach.ActivePRACHSlot = 0;               % Active PRACH slot number within a subframe or a 60 kHz
prach.NPRACHSlot = 0;                   % PRACH slot number
```

The `ConfigurationIndex` and `TimeIndex` properties depend on the PRACH format. The `SubcarrierSpacing`, `ActivePRACHSlot`, and `NPRACHSlot` properties determine whether the PRACH preamble is active. The next two sections discuss how to set these properties.

### How to Set ConfigurationIndex Based on Preferred Format

Tables 6.3.3.2-2 to 6.3.3.2-4 in TS 38.211 define all possible PRACH configurations in the time domain. The combination of frequency range and duplex mode specifies which configuration table to use. Valid combinations are:

- FR1 and FDD (paired spectrum): Table 6.3.3.2-2
- FR1 and SUL (supplementary uplink): Table 6.3.3.2-2
- FR1 and TDD (unpaired spectrum): Table 6.3.3.2-3
- FR2 and TDD (unpaired spectrum): Table 6.3.3.2-4

For more information on how paired and unpaired spectrums relate to duplex mode, see the field `FDD-OrSUL` of the RRC information element `FrequencyInfoUL` in TS 38.331 Section 6.3.2.

You can access these configuration tables through the `Tables` property of the `nrPRACHConfig` object. For example:

```
nrPRACHConfig.Tables.ConfigurationsFR1PairedSUL % TS 38.211 Table 6.3.3.2-2
nrPRACHConfig.Tables.ConfigurationsFR1Unpaired % TS 38.211 Table 6.3.3.2-3
nrPRACHConfig.Tables.ConfigurationsFR2         % TS 38.211 Table 6.3.3.2-4
```

TS 38.211 defines 13 PRACH formats and categorizes them as long or short preambles. Long preambles have a sequence of length  $L_{RA} = 839$ , whereas short preambles have a sequence of length  $L_{RA} = 139$ . The formats associated with long preambles are: 0, 1, 2, 3. The formats associated with short preambles are: A1, A2, A3, B1, B2, B3, B4, C0, C2, including mixed formats A1/B1, A2/B2, and A3/B3.

The configuration indices in Tables 6.3.3.2-2 to 6.3.3.2-4 define the time resources in which each preamble format can be transmitted. Each preamble format is associated with several configuration indices. You can choose a PRACH format without the need to look up the configuration tables by setting the value of ConfigurationIndex based on the preferred format. This value corresponds to the largest range of time resources in which you can transmit the preferred preamble format.

```
format = ; % PRACH preamble format ('0','1','2','3','A1','A2','A3','B1','B2')
```

Select the configuration table based on FrequencyRange and DuplexMode.

```
if strcmpi(prach.FrequencyRange, 'FR1')
    if strcmpi(prach.DuplexMode, 'TDD') % TS 38.211 Table 6.3.3.2-3
        configTable = nrPRACHConfig.Tables.ConfigurationsFR1Unpaired;
    else % TS 38.211 Table 6.3.3.2-2
        configTable = nrPRACHConfig.Tables.ConfigurationsFR1PairedSUL;
    end
else % TS 38.211 Table 6.3.3.2-4
    configTable = nrPRACHConfig.Tables.ConfigurationsFR2;
end
```

Among all configurations corresponding to the same short preamble format in Table 6.3.3.2-2, the second to last configuration has the largest number of time resources for transmitting the PRACH preamble. In all the other cases, including mixed formats in Table 6.3.3.2-2, the last configuration has the largest number of time resources for transmitting the PRACH preamble. This example uses this information to set the value of the ConfigurationIndex property. If you select format B2 or B3, this example sets the maximum value of TimeIndex.

```
if strcmpi(prach.FrequencyRange, 'FR1') && strcmpi(prach.DuplexMode, 'FDD') && ...
    any(strcmpi(format, {'A1', 'A2', 'A3', 'B1', 'B4', 'C0', 'C2'}))
    prach.ConfigurationIndex = find(strcmpi(configTable.PreambleFormat, format), 1, 'last') - 2;
else
    if ~any(strcmpi(format, {'B2', 'B3'}))
        prach.ConfigurationIndex = find(strcmpi(configTable.PreambleFormat, format), 1, 'last') - 1;
    else
        % Format B2 and B3 only appear in mixed formats, so select an
        % appropriate mixed format and set the maximum value of TimeIndex
        prach.ConfigurationIndex = find(endsWith(configTable.PreambleFormat, format), 1, 'last') - 1;
        prach.TimeIndex = prach.NumTimeOccurrences - 1;
    end
end
```

### How to Select SubcarrierSpacing, ActivePRACHSlot, and NPRACHSlot to Generate Active PRACH Preamble

Tables 6.3.3.2-2 to 6.3.3.2-4 in TS 38.211 describe which PRACH slot corresponds to an active PRACH preamble. The third and fourth columns of these tables represent the system frame numbers that correspond to an active PRACH preamble. Depending on the selected frequency range, FR1 or FR2, the fifth column represents the slot numbers for 15 kHz or 60 kHz subcarrier spacing, respectively, corresponding to an active PRACH preamble. If a PRACH preamble is not active in the current time resources, no time transmission can take place.

For example, the selected PRACH configuration is active in any system frame and subframe if PRACH subcarrier spacing is set to 15 kHz, as shown in Table 6.3.3.2-2.

```
disp(configTable(prach.ConfigurationIndex+1, :))
```

ConfigurationIndex	PreambleFormat	x	y	SubframeNumber	StartingSymbol	PR
146	{'A2/B2'}	1	{[0]}	{1x10 double}	0	

To verify that the PRACH preamble is active in the current slot, check the `prachSymbols` output of the `nrPRACH` function. This output is empty if the PRACH preamble is not active in the current slot. To generate an active PRACH preamble, loop through the values of the `NPRACHSlot` property until `prachSymbols` becomes nonempty.

The cases presented in this section show how to check whether the current PRACH short preamble is active. Both cases consider a PRACH short preamble format B2. If you change the format, the PRACH preamble may be active for values of the `NPRACHSlot` and `ActivePRACHSlot` properties different from those shown in this example.

### Case 1: Typical PRACH subcarrier spacing configuration

Set up a PRACH preamble for the selected format with a typical subcarrier spacing configuration. This example considers a 15 kHz subcarrier spacing, which is the typical value for short preambles in FR1. If you change the value of the subcarrier spacing or the format, you may need to change the values of `ActivePRACHSlot` and `NPRACHSlot` to get an active PRACH slot.

```
% Store the user-defined configuration
subcarrierSpacing = prach.SubcarrierSpacing;
activePRACHSlot = prach.ActivePRACHSlot;
nPRACHSlot = prach.NPRACHSlot;

% Set values of SubcarrierSpacing, ActivePRACHSlot, and NPRACHSlot for this
% case
if any(strcmpi(format,{'0','1','2'}))
    prach.SubcarrierSpacing = 1.25;
elseif strcmpi(format,'3')
    prach.SubcarrierSpacing = 5;
else % Short preambles
    if strcmpi(prach.FrequencyRange,'FR1')
        prach.SubcarrierSpacing = 15; % Valid values are (15, 30)
    else % FR2
        prach.SubcarrierSpacing = 60; % Valid values are (60, 120)
    end
end
prach.ActivePRACHSlot = 0;
prach.NPRACHSlot = 0;
```

According to Table 6.3.3.2-2 in TS 38.211, the UE can transmit PRACH in any slot.

```
prachSymbols = nrPRACH(carrier,prach);
active = ~isempty(prachSymbols);
disp(['active: ' num2str(active)])

active: 1
```

### Case 2: Alternative PRACH subcarrier spacing configuration

The PRACH subcarrier spacing is set to 30 kHz, whereas the carrier subcarrier spacing is set to the default value of 15 kHz. This means that each carrier slot contains two PRACH slots. This case does not consider PRACH long preambles and frequency range FR2 because they are not compatible with 30 kHz subcarrier spacing.

In the case of 30 kHz PRACH subcarrier spacing, only one of the two PRACH slots within a 15 kHz subcarrier spacing can be active. According to Table 6.3.3.2-2 in TS 38.211, either the first or the second PRACH slots can be active for PRACH preamble format B2. The value of `prach.ActivePRACHSlot` property defines which PRACH slot is active within the current carrier subframe. This property is the  $n_{slot}^{RA}$  parameter defined in TS 38.211 Section 5.3.2.

This case shows four combinations of the `NPRACHSlot` and `ActivePRACHSlot` property values and tests whether the PRACH preamble is active. This case displays the plot of the time-domain structure of the PRACH preamble for both combinations. The plot shows that the active PRACH preamble occupies the first half of the carrier slot when `ActivePRACHSlot` is 0 and occupies the second half of the carrier slot when `ActivePRACHSlot` is 1. For more details on this plot, see the Plot Time-Domain Structure of Selected PRACH Preamble on page 2-0 section.

```

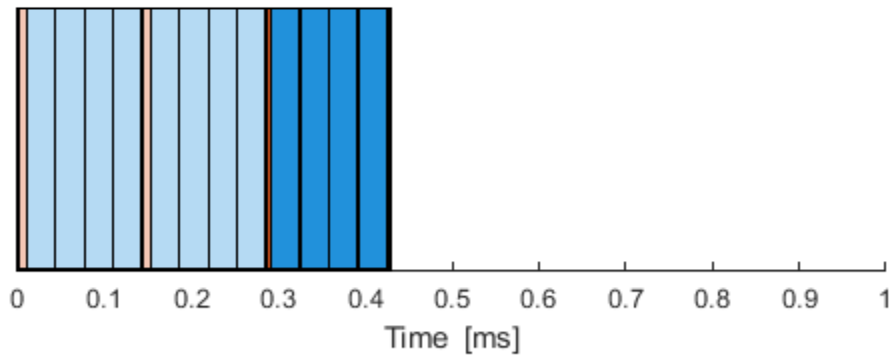
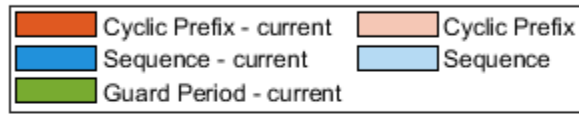
if ~any(strcmpi(format,{'0','1','2','3'})) && strcmpi(prach.FrequencyRange,'FR1') % Short preamble
    % Set subcarrier spacing to 30 kHz for this case
    prach.SubcarrierSpacing = 30;

    % Define all combinations of NPRACHSlot and ActivePRACHSlot to check
    nPRACHSlotCase2 = [0, 1, 2];
    activePRACHSlotCase2 = [0, 1];
    [NPRACHSlotCase2, ActivePRACHSlotCase2] = meshgrid(nPRACHSlotCase2,activePRACHSlotCase2);
    prachActivityTable = table(NPRACHSlotCase2(:),ActivePRACHSlotCase2(:),false*ones(numel(NPRACHSlotCase2),numel(ActivePRACHSlotCase2)),
        'VariableNames',{'NPRACHSlot','ActivePRACHSlot','active'});

    % Loop over all combinations
    for i = 1:numel(NPRACHSlotCase2)
        prach.NPRACHSlot = NPRACHSlotCase2(i);
        prach.ActivePRACHSlot = ActivePRACHSlotCase2(i);
        prachSymbols = nrPRACH(carrier,prach);
        active = ~isempty(prachSymbols); % Check if the PRACH preamble is active in the current slot
        prachActivityTable.active(i) = active;
        if active && prach.NPRACHSlot < 2
            % Plot the time-domain structure of the PRACH preamble for
            % active PRACH preambles in the first two slots
            hPRACHPreamblePlot(carrier,prach);
        end
    end
else
    % Display a message for the filtered cases
    if any(strcmpi(format,{'0','1','2','3'}))
        disp(['PRACH long preamble format ' format ' is not compatible with 30 kHz subcarrier spacing.'])
    else % FR2
        disp('Frequency range FR2 is not compatible with 30 kHz subcarrier spacing.')
    end
end
end

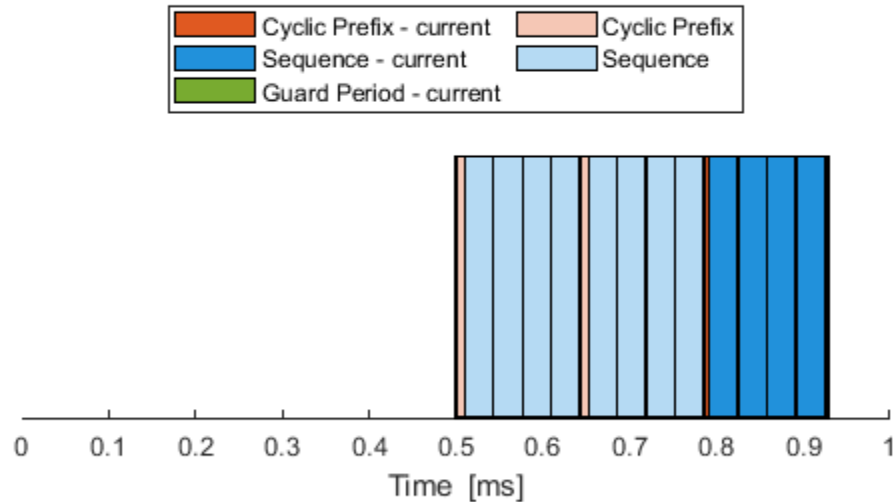
```

**Time-Domain Structure of PRACH Preamble Format A2/B2  
within One 15 kHz Carrier Slot (NPRACHSlot = [0, 1])**





### Time-Domain Structure of PRACH Preamble Format A2/B2 within One 15 kHz Carrier Slot (NPRACHSlot = [0, 1])



For short preamble formats with a 30 kHz subcarrier spacing, this table shows whether the PRACH preamble is active for each combination of the chosen values of the NPRACHSlot and ActivePRACHSlot properties.

```
if ~any(strcmpi(format,{'0','1','2','3'})) && strcmpi(prach.FrequencyRange,'FR1') % Short preamble
    disp(prachActivityTable)
end
```

NPRACHSlot	ActivePRACHSlot	active
0	0	1
0	1	0
1	0	0
1	1	1
2	0	1
2	1	0

Set the PRACH configuration object back to the user-defined configuration

```
prach.SubcarrierSpacing = subcarrierSpacing;
prach.ActivePRACHSlot = activePRACHSlot;
prach.NPRACHSlot = nPRACHSlot;
```

#### Inspect PRACH Configuration

The PRACH configuration object also has read-only properties that provide additional information about the current configuration:

- Preamble format: `Format`
- Length of the Zadoff-Chu preamble sequence: `LRA`
- Maximum number of allowed PRACH time occasions: `NumTimeOccasions`
- Number of OFDM symbols in the PRACH slot grid corresponding to one transmission occasion: `PRACHDuration`
- Location of the first OFDM symbol of the current PRACH occasion: `SymbolLocation`

`disp(prach)`

```
nrPRACHConfig with properties:
    FrequencyRange: 'FR1'
    DuplexMode: 'FDD'
    ConfigurationIndex: 146
    SubcarrierSpacing: 15
    SequenceIndex: 0
    PreambleIndex: 0
    RestrictedSet: 'UnrestrictedSet'
    ZeroCorrelationZone: 0
    RBOffset: 0
    FrequencyStart: 0
    FrequencyIndex: 0
    TimeIndex: 2
    ActivePRACHSlot: 0
    NPRACHSlot: 0

Read-only properties:
    Format: 'B2'
    LRA: 139
    NumTimeOccasions: 3
    PRACHDuration: 4
    SymbolLocation: 8
    SubframesPerPRACHSlot: 1
    PRACHSlotsPerPeriod: 10

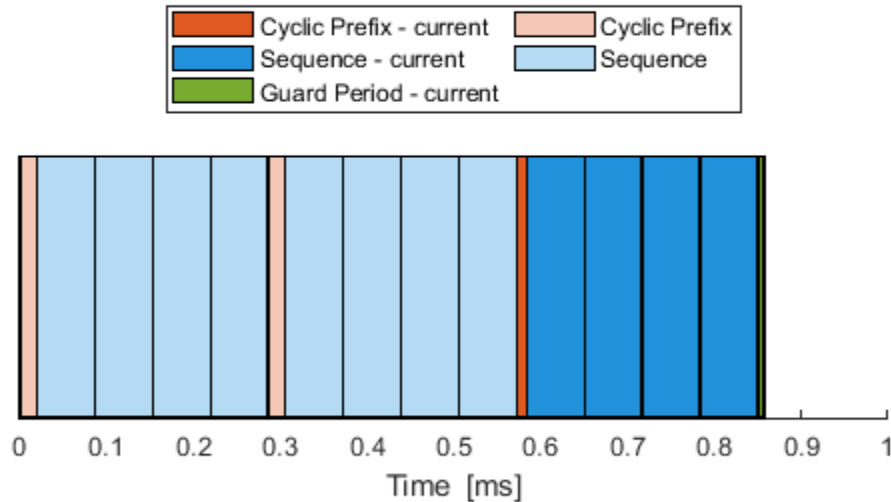
Constant properties:
    Tables: [1x1 struct]
```

### Plot Time-Domain Structure of Selected PRACH Preamble

This plot shows all the possible PRACH occasions (in the current carrier slot) in light colors and the current PRACH occasion (corresponding to the selected `TimeIndex`) in dark colors. This plot contains the cyclic prefix (CP), the PRACH active sequence periods, and a final guard period (GP) in red, blue, and green, respectively. If the PRACH preamble is not active in the current slot, the plot is empty. The plot shows time-related properties of the selected PRACH configuration and the PRACH position in the carrier slot. If the PRACH subcarrier spacing is smaller than the carrier subcarrier spacing, the plot shows the minimum number of carrier slots needed to transmit the PRACH preamble. The last PRACH occasion in time does not always correspond to the end of the carrier slot. The plot is empty for those time values in which no PRACH transmission is allowed for the current PRACH configuration.

```
hPRACHPreamblePlot(carrier,prach);
```

### Time-Domain Structure of PRACH Preamble Format A2/B2 within One 15 kHz Carrier Slot (NPRACHSlot = 0)



### Generate and Map PRACH Symbols to Resource Grid

The PRACH resource grid shows the location of the PRACH preamble in both time and frequency domain. Using this resource grid, you can:

- Inspect the PRACH preamble visually in both time and frequency domain
- Generate the PRACH waveform, which is obtained by modulating the resource grid

The PRACH resource grid generation consists of these steps:

- 1 Generate an empty grid
- 2 Generate the symbols to be transmitted in the PRACH waveform
- 3 Generate the frequency indices and time indices in which the PRACH symbols are located
- 4 Map the PRACH symbols to the PRACH resource grid

Generate an empty PRACH resource grid.

```
prachGrid = nrPRACHGrid(carrier,prach);
size(prachGrid)
```

```
ans = 1×2
```

```
624 14
```

Generate the PRACH symbols. The number of symbols depends on the PRACH configuration. The `prachSymbols` output is empty if the PRACH preamble is not active in the current slot.

```
prachSymbols = nrPRACH(carrier,prach);
```

Generate the PRACH indices. The value in each element of `prachIndices` is the linear index of the location of each element of `prachSymbols` in the PRACH resource grid.

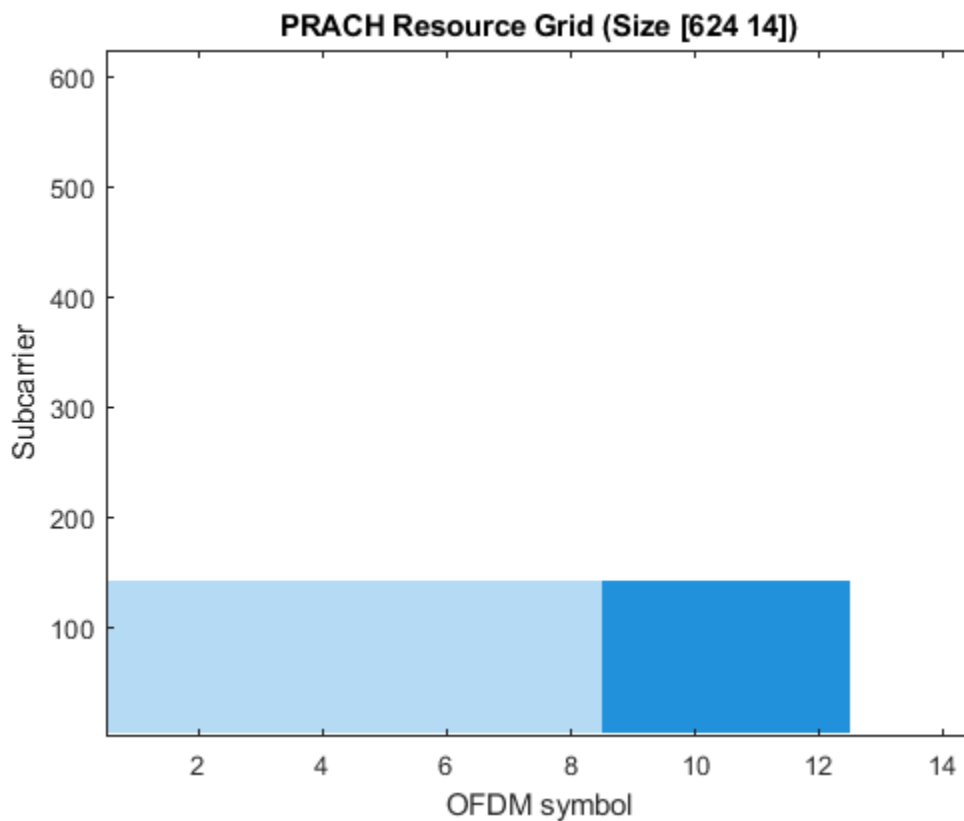
```
prachIndices = nrPRACHIndices(carrier,prach);
```

Map the PRACH symbols to the PRACH resource grid using the indices. To represent  $\beta_{PRACH}$  in TS 38.211 Section 6.3.3.2, the mapping applies a scaling factor of 1 to the PRACH symbols.

```
prachGrid(prachIndices) = 1 * prachSymbols;
```

The `hPRACHResourceGridPlot` helper function plots the PRACH resource grid to show the location of the active PRACH. The plot shows all the time occasions in which the PRACH can be transmitted. The plot shows all the possible PRACH occasions in the current carrier slot in light blue and the current PRACH occasion (corresponding to the selected `TimeIndex`) in dark blue. The plot is empty for OFDM symbols not used by any PRACH occasion for the current configuration. If the PRACH preamble is not active in the current slot, the plot is empty.

```
hPRACHResourceGridPlot(carrier,prach);
```



The PRACH resource grid contains 14 OFDM symbols except for these cases:

- For long preamble format 0, each preamble has one active sequence period that spans one subframe. Therefore, the slot grid related to format 0 has one OFDM symbol.

- For long preamble format 1, each preamble has two active sequence periods that span two subframes. Therefore, the slot grid related to format 1 has two OFDM symbols.
- For long preamble format 2, each preamble has four active sequence periods that span four subframes. Therefore, the slot grid related to format 2 has four OFDM symbols.
- For long preamble format 3, each preamble has four active sequence periods that span one subframe. Therefore, the slot grid related to format 3 has four OFDM symbols.
- For short preamble format C0, each preamble has one active sequence period. However, because of the guard and the cyclic prefix, the preamble spans two OFDM symbols. Therefore, the slot grid related to format C0 has seven OFDM symbols.

You can retrieve the number of active sequence periods from the value of the `PRACHDuration` property of the PRACH configuration object.

### Generate Waveform for Single PRACH Preamble

Generate a time-domain waveform for a single PRACH preamble by modulating the PRACH resource grid. To set the number of time-domain samples over which to apply windowing and overlapping of OFDM symbols, use `windowing`. This example uses the default value for windowing. For more details about windowing, see `nrPRACHOFDMModulate`.

```
windowing = [];
[prachWaveform,prachInfo] = nrPRACHOFDMModulate(carrier,prach,prachGrid,'Windowing',windowing);
```

The output `prachWaveform` is a column vector corresponding to the time-domain waveform. The output `prachInfo` is a structure that provides dimensional information related to the PRACH. This example displays this information by using the `hPRACHInfoDisplay` helper function. The function displays the information related to the number of samples corresponding to CP, PRACH active sequence period  $T_{SEQ}$ , and GP for each OFDM symbol in a tabular format. The table lists all the OFDM symbols that fit in the resource grid. For short preamble formats, the values marked with \* correspond to all possible PRACH occasions except the current one (marked light blue in the resource grid plot). For short preamble formats, the values within angle brackets represent OFDM symbols not used by any PRACH occasion for the current configuration (corresponding to an empty space in time in the resource grid plot).

Check the information related to the OFDM symbols against the `PRACHDuration`, `SymbolLocation`, and `NumTimeOccasions` properties. These properties show that:

- Each PRACH occasion lasts 4 OFDM symbols
- The current PRACH occasion starts at OFDM symbol 8
- 3 PRACH occasions are possible in time

```
hPRACHInfoDisplay(carrier,prach>windowing);
```

Information associated with PRACH:

```
SubcarrierSpacing:    15 kHz
Number of subcarriers: 624
```

Information associated with PRACH OFDM modulation:

```
Nfft:                1024
Windowing:            72
Offset:              0 samples
```

```
Symbol   TCP   TSEQ   GP
-----
```

```

0*      296*   1024*   0*
1*      0*    1024*   0*
2*      0*    1024*   0*
3*      0*    1024*   0*
4*      296*   1024*   0*
5*      0*    1024*   0*
6*      0*    1024*   0*
7*      0*    1024*   0*
8       180    1024    0
9        0    1024    0
10       0    1024    0
11       0    1024    108
<12> < 0> <1024> < 0>
<13> < 0> <1024> < 0>

```

```

* : OFDM symbols for unused PRACH time occasions
<#> : OFDM symbols not used by any PRACH time occasion
    for the current configuration

```

```

Total samples:      15360
Sample rate:        15.360 MHz
Duration:           1.000 ms
Total number of subframes: 1

```

### Summary and Further Exploration

This example shows how to configure the PRACH, set the configuration index based on the selected format, and determine whether a PRACH preamble is active in the current time resources. This example guides you through PRACH resource grid and time-domain waveform generation. Plotting the time-domain structure of the PRACH preamble displays all the available PRACH occasions for the selected configuration within one subframe. Plotting the resource grid displays all the available PRACH occasions for the selected configuration in both the time and frequency domain.

This example shows how to generate a waveform for a single PRACH preamble. For an example that generates a waveform for multiple PRACH preambles, see “5G NR PRACH Waveform Generation” on page 2-94.

### Appendix

This example uses these helper functions:

- hPRACHInfoDisplay
- hPRACHPreamblePlot
- hPRACHResourceGridPlot

### Selected Bibliography

- 1 3GPP TS 38.211. "NR; Physical channels and modulation." *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.*

## See Also

### Functions

nrPRACH | nrPRACHGrid | nrPRACHIndices

### Objects

nrPRACHConfig

## Related Examples

- "5G NR PRACH Waveform Generation" on page 2-94
- "5G NR PRACH Detection Test" on page 2-98

## 5G NR PRACH Waveform Generation

This example implements a 5G NR PRACH waveform generator using 5G Toolbox™. The example shows how to parameterize and generate a 5G New Radio (NR) physical random access channel (PRACH) waveform, as defined in TS 38.211 [ 1 ]. The example demonstrates the parameterization and generation of one PRACH configuration in a single carrier, and displays the positions of the PRACH preambles in the resource grid. You can define the length of the waveform, in terms of subframes, and set the pattern of the active PRACH preambles in the generated waveform.

### Waveform and Carrier Configuration

Configure one carrier and set the length of the generated waveform in terms of 1 ms subframes. Visualize the generated resource grid by setting the `DisplayGrids` field to 1.

Use the `waveconfig` structure to store configuration parameters needed for the PRACH waveform generation. The `waveconfig` structure contains these fields:

- `NumSubframes`: Number of 1 ms subframes in generated waveform.
- `DisplayGrids`: If set to 1, the example displays the resource grid.
- `Windowing`: Number of time-domain samples over which to apply windowing and overlapping of OFDM symbols. For more information, see `nrPRACHOFDMModulate`.
- `Carriers`: Carrier-specific configuration object, as described in `nrCarrierConfig`.
- `PRACH`: Structure containing the PRACH-related configuration, as described in detail in the PRACH Configuration section.

```
waveconfig = [];
waveconfig.NumSubframes = 10; % Number of 1 ms subframes in generated waveform
waveconfig.DisplayGrids = 1; % Display the resource grid
waveconfig.Windowing = []; % Use the default windowing

% Define a carrier configuration object
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 15;
carrier.NSizeGrid = 52;

% Store the carrier into the waveconfig structure
waveconfig.Carriers = carrier;
```

### PRACH Configuration

Set the parameters for the PRACH, taking into account that the numerology of the PRACH can be different from that of the carrier. This example sets the PRACH configuration corresponding to a PRACH short preamble format B2 with 15 kHz subcarrier spacing.

You can also set additional PRACH parameters. For more information, see `nrPRACHConfig`.

Add the field `PRACH` to the `waveconfig` structure to store the PRACH configuration and related parameters. The field `PRACH` is a structure containing these fields:

- `Config`: PRACH configuration object
- `AllocatedPreambles`: Index (0-based) of the allocated PRACH preambles to transmit. This field considers only the active PRACH preambles. Set this value to 'all' to include all the active PRACH preambles in the waveform.



- Power: PRACH power scaling in dB. This parameter represents  $\beta_{PRACH}$  (in dB) in TS 38.211 Section 6.3.3.2.

```

% PRACH configuration
prach = nrPRACHConfig;
prach.FrequencyRange = 'FR1'; % Frequency range ('FR1', 'FR2')
prach.DuplexMode = 'FDD'; % Duplex mode ('FDD', 'TDD', 'SUL')
prach.ConfigurationIndex = 145; % Configuration index (0...255)
prach.SubcarrierSpacing = 15; % Subcarrier spacing (1.25, 5, 15, 30, 60, 120)
prach.FrequencyIndex = 0; % Index of the PRACH transmission occasions in frequency domain
prach.TimeIndex = 2; % Index of the PRACH transmission occasions in time domain (0...6)
prach.ActivePRACHSlot = 0; % Active PRACH slot number within a subframe or a 60 kHz slot (0...1)

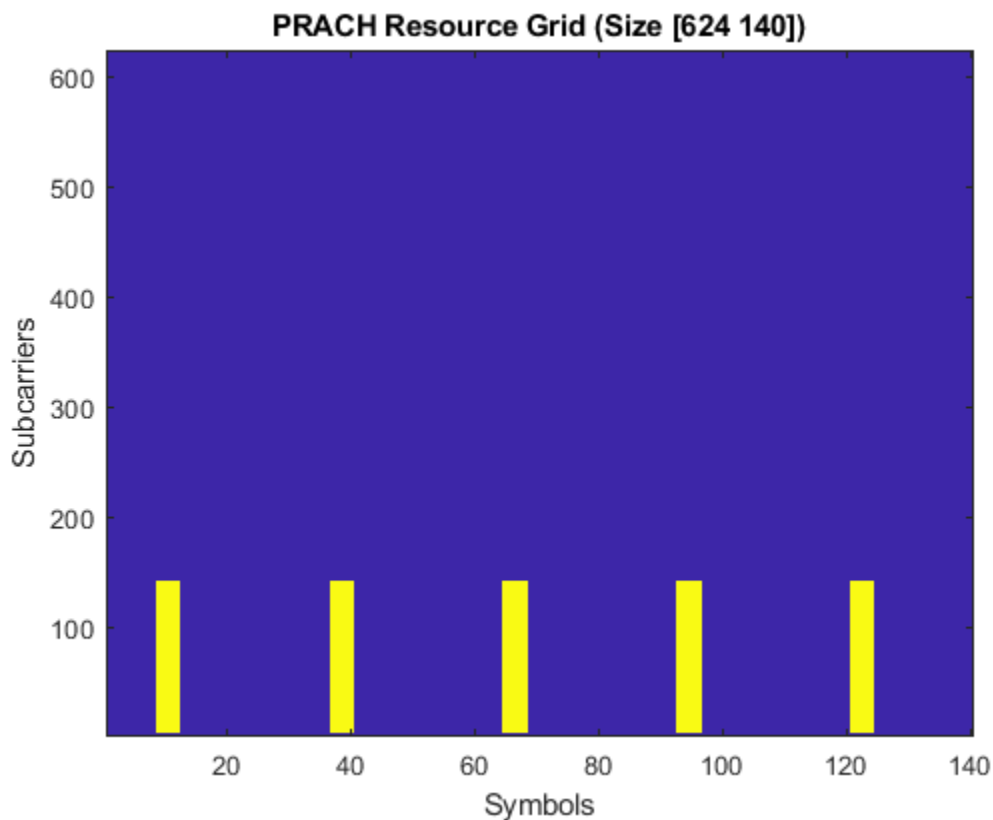
% Store the PRACH configuration and additional parameters in the
% waveconfig structure
waveconfig.PRACH.Config = prach;
waveconfig.PRACH.AllocatedPreambles = 'all'; % Index of the allocated PRACH preambles
waveconfig.PRACH.Power = 0; % PRACH power scaling in dB

```

### Waveform Generation

Generate the PRACH complex baseband waveform by using the parameters stored in the waveconfig structure.

```
[waveform,gridset,winfo] = hNRPRACHWaveformGenerator(waveconfig);
```



When `waveconfig.DisplayGrids` is set to 1, the waveform generator also plots the PRACH resource grid, in PRACH numerology. For more information on the number of OFDM symbols in the resource grid, see 5G NR PRACH Configuration.

The waveform generator function returns the time domain waveform, and two structures: `gridset` and `winfo`.

The structure `winfo` contains these fields:

- `NPRACHSlot`: PRACH slot numbers of each allocated PRACH preamble
- `PRACHSymbols`: PRACH symbols corresponding to each allocated PRACH slot
- `PRACHSymbolsInfo`: Additional information associated with PRACH symbols
- `PRACHIndices`: PRACH indices corresponding to each allocated PRACH slot
- `PRACHIndicesInfo`: Additional information associated with PRACH indices

The structure `gridset` contains these fields:

- `ResourceGrid`: Resource grid corresponding to this carrier
- `Info`: Structure with information corresponding to the PRACH OFDM modulation. If the PRACH is configured for FR2 or the PRACH slot for the current configuration spans more than one subframe, some of the OFDM-related information may be different between PRACH slots. In this case, the info structure is an array of the same length as the number of PRACH slots in the waveform.

```
disp('Information associated with PRACH OFDM modulation for the first PRACH slot:')
disp(gridset.Info(1))
```

```
Information associated with PRACH OFDM modulation for the first PRACH slot:
      Nfft: 1024
      SampleRate: 15360000
CyclicPrefixLengths: [188 0 0 0 188 0 0 0 180 0 0 0 0 0]
      GuardLengths: [0 0 0 108 0 0 0 108 0 0 0 108 0 144]
      SymbolLengths: [1x14 double]
      OffsetLength: 0
      Windowing: 72
```

### Summary and Further Exploration

This example shows how to generate a time-domain waveform for a single PRACH configuration on a single carrier. You can set the length of the generated waveform in terms of number of subframes. You can also set the pattern of PRACH preambles in the generated waveform. The example also shows the OFDM-related information for the PRACH.

To generate a waveform containing multiple PRACH configurations in the same carrier, run this example for several PRACH configurations and add the generated waveforms together.

For more information about the PRACH configuration and the PRACH resource grid, see “5G NR PRACH Configuration” on page 2-81.

### Appendix

This example uses these helper functions:

- hNRPRACHWaveformGenerator.m

### **Selected Bibliography**

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

### **See Also**

#### **Functions**

nrPRACH | nrPRACHGrid | nrPRACHIndices

#### **Objects**

nrPRACHConfig

### **Related Examples**

- "5G NR PRACH Configuration" on page 2-81
- "5G NR PRACH Detection Test" on page 2-98

## 5G NR PRACH Detection Test

This example shows how to model the physical random access channel (PRACH) missed detection conformance test, as defined in TS 38.141-1 [ 1 ]. You can learn how to measure the probability of correct detection of the PRACH preamble in the presence of a preamble signal.

### Introduction

The PRACH is an uplink transmission used by User Equipment (UE) to initiate synchronization with the gNodeB. TS 38.141-1 Section 8.4.1.5 defines the probability of PRACH detection to be greater than or equal to 99% at specific SNR values for a set of PRACH configurations and propagation conditions. There are several detection error cases:

- Detecting an incorrect preamble
- Not detecting a preamble
- Detecting the correct preamble but with the wrong timing estimation

TS 38.141-1 states that a correct detection is achieved when the estimation error of the timing offset of the strongest path is less than the time error tolerance given in Table 8.4.1.1-1. For channel propagation conditions TDLC300-100 and PRACH preamble format 0, the time error tolerance is 2.55 microseconds.

In this example, a PRACH waveform is configured and passed through an appropriate channel. At the receiver side, the example performs PRACH detection and calculates the PRACH detection probability. The example considers the parameters defined in TS 38.141-1 Table 8.4.1.5-1 and Table A.6-1. These are: normal mode (i.e., unrestricted set), 2 receive antennas, TDLC300-100 channel, normal cyclic prefix, burst format 0, SNR -6.0 dB. If you change the PRACH configuration to use one of the other PRACH preamble formats listed in Table A.6-1, you need to update the values of the time error tolerance and the SNR, according to TS 38.141-1 Table 8.4.1.1-1 and Tables 8.4.1.5-1 to 8.4.1.5-3, respectively.

### Simulation Configuration

The example considers 10 subframes at a number of SNRs. You should use a large number of numSubframes to produce meaningful results. You can set SNRdB as an array of values or a scalar. Table 8.4.1.5-1 in TS 38.141-1 specifies the frequency offset foffset that is modeled between transmitter and receiver.

```
numSubframes = 10;           % Number of 1 ms subframes to simulate at each SNR
SNRdB = [-21, -16, -11, -6, -1]; % SNR range in dB
foffset = 400.0;           % Frequency offset in Hz
timeErrorTolerance = 2.55; % Time error tolerance in microseconds
```

### Carrier Configuration

Use the nrCarrierConfig configuration object carrier to specify the carrier settings. The example considers a carrier characterized by a subcarrier spacing of 15 kHz and a bandwidth of 5 MHz. That is, the carrier spans 25 resource blocks, according to Table 5.3.2-1 in TS 38.104 [ 2 ].

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 15;
carrier.NSizeGrid = 25;
```

## PRACH Configuration

Table A.6-1 in TS 38.141-1 specifies the PRACH configurations to use for the PRACH detection conformance test.

Set the PRACH configuration by using the `nrPRACHConfig` configuration object `prach`, according to Table A.6-1 and Section 8.4.1.4.2 in TS 38.141-1.

```
% Define the value of ZeroCorrelationZone using the NCS table stored in
% the |nrPRACHConfig| object
ncsTable = nrPRACHConfig.Tables.NCSFormat012;
NCS = 13;
zeroCorrelationZone = ncsTable.ZeroCorrelationZone(ncsTable.UnrestrictedSet==NCS);

% Set PRACH configuration
prach = nrPRACHConfig;
prach.FrequencyRange = 'FR1'; % Frequency range
prach.DuplexMode = 'FDD'; % Frequency Division Duplexing (FDD)
prach.ConfigurationIndex = 27; % Configuration index for format 0
prach.SubcarrierSpacing = 1.25; % Subcarrier spacing
prach.SequenceIndex = 22; % Logical sequence index
prach.PreambleIndex = 32; % Preamble index
prach.RestrictedSet = 'UnrestrictedSet'; % Normal mode
prach.ZeroCorrelationZone = zeroCorrelationZone; % Cyclic shift index
prach.FrequencyStart = 0; % Frequency location

% Compute the OFDM-related information for this PRACH configuration
windowing = [];
ofdmInfo = nrPRACHOFDMInfo(carrier,prach,'Windowing',windowing);
```

## Propagation Channel Configuration

Use the `nrTDLChannel` object to configure the tapped delay line (TDL) propagation channel model `channel` as described in TS 38.141-1 Table 8.4.1.1-1.

```
channel = nrTDLChannel;
channel.DelayProfile = "TDL-C"; % Delay profile
channel.DelaySpread = 300e-9; % Delay spread in seconds
channel.MaximumDopplerShift = 100.0; % Maximum Doppler shift in Hz
channel.SampleRate = ofdmInfo.SampleRate; % Input signal sample rate in Hz
channel.MIMOCorrelation = "Low"; % MIMO correlation
channel.TransmissionDirection = "Uplink"; % Uplink transmission
channel.NumTransmitAntennas = 1; % Number of transmit antennas
channel.NumReceiveAntennas = 2; % Number of receive antennas
channel.NormalizePathGains = true; % Normalize delay profile power
channel.Seed = 42; % Channel seed. Change this for different channel realizations
channel.NormalizeChannelOutputs = true; % Normalize for receive antennas
```

## Loop for SNR Values

Use a loop to run the simulation for the set of SNR points given by the vector `SNRdB`. The SNR vector configured here is a range of SNR points including a point at -6.0 dB, the SNR at which the test requirement for PRACH detection rate (99%) is to be achieved for preamble format 0, as discussed in Table 8.4.1.5-1 in TS 38.141-1.

`hNRPRACHWaveformGenerator` generates an output signal normalized to the same transmit power as for an uplink data transmission within the 5G Toolbox™. Therefore, the same normalization must

take place on the noise added to the PRACH. The noise added before OFDM demodulation will be amplified by the IFFT by a factor equal to the square root of the size of the IFFT ( $N_{FFT}$ ). To ensure that the power of the noise added is normalized after demodulation, and thus to achieve the desired SNR, the desired noise power is divided by  $N_{FFT}$ . In addition, as real and imaginary parts of the noise are created separately before being combined into complex additive white Gaussian noise, the noise amplitude is scaled by  $1/\sqrt{2}$  so the generated noise power is 1.

At each SNR test point, calculate the probability detection on a subframe by subframe basis using these steps:

- *PRACH Transmission:* Use `hNRPRACHWaveformGenerator` to generate a PRACH waveform. Send the PRACH preambles with the timing offsets defined in TS 38.141-1 Figure 8.4.1.4.2-2. Set a timing offset base value to 50% of the number of cyclic shifts for PRACH generation. This offset is increased for each preamble, adding a step value of 0.1 microseconds, until the end of the tested range, which is 0.9 microseconds for PRACH preamble format 0. This pattern then repeats.
- *Noisy Channel Modeling:* Pass the waveform through a TDL channel and add additive white Gaussian noise. Add additional samples to the end of the waveform to cover the range of delays expected from the channel modeling (a combination of implementation delay and channel delay spread). This implementation delay is then removed to ensure the implementation delay is interpreted as an actual timing offset in the preamble detector.
- *Application of Frequency Offset:* Apply the frequency offset to the received waveform as defined by the specification.
- *PRACH Detection:* Perform PRACH detection using `hPRACHDetect` for all cell preamble indices (0-63). Use the detected PRACH index and offset returned by `hPRACHDetect` to determine where a detection was successful according to the constraints discussed in the Introduction section.

```
% Initialize variables storing probability of detection at each SNR
pDetection = zeros(size(SNRdB));
```

```
% Get the maximum number of delayed samples by a channel multipath
% component. This is calculated from the channel path with the largest
% delay and the implementation delay of the channel filter. The example
% requires this to flush the channel filter to obtain the received signal.
channelInfo = info(channel);
maxChDelay = ceil(max(channelInfo.PathDelays*channel.SampleRate)) + channelInfo.ChannelFilterDelay;
```

```
% Total number of PRACH slots in the simulation period
numPRACHSlots = floor(numSubframes / prach.SubframesPerPRACHSlot);
```

```
% Store the configuration parameters needed to generate the PRACH waveform
waveconfig.NumSubframes = prach.SubframesPerPRACHSlot;
waveconfig.Windowing = windowing;
waveconfig.Carriers = carrier;
waveconfig.PRACH.Config = prach;
```

```
% The temporary variables 'prach_init', 'waveconfig_init', 'ofdmInfo_init',
% and 'channelInfo_init' are used to create the temporary variables
% 'prach', 'waveconfig', 'ofdmInfo', and 'channelInfo' within the SNR loop
% to create independent instances in case of parallel simulation
prach_init = prach;
waveconfig_init = waveconfig;
ofdmInfo_init = ofdmInfo;
channelInfo_init = channelInfo;
```

```

for snrIdx = 1:numel(SNRdB) % comment out for parallel computing
% parfor snrIdx = 1:numel(SNRdB) % uncomment for parallel computing
% To reduce the total simulation time, you can execute this loop in
% parallel by using the Parallel Computing Toolbox. Comment out the 'for'
% statement and uncomment the 'parfor' statement. If the Parallel Computing
% Toolbox(TM) is not installed, 'parfor' defaults to normal 'for' statement

    % Set the random number generator settings to default values
    rng('default');

    % Initialize variables for this SNR point, required for initialization
    % of variables when using the Parallel Computing Toolbox
    prach = prach_init;
    waveconfig = waveconfig_init;
    ofdmInfo = ofdmInfo_init;
    channelInfo = channelInfo_init;

    % Reset the channel so that each SNR point will experience the same
    % channel realization
    reset(channel);

    % Normalize noise power to take account of sampling rate, which is a
    % function of the IFFT size used in OFDM modulation. The SNR is defined
    % per resource element for each receive antenna.
    SNR = 10^(SNRdB(snrIdx)/20);
    N0 = 1/(sqrt(2.*channel.NumReceiveAntennas*double(ofdmInfo.Nfft))*SNR);

    % Detected preamble count
    detectedCount = 0;

    % Loop for each PRACH slot
    numActivePRACHSlots = 0;
    for nSlot = 0:numPRACHSlots-1

        prach.NPRACHSlot = nSlot;

        % Generate PRACH waveform for the current slot
        waveconfig.PRACH.Config.NPRACHSlot = nSlot;
        [waveform,~,winfo] = hNRPRACHWaveformGenerator(waveconfig);

        % Skip this slot if the PRACH is inactive
        if (isempty(winfo.WaveformResources.PRACH))
            continue;
        end

        numActivePRACHSlots = numActivePRACHSlots + 1;

        % Set PRACH timing offset in microseconds as per TS 38.141-1 Figure 8.4.1.4.2-2
        baseOffset = ((winfo.WaveformResources.PRACH.Resources.PRACHSymbolsInfo.NumCyclicShifts/2)
        timingOffset = baseOffset + mod(nSlot,10)/10; % (microseconds)
        sampleDelay = fix(timingOffset / 1e6 * ofdmInfo.SampleRate);

        % Generate transmit waveform
        txwave = [zeros(sampleDelay,1); waveform(1:(end-sampleDelay))];

        % Pass data through channel model. Append zeros at the end of the
        % transmitted waveform to flush channel content. These zeros take
        % into account any delay introduced in the channel. This is a mix

```

```

% of multipath delay and implementation delay. This value may
% change depending on the sampling rate, delay profile and delay
% spread
rxwave = channel([txwave; zeros(maxChDelay, size(txwave,2))]);

% Add noise
noise = N0*complex(randn(size(rxwave)), randn(size(rxwave)));
rxwave = rxwave + noise;

% Remove the implementation delay of the channel modeling
rxwave = rxwave((channelInfo.ChannelFilterDelay + 1):end, :);

% Apply frequency offset
t = ((0:size(rxwave, 1)-1)/channel.SampleRate).';
rxwave = rxwave .* repmat(exp(1i*2*pi*foffset*t), 1, size(rxwave, 2));

% PRACH detection for all cell preamble indices
[detected, offsets] = hPRACHDetect(carrier, prach, rxwave, (0:63).');

% Test for preamble detection
if (length(detected)==1)

    % Test for correct preamble detection
    if (detected==prach.PreambleIndex)

        % Calculate timing estimation error
        trueOffset = timingOffset/1e6; % (s)
        measuredOffset = offsets(1)/channel.SampleRate;
        timingerror = abs(measuredOffset-trueOffset);

        % Test for acceptable timing error
        if (timingerror<=timeErrorTolerance/1e6)
            detectedCount = detectedCount + 1; % Detected preamble
        else
            disp('Timing error');
        end
    else
        disp('Detected incorrect preamble');
    end
else
    disp('Detected multiple or zero preambles');
end

end % of nSlot loop

% Compute final detection probability for this SNR
pDetection(snrIdx) = detectedCount/numActivePRACHSlots;

end % of SNR loop

Detected multiple or zero preambles
Detected multiple or zero preambles

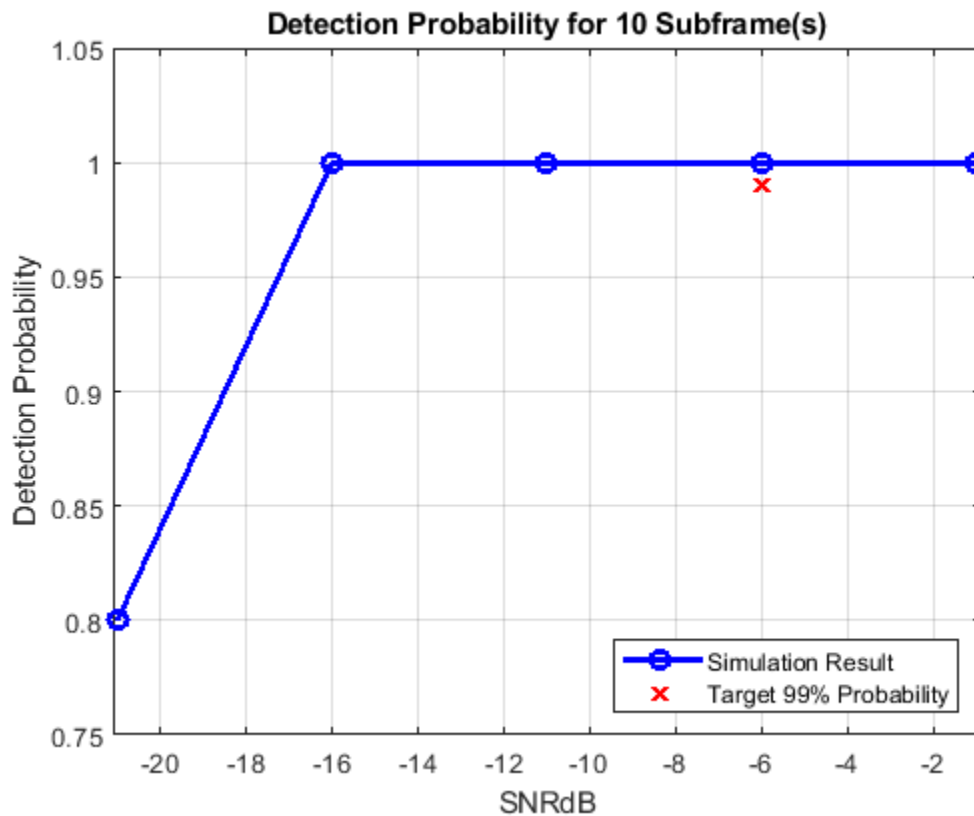
```

### Results

At the end of the SNR loop, the example plots the calculated detection probabilities for each SNR value against the target probability.



```
hPRACHDetectionResults(SNRdB, numSubframes, pDetection);
```



## Appendix

This example uses these helper functions:

- hNRPRACHWaveformGenerator.m
- hPRACHDetect.m
- hPRACHDetectionResults.m

## 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 38.104. "NR; Base Station (BS) radio transmission and reception." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

## See Also

### Functions

nrPRACH | nrPRACHGrid | nrPRACHIndices

### Objects

nrPRACHConfig

### **Related Examples**

- “5G NR PRACH Configuration” on page 2-81
- “5G NR PRACH Waveform Generation” on page 2-94

## NR UCI Multiplexing on PUSCH

This example shows the different processing steps involved in the data and control multiplexing to form a codeword associated with a physical uplink shared channel (PUSCH) using 5G Toolbox™ features.

### Introduction

Uplink control information (UCI) messages consist of a hybrid automatic repeat request acknowledgment (HARQ-ACK), channel state information (CSI), and a scheduling request (SR). These UCI messages are encoded and transmitted through the physical uplink control channel (PUCCH) or are multiplexed on the PUSCH. The CSI reporting configuration can be aperiodic (using a PUSCH), periodic (using a PUCCH), or semi-persistent (using a PUCCH or DCI-activated PUSCH). A CSI report comprises of two parts. CSI part 1 has a fixed payload size and is used to identify the number of information bits in CSI part 2. CSI part 1 must be transmitted completely before the transmission of CSI part 2. The HARQ-ACK (if any) and CSI (if any) is encoded and multiplexed with or without encoded UL-SCH data, and then transmitted on a PUSCH.

### Data and Control Multiplexing

The encoded data, encoded HARQ-ACK, encoded CSI part 1, and encoded CSI part 2 are multiplexed to form a codeword with the steps outlined in TS 38.212 Section 6.2.7 [1] on page 2-0 .

The UCI information is transmitted in only the OFDM symbols that are unused for demodulation reference signal (DM-RS) transmission. In any OFDM symbol used for UCI transmission for a UCI type, the mapping of that UCI type depends on the number of resource elements (REs) available for UCI transmission and the remaining REs required for that UCI type. If the number of remaining REs required for that UCI type in an OFDM symbol is greater than half of the available REs for the UCI transmission, the mapping of the UCI type is contiguous. Otherwise, the mapping is uniformly distributed across available REs in an OFDM symbol to achieve the diversity gain. The number of coded bits that are occupied in an RE for UCI or data transmission, is equal to the product of the modulation order and the number of layers.

The coded HARQ-ACK bits are placed from the OFDM symbol, after the first consecutive DM-RS OFDM symbols. The coded CSI part 1 or part 2 bits are placed at the starting OFDM symbol that is unused for DM-RS in the shared channel symbol allocation. The multiplexing operation depends on the number of HARQ-ACK bits. When the number of HARQ-ACK bits is less than or equal to 2, the coded HARQ-ACK bits are punctured. Otherwise, the coded HARQ-ACK bits are rate-matched.

Multiplexing involves these processing steps.

- Step 1: When the number of HARQ-ACK bits is less than or equal to 2, find the reserved HARQ-ACK locations.
- Step 2: When the number of HARQ-ACK bits is greater than 2, map the coded HARQ-ACK bits (if any).
- Step 3: Map the coded CSI part 1 and CSI part 2 bits (if any).
- Step 4: Map the coded UL-SCH bits (if any).
- Step 5: When the number of HARQ-ACK bits is less than or equal to 2, map the coded HARQ-ACK bits (if any).
- Step 6: Form the codeword.

This example shows steps 1 to 6 for two separate cases for the PUSCH, with one resource block occupying all of the OFDM symbols in a slot, a single layer, a pi/2-BPSK modulation scheme, and DM-

RS symbols occupying OFDM symbols 2, 7, and 11 (0-based). The REs other than DM-RS REs in the DM-RS OFDM symbols are used for data transmission. For the first case, the number of HARQ-ACK bits is less than or equal to 2. For the second case, the number of HARQ-ACK bits is greater than 2. This figure shows the grid with only the DM-RS symbol locations. This grid is populated with the coded types UL-SCH, HARQ-ACK, CSI part 1, and CSI part 2, according to the multiplexing operation for each case. The transport block size is set to 24, and the target code rate is set to 314/1024. The number of payload bits of each CSI part is set to 10, and all of the associated beta factors are set to 1.

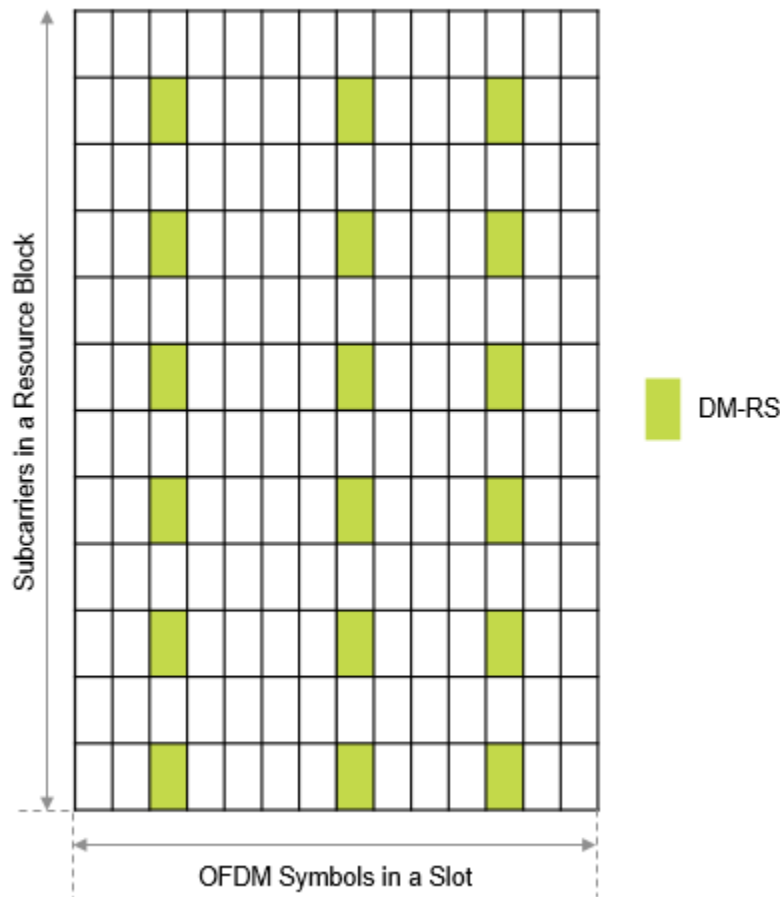


Figure 1: DM-RS Locations

### Configure Carrier Resource Grid and PUSCH

Configure an `nrCarrierConfig` object to create a 15 kHz carrier resource grid with one resource block. Configure an `nrPUSCHConfig` object to get the respective DM-RS locations as shown in the previous figure. Also, configure the UCI parameters (`BetaOffsetACK`, `BetaOffsetCSI1`, `BetaOffsetCSI2`, and `UCIScaling`). The property `BetaOffsetACK` determines the number of resources for multiplexing HARQ-ACK in a PUSCH. The properties `BetaOffsetCSI1` and `BetaOffsetCSI2` determine the number of resources for multiplexing CSI reports in a PUSCH. The property `UCIScaling` limits the number of REs assigned to the UCI on PUSCH.

```
% Set the carrier configuration
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 15; % Subcarrier spacing in kHz (15, 30, 60, 120, 240)
```

```

carrier.CyclicPrefix = 'normal'; % Cyclic prefix ('normal' or 'extended')
carrier.NSizeGrid = 1;           % Number of resource blocks in carrier resource grid (1...275)

% Set the PUSCH configuration
pusch = nrPUSCHConfig;
pusch.Modulation = 'pi/2-BPSK'; % Modulation scheme ('pi/2-BPSK', 'QPSK', '16QAM', '64QAM', ...)
pusch.NumLayers = 1;           % Number of layers (1, 2, 3, or 4)
pusch.SymbolAllocation = [0 14]; % OFDM symbol allocation [S L]
pusch.PRBSets = 0;             % Vector of PRB allocated with values in the range [0, 274]
pusch.MappingType = 'A';      % Mapping type ('A' or 'B')
pusch.FrequencyHopping = 'neither'; % Frequency hopping configuration ('neither', 'interSlot', ...)
pusch.SecondHopStartPRB = 0;   % Second hop start PRB in the range [0, 274], ensure the comb
pusch.TransformPrecoding = 0; % Transform precoding (0 or 1)

% Set the UCI parameters
pusch.BetaOffsetACK = 1; % Beta offset for HARQ-ACK
pusch.BetaOffsetCSI1 = 1; % Beta offset for CSI part 1
pusch.BetaOffsetCSI2 = 1; % Beta offset for CSI part 2
pusch.UCIScaling = 1; % UCI scaling factor

% Set the DM-RS parameters
pusch.DMRS.DMRSConfigurationType = 1; % DM-RS configuration type (1 or 2)
pusch.DMRS.DMRSTypeAPosition = 2; % DM-RS type A position (2 or 3)
pusch.DMRS.DMRSLength = 1; % DM-RS length (1 or 2)
pusch.DMRS.DMRSAdditionalPosition = 2; % Number of DM-RS additional positions (0, 1, 2, or 3)
pusch.DMRS.NumCDMGroupsWithoutData = 1; % Number of CDM groups without data (1, 2, or 3)

% Set the PT-RS parameters
pusch.EnablePTRS = 0; % Disable or Enable PT-RS (0 or 1)
pusch.PTRS.FrequencyDensity = 2; % PT-RS frequency density (2 or 4)
pusch.PTRS.TimeDensity = 1; % PT-RS time density (1, 2, or 4)
pusch.PTRS.REOffset = '00'; % PT-RS resource element offset ('00', '01', '10', or '11')
pusch.PTRS.NumPTRSSamples = 2; % Number of PT-RS samples (2 or 4)
pusch.PTRS.NumPTRSGroups = 2; % Number of PT-RS groups (2, 4, or 8)

% Set the target code rate, transport block size, payload lengths of CSI part 1 and CSI part 2
tcr = 314/1024; % Target code rate in the range (0, 1)
tbs = 24; % Transport block size
ocsi1 = 10; % Number of CSI part 1 bits
ocsi2 = 10; % Number of CSI part 2 bits

```

### Case 1: Number of HARQ-ACK Bits Less Than or Equal to 2

For illustration purposes, the number of HARQ-ACK bits is set to 1 in this example. To obtain the information about the number of coded bits of each type, use the `nrULSCHInfo` function. For the specified PUSCH configuration, target code rate, and payload values, the number of coded HARQ-ACK bits is 2, the number of coded CSI part 1 and CSI part 2 bits is 19 each, and the number of coded UL-SCH bits is 94.

**Step 1:** When the number of HARQ-ACK bits is less than or equal to 2, find the **reserved HARQ-ACK** locations and mark them on the grid.

The number of reserved HARQ-ACK locations is achieved by calculating the rate-matching length of the HARQ-ACK with number of HARQ-ACK bits set to 2. With the specified PUSCH configuration, target code rate, and transport block size, the number of reserved HARQ-ACK locations is 4. The HARQ-ACK is mapped to the REs in the OFDM symbol that is available after the first consecutive DM-RS OFDM symbols. This figure shows the locations of reserved HARQ-ACK.

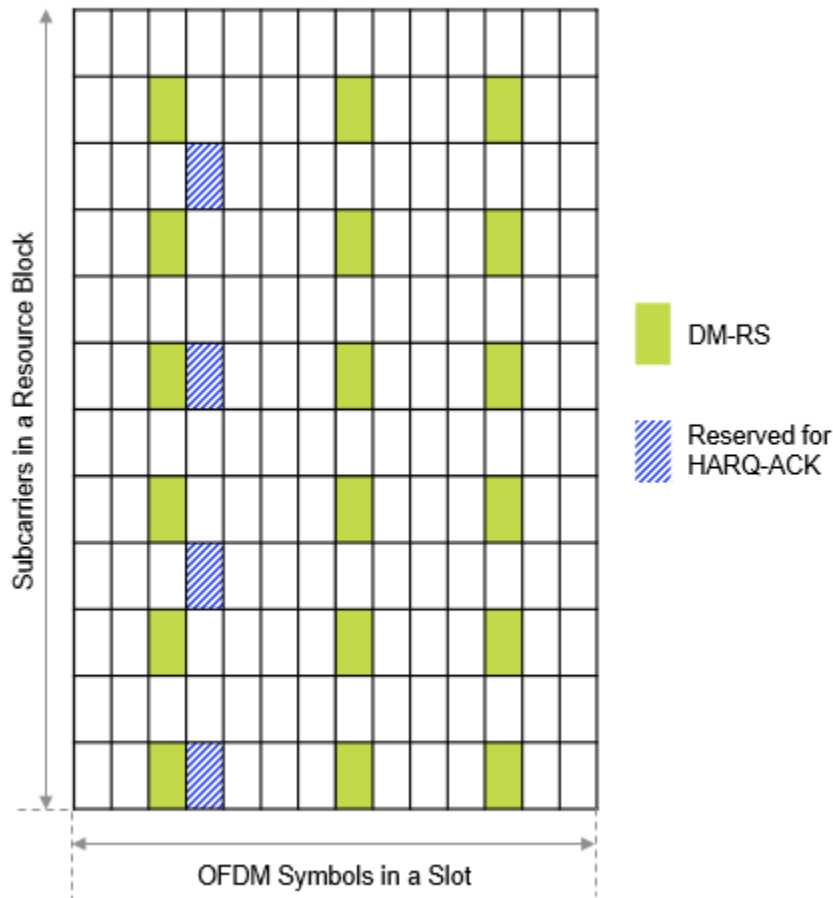


Figure 2: Reserved HARQ-ACK Locations

**Step 2:** When the number of HARQ-ACK bits is greater than 2, map the coded **HARQ-ACK** bits.

Because the number of HARQ-ACK bits is less than or equal to 2 in this case, skip this step.

**Step 3:** Map the coded **CSI part 1** and **CSI part 2** bits

The CSI mapping (CSI part 1 followed by CSI part 2) starts from the first non-DMRS OFDM symbol available in the PUSCH allocation. For the configured setup, the mapping starts from OFDM symbol 0 (0-based). The mapping locations are determined based on the number of REs available and the number of REs required for CSI part 1 transmission. CSI part 1 transmission required 19 REs. Because only 12 REs are available for transmission, every RE is occupied in this case for CSI part 1 in OFDM symbol 0. For the transmission of the remaining CSI part 1 symbols, the mapping goes to the next OFDM symbol not used for DM-RS (that is, OFDM symbol 1). In OFDM symbol 1, 12 REs are available, but CSI part 1 requires only 7 REs. Because the remaining REs required for CSI part 1 are more than half of the REs available for UCI transmission, CSI part 1 is mapped to contiguous REs. This figure shows this scenario.

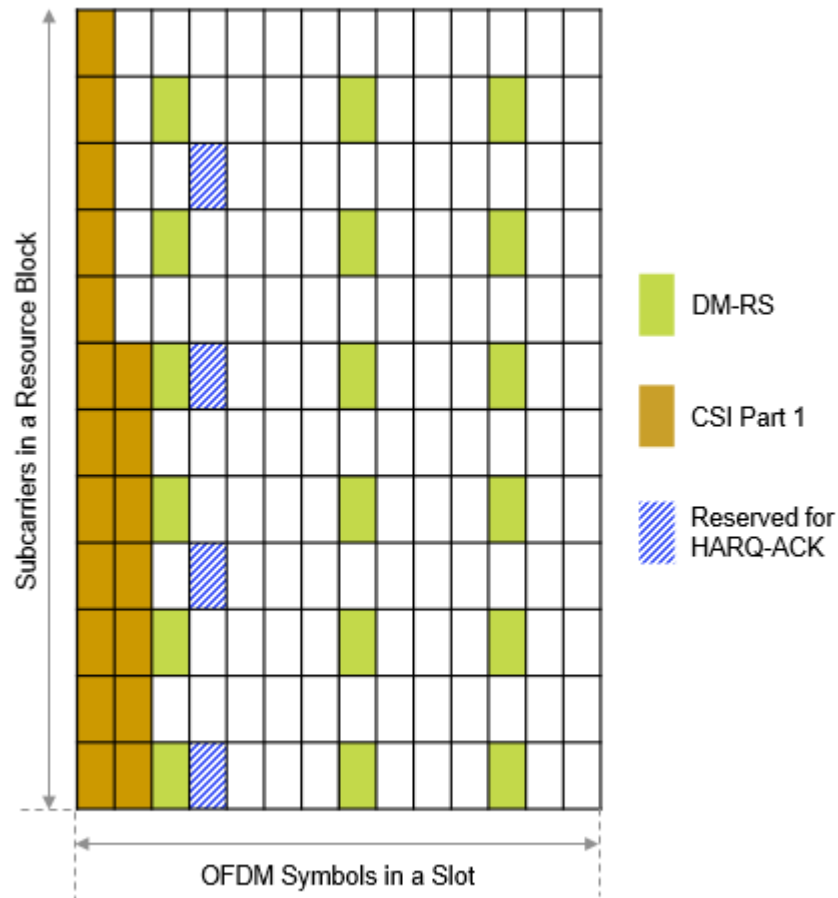


Figure 3: CSI Part 1 Locations

Once the coded CSI part 1 is completely mapped, the mapping of coded CSI part 2 starts. The mapping starts from the first non-DMRS OFDM symbol that is used for PUSCH transmission (that is, OFDM symbol 0). In OFDM symbol 0, no REs are available for UCI transmission, this OFDM symbol is skipped. As a result, mapping moves to the next OFDM symbol (that is, OFDM symbol 1). In OFDM symbol 1, 5 REs are available for UCI transmission, and 19 REs are required for CSI part 2 transmission. This scenario leads to mapping 5 REs with CSI part 2 and then continuing with the same mapping rule as in CSI part 1 for the next OFDM symbols other than DM-RS. Except the coded CSI part 1, the coded CSI part 2 and UL-SCH data can map to the reserved HARQ-ACK locations. This figure shows the mapping of CSI part 2. The reserved HARQ-ACK locations are covered by CSI part 2 in OFDM symbol 3. Also, in OFDM symbol 4, CSI part 2 is distributed because the number of REs required for the remaining CSI part 2 transmission is less than half of the REs available for the UCI transmission.

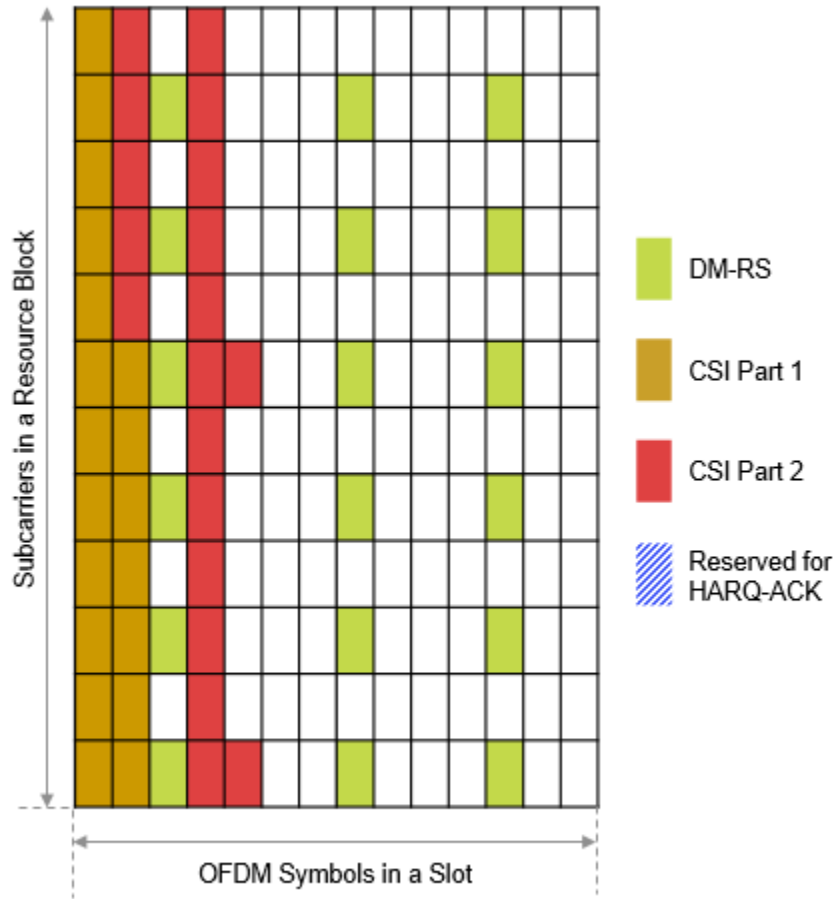


Figure 4: CSI Part 2 Locations

**Step 4:** Map the coded **UL-SCH** data bits.

This figure shows how the UL-SCH data is mapped to the remaining locations in the grid.



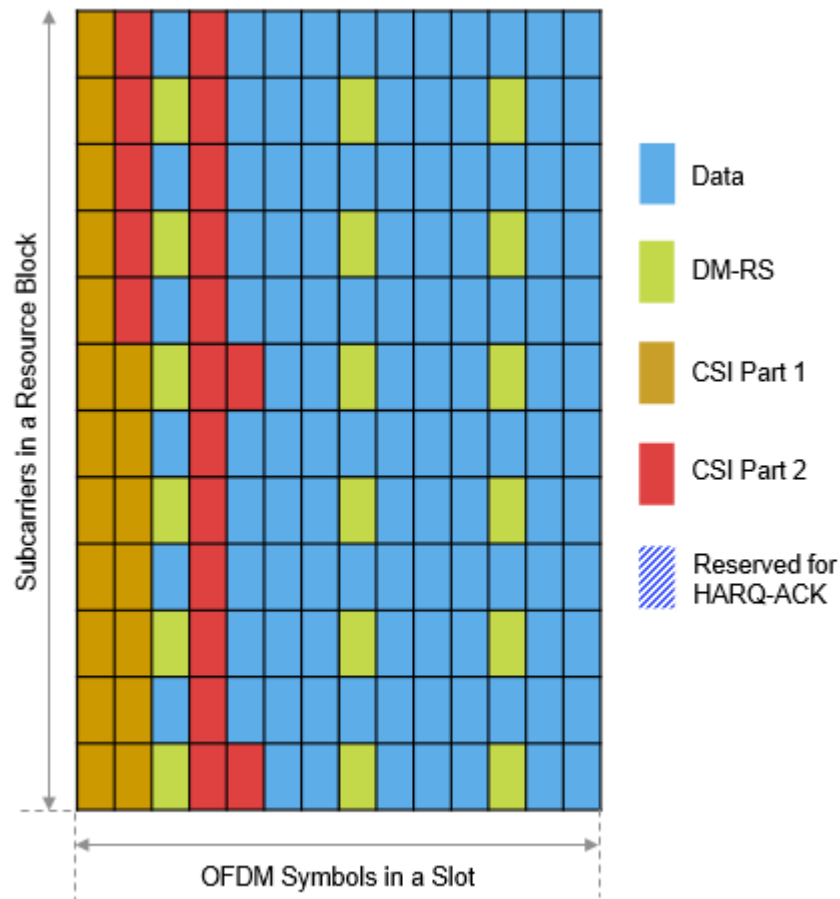


Figure 5: UL-SCH Data Locations

**Step 5:** When the number of HARQ-ACK bits is less than or equal to 2, map the coded **HARQ-ACK** bits.

The HARQ-ACK bits are mapped within the reserved HARQ-ACK locations in a distributed pattern. This figure indicates the HARQ-ACK mapping to the grid. The HARQ-ACK punctures the CSI part 2 that occupied the reserved HARQ-ACK locations.

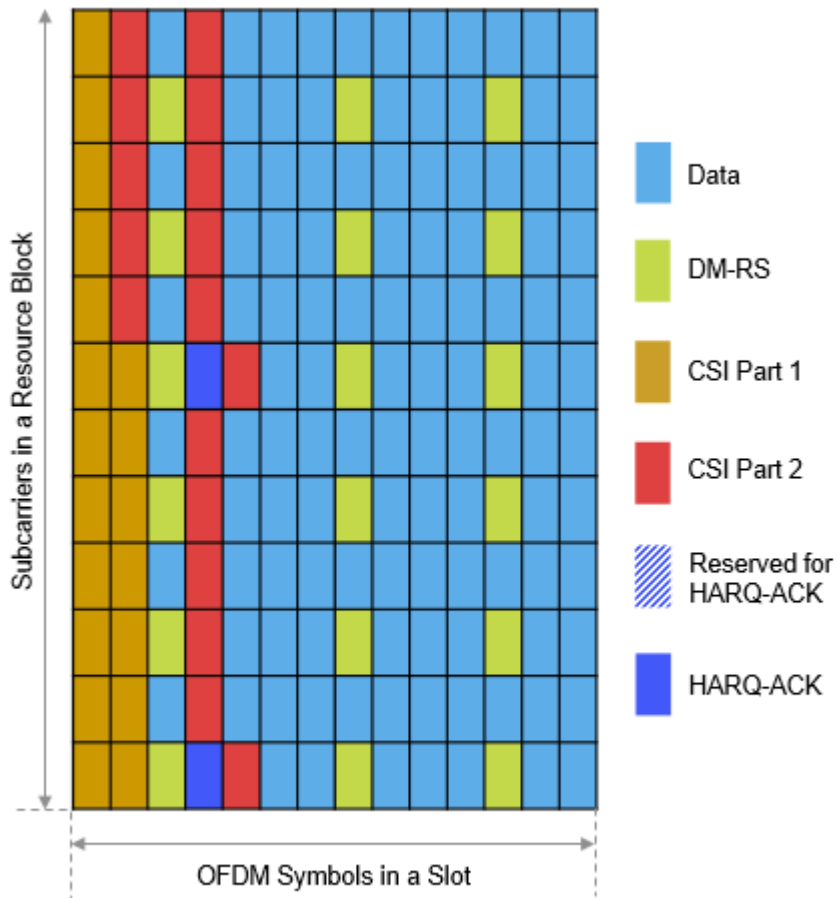


Figure 6: HARQ-ACK Locations

**Step 6:** Form the **codeword** by reading the bits frequency first and time next approach at each RE other than DM-RS REs.

Use the `nrULSCHMultiplex` function to get the codeword, performing the steps 1 to 6.

```
% Set the payload size of the HARQ-ACK to a value less than or equal to 2
oack = 1; % Number of HARQ-ACK bits

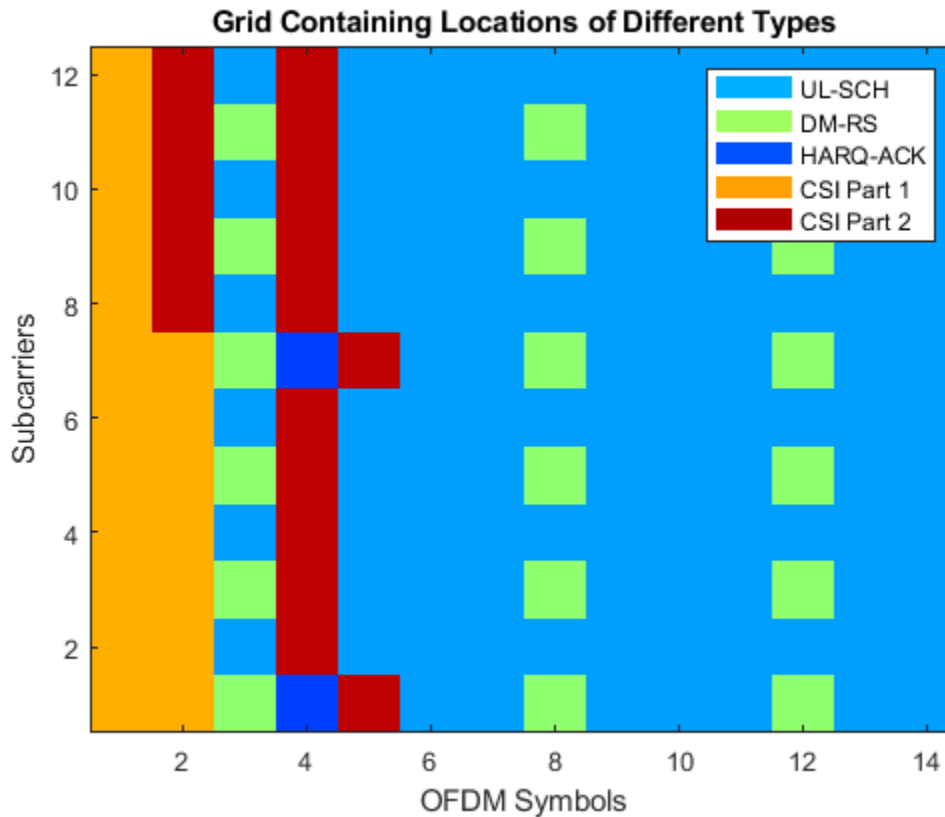
% Get the UL-SCH coding information
info = nrULSCHInfo(pusch,tcr,tbs,oack,ocsi1,ocsi2);

% Set random coded UL-SCH, HARQ-ACK, CSI part 1, and CSI part 2 bits
culsch = randi([0 1],info.GULSCH,1);
cack = randi([0 1],info.GACK,1);
ccsi1 = randi([0 1],info.GCSI1,1);
ccsi2 = randi([0 1],info.GCSI2,1);

% Get the codeword and locations of each type (data and UCI)
[cw,indInfo] = nrULSCHMultiplex(pusch,tcr,tbs,culsch,cack,ccsi1,ccsi2);

% Create and plot the output grid for the first layer with predefined symbol
```

```
% values for the different types
createAndPlotGrid(carrier,pusch,cw,indInfo)
```



### Case 2: Number of HARQ-ACK Bits Greater Than 2

For the same configuration setup mentioned in case 1, change the number of HARQ-ACK bits from 1 to 3. For this setup, the number of coded HARQ-ACK bits is 6, the number of coded CSI part 1 and part 2 bits is 19 each, and the number of coded UL-SCH bits is 106.

**Step 1:** When the number of HARQ-ACK bits is less than or equal to 2, find the **reserved HARQ-ACK** locations and mark them on the grid.

Because the number of HARQ-ACK bits is greater than 2 in this case, skip this step.

**Step 2:** Map the coded **HARQ-ACK** bits.

The HARQ-ACK is mapped to the REs in the OFDM symbol that is available after the first consecutive DM-RS OFDM symbols. The number of REs required for HARQ-ACK is 6. Because this value is not greater than half of the number of REs available for UCI transmission, the mapping of the HARQ-ACK is distributed as shown in this figure.

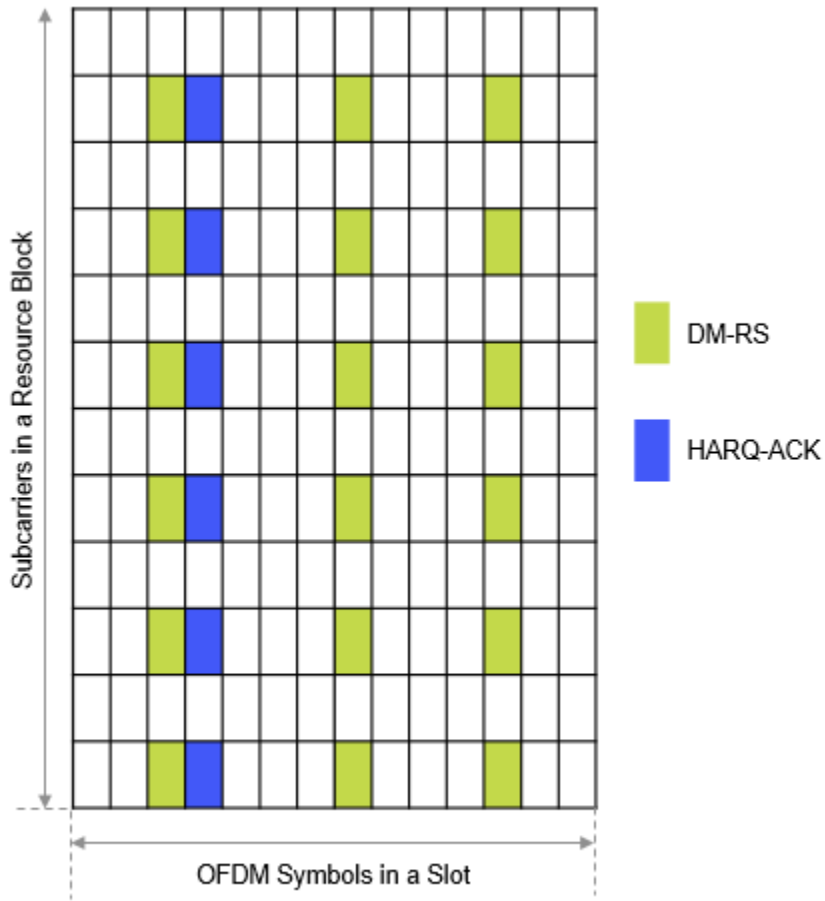


Figure 7: HARQ-ACK Locations

**Step 3:** Map the coded **CSI part 1** and coded **CSI part 2** bits similar to case 1.

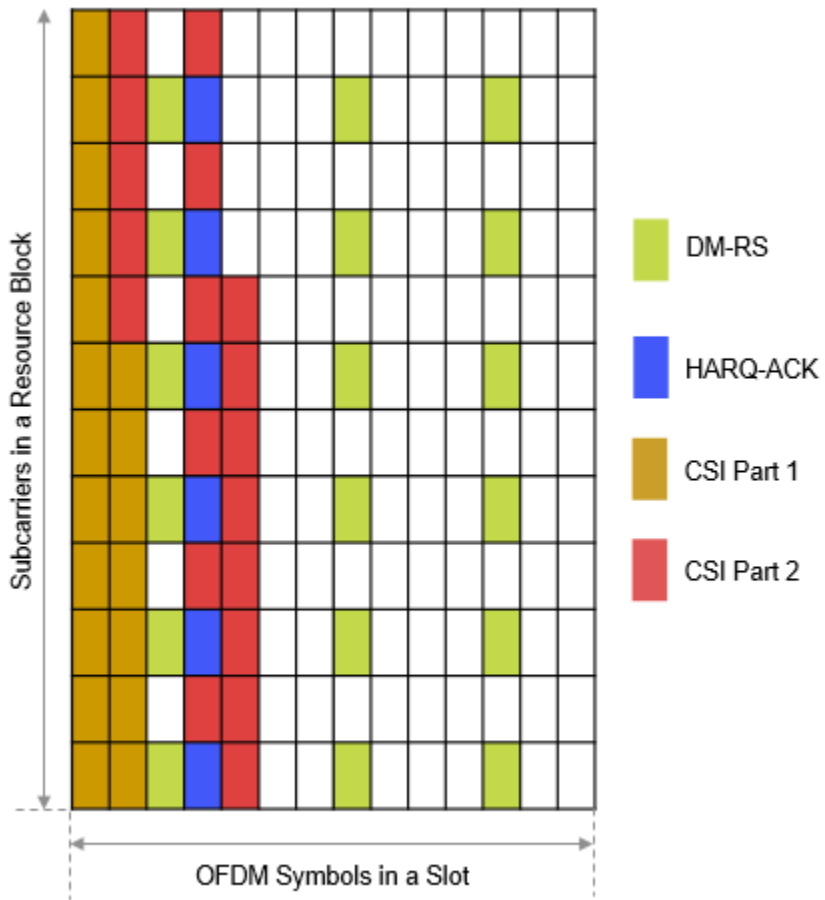


Figure 8: CSI Part 1 and CSI Part 2 Locations

**Step 4:** Map the coded **UL-SCH** data bits similar to case 1.

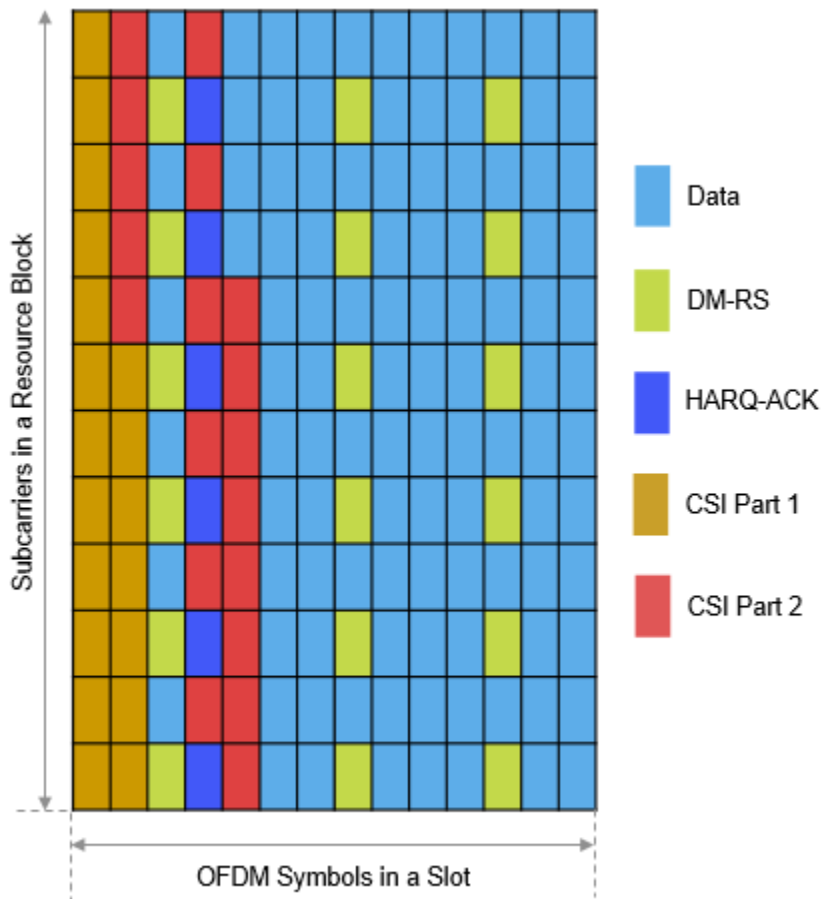


Figure 9: UL-SCH Locations

**Step 5:** When the number of HARQ-ACK bits is less than or equal to 2, map the coded **HARQ-ACK** bits.

Because the number of HARQ-ACK bits is greater than 2 in this case, skip this step.

**Step 6:** Form the **codeword**.

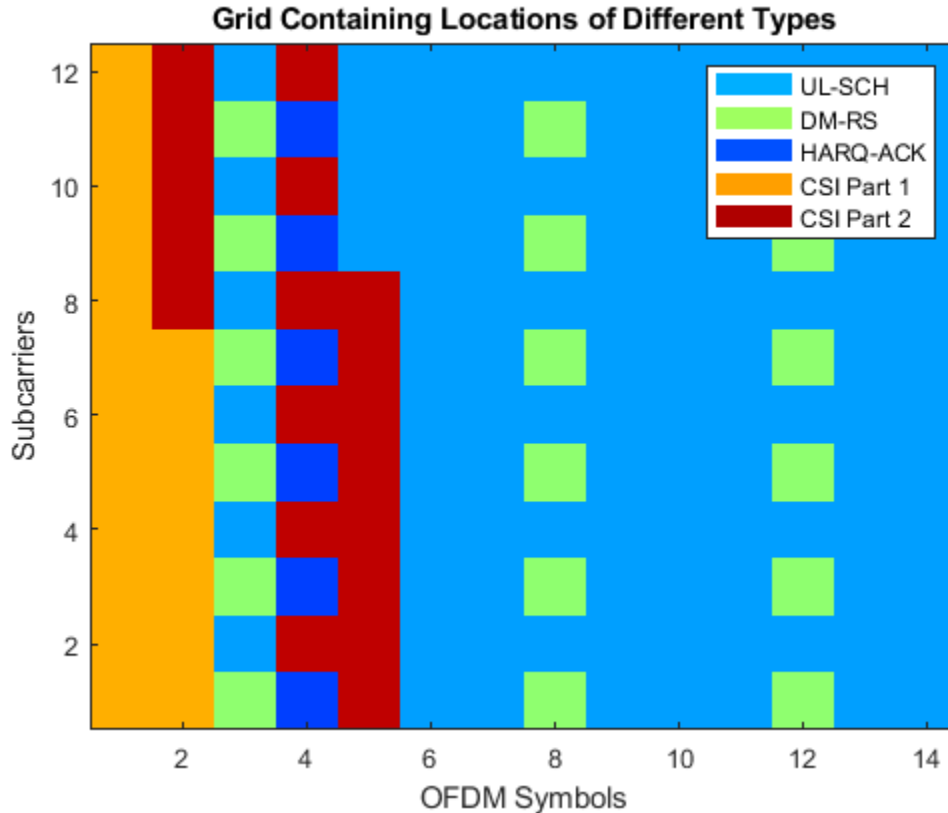
```
% Set the payload size of the HARQ-ACK bits to a value greater than 2
oack = 3; % Number of HARQ-ACK bits

% Get the UL-SCH coding information
info = nrULSCHInfo(pusch, tcr, tbs, oack, ocsi1, ocsi2);

% Set random coded UL-SCH, HARQ-ACK, CSI part 1, and CSI part 2 bits
culsch = randi([0 1], info.GULSCH, 1);
cack = randi([0 1], info.GACK, 1);
ccsi1 = randi([0 1], info.GCSI1, 1);
ccsi2 = randi([0 1], info.GCSI2, 1);

% Get the codeword and locations of each type (data and UCI)
[cw, indInfo] = nrULSCHMultiplex(pusch, tcr, tbs, culsch, cack, ccsi1, ccsi2);
```

```
% Create and plot the output grid for the first layer with predefined symbol
% values for the different types
createAndPlotGrid(carrier,pusch,cw,indInfo)
```



### Further Exploration

Change the different parameters affecting the time or frequency allocation of the PUSCH and the number of payload bits of each UCI type to vary the RE positions of the UCI types.

Enable PT-RS to vary the bit capacities of the UCI types in the codeword.

Enable intra-slot frequency hopping to observe the mapping of different UCI types in each hop.

This example shows how to generate the bit capacities of respective UCI types and perform multiplexing to generate a codeword associated with a PUSCH. The example highlights the different steps involved in multiplexing when the number of HARQ-ACK bits is less than or equal to 2 and greater than 2.

### References

- 1 3GPP TS 38.212. "NR; Multiplexing and channel coding (Release 15)." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

### Local Function

```
function createAndPlotGrid(carrier,pusch,cw,indInfo)
```

```

% Initialize the grid
qm = strcmpi(pusch.Modulation,{'pi/2-BPSK','QPSK','16QAM','64QAM','256QAM'})*[1 2 4 6 8]';
gridOFDM = complex(zeros([carrier.NSizeGrid*12 carrier.SymbolsPerSlot double(pusch.NumLayers

% Specify predefined values for each entity to get the desired visualization
chpLevel = struct;
chpLevel.PUSCH = 0.6;
chpLevel.DMRS = 1.1;
chpLevel.ACK = 0.4;
chpLevel.CSI1 = 1.5;
chpLevel.CSI2 = 2;

% Assign the codeword with predefined values
cw1 = zeros(size(cw));
cw1(indInfo.ULSCHIndices) = chpLevel.PUSCH;
cw1(indInfo.ACKIndices) = chpLevel.ACK ;
cw1(indInfo.CSI1Indices) = chpLevel.CSI1 ;
cw1(indInfo.CSI2Indices) = chpLevel.CSI2 ;

% Legend labels for the plot
labelStr = {'UL-SCH','DM-RS','HARQ-ACK','CSI Part 1','CSI Part 2'};

% Get the PUSCH and DM-RS indices
puschIndices = nrPUSCHIndices(carrier,pusch);
dmrsIndices = nrPUSCHDMRSIndices(carrier,pusch);

% Get the PT-RS indices
if pusch.EnablePTRS
    labelStr{end+1} = 'PT-RS';
    chpLevel.PTRS = 1.6;
    ptrsIndices = nrPUSCHPTRSIndices(carrier,pusch);
end

% Map the DM-RS, PT-RS, and PUSCH indices to the grid
[~,puschExtInd] = nrExtractResources(puschIndices,gridOFDM);
gridOFDM(dmrsIndices) = chpLevel.DMRS;
if pusch.TransformPrecoding
    % DFT-s-OFDM
    titleStr = "Projections of Different Types Before Transform Precoding";
    cwLen = zeros(size(puschIndices,1),pusch.NumLayers*qm);
    if pusch.EnablePTRS
        cwLen(ptrsIndices) = chpLevel.PTRS;
    end
    cwLen(cwLen==0) = cw1;
    gridOFDM(reshape(puschExtInd',[],1)) = cwLen;
else
    % CP-OFDM
    titleStr = "Grid Containing Locations of Different Types";
    if pusch.EnablePTRS
        gridOFDM(ptrsIndices) = chpLevel.PTRS;
    end
    gridOFDM(reshape(puschExtInd',[],1)) = cw1;
end

% Plot the grid
figure
map = jet(64);
cscaling = 30;

```



```

im = image(1:size(gridOFDM,2),1:size(gridOFDM,1),cscaling*abs(gridOFDM(:,:,1)));
colormap(im.Parent,map)

% Add a legend to the image
chpval = struct2cell(chpLevel);
clevels = cscaling*[chpval{:}];
N = length(clevels);
L = line(ones(N),ones(N), 'LineWidth',8); % Generate lines
% Index the color map and associate the selected colors with the lines
set(L,{'color'},mat2cell(map(min(1+clevels,length(map)),:),ones(1,N),3)) % Set the colors ac
% Create the legend
legend(labelStr{:})
axis xy
ylabel('Subcarriers')
xlabel('OFDM Symbols')
title(titleStr)

end

```

## See Also

### Functions

nrULSCHInfo | nrULSCHMultiplex

### Objects

nrCarrierConfig | nrPUSCHConfig

## Related Examples

- “NR PUSCH Resource Allocation and DM-RS and PT-RS Reference Signals” on page 2-16



# Signal Reception

---

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

Create channel estimates including beamforming.

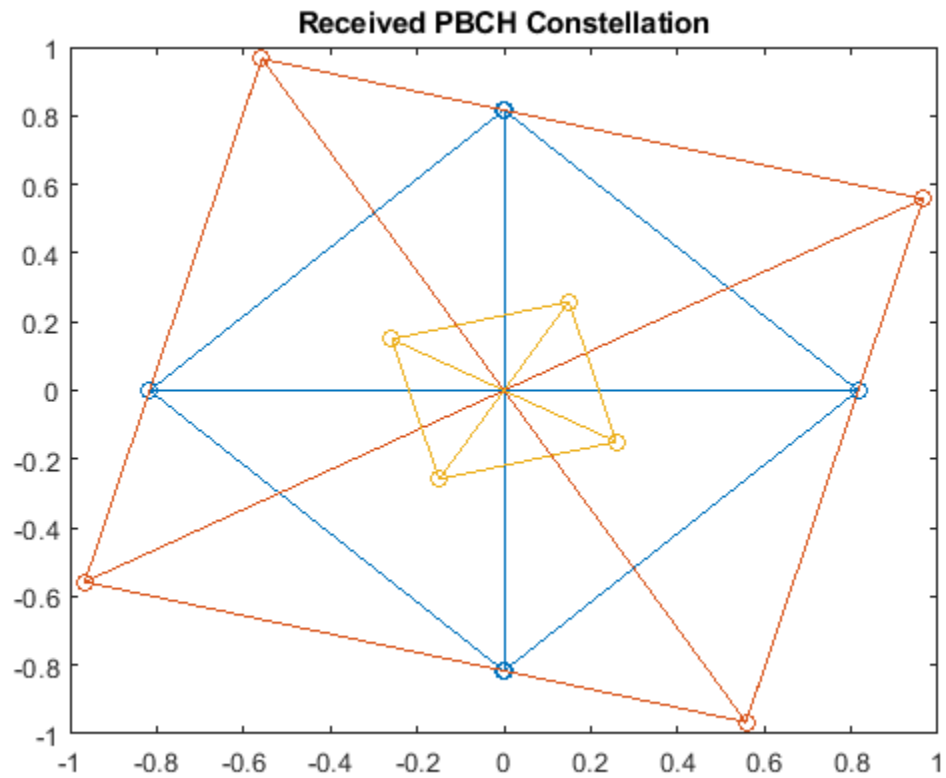
```
hEstGrid = repmat(permute(H.*F',[3 4 1 2]),[240 4]);
nEst = 0;
```

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

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

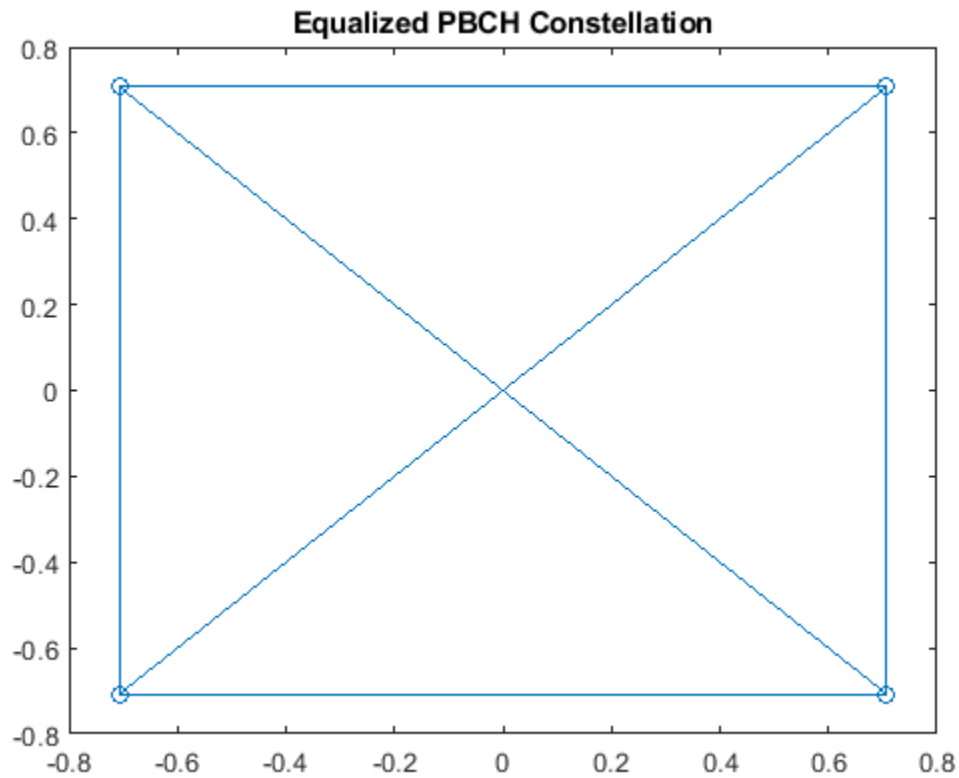
In preparation for PBCH decoding, extract symbols from the received grid and the channel estimate grid.

```
[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  
1010 ×
```

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

## See Also

### Functions

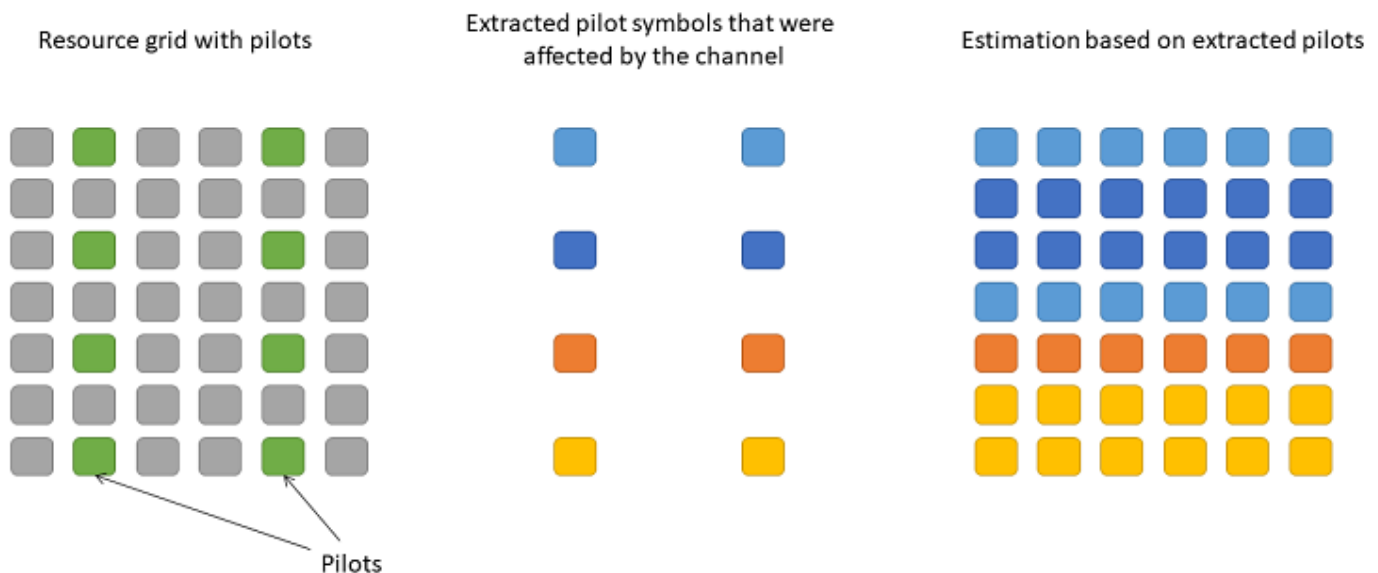
[nrEqualizeMMSE](#) | [nrExtractResources](#)

## Deep Learning Data Synthesis for 5G Channel Estimation

This example shows how to train a convolutional neural network (CNN) for channel estimation using Deep Learning Toolbox™ and data generated with 5G Toolbox™. Using the trained CNN, you perform channel estimation in single-input single-output (SISO) mode, utilizing the physical downlink shared channel (PDSCH) demodulation reference signal (DM-RS).

### Introduction

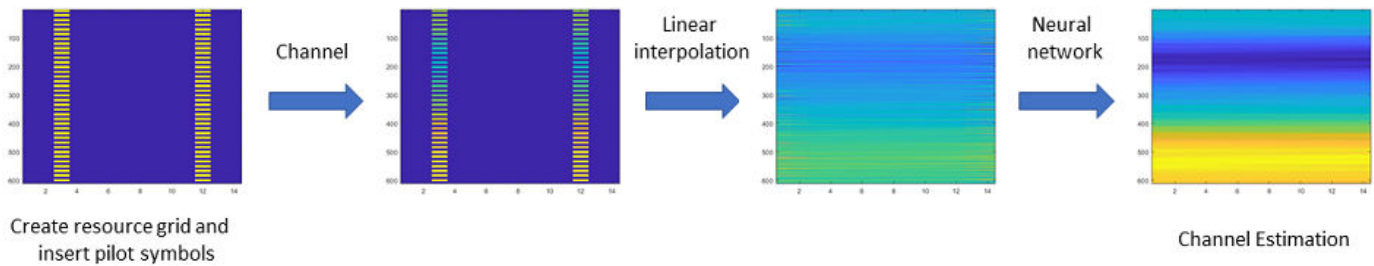
The general approach to channel estimation is to insert known reference pilot symbols into the transmission and then interpolate the rest of the channel response by using these pilot symbols.



For an example showing how to use this channel estimation approach, see “NR PDSCH Throughput” on page 1-56.

You can also use deep learning techniques to perform channel estimation. For example, by viewing the PDSCH resource grid as a 2-D image, you can turn the problem of channel estimation into an image processing problem, similar to denoising or super-resolution, where CNNs are effective.

Using 5G Toolbox, you can customize and generate standard-compliant waveforms and channel models to use as training data. Using Deep Learning Toolbox, you can use this training data to train a channel estimation CNN. This example shows how to generate such training data and how to train a channel estimation CNN. The example also shows how to use the channel estimation CNN to process images that contain linearly interpolated received pilot symbols. The example concludes by visualizing the results of the neural network channel estimator in comparison to practical and perfect estimators.



### Neural Network Training

Neural network training consists of these steps:

- Data generation
- Splitting the generated data into training and validation sets
- Defining the CNN architecture
- Specifying the training options, optimizer, and learning rate
- Training the network

Due to the large number of signals and possible scenarios, training can take several minutes. By default, training is disabled, a pretrained model is used. You can enable training by setting `trainModel` to true.

```
trainModel = false;
```

Train on a GPU if one is available. This requires Parallel Computing Toolbox™ and a CUDA® enabled NVIDIA® GPU with compute capability 3.0 or higher. You can modify this by setting the training options when calling the `trainNetwork` (Deep Learning Toolbox) function.

Data generation is set to produce 256 training examples or training data sets. This amount of data is sufficient to train a functional channel estimation network on a CPU in a reasonable time. For comparison, the pretrained model is based on 16,384 training examples.

Training data of the CNN model has a fixed size dimensionality, the network can only accept 612-by-14-by-1 grids, i.e. 612 subcarriers, 14 OFDM symbols and 1 antenna. Therefore, the model can only operate on a fixed bandwidth allocation, cyclic prefix length, and a single receive antenna.

The CNN treats the resource grids as 2-D images, hence each element of the grid must be a real number. In a channel estimation scenario, the resource grids have complex data. Therefore, the real and imaginary parts of these grids are input separately to the CNN. In this example, the training data is converted from a complex 612-by-14 matrix into a real-valued 612-by-14-by-2 matrix, where the third dimension denotes the real and imaginary components. Because you have to input the real and imaginary grids into the neural network separately when making predictions, the example converts the training data into 4-D arrays of the form 612-by-14-by-1-by-2N, where N is the number of training examples.

To ensure that the CNN does not overfit the training data, the training data is split into validation and training sets. The validation data is used for monitoring the performance of the trained neural network at regular intervals, as defined by `valFrequency`, approximately 5 per epoch. Stop training when the validation loss stops improving. In this instance, the validation data size is the same as the size of a single mini-batch due to the small size of the data set.



The returned channel estimation CNN is trained on various channel configurations based on different delay spreads, doppler shifts, and SNR ranges between 0 and 10 dB.

```

% Set the random seed for reproducibility (this has no effect if a GPU is
% used)
rng(42)

if trainModel
    % Generate the training data
    [trainData,trainLabels] = hGenerateTrainingData(256);

    % Set the number of examples per mini-batch
    batchSize = 32;

    % Split real and imaginary grids into 2 image sets, then concatenate
    trainData = cat(4,trainData(:,:,1,:),trainData(:,:,2,:));
    trainLabels = cat(4,trainLabels(:,:,1,:),trainLabels(:,:,2,:));

    % Split into training and validation sets
    valData = trainData(:,:,1:batchSize);
    valLabels = trainLabels(:,:,1:batchSize);

    trainData = trainData(:,:,batchSize+1:end);
    trainLabels = trainLabels(:,:,batchSize+1:end);

    % Validate roughly 5 times every epoch
    valFrequency = round(size(trainData,4)/batchSize/5);

    % Define the CNN structure
    layers = [ ...
        imageInputLayer([612 14 1],'Normalization','none')
        convolution2dLayer(9,64,'Padding',4)
        reluLayer
        convolution2dLayer(5,64,'Padding',2,'NumChannels',64)
        reluLayer
        convolution2dLayer(5,64,'Padding',2,'NumChannels',64)
        reluLayer
        convolution2dLayer(5,32,'Padding',2,'NumChannels',64)
        reluLayer
        convolution2dLayer(5,1,'Padding',2,'NumChannels',32)
        regressionLayer
    ];

    % Set up a training policy
    options = trainingOptions('adam', ...
        'InitialLearnRate',3e-4, ...
        'MaxEpochs',5, ...
        'Shuffle','every-epoch', ...
        'Verbose',false, ...
        'Plots','training-progress', ...
        'MiniBatchSize',batchSize, ...
        'ValidationData',{valData, valLabels}, ...
        'ValidationFrequency',valFrequency, ...
        'ValidationPatience',5);

    % Train the network. The saved structure trainingInfo contains the
    % training progress for later inspection. This structure is useful for
    % comparing optimal convergence speeds of different optimization

```

```

% methods.
[channelEstimationCNN,trainingInfo] = trainNetwork(trainData, ...
    trainLabels, layers, options);

else
% Load pretrained network if trainModel is set to false
load('trainedChannelEstimationNetwork.mat')
end

```

Inspect the composition and individual layers of the model. The model has 5 convolutional layers. The input layer expects matrices of size 612-by-14, where 612 is the number of subcarriers and 14 is the number of OFDM symbols. Each element is a real number, since the real and imaginary parts of the complex grids are input separately.

```
channelEstimationCNN.Layers
```

```
ans =
```

```
11x1 Layer array with layers:
```

1	'imageinput'	Image Input	612x14x1 images
2	'conv_1'	Convolution	64 9x9x1 convolutions with stride [1 1] and padding [0 0]
3	'relu_1'	ReLU	ReLU
4	'conv_2'	Convolution	64 5x5x64 convolutions with stride [1 1] and padding [0 0]
5	'relu_2'	ReLU	ReLU
6	'conv_3'	Convolution	64 5x5x64 convolutions with stride [1 1] and padding [0 0]
7	'relu_3'	ReLU	ReLU
8	'conv_4'	Convolution	32 5x5x64 convolutions with stride [1 1] and padding [0 0]
9	'relu_4'	ReLU	ReLU
10	'conv_5'	Convolution	1 5x5x32 convolutions with stride [1 1] and padding [0 0]
11	'regressionoutput'	Regression Output	mean-squared-error with response 'Response'

### Create Channel Model for Simulation

Set the simulation noise level in dB.

```
SNRdB = 10;
```

Load the predefined simulation parameters, including the PDSCH parameters and DM-RS configuration. The returned object `carrier` is a valid carrier configuration object and `pdsch` is a PDSCH configuration structure set for a SISO transmission.

```
[gnb,carrier,pdsch] = hDeepLearningChanEstSimParameters();
```

Create a TDL channel model and set channel parameters. To compare different channel responses of the estimators, you can change these parameters later.

```
channel = nrTDLChannel;
channel.Seed = 0;
channel.DelayProfile = 'TDL-A';
channel.DelaySpread = 3e-7;
channel.MaximumDopplerShift = 50;
```

```
% This example supports only SISO configuration
channel.NumTransmitAntennas = 1;
channel.NumReceiveAntennas = 1;
```

```
waveformInfo = nrOFDMInfo(carrier);
channel.SampleRate = waveformInfo.SampleRate;
```

Get the maximum number of delayed samples by a channel multipath component. This number is calculated from the channel path with the largest delay and the implementation delay of the channel filter. This number is needed to flush the channel filter when obtaining the received signal.

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

### Simulate PDSCH Transmission

Simulate a PDSCH transmission by performing these steps:

- Generate PDSCH resource grid
- Insert DM-RS symbols
- Perform OFDM modulation
- Send modulated waveform through the channel model
- Add white Gaussian noise
- Perform perfect timing synchronization
- Perform OFDM demodulation

```
% Generate DM-RS indices and symbols
[~,dmrsIndices,dmrsSymbols,pdschIndicesInfo] = hPDSCHResources(gnb,pdsch);
```

```
% Create PDSCH resource grid
pdschGrid = nrResourceGrid(carrier);
```

```
% Map PDSCH DM-RS symbols to the grid
pdschGrid(dmrsIndices) = pdschGrid(dmrsIndices)+dmrsSymbols;
```

```
% OFDM-modulate associated resource elements
txWaveform = nrOFDMModulate(carrier,pdschGrid);
```

To flush the channel content, append zeros at the end of the transmitted waveform. These zeros take into account any delay introduced in the channel, such as multipath and implementation delay. The number of zeros depends on the sampling rate, delay profile, and delay spread.

```
txWaveform = [txWaveform; zeros(maxChDelay,size(txWaveform,2))];
```

Send data through the TDL channel model.

```
[rxWaveform,pathGains,sampleTimes] = channel(txWaveform);
```

Add additive white Gaussian noise (AWGN) to the received time-domain waveform. To take into account sampling rate, normalize the noise power. The SNR is defined per resource element (RE) for each receive antenna (3GPP TS 38.101-4).

```
SNR = 10^(SNRdB/20); % Calculate linear noise gain
N0 = 1/(sqrt(2.0*gnb.NRxAnts*double(waveformInfo.Nfft))*SNR);
noise = N0*complex(randn(size(rxWaveform)),randn(size(rxWaveform)));
rxWaveform = rxWaveform + noise;
```

Perform perfect synchronization. To find the strongest multipath component, use the information provided by the channel.

```

% Get path filters for perfect channel estimation
pathFilters = getPathFilters(channel);
[offset,~] = nrPerfectTimingEstimate(pathGains,pathFilters);

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

OFDM-demodulate the received data to recreate the resource grid.

rxGrid = nrOFDMDemodulate(carrier,rxWaveform);

% Pad the grid with zeros in case an incomplete slot has been demodulated
[K,L,R] = size(rxGrid);
if (L < carrier.SymbolsPerSlot)
    rxGrid = cat(2,rxGrid,zeros(K,carrier.SymbolsPerSlot-L,R));
end

```

### Compare and Visualize Various Channel Estimations

You can perform and compare the results of perfect, practical, and neural network estimations of the same channel model.

To perform perfect channel estimation, use the `nrPerfectChannelEstimate` function using the value of the path gains provided by the channel.

```

estChannelGridPerfect = nrPerfectChannelEstimate(carrier,pathGains, ...
    pathFilters,offset,sampleTimes);

```

To perform practical channel estimation, use the `nrChannelEstimate` function.

```

[estChannelGrid,~] = nrChannelEstimate(carrier,rxGrid,dmrsIndices, ...
    dmrsSymbols,'CDMLengths',pdschIndicesInfo.CDMLengths);

```

To perform channel estimation using the neural network, you must interpolate the received grid. Then split the interpolated image into its real and imaginary parts and input these images together into the neural network as a single batch. Use the `predict` (Deep Learning Toolbox) function to make predictions on the real and imaginary images. Finally, concatenate and transform the results back into complex data.

```

% Interpolate the received resource grid using pilot symbol locations
interpChannelGrid = hPreprocessInput(rxGrid,dmrsIndices,dmrsSymbols);

% Concatenate the real and imaginary grids along the batch dimension
nnInput = cat(4,real(interpChannelGrid),imag(interpChannelGrid));

% Use the neural network to estimate the channel
estChannelGridNN = predict(channelEstimationCNN,nnInput);

% Convert results to complex
estChannelGridNN = complex(estChannelGridNN(:,:,,1),estChannelGridNN(:,:,,2));

```

Calculate the mean squared error (MSE) of each estimation method.

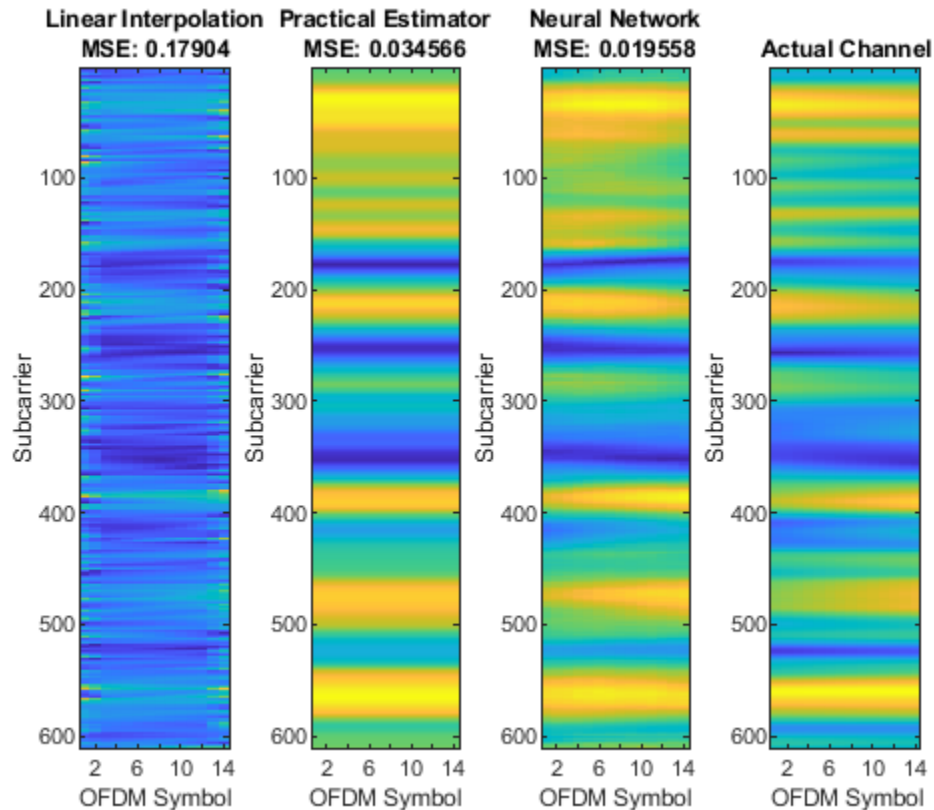
```

neural_mse = mean(abs(estChannelGridPerfect(:) - estChannelGridNN(:)).^2);
interp_mse = mean(abs(estChannelGridPerfect(:) - interpChannelGrid(:)).^2);
practical_mse = mean(abs(estChannelGridPerfect(:) - estChannelGrid(:)).^2);

```

Plot the individual channel estimations and the actual channel realization obtained from the channel filter taps. Both the practical estimator and the neural network estimator outperform linear interpolation.

```
plotChEstimates(interpChannelGrid,estChannelGrid,estChannelGridNN,estChannelGridPerfect,...
    interp_mse,practical_mse,neural_mse);
```



## References

- 1 van de Beek, Jan-Jaap, Ove Edfors, Magnus Sandell, Sarah Kate Wilson, and Per Ola Borjesson. "On Channel Estimation in OFDM Systems." In 1995 IEEE 45th Vehicular Technology Conference. Countdown to the Wireless Twenty-First Century, 2:815-19, July 1995.
- 2 Ye, Hao, Geoffrey Ye Li, and Biing-Hwang Juang. "Power of Deep Learning for Channel Estimation and Signal Detection in OFDM Systems." IEEE Wireless Communications Letters 7, no. 1 (February 2018): 114-17.
- 3 Soltani, Mehran, Vahid Pourahmadi, Ali Mirzaei, and Hamid Sheikhzadeh. "Deep Learning-Based Channel Estimation." Preprint, submitted October 13, 2018.

## Local Functions

```
function hest = hPreprocessInput(rxGrid,dmrsIndices,dmrsSymbols)
% Perform linear interpolation of the grid and input the result to the
% neural network This helper function extracts the DM-RS symbols from
% dmrsIndices locations in the received grid rxGrid and performs linear
% interpolation on the extracted pilots.
```

```

% Obtain pilot symbol estimates
dmrsRx = rxGrid(dmrsIndices);
dmrsEsts = dmrsRx .* conj(dmrsSymbols);

% Create empty grids to fill after linear interpolation
[rxDMRSGrid, hest] = deal(zeros(size(rxGrid)));
rxDMRSGrid(dmrsIndices) = dmrsSymbols;

% Find the row and column coordinates for a given DMRS configuration
[rows,cols] = find(rxDMRSGrid ~= 0);
dmrsSubs = [rows,cols,ones(size(cols))];
[l_hest,k_hest] = meshgrid(1:size(hest,2),1:size(hest,1));

% Perform linear interpolation
f = scatteredInterpolant(dmrsSubs(:,2),dmrsSubs(:,1),dmrsEsts);
hest = f(l_hest,k_hest);

end

function [trainData,trainLabels] = hGenerateTrainingData(dataSize)
% Generate training data examples for channel estimation
% Run dataSize number of iterations to create random channel configurations
% and pass an OFDM-modulated fixed PDSCH grid with only the DM-RS symbols
% inserted. Perform perfect timing synchronization and OFDM demodulation,
% extracting the pilot symbols and performing linear interpolation at each
% iteration. Use perfect channel information to create the
% label data. The function returns 2 arrays - the training data and labels.

fprintf('Starting data generation...\n')

% List of possible channel profiles
delayProfiles = {'TDL-A', 'TDL-B', 'TDL-C', 'TDL-D', 'TDL-E'};

[simParameters, carrier, pdsch] = hDeepLearningChanEstSimParameters();

% Create the channel model object
nTxAnts = simParameters.NTxAnts;
nRxAnts = simParameters.NRxAnts;

channel = nrTDLChannel; % TDL channel object
channel.NumTransmitAntennas = nTxAnts;
channel.NumReceiveAntennas = nRxAnts;

% Use the value returned from <matlab:edit('nrOFDMInfo') nrOFDMInfo> to
% set the channel model sample rate
waveformInfo = nrOFDMInfo(carrier);
channel.SampleRate = waveformInfo.SampleRate;

% Get the maximum number of delayed samples by a channel multipath
% component. This number is calculated from the channel path with the largest
% delay and the implementation delay of the channel filter, and is required
% to flush the channel filter to obtain the received signal.
chInfo = info(channel);
maxChDelay = ceil(max(chInfo.PathDelays*channel.SampleRate)) + chInfo.ChannelFilterDelay;

% Return DM-RS indices and symbols
[~,dmrsIndices,dmrsSymbols,~] = hPDSCHResources(simParameters,pdsch);

```

```

% PDSCH mapping in grid associated with PDSCH transmission period
pdschGrid = nrResourceGrid(carrier,nTxAnts);

% PDSCH DM-RS precoding and mapping
[~,dmrsAntIndices] = nrExtractResources(dmrsIndices,pdschGrid);
pdschGrid(dmrsAntIndices) = pdschGrid(dmrsAntIndices) + dmrsSymbols;

% OFDM modulation of associated resource elements
txWaveform_original = nrOFDMModulate(carrier,pdschGrid);

% Acquire linear interpolator coordinates for neural net preprocessing
[rows,cols] = find(pdschGrid ~= 0);
dmrsSubs = [rows, cols, ones(size(cols))];
hest = zeros(size(pdschGrid));
[l_hest,k_hest] = meshgrid(1:size(hest,2),1:size(hest,1));

% Preallocate memory for the training data and labels
numExamples = dataSize;
[trainData, trainLabels] = deal(zeros([612 14 2 numExamples]));

% Main loop for data generation, iterating over the number of examples
% specified in the function call. Each iteration of the loop produces a
% new channel realization with a random delay spread, doppler shift,
% and delay profile. Every perturbed version of the transmitted
% waveform with the DM-RS symbols is stored in trainData, and the
% perfect channel realization in trainLabels.
for i = 1:numExamples
    % Release the channel to change nontunable properties
    channel.release

    % Pick a random seed to create different channel realizations
    channel.Seed = randi([1001 2000]);

    % Pick a random delay profile, delay spread, and maximum doppler shift
    channel.DelayProfile = string(delayProfiles(randi([1 numel(delayProfiles)])));
    channel.DelaySpread = randi([1 300])*1e-9;
    channel.MaximumDopplerShift = randi([5 400]);

    % Send data through the channel model. Append zeros at the end of
    % the transmitted waveform to flush channel content. These zeros
    % take into account any delay introduced in the channel, such as
    % multipath delay and implementation delay. This value depends on
    % the sampling rate, delay profile, and delay spread
    txWaveform = [txWaveform_original; zeros(maxChDelay, size(txWaveform_original,2))];
    [rxWaveform,pathGains,sampleTimes] = channel(txWaveform);

    % Add additive white Gaussian noise (AWGN) to the received time-domain
    % waveform. To take into account sampling rate, normalize the noise power.
    % The SNR is defined per RE for each receive antenna (3GPP TS 38.101-4).
    SNRdB = randi([0 10]); % Random SNR values between 0 and 10 dB
    SNR = 10^(SNRdB/20); % Calculate linear noise gain
    N0 = 1/(sqrt(2.0*nRxAnts*double(waveformInfo.Nfft))*SNR);
    noise = N0*complex(randn(size(rxWaveform)),randn(size(rxWaveform)));
    rxWaveform = rxWaveform + noise;

    % Perfect synchronization. Use information provided by the channel
    % to find the strongest multipath component
    pathFilters = getPathFilters(channel); % Get path filters for perfect channel estimation

```

```

[offset,~] = nrPerfectTimingEstimate(pathGains,pathFilters);

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

% Perform OFDM demodulation on the received data to recreate the
% resource grid, including padding in case practical
% synchronization results in an incomplete slot being demodulated
rxGrid = nrOFDMDemodulate(carrier,rxWaveform);
[K,L,R] = size(rxGrid);
if (L < carrier.SymbolsPerSlot)
    rxGrid = cat(2,rxGrid,zeros(K,carrier.SymbolsPerSlot-L,R));
end

% Perfect channel estimation, using the value of the path gains
% provided by the channel. This channel estimate does not
% include the effect of transmitter precoding
estChannelGridPerfect = nrPerfectChannelEstimate(carrier,pathGains, ...
    pathFilters,offset,sampleTimes);

% Linear interpolation
dmrsRx = rxGrid(dmrsIndices);
dmrsEsts = dmrsRx .* conj(dmrsSymbols);
f = scatteredInterpolant(dmrsSubs(:,2),dmrsSubs(:,1),dmrsEsts);
hest = f(l_hest,k_hest);

% Split interpolated grid into real and imaginary components and
% concatenate them along the third dimension, as well as for the
% true channel response
rx_grid = cat(3, real(hest), imag(hest));
est_grid = cat(3, real(estChannelGridPerfect), ...
    imag(estChannelGridPerfect));

% Add generated training example and label to the respective arrays
trainData(:,:,:,i) = rx_grid;
trainLabels(:,:,:,i) = est_grid;

% Data generation tracker
if mod(i,round(numExamples/25)) == 0
    fprintf('%3.2f%% complete\n',i/numExamples*100);
end
end
fprintf('Data generation complete!\n')
end

function plotChEstimates(interpChannelGrid,estChannelGrid,estChannelGridNN,estChannelGridPerfect
    interp_mse,practical_mse,neural_mse)
% Plot the different channel estimates and display the measured MSE

figure

subplot(1,4,1)
imagesc(abs(interpChannelGrid));
xlabel('OFDM Symbol');
ylabel('Subcarrier');
title({'Linear Interpolation', ['MSE: ', num2str(interp_mse)]});

subplot(1,4,2)
imagesc(abs(estChannelGrid));

```



```
xlabel('OFDM Symbol');
ylabel('Subcarrier');
title({'Practical Estimator', ['MSE: ', num2str(practical_mse)]});

subplot(1,4,3)
imagesc(abs(estChannelGridNN));
xlabel('OFDM Symbol');
ylabel('Subcarrier');
title({'Neural Network', ['MSE: ', num2str(neural_mse)]});

subplot(1,4,4)
imagesc(abs(estChannelGridPerfect));
xlabel('OFDM Symbol');
ylabel('Subcarrier');
title({'Actual Channel'});
end
```

## See Also

### Functions

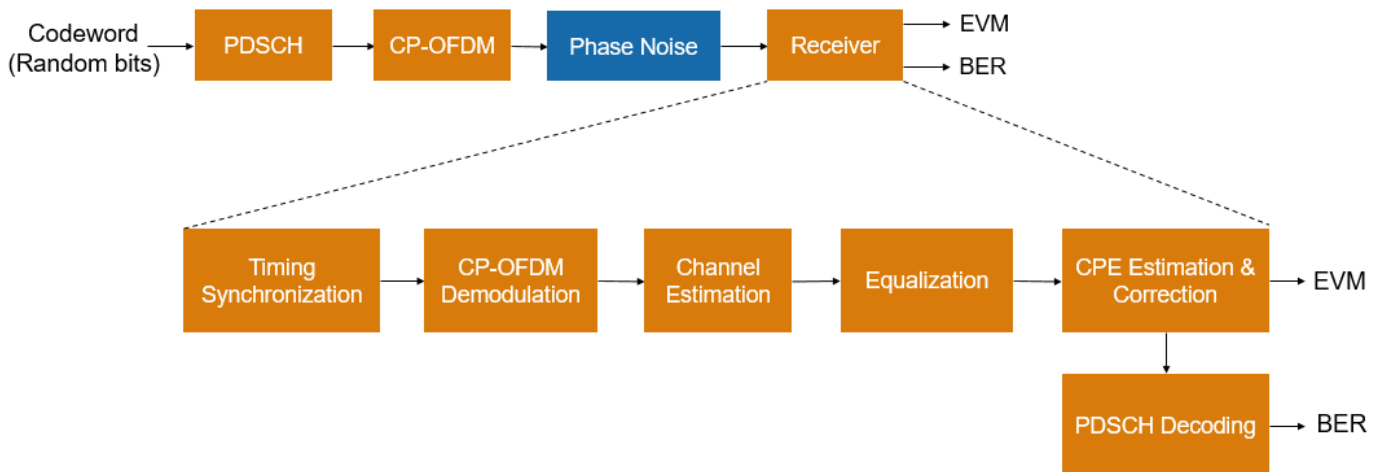
nrPerfectChannelEstimate | nrChannelEstimate | predict | trainNetwork

## NR Phase Noise Modeling and Compensation

This example demonstrates the impact of phase noise in a 5G OFDM system and shows how to use phase tracking reference signal (PT-RS) in compensating the common phase error (CPE). The example measures error vector magnitude (EVM) and bit error rate (BER) with and without CPE compensation.

### Introduction

In 5G NR, 3GPP introduces a new reference signal, named phase tracking reference signal (PT-RS), to deal with oscillator noise. The noise incurred in the oscillators results in phase modulation of the information signal, leading to significant changes in the frequency spectrum and timing properties of the information signal. This phenomenon related to oscillators is called phase noise. Phase noise produced in local oscillators introduces a significant degradation at millimeter wave (mmWave) frequencies, depending on the power spectral density of phase noise. Phase noise leads to CPE and intercarrier interference (ICI). CPE leads to an identical rotation of a received symbol in each subcarrier. ICI leads to a loss of orthogonality between the subcarriers. Due to the distributed structure of PT-RS in frequency domain, the example primarily uses PT-RS to estimate and minimize the effect of CPE on system performance. The example applies phase noise on the waveform consisting of physical downlink shared channel (PDSCH) and shows the change in EVM and BER without and with CPE compensation. This figure shows the processing chain implemented in this example.



### Phase Noise Modeling

The oscillator power spectral density models the phase noise. This example uses a multipole zero model to approximate the power spectral density of the oscillator. Use the `PNModel` field of the `simParameters` structure to select the phase noise model: 'A', 'B', or 'C'. Parameter sets 'A' and 'B' are obtained from practical oscillators operating at 30 GHz and 60 GHz, respectively, as described in TDoc R1-163984. The parameter set 'C' is obtained from the practical oscillator operating at 29.55 GHz, as described in TR 38.803 Section 6.1.10.

The example uses a carrier with a subcarrier spacing of 60 kHz for a transmission bandwidth of 50 MHz.

```
% Configure carrier
carrier = nrCarrierConfig;
```

```
carrier.SubcarrierSpacing = 60;
carrier.CyclicPrefix = 'normal';
carrier.NSizeGrid = 66;

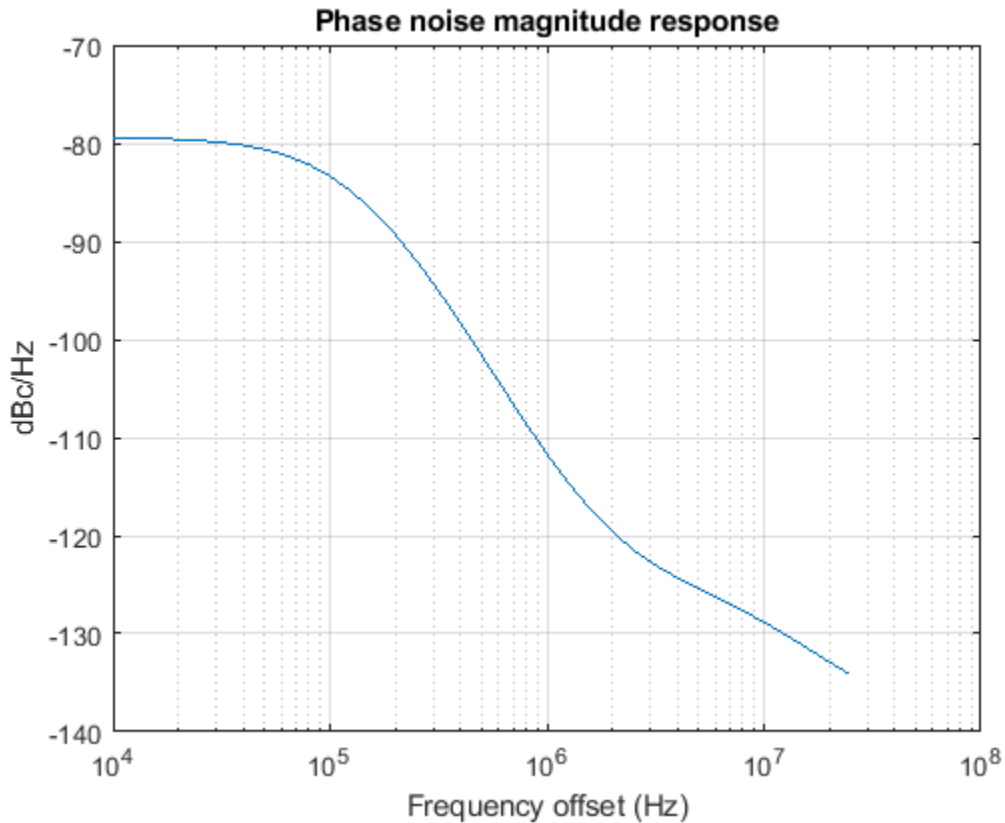
% Set the operating frequency and choose the phase noise model
simParameters = [];
simParameters.Fc = 30e9; % Frequency in Hz
simParameters.PNModel = 'A'; % 'A' (TDoc R1-163984 Set A), 'B' (TDoc R1-163984 Set B), 'C' (TR 38.101-13.2.1)

% Get the sample rate
ofdmInfo = nrOFDMInfo(carrier);
sr = ofdmInfo.SampleRate;

% Phase noise level
foffsetLog = (4:0.1:log10(sr/2)); % Model offset from 1e4 Hz to sr/2 Hz
foffset = 10.^foffsetLog; % Linear frequency offset
pn_PSD = hPhaseNoisePoleZeroModel(foffset,simParameters.Fc,simParameters.PNModel); % dBc/Hz

% Set phase noise level
pnoise = comm.PhaseNoise('FrequencyOffset',foffset,'Level',pn_PSD,'SampleRate',sr);
pnoise.RandomStream = "mt19937ar with seed";

% Visualize spectrum mask of phase noise
figure
semilogx(foffset,pn_PSD)
xlabel('Frequency offset (Hz)')
ylabel('dBc/Hz')
title('Phase noise magnitude response')
grid on
```



### Configure PDSCH

The example configures PDSCH occupying the complete carrier with modulation scheme set to '64QAM' and number of layers set to 1. The example defaults to a single layer and a single codeword of random uncoded bits.

```
% Set PDSCH parameters
pdsch = nrPDSCHConfig;
pdsch.PRBSet = 0:carrier.NSizeGrid-1;
pdsch.SymbolAllocation = [0 14];
pdsch.Modulation = '64QAM';
pdsch.NumLayers = 1;
pdsch.MappingType = 'A';
pdsch.NID = 1;
pdsch.RNTI = 2;

% Set DM-RS parameters
pdsch.DMRS.DMRSConfigurationType = 1;
pdsch.DMRS.DMRSTypeAPosition = 2;
pdsch.DMRS.DMRSAdditionalPosition = 0;
pdsch.DMRS.DMRSLength = 1;
pdsch.DMRS.DMRSPortSet = [];
pdsch.DMRS.NumCDMGroupsWithoutData = 1;
pdsch.DMRS.NIDNSCID = 1;
pdsch.DMRS.NSCID = 0;

% Set PT-RS parameters
```

```
pdsch.EnablePTRS = 1;
pdsch.PTRS.TimeDensity = 1;
pdsch.PTRS.FrequencyDensity = 2;
pdsch.PTRS.REOffset = '00';
pdsch.PTRS.PTRSPortSet = [];
```

### Generate Waveform

The waveform is generated for 2 frames and the field NumFrames of simParameters structure controls the number of frames of the waveform. The steps involved are:

- Generate random codeword with the bit capacity of PDSCH
- Get the PDSCH symbols for the random codeword and map them to grid
- Generate and map DM-RS symbols to grid
- Generate and map PT-RS symbols to grid
- Perform OFDM modulation for the complete grid of all frames

```
% Number of frames to generate the waveform
simParameters.NumFrames = 2;

% Get the number of slots in the waveform and number of symbols in a slot
numSlots = carrier.SlotsPerFrame*simParameters.NumFrames;
nSlotSymb = carrier.SymbolsPerSlot;

% Initialize the grid for specified number of frames
txGrid = zeros(carrier.NSizeGrid*12,nSlotSymb*numSlots,pdsch.NumLayers);

% Processing loop
txbits = [];
rng('default')
for slotIdx = 0:numSlots - 1
    % Set slot number
    carrier.NSlot = slotIdx;

    % Get PDSCH indices and structural information
    [pdschInd,pdschIndicesInfo] = nrPDSCHIndices(carrier,pdsch);

    % Generate random codeword(s)
    numCW = pdsch.NumCodewords; % Number of codewords
    data = cell(1,numCW);
    for i = 1:numCW
        data{i} = randi([0 1],pdschIndicesInfo.G(i),1);
        txbits = [txbits; data{i}]; %#ok<AGROW>
    end

    % Get modulated symbols
    pdschSym = nrPDSCH(carrier,pdsch,data);

    % Get DM-RS symbols and indices
    dmrsSym = nrPDSCHDMRS(carrier,pdsch);
    dmrsInd = nrPDSCHDMRSIndices(carrier,pdsch);

    % Get PT-RS symbols and indices
    ptrsSym = nrPDSCHPTRS(carrier,pdsch);
    ptrsInd = nrPDSCHPTRSIndices(carrier,pdsch);
```

```

% Resource element mapping to slot grid
slotGrid = nrResourceGrid(carrier,pdsch.NumLayers);
slotGrid(pdschInd) = pdschSym;
slotGrid(dmrsInd) = dmrsSym;
slotGrid(ptrsInd) = ptrsSym;

% Generate txGrid for all frames by mapping slotGrid at respective
% locations
txGrid(:,slotIdx*nSlotSymb+1:(slotIdx+1)*(nSlotSymb),:) = slotGrid;
end

% Perform OFDM modulation
carrier.NSlot = 0; % Reset the slot number to 0 for OFDM modulation
txWaveform = nrOFDMModulate(carrier,txGrid);

```

**Apply Phase Noise**

Apply phase noise to the transmitted waveform. To clearly observe the impact of phase noise, the example does not add any thermal noise or channel affects in addition to phase noise. The example applies the same phase noise to all the layers.

```

rxWaveform = zeros(size(txWaveform),'like',txWaveform);
for i = 1:size(txWaveform,2)
    rxWaveform(:,i) = pnoise(txWaveform(:,i));
    release(pnoise)
end

```

**Receiver**

Before returning the equalized PDSCH symbols and decoded bits, the receiver performs these steps:

- Timing synchronization
- OFDM demodulation
- Channel estimation
- Equalization
- CPE estimation and correction
- PDSCH decoding

For the CPE estimation and correction step, the receiver uses the logical field `CompensateCPE` of the `simParameters` structure. Because the example does not use a propagation channel, the steps of timing synchronization, channel estimation, and equalization are not strictly necessary. However, these steps help investigate the phase noise effects if you introduce a channel.

The example shows the equalized constellation symbols, EVM, and bit error rate, with and without CPE compensation.

**Case 1: Without CPE compensation**

To disable the CPE compensation, set the field `CompensateCPE` of `simParameters` structure to 0.

```

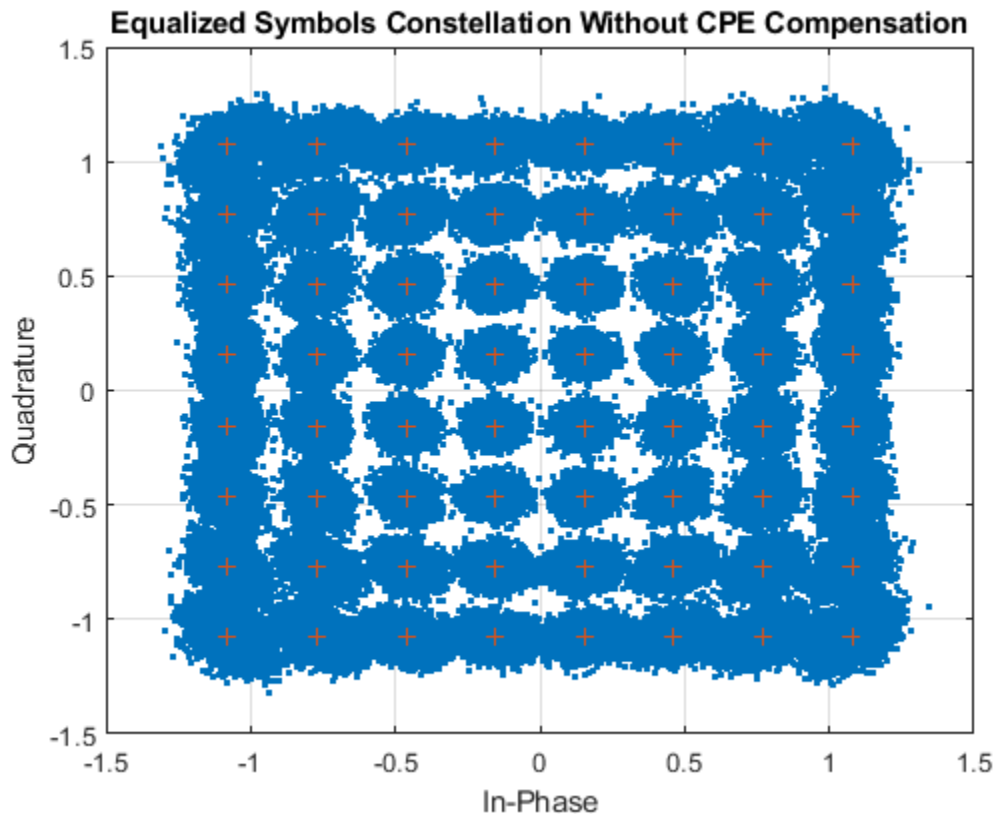
simParameters.CompensateCPE = 0;
[eqSymbols,rxbits] = practicalReceiver(carrier,pdsch,simParameters,rxWaveform);
refSymbols = getConstellationPoints(pdsch);
% Display the constellation diagram
figure

```

```

plot(eqSymbols, '.')
hold on
plot(refSymbols, '+')
title('Equalized Symbols Constellation Without CPE Compensation')
grid on
xlabel('In-Phase')
ylabel('Quadrature')

```



```

% Display RMS EVM
evm = comm.EVM('ReferenceSignalSource','Estimated from reference constellation','ReferenceConstellation',eqSymbols);
fprintf('RMS EVM (in percent) for equalized symbols without CPE compensation: %f%% \n',evm(equivSymbols));

```

RMS EVM (in percent) for equalized symbols without CPE compensation: 5.928819%

```

% Display bit error rate
errorRate = nnz(rxbits-txbits)/numel(txbits);
fprintf('Bit error rate without CPE compensation: %f \n',errorRate)

```

Bit error rate without CPE compensation: 0.000503

### Case 2: With CPE compensation

To enable the CPE compensation, set the field `CompensateCPE` of `simParameters` structure to 0. Use PT-RS to estimate the CPE at all the OFDM symbol locations in a slot. Correct the CPE in the OFDM symbol locations within the range of PT-RS OFDM symbols.

```

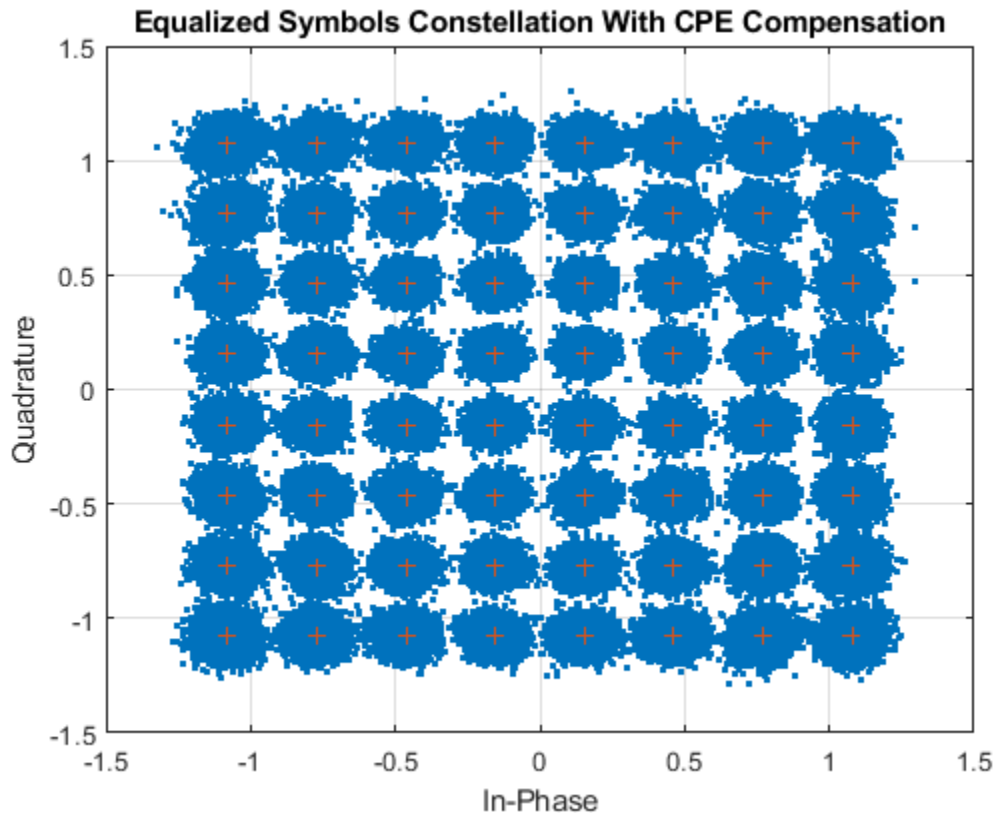
simParameters.CompensateCPE = 1;
[eqSymbolsCPE,rxbitsCPE] = practicalReceiver(carrier,pdsch,simParameters,rxWaveform);

```

```

% Display the constellation diagram
figure
plot(eqSymbolsCPE, '.')
hold on
plot(refSymbols, '+')
title('Equalized Symbols Constellation With CPE Compensation')
grid on
xlabel('In-Phase')
ylabel('Quadrature')

```



```

% Display RMS EVM
fprintf('RMS EVM (in percent) for equalized symbols with CPE compensation: %f%% \n', evm(eqSymbolsCPE))

RMS EVM (in percent) for equalized symbols with CPE compensation: 4.797706%

% Display bit error rate
errorRateCPE = nnz(rxbitsCPE-txbits)/numel(txbits);
fprintf('Bit error rate with CPE compensation: %f \n', errorRateCPE)

Bit error rate with CPE compensation: 0.000066

```

### Further Exploration

- To visualize the impact of phase noise, change the carrier frequency, subcarrier spacing, number of resource blocks, modulation scheme, and the number of frames.
- To see the effects of phase noise on the constellation, change the phase noise model.
- To analyze the effect of CPE compensation with different configurations, change the time and frequency density of PTRS.



- Visualize the impacts of phase noise by including thermal noise and channel models.

## Summary

This example demonstrates the impact of phase noise and shows how to estimate and correct CPE with PT-RS. The example also shows that CPE compensation reduces the EVM and improves bit error rate. The displayed constellation plot shows huge ICI in mmWave frequencies, indicating that ICI compensation needs to be performed in addition to CPE compensation.

## Local Functions

```
function [eqSymbols,rxbits] = practicalReceiver(carrier,pdsch,params,rxWaveform)
% Returns equalized modulated symbols after performing the timing
% estimation, OFDM demodulation, channel estimation, MMSE equalization,
% CPE estimation and correction, and PDSCH decoding.

% Get the current slot number, number of slots, number of symbols
% per slot, and total number of symbols
nSlot = carrier.NSlot;
numSlots = carrier.SlotsPerFrame*params.NumFrames;
nSlotSymb = carrier.SymbolsPerSlot;
numTotalSymbols = numSlots*nSlotSymb;

% Get reference grid with DM-RS symbols
dmrsSymCell = cell(1,numSlots);
dmrsIndCell = cell(1,numSlots);
refGrid = zeros(carrier.NSizeGrid*12,numTotalSymbols,pdsch.NumLayers);
for NSlot = 0:numSlots-1
    carrier.NSlot = NSlot;
    slotGrid = nrResourceGrid(carrier,pdsch.NumLayers);
    dmrsSymCell{NSlot+1} = nrPDSCHDMRS(carrier,pdsch);
    dmrsIndCell{NSlot+1} = nrPDSCHDMRSIndices(carrier,pdsch);
    slotGrid(dmrsIndCell{NSlot+1}) = dmrsSymCell{NSlot+1};
    refGrid(:,NSlot*nSlotSymb+1:(NSlot+1)*(nSlotSymb),:) = slotGrid;
end

% Perform timing estimation and correction
carrier.NSlot = nSlot;
offset = nrTimingEstimate(carrier,rxWaveform,refGrid);
waveformSync = rxWaveform(1+offset:end,:);

% Perform OFDM demodulation on the received data to recreate the
% resource grid, including padding in the event that practical
% synchronization results in an incomplete slots being demodulated
rxGrid = nrOFDMDemodulate(carrier,waveformSync);
[K,L,R] = size(rxGrid);
if (L < numTotalSymbols)
    rxGrid = cat(2,rxGrid,zeros(K,numTotalSymbols-L,R));
end

% Declare storage variables
eqSymbols = []; % equalized symbols for constellation plot
rxbits = [];

for NSlot = 0:numSlots-1
    % Extract grid for current slot
    currentGrid = rxGrid(:,NSlot*nSlotSymb+(1:nSlotSymb),:);
```

```

% Get the PDSCH resources
carrier.NSlot = NSlot;
dmrsSymbols = dmrsSymCell{NSlot+1};
dmrsIndices = dmrsIndCell{NSlot+1};
ptrsSymbols = nrPDSCHPTRS(carrier,pdsch);
ptrsIndices = nrPDSCHPTRSIndices(carrier,pdsch);
[pdschIndices,pdschIndicesInfo] = nrPDSCHIndices(carrier,pdsch);

% Channel estimation
[estChannelGrid,noiseEst] = nrChannelEstimate(currentGrid,dmrsIndices,dmrsSymbols,"CDMLen

% Get PDSCH resource elements from the received grid
[pdschRx,pdschHest] = nrExtractResources(pdschIndices,currentGrid,estChannelGrid);

% Equalization
pdschEq = nrEqualizeMMSE(pdschRx,pdschHest,noiseEst);

% Common phase error (CPE) estimation and correction
if params.CompensateCPE
    % Initialize temporary grid to store equalized symbols
    tempGrid = nrResourceGrid(carrier,pdsch.NumLayers);

    % Extract PT-RS symbols from received grid and estimated
    % channel grid
    [ptrsRx,ptrsHest,~,~,~,ptrsLayerIndices] = nrExtractResources(ptrsIndices,currentGrid,estChannelGrid);

    % Equalize PT-RS symbols and map them to tempGrid
    ptrsEq = nrEqualizeMMSE(ptrsRx,ptrsHest,noiseEst);
    tempGrid(ptrsLayerIndices) = ptrsEq;

    % Estimate the residual channel at the PT-RS locations in
    % tempGrid
    cpe = nrChannelEstimate(tempGrid,ptrsIndices,ptrsSymbols);

    % Sum estimates across subcarriers, receive antennas, and
    % layers. Then, get the CPE by taking the angle of the
    % resultant sum
    cpe = angle(sum(cpe,[1 3 4]));

    % Map the equalized PDSCH symbols to tempGrid
    tempGrid(pdschIndices) = pdschEq;

    % Correct CPE in each OFDM symbol within the range of reference
    % PT-RS OFDM symbols
    if numel(pdschIndicesInfo.PTRSSymbolSet) > 0
        symLoc = pdschIndicesInfo.PTRSSymbolSet(1)+1:pdschIndicesInfo.PTRSSymbolSet(end)-1;
        tempGrid(:,symLoc,:) = tempGrid(:,symLoc,:).*exp(-1i*cpe(symLoc));
    end

    % Extract PDSCH symbols
    pdschEq = tempGrid(pdschIndices);
end

% Store the equalized symbols and output them for all the slots
eqSymbols = [eqSymbols; pdschEq]; %#ok<AGROW>

% Decode the PDSCH symbols and get the hard bits
eqbits = nrPDSCHDecode(carrier,pdsch,pdschEq);

```

```

    for i = 1:numel(eqbits)
        rxbits = [rxbits; double(eqbits{i}<0)]; %#ok<AGROW>
    end

end

end

function sym = getConstellationPoints(pdsch)
%getConstellationPoints Constellation points
% SYM = getConstellationPoints(PDSCH) returns the constellation points
% SYM based on modulation schemes provided in PDSCH configuration object.

sym = [];
modulation = string(pdsch.Modulation); % Convert modulation scheme to string type
ncw = pdsch.NumCodewords; % Number of codewords
if ncw == 2 && (numel(modulation) == 1)
    modulation(end+1) = modulation(1);
end
% Get the constellation points
for cwIndex = 1:ncw
    qm = strcmpi(modulation(cwIndex),{'QPSK','16QAM','64QAM','256QAM'})*[2 4 6 8]';
    sym = [sym; nrSymbolModulate(reshape(de2bi(0:2^qm-1,qm,'left-msb'),[],1),modulation(cwIndex))];
end

end

end

```

## See Also

## More About

- “EVM Measurement of 5G NR PDSCH Waveforms” on page 6-59



# End-To-End Simulation

---

## 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;

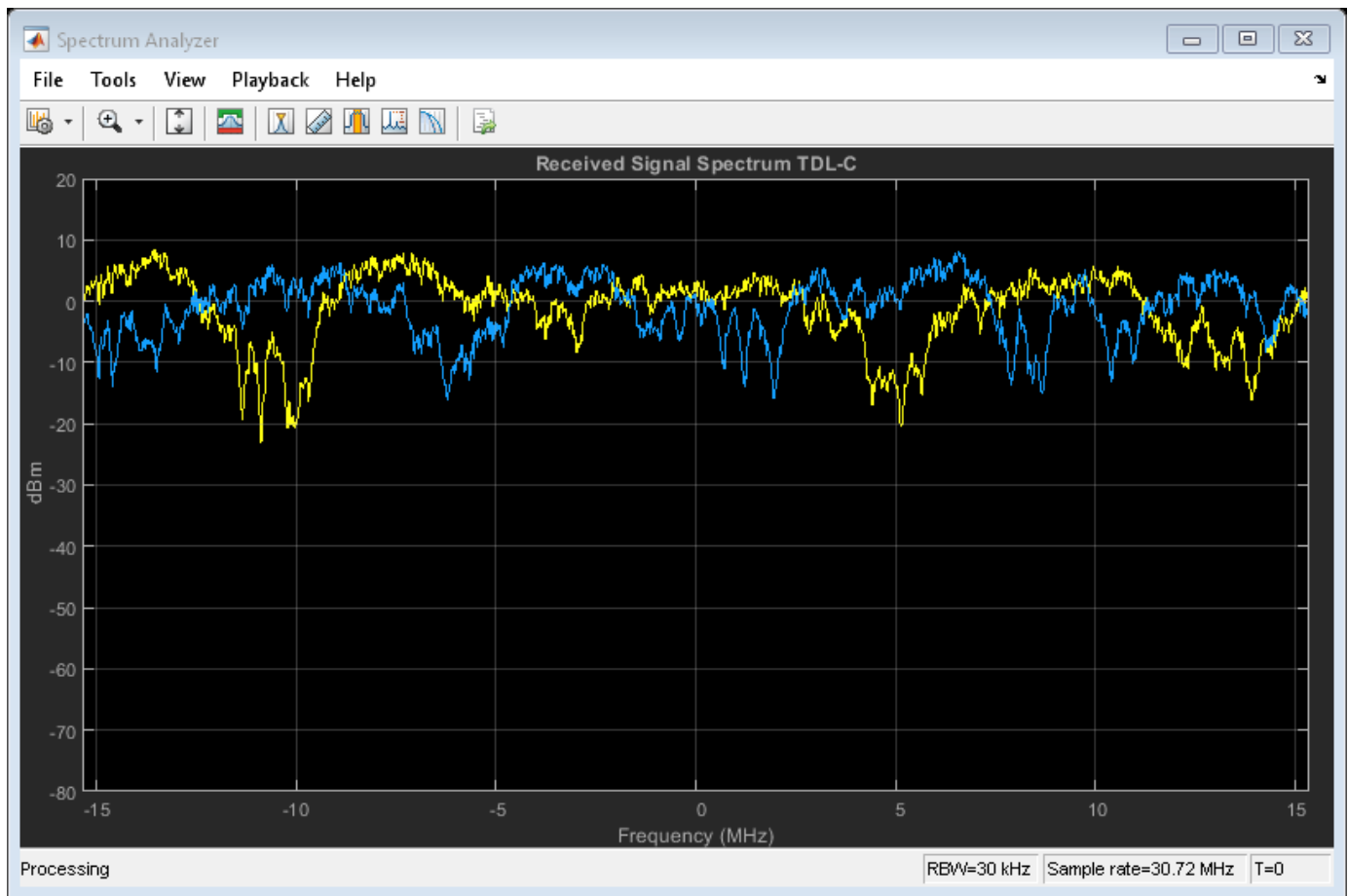
txWaveform = complex(randn(T,Nt),randn(T,Nt));
```

Transmit the input waveform through the channel.

```
rxWaveform = tdl(txWaveform);
```

Plot the received waveform spectrum.

```
analyzer = dsp.SpectrumAnalyzer('SampleRate',tdl.SampleRate,...
    'AveragingMethod','Exponential','ForgettingFactor',0.99 );
analyzer.Title = ['Received Signal Spectrum ' tdl.DelayProfile];
analyzer(rxWaveform);
```



## See Also

### Objects

nrTDLChannel

## 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;
```

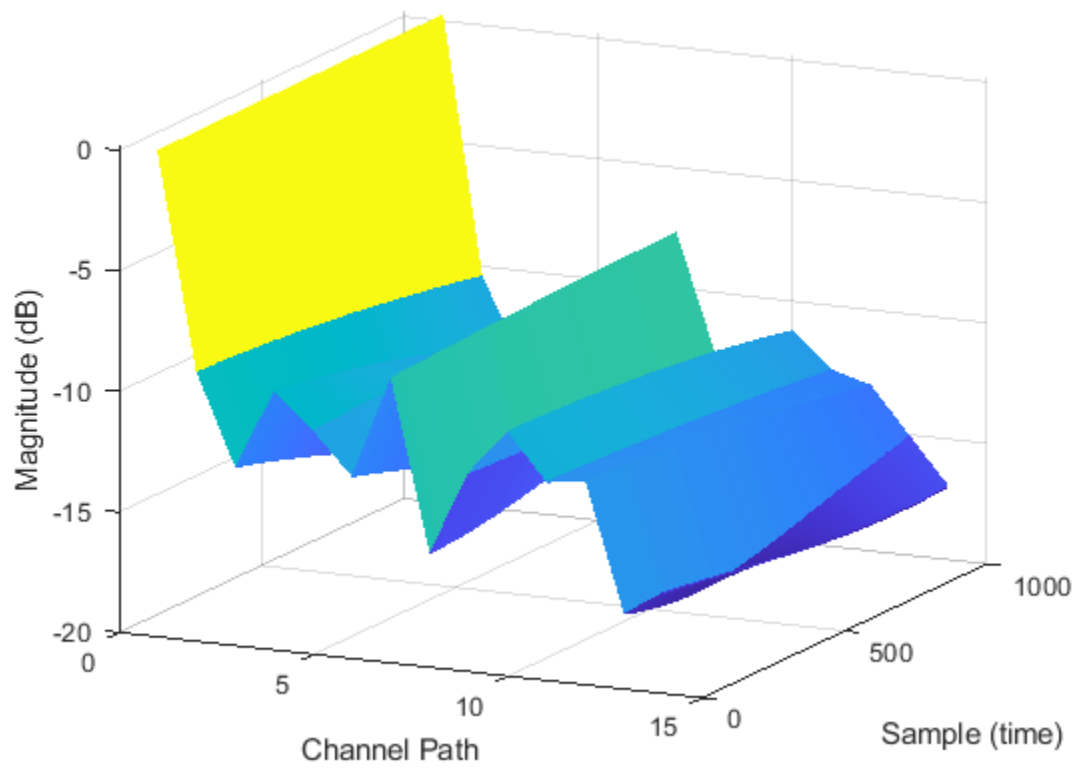
Create a dummy input signal. The length of the input determines the time samples of the generated path gain.

```
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)');
```





## See Also

### Objects

nrTDLChannel

## 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  $[M \ N \ P \ M_g \ N_g] = [2 \ 2 \ 2 \ 1 \ 1]$ , representing 1 panel ( $M_g=1$ ,  $N_g=1$ ) with a 2-by-2 antenna array ( $M=2$ ,  $N=2$ ) and  $P=2$  polarization angles. Configure the receive antenna array as  $[M \ N \ P \ M_g \ N_g] = [1 \ 1 \ 2 \ 1 \ 1]$ , 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);
```

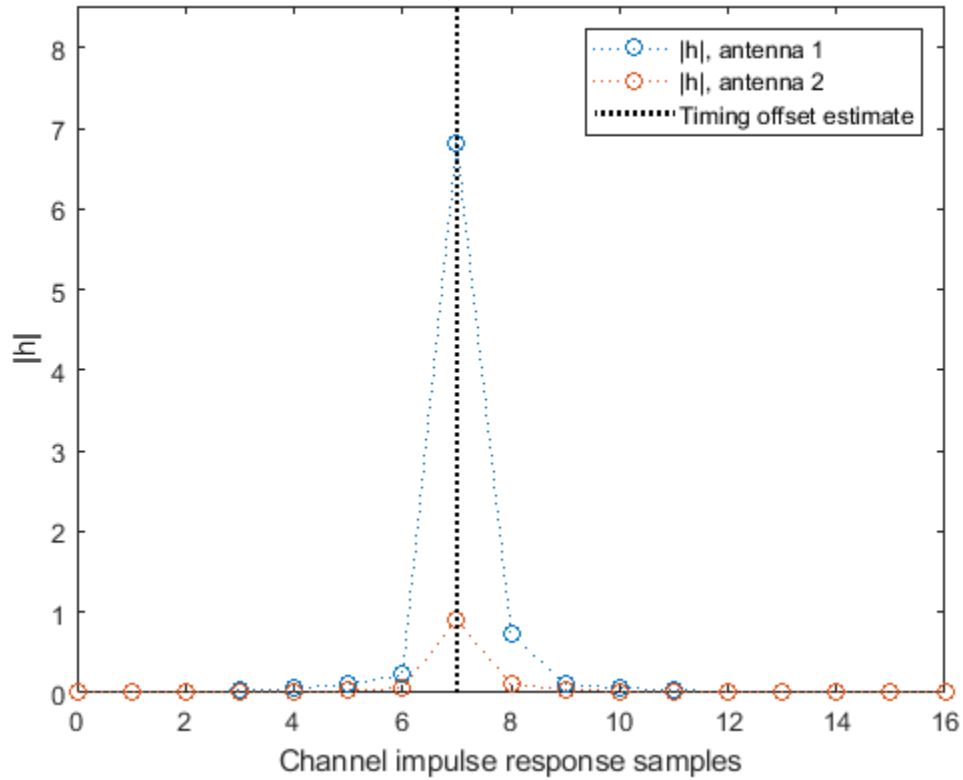
Perform timing offset estimation using `nrPerfectTimingEstimate`.

```
[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');
```



## See Also

### Functions

nrPerfectTimingEstimate

## 5G NR CQI Reporting

The example performs channel quality indicator (CQI) calculation, as defined in TS 38.214 Section 5.2.2, over a clustered delayed line (CDL) channel. The example evaluates the accuracy of the CQI reported by using the practical channel estimation.

### Introduction

CQI is an indicator of channel quality. The UE reports the CQI index to the gNB, as a part of channel state information (CSI) by using the CSI-RS for measurement.

The CQI index is a value between 0 to 15. It provides information about the highest modulation scheme and code rate suitable for the downlink transmission to achieve the required block error rate (BLER) condition. TS 38.214 Section 5.2.2.1 defines two conditions for maximum allowable BLER, which are obtained from the higher layer parameter 'cqi-Table'. This example considers 0.1 BLER condition and cqi-Table as 'table-1' (TS 38.214 Table 5.2.2.1-2).

The example performs CQI reporting for a single input single output (SISO) scenario over a CDL channel with delay profile CDL-C and delay spread 300e-9 sec. The CQI value obtained by using practical channel estimation is compared with CQI obtained by using perfect channel estimation.

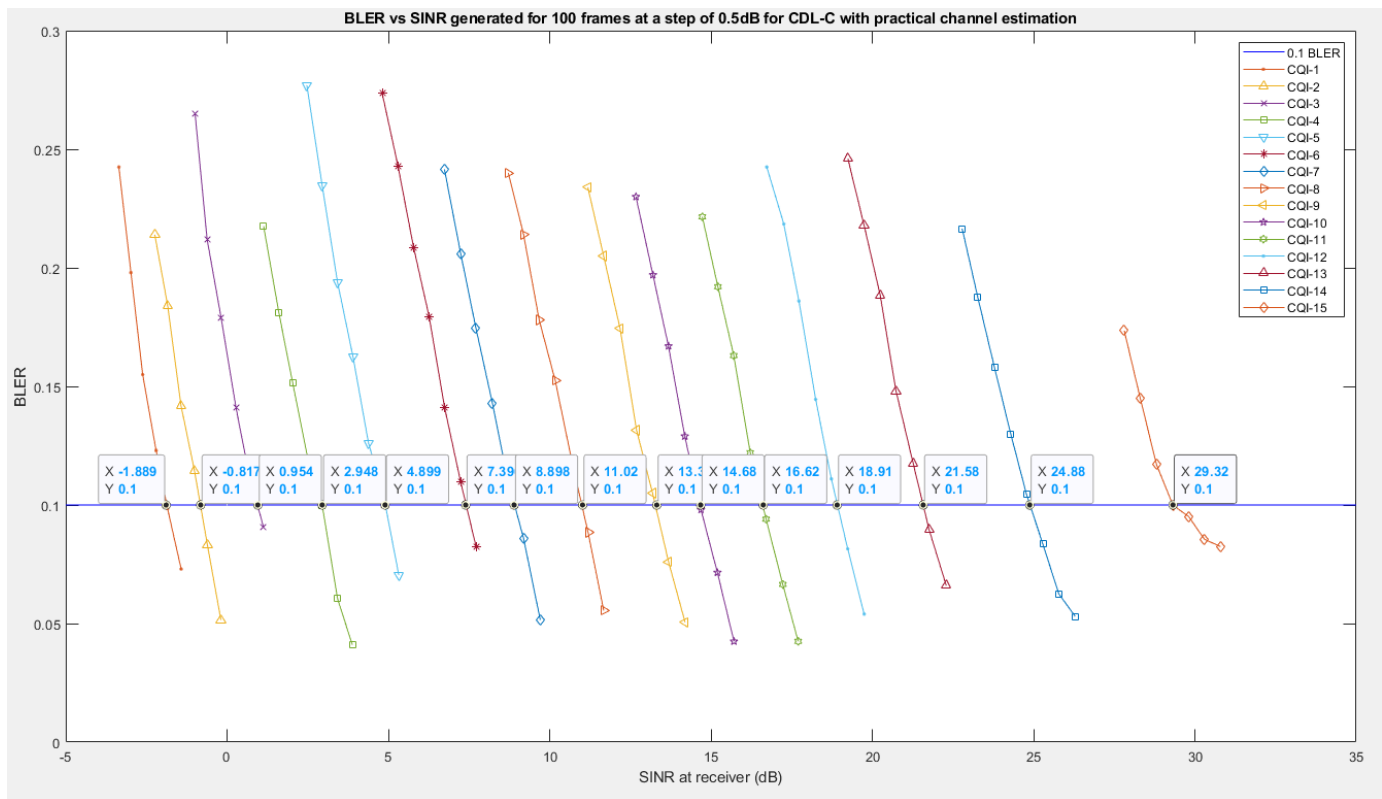
### CQI versus SINR table

Given the signal to interference and noise ratio (SINR), the CQI is obtained by using the pre-calculated lookup tables of CQI versus SINR. The lookup table is calculated to provide a CQI resulting in a maximum of 0.1 BLER for the corresponding SINR. It considers the number of transmission layers, the number of transmit and receive antennas, and the channel conditions.

When CQI reporting is performed, the lookup tables are taken as reference. The observed SINR is compared with the table and the highest CQI value for the SINR, where the BLER is less than 0.1, is reported.

The lookup tables in the example are created with the help of the "NR PDSCH Throughput" on page 1-56 example. Simulations for downlink shared channel throughput are performed considering the modulation scheme and code rate which corresponds to each CQI, across a finite range of signal to noise ratio (SNR) values. The SINR value observed at the receiver, where a 0.1 BLER is achieved, gets mapped against each CQI index. The lookup tables in the example are created for CDL-C channel by using both practical channel estimation and perfect channel estimation.

The following figure represents the BLER simulation results performed across a finite range of SNRs, for all CQI values. Each curve represents the BLER corresponding to the CQI value. The simulation is performed for 100 frames, with the step size of 0.5 dB and TS 38.214 Table 5.2.2.1-2 as cqi-table. The simulations consider the practical channel estimation method, with 'AveragingWindow' as [5 1], for a SISO scenario. 'AveragingWindow' specifies the number of adjacent reference symbols in the frequency domain and time domain, over which averaging must be performed in the channel estimation procedure. The SINR at the receiver, where 0.1 BLER is observed, is shown for each CQI value in the data tip.



### Initialize Configuration Objects

Create a carrier configuration object representing a 20MHz carrier with subcarrier spacing 30 kHz.

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 30;
carrier.NSizeGrid = 52;
```

Create a bandwidth part structure with the active bandwidth part (BWP) size and start fields. BWP start field is relative to common resource block 0 (CRB 0).

```
bwp.NSizeBWP = 41;
bwp.NStartBWP = carrier.NStartGrid; % Consider the start of BWP aligning with the start of the c
```

Create a CSI-RS configuration object with one non-zero-power NZP-CSI-RS resource occupying entire BWP.

```
csirs = nrCSIRSConfig;
csirs.RowNumber = 2; % Possible row numbers for single transmit antenna case are 1 and 2
csirs.NumRB = bwp.NSizeBWP;
csirs.RBOffset = bwp.NStartBWP - carrier.NStartGrid;
csirs.CSIRSPeriod = [4 0];
csirs.SymbolLocations = 0;
```

Configure the number of transmit and receive antennas, considering a SISO scenario.

```
nTxAnts = 1;
nRxAnts = 1;
```

### Generate CSI-RS Symbols and Indices for One Frame and Add to Grid

This section generates the CSI-RS symbols and indices for the specified carrier and CSI-RS configuration parameters.

Initialize an empty transmission grid.

```
txGrid = [];
```

Initialize carrier resource grid for one slot and perform resource element mapping.

```
ports = max(csirs.NumCSIRSPorts);           % Number of antenna ports
txSlotGrid = nrResourceGrid(carrier,ports); % Create the carrier resource grid for one slot

for slotIdx = 0:carrier.SlotsPerFrame - 1
    carrier.NSlot = slotIdx;

    csirsInd = nrCSIRSIndices(carrier,csirs);
    csirsSym = nrCSIRS(carrier,csirs);       % Generate CSI-RS symbols with slot periodicity

    % Placing the CSI-RS in the transmit grid
    txSlotGrid(csirsInd) = csirsSym;        % Mapping CSI-RS symbols to the current slot
    % txGrid for one frame is generated by concatenating slot by slot
    txGrid = [txGrid txSlotGrid];         %#ok<AGROW>
end
```

### Perform OFDM Modulation

Perform OFDM modulation to generate the time-domain waveform.

```
[txWaveform,OFDMInfo] = nrOFDMModulate(carrier,txGrid);
```

### Transmit Waveform Through Channel and Add AWGN to the Received Waveform

Consider a CDL channel with delay profile CDL-C and delay spread 300e-9 sec.

Create a channel object with specified configurations.

```
channel = nrCDLChannel;
channel.DelayProfile = 'CDL-C';
channel.DelaySpread = 300e-9;
% Turn the overall number of antennas into a specific antenna panel array geometry
[channel.TransmitAntennaArray.Size, channel.ReceiveAntennaArray.Size] = ...
    hArrayGeometry(nTxAnts,nRxAnts);
channel.SampleRate = OFDMInfo.SampleRate;

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

Append zeros at the end of the transmitted waveform to flush channel content. These zeros take into account any delay introduced in the channel. This is a mix of multipath delay and implementation delay. This value may change depending on the sampling rate, delay profile and delay spread. Append zeros at the end of the transmitted waveform to flush channel content. These zeros take into account any delay introduced in the channel. This is a mix of multipath delay and implementation delay. This value may change depending on the sampling rate, delay profile and delay spread.

```
txWaveform = [txWaveform; zeros(maxChDelay, size(txWaveform,2))];
```

Transmit the waveform through the channel to obtain the received time-domain waveform and path gains.

```
[rxWaveform,pathGains,sampleTimes] = channel(txWaveform);
```

Generate and add AWGN to the received waveform.

```
SNRdB = 15; % in dB
SNR = 10^(SNRdB/20); % Linear value
N0 = 1/(sqrt(2.0*nRxAnts*double(OFDMInfo.Nfft))*SNR); % Noise variance
rng('default');
noise = N0*complex(randn(size(rxWaveform)),randn(size(rxWaveform)));
rxWaveform = rxWaveform + noise;
```

### Perform CQI Reporting with Practical Channel Estimation

This section shows how CQI reporting with practical channel estimation is performed.

#### Perform Timing Estimate, Offset Correction, and OFDM Demodulation

```
% Generate reference grid for timing estimation
refGrid = txGrid; % Consider reference grid as txGrid since we only have CSI-RS in the txGrid
carrier.NSlot = 0; % To perform timing estimate, note the first NSlot value where the carrier
offsetPractical = 0;
[t,mag] = nrTimingEstimate(carrier,rxWaveform,refGrid);
offsetPractical = hSkipWeakTimingOffset(offsetPractical,t,mag);
rxWaveformPractical = rxWaveform(1+offsetPractical:end, :);

% Perform OFDM demodulation to the rxWaveform
rxGridPractical = nrOFDMDemodulate(carrier,rxWaveformPractical);
```

#### Configure Information Required to Calculate CQI

The information required for CQI reporting is the CQI reporting mode, the size of the subband, and the SINR lookup table for the particular channel model.

CQI reporting modes: The CQI reporting supports wideband or subband modes, as requested by the gNB or decided by the UE. In the case of wideband mode, one CQI value is reported for the entire BWP. In the case of subband mode, one CQI value for each subband is reported as per the subband size indicated by the higher layers. The size of subband, according to Table 5.2.1.4-2 in TS 38.214 depends on the size of BWP.

The following lookup table is computed for 90% throughput (which is equal to 0.1 BLER) over a CDL-C channel with a delay spread of 300e-9 sec by using the practical channel estimator.

```
cqiConfigPractical.CQIMode = 'Subband'; % Select one of two possible choices from 'Wideband' or 'Subband'
cqiConfigPractical.NSBPRB = 4; % The size of subband in resource blocks
cqiConfigPractical.SINR90pc = [-1.89 -0.82 0.95 2.95 4.90 7.39 8.89 ...
    11.02 13.32 14.68 16.62 18.91 21.58 24.88 29.32];
```

#### Perform CQI Calculation

The CQI values are calculated using an NZP-CSI-RS resource with slot periodicity 4 and offset 0 over one frame.

```
cqiPracticalPerSlot = [];
for slotIdx = 0:carrier.SlotsPerFrame - 1
    carrier.NSlot = slotIdx;
```

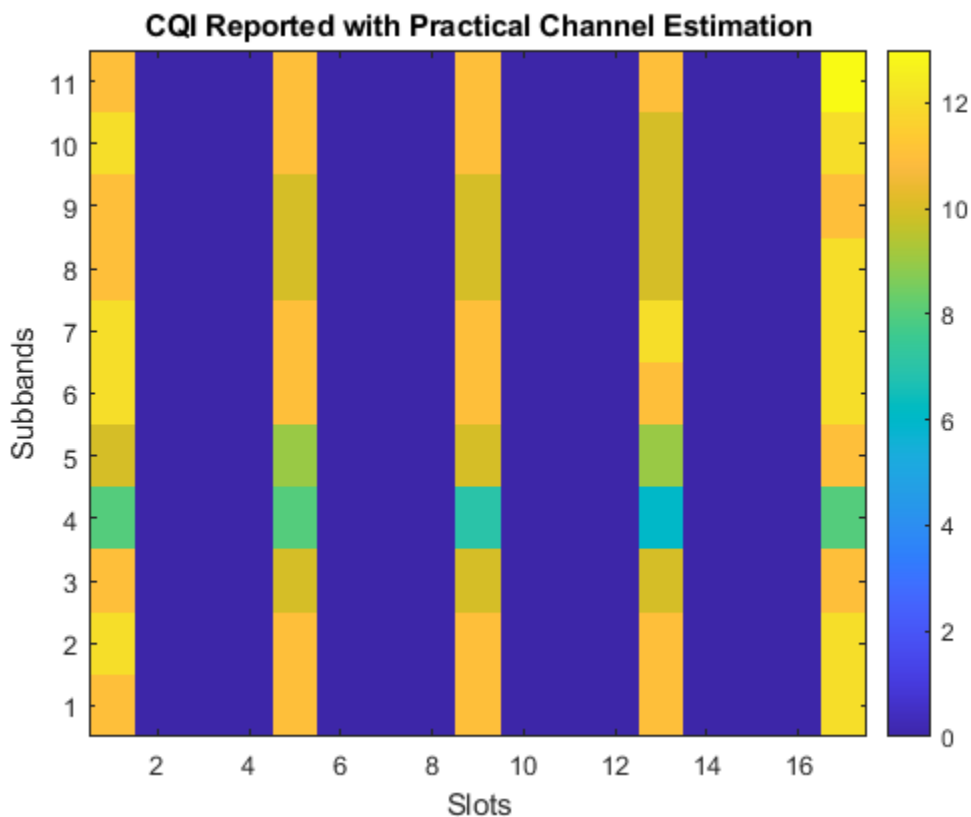
```

csirsSym = nrCSIRS(carrier,csirs);
csirsRefInd = nrCSIRSIndices(carrier,csirs);
if (~isempty(csirsRefInd))
    % Perform channel estimate by considering 'AveragingWindow' as [5 1]
    [Hest,nVar] = nrChannelEstimate(carrier,rxGridPractical(:,(1:carrier.SymbolsPerSlot)+(ca

    % CQI value reported for each slot is stored in a new column
    % In subband case, a column of CQI values is reported, where each element corresponds to
    cqiPracticalPerSlot(:,slotIdx+1) = hCQISelect(carrier,bwp,cqiConfigPractical,csirsRefInd
end
end

% Plot the practical CQI estimates for one frame
plotCQI(cqiPracticalPerSlot,'CQI Reported with Practical Channel Estimation');

```



### Perform CQI Reporting with Perfect Channel Estimation

This section shows how you perform CQI reporting with perfect channel estimation.

### Perform Perfect Timing Estimate, Offset Correction, and OFDM Demodulation

```

pathFilters = getPathFilters(channel); % Get path filters for per
[offsetPerfect,mag] = nrPerfectTimingEstimate(pathGains,pathFilters);
rxWaveformPerfect = rxWaveform(1+offsetPerfect:end, :);
rxGridPerfect = nrOFDMDemodulate(carrier,rxWaveformPerfect);

```



### Configure Required Information to Calculate CQI

The information required for CQI reporting is the CQI reporting mode, the size of the subband, and the SINR lookup table for the particular channel model. Consider the same CQI configuration used for practical channel estimation.

```
cqiConfigPerfect = cqiConfigPractical;
```

The following lookup table is computed for 90% throughput over a CDL-C channel with a delay spread of 300e-9 sec considering the perfect channel estimator.

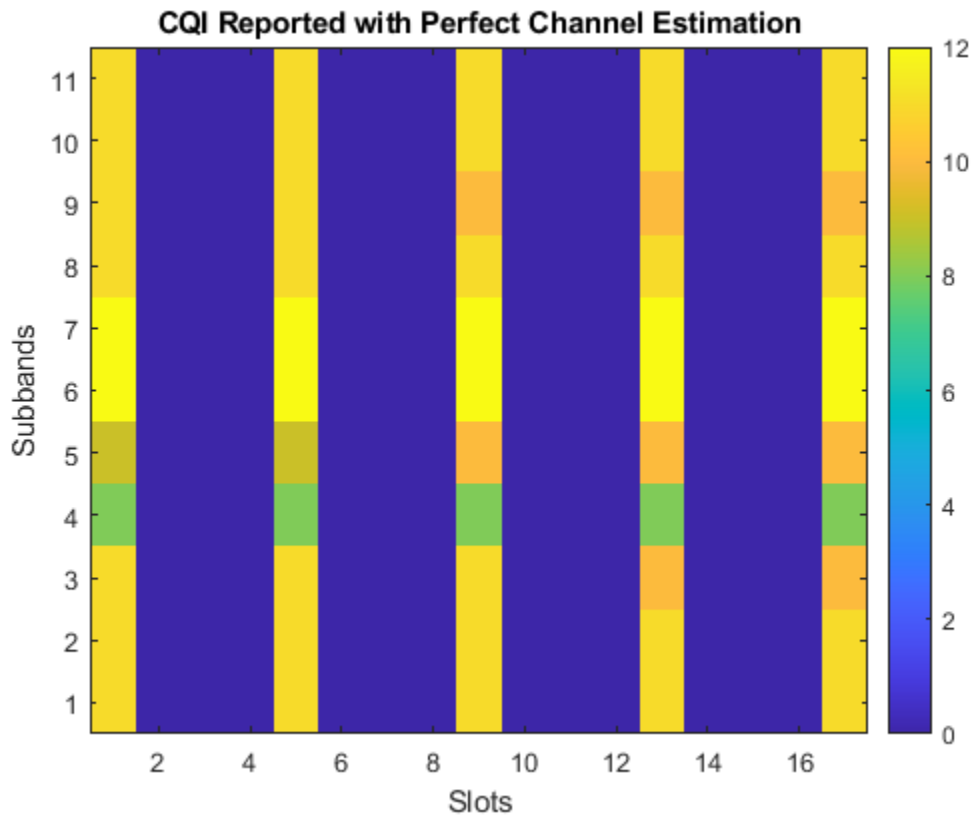
```
cqiConfigPerfect.SINR90pc = [-4.73, -3.04, -0.54, 1.87, 4.17, 6.77, 8.57, 10.57,...
    12.87, 14.27, 16.17, 18.37, 20.77, 23.67, 27.07];
```

### Perform CQI Calculation

The CQI values are calculated using an NZP-CSI-RS resource with slot periodicity 4 and offset 0 over one frame.

```
cqiPerfectPerSlot = [];
for slotIdx = 0:carrier.SlotsPerFrame - 1
    carrier.NSlot = slotIdx;
    csirsSym = nrCSIRS(carrier,csirs);
    csirsRefInd = nrCSIRSIndices(carrier,csirs);
    if (~isempty(csirsRefInd))
        PerfectHest = nrPerfectChannelEstimate(carrier,pathGains,pathFilters,offsetPerfect,sampL
        % Get perfect noise estimate (from the noise realization)
        noiseGrid = nrOFDMDemodulate(carrier,noise(1+offsetPerfect:end ,:));
        nVar = var(noiseGrid(:));
        % CQI value reported for each slot is stored in a new column
        % In subband case, a column of CQI values, one value corresponding to each subband is re
        cqiPerfectPerSlot(:,slotIdx+1) = hCQISelect(carrier,bwp,cqiConfigPerfect,csirsRefInd,Pe
    end
end

% Plot the perfect CQI estimates for one frame
plotCQI(cqiPerfectPerSlot,'CQI Reported with Perfect Channel Estimation');
```



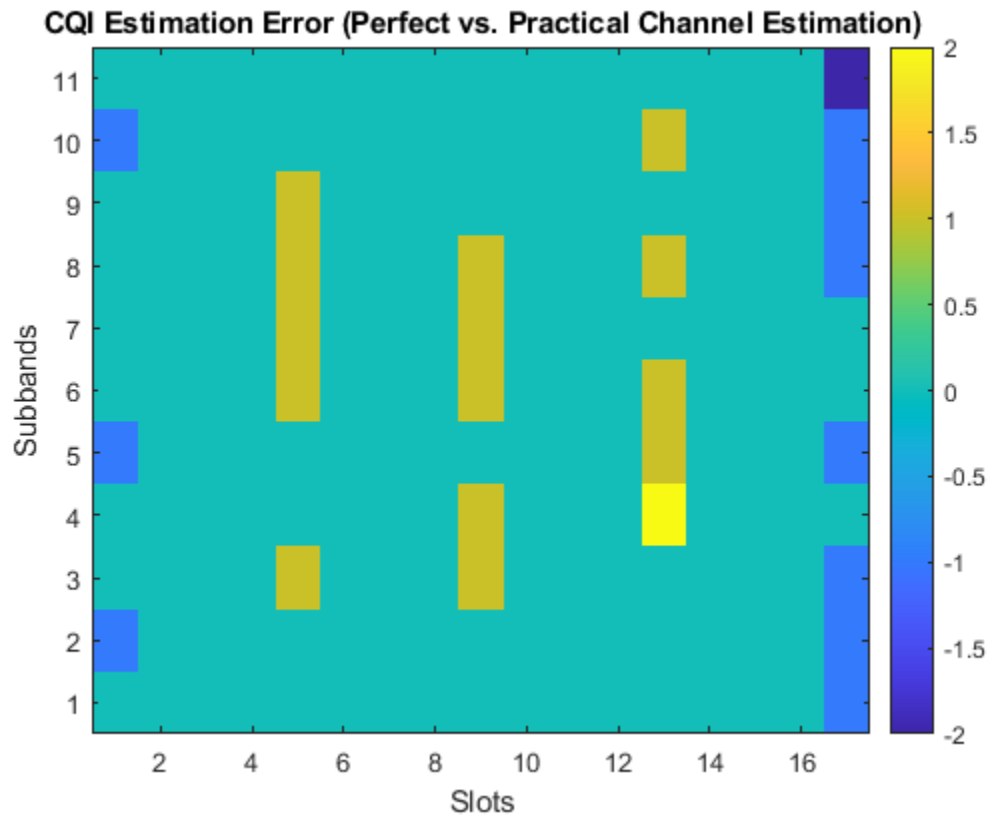
### Compare the CQI Values Reported in Practical and Perfect Channel Estimation Scenarios

Obtain the CQI reporting error due to practical channel estimation error:

```
cqiError = cqiPerfectPerSlot - cqiPracticalPerSlot;
```

```
% Plot the error
```

```
plotCQI(cqiError, 'CQI Estimation Error (Perfect vs. Practical Channel Estimation)');
```



### Local Functions

```
function plotCQI(cqi,Title)
%   plotCQI(CQI,TITLE) plots the CQI values with the title
%   TITLE.

    figure()
    imagesc(cqi)
    axis xy;
    colorbar;

    title(Title);
    xlabel('Slots');
    ylabel('Subbands');
end
```

### 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.*

## See Also

### Functions

nrCSIRS | nrCSIRSIndices | nrChannelEstimate | nrPerfectChannelEstimate | nrPerfectTimingEstimate | nrTimingEstimate

### Objects

nrCSIRSConfig | nrCarrierConfig

## Related Examples

- “5G NR CSI-RS Measurements” on page 1-88
- “NR Channel Estimation Using CSI-RS” on page 1-81

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

```
txSize = [4 3 2 1 2];
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  $[\alpha \beta \gamma]^T = [0 15 0]^T$ , which specifies 0 degrees bearing, 15 degrees downtilt, and 0 degrees slant.

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

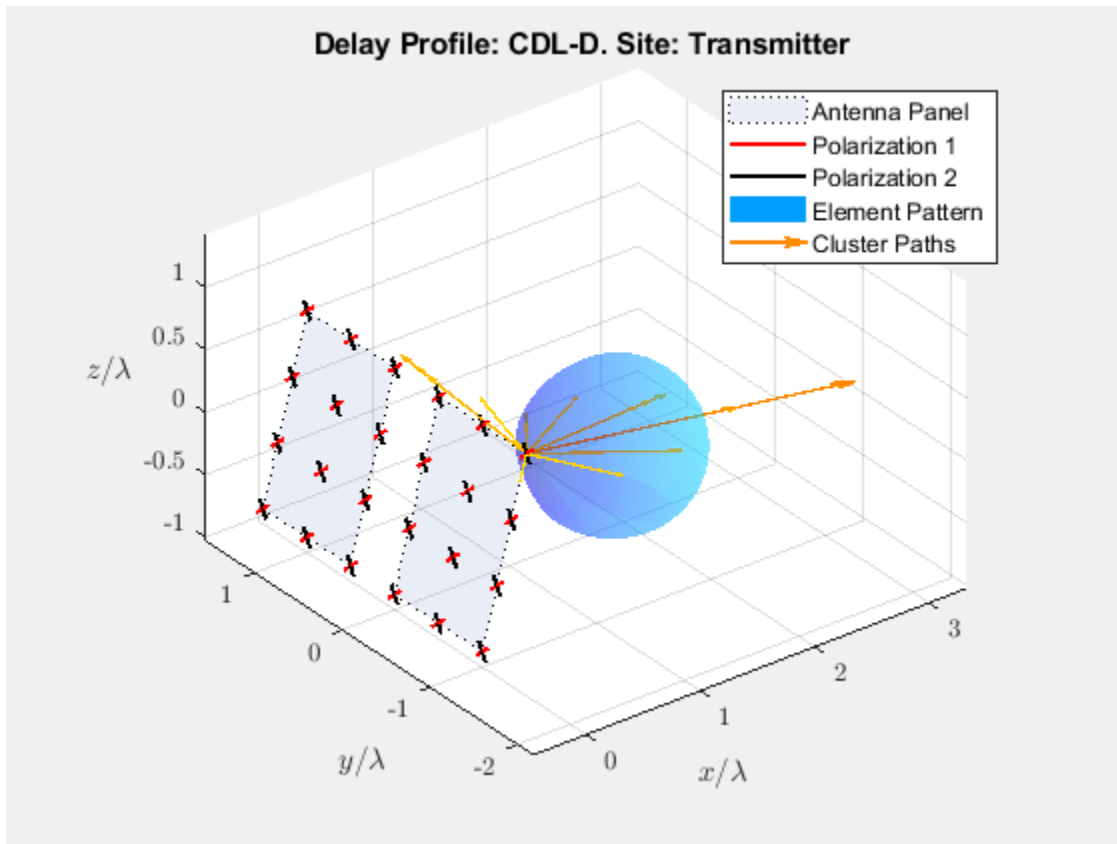
For an overview of all transmit antenna array properties, see the “TransmitAntennaArray” property of the `nrCDLChannel` System object.

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.

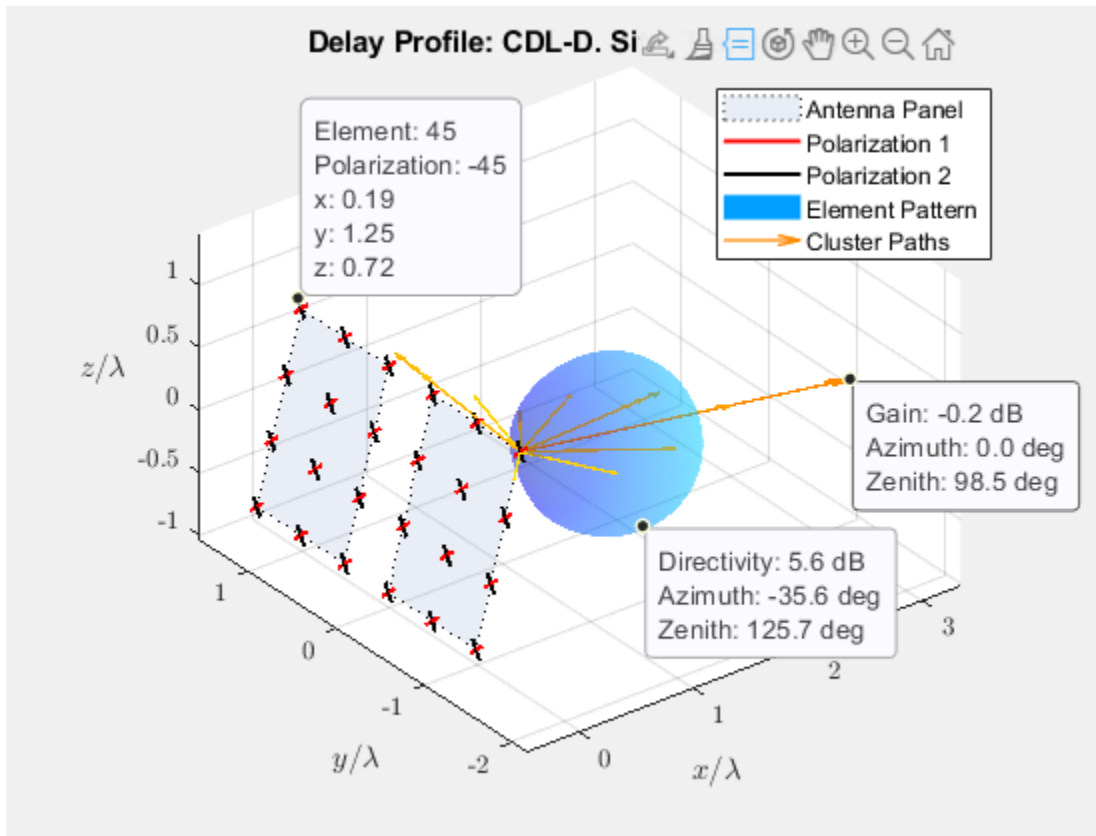
```
datacursormode on;
```



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.

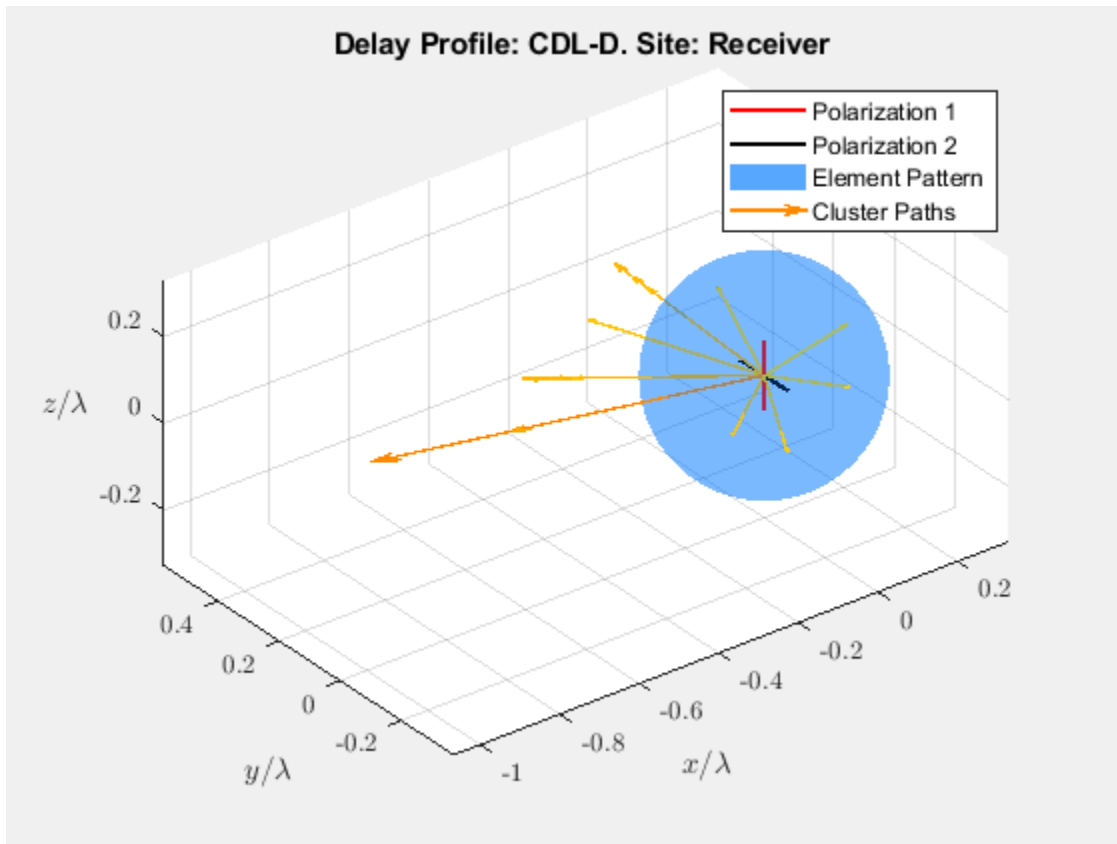


Visualize and explore channel characteristics at the receiver end. To customize the receive antenna array, use the "ReceiveAntennaArray" property of the `nrCDLChannel` System object. Then, display the channel characteristics at the receiver end by calling the `displayChannel` function with the 'LinkEnd', '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;
```



## See Also

### Functions

`displayChannel` | `info`

### Objects

`nrCDLChannel`

## More About

- "Interactively Explore Plotted Data"



# System-Level Simulation

---

## NR PUSCH FDD Scheduling

This example evaluates the throughput and resource share fairness performance of an uplink (UL) scheduling strategy in frequency division duplexing (FDD) mode with radio link control (RLC) layer integrated. The UL scheduling strategy assigns the physical uplink shared channel resources (PUSCH) resources to a set of user equipments (UEs) connected to a gNB. The example uses unacknowledged mode (UM) of RLC layer and a passthrough physical (PHY) layer. The passthrough PHY layer does not involve any physical layer processing and adopts a probability-based approach to model packet reception failures. The example logs the events in the simulation and also shows the run time visualizations to observe the network performance.

### Introduction

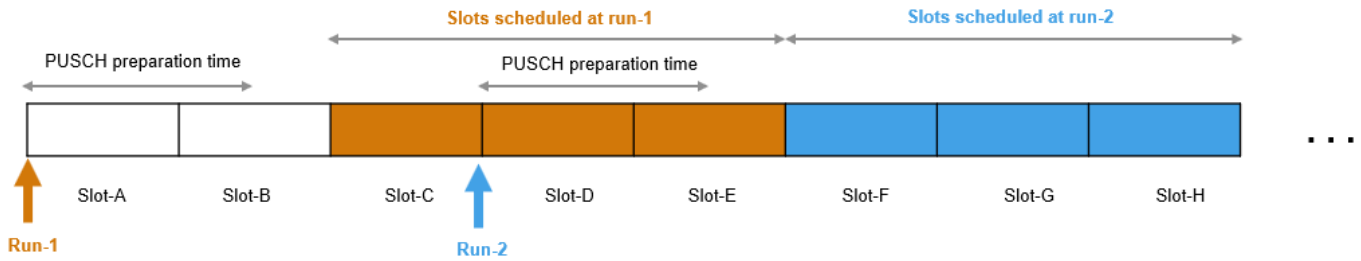
This example shows how a scheduling strategy (controlled by the gNB) assigns UL resources among the UEs. The example considers the following operations within the gNB and UEs that facilitate UL transmissions and receptions.

	Operations
gNB	<ul style="list-style-type: none"> <li>• Run the scheduling algorithm to assign uplink resources</li> <li>• Send the uplink assignments to the UEs</li> <li>• Receive the PUSCH transmission from the UEs</li> </ul>
UEs	<ul style="list-style-type: none"> <li>• Send the pending buffer status report to the gNB</li> <li>• Receive the uplink assignments from the gNB</li> <li>• Adhere to the received uplink assignments from the gNB for PUSCH transmission</li> </ul>

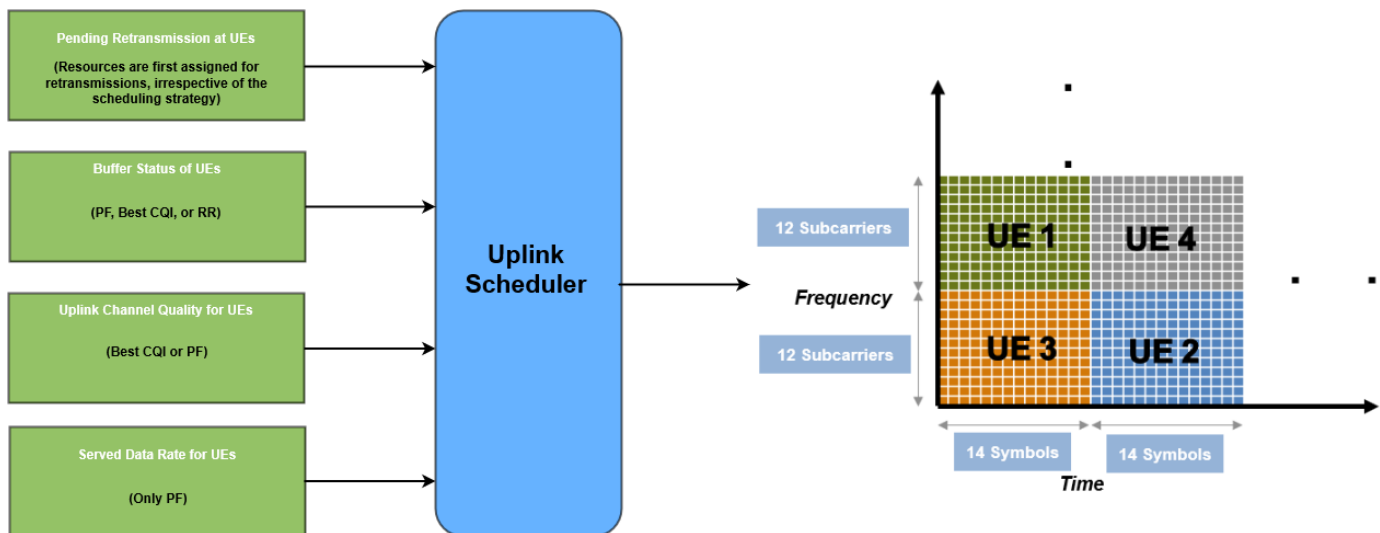
The complete PUSCH packet is transmitted in the first symbol of its allocated symbol set. Receiver processes the packet in the symbol just after the last symbol in the allocated symbol set.

The scheduler runs every  $p$  slots to assign the UL resources, where  $p$  is the configured periodicity of the scheduler. In each run, the number of slots scheduled is equal to the periodicity of scheduler run,  $p$ . The first slot, among the  $p$  slots to be scheduled in a run, is the next upcoming slot which satisfies the PUSCH preparation time capability of UEs. For example, the figure shows the way scheduler selects the slots during two consecutive runs. It assumes scheduler periodicity ( $p$ ) of 3 slots. Hence, the scheduler runs after every 3 slots, and schedules resources for 3 slots. PUSCH preparation time capability for all the UEs is assumed as greater than 1 slot (14 symbols) but less than 2 slots (28 symbols).

- Run-1: When scheduler runs at the start of Slot-A, it schedules 3 slots starting from Slot-C, because for Slot-A and Slot-B the UEs do not get enough PUSCH preparation time (UEs gets a time of 0 symbols at start of Slot-A and 14 symbols at start of Slot-B). For Slot-C, UEs get 28 symbols for PUSCH preparation and this satisfies the PUSCH preparation time capability. Hence, Slot-C, D, and E are scheduled in this run.
- Run-2: When scheduler runs at the start of Slot-D, it schedules the next 3 contiguous slots starting from Slot-F (Slot-F, G, and H).



You can choose any one of the implemented scheduling strategies: proportional fair (PF), best CQI, or round robin (RR). The various supported inputs to the UL scheduler are listed along with the scheduling strategies that consider them.



The two control packets, the buffer status report (BSR) and UL assignment, are assumed to be sent out of band without the need of resources for transmission.

Demodulation reference signal (DM-RS) is not modeled in this example. However, one symbol is kept unused for it in the PUSCH assignments.

This example models:

- Slot based UL scheduling. UL scheduler ensures that the UEs get the required PUSCH preparation time.
- Noncontiguous allocation of frequency-domain resources in terms of resource block groups (RBGs).
- Configurable subcarrier spacing resulting in different slot durations.
- Asynchronous UL hybrid automatic repeat request (HARQ) mechanism.
- UL reception success or failure detection by UEs using new data indicator (NDI) flag present in UL assignment.
- Multiple logical channels to support different application traffic patterns.
- Logical channel prioritization (LCP) at UE to distribute the received UL assignment among logical channels.

- Periodic UL application traffic pattern.
- RLC operating in UM mode.

### Scenario Configuration

For the simulation, set these key configuration parameters:

- Simulation time
- Number of UEs
- Distance of UEs from gNB (affects the UL CQI values for UEs)
- Application traffic pattern at UEs to generate traffic
- RLC configuration for Tx and Rx entities at UEs and gNB respectively
- Logical channel configuration of UEs and gNB
- Scheduling strategy: PF, Best CQI, RR
- Periodicity of BSRs sent by UEs to inform gNB about pending buffer amount
- PUSCH preparation time for UEs
- PUSCH bandwidth in terms of number of resource blocks (RBs)
- Subcarrier spacing
- RBG size configuration type
- Initial UL channel quality and its update mechanism

```
rng('default'); % Reset the random number generator
simParameters = []; % Clear the simulation parameters
simParameters.NumFramesSim = 200; % Simulation time in terms of number of 10 ms frames

% Number of UEs in the simulation. UEs are assumed to have sequential radio
% network temporary identifiers (RNTIs) from 1 to NumUEs. If you change the
% number of UEs, ensure that the simulation parameters
% simParameters.UEDistance and simParameters.PacketPeriodicityUEs are
% array of length equal to NumUEs; and simParameters.PacketSizesUEs,
% properties of simParameters.RLCConfig, and simParameters.LCHConfig
% consists of number of rows equal to numUEs.
simParameters.NumUEs = 4;
% Number of logical channels in each UE. If you change the number of
% logical channels, ensure that the simulation parameters
% simParameters.PacketSizesUEs, properties of simParameters.RLCConfig, and
% simParameters.LCHConfig consists of number of columns equal to
% NumLogicalChannels.
simParameters.NumLogicalChannels = 3;
simParameters.UEDistance = [100; 250; 700; 750]; % Distance of UEs from gNB (in meters)

% Set the application traffic pattern for UEs. For example, the vector
% element value 10 at index 4 represents that for each of the logical
% channels of UE-4, a packet is generated every 10 ms
simParameters.PacketPeriodicityUEs = [30; 20; 30; 10]; % Periodicity at which UEs generate packets
% The N-by-P matrix represents the size of packet generated by
% the UE in each logical channel, where 'N' represents the number of UEs
% and 'P' represents the number of logical channels. For example, the value
% 1000 at index (4, 1) represents a packet of size 1000 bytes generated by
% UE-4 for logical channel ID 1
simParameters.PacketSizesUEs = [4000 6000 8000;
    500 8000 8000;
```

```

12000 8000 4000;
1000 8000 4000];

% RLC configuration
% Each property of RLCConfig must be an N-by-P matrix, where 'N' represents the number of UEs
% and 'P' represents the number of logical channels. A matrix element at position
% (i, j) corresponds to property value of a UE with RNTI value 'i' and logical channel ID 'j'.
% Sequence number (SN) field length (in bits) to be used by each UE for their logical channels
simParameters.RLCConfig.SNFieldLength = [6 6 12;
6 6 12;
6 12 6;
6 6 12];
% Reassembly timer to be used by each UE for their logical channels
simParameters.RLCConfig.ReassemblyTimer = [5 10 15;
5 5 10;
5 5 5;
5 10 15];
% Max number of service data units (SDUs) in the Tx buffer of each logical channel (to model Tx
simParameters.RLCConfig.MaxTxBufferSDUs = [4 3 11;
3 5 4;
6 1 3;
11 6 26];

% Logical channel (LCH) configuration
% Each property of LCHConfig must be an N-by-P matrix, where 'N' represents the number of UEs
% and 'P' represents the number of logical channels. A matrix element at position
% (i, j) corresponds to property value of a UE with RNTI value 'i' and logical channel ID 'j'.
% Mapping between logical channel and logical channel group (LCG) ID
simParameters.LCHConfig.LCGID = [1 3 2;
1 2 2;
1 2 3;
5 1 2];
% Priority of each logical channel
simParameters.LCHConfig.Priority = [1 5 8;
1 1 6;
4 10 4;
10 11 13];
% Prioritized bit rate (PBR) of each logical channel (in kilo bytes per second)
simParameters.LCHConfig.PBR = [8 16 32;
8 128 32;
8 16 32;
8 16 32];
% Bucket size duration (BSD) of each logical channel (in ms)
simParameters.LCHConfig.BSD = [5 10 50;
5 20 20;
5 5 5;
5 10 20];

% Medium access control (MAC) configuration
% Set the scheduler run periodicity in terms of number of slots. Value must be
% less than the number of slots in a 10 ms frame
simParameters.SchedulerPeriodicity = 4;
simParameters.SchedulerStrategy = 'PF'; % Supported scheduling strategies: 'PF', 'RR' and 'BestC
% Moving average weight parameter within the range [0, 1] calculates
% average data rate for a UE. The value is used in the PF scheduling strategy.
% Parameter value closer to 1 implies more weight on the instantaneous
% data rate. Parameter value closer to 0 implies more weight on the past
% data rate

```

```

% AverageDataRate = ((1 - MovingAvgDataRateWeight) * PastDataRate) + (MovingAvgDataRateWeight * ...
simParameters.MovingAvgDataRateWeight = 0.5;
simParameters.BSRPeriodicity = 5; % In ms
simParameters.EnableHARQ = true; % Flag to enable or disable HARQ. If disabled, there are no retransmissions
simParameters.NumHARQ = 16; % Number of HARQ processes in each UE
% PUSCH preparation time. gNB ensures that PUSCH assignment is received at
% UEs PUSCHPrepTime ahead of the transmission time
simParameters.PUSCHPrepTime = 200; % In microseconds
% Maximum RBs allotted to a UE in a slot for a PUSCH transmission (limit is
% applicable for new PUSCH assignments and not for the retransmissions)
simParameters.RBAllocationLimitUL = 15;

% PHY layer and channel configuration
% RB count for 5 MHz band with 15 kHz subcarrier spacing (SCS). The complete
% UL bandwidth is assumed to be allotted for PUSCH
simParameters.NumRBs = 25;
simParameters.SCS = 15; % kHz
simParameters.ULBandwidth = 5e6; % Hz
simParameters.ULCarrierFreq = 2.515e9; % Hz
% Set the RBG size configuration to 1 (configuration-1 RBG table) or 2
% (configuration-2 RBG table) as defined in 3GPP TS 38.214 Section
% 5.1.2.2.1
simParameters.RBGSizeConfig = 1;

% Configure parameters to update channel conditions for the UEs. Channel
% quality is periodically improved or deteriorated by CQIDelta every
% channelUpdatePeriodicity seconds for all RBs of a UE. Whether channel
% conditions for a particular UE improve or deteriorate is randomly
% determined: RBCQI = RBCQI +/- CQIDelta
simParameters.ChannelUpdatePeriodicity = 0.5; % In sec
simParameters.CQIDelta = 1;
% Mapping between distance from gNB (first column in meters) and maximum
% achievable UL CQI value (second column). For example, if a UE is 700
% meters away from the gNB, it can achieve a maximum CQI value of 10 as the
% distance falls within the [501, 800] meters range, as per the mapping. Set
% the distance in increasing order and the achievable CQI value in decreasing
% order
simParameters.CQIvsDistance = [
    200 15;
    500 12;
    800 10;
    1000 8;
    1200 7];

% Logging and visualization configuration
% Flag to enable or disable run time CQI visualization
simParameters.CQIVisualization = true;
% Flag to enable or disable run time visualization of RB assignment. If enabled,
% then for slot based scheduling it updates every frame (10 ms) to show RB
% allocation to the UEs for different slots of the last frame.
simParameters.RBVisualization = true;

% The output metrics plots are updated NumMetricsSteps times during
% simulation
simParameters.NumMetricsSteps = 20;

% MAT-files used for post simulation visualization
simParameters.ParametersLogFile = 'simParameters'; % For logging the simulation parameters

```

```
simParameters.SimulationLogFile = 'simulationLogs'; % For logging the simulation logs
hNRULSchedulingValidateConfig(simParameters); % Validate the simulation configuration
```

### Derived Parameters

Based on the primary configuration parameters, compute the derived parameters.

```
simParameters.NCellID = 1; % Physical cell ID
simParameters.DLCarrierFreq = 2.635e9; % Hz
simParameters.DLBandwidth = 10e6; % Hz
simParameters.GNBPosition = [0 0 0]; % Position of gNB in (x,y,z) coordinates
% Slot duration for the selected SCS and number of slots in a 10 ms frame
slotDuration = 1/(simParameters.SCS/15); % Slot duration in ms
numSlotsFrame = 10/slotDuration; % Number of slots in 10 ms frame
numSlotsSim = simParameters.NumFramesSim * numSlotsFrame; % Number of slots in the simulation

% Packet periodicities of UEs in terms of number of slots
appPeriodicityUEsSlots = simParameters.PacketPeriodicityUEs ./ slotDuration;

% Maximum RLC SDU length (in bytes)
simParameters.maxRLCSDULength = 9000;

% RLC entity direction. Value 0 represents downlink only, 1
% represents UL only and 2 represents both UL and downlink
% directions. Setting entity direction to have only UL
simParameters.RLCConfig.EntityDir = ones(simParameters.NumUEs, simParameters.NumLogicalChannels)

% Logical channel id (logical channel ID of data radio bearers starts from 4)
simParameters.LCHConfig.LCID = ones(simParameters.NumUEs, simParameters.NumLogicalChannels) .* (

% Construct information for RLC logger
lchInfo = repmat(struct('LCID',[],'EntityDir',[]), [simParameters.NumUEs 1]);
for idx = 1:simParameters.NumUEs
    lchInfo(idx).LCID = simParameters.LCHConfig.LCID(idx, :);
    lchInfo(idx).EntityDir = simParameters.RLCConfig.EntityDir(idx, :);
end

% Find maximum achievable CQI value for UEs based on their distance from
% the gNB
maxUECQIs = zeros(simParameters.NumUEs, 1); % To store the maximum achievable CQI value for UEs
for ueIdx = 1:simParameters.NumUEs
    % Based on the distance of the UE from gNB, find matching row in
    % CQIvsDistance mapping
    matchingRowIdx = find(simParameters.CQIvsDistance(:, 1) > simParameters.UEDistance(ueIdx));
    if isempty(matchingRowIdx)
        maxUECQIs(ueIdx) = simParameters.CQIvsDistance(end, 2);
    else
        maxUECQIs(ueIdx) = simParameters.CQIvsDistance(matchingRowIdx(1), 2);
    end
end

% Define initial UL channel quality as an N-by-P matrix,
% where 'N' is the number of UEs and 'P' is the number of RBs in the carrier
% bandwidth. The initial value of CQI for each RB, for each UE, is given
% randomly and is limited by the maximum achievable CQI value corresponding
% to the distance of the UE from gNB
simParameters.InitialChannelQualityUL = zeros(simParameters.NumUEs, simParameters.NumRBs); % To s
```

```

for ueIdx = 1:simParameters.NumUEs
    % Assign random CQI values for the RBs, limited by the maximum achievable CQI value
    simParameters.InitialChannelQualityUL(ueIdx, :) = randi([1 maxUECQIs(ueIdx)], 1, simParameters.NumRBs);
end

% Interval at which metrics visualization updates in terms of number of
% slots. Make sure that MetricsStepSize is an integer
simParameters.MetricsStepSize = ceil(numSlotsSim / simParameters.NumMetricsSteps);
if mod(numSlotsSim, simParameters.NumMetricsSteps) ~= 0
    % Update the NumMetricsSteps parameter if numSlotsSim is not
    % completely divisible by it
    simParameters.NumMetricsSteps = floor(numSlotsSim / simParameters.MetricsStepSize);
end

```

### gNB and UEs Setup

Create the gNB and UE objects, initialize the UL channel condition information for UEs at gNB, and set up the logical channels at gNB and UE. The helper classes `hNRGNB.m` and `hNRUE.m` create gNB and UE nodes respectively, containing the RLC and MAC layer. For MAC layer, `hNRGNB.m` uses the helper class `hNRGNBMAC.m` to implement the gNB MAC functionality and `hNRUE.m` uses `hNRUEMAC.m` to implement the UE MAC functionality. Schedulers are implemented in `hNRSchedulerRoundRobin.m` (Round robin), `hNRSchedulerProportionalFair.m` (Proportional fair), `hNRSchedulerBestCQI.m` (Best CQI). All the schedulers inherit from the base class `hNRScheduler.m` which contains the core scheduling functionality. For RLC layer, both `hNRGNB.m` and `hNRUE.m` use `hNRUMEntity.m` to implement the functionality of the RLC transmitter and receiver. Passthrough PHY layer for UE and gNB is implemented in `hNRUEPassThroughPhy.m` and `hNRGNBPassThroughPhy.m`, respectively.

```

simParameters.Position = simParameters.GNBPosition;
gNB = hNRGNB(simParameters); % Create gNB node
% Create and add scheduler
switch(simParameters.SchedulerStrategy)
    case 'RR' % Round robin scheduler
        scheduler = hNRSchedulerRoundRobin(simParameters);
    case 'PF' % Proportional fair scheduler
        scheduler = hNRSchedulerProportionalFair(simParameters);
    case 'BestCQI' % Best CQI scheduler
        scheduler = hNRSchedulerBestCQI(simParameters);
end
addScheduler(gNB, scheduler); % Add scheduler to gNB

gNB.PhyEntity = hNRGNBPassThroughPhy(simParameters); % Create passthrough PHY
configurePhy(gNB, simParameters);
setPhyInterface(gNB); % Set the interface to PHY layer

% Create the set of UE nodes
UEs = cell(simParameters.NumUEs, 1);
for ueIdx = 1:simParameters.NumUEs
    simParameters.Position = [simParameters.UEDistance(ueIdx) 0 0]; % Position of UE
    UEs{ueIdx} = hNRUE(simParameters, ueIdx);
    simParameters.InitialChannelQualityDL = simParameters.InitialChannelQualityUL;
    UEs{ueIdx}.PhyEntity = hNRUEPassThroughPhy(simParameters, ueIdx); % Add passthrough PHY
    configurePhy(UEs{ueIdx}, simParameters);
    setPhyInterface(UEs{ueIdx}); % Set the interface to PHY layer

    % Initialize the UL CQI values at gNB
    updateChannelQuality(gNB, simParameters.InitialChannelQualityUL(ueIdx, :), 1, ueIdx); % 1 for

```



```

% Setup logical channels
for lcIdx = 1:simParameters.NumLogicalChannels
    % Create RLC channel configuration structure
    rlcChannelConfigStruct.EntityType = simParameters.RLCCConfig.EntityDir(ueIdx, lcIdx);
    rlcChannelConfigStruct.LogicalChannelID = simParameters.LCHConfig.LCID(ueIdx, lcIdx);
    rlcChannelConfigStruct.SeqNumFieldLength = simParameters.RLCCConfig.SNFieldLength(ueIdx,
    rlcChannelConfigStruct.MaxTxBufferSDUs = simParameters.RLCCConfig.MaxTxBufferSDUs(ueIdx,
    rlcChannelConfigStruct.ReassemblyTimer = simParameters.RLCCConfig.ReassemblyTimer(ueIdx,
    rlcChannelConfigStruct.EntityType = simParameters.RLCCConfig.EntityDir(ueIdx, lcIdx);
    rlcChannelConfigStruct.LCGID = simParameters.LCHConfig.LCGID(ueIdx, lcIdx);
    rlcChannelConfigStruct.Priority = simParameters.LCHConfig.Priority(ueIdx, lcIdx);
    rlcChannelConfigStruct.PBR = simParameters.LCHConfig.PBR(ueIdx, lcIdx);
    rlcChannelConfigStruct.BSD = simParameters.LCHConfig.BSD(ueIdx, lcIdx);

    % Setup logical channel at gNB for the UE
    configureLogicalChannel(gNB, ueIdx, rlcChannelConfigStruct);
    % Setup logical channel at UE
    configureLogicalChannel(UEs{ueIdx}, ueIdx, rlcChannelConfigStruct);

    % Add data traffic pattern generators to UE nodes. Application data
    % is pumped to RLC layer as per the installed traffic pattern
    packetSize = simParameters.PacketSizesUEs(ueIdx, lcIdx);
    % Calculate the data rate (in kbps) of On-Off traffic pattern using
    % packet size (in bytes) and packet interval (in ms)
    dataRate = ceil(1000/simParameters.PacketPeriodicityUEs(ueIdx)) * packetSize * 8e-3;
    % Limit the size of the generated application packet to the maximum
    % RLC SDU size. The maximum supported RLC SDU size is 9000 bytes
    if packetSize > simParameters.maxRLCSDULength
        packetSize = simParameters.maxRLCSDULength;
    end
    % Create an object for On-Off network traffic pattern and add it to the
    % specified UE. This object generates the uplink (UL) data traffic on the UE
    app = networkTrafficOnOff('PacketSize', packetSize, 'GeneratePacket', true, ...
        'OnTime', simParameters.NumFramesSim/100, 'OffTime', 0, 'DataRate', dataRate);
    UEs{ueIdx}.addApplication(ueIdx, simParameters.LCHConfig.LCID(ueIdx, lcIdx), app);
end
end

% Setup the UL and DL packet distribution mechanism
simParameters.MaxReceivers = simParameters.NumUEs;
% Create DL packet distribution object
dlPacketDistributionObj = hNRPacketDistribution(simParameters, 0);
% Create UL packet distribution object
ulPacketDistributionObj = hNRPacketDistribution(simParameters, 1);
hNRSetupPacketDistribution(simParameters, gNB, UEs, dlPacketDistributionObj, ulPacketDistributionObj);

```

## Processing Loop

Simulation is run slot by slot. In each slot, these operations are executed:

- Run the MAC and PHY layers of gNB
- Run the MAC and PHY layers of UEs
- Layer specific logging and visualization
- Advance the timer for the nodes. Every 1 ms it also sends trigger to application and RLC layers. Application layer and RLC layer execute their scheduled operations based on 1 ms timer trigger.

```

% To store the following UE metrics for each slot: throughput bytes
% transmitted, goodput bytes transmitted, and pending buffer amount bytes.
% The number of goodput bytes is calculated by excluding the retransmissions from the total
% transmissions
UESlotMetrics = zeros(simParameters.NumUEs, 3);

% To store the RLC statistics for each slot
ueRLCStats = cell(simParameters.NumUEs, 1);
gNBRLCStats = cell(simParameters.NumUEs, 1);

% To store current UL CQI values on the RBs for different UEs
uplinkChannelQuality = zeros(simParameters.NumUEs, simParameters.NumRBs);

% To store last received NDI values for UL HARQ processes
HARQProcessStatus = zeros(simParameters.NumUEs, simParameters.NumHARQ);

% Create an object for MAC scheduling information visualization and logging (UL direction is represented as 1)
simSchedulingLogger = hNRSchedulingLogger(simParameters, 1);

% Create an object for RLC visualization and logging (UL direction is represented as 1)
simRLCLogger = hNRRLCLogger(simParameters, lchInfo, 1);

symbolNum = 0;
% Run processing loop
for slotNum = 1:numSlotsSim

    % Run MAC and PHY layers of gNB
    run(gNB.MACEntity);
    run(gNB.PhyEntity);

    % Run MAC and PHY layers of UEs
    for ueIdx = 1:simParameters.NumUEs
        % Read the last received NDI flags for HARQ processes for
        % logging (Reading it before it gets overwritten by run function of MAC)
        HARQProcessStatus(ueIdx, :) = getLastNDIFlagHarq(UEs{ueIdx}.MACEntity, 1); % 1 for UL
        run(UEs{ueIdx}.MACEntity);
        run(UEs{ueIdx}.PhyEntity);
    end

    % RLC logging
    for ueIdx = 1:simParameters.NumUEs % For all UEs
        % Get RLC statistics
        ueRLCStats{ueIdx} = getRLCStatistics(UEs{ueIdx}, ueIdx);
        gNBRLCStats{ueIdx} = getRLCStatistics(gNB, ueIdx);
    end
    logRLCStats(simRLCLogger, ueRLCStats, gNBRLCStats); % Update RLC statistics logs

    % MAC logging
    % Read UL assignments done by gNB MAC scheduler at current time.
    % Resource assignments returned by a scheduler is empty, if UL
    % scheduler was not scheduled to run at the current time or no
    % resources got scheduled
    resourceAssignmentsUL = getCurrentSchedulingAssignments(gNB.MACEntity);
    for ueIdx = 1:simParameters.NumUEs
        % Read the UL channel quality at gNB for each of the UEs for logging
        uplinkChannelQuality(ueIdx, :) = getChannelQuality(gNB, 1, ueIdx); % 1 for UL
        % Read throughput and goodput bytes transmitted for this UE in the
        % current TTI for logging
    end
end

```

```

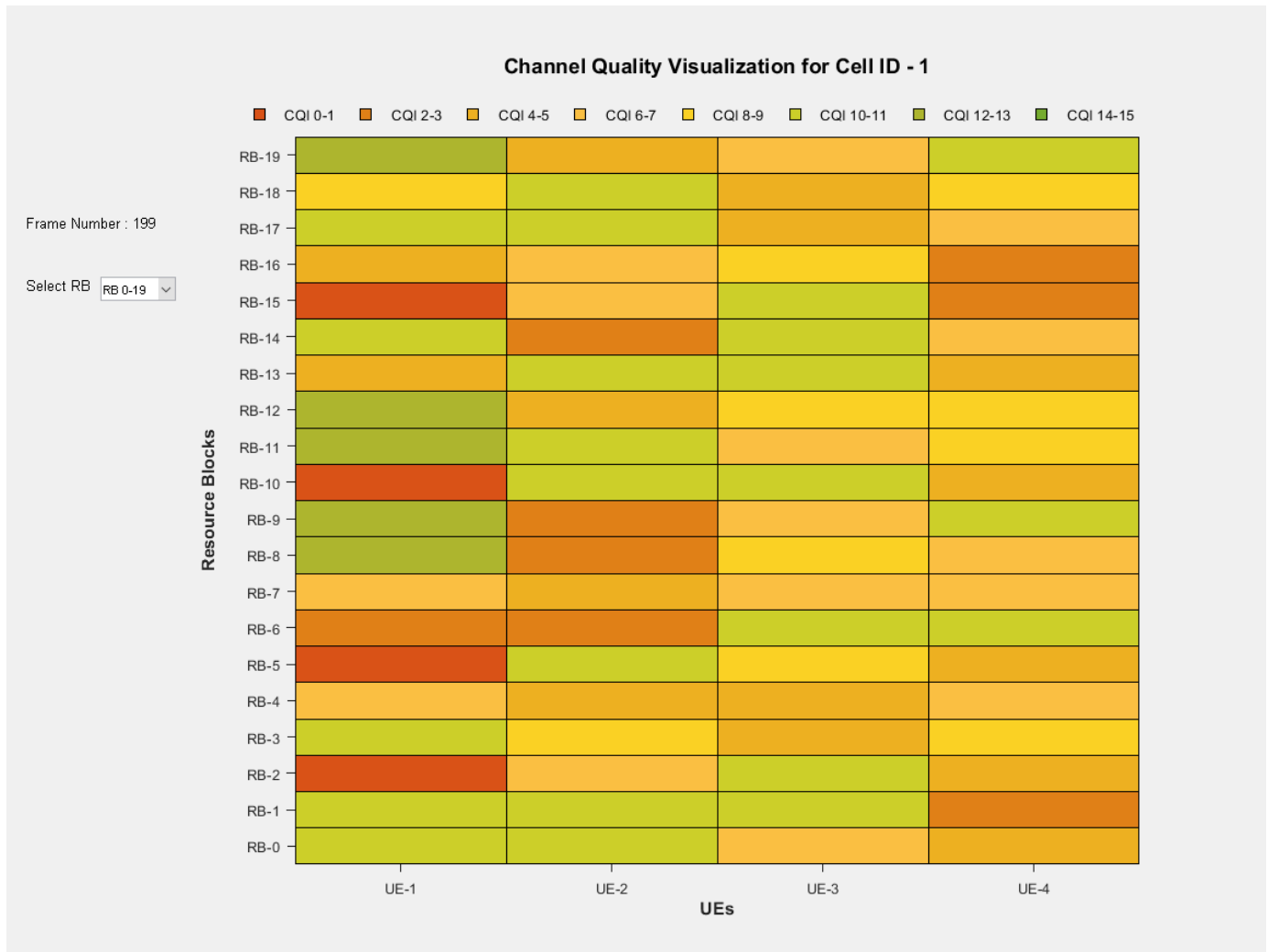
        [UESlotMetrics(ueIdx, 1), UESlotMetrics(ueIdx, 2)] = getTTIBytes(UEs{ueIdx});
        UESlotMetrics(ueIdx, 3) = getBufferStatus(UEs{ueIdx}); % Read pending buffer (in bytes)
    end
    % Update scheduling logs based on the current slot run of UEs and gNB.
    % Logs are updated in each slot, RB grid visualizations are updated
    % every frame, and metrics plots are updated every metricsStepSize
    % slots
    logScheduling(simSchedulingLogger, symbolNum + 1, resourceAssignmentsUL, UESlotMetrics, uplin

% Visualization
% If RB Visualization flag is set, update the RB assignment grid at the
% last slot of each frame
if simParameters.RBVisualization && mod(slotNum, numSlotsFrame) == 0
    plotRBGrids(simSchedulingLogger);
end
% If CQI Visualization flag is set, update the CQI grid at the last
% slot of each frame
if simParameters.CQIVisualization && mod(slotNum, numSlotsFrame) == 0
    plotCQIRBGrids(simSchedulingLogger);
end
% Plot MAC scheduling performance, RLC throughput metrics at every metricsStepSize slots
if mod(slotNum, simParameters.MetricsStepSize) == 0
    plotMetrics(simSchedulingLogger);
    plotMetrics(simRLCLogger);
end

tickGranularity = 14; % Number of symbols in a slot
% Advance timer ticks for gNB and UEs by 'tickGranularity' symbols
advanceTimer(gNB, tickGranularity);
for ueIdx = 1:simParameters.NumUEs % For all UEs
    advanceTimer(UEs{ueIdx}, tickGranularity);
end

% Symbol number in the simulation
symbolNum = symbolNum + tickGranularity;
end

```



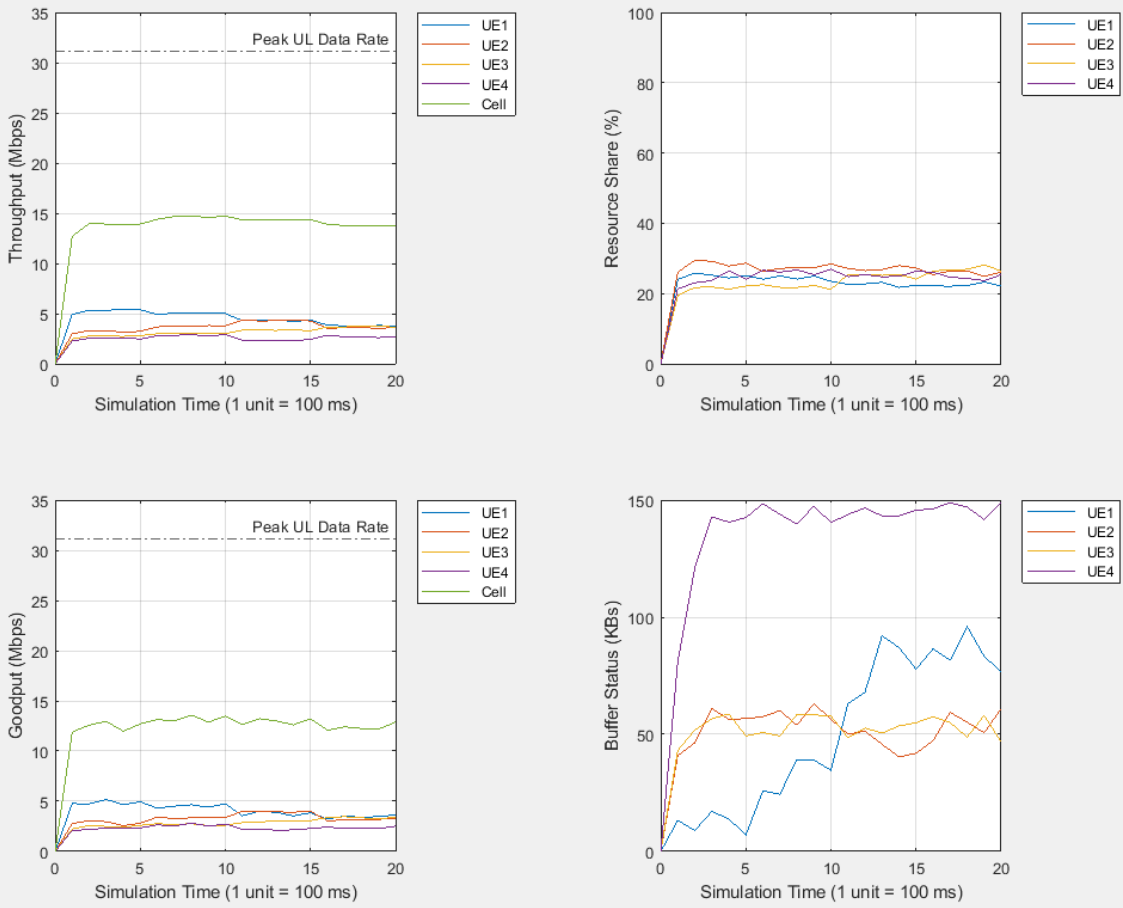
Resource Grid Allocation for Cell ID - 1

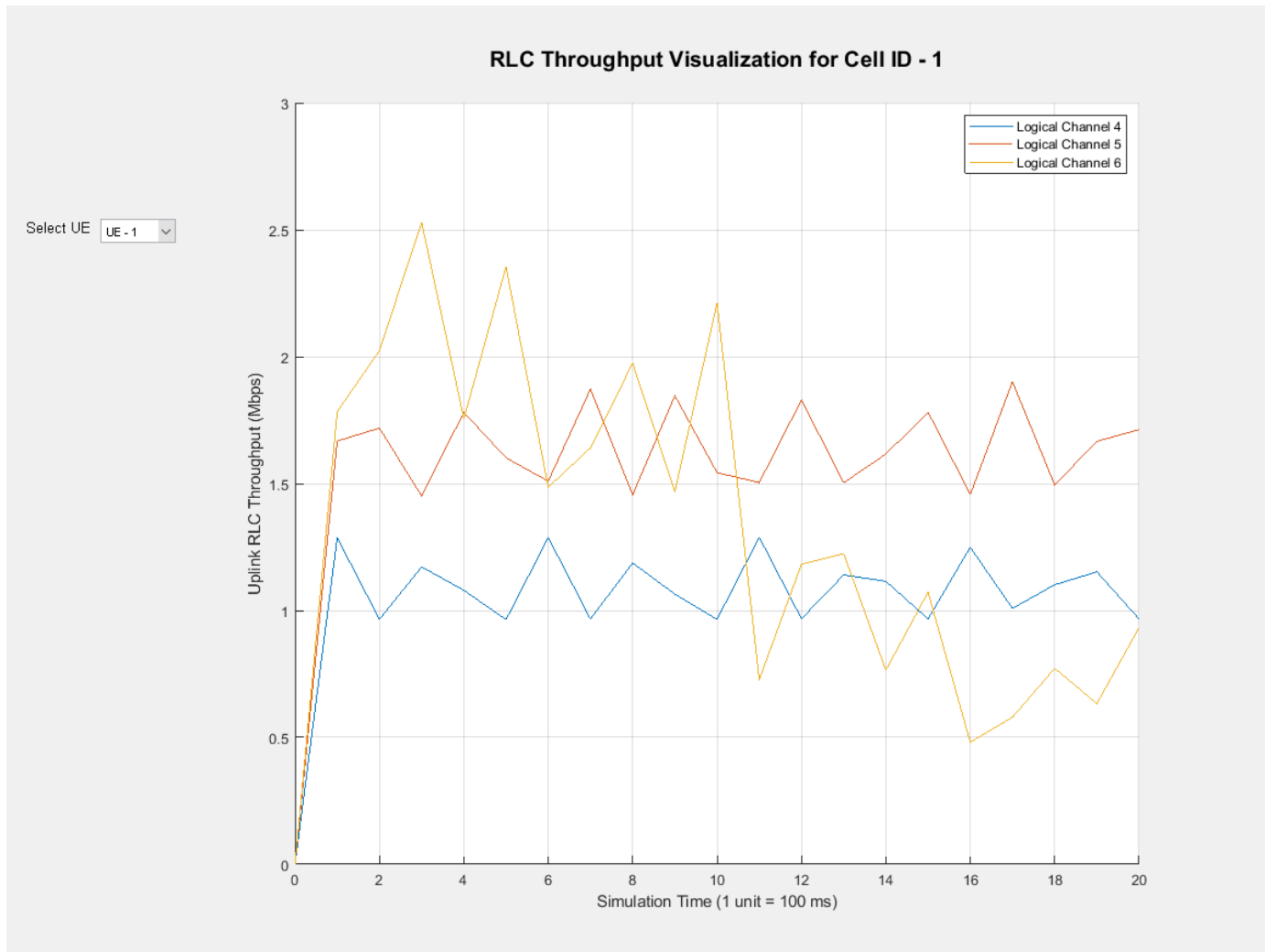
UE-x(n) : Transmission  
 UE-x(n) : Retransmission  
 x : UE RNTI  
 n : HARQ Process ID  
 Frame Number : 199  
 Select RB

Resource Blocks	RB-19	UE-4 (1)	UE-4 (3)	UE-4 (4)	UE-4 (0)	UE-2 (0)	UE-1 (3)	UE-2 (1)	UE-1 (0)	UE-1 (1)	UE-4 (2)
	RB-18	UE-4 (1)	UE-4 (3)	UE-4 (4)	UE-4 (0)	UE-2 (0)	UE-1 (3)	UE-2 (1)	UE-1 (0)	UE-1 (1)	UE-4 (2)
	RB-17	UE-2 (2)	UE-3 (4)	UE-2 (4)	UE-1 (0)	UE-2 (0)	UE-4 (1)	UE-2 (1)	UE-4 (0)	UE-2 (0)	UE-3 (4)
	RB-16	UE-2 (2)	UE-3 (4)	UE-2 (4)	UE-1 (0)	UE-2 (0)	UE-4 (1)	UE-2 (1)	UE-4 (0)	UE-2 (0)	UE-3 (4)
	RB-15	UE-3 (2)	UE-3 (4)	UE-3 (5)	UE-3 (1)	UE-3 (0)	UE-3 (2)	UE-3 (3)	UE-3 (0)	UE-3 (1)	UE-3 (4)
	RB-14	UE-3 (2)	UE-3 (4)	UE-3 (5)	UE-3 (1)	UE-3 (0)	UE-3 (2)	UE-3 (3)	UE-3 (0)	UE-3 (1)	UE-3 (4)
	RB-13	UE-2 (2)	UE-3 (4)	UE-2 (4)	UE-3 (1)	UE-2 (0)	UE-4 (1)	UE-2 (1)	UE-4 (0)	UE-2 (0)	UE-4 (2)
	RB-12	UE-2 (2)	UE-3 (4)	UE-2 (4)	UE-3 (1)	UE-2 (0)	UE-4 (1)	UE-2 (1)	UE-4 (0)	UE-2 (0)	UE-4 (2)
	RB-11	UE-2 (2)	UE-3 (4)	UE-2 (4)	UE-4 (0)	UE-2 (0)	UE-4 (1)	UE-2 (1)	UE-4 (0)	UE-2 (0)	UE-4 (2)
	RB-10	UE-2 (2)	UE-3 (4)	UE-2 (4)	UE-4 (0)	UE-2 (0)	UE-4 (1)	UE-2 (1)	UE-4 (0)	UE-2 (0)	UE-4 (2)
	RB-9	UE-1 (1)	UE-1 (2)	UE-1 (5)	UE-1 (0)	UE-1 (1)	UE-4 (1)	UE-1 (4)	UE-4 (0)	UE-1 (1)	UE-4 (2)
	RB-8	UE-1 (1)	UE-1 (2)	UE-1 (5)	UE-1 (0)	UE-1 (1)	UE-4 (1)	UE-1 (4)	UE-4 (0)	UE-1 (1)	UE-4 (2)
	RB-7	UE-3 (2)	UE-4 (3)	UE-4 (4)	UE-4 (0)	UE-3 (0)	UE-4 (1)	UE-3 (3)	UE-4 (0)	UE-3 (1)	UE-4 (2)
	RB-6	UE-3 (2)	UE-4 (3)	UE-4 (4)	UE-4 (0)	UE-3 (0)	UE-4 (1)	UE-3 (3)	UE-4 (0)	UE-3 (1)	UE-4 (2)
	RB-5	UE-2 (2)	UE-4 (3)	UE-2 (4)	UE-4 (0)	UE-2 (0)	UE-4 (1)	UE-2 (1)	UE-4 (0)	UE-2 (0)	UE-4 (2)
	RB-4	UE-2 (2)	UE-4 (3)	UE-2 (4)	UE-4 (0)	UE-2 (0)	UE-4 (1)	UE-2 (1)	UE-4 (0)	UE-2 (0)	UE-4 (2)
	RB-3	UE-2 (2)	UE-4 (3)	UE-2 (4)	UE-4 (0)	UE-2 (0)	UE-4 (1)	UE-2 (1)	UE-4 (0)	UE-2 (0)	UE-4 (2)
	RB-2	UE-2 (2)	UE-4 (3)	UE-2 (4)	UE-4 (0)	UE-2 (0)	UE-4 (1)	UE-2 (1)	UE-4 (0)	UE-2 (0)	UE-4 (2)
	RB-1	UE-2 (2)	UE-1 (2)	UE-2 (4)	UE-1 (0)	UE-2 (0)	UE-1 (3)	UE-2 (1)	UE-1 (0)	UE-2 (0)	UE-1 (2)
	RB-0	UE-2 (2)	UE-1 (2)	UE-2 (4)	UE-1 (0)	UE-2 (0)	UE-1 (3)	UE-2 (1)	UE-1 (0)	UE-2 (0)	UE-1 (2)
		Slot-0	Slot-1	Slot-2	Slot-3	Slot-4	Slot-5	Slot-6	Slot-7	Slot-8	Slot-9

Slots in 10 ms Frame

Uplink Scheduler Performance Metrics for Cell ID - 1





### Simulation Visualization

The four types of run-time visualization shown are:

- *Display of CQI values for UEs over the PUSCH bandwidth:* For details, see the 'Channel Quality Visualization' figure for more details.
- *Display of resource grid assignment to UEs:* The 2-D time-frequency grid updates every 10 ms (frame length) and shows the RB allocation to the UEs in the previous frame. The HARQ process for the PUSCH assignments is also shown alongside with the RNTI of the UEs. New transmissions are shown in black and retransmissions are shown in blue using the HARQ process ID of each UE, a retransmission assignment can be mapped to its previously failed transmission. For details, see the 'Resource Grid Allocation' figure for more details.
- *Display of UL scheduling metrics plots:* The 'Uplink Scheduler Performance Metrics' figure includes plots of the: UL throughput (per UE and cell), UL goodput (per UE and cell), resource share percentage among UEs (out of the total UL resources) to convey the fairness of scheduling, and pending UL buffer status of the UEs to show whether UEs are getting sufficient resources. The maximum achievable data rate value for UL throughput is shown with a dashed line in

throughput and goodput plots. The performance metrics plots update for every `metricsStepSize` slots.

- *Display of RLC metrics plot:* The 'RLC throughput visualization' figure represents the throughput of RLC layer (per logical channel) for the selected UE. The RLC metrics plot update for every `metricsStepSize` slots.

### Simulation Logs

The parameters used for the simulation and simulation logs are saved in MAT files for post-simulation analysis and visualization. The simulation parameters are saved in a MAT file with the filename as the value of configuration parameter `simParameters.ParametersLogFile`. The per time step logs, scheduling assignment logs and RLC logs are saved in MAT file `simParameters.SimulationLogFile`. After the simulation, open the file to load `ULTimeStepLogs`, `SchedulingAssignmentLogs` and `RLCLogs` in the workspace.

**Time step logs:** The table shows a sample time step entry. Each row of the table represents a slot.

Frame	Slot	RBG Allocation Bitmap	MCS	HARQ Process	NDI	Tx Type	CQI for UEs	HARQ NDI Status	Throughput Bytes	Goodput Bytes	Buffer Status of UEs
5	0	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
5	1	[00110101010000] [11000000001000] [00001010100111] [00000000000000]	[10 12 8 -1]	[0 3 6 -1]	[0 0 1 -1]	['newTx', 'newTx', 'reTx', 'noTx']	[5 8 9 8 7 8 9 8 9 11 12 .....] [5 7 3 4 9 5 9 7 9 10 15 .....] [3 9 9 6 9 4 9 8 9 9 13 .....] [9 7 7 3 7 8 5 8 8 15 9 .....]	[1 0 0 1 0 1 0 . .] [1 1 0 1 0 1 0 . .] [1 0 0 1 0 1 1 . .] [1 0 0 1 0 1 0 . .]	[46 54 38 0]	[46 54 0 0]	[299480 135671 77567 137070]
5	2	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....

Each row of the table represents a slot and contains the following information:

- *Frame:* Frame number.
- *Slot:* Slot number in the frame.
- *RBG Allocation Bitmap:* *N-by-P* bitmap matrix, where *N* is the number of UEs and *P* is the number of RBGs. If an RBG is assigned to a particular UE, the corresponding bit is set to 1. For example, [0 0 1 1 0 1 0 1 0 1 0 0 0; 1 1 0 0 0 0 0 0 0 0 1 0 0; 0 0 0 0 1 0 1 0 1 0 0 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0] means that the UL bandwidth has 13 RBGs and UE-1 is assigned RBG indices: 2, 3, 5, 7 and 9; UE-2 is assigned the RBG indices 0, 1 and 10; UE-3 is assigned the RBG indices 4, 6, 8, 11 and 12 and UE-4 is not assigned any RBG.
- *MCS:* Row vector of length *N*, where *N* is the number of UEs. Each value corresponds to the modulation and coding scheme (MCS) index for the PUSCH transmission. For example, [10 12 8 -1] means that only UE-1, UE-2, and UE-3 are assigned UL resources for this slot and use MCS values 10, 12, and 8, respectively.
- *HARQ Process:* Row vector of length *N*, where *N* is the number of UEs. The value is the HARQ process ID used by UE for the PUSCH transmission. For example, [0 3 6 -1] means that only UE-1, UE-2, and UE-3 are assigned UL resources for this slot and use the HARQ process IDs 0, 3, and 6, respectively.
- *NDI:* Row vector of length *N*, where *N* is the number of UEs. The value is the NDI flag value in the UL assignment for PUSCH transmission. For example, [0 0 1 -1] means that only UE-1, UE-2, and UE-3 are assigned UL resources for this slot and use the NDI flag values (which determine whether a new transmission or a retransmission is used) 0, 0, and 1, respectively.
- *Tx Type:* Tx Type specifies the transmission type (new transmission or retransmission). Row vector of length *N*, where *N* is the number of UEs. Possible values are either 'newTx', 'reTx', or 'noTx'. 'noTx' means that the UE is not allocated PUSCH resources. For example, ['newTx' 'newTx' 'reTx'



'noTx'] means that only UE-1, UE-2, and UE-3 are assigned UL resources for this slot. UE-1 and UE-2 transmit a new packet from the specified HARQ process, while UE-3 retransmits the packet in the buffer of the specified HARQ process.

- *CQI for UEs*:  $N$ -by- $P$  matrix, where  $N$  is the number of UEs and  $P$  is the number of RBs in the bandwidth. A matrix element at position  $(i, j)$  corresponds to the UL CQI value for UE with RNTI  $i$  at RB  $j$ .
- *HARQ NDI Status*:  $N$ -by- $P$  matrix, where  $N$  is the number of UEs and  $P$  is the number of HARQ processes on UEs. A matrix element at position  $(i, j)$  is the last received NDI flag at UE  $i$  for HARQ process ID  $j$ . For new transmissions, this value and the NDI flag in the PUSCH assignment must toggle. For example, in slot 1 of frame 5 described in the scheduling log, UE-1 uses the HARQ ID 0 and the last NDI flag value for HARQ ID 0 at UE-1 is 1. To indicate a new transmission, the NDI flag values changes to 0 in the PUSCH assignment.
- *Throughput Bytes*: Row vector of length  $N$ , where  $N$  is the number of UEs. The values represent MAC bytes transmitted by UEs in this slot.
- *Goodput Bytes*: Row vector of length  $N$ , where  $N$  is the number of UEs. The values represent new transmission MAC bytes transmitted by UEs in this slot.
- *Buffer Status of UEs*: Row vector of length  $N$ , where  $N$  is the number of UEs. The values represent the amount of pending buffers at UEs.

**Scheduling assignment logs:** Information of all the scheduling assignments and related information is logged in this file. The table shows sample log entries.

'RNTI'	'Frame'	'Slot'	'Grant type'	'RBG Allocation Map'	'Start Sym'	'Num Sym'	'MCS'	'HARQ ID'	'NDI Flag'	'RV'	'Tx Type'	'Feedback Slot Offset (DL grants only)'	'CQI on RBs'
1	0	9	'UL'	'[1 1 1 1 1 1 1 0 0 0 0 0 0]'	0	14	16	0	1	0	'newTx'	'NA'	'[13 14 2 14 10 2 5 9 15 15 3 15 15 8 13 3 7 14 12 15 10 1 13 15 11]'
2	0	9	'UL'	'[0 0 0 0 0 0 0 0 1 0 1 1 1]'	0	14	12	0	1	0	'newTx'	'NA'	'[10 9 5 8 3 9 1 4 1 2 10 9 4 12 1 6 5 10 10 3 6 6 8 9 10]'
3	0	9	'UL'	'[0 0 0 0 0 0 0 1 0 0 0 0 0]'	0	14	15	0	1	0	'newTx'	'NA'	'[3 7 7 2 2 5 10 4 6 3 8 3 6 7 9 10 6 2 2 3 9 3 9 3 10]'
4	0	9	'UL'	'[0 0 0 0 0 0 0 0 1 0 0 0 0]'	0	14	15	0	1	0	'newTx'	'NA'	'[4 2 3 7 5 4 9 6 6 10 3 8 8 4 6 1 1 6 8 10 2 6 5 1 4]'
2	1	0	'UL'	'[0 0 0 0 0 0 0 0 0 0 0 1 1]'	0	14	15	1	1	0	'newTx'	'NA'	'[10 9 5 8 3 9 1 4 1 2 10 9 4 12 1 6 5 10 10 3 6 6 8 9 10]'
3	1	0	'UL'	'[1 0 0 1 0 1 1 1 1 0 1 0 0]'	0	14	9	1	1	0	'newTx'	'NA'	'[3 7 7 2 2 5 10 4 6 3 8 3 6 7 9 10 6 2 2 3 9 3 9 3 10]'
4	1	0	'UL'	'[0 1 1 0 1 0 0 0 1 0 0 0 0]'	0	14	10	1	1	0	'newTx'	'NA'	'[4 2 3 7 5 4 9 6 6 10 3 8 8 4 6 1 1 6 8 10 2 6 5 1 4]'

**RLC logs:** Each row in the RLC logs represents a slot and contains this information:

- *Frame*: Frame number.
- *Slot*: Slot number in the frame.
- *UE RLC statistics*:  $N$ -by- $P$  cell, where  $N$  is the product of the number of UEs and the number of logical channels, and  $P$  is the number of statistics collected. Each row represents statistics of a logical channel in a UE. The last row contains the cumulative RLC statistics of the entire simulation.
- *gNB RLC statistics*:  $N$ -by- $P$  cell, where  $N$  is the product of the number of UEs and the number of logical channels, and  $P$  is the number of statistics collected. Each row represents statistics of a logical channel of a UE at gNB. The last row contains the cumulative RLC statistics of the entire simulation.

Each row of the UE and gNB RLC statistics table represents a logical channel of a UE and contains:

- *RNTI*: Radio network temporary identifier of a UE.
- *LCID*: Logical channel identifier.
- *TxDataPDU*: Number of data PDUs sent by RLC to MAC layer.
- *TxDataBytes*: Number of data bytes sent by RLC to MAC layer.
- *ReTxDataPDU*: Number of data PDUs retransmitted by RLC to MAC layer.
- *ReTxDataBytes*: Number of data bytes retransmitted by RLC to MAC layer.

- *TxControlPDU*: Number of control PDUs sent by RLC to MAC layer.
- *TxControlBytes*: Number of control bytes sent by RLC to MAC layer.
- *TxPacketsDropped*: Number of RLC SDUs dropped by RLC due to Tx buffer overflow.
- *TxBytesDropped*: Number of bytes dropped by RLC due to Tx buffer overflow.
- *TimerPollRetransmitTimedOut*: Number of times the poll retransmit timer expired.
- *RxDataPDU*: Number of data PDUs received by RLC from MAC layer.
- *RxDataBytes*: Number of data bytes received by RLC from MAC layer.
- *RxDataPDUDropped*: Number of received data PDUs from MAC which are dropped by RLC layer.
- *RxDataBytesDropped*: Number of received data bytes from MAC which are dropped by RLC layer.
- *RxDataPDUDuplicate*: Number of duplicate PDUs received by RLC from MAC layer.
- *RxDataBytesDuplicate*: Number of duplicate data bytes received by RLC from MAC layer.
- *RxControlPDU*: Number of control PDUs received by RLC from MAC layer.
- *RxControlBytes*: Number of control bytes received by RLC from MAC layer.
- *TimerReassemblyTimedOut*: Number of times the reassembly timer expired.
- *TimerStatusProhibitTimedOut*: Number of times the status prohibit timer expired.

You can run the script `NRPostSimVisualization` to get a post simulation visualization of logs. In the post simulation script, you are provided with variable `isLogReplay`, which provides these options to visualize 'Resource Grid Allocation' and 'Channel Quality Visualization' figures.

- Set `isLogReplay` to `true` for a replay of the simulation logs.
- Set `isLogReplay` to `false` to analyze the simulation logs and input the frame number to visualize the scheduling information of the particular frame

```
% Read the logs and save them in MAT-files
simulationLogs = cell(1, 1);
simulationLogs{1} = struct('ULTimeStepLogs',[], 'SchedulingAssignmentLogs',[] , 'RLCLogs',[]);
[~, simulationLogs{1}.ULTimeStepLogs] = getSchedulingLogs(simSchedulingLogger); % UL time step s
simulationLogs{1}.SchedulingAssignmentLogs = getGrantLogs(simSchedulingLogger); % Scheduling ass
simulationLogs{1}.RLCLogs = getRLCLogs(simRLCLogger); % RLC statistics logs
save(simParameters.SimulationLogFile, 'simulationLogs'); % Save simulation logs in a MAT-file
save(simParameters.ParametersLogFile, 'simParameters'); % Save simulation parameters in a MAT-fi
```

## Appendix

The example uses these helper functions and classes:

- `hNRNode.m`: NR node base class for both gNB and UE
- `hNRGNB.m`: gNB node functionality
- `hNRUE.m`: UE node functionality
- `hNRRLCEntity.m`: Base class for RLC UM and AM entities
- `hNRUMEntity.m`: RLC UM functionality
- `hNRRLCDataPDUInfo.m`: Creates RLC PDU information object
- `hNRRLCDataReassembly.m`: Create an RLC SDU reassembly information object
- `hNRRLCBufferStatus.m`: Generates RLC buffer status information object
- `hNRMAC.m`: NR MAC base class functionality

- hNRGNBMAC.m: gNB MAC functionality
- hNRUEMAC.m: UE MAC functionality
- hNRScheduler.m: Core MAC scheduler functionality
- hNRSchedulerBestCQI.m: Implements best CQI scheduling strategy
- hNRSchedulerProportionalFair.m: Implements proportional fair scheduling strategy
- hNRSchedulerRoundRobin.m: Implements round robin scheduling strategy
- hNRMACBSR.m: Generates buffer status report
- hNRMACBSRParser.m: Parses buffer status report
- hNRMACSubPDU.m: Generates MAC subPDU
- hNRMACPaddingSubPDU.m: Generates MAC subPDU with padding
- hNRMACMultiplex.m: Generates MAC PDU
- hNRMACPDUParser.m: Parses MAC PDU
- hNewHARQProcesses.m: Creates new HARQ process
- hUpdateHARQProcess.m: Updates HARQ process
- hNRPhyInterface.m: PHY layer interface class
- hNRGNBPassthroughPhy.m: gNB passthrough PHY layer
- hNRUEPassthroughPhy.m: UE passthrough PHY layer
- hNRPUSCHInfo.m: PUSCH information structure passed by MAC to PHY layer
- hNRRxIndicationInfo.m: Information structure passed by PHY layer to MAC along with MAC PDU
- hNRPacketDistribution.m: Creates packet distribution object
- hNRUplinkGrantFormat.m: UL assignment format
- hNRRLCLogger.m: Implements RLC statistics logging and visualization functionality
- hNRSchedulingLogger.m: Implements scheduling information logging and visualization functionality
- hNRULSchedulingValidateConfig.m: Validates simulation configuration
- hNRSetUpPacketDistribution.m: Set up packet distribution functionality
- hNRPacketWriter.m: Captures MAC packets
- hNRPacketInfo.m: Metadata format for capturing MAC packets
- NRPostSimVisualization.m: Post simulation visualization script

## References

- [1] 3GPP TS 38.214. "NR; Physical layer procedures for data." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network.*
- [2] 3GPP TS 38.321. "NR; Medium Access Control (MAC) protocol specification." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network.*
- [3] 3GPP TS 38.322. "NR; Radio Link Control (RLC) protocol specification." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network.*

## **See Also**

### **Related Examples**

- “NR FDD Scheduling Performance Evaluation” on page 5-21

## NR FDD Scheduling Performance Evaluation

This example shows scheduling in frequency division duplexing (FDD) mode with medium access control (MAC) logical channel prioritization (LCP) functionality and evaluates the network performance. Scheduling strategy controls the downlink (DL) and uplink (UL) scheduling of resources and can be customized to evaluate the performance. The example shows the functionality of the radio link control (RLC) layer in unacknowledged mode (UM) with LCP to serve multiple logical channels. You can model different application traffic patterns. A passthrough physical (PHY) layer without any physical layer processing is used, which adopts a probability-based approach to model packet reception failures. You can switch to 5G Toolbox™ based PHY layer for high fidelity physical layer processing. The performance of the scheduling strategy is evaluated in terms of achieved throughput and the fairness in resource sharing.

### Introduction

This example shows how a scheduling strategy (controlled by the gNB) assigns UL and DL resources among the UEs. The example considers the following operations within the gNB and UEs that facilitate UL and DL transmissions and receptions.

	Operations
gNB	<ul style="list-style-type: none"> <li>• Run the scheduling algorithm to assign uplink and downlink resources</li> <li>• Send the uplink and downlink assignments to the UEs</li> <li>• Receive the PUSCH transmissions from the UEs</li> <li>• Adhere to the downlink assignments for PDSCH transmission</li> <li>• Receive the feedback of PDSCHs from the UEs</li> </ul>
UEs	<ul style="list-style-type: none"> <li>• Send the pending buffer status report to the gNB</li> <li>• Receive the uplink and downlink assignments from the gNB</li> <li>• Adhere to the received uplink assignments from the gNB for PUSCH transmission</li> <li>• Receive the PDSCH transmissions from the gNB</li> <li>• Send feedback for the received PDSCHs</li> </ul>

The complete PUSCH or PDSCH packet is transmitted in the first symbol of its allocated symbol set. Receiver processes the packet in the symbol just after the last symbol in the allocated symbol set.

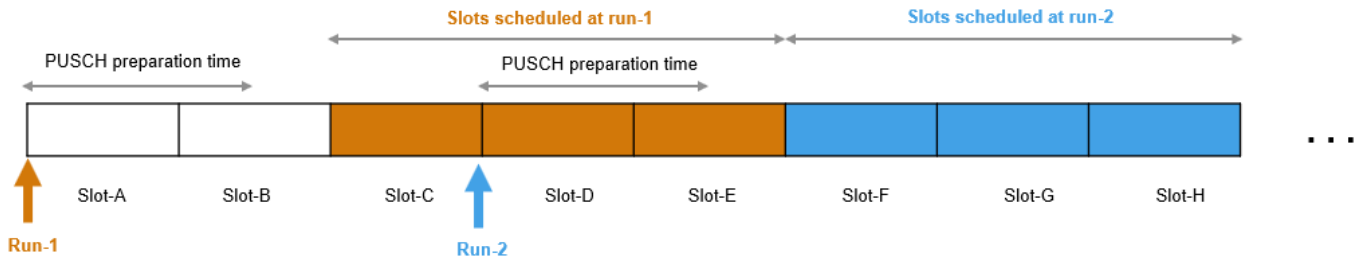
The schedulers (UL and DL) run every  $p$  slots to assign the UL and DL resources, where  $p$  is the configured periodicity of the scheduler. In each run, the number of slots scheduled is equal to the periodicity of scheduler run,  $p$ .

### UL Scheduler

The first slot, among the  $p$  slots to be scheduled in a run, is the nearest upcoming slot which satisfies the PUSCH preparation time capability of UEs. For example, the figure shows the way scheduler selects the slots during two consecutive runs. It assumes scheduler periodicity ( $p$ ) as 3 slots. Hence, the scheduler runs after every 3 slots, and schedules resources for 3 slots. PUSCH preparation time capability for all the UEs is assumed as greater than 1 slot (14 symbols) but less than 2 slots (28 symbols).

- Run-1: When scheduler runs at the start of Slot-A, it schedules 3 slots starting from Slot-C, because for Slot-A and Slot-B the UEs do not get enough PUSCH preparation time (UEs gets a

time of 0 symbols at start of Slot-A and 14 symbols at start of Slot-B). For Slot-C, UEs get 28 symbols for PUSCH preparation and this satisfies the PUSCH preparation time capability. Hence, Slot-C, D, and E are scheduled in this run.

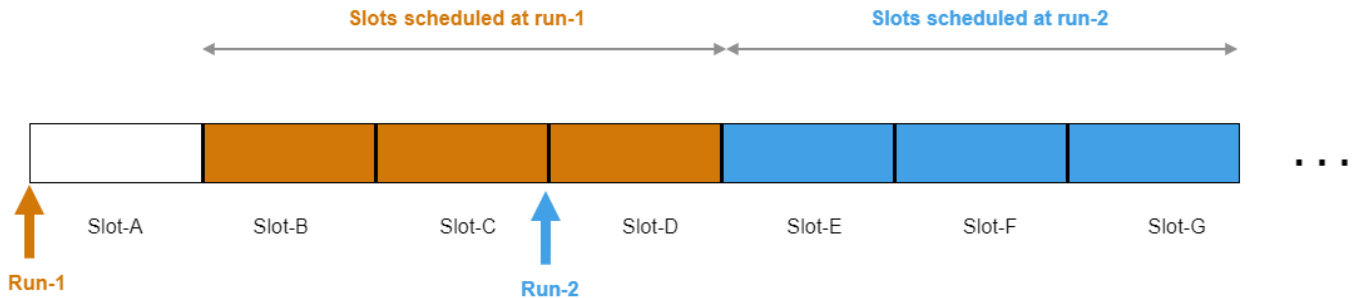


- Run-2: When scheduler runs at the start of Slot-D, it schedules the next 3 contiguous slots starting from Slot-F (Slot-F, G, and H).

**DL Scheduler**

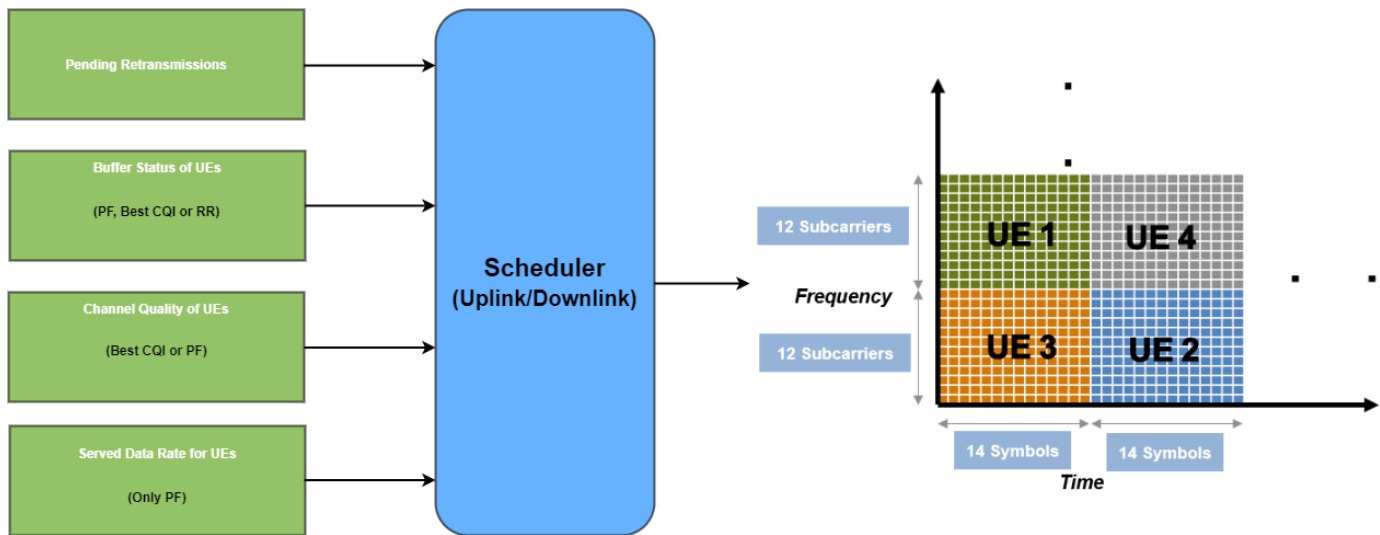
The first slot, among the  $p$  slots to be scheduled in a run, is the immediate next slot.

- Run-1: When scheduler runs at the start of Slot-A, it schedules 3 contiguous slots Slot-B, C, and D.



- Run-2: When scheduler runs at the start of Slot-D, it schedules 3 contiguous slots Slot-E, F, and G.

You can choose any one of the implemented scheduling strategies: proportional fair (PF), best CQI, or round robin (RR). The various supported inputs to the UL scheduler are listed along with the scheduling strategies that consider them.



The control packets required are assumed to be sent out of band without the need of resources for transmission. The control packets are UL assignment, DL assignment, buffer status report (BSR), and PDSCH feedback.

The demodulation reference signal (DM-RS) is not modeled in this example. However, one symbol is kept unused for it in the PUSCH and PDSCH assignments.

This example models:

- Slot-based UL and DL scheduling. The time-domain granularity of the UL assignment and DL assignment is one slot.
- Noncontiguous allocation of frequency-domain resources in terms of resource block groups (RBGs).
- Configurable subcarrier spacing resulting in different slot durations.
- Asynchronous adaptive hybrid automatic repeat request (HARQ) mechanism in UL and DL.
- Multiple logical channels to support different kind of applications.
- Logical channel prioritization (LCP) to distribute the received assignment among logical channels per UE for UL and DL.

### Scenario Configuration

Set the parameters for simulation.

```
rng('default'); % Reset the random number generator
simParameters = []; % Clear the simulation parameters

simParameters.NumFramesSim = 100; % Simulation time in terms of number of 10 ms frames
% Number of UEs in the simulation. UEs are assumed to have sequential radio
% network temporary identifiers (RNTIs) from 1 to NumUEs. If you change the
% number of UEs, ensure that the length of simParameters.UEDistance is equal to NumUEs
simParameters.NumUEs = 4;
simParameters.UEDistance = [100 250 700 750]; % Distance of UEs from gNB (in meters)

% Set the channel bandwidth to 30 MHz and subcarrier spacing (SCS) to 15
% kHz as defined in 3GPP TS 38.104 Section 5.3.2. The complete UL and
```

```

% DL bandwidth is assumed to be allotted for PUSCH and PDSCH. The
% UL and DL carriers are assumed to have symmetric channel
% bandwidth
simParameters.DLBandwidth = 30e6; % Hz
simParameters.ULBandwidth = 30e6; % Hz
simParameters.NumRBs = 160;
simParameters.SCS = 15; % kHz
simParameters.DLCarrierFreq = 2.635e9; % Hz
simParameters.ULCarrierFreq = 2.515e9; % Hz

% Configure parameters to update UL channel quality at gNB and DL
% channel quality at gNB and UE. Channel conditions are periodically
% improved or deteriorated by CQIDelta every channelUpdatePeriodicity
% seconds for all RBs of a UE. Whether channel conditions for a particular
% UE improve or deteriorate is randomly determined
% RBCQI = RBCQI +/- CQIDelta
simParameters.ChannelUpdatePeriodicity = 0.2; % In sec
simParameters.CQIDelta = 2;
% Mapping between distance from gNB (first column in meters) and maximum
% achievable UL CQI value (second column). For example, if a UE is 700
% meters away from the gNB, it can achieve a maximum CQI value of 10 as the
% distance falls within the [501, 800] meters range, as per the mapping.
% Set the distance in increasing order and the maximum achievable CQI value in
% decreasing order
simParameters.CQIvsDistance = [
    200 15;
    500 12;
    800 10;
    1000 8;
    1200 7];

simParameters.BSRPeriodicity = 5; % In ms
simParameters.EnableHARQ = true; % Flag to enable or disable HARQ. If disabled, there are no retransmissions
simParameters.NumHARQ = 16; % Number of HARQ processes

% Set the scheduler run periodicity in terms of number of slots. Value must
% be less than the number of slots in a 10 ms frame
simParameters.SchedulerPeriodicity = 4;
simParameters.SchedulerStrategy = 'PF'; % Supported scheduling strategies: 'PF', 'RR' and 'BestEffort'
% Moving average parameter within the range [0, 1] to calculate average
% data rate for a UE in UL and DL directions. This value is used in
% the PF scheduling strategy. Parameter value closer to 1 implies more
% weight on the instantaneous data rate. Parameter value closer to 0
% implies more weight on the past data rate
% AverageDataRate = ((1 - MovingAvgDataRateWeight) * PastDataRate) + (MovingAvgDataRateWeight * CurrentDataRate)
simParameters.MovingAvgDataRateWeight = 0.5;
% gNB ensures that PUSCH assignment is received at UEs PUSCHPrepTime ahead
% of the transmission time
simParameters.PUSCHPrepTime = 200; % In microseconds
% Maximum RBs allotted to a UE in a slot for the UL and DL
% transmission (limit is applicable for new PUSCH and PDSCH assignments and
% not for the retransmissions)
simParameters.RBAllocationLimitUL = 100; % For PUSCH
simParameters.RBAllocationLimitDL = 100; % For PDSCH

```

Load the logical channel configuration table. Each row in the table represents one logical channel and has these properties as columns.



- RNTI - Radio network temporary identifier of the UE.
- LogicalChannelID - Logical channel identifier.
- LCGID - Logical channel group identifier.
- SeqNumFieldLength - Defines the sequence number field length. It takes either 6 or 12.
- MaxTxBufferSDUs - Maximum Tx buffer size (in terms of number of packets).
- ReassemblyTimer - Defines the reassembly timer (in ms).
- EntityType - Defines the RLC entity type. It takes values 0, 1, and 2, which indicates whether the RLC UM entity is unidirectional DL, unidirectional UL, or bidirectional UM, respectively.
- Priority - Priority of the logical channel.
- PBR - Prioritized bit rate (in kilo bytes per second).
- BSD - Bucket size duration (in ms).

```
load('NRFDDRLCChannelConfig.mat')
simParameters.RLCChannelConfig = RLCChannelConfig;
```

Load the application configuration table containing these fields. Each row in the table represents one application and has these properties as columns.

- PacketInterval - Interval between two consecutive packet generations (in ms).
- PacketSize - Size of the packet (in bytes).
- HostDevice - Defines the device (UE or gNB) on which the application is installed with the specified configuration. The device takes values 0, 1, or 2. The values indicate the application is configured on gNB side, UE side, or both UE and gNB respectively.
- RNTI - Radio network temporary identifier of a UE. This identifies the UE for which the application is installed.
- LCID - Logical channel identifier.

```
load('NRFDDAppConfig.mat');
simParameters.AppConfig = AppConfig;
```

#### Logging and Visualization Configuration

```
% The parameters CQIVisualization and RBVisualization control the display
% of these visualizations: (i) CQI visualization of RBs (ii) RB assignment
% visualization. By default, these plots are disabled. You can enable them
% by setting to 'true'
simParameters.CQIVisualization = false;
simParameters.RBVisualization = false;
% The output metrics plots are updated periodically NumMetricsSteps times within the
% simulation duration
simParameters.NumMetricsSteps = 20;
% MAT-files to write the logs into. They are used for post simulation analysis and visualization
simParameters.ParametersLogFile = 'simParameters'; % For logging the simulation parameters
simParameters.SimulationLogFile = 'simulationLogs'; % For logging the simulation logs

% Validate the simulation configuration
hNRSchedulingFDDValidateConfig(simParameters);
```

#### Derived Parameters

Based on the primary configuration parameters, compute the derived parameters. Additionally, set some example specific constants.

```

simParameters.DuplexMode = 0; % FDD
% Size of sub-band for CQI reporting in terms of number of RBs (Only used
% when 5G Toolbox™ PHY layer processing is enabled)
simParameters.SubbandSize = 16;
simParameters.NumCells = 1; % Number of cells
simParameters.NCellID = 1; % Physical cell ID
simParameters.GNBPosition = [0 0 0]; % Position of gNB in (x,y,z) coordinates
% Slot duration for the selected SCS and number of slots in a 10 ms frame
slotDuration = 1/(simParameters.SCS/15); % In ms
numSlotsFrame = 10/slotDuration; % Number of slots in a 10 ms frame
numSlotsSim = simParameters.NumFramesSim * numSlotsFrame; % Number of slots in the simulation

% Calculate maximum achievable CQI value for the UEs based on their distance from
% the gNB
maxUECQIs = zeros(simParameters.NumUEs, 1); % To store the maximum achievable CQI value for UEs
for ueIdx = 1:simParameters.NumUEs
    % Based on the distance of the UE from gNB, find matching row in
    % CQIvsDistance mapping
    matchingRowIndex = find(simParameters.CQIvsDistance(:, 1) > simParameters.UEDistance(ueIdx));
    if isempty(matchingRowIndex)
        maxUECQIs(ueIdx) = simParameters.CQIvsDistance(end, 2);
    else
        maxUECQIs(ueIdx) = simParameters.CQIvsDistance(matchingRowIndex(1), 2);
    end
end

% Interval at which metrics visualization updates in terms of number of
% slots. As one slot is the finest time-granularity of the simulation, make
% sure that MetricsStepSize is an integer
simParameters.MetricsStepSize = ceil(numSlotsSim / simParameters.NumMetricsSteps);
if mod(numSlotsSim, simParameters.NumMetricsSteps) ~= 0
    % Update the NumMetricsSteps parameter if NumSlotsSim is not
    % completely divisible by it
    simParameters.NumMetricsSteps = floor(numSlotsSim / simParameters.MetricsStepSize);
end

% Define initial UL and DL channel quality as an N-by-P matrix,
% where 'N' is the number of UEs and 'P' is the number of RBs in the carrier
% bandwidth. The initial value of CQI for each RB, for each UE, is given
% randomly and is limited by the maximum achievable CQI value corresponding
% to the distance of the UE from gNB
simParameters.InitialChannelQualityUL = zeros(simParameters.NumUEs, simParameters.NumRBs); % To
simParameters.InitialChannelQualityDL = zeros(simParameters.NumUEs, simParameters.NumRBs); % To
for ueIdx = 1:simParameters.NumUEs
    % Assign random CQI values for the RBs, limited by the maximum achievable CQI value
    simParameters.InitialChannelQualityUL(ueIdx, :) = randi([1 maxUECQIs(ueIdx)], 1, simParameters
    % Initially, DL and UL CQI values are assumed to be equal
    simParameters.InitialChannelQualityDL(ueIdx, :) = simParameters.InitialChannelQualityUL(ueIdx, :);
end

```

### gNB and UEs Setup

Create the gNB and UE objects, initialize the channel quality information for UEs, and set up the logical channel at gNB and UE. The helper classes `hNRGNB.m` and `hNRUE.m` create gNB and UE nodes respectively, containing the RLC and MAC layers. For MAC layer, `hNRGNB.m` uses the helper class `hNRGNBMAC.m` to implement the gNB MAC functionality and `hNRUE.m` uses `hNRUEMAC.m` to implement the UE MAC functionality. Schedulers are implemented in `hNRSchedulerRoundRobin.m`

(RR), hNRSchedulerProportionalFair.m (PF), hNRSchedulerBestCQI.m (Best CQI) . All the schedulers inherit from the base class hNRScheduler.m which contains the core scheduling functionality. For RLC layer, both hNRGNB.m and hNRUE.m use hNRUMEntity.m to implement the functionality of the RLC transmitter and receiver. Passthrough PHY layer for UE and gNB is implemented in hNRUEPassThroughPhy.m and hNRGNBPassThroughPhy.m, respectively.

```

simParameters.Position = simParameters.GNBPosition;
gNB = hNRGNB(simParameters); % Create gNB node

% Create and add scheduler
switch(simParameters.SchedulerStrategy)
    case 'RR' % Round robin scheduler
        scheduler = hNRSchedulerRoundRobin(simParameters);
    case 'PF' % Proportional fair scheduler
        scheduler = hNRSchedulerProportionalFair(simParameters);
    case 'BestCQI' % Best CQI scheduler
        scheduler = hNRSchedulerBestCQI(simParameters);
end
addScheduler(gNB, scheduler); % Add scheduler to gNB

gNB.PhyEntity = hNRGNBPassThroughPhy(simParameters); % Add passthrough PHY
configurePhy(gNB, simParameters);
setPhyInterface(gNB); % Set the interface to PHY layer

% Create the set of UE nodes
UEs = cell(simParameters.NumUEs, 1);
for ueIdx=1:simParameters.NumUEs
    simParameters.Position = [simParameters.UEDistance(ueIdx) 0 0]; % Position of UE
    UEs{ueIdx} = hNRUE(simParameters, ueIdx);
    UEs{ueIdx}.PhyEntity = hNRUEPassThroughPhy(simParameters, ueIdx); % Add passthrough PHY
    configurePhy(UEs{ueIdx}, simParameters);
    setPhyInterface(UEs{ueIdx}); % Set the interface to PHY layer

    % Initialize the UL CQI values at gNB
    updateChannelQuality(gNB, simParameters.InitialChannelQualityUL(ueIdx, :), 1, ueIdx); % 1 for UL
    % Initialize the DL CQI values at gNB and UE. The DL CQI values
    % help gNB in scheduling, and UE in packet error probability estimation
    updateChannelQuality(gNB, simParameters.InitialChannelQualityDL(ueIdx, :), 0, ueIdx); % 0 for DL
    updateChannelQuality(UEs{ueIdx}, simParameters.InitialChannelQualityDL(ueIdx, :));
end

% Setup logical channels
for lchInfoIdx = 1:size(simParameters.RLCChannelConfig, 1)
    rlcChannelConfigStruct = table2struct(simParameters.RLCChannelConfig(lchInfoIdx, :));
    ueIdx = simParameters.RLCChannelConfig.RNTI(lchInfoIdx);
    % Setup the logical channel at gNB and UE
    gNB.configureLogicalChannel(ueIdx, rlcChannelConfigStruct);
    UEs{ueIdx}.configureLogicalChannel(ueIdx, rlcChannelConfigStruct);
end

% Add data traffic pattern generators to gNB and UE nodes
for appIdx = 1:size(simParameters.AppConfig, 1)
    device = simParameters.AppConfig.HostDevice(appIdx);
    rnti = simParameters.AppConfig.RNTI(appIdx);
    lcid = simParameters.AppConfig.LCID(appIdx);
    packetSize = simParameters.AppConfig.PacketSize(appIdx);
    packetInterval = simParameters.AppConfig.PacketInterval(appIdx);

```

```

% Calculate the data rate (in kbps) of On-Off traffic pattern using
% packet size (in bytes) and packet interval (in ms)
dataRate = ceil(1000/packetInterval) * packetSize * 8e-3;
% Limit the size of the generated application packet to the maximum RLC
% SDU size. The maximum supported RLC SDU size is 9000 bytes
if packetSize > 9000
    packetSize = 9000;
end
% Create an object for On-Off network traffic pattern and add it to the
% specified UE. This object generates the uplink data traffic on the UE
if device == 1 || device == 2
    ulApp = networkTrafficOnOff('PacketSize', packetSize, 'GeneratePacket', true, ...
        'OnTime', simParameters.NumFramesSim/100, 'OffTime', 0, 'DataRate', dataRate);
    UEs{rnti}.addApplication(rnti, lcid, ulApp);
end
% Create an object for On-Off network traffic pattern for the specified
% UE and add it to the gNB. This object generates the downlink data
% traffic on the gNB for the UE
if device == 0 || device == 2
    dlApp = networkTrafficOnOff('PacketSize', packetSize, 'GeneratePacket', true, ...
        'OnTime', simParameters.NumFramesSim/100, 'OffTime', 0, 'DataRate', dataRate);
    gNB.addApplication(rnti, lcid, dlApp);
end
end

% Setup the UL and DL packet distribution mechanism
simParameters.MaxReceivers = simParameters.NumUEs;
% Create DL packet distribution object
dlPacketDistributionObj = hNRPacketDistribution(simParameters, 0); % 0 for DL
% Create UL packet distribution object
ulPacketDistributionObj = hNRPacketDistribution(simParameters, 1); % 1 for UL
hNRSetUpPacketDistribution(simParameters, gNB, UEs, dlPacketDistributionObj, ulPacketDistributionObj);

```

### Processing Loop

Simulation is run slot by slot. In each slot, these operations are executed:

- Run the MAC and PHY layers of gNB
- Run the MAC and PHY layers of UEs
- Layer specific logging and visualization
- Advance the timer for the nodes. Every 1 ms it also sends trigger to application and RLC layers. Application layer and RLC layer execute their scheduled operations based on 1 ms timer trigger.

```

% To store these UE metrics for each slot: throughput bytes
% transmitted, goodput bytes transmitted, and pending buffer amount bytes.
% The number of goodput bytes is calculated by excluding the
% retransmissions from the total transmissions
UESlotMetricsUL = zeros(simParameters.NumUEs, 3);
UESlotMetricsDL = zeros(simParameters.NumUEs, 3);

% To store the RLC statistics for each slot
ueRLCStats = cell(simParameters.NumUEs, 1);
gNBRLCStats = cell(simParameters.NumUEs, 1);

% To store current UL and DL CQI values on the RBs for different UEs
uplinkChannelQuality = zeros(simParameters.NumUEs, simParameters.NumRBs);
downlinkChannelQuality = zeros(simParameters.NumUEs, simParameters.NumRBs);

```

```

% To store last received UL and DL HARQ process NDI flag value at UE
HARQProcessStatusUL = zeros(simParameters.NumUEs, simParameters.NumHARQ);
HARQProcessStatusDL = zeros(simParameters.NumUEs, simParameters.NumHARQ);

% Create an object for MAC (UL & DL) scheduling information visualization and logging
simSchedulingLogger = hNRSchedulingLogger(simParameters);

% To store the logical channel information associated per UE
lchInfo = repmat(struct('LCID',[],'EntityDir',[']), [simParameters.NumUEs 1]);
for ueIdx = 1:simParameters.NumUEs
    lchInfo(ueIdx).LCID = simParameters.RLCChannelConfig.LogicalChannelID(simParameters.AppConfig);
    lchInfo(ueIdx).EntityDir = simParameters.RLCChannelConfig.EntityType(simParameters.AppConfig);
end

% Create an object for RLC visualization and logging
simRLCLogger = hNRRLCLogger(simParameters, lchInfo);

symbolNum = 0;
% Run processing loop
for slotNum = 1:numSlotsSim

    % Run MAC and PHY layers of gNB
    run(gNB.MACEntity);
    run(gNB.PhyEntity);

    % Run MAC and PHY layers of UEs
    for ueIdx = 1:simParameters.NumUEs
        % Read the last received NDI flags for HARQ processes for
        % logging (Reading it before it gets overwritten by run function of MAC)
        HARQProcessStatusUL(ueIdx, :) = getLastNDIFlagHarq(UEs{ueIdx}.MACEntity, 1); % 1 for UL
        HARQProcessStatusDL(ueIdx, :) = getLastNDIFlagHarq(UEs{ueIdx}.MACEntity, 0); % 0 for DL
        run(UEs{ueIdx}.MACEntity);
        run(UEs{ueIdx}.PhyEntity);
    end

    % RLC logging
    for ueIdx = 1:simParameters.NumUEs % For all UEs
        % Get RLC statistics
        ueRLCStats{ueIdx} = getRLCStatistics(UEs{ueIdx}, ueIdx);
        gNBRLCStats{ueIdx} = getRLCStatistics(gNB, ueIdx);
    end
    logRLCStats(simRLCLogger, ueRLCStats, gNBRLCStats); % Update RLC statistics logs

    % MAC logging
    % Read UL and DL assignments done by gNB MAC scheduler
    % at current time. Resource assignments returned by a scheduler (either
    % UL or DL) is empty, if either scheduler was not scheduled to run at
    % the current time or no resources got scheduled
    [resourceAssignmentsUL, resourceAssignmentsDL] = getCurrentSchedulingAssignments(gNB.MACEntity);
    % Read throughput and goodput bytes sent for each UE
    [UESlotMetricsDL(:, 1), UESlotMetricsDL(:, 2)] = getTTIBytes(gNB);
    UESlotMetricsDL(:, 3) = getBufferStatus(gNB); % Read pending buffer (in bytes) on gNB
    for ueIdx = 1:simParameters.NumUEs
        % Read the UL channel quality at gNB for each of the UEs for logging
        uplinkChannelQuality(ueIdx, :) = getChannelQuality(gNB, 1, ueIdx); % 1 for UL
        % Read the DL channel quality at gNB for each of the UEs for logging
        downlinkChannelQuality(ueIdx, :) = getChannelQuality(gNB, 0, ueIdx); % 0 for DL
    end
end

```

```

        % Read throughput and goodput bytes transmitted for this UE in the current TTI for logging
        [UESlotMetricsUL(ueIdx, 1), UESlotMetricsUL(ueIdx, 2)] = getTTIBytes(UEs{ueIdx});
        UESlotMetricsUL(ueIdx, 3) = getBufferStatus(UEs{ueIdx}); % Read pending buffer (in bytes)
    end
    % Update scheduling logs based on the current slot run of UEs and gNB.
    % Logs are updated in each slot, RB grid visualizations are updated
    % every frame, and metrics plots are updated every metricsStepSize
    % slots
    logScheduling(simSchedulingLogger, symbolNum + 1, resourceAssignmentsUL, UESlotMetricsUL, up);
    logScheduling(simSchedulingLogger, symbolNum + 1, resourceAssignmentsDL, UESlotMetricsDL, down);

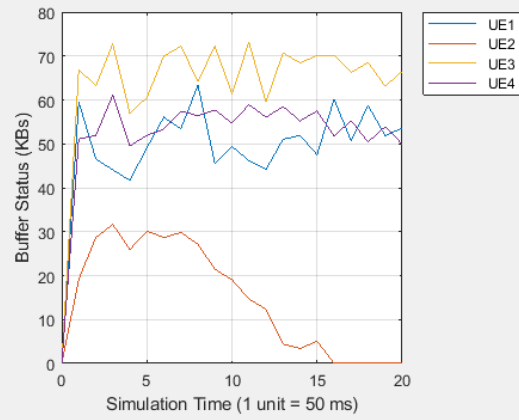
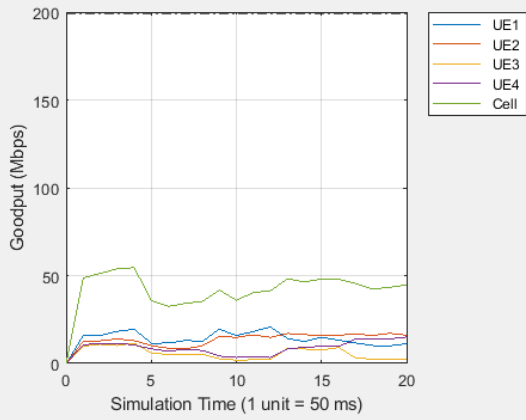
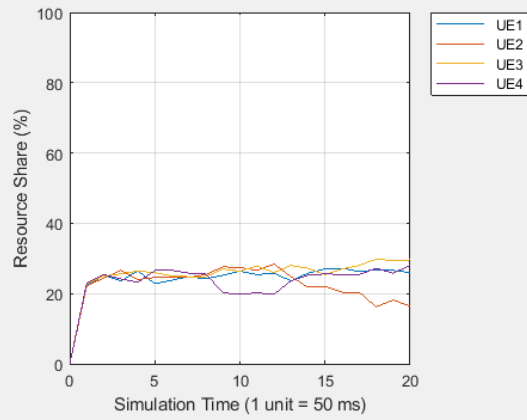
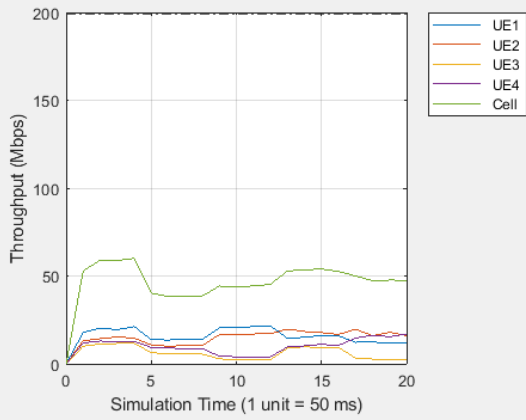
    % Visualization
    % RB assignment visualization (if enabled)
    if simParameters.RBVisualization
        if mod(slotNum, numSlotsFrame) == 0
            plotRBGrids(simSchedulingLogger);
        end
    end
    % CQI grid visualization (if enabled)
    if simParameters.CQIVisualization
        if mod(slotNum, numSlotsFrame) == 0
            plotCQIRBGrids(simSchedulingLogger);
        end
    end
    % Plot scheduler metrics and RLC metrics visualization at slot
    % boundary, if the update periodicity is reached
    if mod(slotNum, simParameters.MetricsStepSize) == 0
        plotMetrics(simSchedulingLogger);
        plotMetrics(simRLCLogger);
    end

    % Advance timer ticks for gNB and UEs by the number of symbols per slot
    advanceTimer(gNB, 14);
    for ueIdx = 1:simParameters.NumUEs
        advanceTimer(UEs{ueIdx}, 14);
    end

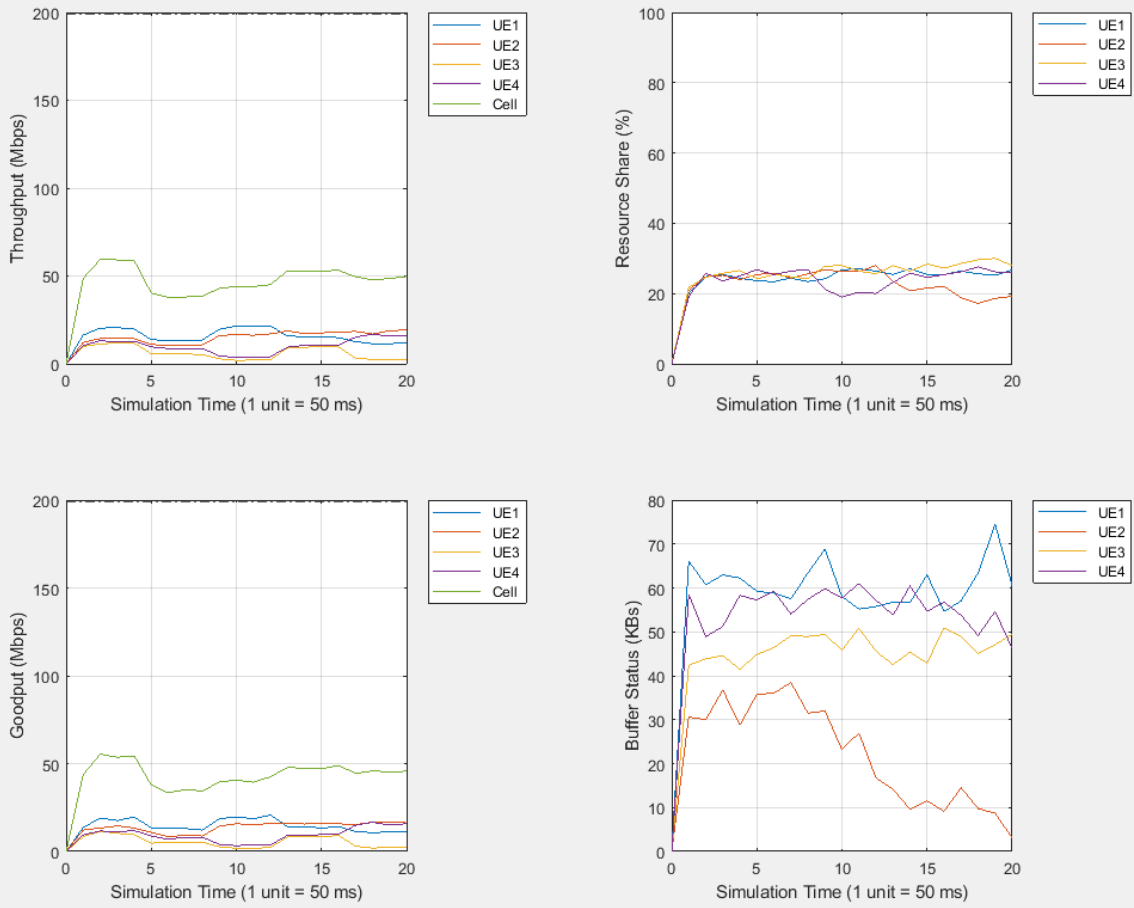
    % Symbol number in the simulation
    symbolNum = symbolNum + 14;
end

```

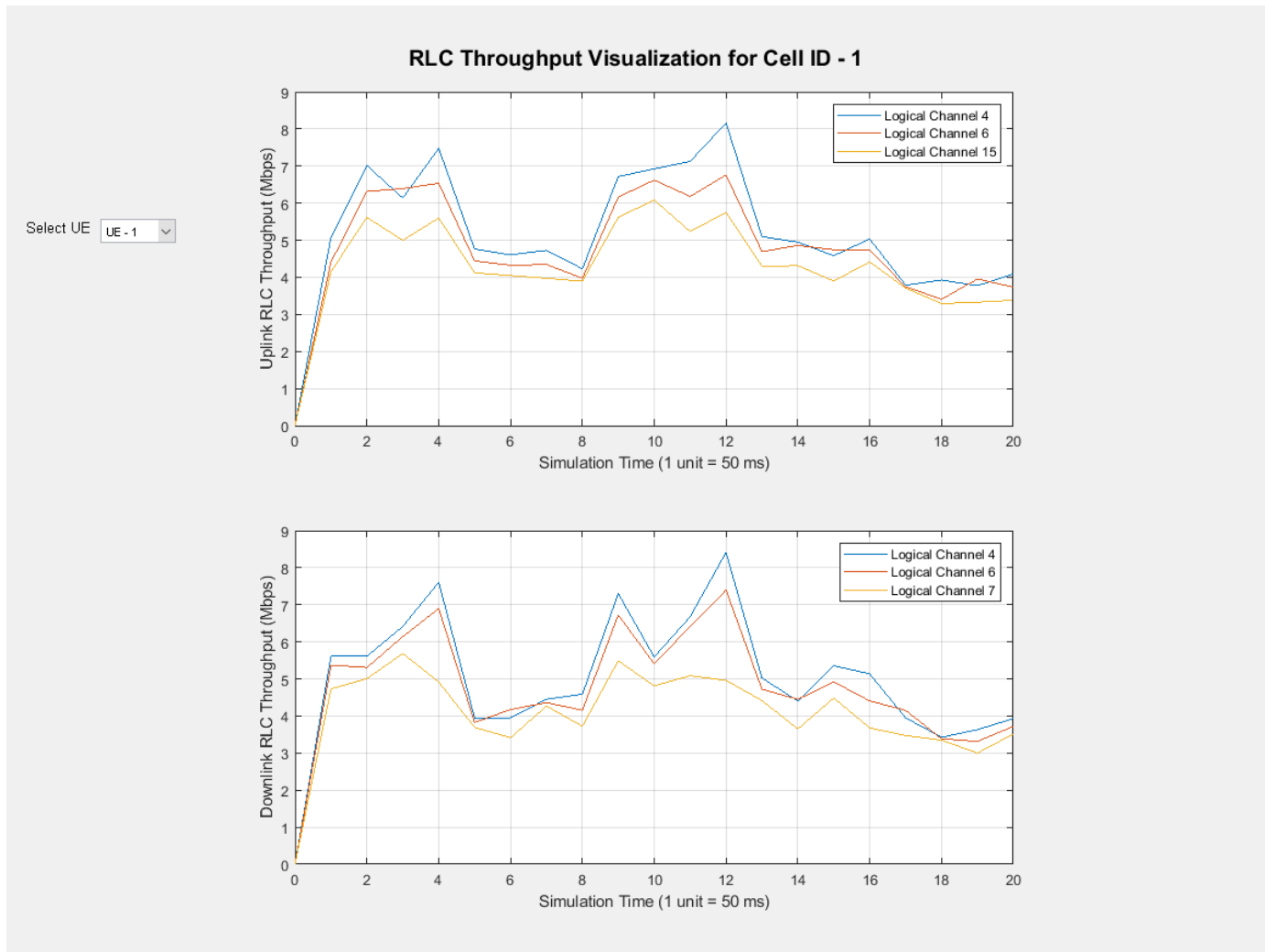
**Downlink Scheduler Performance Metrics for Cell ID - 1**



Uplink Scheduler Performance Metrics for Cell ID - 1







### Simulation Visualization

The five types of run-time visualization shown are:

- *Display of CQI values for UEs over the PUSCH and PDSCH bandwidth:* You can enable this visualization in the Logging and Visualization Configuration on page 5-0 section. For details, see the 'Channel Quality Visualization' figure description in “NR PUSCH FDD Scheduling” on page 5-2 example.
- *Display of resource grid assignment to UEs:* The 2D time-frequency grid shows the resource allocation to the UEs. You can enable this visualization in the Logging and Visualization Configuration on page 5-0 section. For details, see the 'Resource Grid Allocation' figure in “NR PUSCH FDD Scheduling” on page 5-2 example.
- *Display of UL scheduling metrics plots:* The four plots displayed in 'Uplink Scheduler Performance Metrics' figure represent: UL throughput (per UE and cell), UL goodput (per UE and cell), resource share percentage among UEs (out of the total UL resources) to convey the fairness of scheduling, and pending UL buffer-status of UEs to show whether UEs are getting sufficient resources. The maximum achievable data rate value for UL throughput is shown with a dashed

line in throughput and goodput plots. The performance metrics plots update for every `metricsStepSize` slots.

- *Display of DL scheduling metrics plots:* Like UL metrics plots, 'Downlink Scheduler Performance Metrics' displays corresponding subplots for DL direction. The performance metrics plots update for every `metricsStepSize` slots.
- *Display of RLC metrics plots:* The 'RLC Throughput Visualization' figure shows the throughput of the RLC logical channel for the selected UE. The RLC metrics plot update for every `metricsStepSize` slots.

## Simulation Logs

The parameters used for simulation and the simulation logs are saved in MAT-files for post simulation analysis and visualization. The simulation parameters are saved in a MAT-file with the file name as the value of configuration parameter `simParameters.ParametersLogFile`. The per time step logs, scheduling assignment logs, and RLC logs are saved in the MAT-file `simParameters.SimulationLogFile`. After the simulation, open the file to load `DLTimeStepLogs`, `ULTimeStepLogs`, `SchedulingAssignmentLogs`, and `RLCLogs` in the workspace.

**Time step logs:** Both the DL and UL time step logs follow the same format. For details of log format, see the 'Simulation Logs' section of “NR PUSCH FDD Scheduling” on page 5-2.

**Scheduling assignment logs:** Information of all the scheduling assignments and the related information is logged in this file. The table shows sample log entries.

'RNTI'	'Frame'	'Slot'	'Grant type'	'RBG Allocation Map'	'Start Sym'	'Num Sym'	'MCS'	'HARQ ID'	'NDI Flag'	'RV'	'Tx Type'	'Feedback Slot Offset (DL grants only)'	'CQI on RBs'
3	0	8	'DL'	'[1 0 0 0 0 0 0 1 1 0 1 1 1]'	0	14	10	2	1	0	'newTx'	2	'[3 7 7 2 2 5 10 4 6 3 8 3 6 7 9 10 6 2...'
4	0	8	'DL'	'[0 1 1 1 1 1 1 0 0 1 0 0 0]'	0	14	9	3	1	0	'newTx'	2	'[4 2 3 7 5 4 9 6 6 10 3 8 8 4 6 1 1 6 ...'
1	0	9	'UL'	'[1 1 1 1 1 1 1 0 0 0 0 0 0]'	0	14	16	0	1	0	'newTx'	'NA'	'[13 14 2 14 10 2 5 9 15 15 3 15 15 8...'
2	0	9	'UL'	'[0 0 0 0 0 0 0 0 1 0 1 1 1]'	0	14	12	0	1	0	'newTx'	'NA'	'[10 9 5 8 3 9 1 4 1 2 10 9 4 12 1 6 5 ...'
3	0	9	'UL'	'[0 0 0 0 0 0 0 1 0 0 0 0 0]'	0	14	15	0	1	0	'newTx'	'NA'	'[3 7 7 2 2 5 10 4 6 3 8 3 6 7 9 10 6 2...'
4	0	9	'UL'	'[0 0 0 0 0 0 0 0 1 0 0 0 0]'	0	14	15	0	1	0	'newTx'	'NA'	'[4 2 3 7 5 4 9 6 6 10 3 8 8 4 6 1 1 6 ...'

**RLC logs:** For more information on the RLC log format, see “NR PUSCH FDD Scheduling” on page 5-2.

You can run the script `NRPostSimVisualization` to get a post simulation visualization of logs. In the post simulation script, you are provided with variable `isLogReplay`, which provides these options to visualize 'Resource Grid Allocation' and 'Channel Quality Visualization' figures.

- Set `isLogReplay` to true for replay of simulation logs.
- Set `isLogReplay` to false to analyze the details of a particular frame. In the 'Resource Grid Allocation' window, input the frame number to visualize the resource assignment for the entire frame. The frame number entered here controls the frame number for the 'Channel Quality Visualization' figure too.

```
% Get the logs
simulationLogs = cell(1,1);
logInfo = struct('DLTimeStepLogs',[], 'ULTimeStepLogs',[], 'SchedulingAssignmentLogs',[], 'RLCLogs',[]);
[logInfo.DLTimeStepLogs, logInfo.ULTimeStepLogs] = getSchedulingLogs(simSchedulingLogger);
logInfo.SchedulingAssignmentLogs = getGrantLogs(simSchedulingLogger); % Scheduling assignments to RLC
logInfo.RLCLogs = getRLCLogs(simRLCLogger);
simulationLogs{1} = logInfo;
save(simParameters.ParametersLogFile, 'simParameters'); % Save simulation parameters in a MAT-file
save(simParameters.SimulationLogFile, 'simulationLogs'); % Save simulation logs in a MAT-file
```

## Further Exploration

You can use this example to further explore these options.

### Custom scheduling

You can modify the existing scheduling strategy to implement a custom one. Perform following steps to accomplish this.

#### Create a custom scheduler class

Create a new class `customStrategy.m` and inherit from `hNRScheduler.m`. Implement the constructor of the class to call the base class constructor as shown below. See the constructor of `hNRSchedulerRoundRobin.m`, `hNRSchedulerProportionalFair.m`, `hNRSchedulerBestCQI.m` for more details.

```
function obj = customStrategy(param)
    obj = obj@hNRScheduler(simParameters); % Invoke the super class constructor to initialize the
    % Initialize any properties specific to this custom scheduling strategy
end
```

#### Implement custom UL scheduling

Override the `scheduleULResourcesSlot` function of the base class by implementing it in the class `customStrategy.m` function.

```
function uplinkGrants = scheduleULResourcesSlot(obj, slotNum)
    % Implement custom UL scheduling to populate the output 'uplinkGrants'
end
```

First input `obj` is the custom scheduler object. By virtue of inheritance, it contains the context of base class `hNRScheduler.m` which can be used as input for taking scheduling decisions. Second input `slotNum` is the slot number in the 10 ms frame whose UL resources are getting scheduled. Note that it is not the current slot in which UL scheduler is running but the slot which is getting scheduled. Using the context in `obj`, do the custom scheduling and populate the output `uplinkGrants` of this function with an array of valid UL scheduling assignments. `uplinkGrants` can have maximum one assignment per UE. Each element in this array is PUSCH assignment for a UE. Also, update the context of UL HARQ processes as per the HARQ process ID assigned in the UL assignments. See the `scheduleULResourcesSlot` function in `hNRScheduler.m` for more details.

#### Implement custom DL scheduling

Follow the procedure like custom UL scheduling to override the `scheduleDLResourcesSlot` function of the base class by implementing it in the class `customStrategy.m` function.

#### Create and install the custom scheduler

To use the custom scheduler in this example, create its object and install it on the gNB in the gNB and UEs Setup on page 5-0 section.

```
scheduler = customScheduler(simParameters);
addScheduler(gNB, scheduler); % Add scheduler to gNB
```

### **Use 5G Toolbox™ physical layer**

You can also switch from passthrough PHY layer to 5G Toolbox™ physical layer processing by creating PHY objects using `hNRGNBPhy.m` and `hNRUEPhy.m`. For more details, see 'gNB and UEs Setup' section of “NR Cell Performance Evaluation with Physical Layer Integration” on page 5-61.

Based on the chosen scheduling strategy, this example shows the assignment of UL and DL resources to multiple UEs by the gNB. A run-time visualization display of the resource grid gives detailed information about the RBs allocated to each UE and the HARQ process ID assigned for transmission. UL and DL scheduling performance is analyzed based on run-time plots of throughput and goodput, resource share fairness, and pending buffer status on the UEs. A more thorough post simulation analysis by using the saved logs gives a detailed picture of the operations happening on a per slot basis.

### **Use RLC AM**

You can also switch the operating mode of an RLC entity from UM to acknowledged mode (AM) by modifying the input structure fields `EntityType` and `SeqNumFieldLength` in the `configureLogicalChannel` function of `hNRNode.m`. Set the `EntityType` to 3 and `SeqNumFieldLength` to either 12 or 18. You can also add and set the following fields to the input structure to explore the RLC AM functionality:

- `PollRetransmitTimer`: Timer used by the transmitting side of an RLC AM entity in order to retransmit a poll
- `PollPDU`: Parameter used by the transmitting side of an RLC AM entity to trigger a poll based on number of PDUs
- `PollByte`: Parameter used by the transmitting side of an RLC AM entity to trigger a poll based on number of SDU bytes
- `MaxRetransmissions`: Maximum number of retransmissions corresponding to an RLC SDU, including its segments
- `StatusProhibitTimer`: Timer used by the receiving side of an RLC AM entity in order to prohibit frequent transmission of status PDUs

### **Appendix**

The example uses these helper functions and classes:

- `hNRNode.m`: NR node base class for both gNB and UE
- `hNRGNB.m`: gNB node functionality
- `hNRUE.m`: UE node functionality
- `hNRRLCEntity.m`: Base class for RLC UM and AM entities
- `hNRUMEntity.m`: RLC UM functionality
- `hNRAMEntity.m`: RLC AM functionality
- `hNRRLCDataPDUInfo.m`: Creates RLC PDU information object
- `hNRRLCBufferStatus.m`: Generates RLC buffer status information object
- `hNRRLCDataReassembly.m`: Create an RLC SDU reassembly information object
- `hNRMAC.m`: NR MAC base class functionality
- `hNRGNBMAC.m`: gNB MAC functionality

- hNRUEMAC.m: UE MAC functionality
- hNRScheduler.m: Core MAC scheduler functionality
- hNRSchedulerBestCQI.m: Implements best CQI scheduling strategy
- hNRSchedulerProportionalFair.m: Implements proportional fair scheduling strategy
- hNRSchedulerRoundRobin.m: Implements round robin scheduling strategy
- hNRMACBSR.m: Generates buffer status report
- hNRMACBSRParser.m: Parses buffer status report
- hNRMACSubPDU.m: Generates MAC subPDU
- hNRMACPaddingSubPDU.m: Generates MAC subPDU with padding
- hNRMACMultiplex.m: Generates MAC PDU
- hNRMACPDUParser.m: Parses MAC PDU
- hNewHARQProcesses.m: Creates new HARQ process
- hUpdateHARQProcess.m: Updates HARQ process
- hNRPhyInterface.m: PHY layer interface class
- hNRGNBPassthroughPhy.m: gNB passthrough PHY layer
- hNRUEPassthroughPhy.m: UE passthrough PHY layer
- hNRGNBPhy.m: gNB PHY functionality
- hNRUEPhy.m: UE PHY functionality
- hNRPUSCHInfo.m: PUSCH information structure passed by MAC to PHY layer
- hNRPDSCHInfo.m: PDSCH information structure passed by MAC to PHY layer
- hNRRxIndicationInfo.m: Information structure passed by PHY layer to MAC along with MAC PDU
- hNRUplinkGrantFormat.m: UL grant format
- hNRDownlinkGrantFormat.m: DL grant format
- hNRPacketDistribution.m: Creates packet distribution object
- hNRPhyRxBuffer.m: Creates PHY signal reception buffer object
- hSkipWeakTimingOffset.m: Skip timing offset estimates with weak correlation
- hNRRLCLogger.m: Implements RLC statistics logging and visualization functionality
- hNRSchedulingLogger.m: Implements scheduling information logging and visualization functionality
- hNRPhyLogger.m: Implements uplink and downlink block error rate logging and visualization functionality
- hNRSchedulingFDDValidateConfig.m: Validates simulation configuration
- hNRSetUpPacketDistribution.m: Set up packet distribution functionality
- hNRPacketWriter.m: Captures MAC packets
- hNRPacketInfo.m: Metadata format for capturing MAC packets
- NRPostSimVisualization.m: Post simulation visualization script

## References

- [1] 3GPP TS 38.104. "NR; Base Station (BS) radio transmission and reception." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.214. "NR; Physical layer procedures for data." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network.*

[3] 3GPP TS 38.321. "NR; Medium Access Control (MAC) protocol specification." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network.*

[4] 3GPP TS 38.322. "NR; Radio Link Control (RLC) protocol specification." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network.*

[5] 3GPP TS 38.331. "NR; Radio Resource Control (RRC) protocol specification." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network.*

### **See Also**

### **Related Examples**

- "NR PUSCH FDD Scheduling" on page 5-2
- "NR TDD Symbol Based Scheduling Performance Evaluation" on page 5-39

## NR TDD Symbol Based Scheduling Performance Evaluation

This example shows how to implement a symbol based scheduling scheme in time division duplexing (TDD) mode to assign uplink (UL) and downlink (DL) resources and evaluates the network performance. Symbol based scheduling of resources allows shorter transmission durations spanning a few symbols in the slot. You can also switch to slot based scheduling. In TDD mode, physical uplink shared channel (PUSCH) and physical downlink shared channel (PDSCH) transmissions are scheduled in the same frequency band with separation in time domain. Scheduling strategy used in the example can be customized to evaluate the performance. A passthrough physical (PHY) layer without any physical layer processing is used, which adopts a probability-based approach to model packet reception failures. The performance of the scheduling strategy is evaluated in terms of achieved throughput and the fairness in resource sharing.

### Introduction

The example considers the following operations within the gNB and UEs that facilitate UL and DL transmissions and receptions.

	Operations
gNB	<ul style="list-style-type: none"> <li>• Run the scheduling algorithm to assign uplink and downlink resources</li> <li>• Send the uplink and downlink assignments to the UEs</li> <li>• Receive the PUSCH transmissions from the UEs</li> <li>• Adhere to the downlink assignments for PDSCH transmission</li> <li>• Receive the feedback of PDSCHs from the UEs</li> </ul>
UEs	<ul style="list-style-type: none"> <li>• Send the pending buffer status report to the gNB</li> <li>• Receive the uplink and downlink assignments from the gNB</li> <li>• Adhere to the received uplink assignments from the gNB for PUSCH transmission</li> <li>• Receive the PDSCH transmissions from the gNB</li> <li>• Send feedback for the received PDSCHs</li> </ul>

The complete PUSCH or PDSCH packet is transmitted in the first symbol of its allocated symbol set. Receiver processes the packet in the symbol just after the last symbol in the allocated symbol set.

This example models:

- Configurable TDD DL-UL pattern.
- Slot based and symbol based DL and UL scheduling. UL scheduler ensures that the UEs get the required PUSCH preparation time.
- Noncontiguous allocation of frequency domain resources in terms of resource block groups (RBGs).
- Configurable subcarrier spacing resulting in different slot durations.
- Asynchronous hybrid automatic repeat request (HARQ) mechanism.
- Periodic DL and UL application traffic pattern.
- RLC operating in UM mode.
- Single bandwidth part covering the entire carrier bandwidth.

Following control packets are assumed to be sent out of band i.e. without the need of resources for transmission and assured error-free reception: UL assignment, DL assignment, buffer status report (BSR), PDSCH feedback, and CQI report. These control packets follow the TDD DL and UL timings. For example, BSR and PDSCH feedback are sent in UL time while resource assignments are sent in DL time.

### TDD DL-UL Pattern Configuration

NR provides a flexible way of configuring the DL and UL resources. The parameters used to define a custom TDD configuration are:

- 1 DL-UL transmission periodicity in ms.
- 2 Reference subcarrier spacing to calculate the number of slots in the DL-UL pattern. In this example, it is assumed to be same as actual subcarrier spacing used for transmission.
- 3 Number of consecutive full DL slots at the beginning of each DL-UL pattern.
- 4 Number of consecutive DL symbols in the beginning of the slot following the last full DL slot.
- 5 Number of consecutive full UL slots at the end of each DL-UL pattern.
- 6 Number of consecutive UL symbols in the end of the slot preceding the first full UL slot.

The example does not model flexible symbols, so the symbols with unspecified symbol type are assumed for guard period. Here is an example of resulting TDD DL-UL pattern based on these parameters. This DL-UL pattern repeats itself in the timeline.

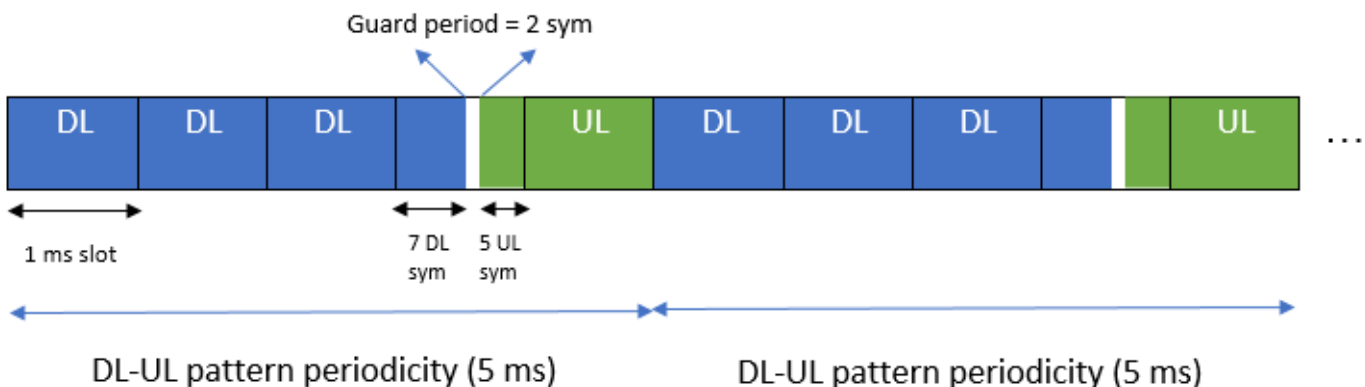
- reference\_scs = 15 kHz (i.e. 1 ms slot), DLULPeriodicity = 5 ms, numDLSlots = 3, numDLSyms = 7, numULSlots = 1, numULSyms = 5

Number of slots in DL-UL periodicity with respect to reference SCS of 15 kHz,  
 NumSlotsDLULPeriodicity = 5

NumberOfGuardSymbols = TotalSymbolsInPattern - TotalSymbolsWithTypeSpecified

$$= (14 * \text{NumSlotsDLULPeriodicity}) - (\text{numDLSlots} * 14 + \text{numDLSyms} + \text{numULSyms} + \text{numULSlots} * 14)$$

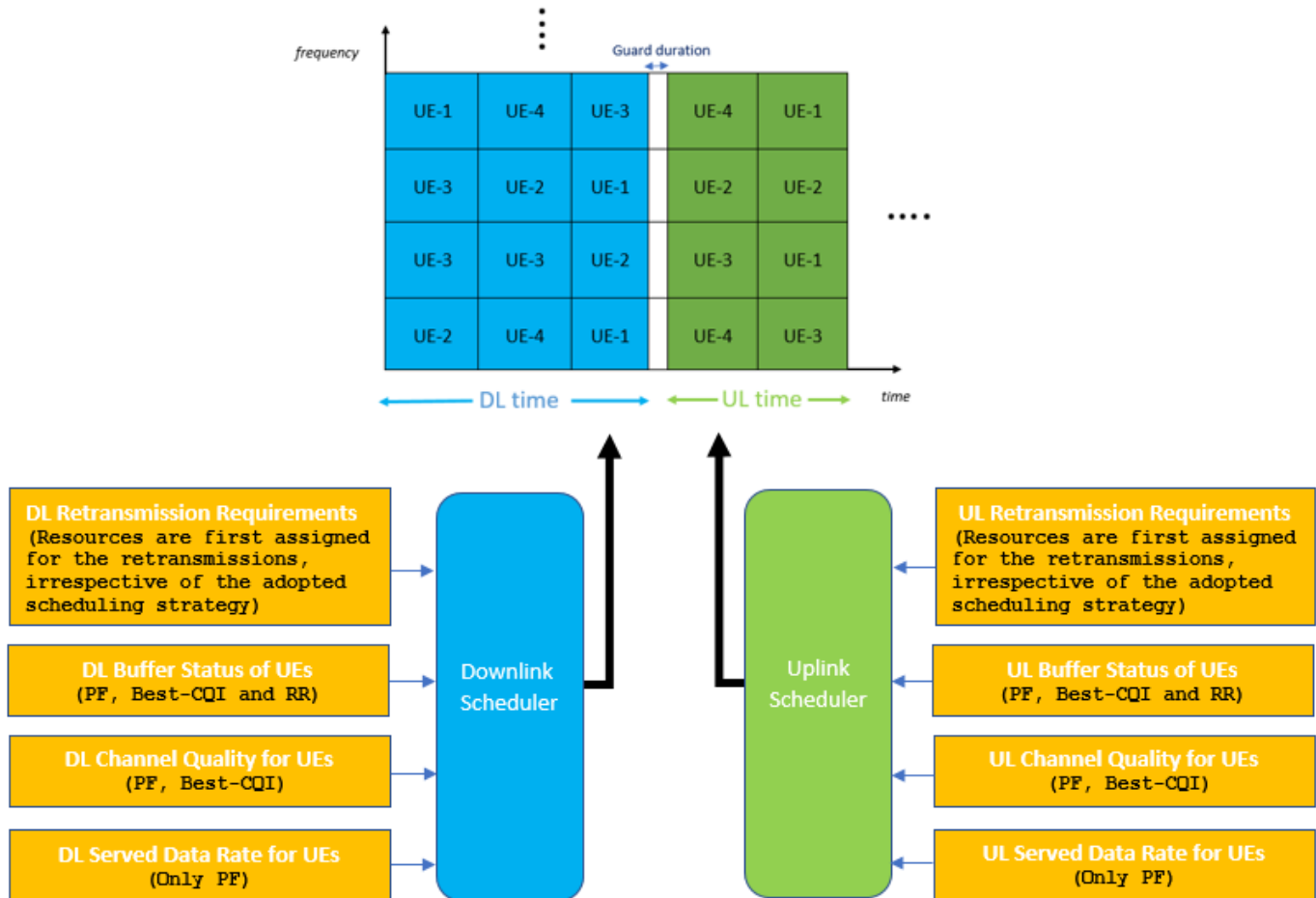
$$= 2 \text{ symbols}$$





## Scheduler

UL and DL schedulers distribute the UL and DL resources respectively among the UEs. You can choose any one of the implemented scheduling strategies: proportional fair (PF), best CQI, or round robin (RR). The various supported inputs to the schedulers are shown along with the scheduling strategies that consider them.

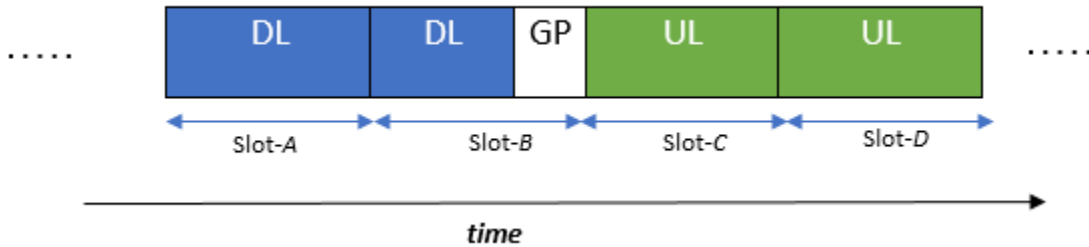


Both UL and DL schedulers run at the start of a slot when the first symbol is a DL symbol. Schedulers run in DL time so that assignments (UL and DL) can be instantly sent to UEs in the DL direction. The run time of the scheduler algorithm as well as the propagation delay is assumed to be zero. The output of scheduling operations is an array of assignments. Each assignment contains the information fields to fully define a PUSCH or PDSCH transmission.

### • UL Scheduler

The UL scheduling operation follows these two steps.

- 1 *Select slots to be scheduled:* The criteria used in this example selects all the upcoming slots (including the current one) containing unscheduled UL symbols that must be scheduled now. Such slots must be scheduled now as they cannot be scheduled in the next slot with DL symbols, depending on the value of PUSCH preparation time capability of the UEs. It ensures that the UL resources are scheduled as close as possible to the actual transmission time.



Below are 2 examples to explain how UL slots are selected in this example for scheduling, based on PUSCH preparation time.

(i) Assuming that the UEs require PUSCH preparation time equivalent to 10 symbols, when the UL scheduler runs in Slot-A, it does not select any slot for scheduling. Because scheduling in the next slot (i.e. Slot-B) provides enough PUSCH preparation time (14 symbols) for the UL transmission in Slot-C. Later, when the UL scheduler runs in Slot-B, it selects both Slot-C and Slot-D for scheduling. Slot-D is scheduled in Slot-B itself (and not in Slot-C) because Slot-C is a full UL slot and hence does not have any DL symbols for sending the assignments in the DL direction.

(ii) Assuming that the UEs require PUSCH preparation time equivalent to 16 symbols, Slot-C is scheduled in Slot-A itself. Because scheduling in Slot-B would only provide 14 symbols of PUSCH preparation time to start transmission in Slot-C. Slot-D is scheduled in Slot-B.

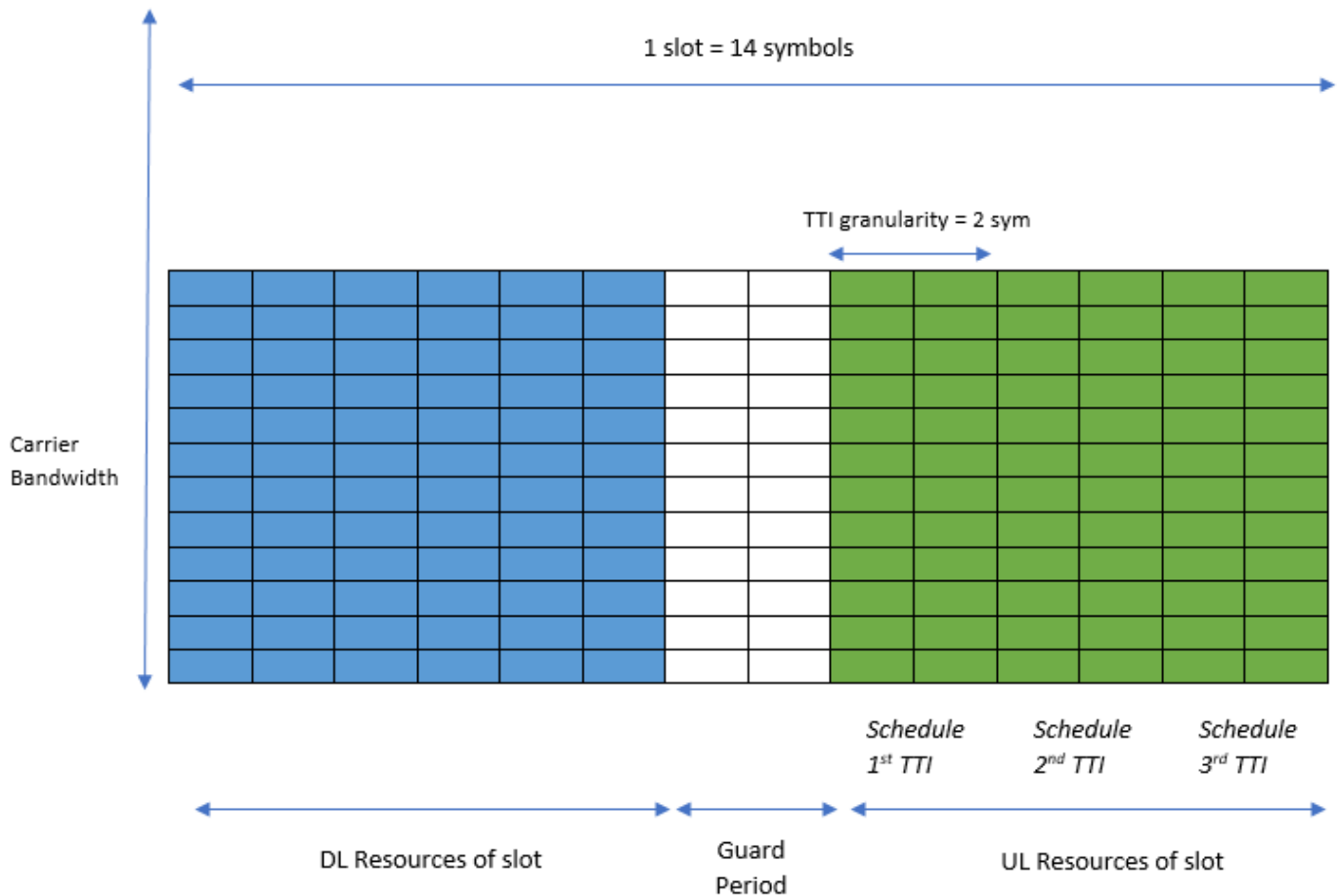
2. *Resource scheduling*: If any slots are selected in the first step, assign the UL resources of those slots to the UEs.

- **DL scheduler**

The DL scheduler runs at each slot beginning with a DL symbol and assign resources of the first upcoming slot containing DL symbols. So, when the DL scheduler runs at the start of Slot-A, it schedules DL resources of Slot-B.

### Symbol based scheduling

NR allows the TTI to start at any symbol position in the slot and with TTI granularity in symbols. The figure shows the way UL scheduler operates in this example to schedule UL symbols of a slot with TTI granularity of two symbols. The slot shown contains six UL symbols. Scheduler completes the iteration of the UL symbols in three iterations with each iteration distributing the frequency resources of two symbols. The DL scheduler also follows a similar approach for DL scheduling.



### Scenario Configuration

Set the parameters for simulation.

```
rng('default'); % Reset the random number generator
simParameters = []; % Clear simParameters variable
simParameters.NumFramesSim = 100; % Simulation time in terms of number of 10 ms frames

% Number of UEs in the simulation. UEs are assumed to have sequential radio
% network temporary identifiers (RNTIs) from 1 to NumUEs. If you change the
% number of UEs, ensure that the following simulation parameters are array
% of length equal to NumUEs: simParameters.UEDistance,
% simParameters.ULPacketPeriodicityUEs, simParameters.ULPacketSizesUEs,
% simParameters.DLPacketPeriodicityUEs, simParameters.DLPacketSizesUEs
simParameters.NumUEs = 4;
simParameters.UEDistance = [100; 150; 300; 400]; % Distance of UEs from gNB (in meters)

% Set the value of scheduling type to 0 (slot based scheduling) or 1
% (symbol based scheduling). If value is not set, the default value is 0
simParameters.SchedulingType = 1;

% Set the minimum time-domain assignment in terms of symbol duration.
% Applicable only for symbol based scheduling. For slot based scheduling,
```

```

% it is always full slot (14 symbols)
simParameters.TTIGranularity = 4;

% Define the TDD DL-UL pattern. The reference subcarrier spacing used for
% calculating slot duration for the pattern is assumed to be same as actual
% subcarrier spacing used for transmission as defined by simParameters.SCS.
% Keep only the symbols intended for guard period during DLULPeriodicity
% with type (DL or UL) unspecified
simParameters.DLULPeriodicity = 5; % Duration of the DL-UL pattern in ms
simParameters.NumDLSlots = 2; % Number of consecutive full DL slots at the beginning of each DL-UL
% Define the number of consecutive DL symbols following the full DL slots
% and the number of consecutive UL symbols preceding the full UL slots. In
% the case of slot based scheduling, only simParameters.NumDLSyms is
% applicable and it defines the number of DL symbols at the start of the
% slot, while the remaining symbols of the slot are assumed to be allotted
% for guard period
simParameters.NumDLSyms = 8; % Number of consecutive DL symbols in the beginning of the slot following
simParameters.NumULSyms = 4; % Number of consecutive UL symbols in the end of the slot preceding
simParameters.NumULSlots = 2; % Number of consecutive full UL slots at the end of each DL-UL pattern

% Mapping between distance from gNB (first column in meters) and maximum
% achievable UL CQI value (second column). For example, if a UE is 700
% meters away from the gNB, it can achieve a maximum CQI value of 8 as the
% distance falls within the [501, 1000] meters range, as per the mapping.
% Set the distance in increasing order and the maximum achievable CQI value
% in decreasing order
simParameters.CQIvsDistance = [
    200 15;
    300 12;
    500 10;
    1000 8;
    1200 7];

% Set the periodic UL and DL application traffic pattern for UEs
simParameters.ULPacketPeriodicityUEs = [20; 30; 20; 30]; % Periodicity (in ms) at which the UL packets are generated
simParameters.ULPacketSizesUEs = [4000; 6000; 5500; 4000]; % Size of the UL packets (in bytes) generated
simParameters.DLPacketPeriodicityUEs = [20; 15; 15; 20]; % Periodicity (in ms) at which the DL packets are generated
simParameters.DLPacketSizesUEs = [6000; 5000; 10000; 8000]; % Size of the DL packets generated (in bytes)

% Medium access control (MAC) configuration
simParameters.SchedulerStrategy = 'PF'; % Supported scheduling strategies: 'PF', 'RR' and 'BestEffort'
% PUSCH preparation time. gNB ensures that PUSCH assignment is received at
% UEs PUSCHPrepTime ahead of the transmission time
simParameters.PUSCHPrepTime = 200; % In microseconds
% Maximum RBs allotted to a UE for PUSCH and PDSCH transmissions (limit is
% applicable for new transmission assignments and not for the retransmissions)
simParameters.RBAllocationLimitUL = 15; % For PUSCH
simParameters.RBAllocationLimitDL = 15; % For PDSCH
simParameters.BSRPeriodicity = 1; % Buffer status report transmission periodicity (in ms)

% PHY layer and channel configuration
% RB count for 5 MHz band with 15 kHz subcarrier spacing (SCS). The complete
% bandwidth is assumed to be allotted for PUSCH or PDSCH
simParameters.NumRBs = 25;
simParameters.SCS = 15; % kHz
simParameters.DLBandwidth = 5e6; % Hz
simParameters.ULBandwidth = 5e6; % Hz
simParameters.DLCarrierFreq = 2.595e9; % Hz

```

```

simParameters.ULCarrierFreq = 2.595e9; % Hz

% Configure parameters to update channel conditions for the UEs. Channel
% quality is periodically improved or deteriorated by CQIDelta every
% channelUpdatePeriodicity seconds for all RBs of a UE. Whether channel
% conditions for a particular UE improve or deteriorate is randomly
% determined: RB_CQI = RB_CQI +/- CQIDelta
simParameters.ChannelUpdatePeriodicity = 0.2; % In sec
simParameters.CQIDelta = 1;

% Logging and visualization configuration
% Flag to enable or disable runtime CQI visualization
simParameters.CQIVisualization = true;
% Flag to enable or disable run time visualization of RB assignment. If enabled,
% then for slot based scheduling it updates every frame (10 ms) to show RB
% allocation to the UEs for different slots of the last frame. For
% symbol based scheduling, it updates every slot to show RB allocation to
% the UEs over different symbols of the last slot
simParameters.RBVisualization = false;
% The output metrics plots are updated periodically NumMetricsSteps times within the
% simulation duration
simParameters.NumMetricsSteps = 20;
% MAT-files to write the logs into. They are used for post simulation analysis and visualization
simParameters.ParametersLogFile = 'simParameters'; % For logging the simulation parameters
simParameters.SimulationLogFile = 'simulationLogs'; % For logging the simulation logs

hNRSchedulingTDDValidateConfig(simParameters); % Validate the simulation configuration

```

### Derived Parameters

Based on the primary configuration parameters, compute the derived parameters. Additionally, set some example specific constants.

```

simParameters.DuplexMode = 1; % TDD (example specific constant)
simParameters.NCellID = 1; % Physical cell ID
simParameters.GNBPosition = [0 0 0]; % Position of gNB in (x,y,z) coordinates
simParameters.NumLogicalChannels = 1; % Only 1 logical channel is assumed in each UE (Example sp
slotDuration = 1/(simParameters.SCS/15); % Slot duration in ms
numSlotsFrame = 10/slotDuration; % Number of slots in 10 ms frame
numSlotsSim = simParameters.NumFramesSim * numSlotsFrame; % Simulation time in units of slot dura
numSymbolsSim = numSlotsSim * 14; % Simulation time in units of symbol duration
% Interval at which metrics visualization updates in terms of number of
% slots. Make sure that MetricsStepSize is an integer
simParameters.MetricsStepSize = ceil(numSlotsSim / simParameters.NumMetricsSteps);
if mod(numSlotsSim, simParameters.NumMetricsSteps) ~= 0
    % Update the NumMetricsSteps parameter if numSlotsSim is not
    % completely divisible by it
    simParameters.NumMetricsSteps = floor(numSlotsSim / simParameters.MetricsStepSize);
end

% DL and UL packet periodicities for UEs in terms of number of slots
appPeriodicityUEsSlotsUL = simParameters.ULPacketPeriodicityUEs ./ slotDuration;
appPeriodicityUEsSlotsDL = simParameters.DLPacketPeriodicityUEs ./ slotDuration;

% Mapping between logical channel and logical channel group identifier (LCGID)
simParameters.LCHConfig.LCGID = ones(simParameters.NumUEs, simParameters.NumLogicalChannels);
% Priority of each logical channel
simParameters.LCHConfig.Priority = ones(simParameters.NumUEs, simParameters.NumLogicalChannels);

```

```

% Prioritized bitrate (PBR) of each logical channel (in kilo bytes per second)
simParameters.LCHConfig.PBR = 8*ones(simParameters.NumUEs, simParameters.NumLogicalChannels);
% Bucket size duration (BSD) of each logical channel (in ms). However, the priority,
% PBR and BSD for logical channel is not relevant with single logical
% channel assumed in this example
simParameters.LCHConfig.BSD = 10*ones(simParameters.NumUEs, simParameters.NumLogicalChannels);
% Logical channel id (logical channel ID of data radio bearers starts from 4)
simParameters.LCHConfig.LCID = ones(simParameters.NumUEs, simParameters.NumLogicalChannels) .* (4

% RLC entity direction. Value 0 represents DL only, 1
% represents UL only and 2 represents both UL and DL
% directions. Setting entity direction to have both UL and DL
simParameters.RLCCConfig.EntityDir = 2*ones(simParameters.NumUEs, simParameters.NumLogicalChannels);
% Construct information for RLC logger
lchInfo = repmat(struct('LCID',[],'EntityDir',[']), [simParameters.NumUEs 1]);
for idx = 1:simParameters.NumUEs
    lchInfo(idx).LCID = simParameters.LCHConfig.LCID(idx, :);
    lchInfo(idx).EntityDir = simParameters.RLCCConfig.EntityDir(idx, :);
end

% Maximum RLC SDU length (in bytes) as per 3GPP TS 38.323
simParameters.maxRLCSDULength = 9000;

% Calculate maximum achievable CQI value for the UEs based on their distance
% from the gNB
maxUECQIs = zeros(simParameters.NumUEs, 1); % To store the maximum achievable CQI value for UEs
for ueIdx = 1:simParameters.NumUEs
    % Based on the distance of the UE from gNB, find matching row in CQIvsDistance mapping
    matchingRowIndex = find(simParameters.CQIvsDistance(:, 1) > simParameters.UEDistance(ueIdx));
    if isempty(matchingRowIndex)
        maxUECQIs(ueIdx) = simParameters.CQIvsDistance(end, 2);
    else
        maxUECQIs(ueIdx) = simParameters.CQIvsDistance(matchingRowIndex(1), 2);
    end
end

% Define initial UL and DL channel quality as an N-by-P matrix,
% where 'N' is the number of UEs and 'P' is the number of RBs in the carrier
% bandwidth. The initial value of CQI for each RB, for each UE, is given
% randomly and is limited by the maximum achievable CQI value corresponding
% to the distance of the UE from gNB
simParameters.InitialChannelQualityUL = zeros(simParameters.NumUEs, simParameters.NumRBs); % To
simParameters.InitialChannelQualityDL = zeros(simParameters.NumUEs, simParameters.NumRBs); % To
for ueIdx = 1:simParameters.NumUEs
    % Assign random CQI values for the RBs, limited by the maximum achievable CQI value
    simParameters.InitialChannelQualityUL(ueIdx, :) = randi([1 maxUECQIs(ueIdx)], 1, simParameters.NumRBs);
    % Initially, DL and UL CQI values are assumed to be equal
    simParameters.InitialChannelQualityDL(ueIdx, :) = simParameters.InitialChannelQualityUL(ueIdx, :);
end

% Update periodicity of RB assignment visualization in terms of number of slots
if ~isfield(simParameters, 'SchedulingType') || simParameters.SchedulingType == 0 % If no scheduling
    rbAssignmentPlotPeriodicity = numSlotsFrame; % Update every frame (10 ms)
else % Symbol based scheduling
    rbAssignmentPlotPeriodicity = 1; % Update every slot
end

```

## gNB and UEs Setup

Create the gNB and UE objects, initialize the channel quality information for UEs and set up the logical channel at gNB and UE. The helper classes hNRGNB.m and hNRUE.m create gNB and UE node respectively, containing the RLC and MAC layer. For MAC layer, hNRGNB.m uses the helper class hNRGNBMAC.m to implement the gNB MAC functionality and hNRUE.m uses hNRUEMAC.m to implement the UE MAC functionality. Schedulers are implemented in hNRSchedulerRoundRobin.m (RR), hNRSchedulerProportionalFair.m (PF), hNRSchedulerBestCQI.m (Best CQI) . All the schedulers inherit from the base class hNRScheduler.m which contains the core scheduling functionality. For RLC layer, both hNRGNB.m and hNRUE.m use hNRUMTransmitter.m to implement the functionality of the RLC transmitter and hNRUMReceiver.m to implement the functionality of the RLC receiver. Passthrough PHY layer for UE and gNB is implemented in hNRUEPassThroughPhy.m and hNRGNBPassThroughPhy.m, respectively.

```

simParameters.Position = simParameters.GNBPosition;
gNB = hNRGNB(simParameters); % Create the gNB node
% Create and add scheduler
switch(simParameters.SchedulerStrategy)
    case 'RR' % Round robin scheduler
        scheduler = hNRSchedulerRoundRobin(simParameters);
    case 'PF' % Proportional fair scheduler
        scheduler = hNRSchedulerProportionalFair(simParameters);
    case 'BestCQI' % Best CQI scheduler
        scheduler = hNRSchedulerBestCQI(simParameters);
end
addScheduler(gNB, scheduler); % Add scheduler to gNB

gNB.PhyEntity = hNRGNBPassThroughPhy(simParameters); % Add passthrough PHY
configurePhy(gNB, simParameters);
setPhyInterface(gNB); % Set the interface to PHY layer

% Create the set of UE nodes
UEs = cell(simParameters.NumUEs, 1);
for ueIdx = 1:simParameters.NumUEs
    simParameters.Position = [simParameters.UEDistance(ueIdx) 0 0]; % Position of UE
    UEs{ueIdx} = hNRUE(simParameters, ueIdx);
    UEs{ueIdx}.PhyEntity = hNRUEPassThroughPhy(simParameters, ueIdx); % Add passthrough PHY
    configurePhy(UEs{ueIdx}, simParameters);
    setPhyInterface(UEs{ueIdx}); % Set the interface to PHY layer

    % Initialize the UL CQI values at gNB
    updateChannelQuality(gNB, simParameters.InitialChannelQualityUL(ueIdx, :), 1, ueIdx); % 1 for
    % Initialize the DL CQI values at gNB and UE. The DL CQI values
    % help gNB in scheduling, and UE in packet error probability estimation
    updateChannelQuality(gNB, simParameters.InitialChannelQualityDL(ueIdx, :), 0, ueIdx); % 0 for
    updateChannelQuality(UEs{ueIdx}, simParameters.InitialChannelQualityDL(ueIdx, :));

    % Setup the logical channel and add application traffic
    for lcIdx = 1:simParameters.NumLogicalChannels
        % Create RLC channel configuration structure
        rlcChannelConfigStruct.EntityType = simParameters.RLCCConfig.EntityDir(ueIdx, lcIdx);
        rlcChannelConfigStruct.LogicalChannelID = simParameters.LCHConfig.LCID(ueIdx, lcIdx);
        rlcChannelConfigStruct.LCGID = simParameters.LCHConfig.LCGID(ueIdx, lcIdx);
        rlcChannelConfigStruct.Priority = simParameters.LCHConfig.Priority(ueIdx, lcIdx);
        rlcChannelConfigStruct.PBR = simParameters.LCHConfig.PBR(ueIdx, lcIdx);
        rlcChannelConfigStruct.BSD = simParameters.LCHConfig.BSD(ueIdx, lcIdx);
        % Setup logical channel at gNB for the UE

```

```

configureLogicalChannel(gNB, ueIdx, rlcChannelConfigStruct);
% Setup logical channel at UE
configureLogicalChannel(UEs{ueIdx}, ueIdx, rlcChannelConfigStruct);

% Add data traffic pattern generators to gNB and UE nodes
ulPacketSize = simParameters.ULPacketSizesUEs(ueIdx);
% Calculate the data rate (in kbps) of On-Off traffic pattern using
% packet size (in bytes) and packet interval (in ms)
ulDataRate = ceil(1000/simParameters.ULPacketPeriodicityUEs(ueIdx)) * ulPacketSize * 8e-3;
% Limit the size of the generated application packet to the maximum RLC
% SDU size. The maximum supported RLC SDU size is 9000 bytes
if ulPacketSize > simParameters.maxRLCSDULength
    ulPacketSize = simParameters.maxRLCSDULength;
end
% Create an object for On-Off network traffic pattern and add it to the
% specified UE. This object generates the uplink (UL) data traffic on the UE
ulApp = networkTrafficOnOff('PacketSize', ulPacketSize, 'GeneratePacket', true, ...
    'OnTime', simParameters.NumFramesSim/100, 'OffTime', 0, 'DataRate', ulDataRate);
UEs{ueIdx}.addApplication(ueIdx, simParameters.LCHConfig.LCID(ueIdx, lcIdx), ulApp);

dlPacketSize = simParameters.DLPacketSizesUEs(ueIdx);
dlDataRate = ceil(1000/simParameters.DLPacketPeriodicityUEs(ueIdx)) * dlPacketSize * 8e-3;
if dlPacketSize > simParameters.maxRLCSDULength
    dlPacketSize = simParameters.maxRLCSDULength;
end
% Create an object for On-Off network traffic pattern for the specified
% UE and add it to the gNB. This object generates the downlink (DL) data
% traffic on the gNB for the UE
dlApp = networkTrafficOnOff('PacketSize', dlPacketSize, 'GeneratePacket', true, ...
    'OnTime', simParameters.NumFramesSim/100, 'OffTime', 0, 'DataRate', dlDataRate);
gNB.addApplication(ueIdx, simParameters.LCHConfig.LCID(ueIdx, lcIdx), dlApp);
end
end

% Setup the UL and DL packet distribution mechanism
simParameters.MaxReceivers = simParameters.NumUEs;
% Create DL packet distribution object
dlPacketDistributionObj = hNRPacketDistribution(simParameters, 0); % 0 for DL
% Create UL packet distribution object
ulPacketDistributionObj = hNRPacketDistribution(simParameters, 1); % 1 for UL
hNRSetUpPacketDistribution(simParameters, gNB, UEs, dlPacketDistributionObj, ulPacketDistributionObj);

```

### Processing Loop

Simulation is run symbol by symbol to execute corresponding operations. If slot based scheduling is chosen, the execution jumps from the current slot boundary to the next slot boundary. The operations executed are:

- Run the MAC and PHY layers of gNB
- Run the MAC and PHY layers of UEs
- Layer specific logging and visualization
- Advance the timer for the nodes. Every 1 ms it also sends trigger to application and RLC layers. Application layer and RLC layer execute their scheduled operations based on 1 ms timer trigger.

```

% To store the following UE metrics for each symbol: throughput bytes
% transmitted, goodput bytes transmitted, and pending buffer amount bytes

```



```

% (for logging purpose). The number of goodput bytes is calculated by
% excluding the retransmissions from the total transmissions. The complete
% transmission is assumed to be done in the first symbol of transmission.
% Metrics are for DL direction if symbol type is DL (likewise, for UL
% symbol)
UEMetrics = zeros(simParameters.NumUEs, 3);

% To store current CQI values on the RBs for the UEs for logging
channelQuality = zeros(simParameters.NumUEs, simParameters.NumRBs);

% To store the current value of last received NDI flag for HARQ processes
% of UEs for logging. NDI values are for DL HARQ processes, if symbol type is DL
% (likewise, for UL symbol)
HARQProcessStatus = zeros(simParameters.NumUEs, 16);

% To store the RLC statistics of a slot for logging
ueRLCStats = cell(simParameters.NumUEs, 1);
gnBRLCStats = cell(simParameters.NumUEs, 1);

% Create logging and visualization objects for MAC scheduling and RLC
simSchedulingLogger = hNRSchedulingLogger(simParameters);
simRLCLogger = hNRRLCLogger(simParameters, lchInfo);
if ~isfield(simParameters, 'SchedulingType') || simParameters.SchedulingType == 0 % If no scheduling
    tickGranularity = 14; % Operations are only at slot boundary. Simulation jumps from one slot
else % Symbol based scheduling
    tickGranularity = 1; % Operations can be at symbol boundary. Simulation moves symbol-by-symbol
end
end

slotNum = 0;
% Execute all the symbols in the simulation
for symbolNum = 1 : tickGranularity : numSymbolsSim
    UEMetrics(:) = 0;
    HARQProcessStatus(:) = 0;
    symbolType = currentSymbolType(gNB); % Get current symbol type: DL/UL/Guard
    if mod(symbolNum - 1, 14) == 0
        slotNum = slotNum + 1;
    end

    % Run MAC and PHY layers of gNB
    run(gNB.MACEntity);
    run(gNB.PhyEntity);

    % Run MAC and PHY layers of UEs
    for ueIdx = 1:simParameters.NumUEs
        % Read the last received NDI flags for HARQ processes for
        % logging (Reading it before it gets overwritten by run function of MAC)
        if symbolType == 0 % DL
            HARQProcessStatus(ueIdx, :) = getLastNDIFlagHarq(UEs{ueIdx}.MACEntity, 0); % 0 for DL
        else
            HARQProcessStatus(ueIdx, :) = getLastNDIFlagHarq(UEs{ueIdx}.MACEntity, 1); % 1 for UL
        end
        run(UEs{ueIdx}.MACEntity);
        run(UEs{ueIdx}.PhyEntity);
    end

    % RLC logging (only at slot boundary)
    if mod(symbolNum - 1, 14) == 0 %
        for ueIdx = 1:simParameters.NumUEs % For all UEs

```

```

        % Get RLC statistics
        ueRLCStats{ueIdx} = getRLCStatistics(UEs{ueIdx}, ueIdx);
        gNBRLCStats{ueIdx} = getRLCStatistics(gNB, ueIdx);
    end
    logRLCStats(simRLCLogger, ueRLCStats, gNBRLCStats); % Update RLC statistics logs
end

% MAC logging
% Read UL and DL assignments done by gNB MAC scheduler
% at current time. Resource assignments returned by a scheduler (either
% UL or DL) is empty, if either scheduler was not scheduled to run at
% the current time or no resources got scheduled
[resourceAssignmentsUL, resourceAssignmentsDL] = getCurrentSchedulingAssignments(gNB.MACEnti
% Read throughput and goodput bytes sent for each UE
if symbolType == 0 % DL
    [UEMetrics(:, 1), UEMetrics(:, 2)] = getTTIBytes(gNB);
    UEMetrics(:, 3) = getBufferStatus(gNB); % Read pending buffer (in bytes) on gNB, for all
end
for ueIdx = 1:simParameters.NumUEs
    if symbolType == 0 % DL
        % Read the DL channel quality for the UE
        channelQuality(ueIdx,:) = getChannelQuality(gNB, 0, ueIdx); % 0 for DL
    else
        % Read the UL channel quality for the UE
        channelQuality(ueIdx,:) = getChannelQuality(gNB, 1, ueIdx); % 1 for UL
        % Read throughput and goodput bytes transmitted by the UE in the current TTI
        [UEMetrics(ueIdx, 1), UEMetrics(ueIdx, 2)] = getTTIBytes(UEs{ueIdx});
        UEMetrics(ueIdx, 3) = getBufferStatus(UEs{ueIdx}); % Read pending buffer (in bytes)
    end
end
% Update logs based on the current symbol run of UEs and gNB
logScheduling(simSchedulingLogger, symbolNum, [resourceAssignmentsUL resourceAssignmentsDL],

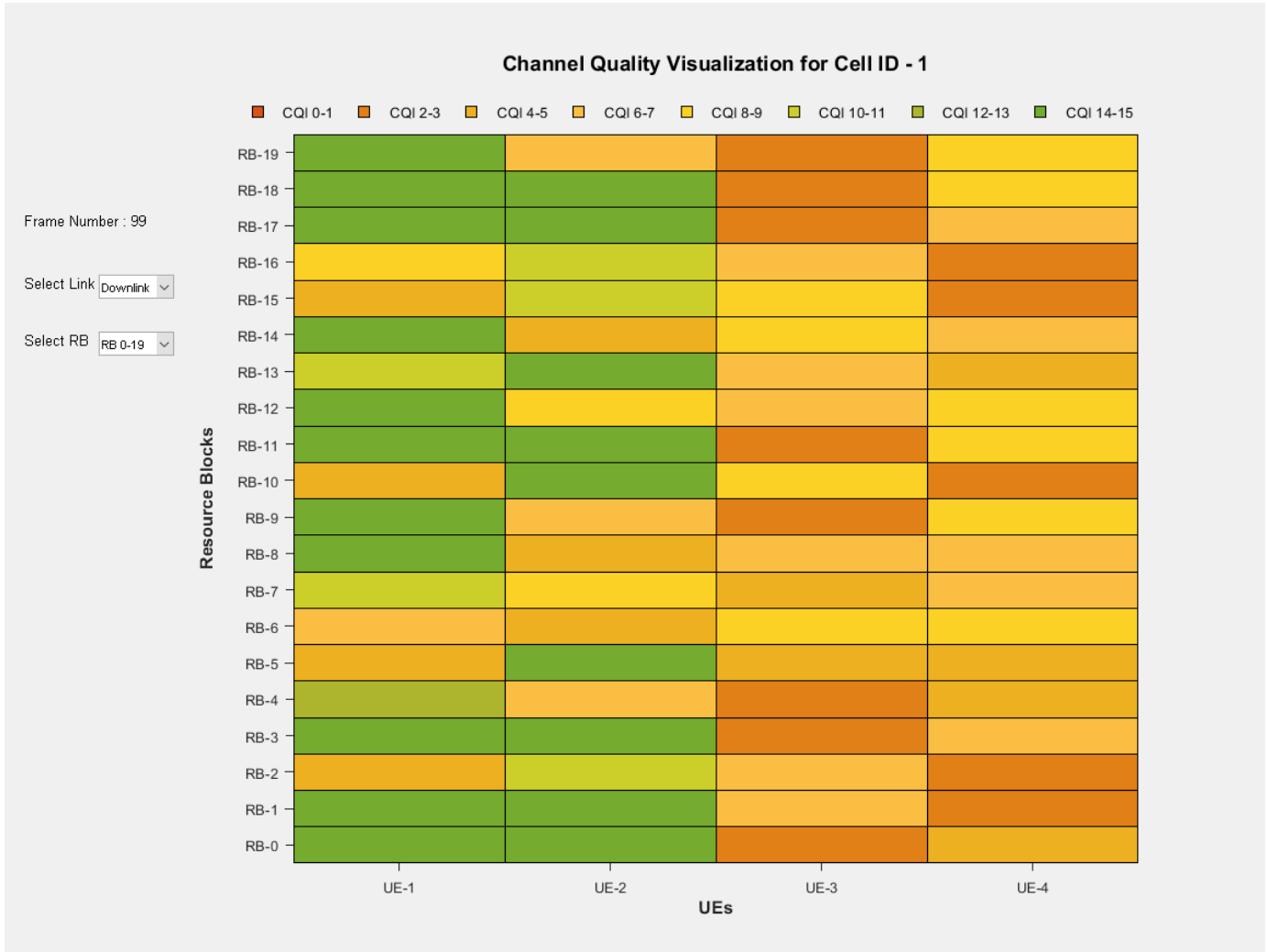
% Visualization
% RB assignment visualization (if enabled)
if simParameters.RBVisualization
    if symbolNum > 1 && (simParameters.SchedulingType == 1 && (mod(symbolNum, 14) == 0)) ||
        % Plot at slot boundary, if the update periodicity is
        % reached
        plotRBGrids(simSchedulingLogger);
    end
end
% CQI grid visualization (if enabled)
if simParameters.CQIVisualization
    if symbolNum > 1 && mod(symbolNum-1, 14) == 0 && mod(slotNum, numSlotsFrame) == 0 % Plot
        plotCQIRBGrids(simSchedulingLogger);
    end
end
% Plot scheduler metrics and RLC metrics visualization at slot
% boundary, if the update periodicity is reached
if symbolNum > 1 && mod(symbolNum-1, 14) == 0 && mod(slotNum, simParameters.MetricsStepSize)
    plotMetrics(simSchedulingLogger);
    plotMetrics(simRLCLogger);
end

% Advance timer ticks for gNB and UEs by 'tickGranularity' symbols
advanceTimer(gNB, tickGranularity);
for ueIdx = 1:simParameters.NumUEs % For all UEs

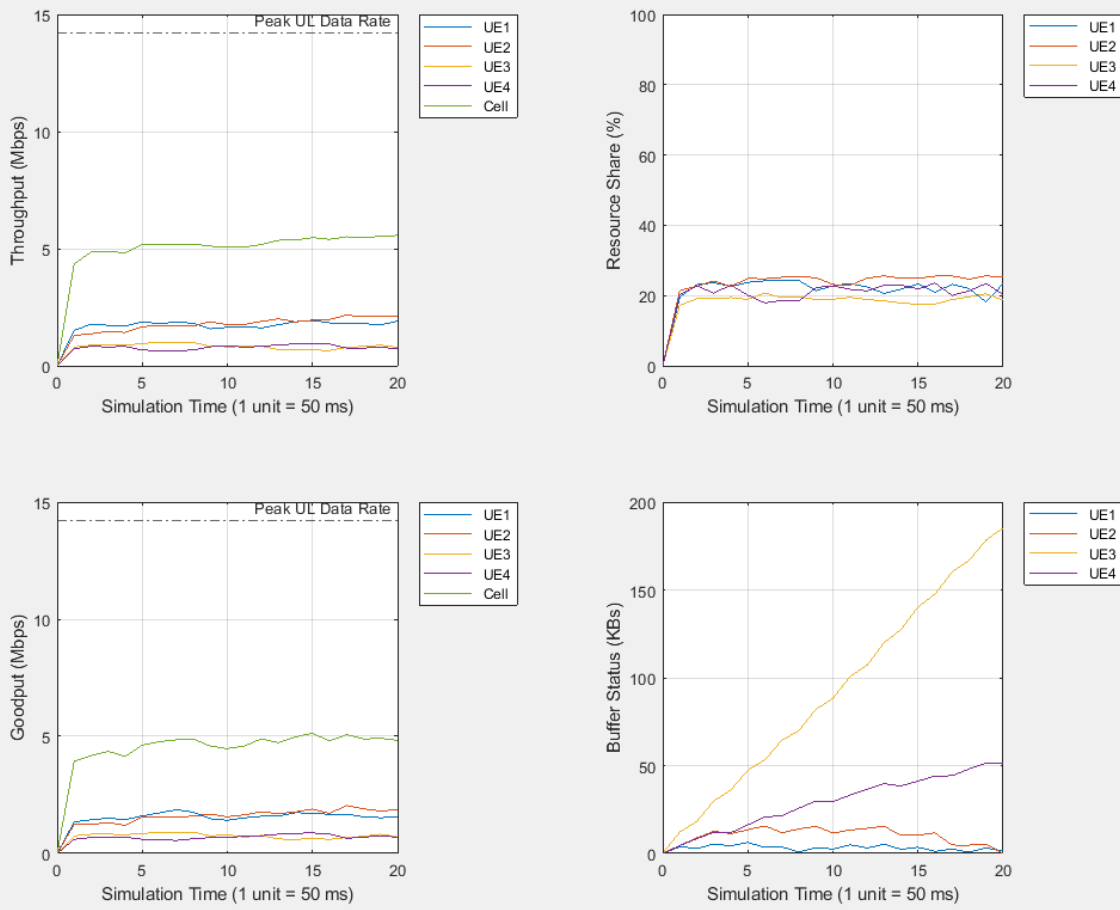
```

```

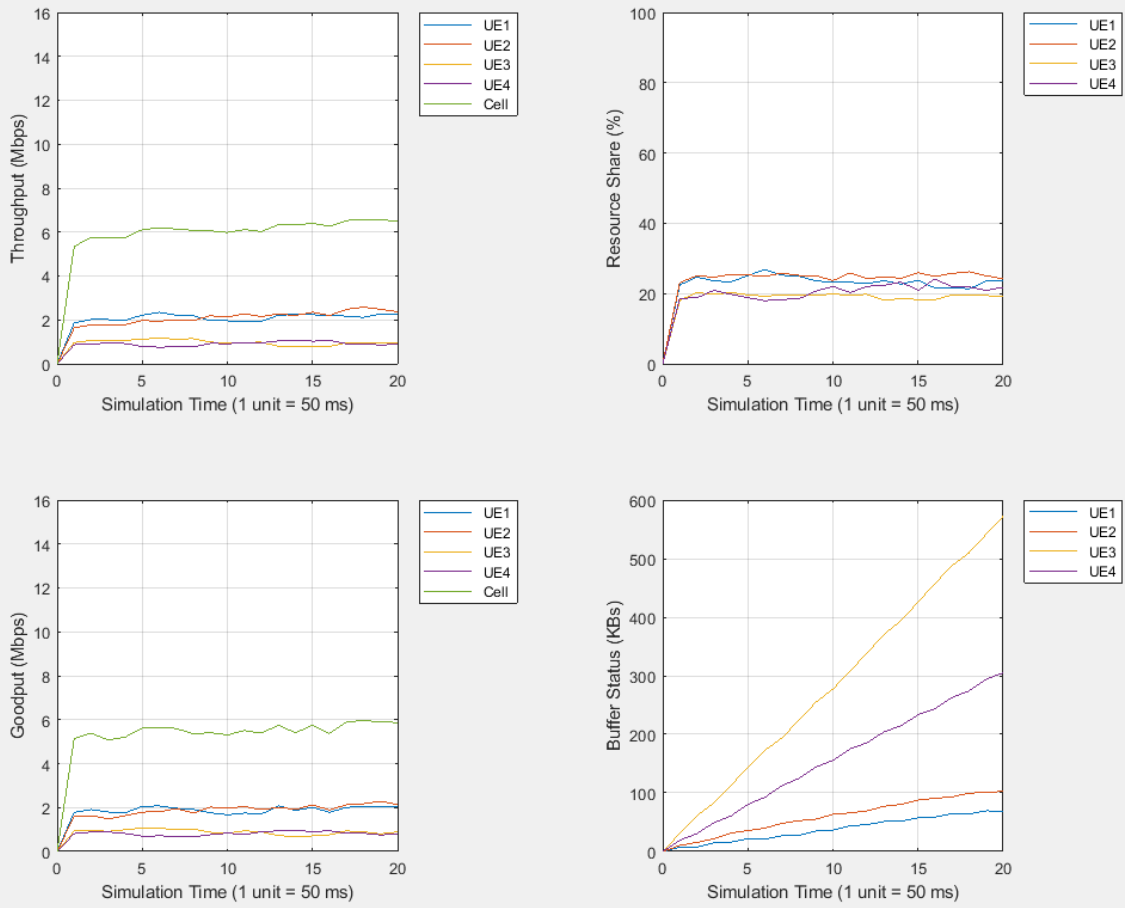
advanceTimer(UEs{ueIdx}, tickGranularity);
end
end
    
```

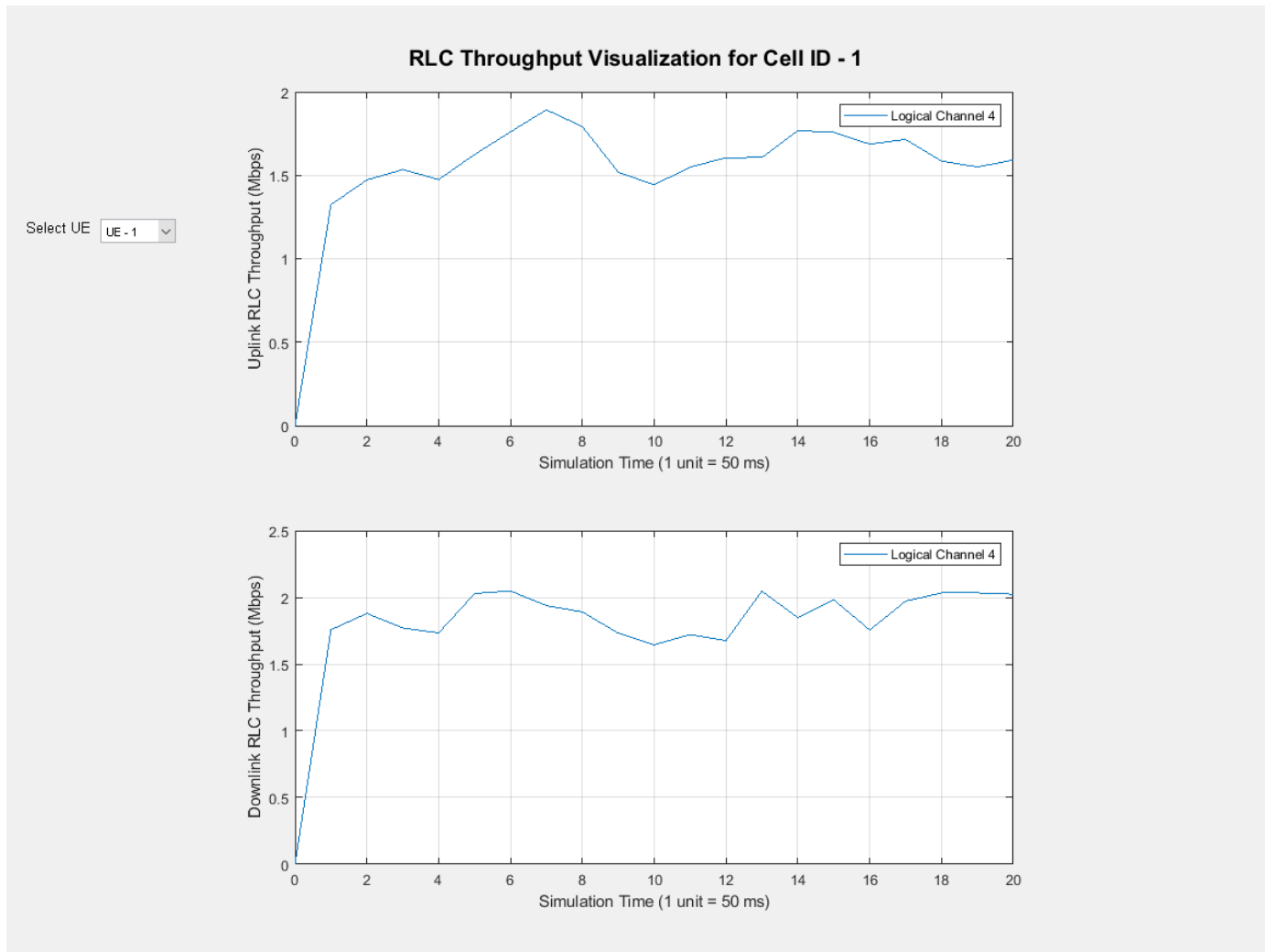


Uplink Scheduler Performance Metrics for Cell ID - 1



**Downlink Scheduler Performance Metrics for Cell ID - 1**



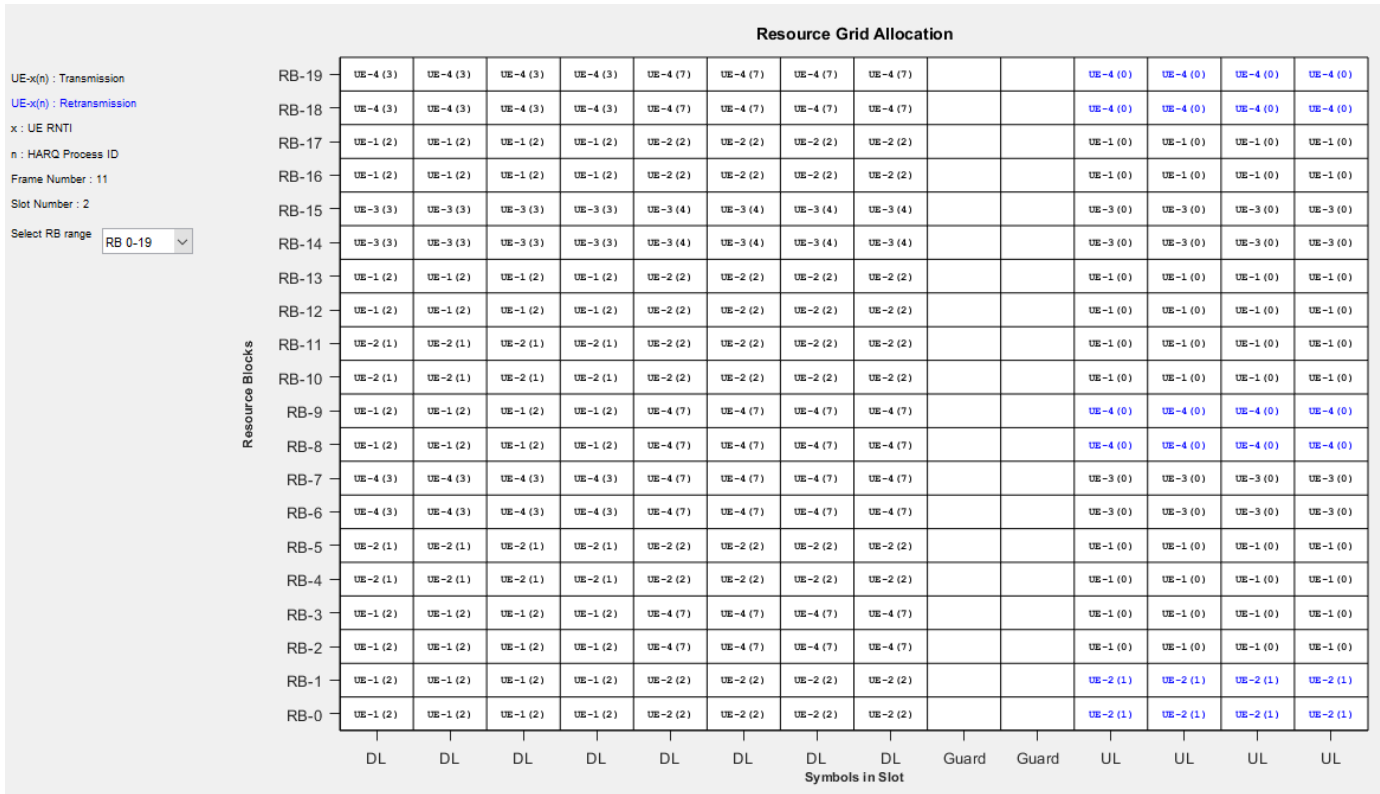


### Simulation Visualization

The five types of runtime visualization shown are:

- *Display of CQI values for UEs over the PUSCH or PDSCH bandwidth:* For details, see the 'Channel Quality Visualization' figure.
- *Display of resource grid assignment to UEs:* The 2-D time-frequency grid shows the resource allocation to the UEs. For slot based scheduling, it updates every 10 ms (frame length), and shows the RB allocation to the UEs in the previous frame. For symbol based scheduling, it updates every slot, and shows the RB allocation of symbols of the previous slot. For details, see the 'Resource Grid Allocation' figure.
- *Display of UL scheduling metrics plots:* The 'Uplink Scheduler Performance Metrics' figure includes plots of the: UL throughput (per UE and cell), UL goodput (per UE and cell), resource share percentage among UEs (out of the total UL resources) to convey the fairness of scheduling, and pending UL buffer status of the UEs to show whether UEs are getting sufficient resources. The maximum achievable data rate value for UL throughput is `metricsStepSize` slots. shown with a dashed line in throughput and goodput plots. The performance metrics plots update every

- *Display of DL scheduling metrics plots:* Like uplink metrics plots, the 'Downlink Scheduler Performance Metrics' displays corresponding subplots for DL direction. The performance metrics plots update every metricsStepSize slots
- *Display of RLC metrics plot:* The 'RLC Throughput Visualization' figure shows the throughput of RLC logical channel for the selected UE. The RLC metrics plot updates every metricsStepSize slots.



### Simulation Logs

The parameters used for simulation and the simulation logs are saved in MAT-files for post simulation analysis and visualization. The simulation parameters are saved in MAT-file with filename as the value of configuration parameter `simParameters.ParametersLogFile`. The per time step logs, scheduling assignment logs, and RLC logs are saved in the MAT-file `simParameters.SimulationLogFile`. After the simulation, open it to load `TimeStepLogs`, `SchedulingAssignmentLogs`, and `RLCLogs` in the workspace.

**Time step logs:** The table shows a sample time step entry. Each row of the table represents a symbol or a slot, based on the chosen scheduling type (symbol based or slot based). The information in a row is for DL, if the type of symbol (or slot) is DL. Likewise, for UL symbol (or slot).

Frame	Slot	Symbol	Type	RBG Allocation Bitmap	MCS	HARQ Process	NDI	Tx Type	CQI for UEs	HARQ NDI Status	Throughput Bytes	Goodput Bytes	Buffer Status of UEs
5	0	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
5	1	4	UL	[0 0 1 1 0 1 0 1 0 1 0 0 0] [1 1 0 0 0 0 0 0 0 1 0 0] [0 0 0 0 1 0 1 0 1 0 0 1 1] [0 0 0 0 0 0 0 0 0 0 0 0 0]	[10 12 8 -1]	[0 3 6 -1]	[0 0 1 -1]	['newTx', 'newTx', 'reTx', 'noTx']	[5 8 9 8 7 8 9 8 9 11 12 .....] [5 7 3 4 9 5 9 7 9 10 15 .....] [3 9 9 6 9 4 9 8 9 9 13 .....] [9 7 7 3 7 8 5 8 8 15 9 .....]	[1 0 0 1 0 1 0 . .] [1 1 0 1 0 1 0 . .] [1 0 0 1 0 1 1 . .] [1 0 0 1 0 1 0 . .]	[46 54 38 0]	[46 54 0 0]	[299480 135671 77567 137070]
5	2	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....

Each row contains the following information:

- *Frame*: Frame number.
- *Slot*: Slot number in the frame.
- *Symbol*: Symbol number in the slot (Only for symbol based scheduling).
- *Type*: Symbol (or Slot) type as 'DL', 'UL', or 'Guard'. For slot based scheduling, type can only be DL/UL. As the slot containing the guard symbols is assumed to be a DL slot with guard symbols at the end of the slot.
- *RBG Allocation Bitmap*:  $N$ -by- $P$  bitmap matrix, where  $N$  is the number of UEs and  $P$  is the number of RBGs in the bandwidth. If an RBG is assigned to a particular UE, the corresponding bit is set to 1. For example, [0 0 1 1 0 1 0 1 0 1 0 0 0; 1 1 0 0 0 0 0 0 0 0 1 0 0; 0 0 0 0 1 0 1 0 1 0 0 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0] means, that the bandwidth has 13 RBGs and UE-1 is assigned the RBG indices 2, 3, 5, 7, and 9; UE-2 is assigned the RBG indices: 0, 1, and 10; UE-3 is assigned the RBG indices: 4, 6, 8, 11, and 12; and UE-4 is not assigned any RBG.
- *MCS*: Row vector of length  $N$  where  $N$  is the number of UEs. Each value corresponds to the modulation and coding scheme (MCS) index for the PUSCH or PDSCH transmission. For example, [10 12 8 -1] means that only UE-1, UE-2, and UE-3 are assigned UL resources (symbol type is 'UL') for this symbol and use MCS values 10, 12, and 8, respectively.
- *HARQ Process*: Row vector of length  $N$ , where  $N$  is the number of UEs. The value is the HARQ process ID used by UE for the PUSCH transmission or used by gNB for PDSCH transmission. For example, [0 3 6 -1] means that only UE-1, UE-2, and UE-3 are assigned UL resources (symbol type is 'UL') for this symbol and use the HARQ process IDs 0, 3, and 6, respectively.
- *NDI*: Row vector of length  $N$ , where  $N$  is the number of UEs. The value is the NDI flag value in the assignment for PUSCH or PDSCH transmission. For example, [0 0 1 -1] means that only UE-1, UE-2, and UE-3 are assigned UL resources for this symbol and use the NDI flag values (which determine whether a new transmission or a retransmission is done) are 0, 0, and 1, respectively.
- *Tx Type*: Tx Type specifies the transmission type (new transmission or retransmission). Row vector of length  $N$ , where  $N$  is the number of UEs. Possible values are either 'newTx', 'reTx', or 'noTx'. 'noTx' means that the UE is not allotted PUSCH or PDSCH resources. For example: ['newTx' 'newTx' 'reTx' 'noTx'] means that only UE-1, UE-2, and UE-3 are assigned UL resources for this symbol. UE-1 and UE-2 transmit a new packet from the specified HARQ process, while UE-3 retransmits the packet in the buffer of the specified HARQ process.
- *CQI for UEs*:  $N$ -by- $P$  matrix, where  $N$  is the number of UEs and  $P$  is the number of RBs in the bandwidth. A matrix element at position  $(i, j)$  corresponds to the CQI value for UE with RNTI  $i$  at RB  $j$ .
- *HARQ NDI Status*:  $N$ -by- $P$  matrix, where  $N$  is the number of UEs and  $P$  is the number of HARQ processes. A matrix element at position  $(i, j)$  is the last received NDI flag at UE  $i$  for DL or UL HARQ process ID  $j$ . For new transmissions, this value and the NDI flag in the PUSCH or PDSCH assignment must toggle for the HARQ process in the assignment.



- *Throughput Bytes*: Row vector of length  $N$ , where  $N$  is the number of UEs. The values represent UL or DL MAC bytes transmitted by or for the UEs in this symbol. Note that the total throughput bytes for the complete PUSCH or PDSCH transmission are shown in the row corresponding to the first symbol of the transmission.
- *Goodput Bytes*: Row vector of length  $N$ , where  $N$  is the number of UEs. The values represent new UL or DL transmission MAC bytes transmitted by or for the UEs in this symbol. Like throughput, all the goodput bytes for the complete PUSCH or PDSCH are shown in the row corresponding to first symbol of the transmission.
- *Buffer Status of UEs*: Row vector of length  $N$ , where  $N$  is the number of UEs. The values represent the amount of UL direction pending buffers at UEs (or DL direction pending buffers for UEs at gNB).

**Scheduling assignment logs:** Information of all the scheduling assignments and related information is logged in this table. Each row is one UL or DL assignment. For details of log format, see the 'Simulation Logs' section of “NR FDD Scheduling Performance Evaluation” on page 5-21 example.

**RLC logs:** For more information on the RLC log format, see “NR PUSCH FDD Scheduling” on page 5-2.

You can run the script NRPostSimVisualization to get a post simulation visualization of logs. In the post simulation script, you are provided with variable `isLogReplay`, which provides these options to visualize 'Resource Grid Allocation' and 'Channel Quality Visualization' figures.

- Set `isLogReplay` to `true` for a replay of the simulation logs.
- Set `isLogReplay` to `false` to analyze the details of a particular frame or a particular slot of a frame. In the 'Resource Grid Allocation' window, input the frame number and slot number to visualize the resource assignment of the particular slot, if scheduling type is symbol based. For slot based scheduling, enter the frame number to visualize the resource assignment for the entire frame. The frame number entered here controls the frame number for 'Channel Quality Visualization' figure too.

```
% Read the logs and write them to MAT-files
% Get the logs
simulationLogs = cell(1,1);
logInfo = struct('TimeStepLogs',[], 'SchedulingAssignmentLogs',[] , 'RLCLogs',[]);
[logInfo.TimeStepLogs] = getSchedulingLogs(simSchedulingLogger);
logInfo.SchedulingAssignmentLogs = getGrantLogs(simSchedulingLogger); % Scheduling assignments l
logInfo.RLCLogs = getRLCLogs(simRLCLogger); % RLC statistics logs
simulationLogs{1} = logInfo;

save(simParameters.SimulationLogFile, 'simulationLogs'); % Save simulation logs in a MAT-file
save(simParameters.ParametersLogFile, 'simParameters'); % Save simulation parameters in a MAT-fi
```

### Further Exploration

You can use this example to further explore these options.

### Custom scheduling

You can modify the existing scheduling strategy to implement a custom one. Refer to 'Further Exploration' section of “NR FDD Scheduling Performance Evaluation” on page 5-21 example to see the steps involved.

### **Use 5G Toolbox™ physical layer**

You can also switch from passthrough PHY layer to 5G Toolbox™ physical layer processing by creating PHY objects using `hNRGNBPhy.m` and `hNRUEPhy.m`. For more details, see 'gNB and UEs Setup' section of “NR Cell Performance Evaluation with Physical Layer Integration” on page 5-61. Set the simulation parameter `simParameters.SchedulingType` to 0 as `hNRGNBPhy.m` and `hNRUEPhy.m` only support slot based scheduling.

Based on the chosen scheduling strategy, this example demonstrates the assignment of UL and DL resources to multiple UEs by the gNB. UL and DL scheduling performance is analyzed based on runtime plots of throughput, goodput, resource share fairness, and pending buffer status of the UEs. A more thorough post simulation analysis by using the saved logs gives a detailed picture of the operations happening on a per symbol or per slot basis.

### **Use RLC AM**

You can also switch the operating mode of an RLC entity from UM to acknowledged mode (AM) by modifying the input structure fields `EntityType` and `SeqNumFieldLength` in the `configureLogicalChannel` function of `hNRNode.m`. For more details, see 'Further Exploration' section of “NR FDD Scheduling Performance Evaluation” on page 5-21.

### **Appendix**

The example uses these helper functions and classes:

- `hNRNode.m`: NR node base class
- `hNRGNB.m`: gNB node functionality
- `hNRUE.m`: UE node functionality
- `hNRRLCEntity.m`: Base class for RLC UM and AM entities
- `hNRUMEntity.m`: RLC UM functionality
- `hNRAMEntity.m`: RLC AM functionality
- `hNRRLCDataPDUInfo.m`: Creates RLC PDU information object
- `hNRRLCBufferStatus.m`: Generates RLC buffer status information object
- `hNRRLCDataReassembly.m`: Create an RLC SDU reassembly information object
- `hNRMAC.m`: NR MAC base class functionality
- `hNRGNBMAC.m`: gNB MAC functionality
- `hNRUEMAC.m`: UE MAC functionality
- `hNRScheduler.m`: Core MAC scheduler functionality
- `hNRSchedulerBestCQI.m`: Implements best CQI scheduling strategy
- `hNRSchedulerProportionalFair.m`: Implements proportional fair scheduling strategy
- `hNRSchedulerRoundRobin.m`: Implements round robin scheduling strategy
- `hNRMACBSR.m`: Generates buffer status report
- `hNRMACBSRParser.m`: Parses buffer status report
- `hNRMACSubPDU.m`: Generates MAC subPDU
- `hNRMACPaddingSubPDU.m`: Generates MAC subPDU with padding
- `hNRMACMultiplex.m`: Generates MAC PDU

- hNRMACPDUParser.m: Parses MAC PDU
- hNewHARQProcesses.m: Creates new HARQ process
- hUpdateHARQProcess.m: Updates HARQ process
- hNRPhyInterface.m: PHY layer interface class
- hNRGNBPassthroughPhy.m: gNB passthrough PHY layer
- hNRUEPassthroughPhy.m: UE passthrough PHY layer
- hNRGNBPhy.m: gNB PHY functionality
- hNRUEPhy.m: UE PHY functionality
- hNRPUSCHInfo.m: PUSCH information structure passed by MAC to PHY layer
- hNRPDSCHInfo.m: PDSCH information structure passed by MAC to PHY layer
- hNRRxIndicationInfo.m: Information structure passed by PHY layer to MAC along with MAC PDU
- hNRUplinkGrantFormat.m: UL assignment format
- hNRDownlinkGrantFormat.m: DL assignment format
- hNRPacketDistribution.m: Creates packet distribution object
- hNRPhyRxBuffer.m: Creates PHY signal reception buffer object
- hSkipWeakTimingOffset.m: Skip timing offset estimates with weak correlation
- hNRRLCLogger.m: Implements RLC statistics logging and visualization functionality
- hNRSchedulingLogger.m: Implements scheduling information logging and visualization functionality
- hNRPhyLogger.m: Implements uplink and downlink block error rate logging and visualization functionality
- hNRSchedulingTDDValidateConfig.m: Validates simulation configuration
- hNRSetUpPacketDistribution.m: Set up packet distribution functionality
- hNRPacketWriter.m: Captures MAC packets
- hNRPacketInfo.m: Metadata format for capturing MAC packets
- NRPostSimVisualization.m: Post simulation visualization script

## References

- [1] 3GPP TS 38.214. "NR; Physical layer procedures for data." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
- [2] 3GPP TS 38.321. "NR; Medium Access Control (MAC) protocol specification." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
- [3] 3GPP TS 38.322. "NR; Radio Link Control (RLC) protocol specification." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
- [4] 3GPP TS 38.331. "NR; Radio Resource Control (RRC) protocol specification." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

## **See Also**

### **Related Examples**

- “NR FDD Scheduling Performance Evaluation” on page 5-21

## NR Cell Performance Evaluation with Physical Layer Integration

This example models a 5G New Radio (NR) cell by integrating the higher layers of the NR stack with 5G Toolbox™ physical layer processing and evaluates the network performance in the presence of channel impairments. The cell has a set of user equipments (UEs) connected to a gNB with radio link control (RLC), medium access control (MAC), and physical (PHY) layers of the NR stack installed on the network nodes. For faster MAC focused simulations you can switch to passthrough PHY layer or you can integrate with a custom PHY layer.

### Introduction

The example considers the following operations within gNB and UEs that facilitate uplink (UL) and downlink (DL) transmissions and receptions.

	<b>Operations</b>
<b>gNB</b>	<ul style="list-style-type: none"> <li>• Run the scheduling algorithm to assign uplink and downlink resources</li> <li>• Send the uplink and downlink assignments to the UEs</li> <li>• Receive the physical uplink shared channel (PUSCH) transmissions from the UEs</li> <li>• Adhere to the downlink assignments for physical downlink shared channel (PDSCH) transmission</li> <li>• Receive the feedback from the PDSCHs from the UEs</li> <li>• Send the channel state information reference signal (CSI-RS) to the UEs</li> </ul>
<b>UEs</b>	<ul style="list-style-type: none"> <li>• Send the pending buffer status report to gNB</li> <li>• Receive the uplink and downlink assignments from the gNB</li> <li>• Adhere to the received uplink assignments from the gNB for PUSCH transmission</li> <li>• Receive the PDSCH transmissions from the gNB</li> <li>• Send feedback for the received PDSCHs</li> <li>• Measure the channel quality on received CSI-RS and report it to gNB</li> </ul>

The complete PUSCH or PDSCH packet is transmitted in the first symbol of its allocated symbol set. Receiver processes the packet in the symbol just after the last symbol in the allocated symbol set.

This example models:

- Slot-based UL and DL scheduling of PUSCH and PDSCH resources, respectively. The time-domain granularity of the UL assignment and DL assignment is one slot.
- Configurable subcarrier spacing resulting in different slot durations.
- Noncontiguous allocation of frequency-domain resources in terms of resource block groups (RBGs).
- Asynchronous adaptive hybrid automatic repeat request (HARQ) mechanism in UL and DL.
- PUSCH demodulation reference signal (DM-RS) and PDSCH DM-RS.
- DL channel quality measurement by UEs based on the CSI-RS received from gNB. By default, the CSI-RS resource element is transmitted in each slot for each resource block (RB) in DL bandwidth for all UEs. The same CSI-RS configuration is applicable to all the UEs. The example does not

model the sounding reference signal (SRS) for measuring UL channel quality. UL channel quality is assumed to be the same as the DL channel quality measured on CSI-RS.

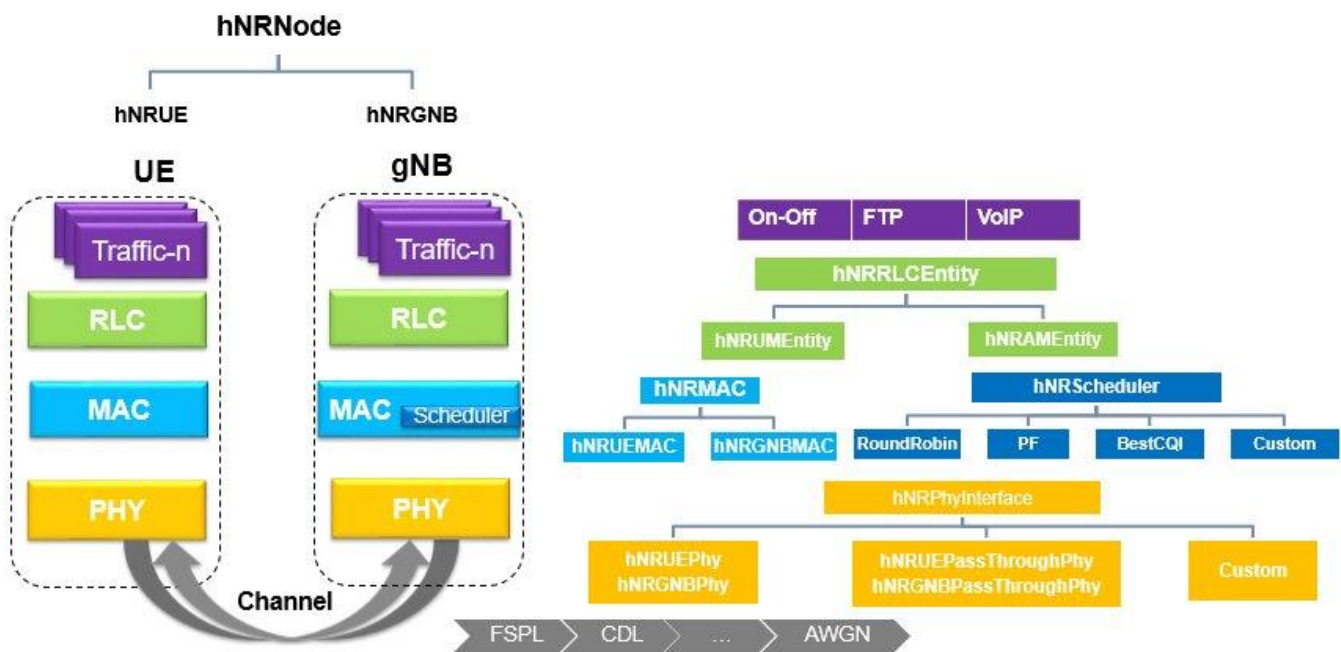
- Free space path loss (FSPL), additive white Gaussian noise (AWGN), and clustered delay line (CDL) propagation channel model.
- Single input single output (SISO) antenna configuration.
- Single bandwidth part across the whole carrier.

Control packets such as UL assignment, DL assignment, buffer status report (BSR), PDSCH feedback, and channel quality indicator (CQI) report, are assumed to be sent out of band, that is, without the need of resources for transmission and assured error-free reception.

**NR Protocol Stack**

A node (gNB or UE) is a composition of NR stack layers. The helper classes hNRGNB.m and hNRUE.m create gNB and UE nodes respectively, containing the RLC, MAC, and PHY layers.

**5G Node Composition**



**RLC Layer**

RLC operates in unacknowledged mode (UM) with a single logical channel (LCH). For the RLC layer, both hNRGNB.m and hNRUE.m use hNRUMEntity.m to implement the functionality of both the RLC transmitter and receiver.

**MAC Layer**

For the MAC layer, hNRGNB.m uses the helper class hNRGNBMAC.m to implement the gNB MAC functionality and hNRUE.m uses hNRUEMAC.m to implement the UE MAC functionality. gNB MAC has UL and DL schedulers that assign UL and DL resources, respectively to the UEs. For more details about UL and DL scheduling to assign the PUSCH and the PDSCH resources, see the “NR FDD

Scheduling Performance Evaluation” on page 5-21 example. Schedulers are implemented in `hNRSchedulerRoundRobin.m` (Round robin strategy), `hNRSchedulerProportionalFair.m` (Proportional fair strategy), and `hNRSchedulerBestCQI.m` (Best CQI strategy) helper classes. All these schedulers are inherited from the base class `hNRScheduler.m`, which contains the core scheduling functionality.

### PHY Layer and Channel Modeling

The example uses 5G Toolbox™ for PHY layer operations of UE and gNB. On the Tx side, the operations involve the physical layer processing of a transport block received from MAC and its transmission. On the Rx side, there is processing of received waveform and sending the decoded information to MAC. For more details on PDSCH and PUSCH processing chains, refer to the “NR PDSCH Throughput” on page 1-56 example and “NR PUSCH Throughput” on page 2-34 example, respectively. For the PHY layer, `hNRGNB.m` uses the helper class `hNRGNBPhy.m` to implement the gNB PHY layer functionality and `hNRUE.m` uses `hNRUEPhy.m` to implement the UE PHY layer functionality. For channel impairments, the example models FSPL, AWGN, and the CDL propagation channel model.

The example uses a lookup table to map the received signal to interference and noise ratio (SINR) to CQI index for 0.1 block error rate (BLER). The lookup table corresponds to the CQI table as per 3GPP TS 38.214 Table 5.2.2.1-3. For more information about the process of generating this lookup table, refer to “5G NR CQI Reporting” on page 4-8 example.

### MAC-PHY Interface

Following are the major interface calls between the MAC layer and PHY layer. For more details, refer to `hNRPhyInterface.m`.

- `txDataRequest`: The request from MAC to PHY to transmit either PDSCH (by gNB) or PUSCH (by UE). MAC calls this request at the start of Tx time. The PHY processing time is not modeled in this example.
- `rxDataRequest`: The request from MAC to PHY to receive either PUSCH (by gNB) or PDSCH (by UE). MAC calls this request at the start of Rx time.
- `dLTTIRequest`: The request from MAC to PHY for non-data downlink transmissions or receptions. For gNB, this request is sent by gNB MAC for DL transmissions. For UE, it is sent by UE MAC for DL receptions. MAC sends the request at the start of a DL slot for all the scheduled DL transmission or receptions in the slot. This interface is used for all the DL transmission and receptions, except PDSCH. `txDataRequest` and `rxDataRequest` are used for PDSCH. In this example, gNB MAC uses this interface to send CSI-RS, and UE MAC uses it to receive CSI-RS.
- `registerMACInterfaceFcn`: The one-time setup call to register MAC callback functions at PHY. PHY uses the callbacks to send information up the stack to MAC. gNB PHY uses the callback to send decoded UL packets to MAC. UE PHY uses the callbacks to send decoded DL packets and DL channel quality measured on CSI-RS to MAC.

### Pluggable PHY

You can plug and use different variations of PHY layer in your system. In a simulation run, all the nodes use the same variation of the PHY layer. The MAC is unaware of the type of PHY layer underneath, because the MAC uses the MAC-PHY interface to interact with the PHY layer. By default the example uses 5G Toolbox™ to model the PHY layer. To use a passthrough PHY layer, refer to “NR PUSCH FDD Scheduling” on page 5-2, “NR FDD Scheduling Performance Evaluation” on page 5-21, and “NR TDD Symbol Based Scheduling Performance Evaluation” on page 5-39 examples. A passthrough PHY layer does not do any physical layer processing of packets.

## Scenario Configuration

Set the parameters for simulation.

```

rng('default'); % Reset the random number generator
simParameters = []; % Clear the simParameters variable

simParameters.NumFramesSim = 30; % Simulation time in terms of number of 10 ms frames
% Number of UEs in the simulation. UEs are assumed to have sequential radio
% network temporary identifiers (RNTIs) from 1 to NumUEs. If you change the
% number of UEs, ensure that the following simulation parameters are array
% of length equal to NumUEs: simParameters.UEDistance,
% simParameters.ULPacketPeriodicityUEs, simParameters.ULPacketSizesUEs,
% simParameters.DLPacketPeriodicityUEs, simParameters.DLPacketSizesUEs
simParameters.NumUEs = 4;
simParameters.UEDistance = [100; 400; 1500; 1000];

% Set the channel bandwidth to 5 MHz and subcarrier spacing (SCS) to 15
% kHz as defined in 3GPP TS 38.104 Section 5.3.2
% RB count for 5 MHz band with 15 kHz SCS. The complete
% bandwidth is assumed to be allotted for PUSCH/PDSCH
simParameters.NumRBs = 25;
simParameters.SCS = 15; % kHz
simParameters.DLCarrierFreq = 2.635e9; % Hz
simParameters.ULCarrierFreq = 2.515e9; % Hz
% The UL and DL carriers are assumed to have symmetric channel
% bandwidth
simParameters.DLBandwidth = 5e6; % Hz
simParameters.ULBandwidth = 5e6; % Hz

simParameters.UETxPower = 23; % Tx power for all the UEs in dBm
simParameters.GNBTxPower = 29; % Tx power for gNB in dBm
simParameters.RxGain = 11; % Receiver antenna gain at gNB

% SINR to CQI mapping table for 0.1 BLER
simParameters.SINR90pc = [-5.46 -0.46 4.54 9.05 11.54 14.04 15.54 18.04 ...
    20.04 22.43 24.93 25.43 27.43 30.43 33.43];

simParameters.SchedulerStrategy = 'PF'; % Supported scheduling strategies: 'PF', 'RR', and 'Best'
% Maximum limit on the RBs allotted for PUSCH and PDSCH
% Transmission limit is applicable for new PUSCH and PDSCH assignments and
% not for the retransmissions
simParameters.RBAllocationLimitUL = 15; % For PUSCH
simParameters.RBAllocationLimitDL = 15; % For PDSCH

% Logging and Visualization Configuration
% The parameters CQIVisualization and RBVisualization control the display
% of these visualizations: (i) CQI visualization of RBs (ii) RB assignment
% visualization. By default, the 'RBVisualization' plot is disabled. You
% can enable it by setting to 'true'
simParameters.CQIVisualization = true;
simParameters.RBVisualization = false;
% The output metrics plots are updated periodically NumMetricsSteps times within the
% simulation duration
simParameters.NumMetricsSteps = 20;
% MAT-files to write the logs into. They are used for post simulation analysis and visualization
simParameters.ParametersLogFile = 'simParameters'; % For logging the simulation parameters
simParameters.SimulationLogFile = 'simulationLogs'; % For logging the simulation logs

```



```

% Enable packet capture (PCAP)
simParameters.PCAPLogging = false; % Set the value to true to enable packet capture for UeOfInterest
simParameters.UeOfInterest = 1; % Log the packets of UE with this RNTI

% Set the periodic UL and DL application traffic pattern for UEs
simParameters.ULPacketPeriodicityUEs = [10; 20; 20; 30]; % Periodicity (in ms) at which the UL packets are generated
simParameters.ULPacketSizesUEs = [8000; 8000; 7500; 8500]; % Size of the UL packets (in bytes) generated
simParameters.DLPacketPeriodicityUEs = [10; 20; 30; 30]; % Periodicity (in ms) at which the DL packets are generated
simParameters.DLPacketSizesUEs = [10000; 9000; 7500; 7500]; % Size of the DL packets generated (in bytes)

% Validate the simulation configuration
hnrCellPerformanceWithPhysicalLayerValidateConfig(simParameters);

```

### Derived Parameters

Based on the primary configuration parameters, compute the derived parameters. Additionally, set some example specific constants.

```

simParameters.DuplexMode = 0; % FDD
simParameters.NCellID = 1; % Physical cell ID
simParameters.Position = [0 0 0]; % Position of gNB in (x,y,z) coordinates
simParameters.CSIRSRowNumber = 2; % Possible row numbers for single transmit antenna case are 1 and 2
simParameters.SubbandSize = 8; % Size of sub-band for CQI reporting in terms of number of RBs
simParameters.ChannelModelType = 'CDL'; % To model CDL propagation channel
% gNB ensures that PUSCH assignment is received at UEs PUSCHPrepTime ahead
% of the transmission time
simParameters.PUSCHPrepTime = 200; % In microseconds
% Slot duration for the selected SCS and number of slots in a 10 ms frame
slotDuration = 1/(simParameters.SCS/15); % In ms
numSlotsFrame = 10/slotDuration; % Number of slots in a 10 ms frame
numSlotsSim = simParameters.NumFramesSim * numSlotsFrame; % Number of slots in the simulation

% Interval at which metrics visualization updates in terms of number of
% slots. As one slot is the finest time-granularity of the simulation, make
% sure that MetricsStepSize is an integer
simParameters.MetricsStepSize = ceil(numSlotsSim / simParameters.NumMetricsSteps);
if mod(numSlotsSim, simParameters.NumMetricsSteps) ~= 0
    % Update the NumMetricsSteps parameter if NumSlotsSim is not
    % completely divisible by it
    simParameters.NumMetricsSteps = floor(numSlotsSim / simParameters.MetricsStepSize);
end

% DL and UL packet periodicities for UEs in terms of number of slots
appPeriodicityUEsSlotsUL = simParameters.ULPacketPeriodicityUEs ./ slotDuration;
appPeriodicityUEsSlotsDL = simParameters.DLPacketPeriodicityUEs ./ slotDuration;

simParameters.NumLogicalChannels = 1; % Only 1 logical channel is assumed in each UE in this example

% Mapping between logical channel and logical channel group ID
simParameters.LCHConfig.LCGID = ones(simParameters.NumUEs, simParameters.NumLogicalChannels);
% Priority of each logical channel
simParameters.LCHConfig.Priority = ones(simParameters.NumUEs, simParameters.NumLogicalChannels);
% Prioritized bitrate (in kilo bytes per second) of each logical channel
simParameters.LCHConfig.PBR = 8*ones(simParameters.NumUEs, simParameters.NumLogicalChannels);
% Bucket size duration (BSD) of each logical channel (in ms). However, the priority,
% PBR and BSD for logical channel is not relevant with single logical
% channel assumed in this example

```

```

simParameters.LCHConfig.BSD = 10*ones(simParameters.NumUEs, simParameters.NumLogicalChannels);
% Logical channel id (logical channel ID of data radio bearers starts from 4)
simParameters.LCHConfig.LCID = ones(simParameters.NumUEs, simParameters.NumLogicalChannels) .* (4

% RLC entity direction. Value 0 represents DL only, 1
% represents UL only and 2 represents both UL and DL
% direction. Setting entity direction to have both UL and DL
simParameters.RLCCConfig.EntityDir = 2*ones(simParameters.NumUEs, simParameters.NumLogicalChannels);

% Maximum RLC SDU length (in bytes) as per 3GPP TS 38.323
simParameters.maxRLCSDULength = 9000;

```

### gNB and UEs Setup

Create the gNB and UE objects, initialize the channel quality information for UEs, and set up the logical channel at gNB and UE. The helper classes hNRGNB.m and hNRUE.m create gNB node and UE node respectively, containing the RLC, MAC and PHY layers.

```

gNB = hNRGNB(simParameters); % Create gNB node
% Create scheduler
switch(simParameters.SchedulerStrategy)
    case 'RR' % Round robin scheduler
        scheduler = hNRSchedulerRoundRobin(simParameters);
    case 'PF' % Proportional fair scheduler
        scheduler = hNRSchedulerProportionalFair(simParameters);
    case 'BestCQI' % Best CQI scheduler
        scheduler = hNRSchedulerBestCQI(simParameters);
end
addScheduler(gNB, scheduler); % Add scheduler to gNB

gNB.PhyEntity = hNRGNBPhy(simParameters); % Create the PHY layer instance
configurePhy(gNB, simParameters); % Configure the PHY layer
setPhyInterface(gNB); % Set the interface to PHY layer

% Create the set of UE nodes
UEs = cell(simParameters.NumUEs, 1);
for ueIdx=1:simParameters.NumUEs
    simParameters.Position = [simParameters.UEDistance(ueIdx) 0 0]; % Position of UE
    UEs{ueIdx} = hNRUE(simParameters, ueIdx);
    UEs{ueIdx}.PhyEntity = hNRUEPhy(simParameters, ueIdx); % Create the PHY layer instance
    configurePhy(UEs{ueIdx}, simParameters); % Configure the PHY layer
    setPhyInterface(UEs{ueIdx}); % Set up the interface to PHY layer

% Setup the logical channels and add application traffic
for lcIdx = 1:simParameters.NumLogicalChannels
    % Create RLC channel configuration structure
    rlcChannelConfigStruct.EntityType = simParameters.RLCCConfig.EntityDir(ueIdx, lcIdx);
    rlcChannelConfigStruct.LogicalChannelID = simParameters.LCHConfig.LCID(ueIdx, lcIdx);
    rlcChannelConfigStruct.LCGID = simParameters.LCHConfig.LCGID(ueIdx, lcIdx);
    rlcChannelConfigStruct.Priority = simParameters.LCHConfig.Priority(ueIdx, lcIdx);
    rlcChannelConfigStruct.PBR = simParameters.LCHConfig.PBR(ueIdx, lcIdx);
    rlcChannelConfigStruct.BSD = simParameters.LCHConfig.BSD(ueIdx, lcIdx);
    % Setup logical channel at gNB for the UE
    configureLogicalChannel(gNB, ueIdx, rlcChannelConfigStruct);
    % Setup logical channel at UE
    configureLogicalChannel(UEs{ueIdx}, ueIdx, rlcChannelConfigStruct);

    % Add data traffic pattern generators to gNB and UE nodes

```

```

    ulPacketSize = simParameters.ULPacketSizesUEs(ueIdx);
    % Calculate the data rate (in kbps) of On-Off traffic pattern using
    % packet size (in bytes) and packet interval (in ms)
    ulDataRate = ceil(1000/simParameters.ULPacketPeriodicityUEs(ueIdx)) * ulPacketSize * 8e-3;
    % Limit the size of the generated application packet to the maximum RLC
    % SDU size. The maximum supported RLC SDU size is 9000 bytes
    if ulPacketSize > simParameters.maxRLCSDULength
        ulPacketSize = simParameters.maxRLCSDULength;
    end

    % Create an object for On-Off network traffic pattern and add it to the
    % specified UE. This object generates the uplink data traffic on the UE
    ulApp = networkTrafficOnOff('PacketSize', ulPacketSize, 'GeneratePacket', true, ...
        'OnTime', simParameters.NumFramesSim/100, 'OffTime', 0, 'DataRate', ulDataRate);
    UEs{ueIdx}.addApplication(ueIdx, simParameters.LCHConfig.LCID(ueIdx, lcIdx), ulApp);

    dlPacketSize = simParameters.DLPacketSizesUEs(ueIdx);
    dlDataRate = ceil(1000/simParameters.DLPacketPeriodicityUEs(ueIdx)) * dlPacketSize * 8e-3;
    if dlPacketSize > simParameters.maxRLCSDULength
        dlPacketSize = simParameters.maxRLCSDULength;
    end

    % Create an object for On-Off network traffic pattern for the specified
    % UE and add it to the gNB. This object generates the downlink data
    % traffic on the gNB for the UE
    dlApp = networkTrafficOnOff('PacketSize', dlPacketSize, 'GeneratePacket', true, ...
        'OnTime', simParameters.NumFramesSim/100, 'OffTime', 0, 'DataRate', dlDataRate);
    gNB.addApplication(ueIdx, simParameters.LCHConfig.LCID(ueIdx, lcIdx), dlApp);
end
end
% Setup the UL and DL packet distribution mechanism
simParameters.MaxReceivers = simParameters.NumUEs;
% Create DL packet distribution object
dlPacketDistributionObj = hNRPacketDistribution(simParameters, 0); % 0 for DL
% Create UL packet distribution object
ulPacketDistributionObj = hNRPacketDistribution(simParameters, 1); % 1 for UL
hNRSetupPacketDistribution(simParameters, gNB, UEs, dlPacketDistributionObj, ulPacketDistributionObj);

% Enable PCAP logging
if simParameters.PCAPLogging
    % To generate unique file name for every simulation run
    ueCapturefileName = strcat('CellID-', num2str(simParameters.NCellID), '_ue-', num2str(simParameters.NumUEs));
    enablePacketLogging(UEs{simParameters.UEofInterest}.PhyEntity, ueCapturefileName);

    % Uncomment the below code to enable packet capture at gNB
    % gnbCapturefileName = strcat('CellID-', num2str(simParameters.NCellID), '_gNB-', num2str(simParameters.NCellID));
    % enablePacketLogging(gNB.PhyEntity, gnbCapturefileName);
end
end

```

## Processing Loop

Simulation is run slot by slot. In each slot, these operations are executed:

- Run the MAC and PHY layers of gNB
- Run the MAC and PHY layers of UEs
- Layer-specific logging and visualization

- Advance the timer for the nodes. Every 1 ms it also sends trigger to application and RLC layers. Application layer and RLC layer execute their scheduled operations based on 1 ms timer trigger.

```

% To store these UE metrics for each slot: throughput bytes
% transmitted, goodput bytes transmitted, and pending buffer amount bytes.
% The number of goodput bytes is calculated by excluding the
% retransmissions from the total transmissions
UESlotMetricsUL = zeros(simParameters.NumUEs, 3);
UESlotMetricsDL = zeros(simParameters.NumUEs, 3);

% To store current UL and DL CQI values on the RBs for the UEs
uplinkChannelQuality = zeros(simParameters.NumUEs, simParameters.NumRBs);
downlinkChannelQuality = zeros(simParameters.NumUEs, simParameters.NumRBs);

% To store last received new data indicator (NDI) values for UL and DL HARQ processes
HARQProcessStatusUL = zeros(simParameters.NumUEs, 16);
HARQProcessStatusDL = zeros(simParameters.NumUEs, 16);

% Create an object for MAC (UL & DL) scheduling information visualization and logging
simSchedulingLogger = hNRSchedulingLogger(simParameters);
% Create an object for PHY metrics logging and visualization
simPhyLogger = hNRPhyLogger(simParameters);
symbolNum = 0;

% Run processing loop
for slotNum = 1:numSlotsSim

    % Run MAC and PHY of gNB
    run(gNB.MACEntity);
    run(gNB.PhyEntity);

    % Run MAC and PHY of UEs
    for ueIdx = 1:simParameters.NumUEs
        % Read the last received NDI flags for HARQ processes for
        % logging (Reading it before it gets overwritten by run function of MAC)
        HARQProcessStatusUL(ueIdx, :) = getLastNDIFlagHarq(UEs{ueIdx}.MACEntity, 1); % 1 for UL
        HARQProcessStatusDL(ueIdx, :) = getLastNDIFlagHarq(UEs{ueIdx}.MACEntity, 0); % 0 for DL
        run(UEs{ueIdx}.MACEntity);
        run(UEs{ueIdx}.PhyEntity);
    end

    % MAC logging
    % Read UL and DL assignments done by gNB MAC scheduler
    % at current time. Resource assignments returned by a scheduler (either
    % UL or DL) is empty, if either scheduler was not scheduled to run at
    % the current time or no resources got scheduled
    [resourceAssignmentsUL, resourceAssignmentsDL] = getCurrentSchedulingAssignments(gNB.MACEntity);
    % Read throughput and goodput bytes sent for each UE
    [UESlotMetricsDL(:, 1), UESlotMetricsDL(:, 2)] = getTTIBytes(gNB);
    UESlotMetricsDL(:, 3) = getBufferStatus(gNB); % Read pending buffer (in bytes) on gNB, for a
    for ueIdx = 1:simParameters.NumUEs
        % Read the UL channel quality at gNB for each of the UEs for logging
        uplinkChannelQuality(ueIdx, :) = getChannelQuality(gNB, 1, ueIdx); % 1 for UL
        % Read the DL channel quality at gNB for each of the UEs for logging
        downlinkChannelQuality(ueIdx, :) = getChannelQuality(gNB, 0, ueIdx); % 0 for DL
        % Read throughput and goodput bytes transmitted for this UE in the
        % current TTI for logging
        [UESlotMetricsUL(ueIdx, 1), UESlotMetricsUL(ueIdx, 2)] = getTTIBytes(UEs{ueIdx});
    end
end

```

```

        UESlotMetricsUL(ueIdx, 3) = getBufferStatus(UEs{ueIdx}); % Read pending buffer (in bytes)
    end
    % Log the scheduling logs
    logScheduling(simSchedulingLogger, symbolNum + 1, resourceAssignmentsUL, UESlotMetricsUL, up);
    logScheduling(simSchedulingLogger, symbolNum + 1, resourceAssignmentsDL, UESlotMetricsDL, down);

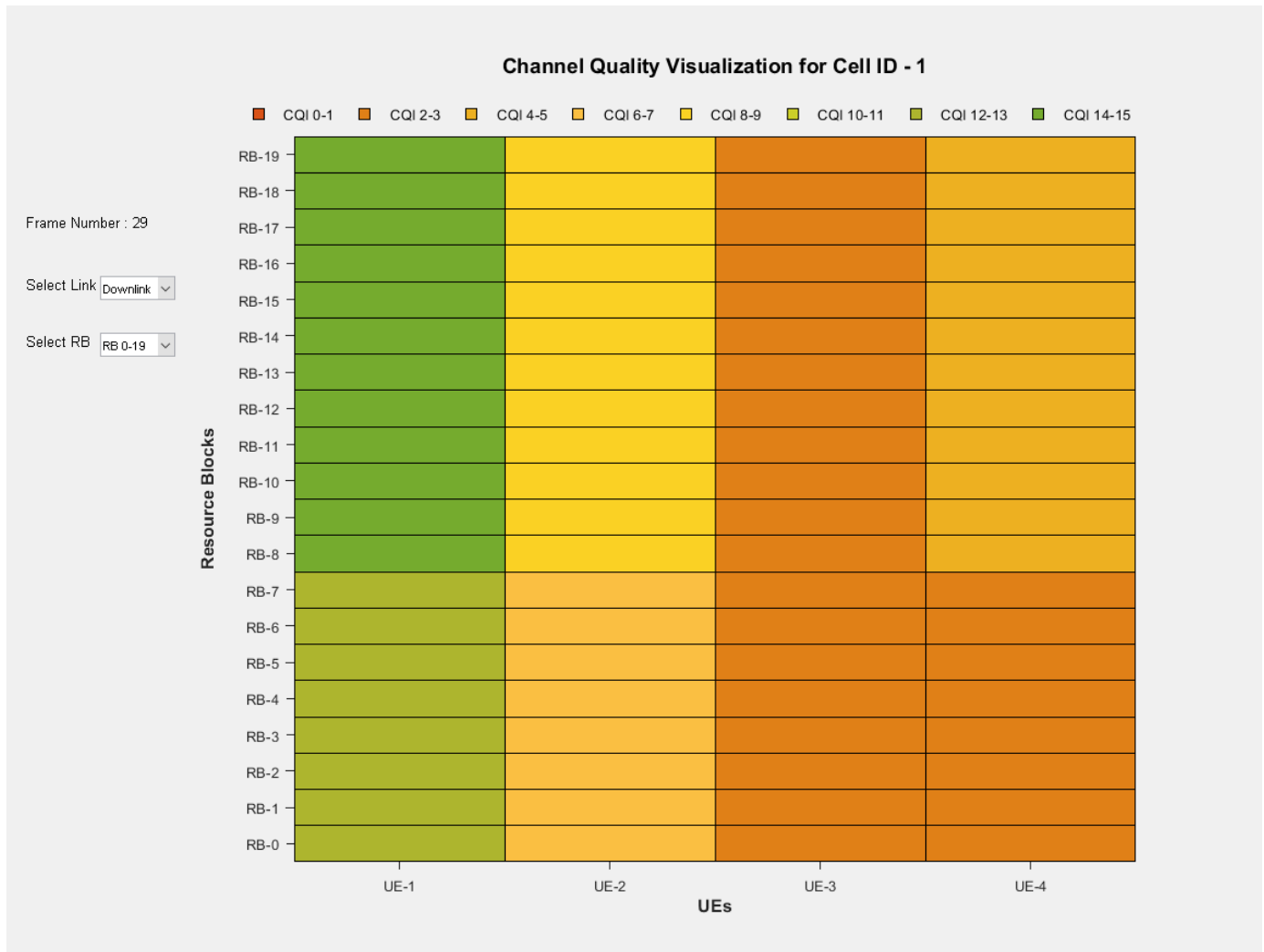
    % PHY logging
    % Read the DL BLER for each UE
    ueBLERStats = zeros(simParameters.NumUEs, 2);
    for ueIdx = 1:simParameters.NumUEs
        ueBLERStats(ueIdx, :) = getDLBLER(UEs{ueIdx}.PhyEntity);
    end
    % Read the UL BLER for each UE
    gNBBLERStats = getULBLER(gNB.PhyEntity);
    % Log the UL and DL error logs
    logBLERStats(simPhyLogger, ueBLERStats, gNBBLERStats);

    % Visualization
    % RB assignment visualization (if enabled)
    if simParameters.RBVisualization
        if mod(slotNum, numSlotsFrame) == 0
            plotRBGrids(simSchedulingLogger);
        end
    end
    % CQI grid visualization (if enabled)
    if simParameters.CQIVisualization
        if mod(slotNum, numSlotsFrame) == 0
            plotCQIRBGrids(simSchedulingLogger);
        end
    end
    % If the update periodicity is reached, plot scheduler metrics and PHY metrics visualization
    % at slot boundary
    if mod(slotNum, simParameters.MetricsStepSize) == 0
        plotMetrics(simSchedulingLogger);
        plotMetrics(simPhyLogger);
    end

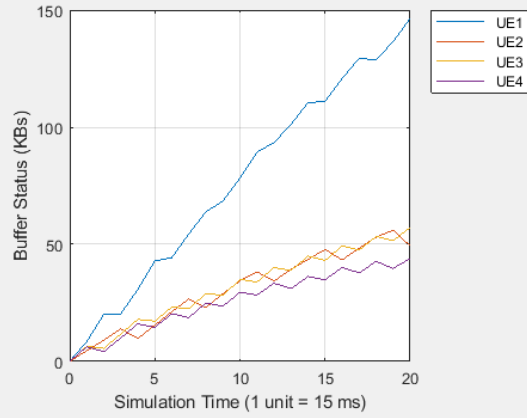
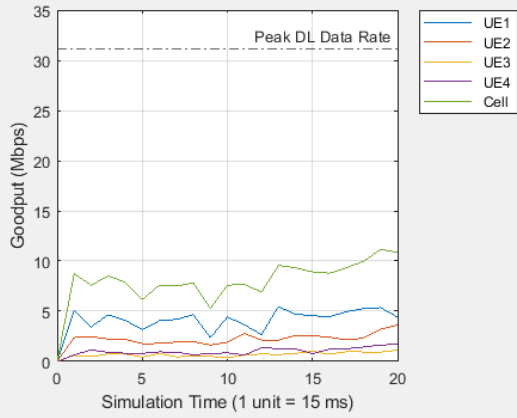
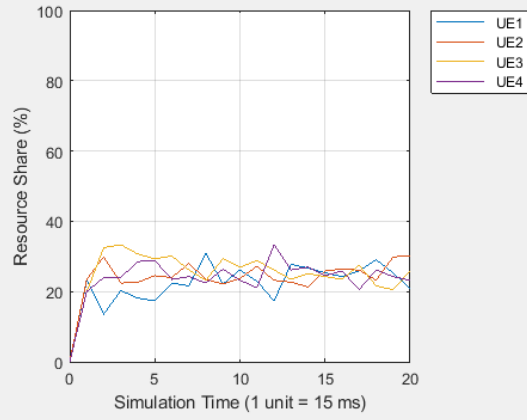
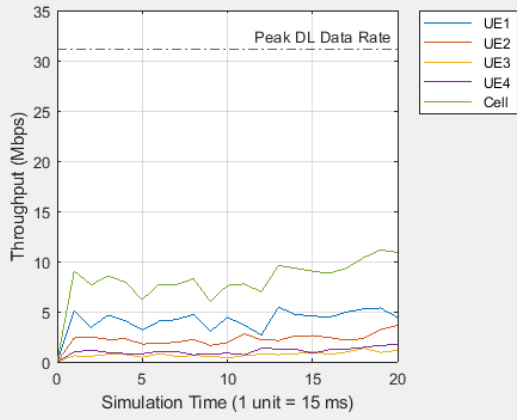
    % Advance timer ticks for gNB and UEs by 14 symbols
    advanceTimer(gNB, 14);
    for ueIdx = 1:simParameters.NumUEs
        advanceTimer(UEs{ueIdx}, 14);
    end

    % Symbol number in the simulation
    symbolNum = symbolNum + 14;
end

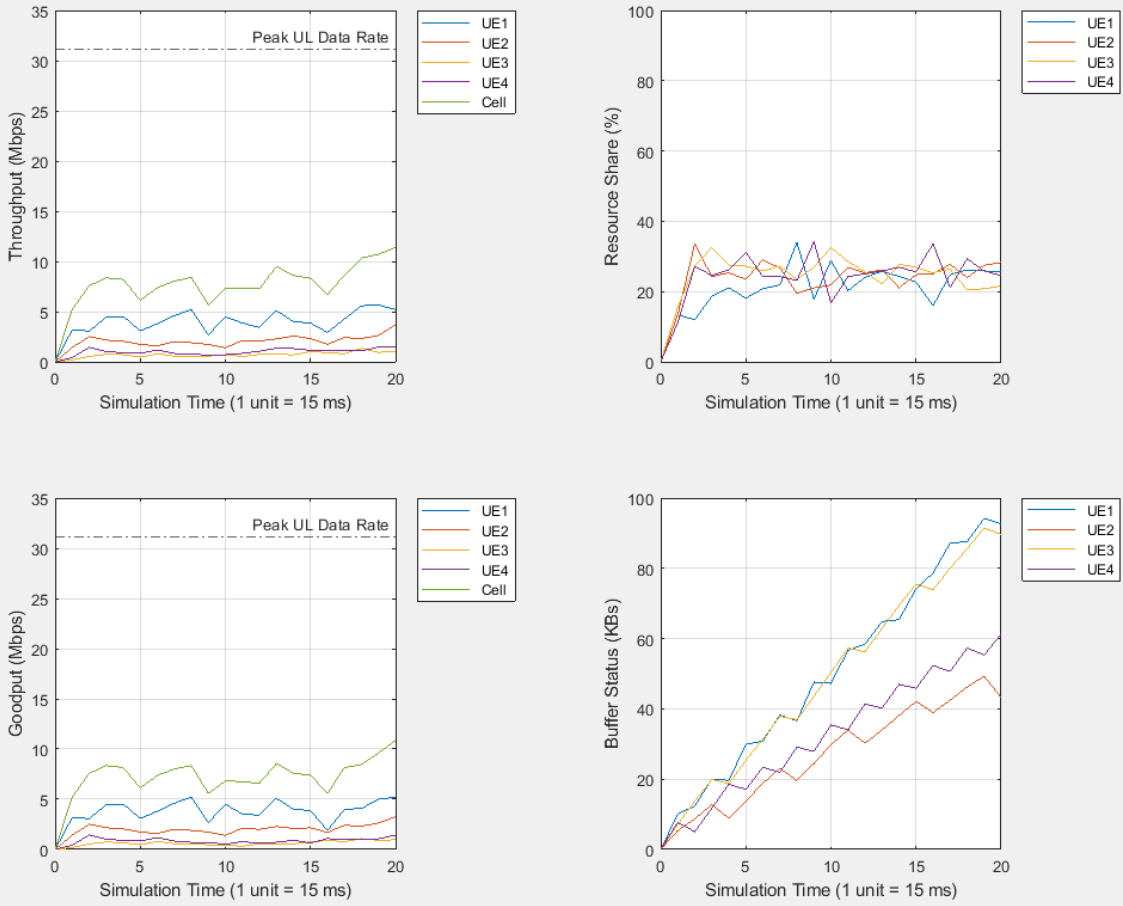
```



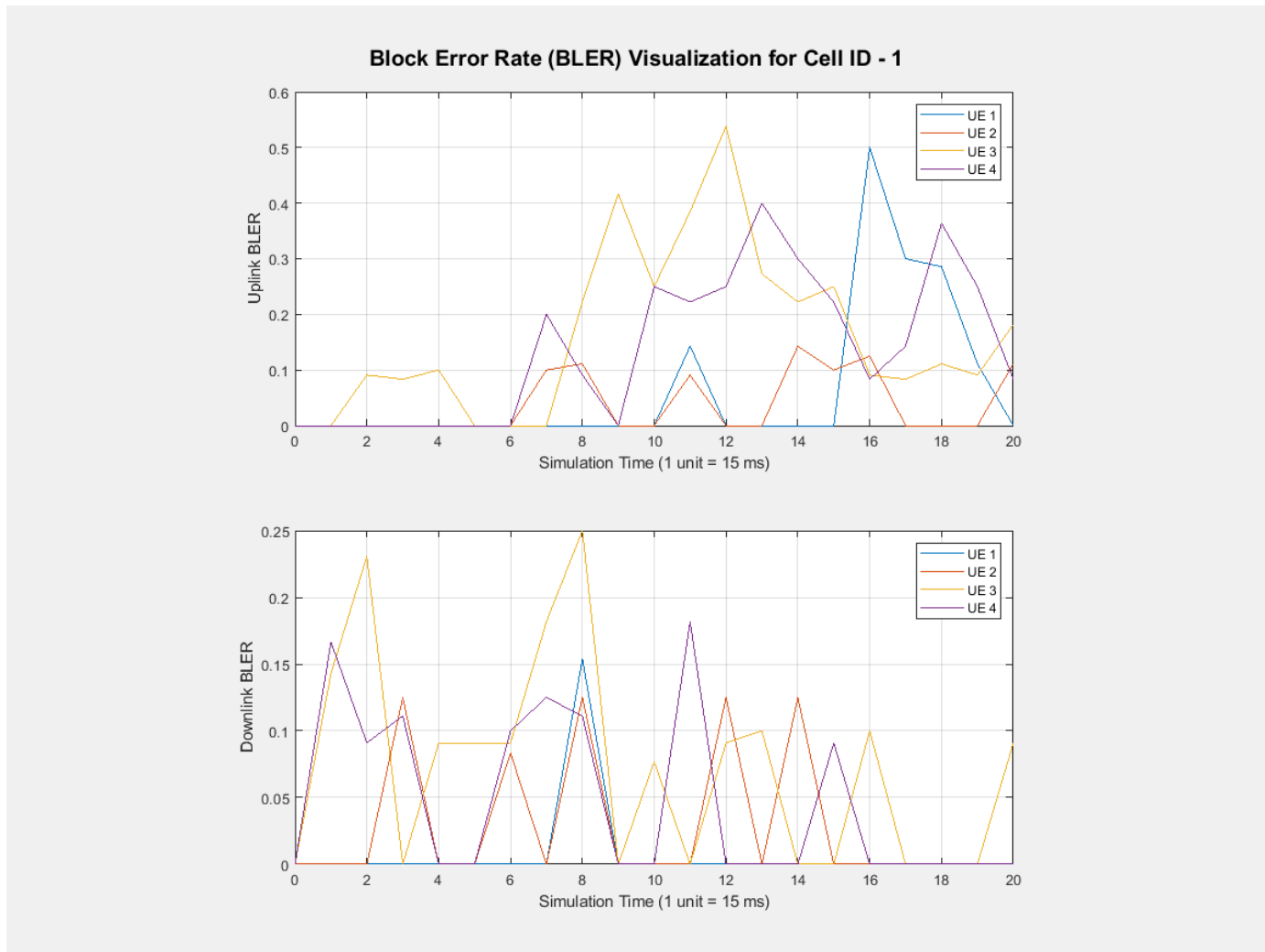
**Downlink Scheduler Performance Metrics for Cell ID - 1**



Uplink Scheduler Performance Metrics for Cell ID - 1







### Simulation Visualization

The five types of run-time visualization shown are:

- *Display of CQI values for UEs over the PUSCH/PDSCH bandwidth:* For details, see the 'Channel Quality Visualization' figure description in "NR PUSCH FDD Scheduling" on page 5-2 example.
- *Display of resource grid assignment to UEs:* The 2D time-frequency grid shows the resource allocation to the UEs. You can enable this visualization in the Scenario Configuration on page 5-0 section. For details, see the 'Resource Grid Allocation' figure description in "NR PUSCH FDD Scheduling" on page 5-2 example.
- *Display of UL scheduling metrics plots:* For details, see 'Uplink Scheduler Performance Metrics' figure description in "NR FDD Scheduling Performance Evaluation" on page 5-21 example.
- *Display of DL scheduling metrics plots:* For details, see 'Downlink Scheduler Performance Metrics' figure description in "NR FDD Scheduling Performance Evaluation" on page 5-21 example.
- *Display of DL and UL Block Error Rates:* The two sub-plots displayed in 'Block Error Rate (BLER) Visualization' shows the block error rate (for each UE) observed in the uplink and downlink directions, as the simulation progresses. The plot is updated every metricsStepSize slots.

## Simulation Logs

The parameters used for simulation and the simulation logs are saved in MAT-files for post simulation analysis and visualization. The simulation parameters are saved in a MAT-file with the file name as the value of configuration parameter `simParameters.ParametersLogFile`. The per time step logs, scheduling assignment logs, and BLER logs are saved in the MAT-file `simParameters.SimulationLogFile`. After the simulation, open the file to load `DLTimeStepLogs`, `ULTimeStepLogs`, `SchedulingAssignmentLogs`, and `RLCLogs` in the workspace.

**Time step logs:** Both the DL and UL time step logs follow the same format. For details of log format, see the 'Simulation Logs' section of "NR PUSCH FDD Scheduling" on page 5-2.

**Scheduling assignment logs:** Information of all the scheduling assignments and related information is logged in this file. For details of log format, see the 'Simulation Logs' section of "NR FDD Scheduling Performance Evaluation" on page 5-21 example.

**Block Error Rate logs:** Block error information observed in the uplink and downlink directions are logged in this file. This table shows the sample log entries.

'Frame number'	'Slot number'	'Number of Erroneous Packets(DL)'	'Number of Packets(DL)'	'Number of Erroneous Packets(UL)'	'Number of Packets(UL)'
0	0	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
0	1	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
0	2	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
0	3	[0;0;0;0]	[1;1;0;0]	[0;0;0;0]	[0;0;0;0]
0	4	[0;0;1;0]	[0;0;1;1]	[0;0;0;0]	[0;0;0;0]
0	5	[0;0;0;0]	[1;1;0;0]	[0;0;0;0]	[0;0;0;0]
0	6	[0;0;0;0]	[0;1;0;1]	[0;0;0;0]	[0;0;0;0]
0	7	[0;0;0;0]	[1;0;1;0]	[0;0;0;0]	[0;0;0;0]
0	8	[0;0;0;0]	[0;1;0;1]	[0;0;0;0]	[1;1;0;0]
0	9	[0;0;0;0]	[1;1;1;0]	[0;0;0;0]	[0;0;1;1]

Each row of the log represents one slot. The column contains the information vector of length equal to the number of UEs. Information about a UE is at an index equal to its RNTI.

At the end of the simulation, the achieved value for system performance indicators is compared to their theoretical peak values (considering zero overheads). Performance indicators displayed are achieved data rate (UL and DL), achieved spectral efficiency (UL and DL), and BLER observed for UEs (DL and UL). The peak values are calculated as per 3GPP TR 37.910.

```
simulationLogs = cell(1,1);
logInfo = struct('DLTimeStepLogs',[], 'ULTimeStepLogs',[], 'SchedulingAssignmentLogs',[], 'BLERLogs',[]);
[dlStats, ulStats] = simSchedulingLogger.getPerformanceIndicators(simParameters.ULBandwidth, simParameters.DLBandwidth, logInfo);
logInfo.BLERLogs = getBLERLogs(simPhyLogger); % Block Error rate logs
fprintf('Peak UL throughput: %0.2f Mbps. Achieved average UL Throughput: %0.2f Mbps', ulStats(1,1), ulStats(1,2));
Peak UL throughput: 31.11 Mbps. Achieved average UL Throughput: 7.67 Mbps
fprintf('\nPeak DL throughput: %0.2f Mbps. Achieved average DL Throughput: %0.2f Mbps', dlStats(1,1), dlStats(1,2));
Peak DL throughput: 31.11 Mbps. Achieved average DL Throughput: 8.09 Mbps
fprintf('\nPeak UL spectral efficiency: %0.2f bits/s/Hz. Achieved average UL spectral efficiency: %0.2f bits/s/Hz', ulStats(2,1), ulStats(2,2));
Peak UL spectral efficiency: 6.22 bits/s/Hz. Achieved average UL spectral efficiency: 1.53 bits/s/Hz
```

```
fprintf('\nPeak DL spectral efficiency: %0.2f bits/s/Hz. Achieved average DL spectral efficiency
Peak DL spectral efficiency: 6.22 bits/s/Hz. Achieved average DL spectral efficiency: 1.62 bits/s/
fprintf('\nBlock error rate for each UE in the uplink direction: %0.2f %0.2f %0.2f %0.2f', logIn
Block error rate for each UE in the uplink direction: 0.07 0.04 0.19 0.15
fprintf('\nBlock error rate for each UE in the downlink direction: %0.2f %0.2f %0.2f %0.2f \n',
Block error rate for each UE in the downlink direction: 0.01 0.03 0.08 0.05
```

You can run the script `NRPostSimVisualization` to get a post-simulation visualization of logs. For more details about the options to run this script, refer to the “NR FDD Scheduling Performance Evaluation” on page 5-21 example

```
% Read the logs and save them in MAT-files
[logInfo.DLTimeStepLogs, logInfo.ULTimeStepLogs] = getSchedulingLogs(simSchedulingLogger);
logInfo.SchedulingAssignmentLogs = getGrantLogs(simSchedulingLogger); % Scheduling assignments l
simulationLogs{1} = logInfo;
save(simParameters.ParametersLogFile, 'simParameters'); % Save simulation parameters in a MAT-fi
save(simParameters.SimulationLogFile, 'simulationLogs'); % Save simulation logs in a MAT-file
```

### Further Exploration

You can use this example to further explore these options.

#### Custom scheduling

You can modify the existing scheduling strategy to implement a custom one. Refer to 'Further Exploration' section of “NR FDD Scheduling Performance Evaluation” on page 5-21 example to see the steps involved.

#### Use passthrough physical layer

For MAC focused simulations, you can use the passthrough PHY layer by installing passthrough PHY layer object on nodes. For gNB create an object of type `hNRGNBPassthroughPhy`, and for UE create an object of type `hNRUEPassthroughPhy`. For details, see 'gNB and UEs setup' section of “NR FDD Scheduling Performance Evaluation” on page 5-21 example.

Based on described simulation parameters, the example evaluates the performance of the system measured in terms of various metrics. Different visualizations show the run time performance of the system. A more thorough post simulation analysis by using the saved logs gives a detailed picture of the operations on a per slot basis.

#### Use RLC AM

You can also switch the operating mode of an RLC entity from UM to acknowledged mode (AM) by modifying the input structure fields `EntityType` and `SeqNumFieldLength` in the `configureLogicalChannel` function of `hNRNode.m`. For more details, see the 'Further Exploration' section of “NR FDD Scheduling Performance Evaluation” on page 5-21.

### Appendix

The example uses these helper functions and classes:

- `hNRNode.m`: NR node base class for both gNB and UE

- hNRGNB.m: gNB node functionality
- hNRUE.m: UE node functionality
- hNRRLCEntity.m: Base class for RLC UM and AM entities
- hNRUMEntity.m: RLC UM functionality
- hNRAMEntity.m: RLC AM functionality
- hNRRLCDataPDUInfo.m: Creates RLC PDU information object
- hNRRLCBufferStatus.m: Generates RLC buffer status information object
- hNRRLCDataReassembly.m: Create an RLC SDU reassembly information object
- hNRMAC.m: NR MAC base class functionality
- hNRGNBMAC.m: gNB MAC functionality
- hNRUEMAC.m: UE MAC functionality
- hNRScheduler.m: Core MAC scheduler functionality
- hNRSchedulerBestCQI.m: Implements best CQI scheduling strategy
- hNRSchedulerProportionalFair.m: Implements proportional fair scheduling strategy
- hNRSchedulerRoundRobin.m: Implements round robin scheduling strategy
- hNRMACBSR.m: Generates buffer status report
- hNRMACBSRParser.m: Parses buffer status report
- hNRMACSubPDU.m: Generates MAC subPDU
- hNRMACPaddingSubPDU.m: Generates MAC subPDU with padding
- hNRMACMultiplex.m: Generates MAC PDU
- hNRMACPDUParser.m: Parses MAC PDU
- hNewHARQProcesses.m: Creates new HARQ process
- hUpdateHARQProcess.m: Updates HARQ process
- hNRPhyInterface.m: NR PHY base class functionality
- hNRGNBPhy.m: gNB PHY functionality
- hNRUEPhy.m: UE PHY functionality
- hNRGNBPassthroughPhy.m: gNB passthrough PHY layer
- hNRUEPassthroughPhy.m: UE passthrough PHY layer
- hNRPUSCHInfo.m: PUSCH information structure passed by MAC to PHY layer
- hNRPDSCHInfo.m: PDSCH information structure passed by MAC to PHY layer
- hNRRxIndicationInfo.m: Information structure passed by PHY layer to MAC along with MAC PDU
- hNRUplinkGrantFormat.m: UL grant format
- hNRDownlinkGrantFormat.m: DL grant format
- hNRPacketDistribution.m: Creates packet distribution object
- hNRPhyRxBuffer.m: Creates PHY signal reception buffer object
- hSkipWeakTimingOffset.m: Skip timing offset estimates with weak correlation
- hNRRLCLogger.m: Implements RLC statistics logging and visualization functionality
- hNRSchedulingLogger.m: Implements scheduling information logging and visualization functionality

- hNRPhyLogger.m: Implements uplink and downlink block error rate logging and visualization functionality
- hNRCellPerformanceWithPhysicalLayerValidateConfig.m: Validates simulation configuration
- hNRSetUpPacketDistribution.m: Set up packet distribution functionality
- hNRPacketWriter.m: Captures MAC packets
- hNRPacketInfo.m: Metadata format for capturing MAC packets
- NRPostSimVisualization.m: Post simulation visualization script

## References

- [1] 3GPP TS 38.104. "NR; Base Station (BS) radio transmission and reception." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
- [2] 3GPP TS 38.214. "NR; Physical layer procedures for data." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
- [3] 3GPP TS 38.321. "NR; Medium Access Control (MAC) protocol specification." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
- [4] 3GPP TS 38.322. "NR; Radio Link Control (RLC) protocol specification." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
- [5] 3GPP TS 38.323. "NR; Packet Data Convergence Protocol (PDCP) specification." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
- [6] 3GPP TS 38.331. "NR; Radio Resource Control (RRC) protocol specification." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
- [7] 3GPP TR 37.910. "Study on self evaluation towards IMT-2020 submission." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

## See Also

### Related Examples

- "NR FDD Scheduling Performance Evaluation" on page 5-21
- "NR Intercell Interference Modeling" on page 5-78
- "5G NR CQI Reporting" on page 4-8

## NR Intercell Interference Modeling

This example models the impact of intercell downlink (DL) interference caused by gNBs in a 5G New Radio (NR) network of multiple cells operating in the same frequency band and evaluates the network performance. Each cell has a gNB placed at the center of the cell that serves a set of user equipments (UEs). The network nodes in the example model the radio link control (RLC) layer, the medium access control (MAC) layer, and the physical (PHY) layer of the 5G NR stack.

### Introduction

The example considers the following operations within gNB and UEs that facilitate DL transmissions and receptions.

	Operations
gNB	<ul style="list-style-type: none"> <li>• Run the scheduling algorithm to assign downlink resources</li> <li>• Send the downlink grants to the UEs</li> <li>• Adhere to the downlink grants for physical downlink shared channel (PDSCH) transmission</li> <li>• Receive the feedback of PDSCHs from the UEs</li> <li>• Send the channel state information reference signals (CSI-RS) to the UEs</li> </ul>
UEs	<ul style="list-style-type: none"> <li>• Receive the downlink grants from the gNB</li> <li>• Receive the PDSCH transmissions from the gNB</li> <li>• Send feedback for the received PDSCHs</li> <li>• Measure channel quality on CSI-RS</li> </ul>

The complete PDSCH packet is transmitted in the first symbol of its allocated symbol set. Receiver processes the packet in the symbol just after the last symbol in the allocated symbol set.

This example models:

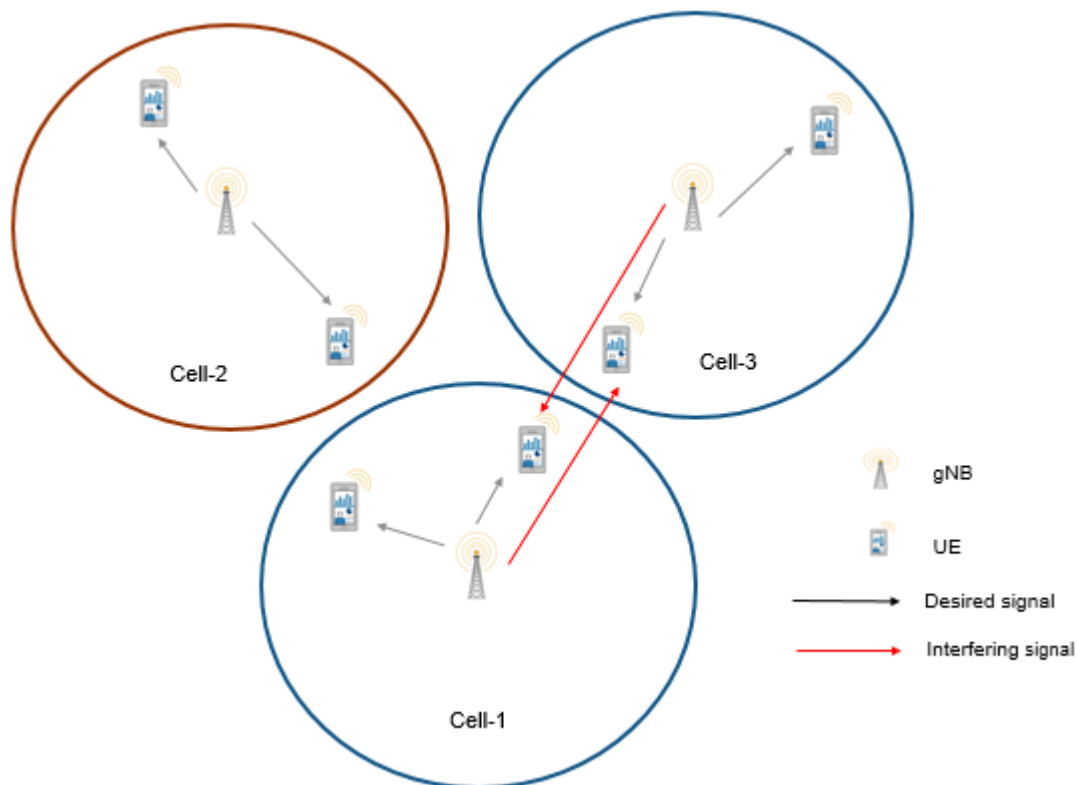
- Co-channel intercell interference.
- Slot-based DL scheduling of PDSCH resources. The time-domain granularity of the DL assignment is one slot.
- Configurable subcarrier spacing resulting in different slot durations.
- Noncontiguous allocation of frequency-domain resources in terms of resource block groups (RBGs).
- PDSCH demodulation reference signal (DM-RS).
- DL channel quality measurement by UEs based on the CSI-RS received from gNB. By default, the CSI-RS resource element is transmitted in each slot for each resource block (RB) in DL bandwidth, for all UEs. The same CSI-RS configuration is applicable to all the UEs in the cell.
- Free space path loss (FSPL), and additive white Gaussian noise (AWGN).
- Single input single output (SISO) antenna configuration.
- Single bandwidth part across the whole carrier.

Control packets such as DL assignment, PDSCH feedback, and channel quality indicator (CQI) report, are assumed to be sent out of band, that is, without the need for resources for transmission and assured error-free reception.

### Co-Channel Interference

In cellular systems, each cell operates on a particular carrier frequency. The cells operating on the same carrier frequency are called co-channel cells. Co-channel cells can interfere in transmissions between them.

Consider this sample network topology consisting of 3 cells. Cell-1 and cell-3 operate on the same frequency band. Cell-2 operates on a different frequency band and does not interfere with cell-1 or cell-3.



### NR Protocol Stack

A node (gNB or UE) consists of applications generating traffic, in addition to RLC, medium access control (MAC) and PHY. The helper classes `hNRGNB.m` and `hNRUE.m` create gNB and UE nodes respectively, containing RLC, MAC, and PHY layers. For more details on each layer, refer to the 'NR Protocol Stack' section in the "NR Cell Performance Evaluation with Physical Layer Integration" on page 5-61 example.

### Scenario Configuration

For simulation, set the following key configuration parameters.

- Simulation time

- Number of cells
- Cell radius (all the UEs connected to a gNB are within this distance)
- Positions of gNBs
- Number of UEs in each cell
- Signal to interference and noise ratio (SINR) to CQI mapping table for 0.1 block error rate (BLER). The lookup table to map the received SINR to CQI index for 0.1 BLER. The lookup table corresponds to the CQI table as per 3GPP TS 38.214 Table 5.2.2.1-3. For more information about the process of generating this lookup table, refer to “5G NR CQI Reporting” on page 4-8 example.
- Transmit power of gNB
- DL carrier bandwidth in terms of number of resource blocks (RBs)
- DL carrier frequency
- Subcarrier spacing
- DL application traffic model

```
rng('default'); % Reset the random number generator
simParameters = []; % Clear the simParameters variable
```

```
simParameters.NumFramesSim = 30; % Simulation time in terms of number of 10 ms frames
```

Number of cells in the simulation. Cells are assumed to have sequential cell identifiers (NCellIDs) from 0 to NumCells-1. If you change the number of cells, ensure that the number of rows in simParameters.GNBPosition is equal to NumCells.

```
simParameters.NumCells = 3; % Number of cells
simParameters.CellRadius = 500; % Radius of each cell (in meters)
```

The N-by-2 matrix represents the position of gNBs in (X, Y) coordinates, where 'N' is the number of cells in the simulation. The row number 'P' represents the X and Y coordinates of the gNB in the cell having cell ID 'P-1'. For example, the value [3000, 600] in the 2nd row represents the (X, Y) coordinates of the gNB in the cell having cell ID 1.

```
simParameters.GNBPosition = [1700 600;
    3000 600;
    2500 2000];
```

```
% Number of UEs in a cell. Each cell contains same number of UEs
simParameters.NumUEsCell = 4;
```

```
simParameters.GNBTxPower = 32; % Tx power for gNB (in dBm)
```

```
% Set the channel bandwidth to 10 MHz and subcarrier spacing (SCS) to 30 kHz
% as defined in 3GPP TS 38.104 Section 5.3.2.
```

```
simParameters.NumRBs = 24;
simParameters.SCS = 30; % kHz
simParameters.DLCarrierFreq = 2.635e9; % Hz
simParameters.DLBandwidth = 10e6; % Hz
```

```
% SINR to CQI mapping table for 0.1 BLER
```

```
simParameters.SINR90pc = [-5.46 -0.46 4.54 9.05 11.54 14.04 15.54 18.04 ...
    20.04 22.43 24.93 25.43 27.43 30.43 33.43];
```

```
% Maximum limit on the RBs allotted for PDSCH. Transmission limit is
% applicable for new PDSCH assignments and not for the retransmissions
simParameters.RBAllocationLimitDL = 15; % For PDSCH
```



Logging and visualization configuration.

```
% Specify the ID of cell of interest. All the visualizations and metrics are shown for this cell
simParameters.CellOfInterest = 2; % Set a value from 0 to NumCells-1

% The parameters CQIVisualization and RBVisualization control the display
% of these visualizations: (i) CQI visualization of RBs (ii) RB assignment
% visualization. By default, the 'RBVisualization' plot is disabled. You
% can enable it by setting to 'true'
simParameters.CQIVisualization = true;
simParameters.RBVisualization = false;
% The output metrics plots are updated periodically NumMetricsSteps times within the
% simulation duration
simParameters.NumMetricsSteps = 20;
% MAT-files to write the logs into. They are used for post simulation analysis and visualization
simParameters.ParametersLogFile = 'simParameters'; % For logging the simulation parameters
simParameters.SimulationLogFile = 'simulationLogs'; % For logging the simulation logs
```

Application traffic configuration.

```
% Set the periodic DL application traffic model for UEs. The following
% configuration applies for each cell
simParameters.DLPacketPeriodicityUEs = 10; % Periodicity (in ms) at which the DL packets are generated
simParameters.DLPacketSizesUEs = 2e4; % Size of the DL packets generated (in bytes) for UEs at given cell

% Validate the simulation configuration
hNRIntercellInterferenceValidateConfig(simParameters);
```

### Derived Parameters

Based on the primary configuration parameters, compute the derived parameters. Additionally, set some example specific constants.

```
simParameters.DuplexMode = 0; % FDD
simParameters.ULBandwidth = 10e6; % Hz
simParameters.ULCarrierFreq = 2.515e9; % Hz
simParameters.NumUEs = simParameters.NumUEsCell; % Number of UEs in a cell
simParameters.NCellIDList = 0:simParameters.NumCells-1; % List of physical cell IDs
% To store waveforms at the receiver, set the value to twice the number of transmitters
simParameters.UERxBufferSize = 2 * simParameters.NumCells; % Reception buffer size

% CSI-RS resource configuration. All UEs are assumed to measure channel quality on same CSI-RS resource
simParameters.CSIRSRowNumber = 2; % Possible row numbers for single transmit antenna case are 1 and 2
simParameters.SubbandSize = 8; % Size of sub-band for CQI reporting in terms of number of RBs

% Set the BSRPeriodicity to 'inf' as there is no UL traffic
simParameters.BSRPeriodicity = inf; % In ms

% Slot duration for the selected SCS and number of slots in a 10 ms frame
slotDuration = 1/(simParameters.SCS/15); % In ms
numSlotsFrame = 10/slotDuration; % Number of slots in a 10 ms frame
numSlotsSim = simParameters.NumFramesSim * numSlotsFrame; % Number of slots in the simulation

% Interval at which metrics visualization updates in terms of number of
% slots. As one slot is the finest time-granularity of the simulation, make
% sure that MetricsStepSize is an integer
simParameters.MetricsStepSize = ceil(numSlotsSim / simParameters.NumMetricsSteps);
if mod(numSlotsSim, simParameters.NumMetricsSteps) ~= 0
```

```

    % Update the NumMetricsSteps parameter if NumSlotsSim is not
    % completely divisible by it
    simParameters.NumMetricsSteps = floor(numSlotsSim / simParameters.MetricsStepSize);
end

% DL packet periodicity for UEs in terms of number of slots
appPeriodicityUEsSlotsDL = simParameters.DLPacketPeriodicityUEs / slotDuration;

simParameters.NumLogicalChannels = 1; % Only 1 logical channel is assumed in each UE in this exam

% Logical channel configuration applies for all the nodes (UEs and gNBs) in the simulation
% Mapping between logical channel and logical channel group ID
simParameters.LCHConfig.LCGID = 1;
% Priority of each logical channel
simParameters.LCHConfig.Priority = 1;
% Prioritized bitrate (PBR) of each logical channel (in kilo bytes per second)
simParameters.LCHConfig.PBR = 8;
% Bucket size duration (BSD) of each logical channel (in ms). However, the priority,
% PBR and BSD for logical channel is not relevant in this example as single logical
% channel is assumed
simParameters.LCHConfig.BSD = 10;
% Logical channel ID (logical channel ID of data radio bearers starts from 4)
simParameters.LCHConfig.LCID = 4;

% RLC entity direction. Value 0 represents DL only, 1
% represents UL only and 2 represents both UL and DL
% directions. Setting entity direction to have both UL and DL
simParameters.RLCConfig.EntityDir = 0;

% Maximum RLC service data unit (SDU) length (in bytes) as per 3GPP TS 38.323
simParameters.maxRLCSDULength = 9000;

% Generate the positions of UEs in each cell
simParameters.UEPosition = generateUEPositions(simParameters);

% Total number of UEs in the simulation
simParameters.MaxReceivers = simParameters.NumCells * simParameters.NumUEsCell;

```

### Multicell Setup

Set up the cells with an individual cell consisting of one gNB and multiple UEs. For each cell, create the gNB and UE objects, initialize the channel quality information for UEs, and set up the logical channel at gNB and UEs. The helper classes hNRGNB.m and hNRUE.m create gNB and UE nodes respectively, containing the RLC, MAC, and PHY layers.

```

gNB = cell(simParameters.NumCells, 1);
UEs = cell(simParameters.NumCells, simParameters.NumUEsCell);
% Create DL packet distribution object
dlPacketDistributionObj = hNRPacketDistribution(simParameters, 0); % 0 for DL
% Create UL packet distribution object
ulPacketDistributionObj = hNRPacketDistribution(simParameters, 1); % 1 for UL

for cellIdx = 1:simParameters.NumCells % For each cell

    simParameters.NCellID = simParameters.NCellIDList(cellIdx); % Cell ID
    simParameters.Position = [simParameters.GNBPosition(cellIdx, :) 0]; % gNB position in (x,y,z)
    gNB{cellIdx} = hNRGNB(simParameters); % Create gNB node
    scheduler = hNRSchedulerProportionalFair(simParameters); % Create proportional fair scheduler

```

```

addScheduler(gNB{cellIdx}, scheduler); % Add scheduler to gNB

gNB{cellIdx}.PhyEntity = hNRGNBPhy(simParameters); % Create the PHY layer instance
configurePhy(gNB{cellIdx}, simParameters); % Configure the PHY layer
setPhyInterface(gNB{cellIdx}); % Set up the interface to PHY layer

% For each cell, create the set of UE nodes and place them randomly within the cell radius
for ueIdx = 1:simParameters.NumUEsCell
    simParameters.Position = [simParameters.UEPosition{cellIdx}(ueIdx, :) 0]; % Position of UE
    UEs{cellIdx, ueIdx} = hNRUE(simParameters, ueIdx);
    UEs{cellIdx, ueIdx}.PhyEntity = hNRUEPhy(simParameters, ueIdx); % Create the PHY layer instance
    configurePhy(UEs{cellIdx, ueIdx}, simParameters); % Configure the PHY layer
    setPhyInterface(UEs{cellIdx, ueIdx}); % Set up the interface to PHY

    % Create RLC channel configuration structure
    rlcChannelConfigStruct.EntityType = simParameters.RLCCConfig.EntityDir;
    rlcChannelConfigStruct.LogicalChannelID = simParameters.LCHConfig.LCID;
    rlcChannelConfigStruct.LCGID = simParameters.LCHConfig.LCGID;
    rlcChannelConfigStruct.Priority = simParameters.LCHConfig.Priority;
    rlcChannelConfigStruct.PBR = simParameters.LCHConfig.PBR;
    rlcChannelConfigStruct.BSD = simParameters.LCHConfig.BSD;
    % Setup logical channel at gNB for the UE
    configureLogicalChannel(gNB{cellIdx}, ueIdx, rlcChannelConfigStruct);
    % Setup logical channel at UE
    configureLogicalChannel(UEs{cellIdx, ueIdx}, ueIdx, rlcChannelConfigStruct);

    % Add data traffic pattern generators to gNB node
    packetSize = simParameters.DLPacketSizesUEs;
    % Calculate the data rate (in kbps) of On-Off traffic pattern using
    % packet size (in bytes) and packet interval (in ms)
    dataRate = ceil(1000/simParameters.DLPacketPeriodicityUEs) * packetSize * 8e-3;
    % Limit the size of the generated application packet to the maximum
    % RLC SDU size. The maximum supported RLC SDU size is 9000 bytes
    if packetSize > simParameters.maxRLCSDULength
        packetSize = simParameters.maxRLCSDULength;
    end
    % Create an object for On-Off network traffic pattern for the specified
    % UE and add it to the gNB. This object generates the downlink data
    % traffic on the gNB for the UE
    app = networkTrafficOnOff('PacketSize', packetSize, 'GeneratePacket', true, ...
        'OnTime', simParameters.NumFramesSim/100, 'OffTime', 0, 'DataRate', dataRate);
    gNB{cellIdx}.addApplication(ueIdx, simParameters.LCHConfig.LCID, app);
end

% Setup the UL and DL packet distribution mechanism
hNRSetUpPacketDistribution(simParameters, gNB{cellIdx}, UEs(cellIdx, :), dlPacketDistribution);
end

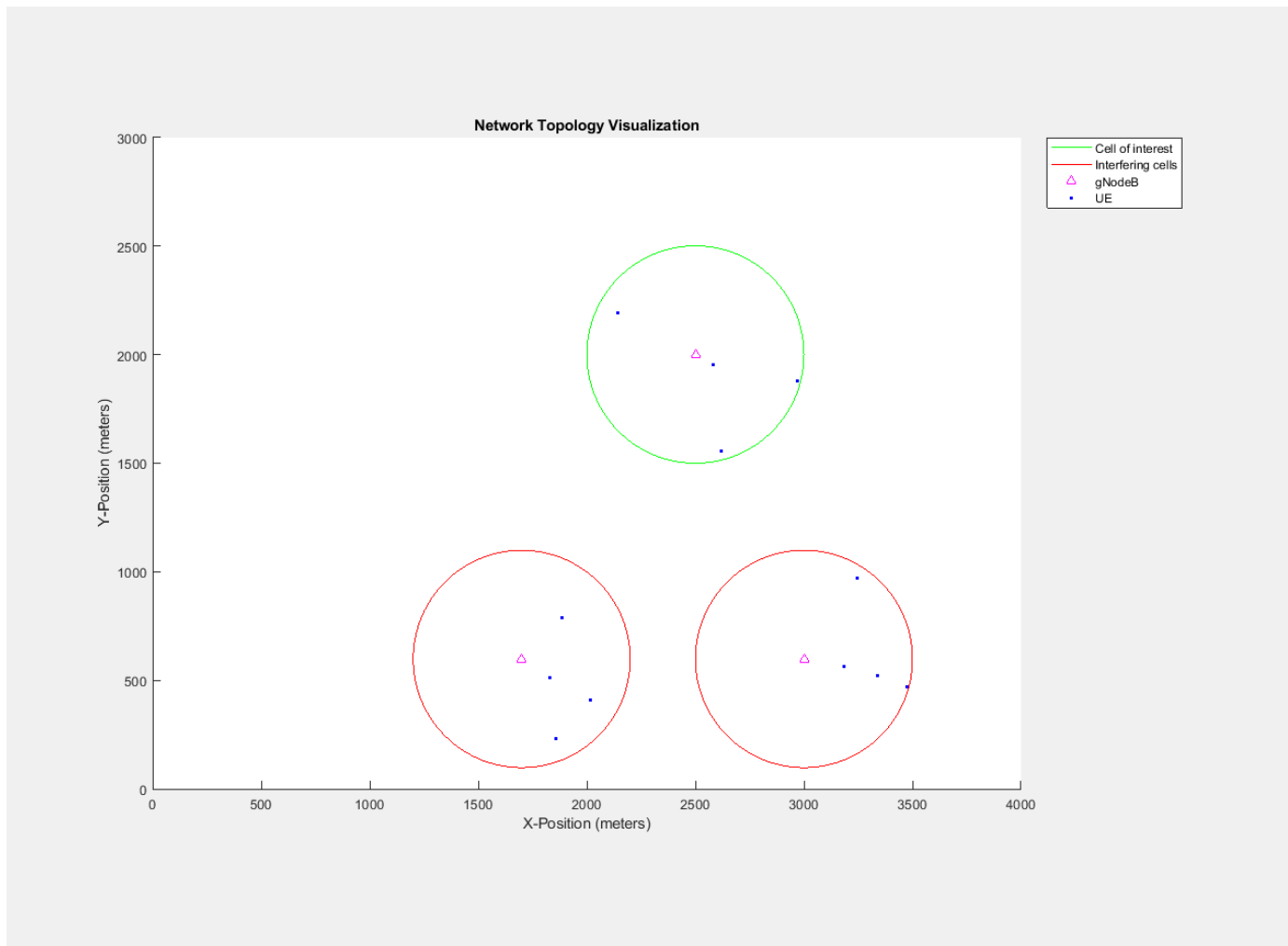
```

## Processing Loop

Simulation is run slot by slot. For each cell, in each slot, these operations are executed:

- Run the MAC and PHY layers of gNB
- Run the MAC and PHY layers of UEs
- Layer-specific logging and visualization
- Advance the timer for the nodes. Every 1 ms it also sends trigger to application and RLC layers. Application layer and RLC layer execute their scheduled operations based on 1 ms timer trigger.

```
% Display network topology
plotNetwork(simParameters);
```



```
% To store these UE metrics for each slot: throughput bytes
% transmitted, goodput bytes transmitted, and pending buffer amount bytes.
% The number of goodput bytes is calculated by excluding the
% retransmissions from the total transmissions
UESlotMetricsDL = zeros(simParameters.NumUEsCell, 3);

% To store last received new data indicator (NDI) values for DL HARQ process
HARQProcessStatusDL = zeros(simParameters.NumUEsCell, 16); % Max 16 HARQ process

% To store current DL CQI values on the RBs for different UEs
downlinkChannelQuality = zeros(simParameters.NumUEsCell, simParameters.NumRBs);

% Store the DL BLER for each UE
dlBLERStats = zeros(simParameters.NumUEsCell, 2);

simSchedulingLogger = cell(simParameters.NumCells, 1);
simPhyLogger = cell(simParameters.NumCells, 1);
for cellIdx = 1:simParameters.NumCells
```

```

% Create an object for MAC DL scheduling information visualization and logging
simParameters.NCellID = simParameters.NCellIDList(cellIdx);
simSchedulingLogger{cellIdx} = hNRSchedulingLogger(simParameters, 0); % 0 for DL
% Create an object for BLER visualization and PHY layer metrics logging
simPhyLogger{cellIdx} = hNRPhyLogger(simParameters, 0); % 0 for DL
end

% Store the index of cell ID of interest
cellOfInterestIdx = find(simParameters.CellOfInterest == simParameters.NCellIDList);

symbolNum = 0;
% Run processing loop
for slotNum = 1:numSlotsSim

% All the cells operating on same SCS hence slot durations are same
for cellIdx = 1:simParameters.NumCells % For each cell

% Run MAC and PHY layers of gNB
run(gNB{cellIdx}.MACEntity);
run(gNB{cellIdx}.PhyEntity);

% Run MAC and PHY layers of UEs
for ueIdx = 1:simParameters.NumUEsCell
% Read the last received NDI flags for DL HARQ processes for
% logging (Reading it before it gets overwritten by run function of MAC)
HARQProcessStatusDL(ueIdx, :) = getLastNDIFlagHarq(UEs{cellIdx, ueIdx}.MACEntity, 0)
run(UEs{cellIdx, ueIdx}.MACEntity);
run(UEs{cellIdx, ueIdx}.PhyEntity);
end

% MAC logging
% Read DL assignments done by gNB MAC scheduler at current time.
% Resource assignments returned by a scheduler is empty, if
% scheduler was not scheduled to run at the current time or no
% resources got scheduled
[~, resourceAssignmentsDL] = getCurrentSchedulingAssignments(gNB{cellIdx}.MACEntity);
% Read throughput and goodput bytes sent for each UE
[UESlotMetricsDL(:, 1), UESlotMetricsDL(:, 2)] = getTTIBytes(gNB{cellIdx});
UESlotMetricsDL(:, 3) = getBufferStatus(gNB{cellIdx}); % Read pending buffer (in bytes)

for ueIdx = 1:simParameters.NumUEsCell
% Read the DL channel quality at gNB for each of the UEs for logging
downlinkChannelQuality(ueIdx,:) = getChannelQuality(gNB{cellIdx}, 0, ueIdx); % 0 for
end

% Update DL scheduling logs based on the current slot run of UEs
% and gNB. Logs are updated in each slot, RB grid visualizations
% are updated every frame, and metrics plots are updated every
% metricsStepSize slots
logScheduling(simSchedulingLogger{cellIdx}, symbolNum + 1, resourceAssignmentsDL, UESlotMetricsDL);

% PHY logging
for ueIdx = 1:simParameters.NumUEsCell
dBLERStats(ueIdx, :) = getDLBLER(UEs{cellIdx, ueIdx}.PhyEntity);
end
% Log the DL BLER statistics
logBLERStats(simPhyLogger{cellIdx}, dBLERStats, []);
end
end

```

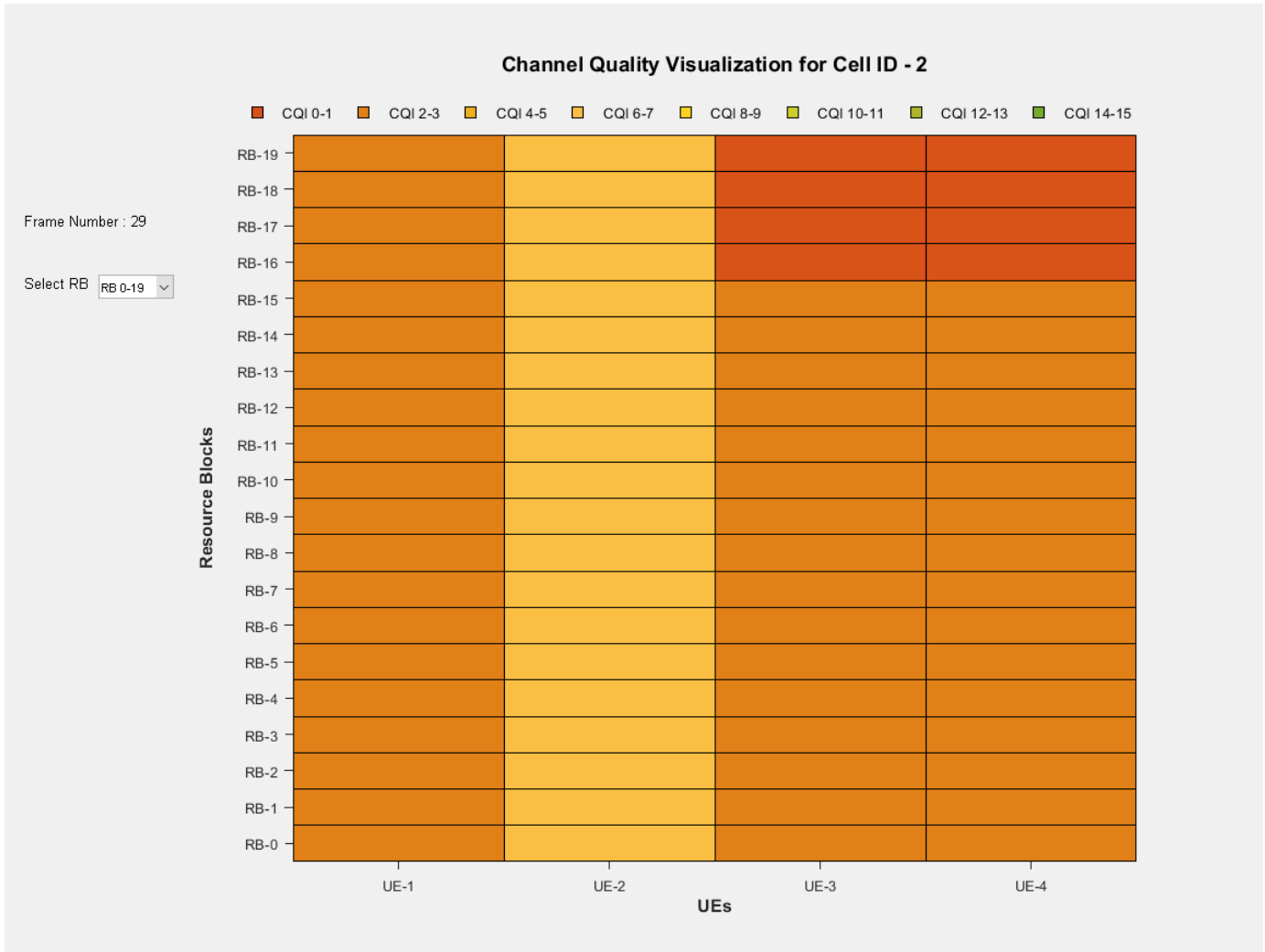
```
for cellIdx = 1:simParameters.NumCells
    % Advance timer ticks for gNB and UEs by the number of symbols per slot
    advanceTimer(gNB{cellIdx}, 14);
    for ueIdx = 1:simParameters.NumUEsCell
        advanceTimer(UEs{cellIdx, ueIdx}, 14);
    end
end

% RB assignment visualization (if enabled)
if simParameters.RBVisualization
    if mod(slotNum, numSlotsFrame) == 0
        plotRBGrids(simSchedulingLogger{cellOfInterestIdx});
    end
end

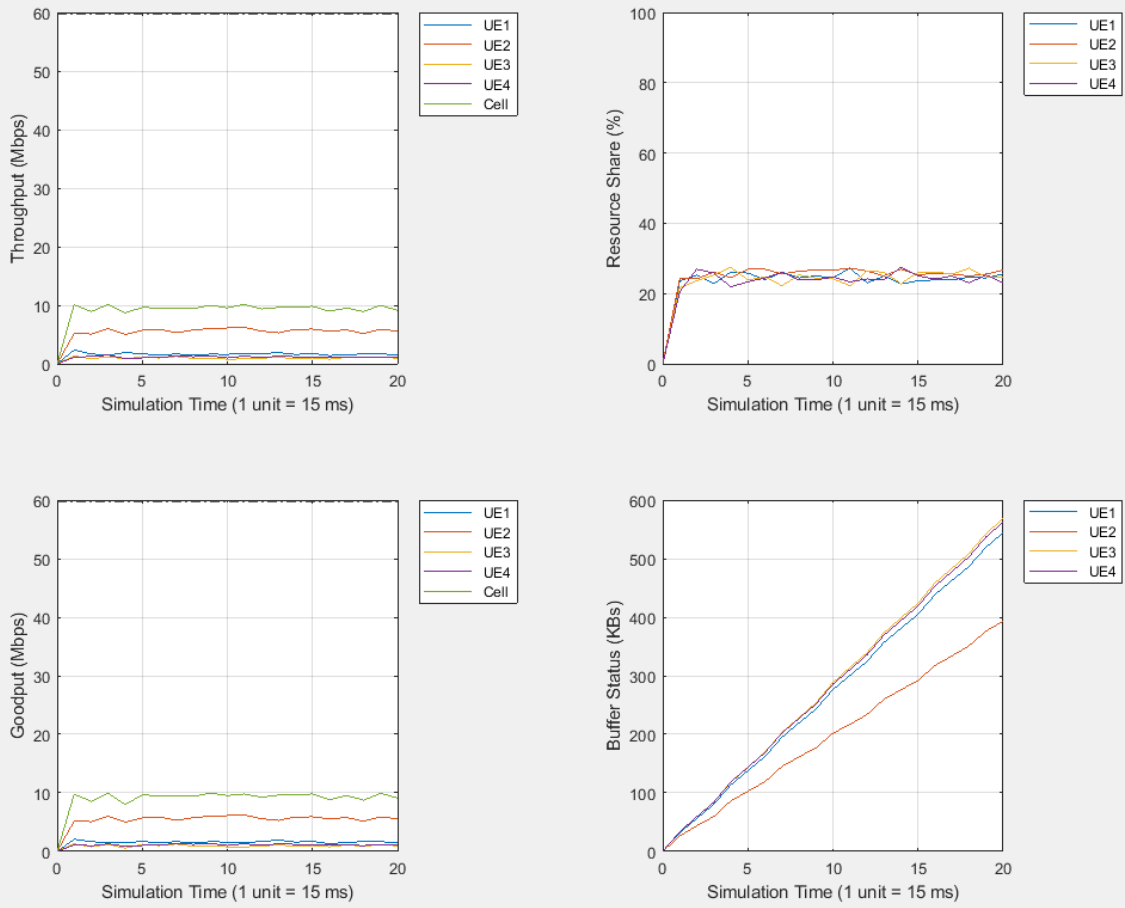
% CQI grid visualization (if enabled)
if simParameters.CQIVisualization
    if mod(slotNum, numSlotsFrame) == 0
        plotCQIRBGrids(simSchedulingLogger{cellOfInterestIdx});
    end
end

% Plot scheduler metrics and PHY metrics visualization at slot
% boundary, if the update periodicity is reached
if mod(slotNum, simParameters.MetricsStepSize) == 0
    plotMetrics(simSchedulingLogger{cellOfInterestIdx});
    plotMetrics(simPhyLogger{cellOfInterestIdx});
end

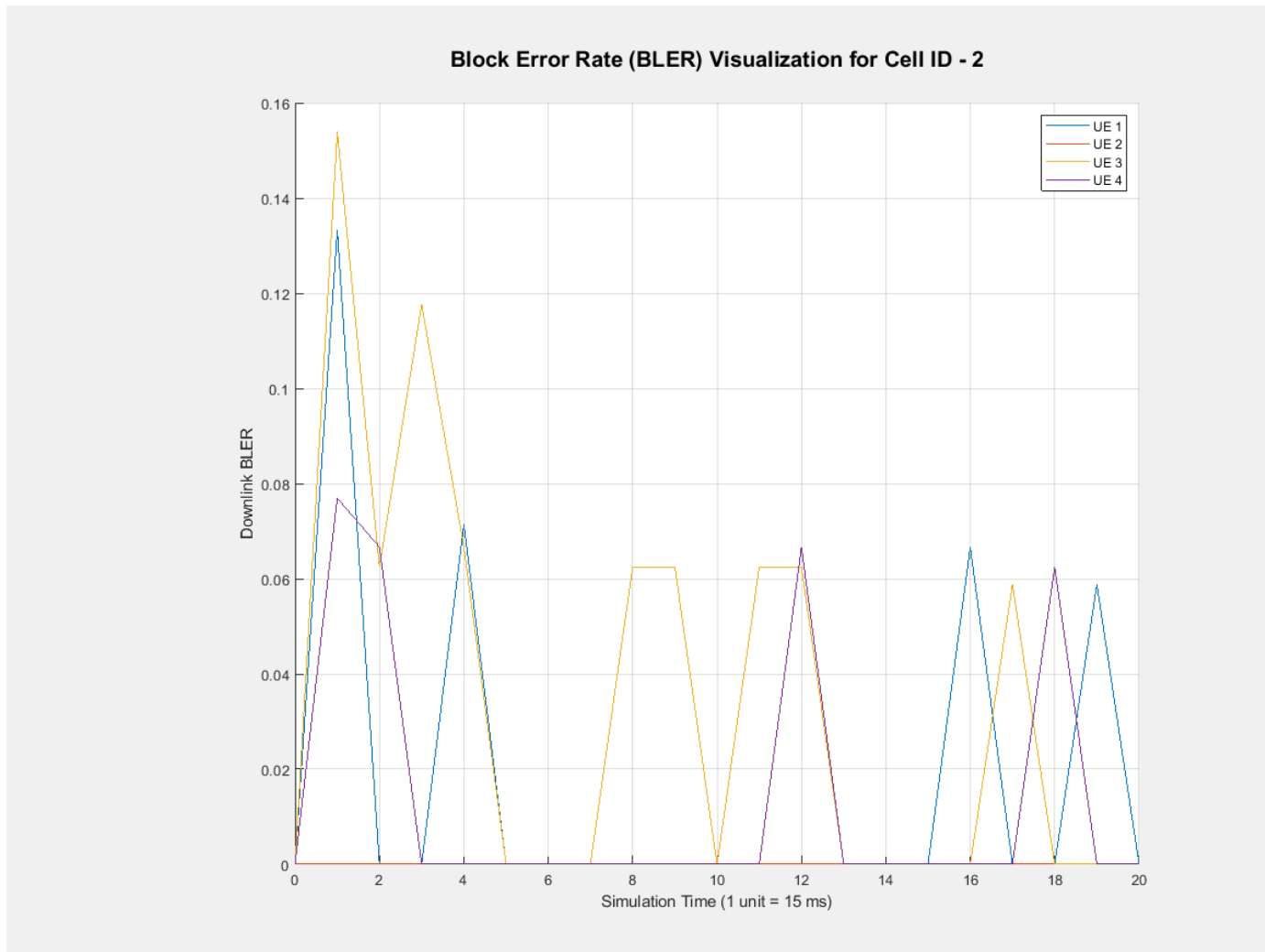
% Symbol number in the simulation
symbolNum = symbolNum + 14;
end
```



Downlink Scheduler Performance Metrics for Cell ID - 2







### Simulation Visualization

The five types of run-time visualization shown are:

- *Display of Network Topology*: The figure shows the configured cell topology. For each cell, it shows the position of the gNB and the connected UEs.
- *Display of CQI values for UEs over the PDSCH bandwidth*: For details, see the 'Channel Quality Visualization' figure description in “NR PUSCH FDD Scheduling” on page 5-2 example.
- *Display of resource grid assignment to UEs*: The 2D time-frequency grid shows the resource allocation to the UEs. You can enable this visualization in the Logging and Visualization Configuration on page 5-0 section. For details, see the 'Resource Grid Allocation' figure description in “NR PUSCH FDD Scheduling” on page 5-2 example.
- *Display of DL scheduling metrics plots*: For details, see 'Downlink Scheduler Performance Metrics' figure description in “NR FDD Scheduling Performance Evaluation” on page 5-21 example.
- *Display of DL Block Error Rates*: For details, see 'Block Error Rate (BLER) Visualization' figure description in “NR Cell Performance Evaluation with Physical Layer Integration” on page 5-61 example.

## Simulation Logs

The parameters used for simulation and the simulation logs are saved in MAT-files for post simulation analysis and visualization. The simulation parameters are saved in a MAT-file with the file name as the value of configuration parameter `simParameters.ParametersLogFile`. The per time step logs, scheduling assignment logs, and BLER logs are captured for each cell in the simulation and saved in the MAT-file `simParameters.SimulationLogFile`. After the simulation, open the file to load `NCellID`, `DLTimeStepLogs`, `SchedulingAssignmentLogs`, and `BLERLogs` in the workspace.

**NCellID:** This stores the cell ID and represents cell to which the simulation logs belong.

**DL Time step logs:** Stores the per slot logs of the simulation with each slot as one row in the simulation. For details of log format, see the 'Simulation Logs' section of “NR PUSCH FDD Scheduling” on page 5-2.

**Scheduling Assignment logs:** Information of all the scheduling assignments and related information is logged in this file. For details of log format, see the 'Simulation Logs' section in the “NR FDD Scheduling Performance Evaluation” on page 5-21 example.

**Block Error Rate logs:** Information of all the scheduling assignments and related information is logged in this file. For details of log format, see the 'Simulation Logs' section in “NR Cell Performance Evaluation with Physical Layer Integration” on page 5-61 example.

You can run the script `NRPostSimVisualization` to get a post-simulation visualization of logs. For more details about the options to run this script, refer to the “NR FDD Scheduling Performance Evaluation” on page 5-21 example.

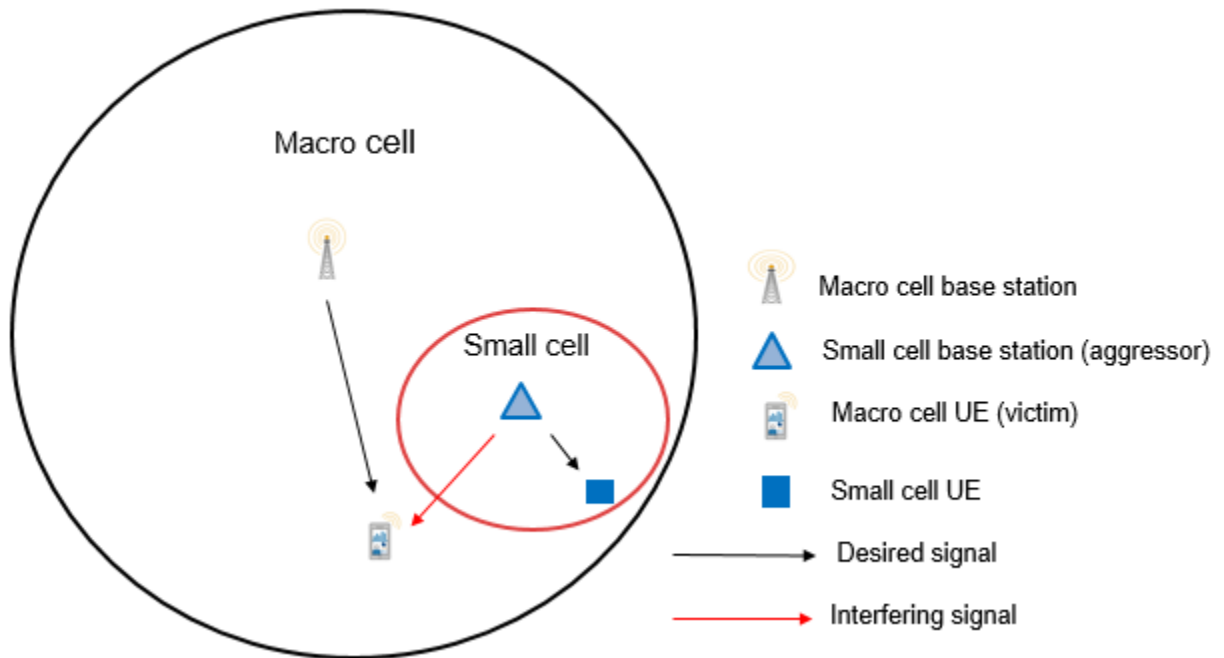
`% Get the logs`

```
logInfo = struct('NCellID',[], 'DLTimeStepLogs',[], 'SchedulingAssignmentLogs',[] , 'BLERLogs',[]);
simulationLogs = cell(simParameters.NumCells, 1);
for cellIdx = 1:simParameters.NumCells
    logInfo.NCellID = simParameters.NCellIDList(cellIdx);
    [logInfo.DLTimeStepLogs, ~] = getSchedulingLogs(simSchedulingLogger{cellIdx});
    logInfo.SchedulingAssignmentLogs = getGrantLogs(simSchedulingLogger{cellIdx}); % Scheduling a
    logInfo.BLERLogs = getBLERLogs(simPhyLogger{cellIdx}); % Block error rate logs
    simulationLogs{cellIdx, 1} = logInfo;
end
save(simParameters.ParametersLogFile, 'simParameters'); % Save simulation parameters in a MAT-fi
save(simParameters.SimulationLogFile, 'simulationLogs'); % Save simulation logs in a MAT-file
```

## Further Exploration

You can use this example to further explore these options.

- Model the uplink interference between the nodes by configuring uplink-related configuration. For details, see the “NR Cell Performance Evaluation with Physical Layer Integration” on page 5-61 example.
- Model the Aggressor-Victim scenarios: The aggressor is the source of interference and the victim suffers due to the interference. Consider the DL scenario in the following figure. The macro UE is far away from the macro base station (BS) and near to the small cell. The small cell BS interferes with macro BS transmission for macro UE in DL. Due to that macro UE suffers from interference by the small cell BS. The small cell BS is called the aggressor and macro UE is called the victim.



- To model multiple clusters, where each cluster consists of cells operating on different frequencies, and analyze the impact of interference on the cell edge users.

Based on the described simulation parameters, the example evaluates the performance of the system measured in terms of various metrics. Different visualizations show the run time performance of the system. A more thorough post simulation analysis by using the saved logs gives a detailed picture of the operations happening on a per slot basis.

## Appendix

The example uses these helper functions and classes:

- hNRNode.m: NR node base class for both gNB and UE
- hNRGNB.m: gNB node functionality
- hNRUE.m: UE node functionality
- hNRRLCEntity.m: Base class for RLC UM and AM entities
- hNRUMEntity.m: RLC UM functionality
- hNRAMEntity.m: RLC AM functionality
- hNRRLCDataPDUInfo.m: Creates RLC PDU information object
- hNRRLCBufferStatus.m: Generates RLC buffer status information object
- hNRRLCDataReassembly.m: Create an RLC SDU reassembly information object
- hNRMAC.m: NR MAC base class functionality
- hNRGNBMAC.m: gNB MAC functionality
- hNRUEMAC.m: UE MAC functionality
- hNRScheduler.m: Core MAC scheduler functionality
- hNRSchedulerBestCQI.m: Implements best CQI scheduling strategy

- `hNRSchedulerProportionalFair.m`: Implements proportional fair scheduling strategy
- `hNRSchedulerRoundRobin.m`: Implements round robin scheduling strategy
- `hNRMACBSR.m`: Generates buffer status report
- `hNRMACBSRParser.m`: Parses buffer status report
- `hNRMACSubPDU.m`: Generates MAC subPDU
- `hNRMACPaddingSubPDU.m`: Generates MAC subPDU with padding
- `hNRMACMultiplex.m`: Generates MAC PDU
- `hNRMACPDUParser.m`: Parses MAC PDU
- `hNewHARQProcesses.m`: Creates new HARQ process
- `hUpdateHARQProcess.m`: Updates HARQ process
- `hNRPhyInterface.m`: NR PHY base class functionality
- `hNRGNBPhy.m`: gNB PHY functionality
- `hNRUEPhy.m`: UE PHY functionality
- `hNRPUSCHInfo.m`: PUSCH information structure passed by MAC to PHY layer
- `hNRPDSCHInfo.m`: PDSCH information structure passed by MAC to PHY layer
- `hNRRxIndicationInfo.m`: Information structure passed by PHY layer to MAC along with MAC PDU
- `hNRUplinkGrantFormat.m`: UL grant format
- `hNRDownlinkGrantFormat.m`: DL grant format
- `hNRPacketDistribution.m`: Creates packet distribution object
- `hNRPhyRxBuffer.m`: Creates PHY signal reception buffer object
- `hSkipWeakTimingOffset.m`: Skip timing offset estimates with weak correlation
- `hNRRLCLogger.m`: Implements RLC statistics logging and visualization functionality
- `hNRSchedulingLogger.m`: Implements scheduling information logging and visualization functionality
- `hNRPhyLogger.m`: Implements uplink and downlink block error rate logging and visualization functionality
- `hNRIntercellInterferenceValidateConfig.m`: Validates simulation configuration
- `hNRSetUpPacketDistribution.m`: Set up packet distribution functionality
- `hNRPacketWriter.m`: Captures MAC packets
- `hNRPacketInfo.m`: Metadata format for capturing MAC packets
- `NRPostSimVisualization.m`: Post simulation visualization script

### Local functions

```
function plotNetwork(simParameters)
% Create the figure
figure('Name', 'Network Topology Visualization', 'units', 'normalized', 'outerposition', [0 0 1 1]);
title('Network Topology Visualization');
hold on;

for cellIdx = 1:simParameters.NumCells

    % Plot the circle
    th = 0:pi/60:2*pi;
```

```

xunit = simParameters.CellRadius * cos(th) + simParameters.GNBPosition(cellIdx, 1);
yunit = simParameters.CellRadius * sin(th) + simParameters.GNBPosition(cellIdx, 2);
if simParameters.CellOfInterest == simParameters.NCellIDList(cellIdx)
    h1 = plot(xunit, yunit, 'Color', 'green'); % Cell of interest
else
    h2 = plot(xunit, yunit, 'Color', 'red');
end
xlabel('X-Position (meters)')
ylabel('Y-Position (meters)')
% Add tool tip data for gNBs
s1 = scatter(simParameters.GNBPosition(cellIdx, 1), simParameters.GNBPosition(cellIdx, 2), 'o', 'MarkerEdgeColor', 'blue');
cellIdRow = dataTipTextRow('Cell - ', {num2str(simParameters.NCellIDList(cellIdx))});
s1.DataTipTemplate.DataTipRows(1) = cellIdRow;
posRow = dataTipTextRow('Position[X, Y]: ', {[ ' ' num2str(simParameters.GNBPosition(cellIdx, 1)) ' ' num2str(simParameters.GNBPosition(cellIdx, 2)) ' ']});
s1.DataTipTemplate.DataTipRows(2) = posRow;

% Add tool tip data for UEs
uePosition = simParameters.UEPosition{cellIdx};
for ueIdx = 1:size(uePosition, 1)
    s2 = scatter(uePosition(ueIdx, 1), uePosition(ueIdx, 2), '.', 'MarkerEdgeColor', 'blue');
    ueIdRow = dataTipTextRow('UE - ', {num2str(ueIdx)});
    s2.DataTipTemplate.DataTipRows(1) = ueIdRow;
    posRow = dataTipTextRow('Position[X, Y]: ', {[ ' ' num2str(uePosition(ueIdx, 1)) ' ' num2str(uePosition(ueIdx, 2)) ' ']});
    s2.DataTipTemplate.DataTipRows(2) = posRow;
end
end
% Create the legend
if simParameters.NumCells > 1
    legend([h1 h2 s1 s2], 'Cell of interest', 'Interfering cells', 'gNodeB', 'UE', 'Location', 'Location');
else
    legend([h1 s1 s2], 'Cell of interest', 'gNodeB', 'UE', 'Location', 'northeastoutside')
end
axis([0 4000 0 3000]); % Set axis limits
hold off;
daspect([1000,1000,1]); % Set data aspect ratio
end

function uePositions = generateUEPositions(simParameters)
% Return the position of UEs in each cell

uePositions = cell(simParameters.NumCells, 1);
for cellIdx=1:simParameters.NumCells
    gnbXCo = simParameters.GNBPosition(cellIdx, 1); % gNB X-coordinate
    gnbYCo = simParameters.GNBPosition(cellIdx, 2); % gNB Y-coordinate
    theta = rand(simParameters.NumUEsCell, 1)*(2*pi);
    % Expression to calculate position of UEs with in the cell. By default,
    % it will place the UEs randomly with in the cell
    r = sqrt(rand(simParameters.NumUEsCell, 1))*simParameters.CellRadius;
    x = round(gnbXCo + r.*cos(theta));
    y = round(gnbYCo + r.*sin(theta));
    uePositions{cellIdx} = [x y];
end

```

end  
end

### References

- [1] 3GPP TS 38.104. "NR; Base Station (BS) radio transmission and reception." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
- [2] 3GPP TS 38.214. "NR; Physical layer procedures for data." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
- [3] 3GPP TS 38.321. "NR; Medium Access Control (MAC) protocol specification." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
- [4] 3GPP TS 38.322. "NR; Radio Link Control (RLC) protocol specification." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
- [5] 3GPP TS 38.323. "NR; Packet Data Convergence Protocol (PDCP) specification." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
- [6] 3GPP TS 38.331. "NR; Radio Resource Control (RRC) protocol specification." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
- [7] 3GPP TR 37.910. "Study on self evaluation towards IMT-2020 submission." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

### See Also

### Related Examples

- "NR Cell Performance Evaluation with Physical Layer Integration" on page 5-61
- "NR FDD Scheduling Performance Evaluation" on page 5-21

## Generate and Visualize FTP Application Traffic Pattern

This example shows how to generate a file transfer protocol (FTP) application traffic pattern based on the IEEE® 802.11ax™ Evaluation Methodology [ 1 on page 5-0 ] and the 3GPP TR 36.814 specification [ 2 on page 5-0 ].

### FTP Application Traffic Model

Multinode communication systems involve modeling of different application traffic models. Each application is characterized by parameters such as the data rate, packet inter arrival time, and packet size. To evaluate various algorithms and protocols, standardization bodies such as IEEE and 3GPP define certain application traffic patterns such as Voice over Internet Protocol (VoIP), video conferencing, and FTP. This example generates and visualizes an FTP application traffic pattern.

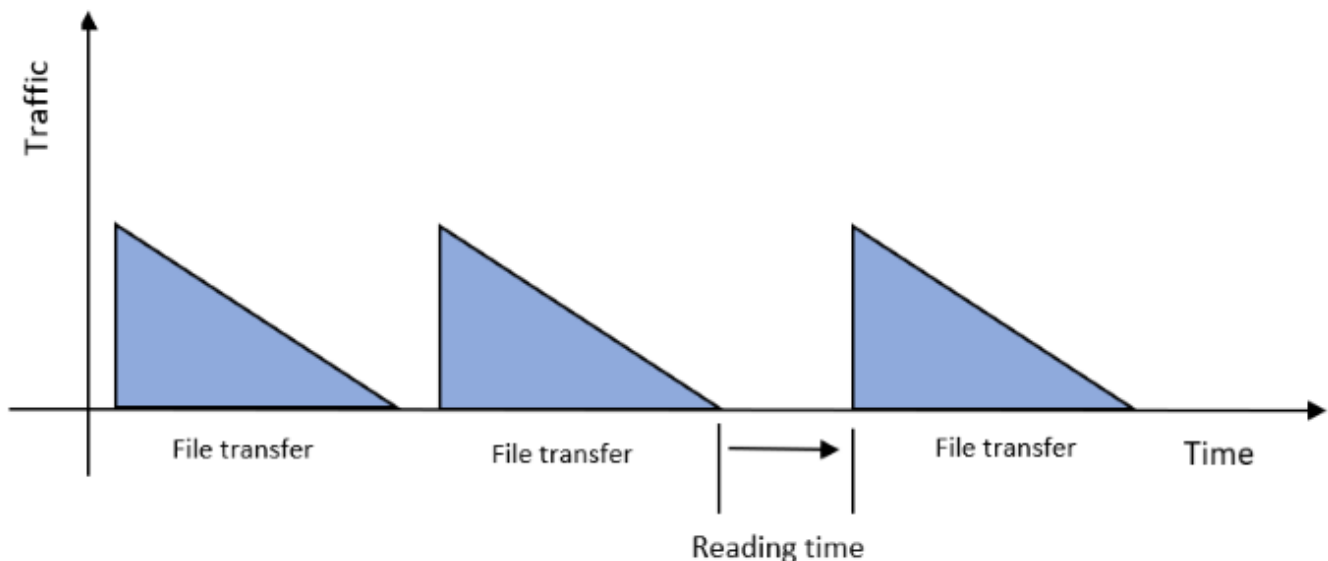
The FTP application traffic pattern is modeled as a sequence of file transfers separated by reading time. The reading time specifies the time interval between two successive file transfers. The file is generated as multiple packets separated by packet inter arrival time. The packet inter arrival time specifies the time interval between two successive packet transfers.

The 11ax Evaluation Methodology [ 1 on page 5-0 ] specifies this FTP application traffic model:

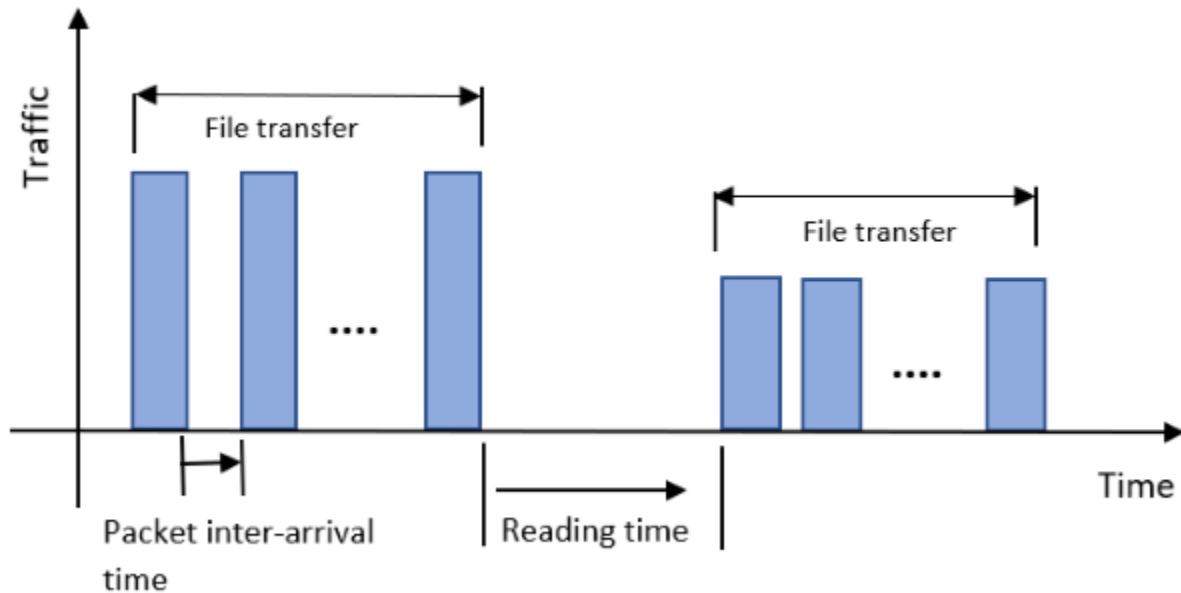
- **Local FTP traffic model** - This model is characterized by truncated Lognormal file size and exponential reading time.

The 3GPP TR 36.814 specification [ 2 on page 5-0 ] specifies these FTP application traffic models:

- **FTP traffic model 2** - This model is characterized by 2/0.5 megabytes file size and exponential reading time. This figure shows the traffic pattern of this FTP model.



- **FTP traffic model 3** - This model is characterized by a 0.5 megabytes file, exponential reading time, and Poisson packet arrival rate. This figure shows the traffic pattern of this FTP model.



This example demonstrates the local FTP traffic model specified in 11-ax Evaluation Methodology [ 1 on page 5-0 ]. Similarly, you can use the FTP traffic models 2 and 3 specified in 3GPP TR 36.814 specification [ 2 on page 5-0 ] using the file size and packet arrival rate properties.

### Configure FTP Application Traffic Pattern Object

Create a configuration object to generate the FTP application traffic pattern.

```
% Create FTP application traffic pattern object with default properties
ftpObj = networkTrafficFTP;

% Set exponential distribution mean value for reading time in milliseconds
ftpObj.ExponentialMean = 50;

% Set truncated Lognormal distribution mu value for file size calculation
ftpObj.LogNormalMu = 10;

% Set truncated Lognormal distribution sigma value for file size calculation
ftpObj.LogNormalSigma = 1;

% Set truncated Lognormal distribution upper limit in megabytes
ftpObj.UpperLimit = 5;

% Display object
disp(ftpObj);

networkTrafficFTP with properties:

    LogNormalMu: 10
    LogNormalSigma: 1
        UpperLimit: 5
    ExponentialMean: 50
```



```

PacketInterArrivalTime: 0
GeneratePacket: 0

```

### Generate and Visualize FTP Application Traffic Pattern

Generate FTP application traffic pattern using the generate object function of the networkTrafficFTP object.

```

% Set simulation time in milliseconds
simTime = 10000;

% Set step time in milliseconds
stepTime = 1;

% Validate simTime, simTime must be greater than or equal to stepTime
validateattributes(simTime, {'numeric'}, {'real', 'scalar', 'finite', '>='}, stepTime);
rng('default');

% Time after which the generate method must be invoked again
nextInvokeTime = 0;

% Generated packet count
packetCount = 0;

% Initialize arrays to store outputs for visualization
% Packet generation times in milliseconds
generationTime = zeros(5000, 1);

% Time interval between two consecutive packet transfers in milliseconds
packetIntervals = zeros(5000, 1);

% Packet sizes in bytes
packetSizes = zeros(5000, 1);

% Loop over the simulation time, generating FTP application traffic pattern
% and saving the dt and packet size values for visualization
while simTime
    if nextInvokeTime <= 0 % Time to generate the packet
        packetCount = packetCount+1; % Increment packet count
        % Call generate method and store outputs for visualization
        [packetIntervals(packetCount), packetSizes(packetCount)] = generate(ftpObj);
        % Set next invoke time
        nextInvokeTime = packetIntervals(packetCount);
        % Store packet generation time for visualization
        generationTime(packetCount+1) = generationTime(packetCount) + packetIntervals(packetCount);
    end

    % Update next invoke time
    nextInvokeTime = nextInvokeTime - stepTime;

    % Update simulation time
    simTime = simTime - stepTime;
end

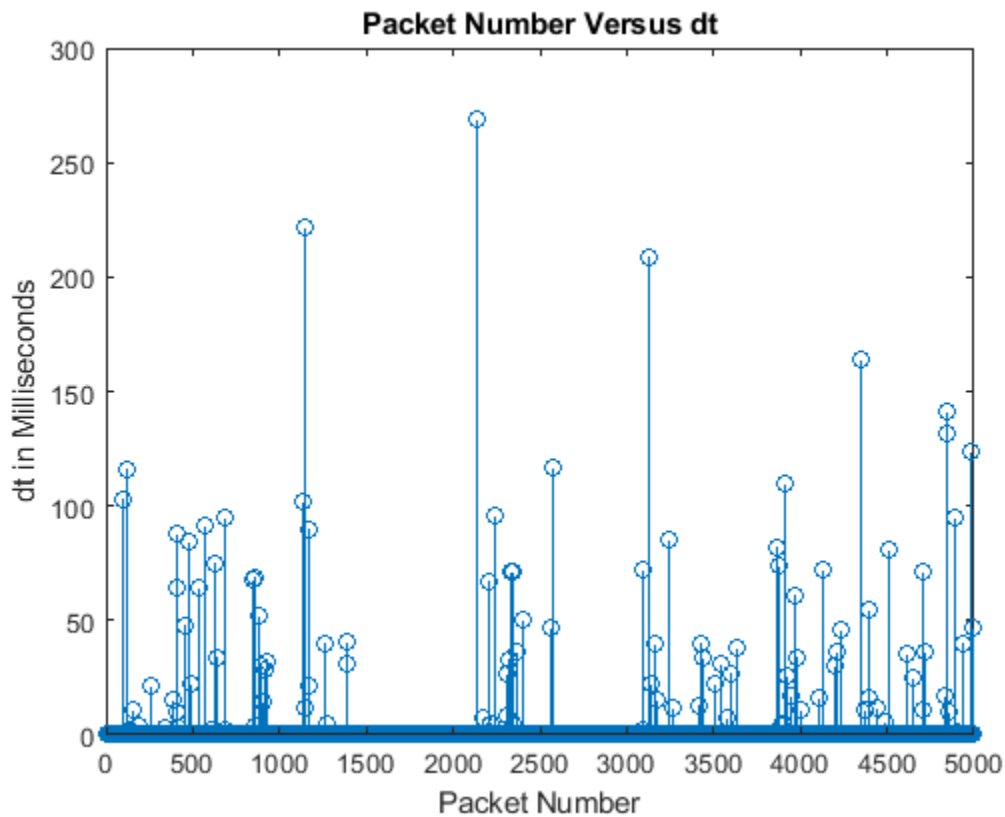
```

Visualize the generated FTP application traffic pattern. In this plot, dt is the time interval between two successive FTP application packets.

```

% Packet Number Versus Packet Intervals (dt)
% Stem graph to see packet intervals
pktIntervalsFig = figure('Name', 'Packet intervals', 'NumberTitle', 'off');
pktIntervalsAxes = axes(pktIntervalsFig);
stem(pktIntervalsAxes, packetIntervals(1:packetCount));
title(pktIntervalsAxes, 'Packet Number Versus dt');
xlabel(pktIntervalsAxes, 'Packet Number');
ylabel(pktIntervalsAxes, 'dt in Milliseconds');

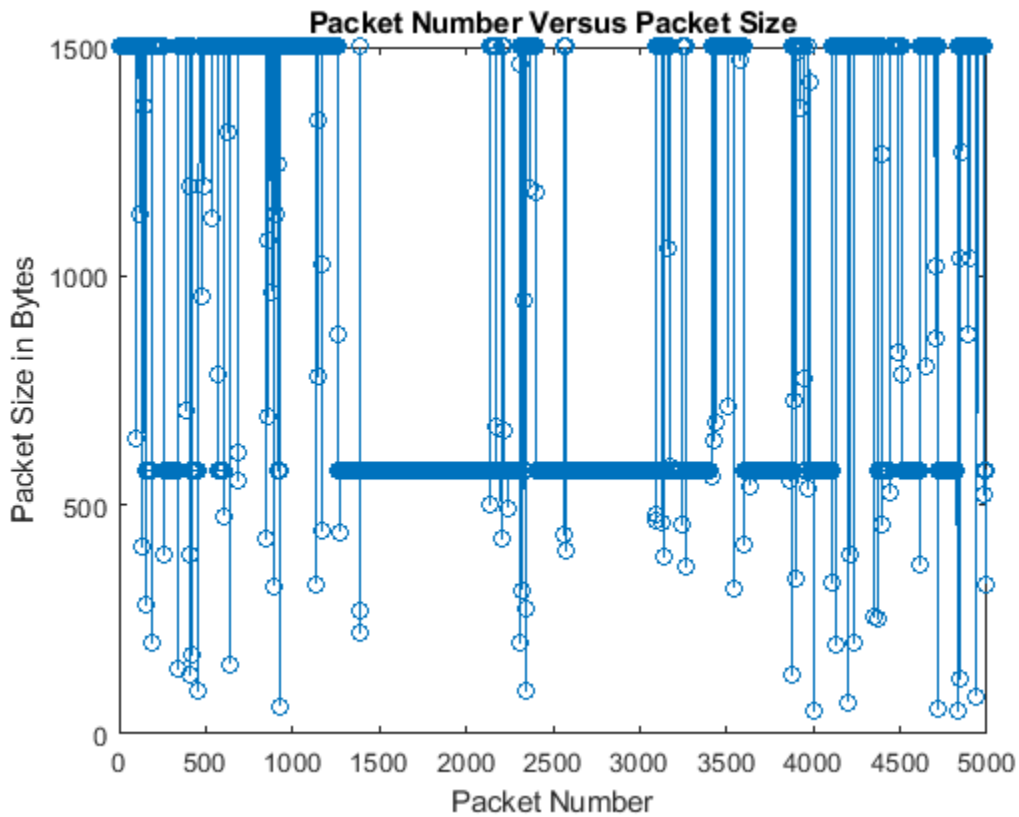
```



```

% Plot to see different packet sizes
pktSizesFig = figure('Name', 'Packet sizes', 'NumberTitle', 'off');
pktSizesAxes = axes(pktSizesFig);
plot(pktSizesAxes, packetSizes(1:packetCount), 'marker', 'o');
title(pktSizesAxes, 'Packet Number Versus Packet Size');
xlabel(pktSizesAxes, 'Packet Number');
ylabel(pktSizesAxes, 'Packet Size in Bytes');

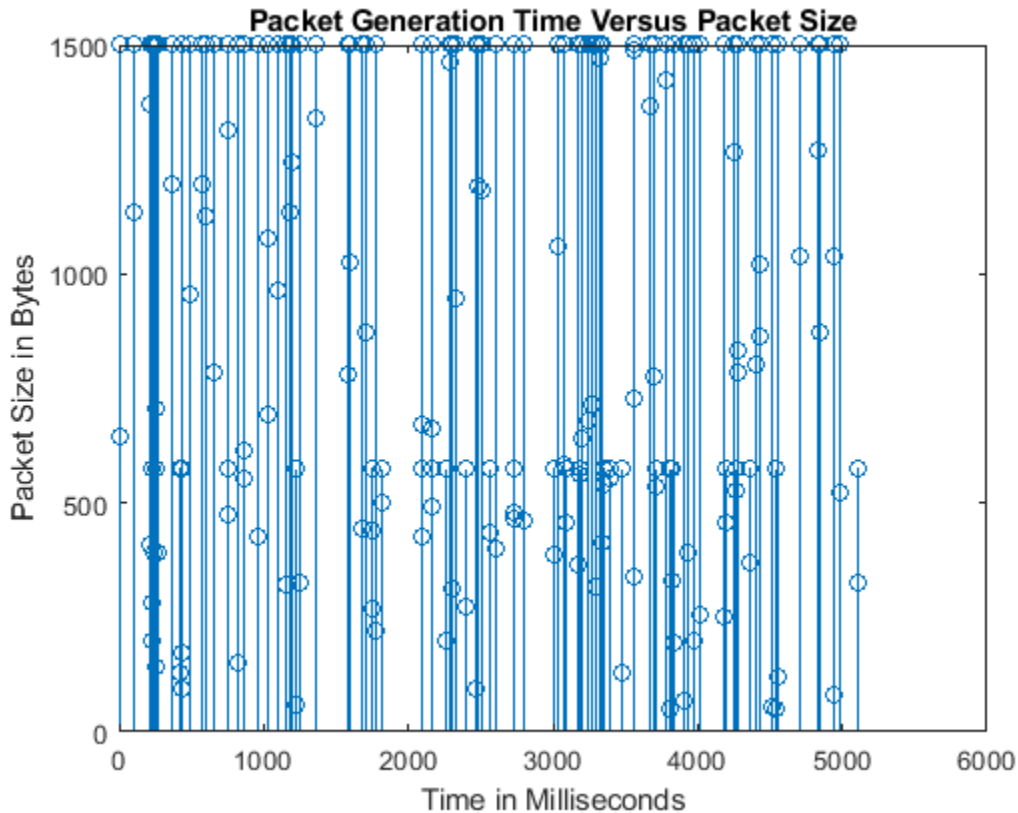
```



```

% Stem graph of FTP application traffic pattern (Packet sizes of different files
% at different packet generation times)
ftpPatternFig = figure('Name', 'FTP application traffic pattern', 'NumberTitle', 'off');
ftpPatternAxes = axes(ftpPatternFig);
stem(ftpPatternAxes, generationTime(1:packetCount), packetSizes(1:packetCount), 'Marker', 'o');
title(ftpPatternAxes, 'Packet Generation Time Versus Packet Size');
ylabel(ftpPatternAxes, 'Packet Size in Bytes');
xlabel(ftpPatternAxes, 'Time in Milliseconds');

```



### Further Exploration

This example generates an FTP traffic pattern as per the 11ax Evaluation Methodology [ 1 on page 5-0 ] and 3GPP specification [ 2 on page 5-0 ]. Similarly, you can use `networkTrafficVoIP` and `networkTrafficOnOff` objects to generate VoIP and On-Off application traffic patterns, respectively. You can use these different application traffic patterns in system-level simulations to accurately model the real-world data traffic.

### References

- 1 IEEE 802.11-14/0571r12 . "11ax Evaluation Methodology". IEEE P802.11. Wireless LANs.
- 2 3GPP TR 36.814. "Evolved Universal Terrestrial Radio Access (E-UTRA). Further advancements for E-UTRA physical layer aspects". *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

### See Also

#### Objects

`networkTrafficFTP` | `networkTrafficOnOff` | `networkTrafficVoIP`

# Test and Measurement

---

## 5G NR-TM and FRC Waveform Generation

This example shows how to generate standard-compliant 5G NR test models (NR-TMs) and uplink and downlink fixed reference channels (FRCs) for frequency range 1 (FR1) and frequency range 2 (FR2). For the NR-TM and FRC waveform generation, you can specify the NR-TM or FRC name, the channel bandwidth, the subcarrier spacing, and the duplexing mode.

### Introduction

The 3GPP 5G NR standard defines sets of link and waveform configurations for the purposes of conformance testing. Two specific types of downlink conformance waveforms are NR test models (NR-TM), for the purpose of base station (BS) RF testing, and downlink fixed reference channels (FRC), for user equipment (UE) input testing.

The NR-TMs for FR1 are defined in TS 38.141-1 Section 4.9.2, and the NR-TMs for FR2 are defined in TS 38.141-2 Section 4.9.2.

They are used in a range of RF tests, including:

- BS output power
- Timing alignment error (TAE)
- Occupied bandwidth emissions
- Adjacent channel leakage ratio (ACLR)
- Operating band unwanted emissions
- Transmitter spurious emissions
- Transmitter intermodulation

Specific test models are aimed at specific sets of measurements.

The physical downlink shared channel (PDSCH) FRC for FR1 are defined in TS 38.101-1 Annex A.3, and for FR2 are defined in TS 38.101-2 Annex A.3.

They are used in a number of UE tests, including:

- UE receiver requirements
- Maximum UE input level testing

The physical uplink shared channel (PUSCH) FRC for FR1 and FR2 are defined in TS 38.104 Annex A.

They are used in a number of base station reception tests, including:

- Reference sensitivity
- Adjacent channel selectivity (ACS)
- In-band and out-of-band blocking
- Receiver intermodulation
- In-channel selectivity
- Dynamic range
- Performance requirements

NR-TMs and FRCs are defined across a standardized set of transmission bandwidth configurations for a valid range of channel bandwidth and subcarrier spacing combinations.

This reference application example uses the MATLAB class `hNRReferenceWaveformGenerator`. This class provides access to the bandwidth configuration tables, the Release 15 test model and FRC lists, and provides baseband waveform generation and resource grid visualization.

The `hNRReferenceWaveformGenerator` class contains two constant MATLAB table properties. The `FR1BandwidthTable` property contains the FR1 transmission bandwidth configurations defined in TS 38.104 Table 5.3.2-1. See also the FR1 maximum transmission bandwidth configurations defined in TS 38.101-1 Table 5.3.2-1. The `FR2BandwidthTable` property contains the FR2 transmission bandwidth configurations defined in TS 38.104 Table 5.3.2-2. See also the FR2 maximum transmission bandwidth configurations defined in TS 38.101-2 Table 5.3.2-1.

```
% Release 15 transmission bandwidth configurations
```

```
fr1bandwidthtable = hNRReferenceWaveformGenerator.FR1BandwidthTable
```

```
fr1bandwidthtable=3x12 table
```

	5MHz	10MHz	15MHz	20MHz	25MHz	30MHz	40MHz	50MHz	60MHz	80MHz
15kHz	25	52	79	106	133	160	216	270	NaN	NaN
30kHz	11	24	38	51	65	78	106	133	162	216
60kHz	NaN	11	18	24	31	38	51	65	79	107

```
fr2bandwidthtable = hNRReferenceWaveformGenerator.FR2BandwidthTable
```

```
fr2bandwidthtable=2x4 table
```

	50MHz	100MHz	200MHz	400MHz
60kHz	66	132	264	NaN
120kHz	32	66	132	264

The `hNRReferenceWaveformGenerator` class also contains two constant properties which list the test model names for FR1 (TS 38.141-1 Section 4.9.2) and test model names for FR2 (TS 38.141-2 Section 4.9.2).

```
% Release 15 NR-TM test models for FR1 and FR2
```

```
fr1testmodels = hNRReferenceWaveformGenerator.FR1TestModels
```

```
fr1testmodels = 8x1 string
```

```
"NR-FR1-TM1.1"  
"NR-FR1-TM1.2"  
"NR-FR1-TM2"  
"NR-FR1-TM2a"  
"NR-FR1-TM3.1"  
"NR-FR1-TM3.1a"  
"NR-FR1-TM3.2"  
"NR-FR1-TM3.3"
```

```
fr2testmodels = hNRReferenceWaveformGenerator.FR2TestModels
```

```
fr2testmodels = 3x1 string
```

```
"NR-FR2-TM1.1"
```

```
"NR-FR2-TM2"  
"NR-FR2-TM3.1"
```

For the downlink FRCs, the class contains additional constant properties which list the downlink FRC names for FR1 (TS 38.101-1 Annex A.3) and for FR2 (TS 38.101-2 Annex A.3).

```
% Release 15 downlink fixed reference channels for FR1 and FR2  
fr1downlinkfrc = hNRReferenceWaveformGenerator.FR1DownlinkFRC
```

```
fr1downlinkfrc = 3x1 string  
"DL-FRC-FR1-QPSK"  
"DL-FRC-FR1-64QAM"  
"DL-FRC-FR1-256QAM"
```

```
fr2downlinkfrc = hNRReferenceWaveformGenerator.FR2DownlinkFRC
```

```
fr2downlinkfrc = 3x1 string  
"DL-FRC-FR2-QPSK"  
"DL-FRC-FR2-16QAM"  
"DL-FRC-FR2-64QAM"
```

For the uplink FRCs, the class contains two constant properties which list the uplink FRC names for FR1 and FR2 (TS 38.104 Annex A).

```
% Release 15 uplink fixed reference channels for FR1 and FR2  
fr1uplinkfrc = hNRReferenceWaveformGenerator.FR1UplinkFRC
```

```
fr1uplinkfrc = 89x1 string  
"G-FR1-A1-1"  
"G-FR1-A1-2"  
"G-FR1-A1-3"  
"G-FR1-A1-4"  
"G-FR1-A1-5"  
"G-FR1-A1-6"  
"G-FR1-A1-7"  
"G-FR1-A1-8"  
"G-FR1-A1-9"  
"G-FR1-A2-1"  
"G-FR1-A2-2"  
"G-FR1-A2-3"  
"G-FR1-A2-4"  
"G-FR1-A2-5"  
"G-FR1-A2-6"  
"G-FR1-A3-1"  
"G-FR1-A3-2"  
"G-FR1-A3-3"  
"G-FR1-A3-4"  
"G-FR1-A3-5"  
"G-FR1-A3-6"  
"G-FR1-A3-7"  
"G-FR1-A3-8"  
"G-FR1-A3-9"  
"G-FR1-A3-10"  
"G-FR1-A3-11"  
"G-FR1-A3-12"  
"G-FR1-A3-13"
```



```

"G-FR1-A3-14"
"G-FR1-A3-15"
:

fr2uplinkfrc = hNRReferenceWaveformGenerator.FR2UplinkFRC
fr2uplinkfrc = 37x1 string
"G-FR2-A1-1"
"G-FR2-A1-2"
"G-FR2-A1-3"
"G-FR2-A1-4"
"G-FR2-A1-5"
"G-FR2-A3-1"
"G-FR2-A3-2"
"G-FR2-A3-3"
"G-FR2-A3-4"
"G-FR2-A3-5"
"G-FR2-A3-6"
"G-FR2-A3-7"
"G-FR2-A3-8"
"G-FR2-A3-9"
"G-FR2-A3-10"
"G-FR2-A3-11"
"G-FR2-A3-12"
"G-FR2-A4-1"
"G-FR2-A4-2"
"G-FR2-A4-3"
"G-FR2-A4-4"
"G-FR2-A4-5"
"G-FR2-A4-6"
"G-FR2-A4-7"
"G-FR2-A4-8"
"G-FR2-A4-9"
"G-FR2-A4-10"
"G-FR2-A5-1"
"G-FR2-A5-2"
"G-FR2-A5-3"
:

```

For more information, access the help of `hNRReferenceWaveformGenerator` by typing `'doc hNRReferenceWaveformGenerator'`.

### NR-TM and PDSCH FRC Waveform Generation

Each PDSCH reference waveform is defined by a combination of:

- NR-TM or FRC name
- Channel bandwidth
- Subcarrier spacing
- Duplexing mode

Different NR-TMs are defined for FR1 and FR2. Depending on the test model purposes, NR-TMs have varying PDSCH characteristics. For example: full band, single modulation scheme, or full band, multiple modulation schemes with varying power boosting/deboosting or single, varying PRB allocation. Common features to all NR-TMs are: no SS burst, PDSCH mapping type A with one (FR2)

or two (FR1) DM-RS positions per slot transmission, and a single PDCCH across two symbols with NCCE = 1. There is no transport or DCI coding used and the input to the PDSCH and PDCCH is all 0's. FDD NR-TM waveforms are 10 ms in length and TDD cases are 20 ms. PT-RS are specified for FR2 NR-TM.

By comparison, downlink FRC waveforms contain transport coded PDSCH using RV = 0. The reference PDSCH are not defined in slots which overlap the SS burst (slot 0 or slots 0 and 1). They use front loaded PDSCH mapping type A with 2 additional DM-RS positions. There is no FDM between the PDSCH and the DM-RS. The full-band PDSCH start at symbol 2 and the first 2 symbols in a slot contain a full occupied CORESET. The FRC waveforms generated in this example do not contain additional OCNB. Power levels for all resource elements are uniform. The transport block data source is ITU PN9.

The channel bandwidth and subcarrier spacing combination have to be a valid pair from the associated FR bandwidth configuration table. The standard only defines FR2 NR-TM and FRC for TDD but with this example you can also create FDD waveforms.

This MATLAB code creates an `hNRReferenceWaveformGenerator` object for the selected NR-TM or FRC configuration. You can use this object to generate the associated baseband waveform and to display the underlying PRB and subcarrier-level resource grids.

`% Select the NR-TM or PDSCH FRC waveform parameters`

```
dlnrref = NR-FR1-TM3.2 (... ▾); % Model name and properties
bw      = 10MHz (FR1) ▾; % Channel bandwidth
scs     = 15kHz (FR1) ▾; % Subcarrier spacing
dm      = FDD ▾; % Duplexing mode
ncellid = 1 ▾; % NCellID
sv      = V15.2.0 ▾; % TS 38.141-x version (NR-TM only)
```

`% Run this entire section to generate the required waveform`

`Generate`

`% Create generator object for the above NR-TM/PDSCH FRC reference model`

```
dlnrefwavegen = hNRReferenceWaveformGenerator(dlnrref,bw,scs,dm,ncellid,sv)
```

```
dlnrefwavegen =
```

```
hNRReferenceWaveformGenerator with properties:
```

```
FR1BandwidthTable: [3x12 table]
FR2BandwidthTable: [2x4 table]
FR1TestModels: [8x1 string]
FR2TestModels: [3x1 string]
FR1DownlinkFRC: [3x1 string]
FR2DownlinkFRC: [3x1 string]
FR1UplinkFRC: [89x1 string]
FR2UplinkFRC: [37x1 string]
Config: [1x1 struct]
IsReadOnly: 1
ConfiguredModel: {1x6 cell}
```

```
% Generate waveform
[dlrefwaveform,dlrefwaveinfo,dlresourceinfo] = generateWaveform(dlrefwavegen);
```

```
% View transmission information about the set of PDSCH within the waveform
dlresourceinfo.WaveformResources.PDSCH
```

```
ans=1x3 struct array with fields:
    Name
    CDMLengths
    Resources
```

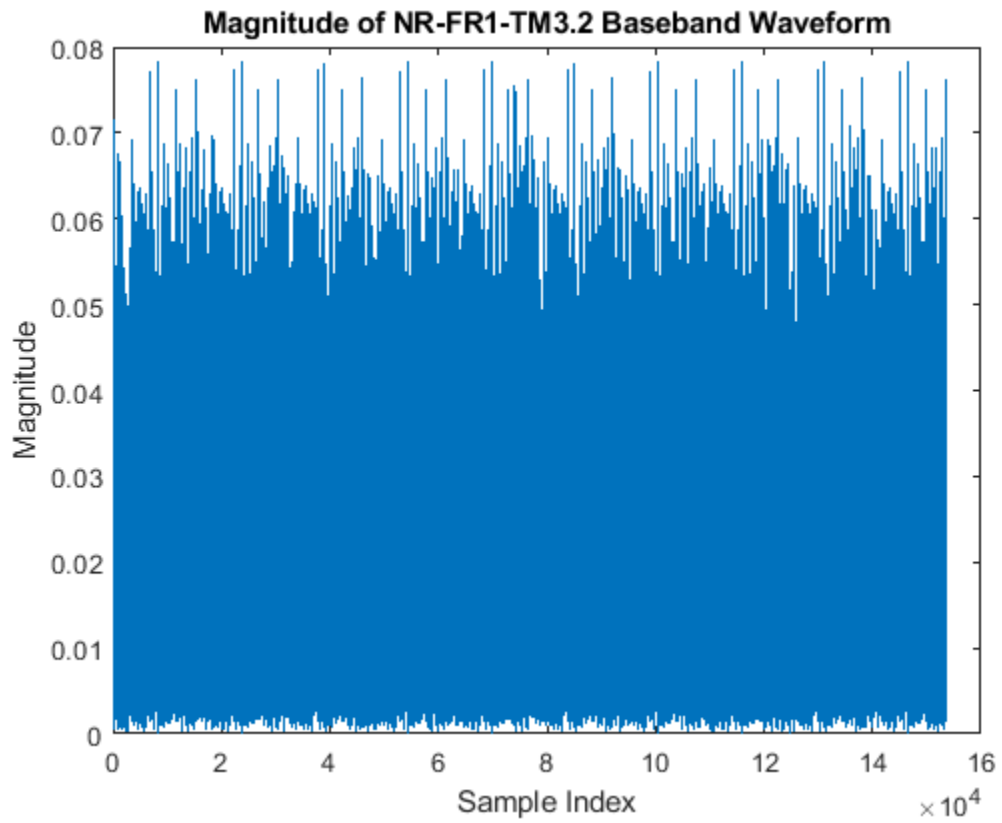
```
% View detailed information about one of the PDSCH sequences
dlresourceinfo.WaveformResources.PDSCH(1).Resources
```

```
ans=1x10 struct array with fields:
    NSlot
    TransportBlockSize
    TransportBlock
    RV
    Codeword
    G
    Gd
    ChannelIndices
    ChannelSymbols
    DMRSSymbolSet
    DMRSSymbols
    PTRSSymbolSet
    PTRSIndices
    PTRSSymbols
```

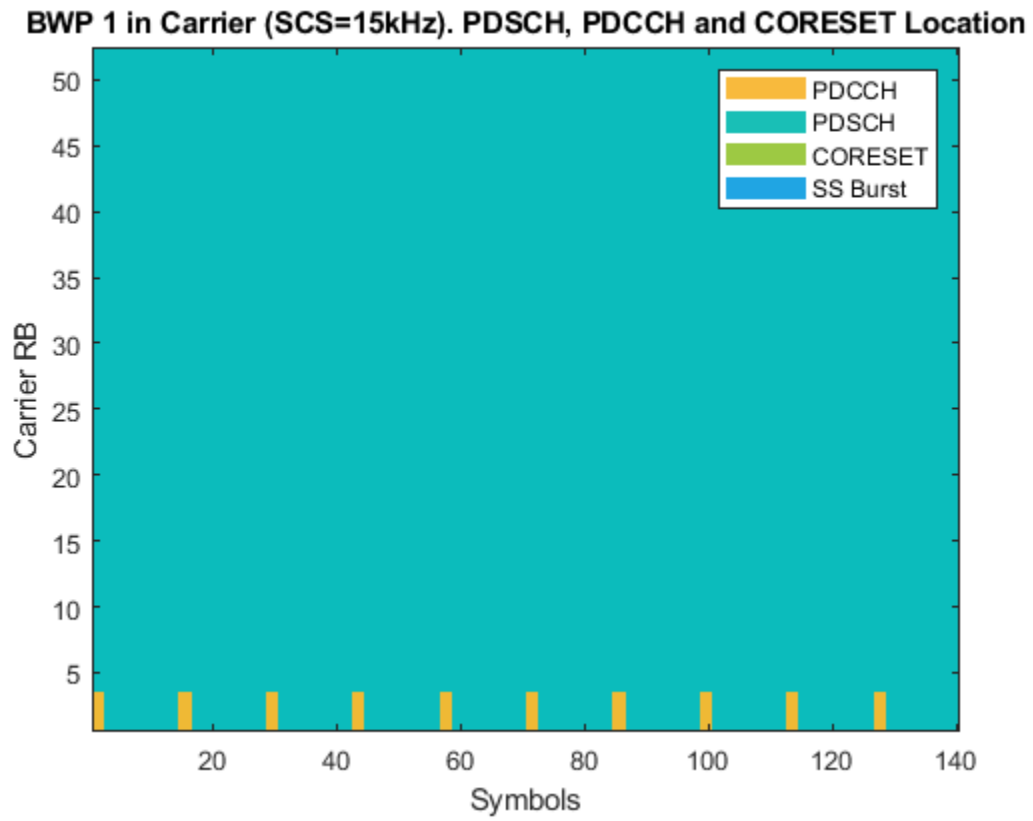
```
% Waveform sampling rate (Hz)
samplingrate = dlrefwaveinfo.Info.SamplingRate

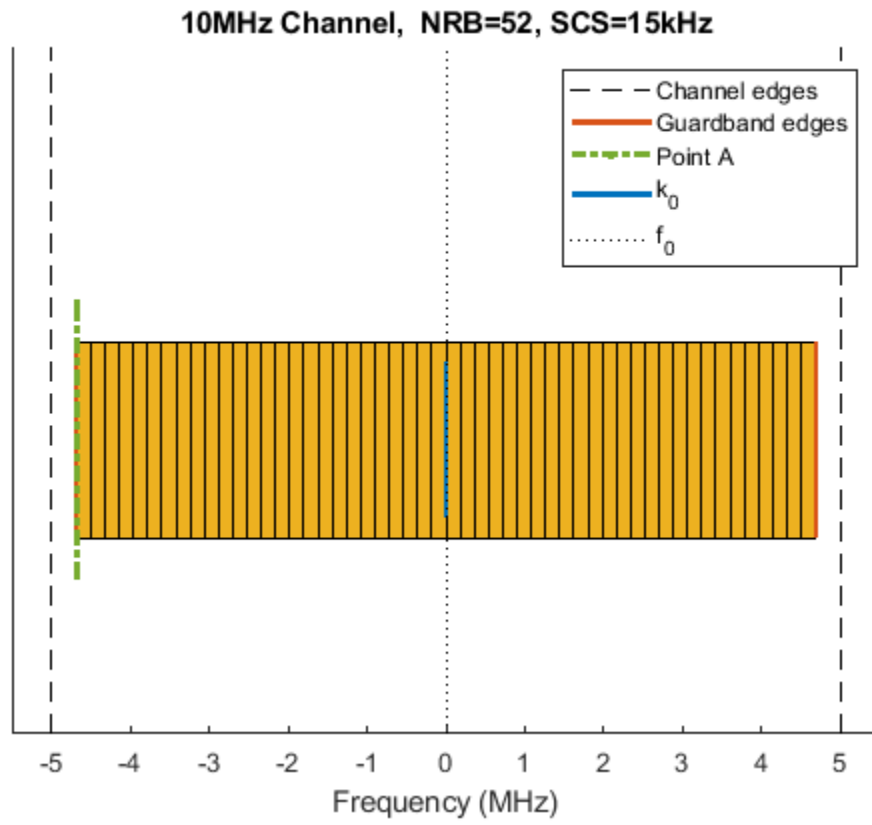
samplingrate = 15360000
```

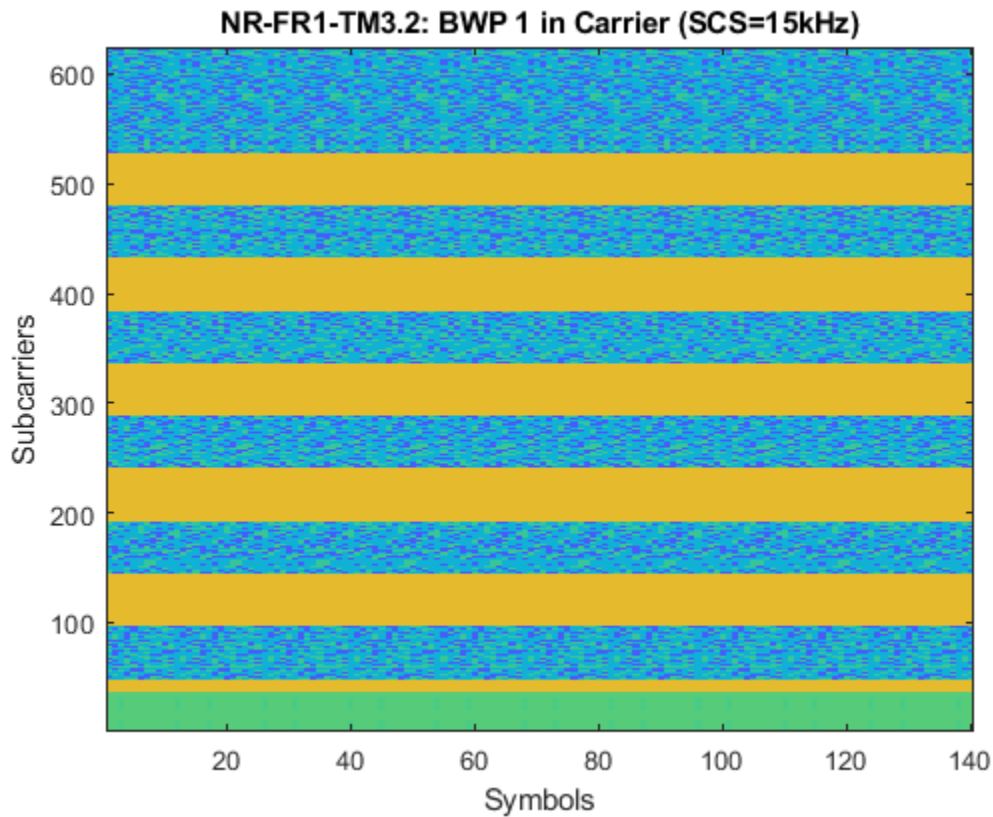
```
plot(abs(dlrefwaveform)); title(sprintf('Magnitude of %s Baseband Waveform',dlnrref)); xlabel('S
```



```
% Visualize the associated PRB and subcarrier resource grids  
displayResourceGrid(dlrefwavegen);
```







```

fullparameterset = dlrefwavegen.Config % Full low-level parameter set

fullparameterset = struct with fields:
    Name: "NR-FR1-TM3.2"
    NCellID: 1
    ChannelBandwidth: 10
    FrequencyRange: "FR1"
    NumSubframes: 10
    Windowing: []
    DisplayGrids: 0
    SSBurst: [1x1 struct]
    Carriers: [1x1 struct]
    BWP: [1x1 struct]
    CORESET: [1x1 struct]
    PDCCH: [1x1 struct]
    CSIRS: [1x1 struct]
    PDSCH: [1x3 struct]

% Make the Config parameters writable and boost the power on all PDSCH DM-RS
dlrefwavegen = makeConfigWritable(dlrefwavegen)

dlrefwavegen =
    hNRReferenceWaveformGenerator with properties:
        FR1BandwidthTable: [3x12 table]
        FR2BandwidthTable: [2x4 table]

```

```

    FR1TestModels: [8x1 string]
    FR2TestModels: [3x1 string]
    FR1DownlinkFRC: [3x1 string]
    FR2DownlinkFRC: [3x1 string]
    FR1UplinkFRC: [89x1 string]
    FR2UplinkFRC: [37x1 string]
    Config: [1x1 struct]
    IsReadOnly: 0
    ConfiguredModel: {1x6 cell}

```

```
[dlrefwavegen.Config.PDSCH.PowerDMRS] = deal(3);
```

### PUSCH FRC Waveform Generation

Each PUSCH FRC reference channel definition in TS 38.104 Annex A explicitly defines a number of key parameters including:

- Frequency range
- Channel bandwidth
- Subcarrier spacing
- Code rate
- Modulation
- DM-RS configuration

Additionally, the associated receiver tests introduce some additional parameters that are not specified in the TS 38.104 Annex A tables, for example, the general test parameters defined in:

- Table 8.2.1.1-1 (Conducted performance requirements for PUSCH without transform precoding)
- Table 8.2.2.1-1 (Conducted performance requirements for PUSCH with transform precoding)
- Table 11.2.2.1.1-1 (Radiated performance requirements for BS type 2-O for PUSCH without transform precoding)
- Table 11.2.2.2.1-1 (Radiated performance requirements for BS type 2-O for PUSCH with transform precoding)

The parameter sets that are captured in the MATLAB reference waveform generator use the above specification sources. Since a given FRC can be used in different tests with different parameters requirements, the following general rules apply to the default generator configurations. All parameters can be modified after construction. Transform precoding is enabled for appropriate FRC. FR2 waveforms are TDD and 20ms in length, and FR1 waveforms are FDD and 10ms. The PUSCH FRC are defined with type A mapping, type B mapping, or, in some cases, either mapping type. In the latter case, type A mapping is configured. FR2 waveforms without transform precoding are configured with PT-RS, otherwise PT-RS are off. Scrambling identities are set to 0. Power levels for all resource elements are uniform. The transport block data source is ITU PN9 with RV = 0 i.e. no retransmissions.

This MATLAB code creates an `hNRReferenceWaveformGenerator` object for the selected PUSCH FRC configuration. Due to the large number of FRC, the live script FRC drop-down lists only those from section TS 38.104 A.1 (reference sensitivity, ACS, in-band blocking etc.) and A.2 (dynamic range). The performance test FRC defined in A.3, A.4, A.5 can be chosen by specifying the FRC name string directly in the code below. After the generator object is created, all configuration parameters can be changed by making them writable using the `makeConfigWritable` function.



```

% Select the PUSCH FRC waveform
ulnrref = ; % This live script down-drop list is preconfigured for TS 38.1

% Possible overrides to Annex A definitions (empty values provide the Annex A defaults)
bw      = []; % Bandwidth override (5,10,15,20,25,30,40,50,60,80,90,100,200,400 MHz)
scs     = []; % Subcarrier spacing override (15,30,60,120 kHz)
dm      = []; % Duplexing mode override ("FDD","TDD")
ncellid = []; % Cell identity override (used to control scrambling identities)

% Run this entire section to generate the required waveform


% Create generator object for the above PUSCH FRC reference model
ulrefwavegen = hNRReferenceWaveformGenerator(ulnrref,bw,scs,dm,ncellid)

ulrefwavegen =
  hNRReferenceWaveformGenerator with properties:
    FR1BandwidthTable: [3x12 table]
    FR2BandwidthTable: [2x4 table]
    FR1TestModels: [8x1 string]
    FR2TestModels: [3x1 string]
    FR1DownlinkFRC: [3x1 string]
    FR2DownlinkFRC: [3x1 string]
    FR1UplinkFRC: [89x1 string]
    FR2UplinkFRC: [37x1 string]
    Config: [1x1 struct]
    IsReadOnly: 1
    ConfiguredModel: {"G-FR1-A1-1"} [] [] ["FDD"] [0]}

% Generate waveform
[ulrefwaveform,ulrefwaveinfo,ulresourceinfo] = generateWaveform(ulrefwavegen);

% View transmission information about the set of PUSCH within the waveform
ulresourceinfo.WaveformResources.PUSCH

ans = struct with fields:
    Name: "PUSCH sequence for G-FR1-A1-1"
    CDMLengths: [1 1]
    Resources: [1x10 struct]

% View detailed information about one of the PUSCH sequences
ulresourceinfo.WaveformResources.PUSCH(1).Resources

ans=1x10 struct array with fields:
    NSlot
    TransportBlockSize
    TransportBlock
    RV
    Codeword
    G
    Gd
    ChannelIndices
    ChannelSymbols

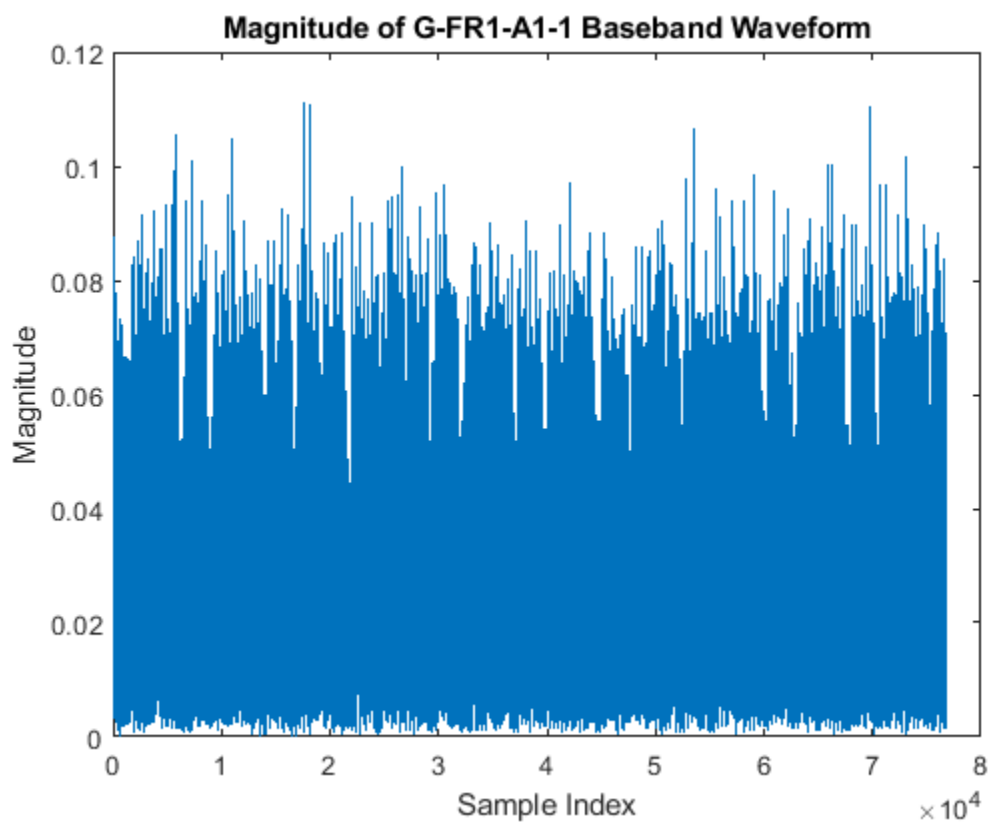
```

```
DMRSSymbolSet
DMRSIndices
DMRSSymbols
PTRSSymbolSet
PTRSIndices
PTRSSymbols

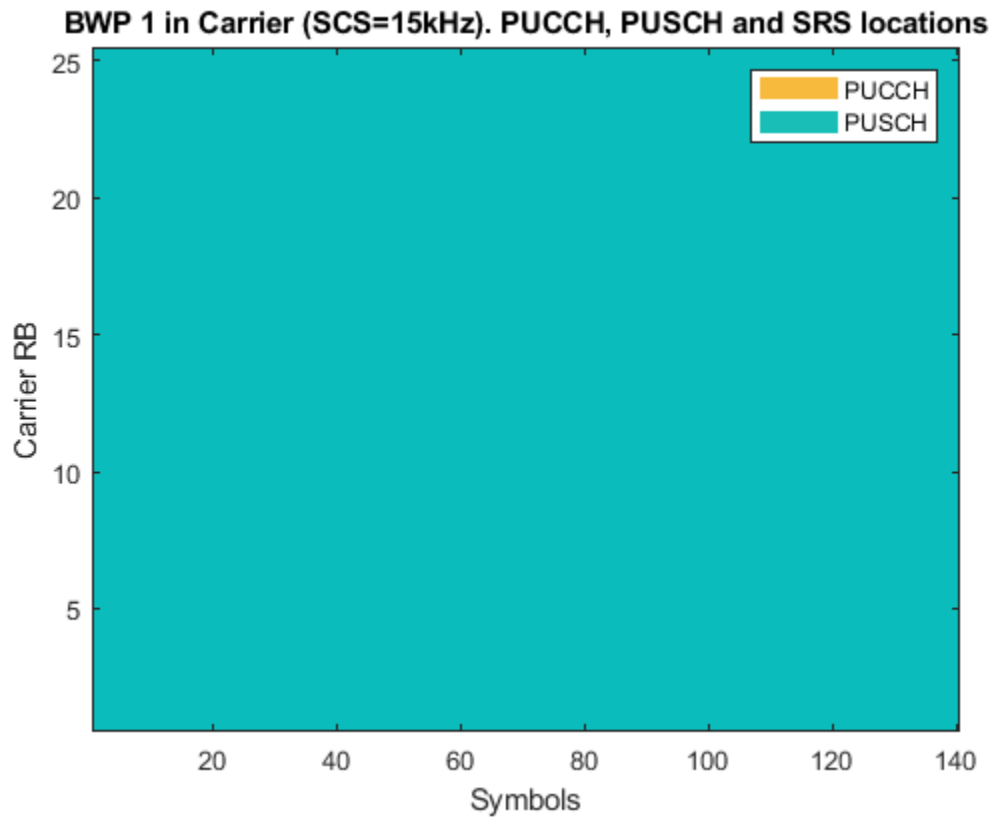
% Waveform sampling rate (Hz)
samplingrate = ulrefwaveinfo.Info.SamplingRate

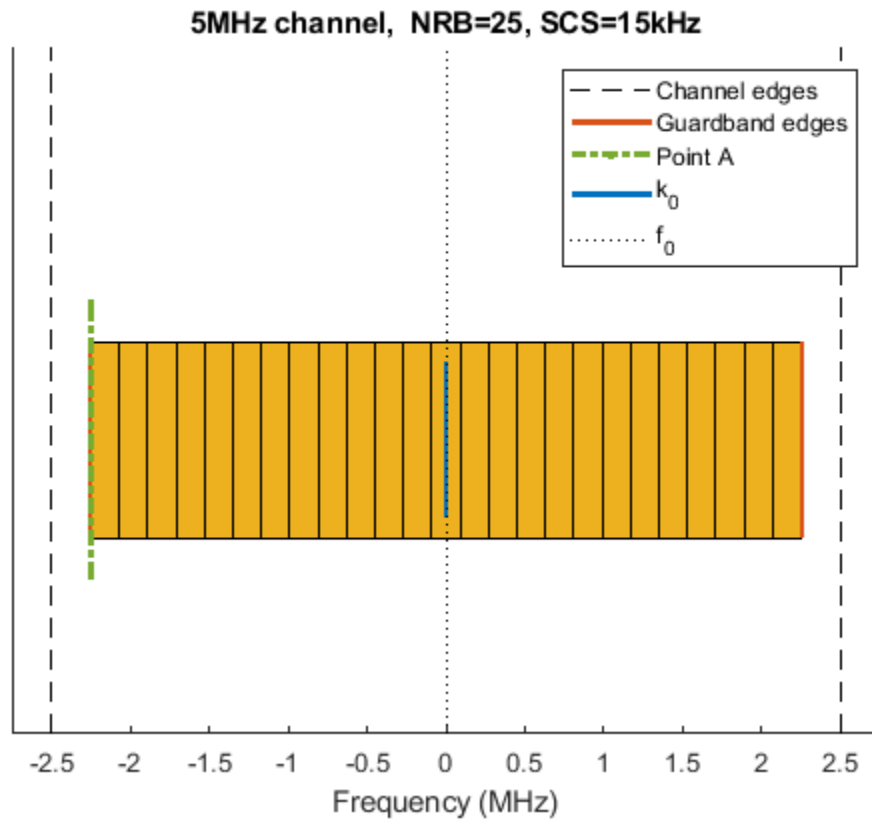
samplingrate = 7680000

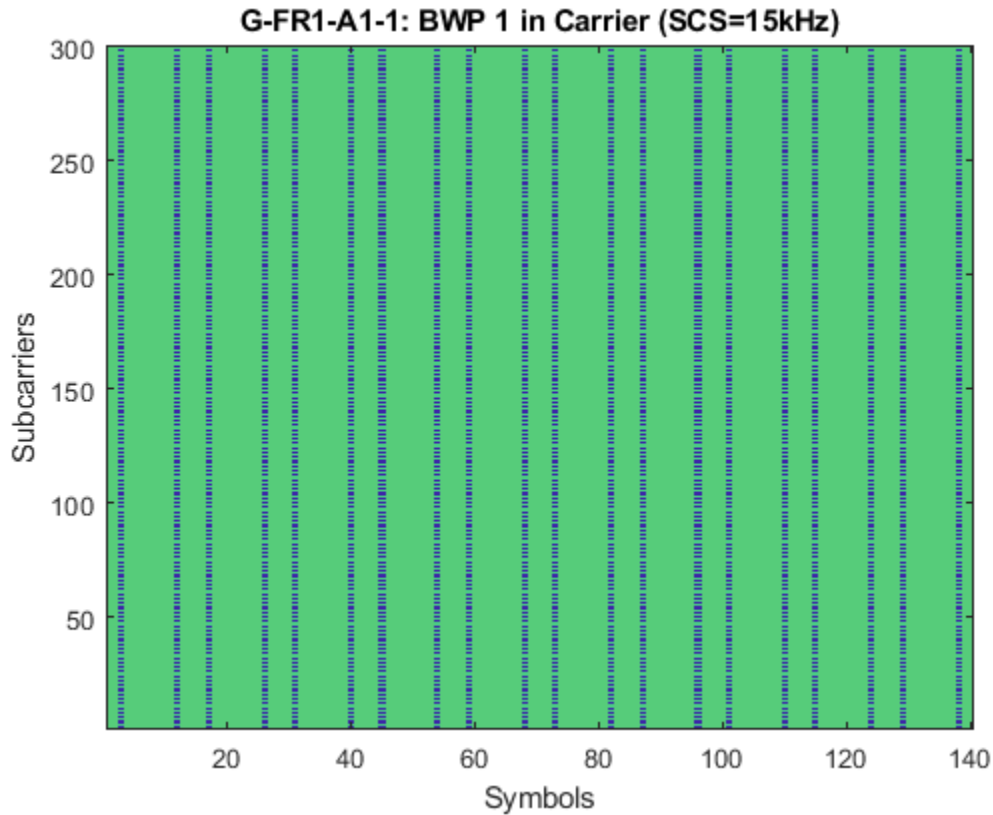
plot(abs(ulrefwaveform)); title(sprintf('Magnitude of %s Baseband Waveform',ulnrref)); xlabel('S
```



```
% Visualize the associated PRB and subcarrier resource grids
displayResourceGrid(ulrefwavegen);
```







```
fullparameterset = ulrefwavegen.Config % Full low-level parameter set
```

```
fullparameterset = struct with fields:
```

```
    Name: 'G-FR1-A1-1'
    NCellID: 0
    ChannelBandwidth: 5
    FrequencyRange: "FR1"
    NumSubframes: 10
    Windowing: []
    DisplayGrids: 0
    Carriers: [1x1 struct]
    BWP: [1x1 struct]
    PUCCH: [1x1 struct]
    PUSCH: [1x1 struct]
```

## 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 38.141-1. "NR; Base Station (BS) conformance testing Part 1: Conducted conformance testing." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[5] 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*.

## **See Also**

### **Apps**

#### **5G Waveform Generator**

## **Related Examples**

- "5G NR Downlink Carrier Waveform Generation" on page 1-2
- "5G NR Uplink Carrier Waveform Generation" on page 2-2
- "App-Based 5G NR-TM and FRC Waveform Generation" on page 6-19

## App-Based 5G NR-TM and FRC Waveform Generation

This example shows how to use the **5G Waveform Generator** app to generate NR test models (NR-TM) and NR uplink and downlink fixed reference channel (FRC) waveforms.

### 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

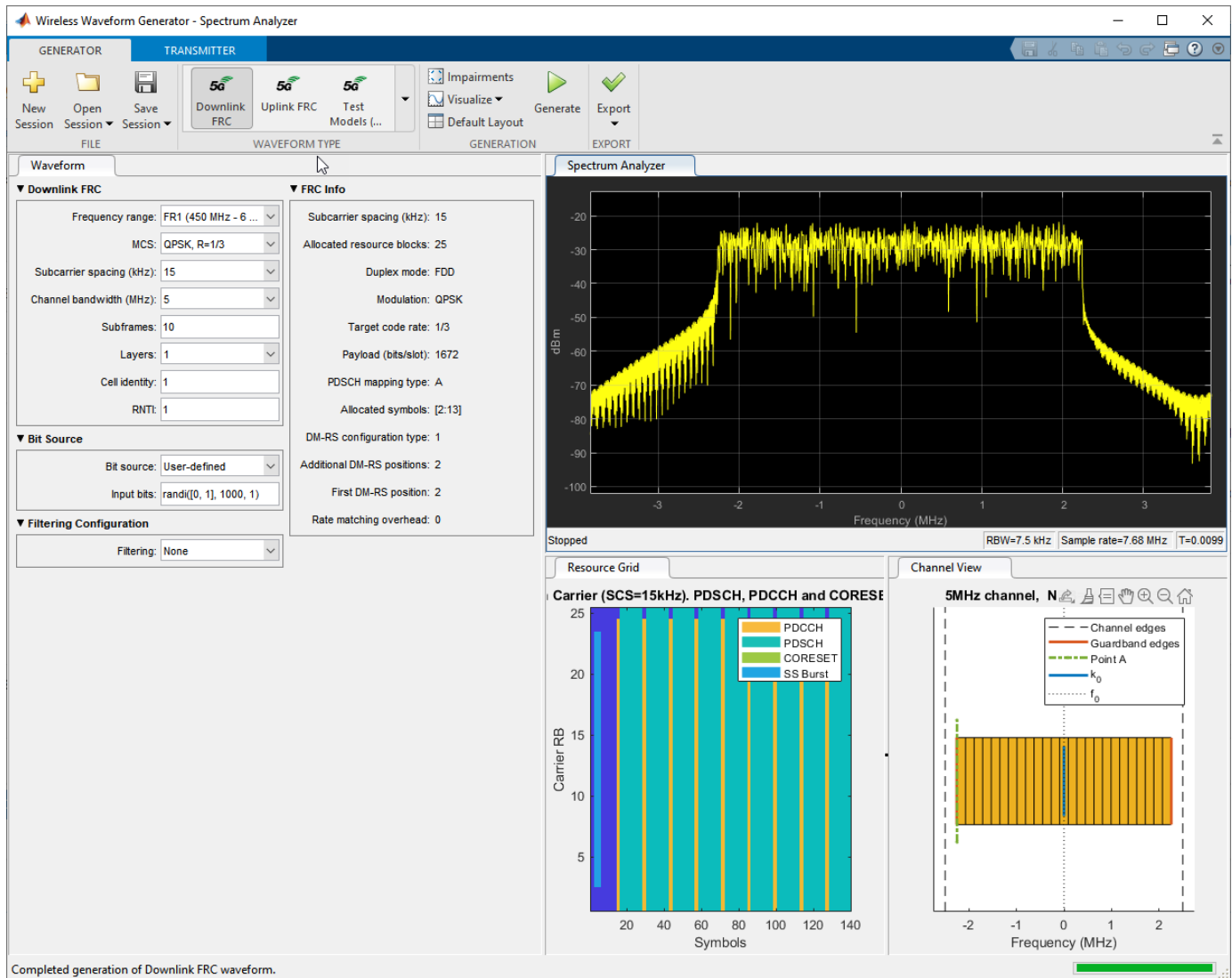
In the **Waveform Type** section on the app toolstrip, click the waveform you want to generate. You can select any of these waveforms.

- 5G Downlink FRC
- 5G Uplink FRC
- 5G Test Models

### Generate 5G NR Waveform

In the left pane of the app, on the **Waveform** tab, 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. To visualize the waveform, click **Generate**. You can export the generated waveform to the MATLAB workspace as a structure in a `.mat` or `.bb` file.

For example, this picture shows the visualization results of a downlink FRC waveform using default parameters.



## Transmit 5G NR Waveform

This feature requires “Instrument Control Toolbox”. To transmit the generated waveform, on the app toolstrip, click on the **Transmitter** tab and set up the instruments. You can use all the instruments supported by the `rfsiggen` (Instrument Control Toolbox) function.

## See Also

**Apps**  
5G Waveform Generator

## Related Examples

- “5G NR-TM and FRC Waveform Generation” on page 6-2



## **More About**

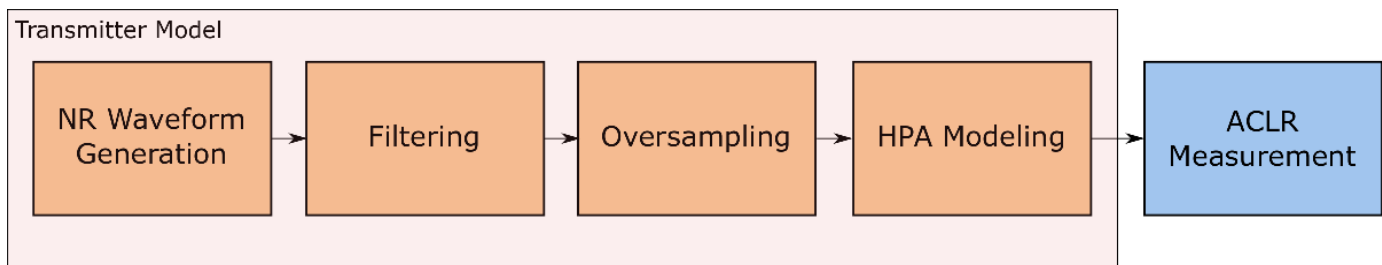
- “Using Wireless Waveform Generator App”

## 5G NR Downlink ACLR Measurement

This example shows how to measure the adjacent channel leakage ratio (ACLR) for 5G NR test models (NR-TMs) in frequency range 1 (FR1) and FR2 using 5G Toolbox™.

### Introduction

The ACLR is the ratio of the filtered mean power centered on the assigned channel frequency to the filtered mean power centered on an adjacent channel frequency. This example performs ACLR measurements for an NR downlink waveform, as defined in TS 38.104 Section 6.6.3. To model the effect of out-of-band spectral emissions, the example applies spectral regrowth on an oversampled waveform by using a high power amplifier (HPA) model.



### Generate NR-TM Waveform

Use the MATLAB class `hNRReferenceWaveformGenerator` to generate 5G NR-TMs for FR1 and FR2. You can generate the NR-TM waveforms by specifying these parameters:

- NR-TM name
- Channel bandwidth
- Subcarrier spacing
- Duplexing mode

For more information, see the “5G NR-TM and FRC Waveform Generation” on page 6-2 example.

`% Select the NR-TM waveform parameters`

```

nrtm = NR-FR1-TM1.2 (...); % NR-TM name and properties
bw = 20MHz (FR1); % Channel bandwidth
scs = 15kHz (FR1); % Subcarrier spacing
dm = FDD; % Duplexing mode
  
```

`% Create generator object for the above NR-TM`

```
tmWaveGen = hNRReferenceWaveformGenerator(nrtm,bw,scs,dm);
```

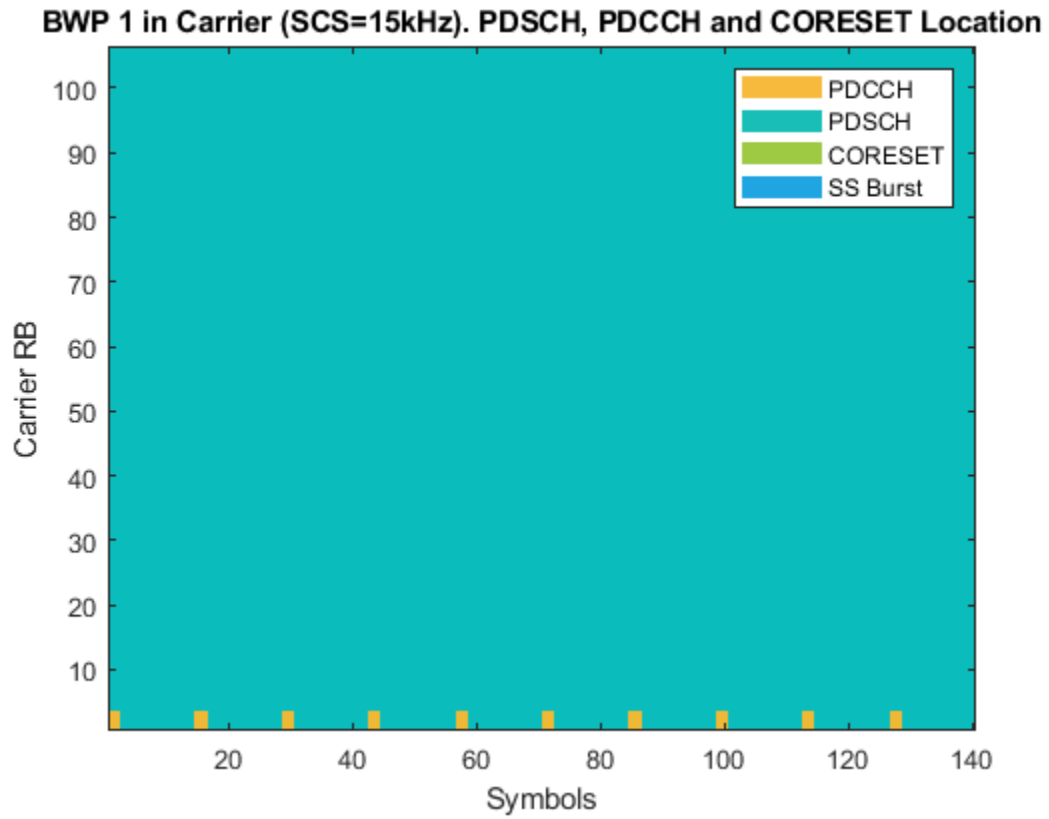
`% Select no windowing to highlight impact of filtering on ACLR`

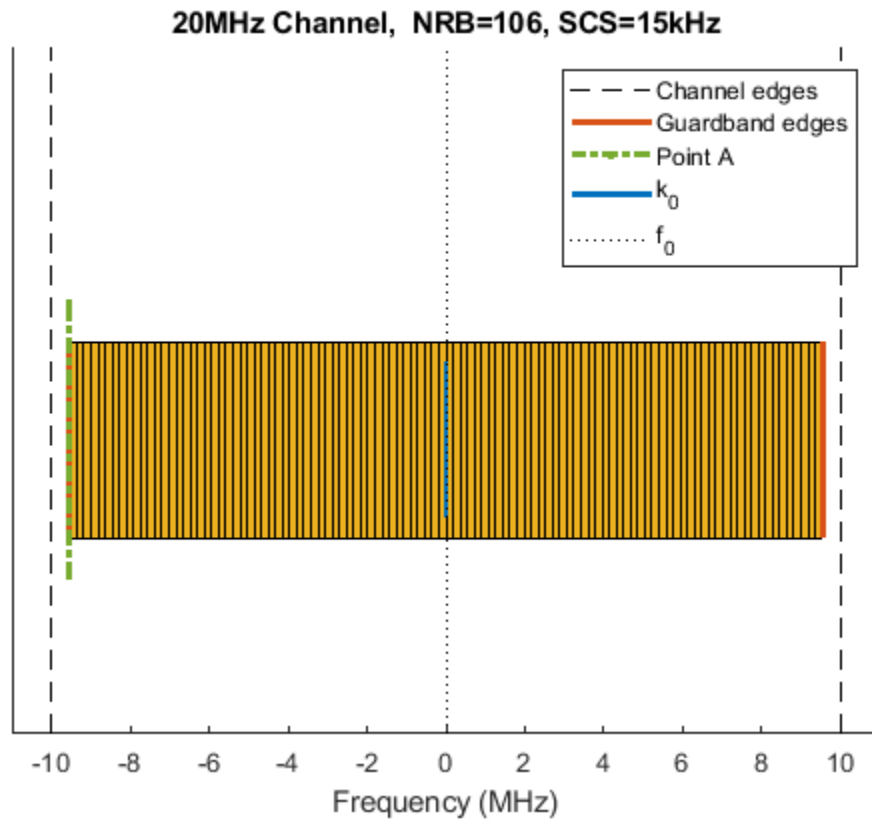
```
tmWaveGen = makeConfigWritable(tmWaveGen);
tmWaveGen.Config.Windowing = 0;
```

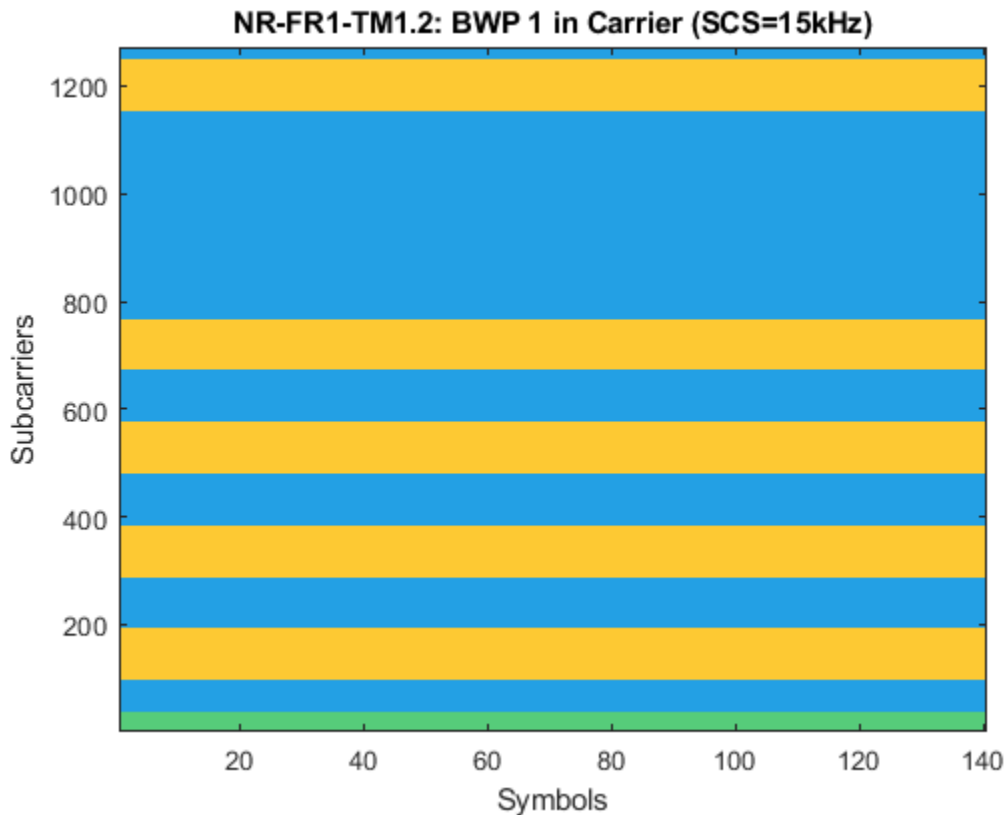
`% Generate waveform`

```
[tmWaveform,tmWaveInfo] = generateWaveform(tmWaveGen);
samplingRate = tmWaveInfo.Info.SamplingRate; % Waveform sampling rate (Hz)
```

```
% Visualize the associated PRB and subcarrier resource grids  
displayResourceGrid(tmWaveGen);
```







### Display Results

- The associated PRB resource grid (top) depicts the allocation of the different components (PDCCH, PDSCH, CORESET and SS Burst) in each BWP. The grid does not plot the amplitude of the signals only the signal locations in the grid.
- The SCS specific carrier resource grids (middle) along with the minimum guardbands, aligned relative to the overall channel bandwidth.
- The subcarrier resource grid (bottom) indicates the amplitude levels of the generated waveform. If just one color is shown, all the components have the same amplitude.

### Calculate ACLR Parameters

The helper function `hACLRParametersNR.m` calculates the parameters required for ACLR measurement.

The function determines the required oversampling. If the input waveform sampling rate (`samplingRate`) is not sufficient to span the entire bandwidth (`aclr.BandwidthACLR`) of the adjacent channels (allowing for a maximum of 85% bandwidth occupancy), you can use an upsampled version of the waveform for ACLR calculations. The upsampling factor is `aclr.OSR`.

```
aclrParameters = hACLRParametersNR(tmWaveGen.Config);
disp(aclrParameters);
```

```
Bandwidth: 20000000
SubcarrierSpacing: 15000
BandwidthConfig: 19080000
```

```

BandwidthACLR: 100000000
OSR: 4
SamplingRate: 122880000

```

### Filter Waveform to Improve ACLR

The generated waveform has no filtering, so there are significant out-of-band spectral emissions owing to the implicit rectangular pulse shaping in the OFDM modulation (each OFDM subcarrier has a sinc shape in the frequency domain). Filtering the waveform improves ACLR performance.

Design a filter with a transition band that starts at the edge of the occupied transmission bandwidth (`acLR.BandwidthConfig`) and stops at the edge of the overall channel bandwidth (`acLR.Bandwidth`). This filter involves no rate change, it just shapes the spectrum within the original bandwidth of the waveform.

```

% Design filter
lpFilt = designfilt('lowpassfir',...
    'PassbandFrequency',acLRParameters.BandwidthConfig/2,...
    'StopbandFrequency',acLRParameters.Bandwidth/2,...
    'PassbandRipple',0.1,...
    'StopbandAttenuation',80,...
    'SampleRate',samplingRate);

% Apply filter
filtTmWaveform = filter(lpFilt,tmWaveform);

```

### Oversampling and HPA Nonlinearity Model

To create a signal capable of representing 1st and 2nd adjacent carriers, for example, to represent `acLR.BandwidthACLR` with at most 85% bandwidth occupancy, oversample the NR waveform. After oversampling the signal, employ an HPA model to generate out-of-band distortion. For example, to simulate the HPA behaviour, you can use the Rapp method, which is widely used in wireless applications to generate AM/AM distortion. In MATLAB®, you can use the Memoryless Nonlinearity object to model the Rapp method. To highlight the impact of filtering on the ACLR measurements, apply the oversampling and HPA nonlinearities first to the filtered NR signal and then to the same NR signal without filtering.

```

% Apply required oversampling
resampled = resample(tmWaveform,acLRParameters.OSR,1);           % Not filtered
filtResampled = resample(filtTmWaveform,acLRParameters.OSR,1); % Filtered

% Create and configure a memoryless nonlinearity to model the amplifier
nonLinearity = comm.MemorylessNonlinearity;
nonLinearity.Method = 'Rapp model';
nonLinearity.Smoothness = 3;           % p parameter
nonLinearity.LinearGain = 0.5;         % dB
nonLinearity.OutputSaturationLevel = 2; % It limits the output signal level

% Signal conditioning to control the HPA input back-off level
resampled = resampled/max(abs(resampled)); % Not filtered
filtResampled = filtResampled/max(abs(filtResampled)); % Filtered

% Apply the amplifier model to the NR waveforms
txWaveform = nonLinearity(resampled); % Not filtered
txFiltWaveform = nonLinearity(filtResampled); % Filtered

```

## Calculate NR ACLR

The `hACLRMeasurementNR.m` helper function measures the NR ACLR using a square window on adjacent channels. This function takes a DFT of the measurement signal and uses the energy of the appropriate bins to calculate the adjacent channel powers.

```
% Calculate NR ACLR
aclr = hACLRMeasurementNR(aclrParameters,txWaveform);           % Not filtered
filtAclr = hACLRMeasurementNR(aclrParameters,txFiltWaveform); % Filtered
```

The `hACLRMeasurementNR.m` helper function returns the ACLR measurements in a structure with these fields:

- **Bandwidth:** the channel bandwidth associated with `tmWaveform`, in Hertz. This is the overall bandwidth of the assigned channel.
- **SubcarrierSpacing:** the subcarrier spacing associated with `tmWaveform`, in Hertz.
- **BandwidthConfig:** the transmission bandwidth configuration associated with `tmWaveform`, in Hertz. This is the bandwidth within the channel bandwidth that contains active subcarriers.
- **BandwidthACLR:** the bandwidth required to represent 1st and 2nd adjacent carriers; the sampling rate used internally for ACLR measurements will support this bandwidth with at most 85% bandwidth occupancy.
- **OSR:** the integer oversampling ratio of the input `tmWaveform` required to create a signal capable of representing 1st and 2nd adjacent carriers.
- **SamplingRate:** the sampling rate of the internal measurement signal from which the ACLR is calculated. If `OSR = 1`, this signal is the input waveform; if `OSR > 1`, this signal is the input waveform upsampled by `OSR`. Therefore: `aclr.SamplingRate = OSR*samplingRate`.
- **CarrierFrequency:** a vector of NR center frequencies, in Hertz, for adjacent channels `[-2,-1,1,2]`.
- **SignalPowerdBm:** the power, in decibels relative to 1 mW in 1 ohm, of the input within the NR channel of interest, for example in a square filter of bandwidth `aclr.BandwidthConfig` centered at 0 Hz.
- **ACLRdB:** a vector of NR ACLRs, in decibels relative to `aclr.SignalPowerdBm`, measured for adjacent channels `[-2,-1,1,2]`.

## Display Results

The `hACLRResultsNR.m` helper function displays the NR ACLR and plots the NR spectrum and the adjacent channel leakage ratios.

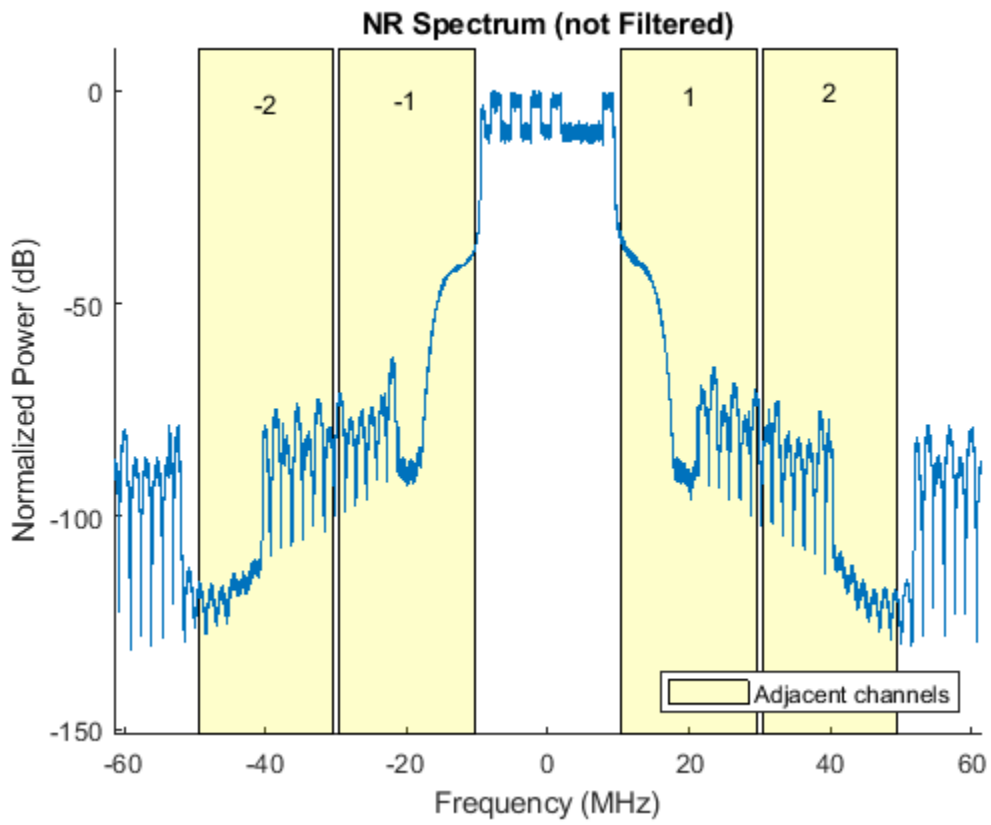
### Not Filtered

According to TS 38.104 Section 6.6.3.2, the minimum required ACLR for conducted measurements is 45 dB. As some of these ACLR values are lower than 45 dB, they do not fall within the requirements.

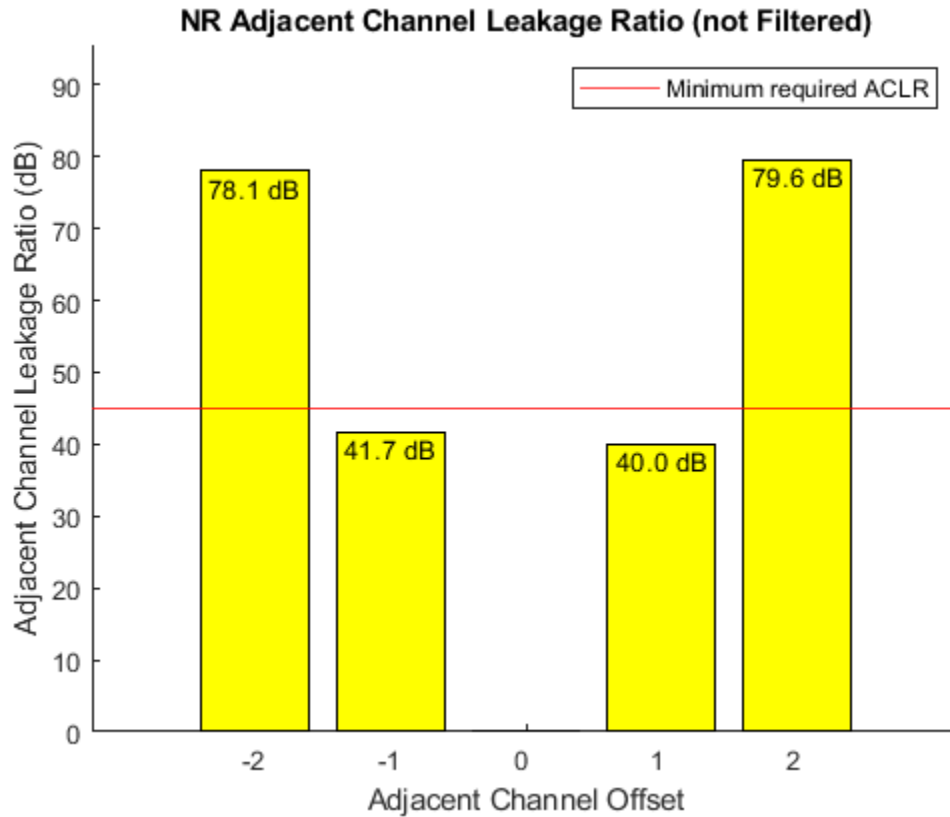
```
hACLRResultsNR(aclr,txWaveform,'(not Filtered)');

    Bandwidth: 20000000
SubcarrierSpacing: 15000
    BandwidthConfig: 19080000
    BandwidthACLR: 100000000
           OSR: 4
    SamplingRate: 122880000
CarrierFrequency: [-40000000 -20000000 20000000 40000000]
```

SignalPowerdBm: 19.7642  
ACLRdB: [78.1435 41.6526 40.0279 79.6265]



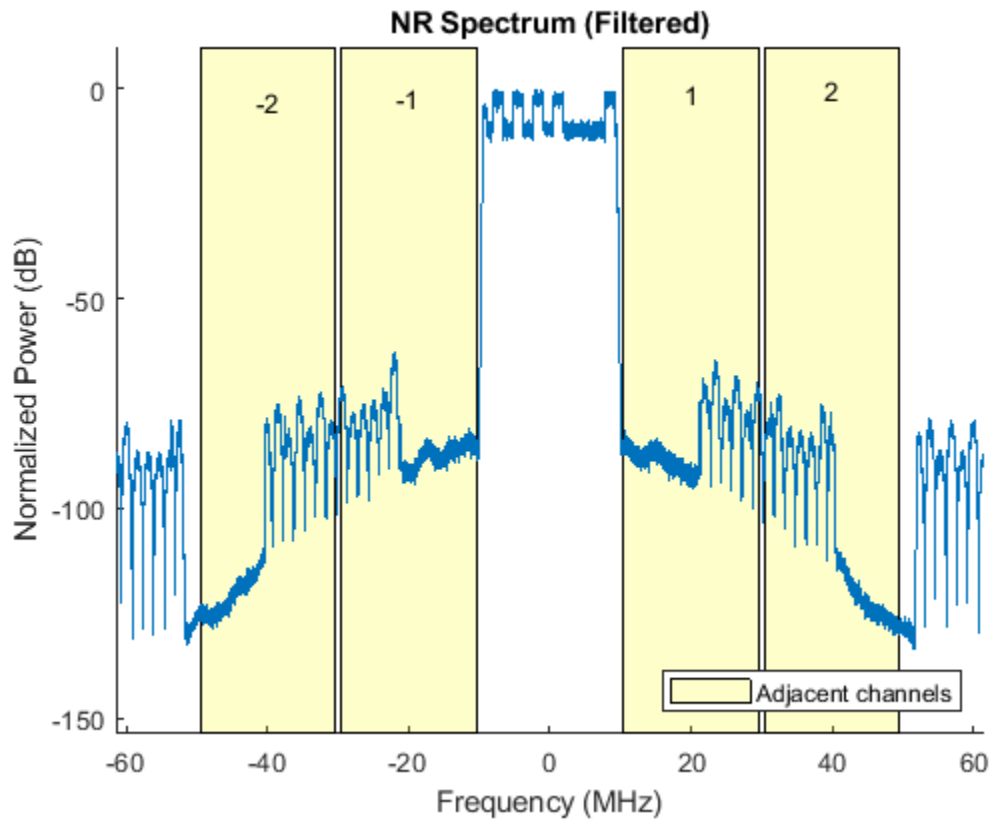


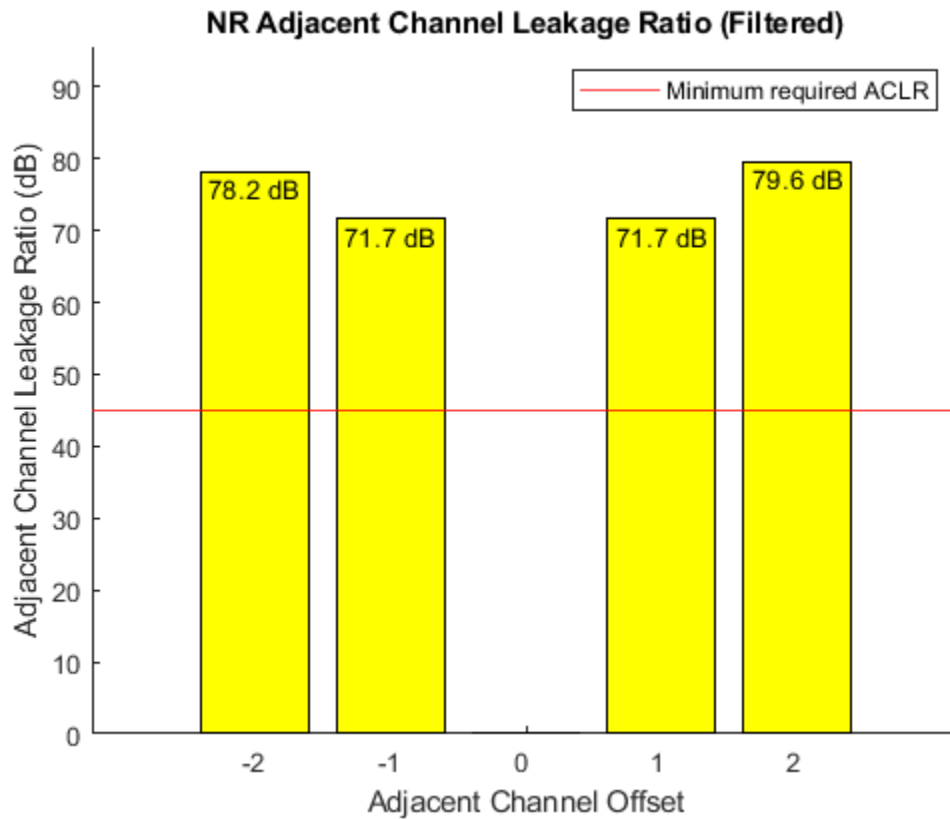


### Filtered

The performance improves when the generated waveform is filtered. The ACLR results with the filtered waveform are higher than the minimum required value.

```
hACLRResultsNR(filtAclr,txFiltWaveform,'(Filtered)');
    Bandwidth: 20000000
SubcarrierSpacing: 15000
BandwidthConfig: 19080000
BandwidthACLR: 100000000
    OSR: 4
    SamplingRate: 122880000
CarrierFrequency: [-40000000 -20000000 20000000 40000000]
SignalPowerBm: 19.6869
    ACLRdB: [78.1614 71.6984 71.6686 79.6094]
```





## Appendix

This example uses the following helper functions and classes:

- hACLRMeasurementNR.m
- hACLRParametersNR.m
- hACLRResultsNR.m
- hNRReferenceWaveformGenerator.m

## References

- [1] 3GPP TS 38.104. "NR; Base Station (BS) radio transmission and reception." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network.*

## See Also

## Related Examples

- "5G NR-TM and FRC Waveform Generation" on page 6-2

## Modeling and Testing an NR RF Transmitter

The example shows how to characterize the impact of RF impairments, such as in-phase and quadrature (IQ) imbalance, phase noise, and power amplifier (PA) nonlinearities in the performance of a new radio (NR) radio frequency (RF) transmitter. The NR RF transmitter is modeled in Simulink® using 5G Toolbox™ and RF Blockset™.

### Introduction

This example shows how to characterize the impact of RF impairments such as IQ imbalance, phase noise, and PA nonlinearities in the performance of an NR RF transmitter. To evaluate the performance, the example considers these measurements:

- Error vector magnitude (EVM): vector difference at a given time between the ideal (transmitted) signal and the measured (received) signal.
- Adjacent channel leakage ratio (ACLR): measure of the amount of power leaking into adjacent channels and is defined as the ratio of the filtered mean power centered on the assigned channel frequency to the filtered mean power centered on an adjacent channel frequency.
- Occupied bandwidth: bandwidth that contains 99% of the total integrated power of the signal, centered on the assigned channel frequency.
- Channel power: filtered mean power centered on the assigned channel frequency.
- Complementary cumulative distribution function (CCDF): probability of a signal's instantaneous power to be a level specified above its average power.

The model works on a subframe by subframe basis. For each subframe, the workflow consists of these steps:

- 1 Generate the baseband waveform using 5G Toolbox functions.
- 2 Upconvert the generated waveform to the passband frequency and apply RF filtering and amplification using RF Blockset.
- 3 Downconvert the transmitted waveform to baseband frequency.
- 4 Calculate the ACLR/ACPR, occupied bandwidth, channel power, and CCDF using the Spectrum Analyzer block.
- 5 Demodulate the waveform at the receiver to measure EVM.

The example uses a Simulink model to perform these operations. Baseband signal processing (steps 1 and 5) uses MATLAB® Function blocks, whereas the RF transmitter modeling (steps 2-4) uses RF Blockset. This model supports Normal and Accelerator simulation modes.

### Simulink Model Structure

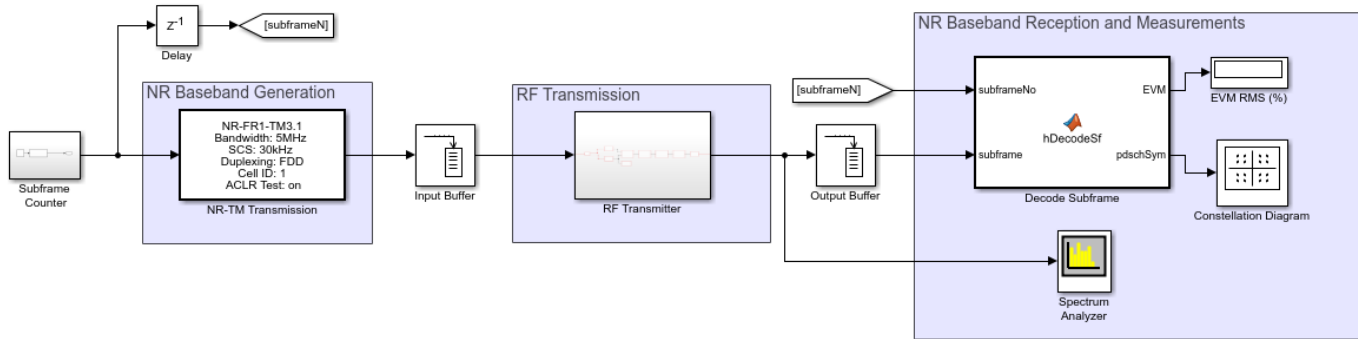
The model contains three main components:

- NR Baseband Generation: generates the baseband waveform.
- RF Transmission: upconverts the waveform from baseband to passband, applies RF filtering, and amplification and performs downconversion.
- NR Baseband Reception and Measurements: performs the RF measurements and demodulates the baseband waveform to calculate the EVM.

```
modelName = 'NRModelingAndTestingRFTransmitterModel';  
open_system(modelName);
```

### Modeling and Testing an NR RF Transmitter

Input signal: NR Test Model  
 Tests: EVM, ACLR, occupied bandwidth, channel power and CCDF

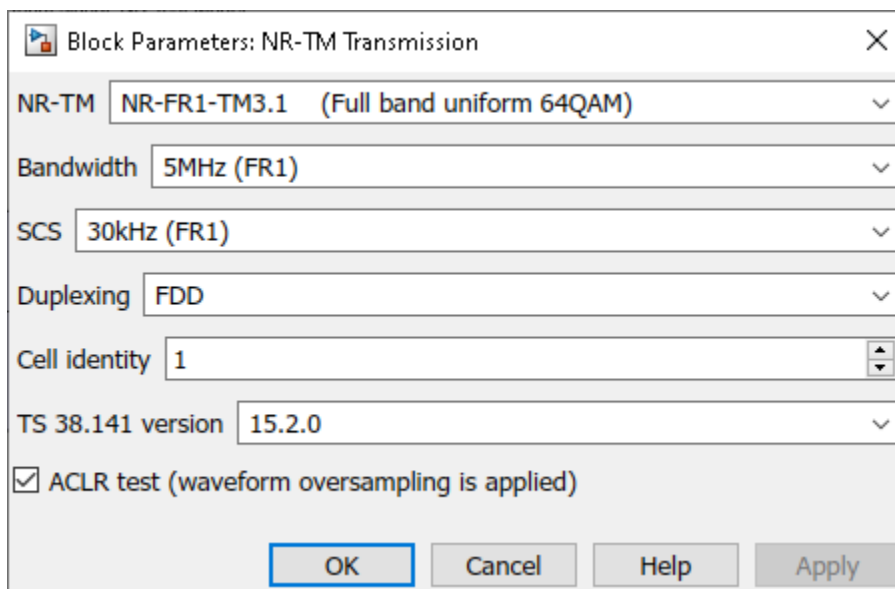


Copyright 2019-2020 The MathWorks, Inc.

### NR Baseband Generation

The NR-TM Transmission block transmits standard-compliant 5G NR test model (NR-TM) waveforms for frequency range 1 (FR1) and frequency range 2 (FR2) [ 1 ].

For the NR-TM waveform generation, you can specify the NR-TM name, the channel bandwidth, the subcarrier spacing (SCS), the duplexing mode, the cell identity, and the TS 38.141 version using the NR-TM Transmission block mask. Additionally, this block provides the option to enable or disable the ACLR test. When the ACLR measurement is enabled, the waveform is oversampled to visualize the spectral regrowth.



For more information on how to generate NR-TMs, see “5G NR-TM and FRC Waveform Generation” on page 6-2.

As the example works on a subframe by subframe basis, the NR-TM Transmission block sends one subframe at a time. Transmitting ten subframes, corresponding to one frame in the case of FDD duplexing mode, lasts 10 ms. If the simulation time is longer than 10 ms, the NR-TM Transmission block transmits the same frame cyclically. The Subframe Counter block stores the number of the currently transmitted subframe. If the simulation time is longer than a frame period, the Subframe Counter block resets to 0.

### RF Transmission

The RF Transmitter block is based on a superheterodyne transmitter architecture. This architecture upconverts the waveform to the passband frequency and applies RF filtering and amplification before transmitting the signal. Typical components of the superheterodyne transmitter are:

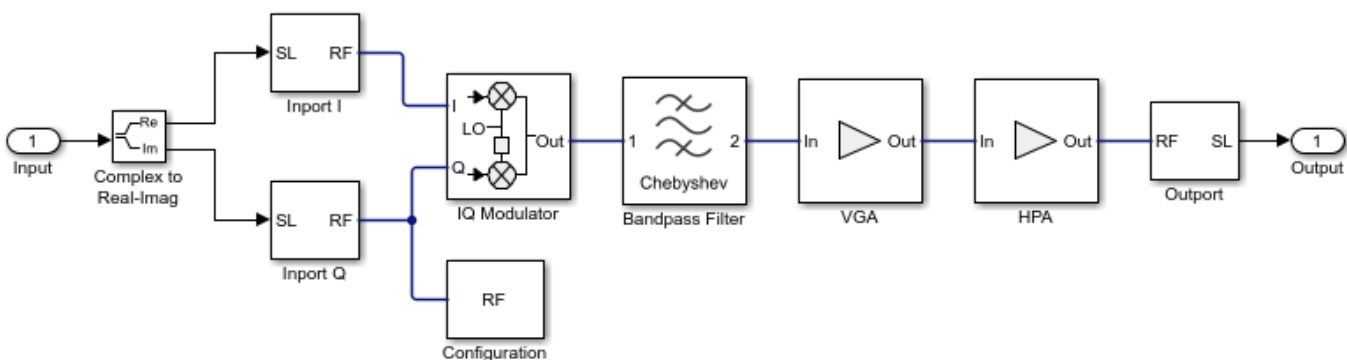
- IQ modulator consisting of mixers, phase shifter, and local oscillator
- Bandpass filter
- Power amplifier

In addition to these components, this RF Transmitter block also includes a variable gain amplifier (VGA) to control the input back-off (IBO) level of the high power amplifier (HPA).

```
set_param(modelName, 'Open', 'off');
set_param([modelName '/RF Transmitter'], 'Open', 'on');
```

## RF Superheterodyne Transmitter

This architecture upconverts the waveform to the carrier frequency 2140MHz (default) and applies RF filtering and amplification before transmitting the signal.

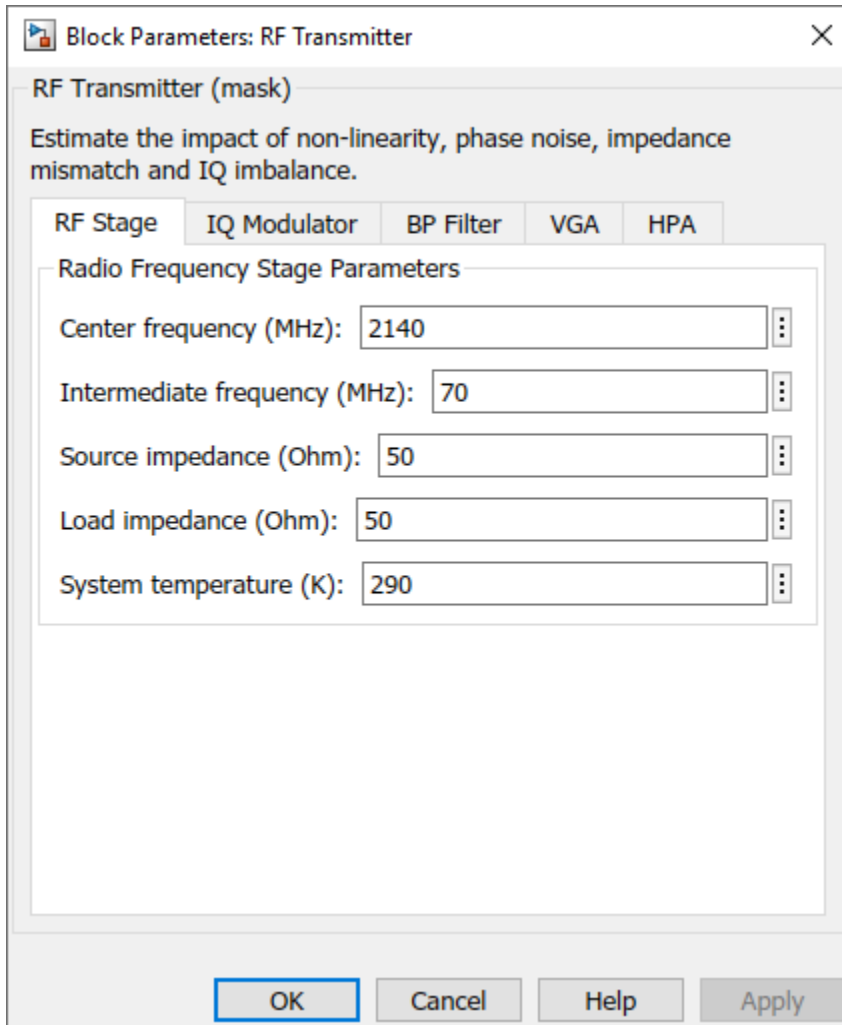


The Inport block converts the complex baseband waveform into an RF signal and the Outport block converts the RF signal back into complex baseband. Because the RF Transmitter accepts a maximum of 1024 samples per subframe, the Input Buffer, before the RF Transmitter block, reduces the number of samples sent to the RF Transmitter. In the current configuration, the Input Buffer sends 1024 samples at a time.

Before sending the samples onto the Decode Subframe block, the Output Buffer (after the RF Transmitter) buffers all samples within a subframe.

The Delay block accounts for buffer-induced delays. As the duration of the delay is equivalent to the transmission of a subframe, the Decode Subframe block does not demodulate the first information received during a subframe period.

You can configure the RF Transmitter components using the RF Transmitter block mask.



The RF Transmitter block exhibits typical impairments, including:

- I/Q imbalance as a result of gain or phase mismatches between the parallel sections of the transmitter chain dealing with the IQ signal paths.
- Phase noise as a secondary effect directly related to the thermal noise within the active devices of the oscillator.
- HPA nonlinearities due to DC power limitation when the amplifier works in saturation region.

This example highlights the effect of nonlinear behavior of the HPA.

## NR Baseband Reception and Measurements

The Decode Subframe block performs OFDM demodulation of the received subframe, channel estimation, and equalization to recover and plot the PDSCH symbols in the Constellation Diagram. This block also averages the EVM over time and frequency and plots these values:

- EVM per OFDM symbol: EVM averaged over each OFDM symbol.
- EVM per slot: EVM averaged over the allocated PDSCH symbols within a slot.
- EVM per subcarrier: EVM averaged over the allocated PDSCH symbols within a subcarrier.
- Overall EVM: EVM averaged over the allocated PDSCH symbols transmitted.

According to TS 38.141-1 [ 1 ], not all PDSCH symbols are considered for the EVM evaluation. Using the RNTI, the helper function `hListTargetPDSCHs` selects the target PDSCH symbols to analyze.

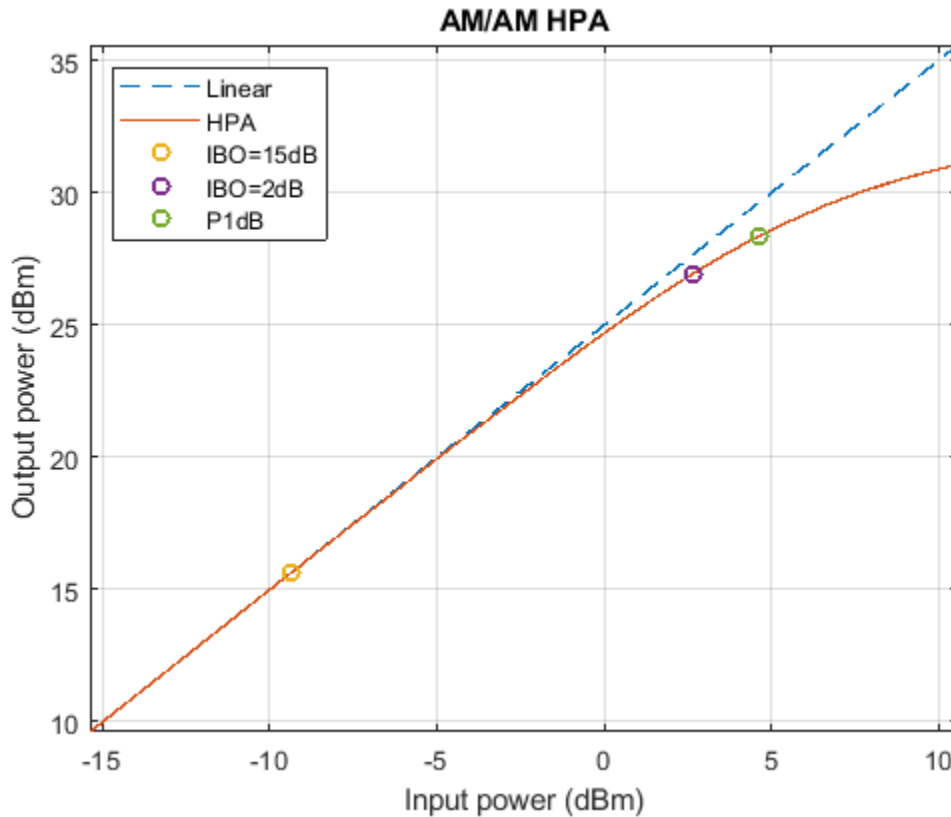
The Spectrum Analyzer block provides frequency-domain measurements such as ACLR (referred to as ACPR), occupied bandwidth, channel power, and CCDF. To visualize the spectral regrowth, the ACLR test oversamples the waveform. The oversampling factor depends on the waveform configuration and should be set such that the generated signal is capable of representing first and second adjacent channels. The ACLR evaluation follows the specifications in TS 38.141-1 [ 1 ].

## Model Performance

To characterize the impact of HPA nonlinearities in the EVM and ACLR evaluations, you can measure the amplitude-to-amplitude modulation (AM/AM) of the HPA. The AM/AM refers to the output power levels in terms of the input power levels. The helper function `hPlotHPACurve` displays the AM/AM characteristic of the HPA selected for this model.

```
hPlotHPACurve();  
figHPA =(gcf);
```





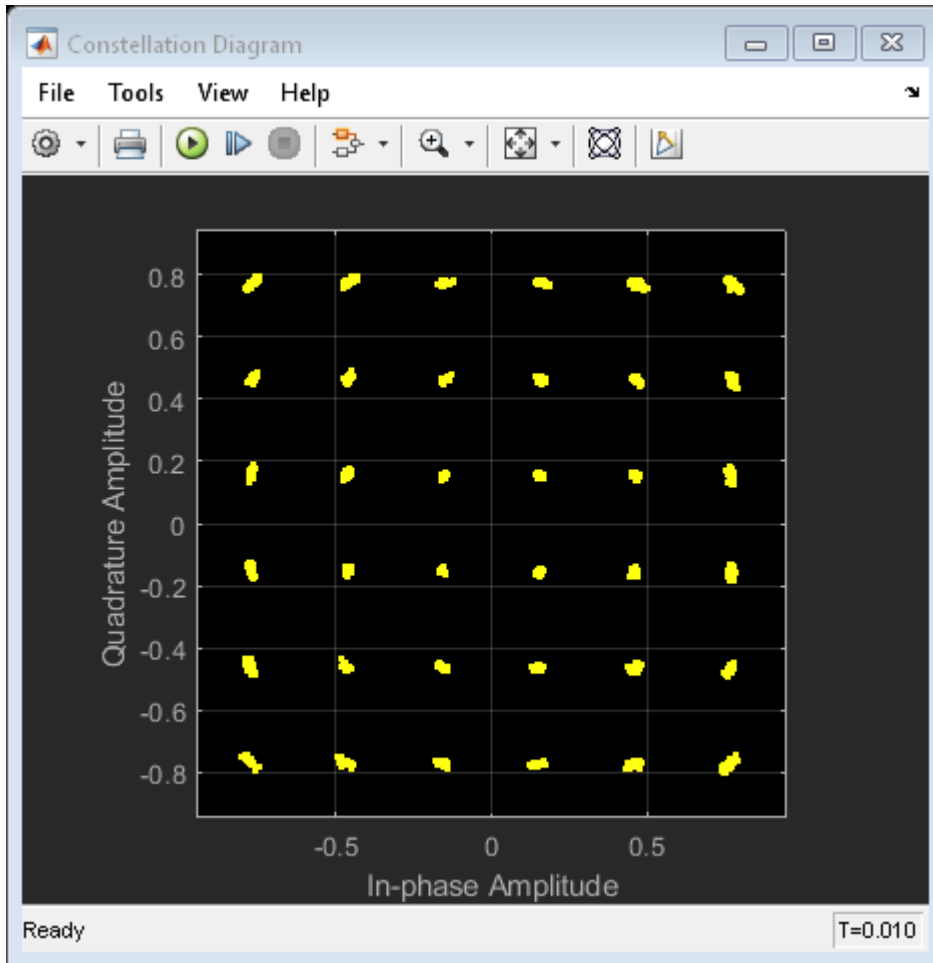
P1dB is the power at 1 dB compression point and is usually used as a reference when selecting the IBO level of the HPA. You can see the HPA impact on the RF transmitter by analyzing the EVM and ACLR results for different operating points of the HPA. For example, compare the case when IBO = 15 dB, corresponding to HPA operating in linear region, with the case when IBO = 2 dB, corresponding to HPA operating in full saturation. The gain of the VGA controls the IBO level. To keep a VGA linear behavior, select gain values lower than 20 dB.

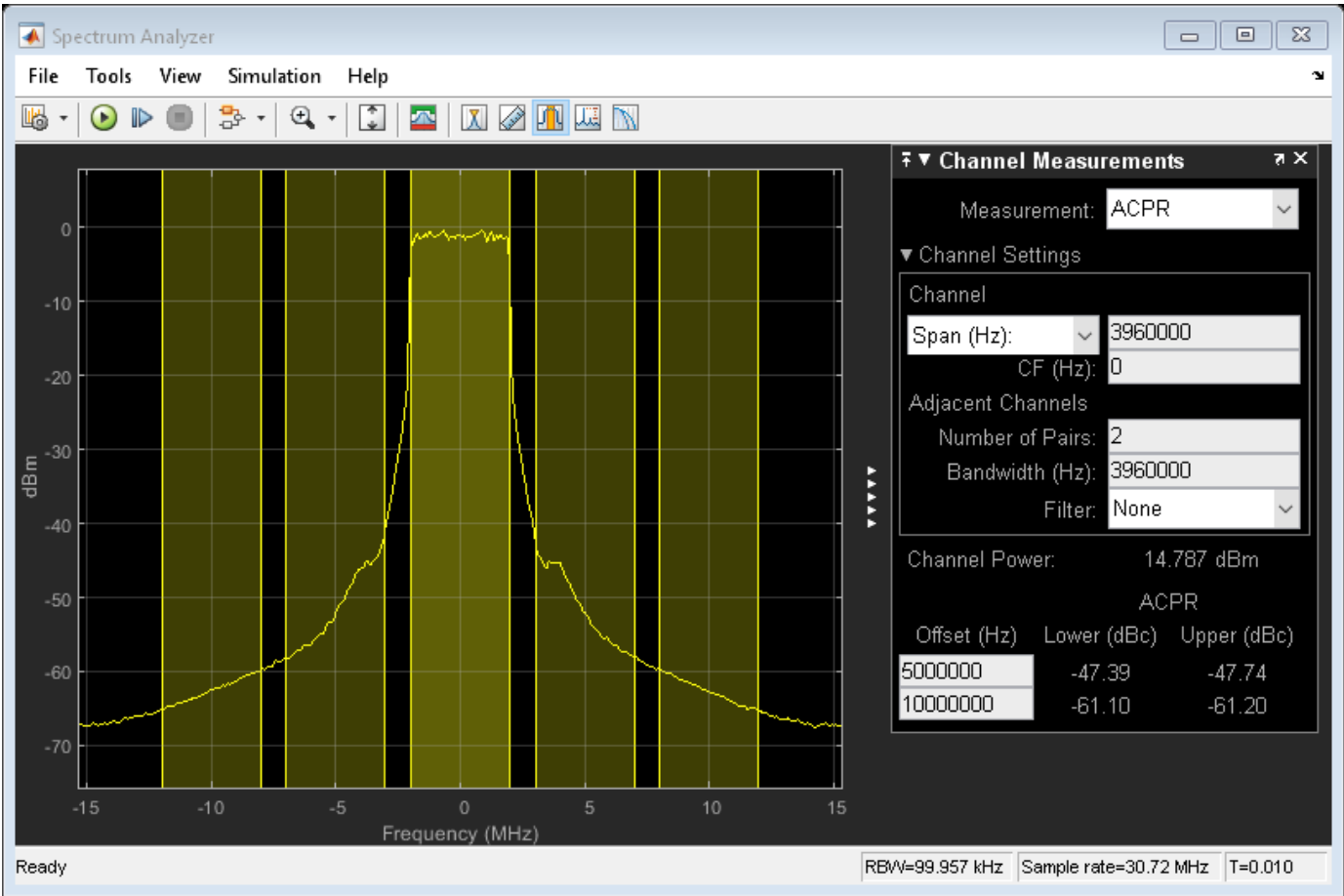
- *Linear HPA (IBO = 15 dB)*. To operate at an IBO level of 15 dB, set the **Available power gain** parameter of the VGA block to 0 dB. To simulate a whole frame, run the simulation long enough to capture 10 subframes (10 ms). During simulation, the model displays the EVM and ACLR measurements and the constellation diagram.

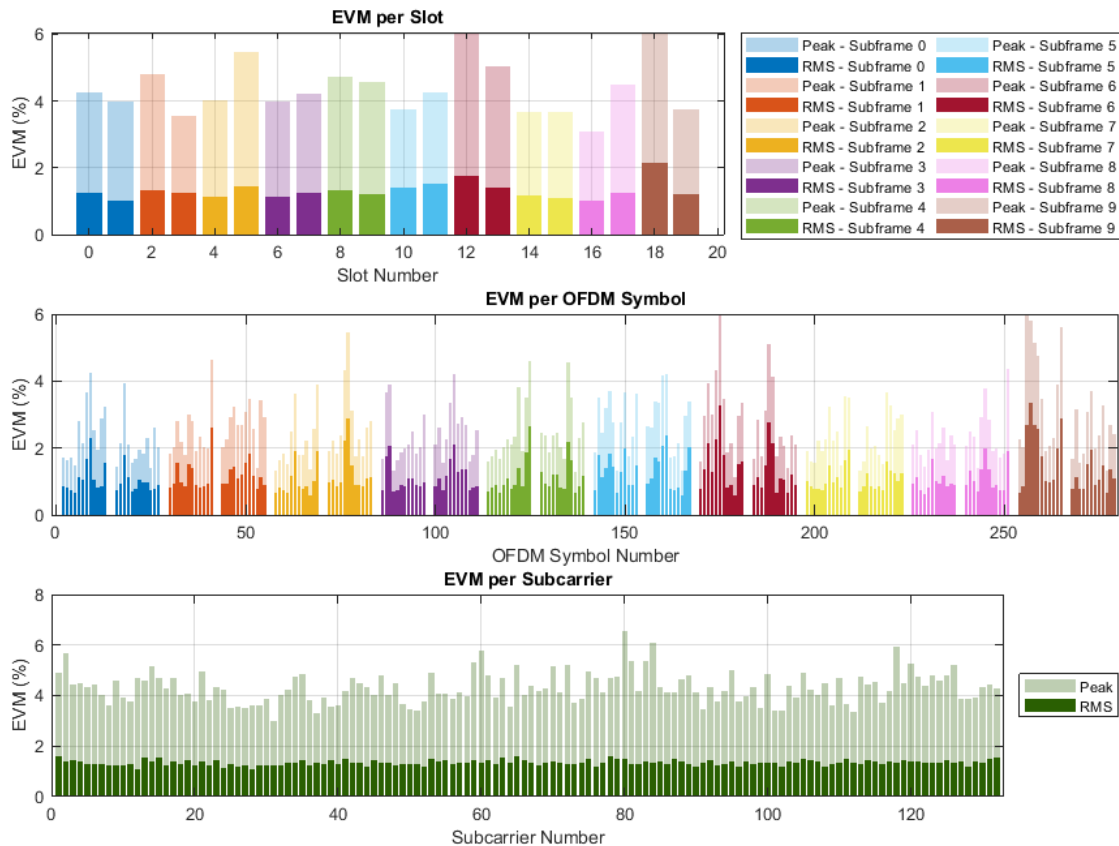
```
set_param([modelName '/RF Transmitter'], 'vgaGain', '0');
sim(modelName);
```

```
--- Starting simulation ---
Transmitting subframe 0 ...
Transmitting subframe 1 ...
Transmitting subframe 2 ...
Transmitting subframe 3 ...
Transmitting subframe 4 ...
Transmitting subframe 5 ...
Transmitting subframe 6 ...
Transmitting subframe 7 ...
Transmitting subframe 8 ...
Transmitting subframe 9 ...
```

--- End of simulation ---







According to TS 38.104 [ 2 ], the minimum required ACLR for conducted measurements is 45 dB and the maximum required EVM when the constellation is 64-QAM is 8%. As the ACLR values are higher than 45 dB, and the overall EVM, around 1.3%, is lower than 8%, both measurements fall within the requirements.

- *Nonlinear HPA (IBO = 2 dB).* To operate at an IBO level of 2 dB, set the **Available power gain** parameter of the VGA block to 12 dB.

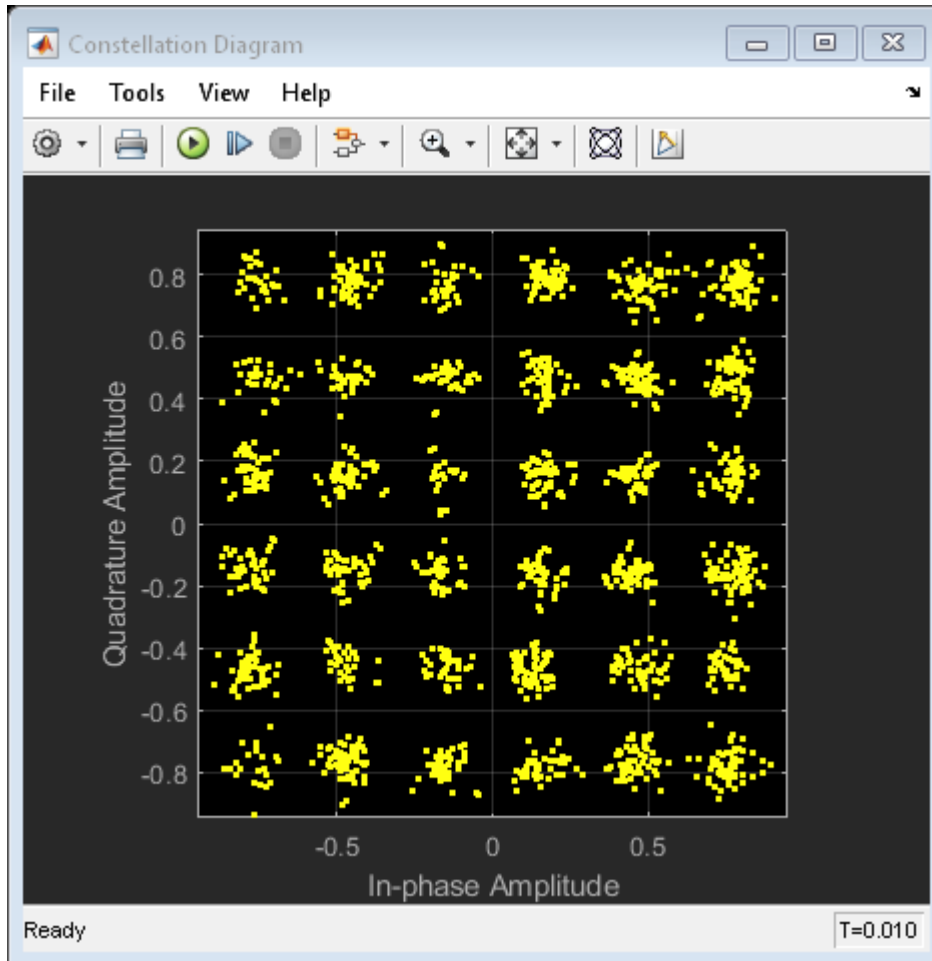
```
set_param([modelName '/RF Transmitter'], 'vgaGain', '12');
sim(modelName);
slmsgviewer.DeleteInstance();
```

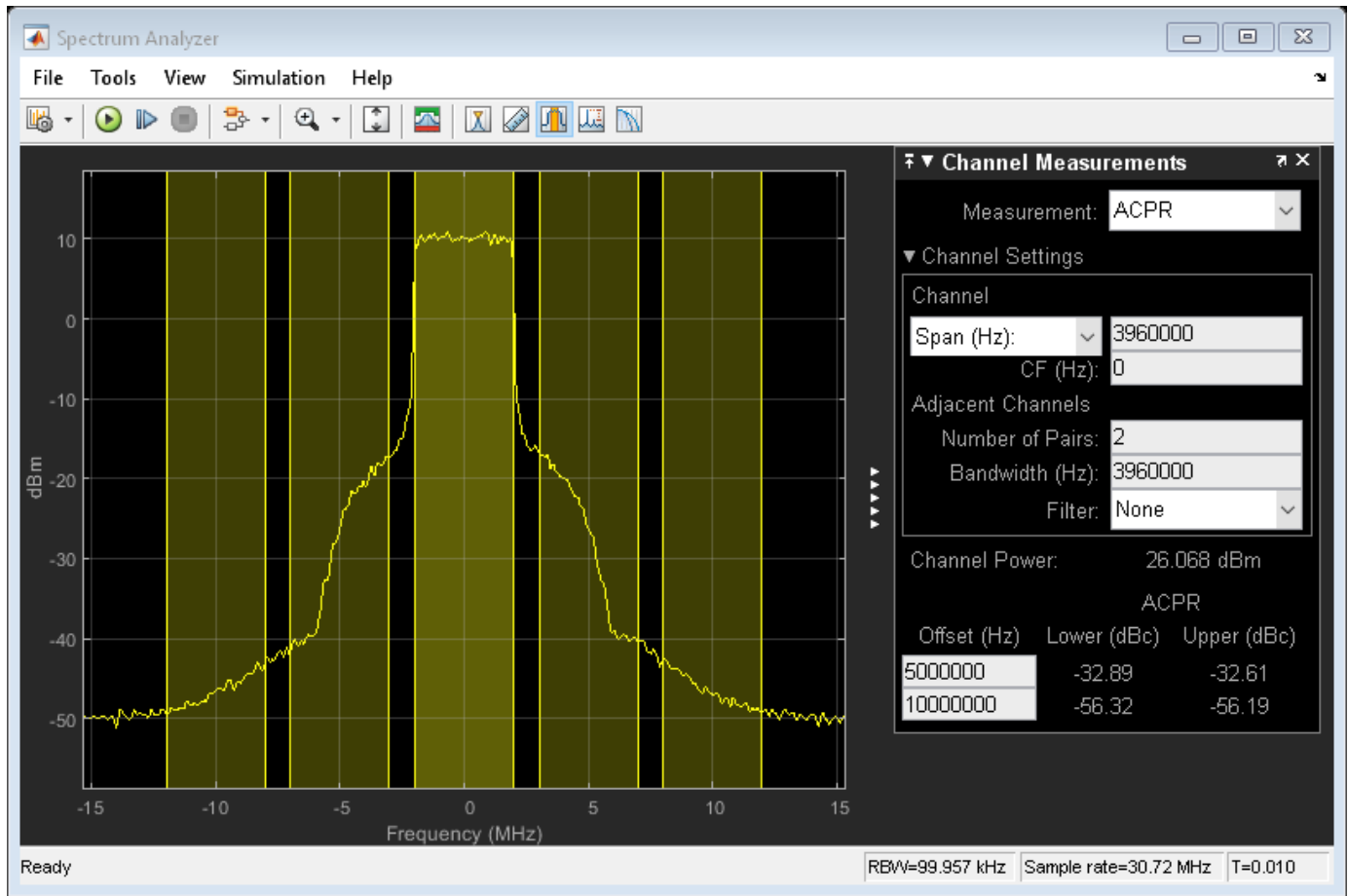
```
% Restore to default parameters
set_param([modelName '/RF Transmitter'], 'vgaGain', '0');
```

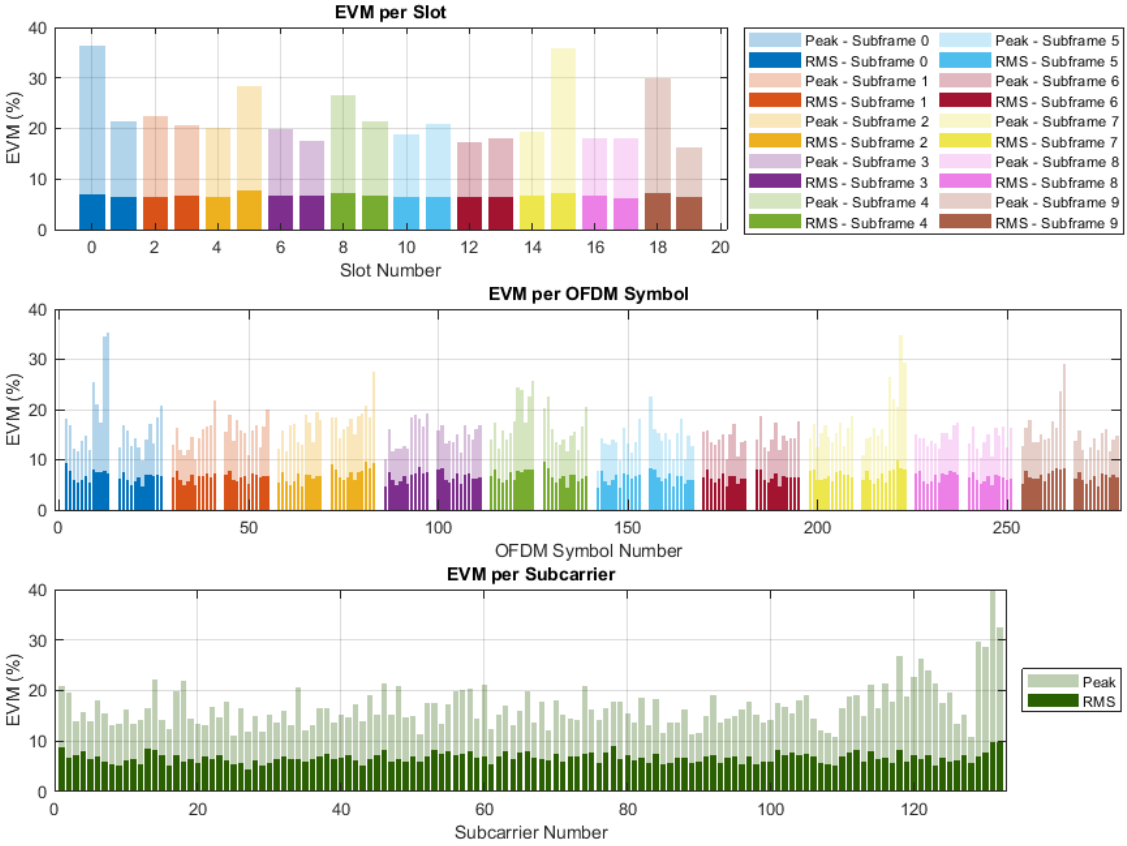
```
--- Starting simulation ---
Transmitting subframe 0 ...
Transmitting subframe 1 ...
Transmitting subframe 2 ...
Transmitting subframe 3 ...
Transmitting subframe 4 ...
Transmitting subframe 5 ...
Transmitting subframe 6 ...
Transmitting subframe 7 ...
```

```
Transmitting subframe 8 ...  
Transmitting subframe 9 ...
```

```
--- End of simulation ---
```







Compared to the previous case, the constellation diagram is distorted and the spectral regrowth is higher. In terms of the measurements, the first adjacent channel ACLR does not fall within the requirements of TS 38.104 [ 2 ] and the overall EVM, around 7%, is higher.

The RF Transmitter is configured to work with the default values of the NR-TM Transmission block and with a carrier centered at 2140 MHz (FR1). If you change the carrier frequency or the values in the Waveform Parameters block, you may need to update the parameters of the RF Transmitter components as these parameters have been selected to work for the default configuration of the example. For more information, see the Summary and Further Exploration section of this example.

### Summary and Further Exploration

This example demonstrates how to model and test an NR RF transmitter in Simulink. The RF transmitter consists of an IQ modulator, a bandpass filter and amplifiers. To evaluate the performance, the Simulink model considers ACLR and EVM measurements. The example highlights the effect of HPA nonlinearities on the performance of the RF Transmitter. You can explore the impact of altering other impairments as well. For example:

- Increase I/Q imbalance by using the **I/Q gain mismatch (dB)** and **I/Q phase mismatch (Deg)** parameters on the **IQ Modulator** tab of the RF Transmitter block.
- Increase the phase noise by using **Phase noise offset (Hz)** and **Phase noise level (dBc/Hz)** parameters on the **IQ Modulator** tab of the RF Transmitter block.

Additionally, you can check the occupied bandwidth, the channel power, and the CCDF measurements by using the Spectrum Analyzer block.

If you change the carrier frequency or the values in the Waveform Parameters block, you may need to update the parameters of the RF Transmitter components as these parameters have been selected to work for the default configuration of the example. For instance, a change in the carrier frequency requires revising the bandwidth of the filter. If you select a bandwidth wider than 20MHz, you may need to update the **Impulse response duration** and **Phase noise frequency offset (Hz)** parameters of the IQ Modulator block. The phase noise offset determines the lower limit of the impulse response duration. If the phase noise frequency offset resolution is too high for a given impulse response duration, a warning message appears, specifying the minimum duration suitable for the required resolution. For more information, see IQ Modulator (RF Blockset).

This example could be the basis for testing NR-TM waveforms for different RF configurations. You can try replacing the RF Transmitter block by another RF subsystem of your choice and configuring the model accordingly.

### 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 38.104. "NR; Base Station (BS) radio transmission and reception." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

### See Also

### Related Examples

- "5G NR-TM and FRC Waveform Generation" on page 6-2
- "Modeling and Testing an NR RF Receiver with LTE Interference" on page 6-45



# Modeling and Testing an NR RF Receiver with LTE Interference

The example shows how to characterize the impact of radio frequency (RF) impairments in the RF reception of a new radio (NR) waveform when coexisting with a long-term evolution (LTE) interference. The baseband waveforms are generated using 5G Toolbox™ and LTE Toolbox™, and the RF receiver is modeled using RF Blockset™.

## Introduction

This example characterizes the impact of LTE interference in the RF reception of an NR waveform. To evaluate the impact of the interference, the example performs these measurements:

- Error vector magnitude (EVM): vector difference at a given time between the ideal (transmitted) signal and the measured (received) signal.
- Adjacent channel leakage ratio (ACLR): measure of the amount of power leaking into adjacent channels. It is defined as the ratio of the filtered mean power centered on the assigned channel frequency to the filtered mean power centered on an adjacent channel frequency.
- Occupied bandwidth: bandwidth that contains 99% of the total integrated power of the signal, centered on the assigned channel frequency.
- Channel power: filtered mean power centered on the assigned channel frequency.
- Complementary cumulative distribution function (CCDF): probability of a signal's instantaneous power to be a level specified above its average power.

The impact of the receiver RF impairments such as in-phase and quadrature (IQ) imbalance, phase noise, and power amplifier (PA) nonlinearities are also considered.

The example works on a subframe by subframe basis. For each subframe, the workflow consists of these steps:

- 1 Generate the baseband NR waveform using 5G Toolbox.
- 2 Generate the baseband LTE waveform (interference) using LTE Toolbox.
- 3 Upconvert both waveforms to their carrier frequencies using RF Blockset.
- 4 Use an RF receiver to downconvert the waveform centered at the NR carrier to baseband frequency.
- 5 Calculate the ACLR/ACPR, occupied bandwidth, channel power, and CCDF using the Spectrum Analyzer block.
- 6 Demodulate the NR baseband waveform to measure EVM.

This example uses a Simulink model to perform these operations. Baseband signal processing (steps 1, 2, and 6) uses MATLAB® Function blocks, whereas the RF receiver modeling (steps 3 and 4) uses RF Blockset. This model supports Normal and Accelerator simulation modes.

## Simulink Model Structure

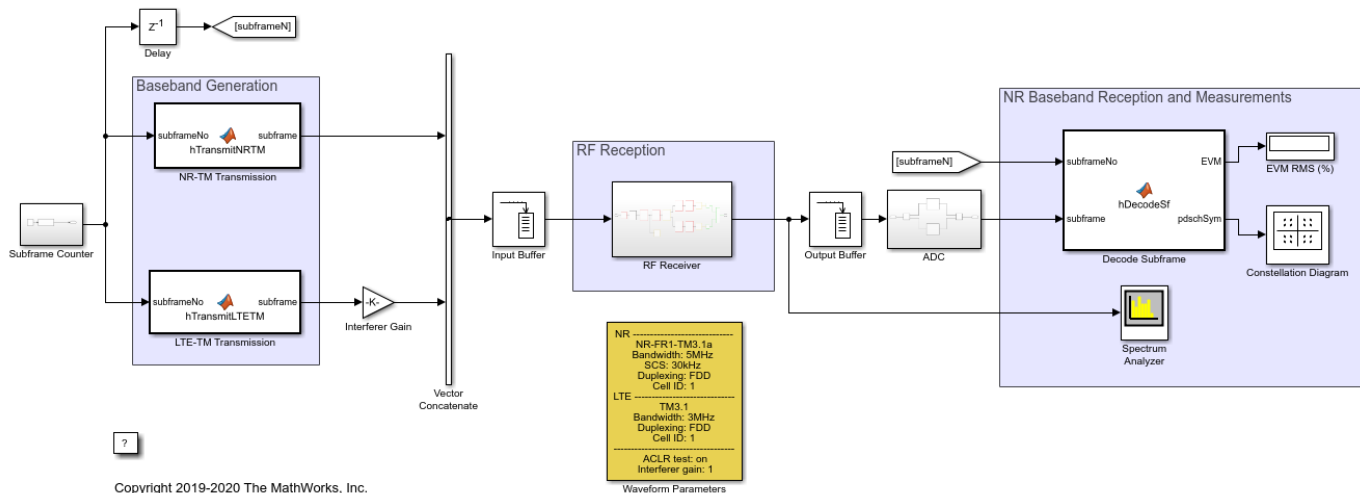
The model contains three main parts:

- Baseband Generation: generates the baseband NR and LTE waveforms.
- RF Reception: the upconverted waveforms pass through an RF receiver which downconverts the waveform centered at the NR carrier.
- NR Baseband Reception and Measurements: performs the RF measurements and demodulates the baseband waveform to calculate the EVM.

```
modelName = 'NewRadioRFReceiverWithLTEInterferenceModel';
open_system(modelName);
```

### Modeling and Testing an NR RF Receiver with LTE Interference

Input signal: NR Test Model  
 Interferer: LTE Test Model  
 Tests: EVM, ACLR, occupied bandwidth, channel power and CCDF



Copyright 2019-2020 The MathWorks, Inc.

### Baseband Generation

The NR-TM Transmission block transmits standard-compliant 5G NR test model (NR-TM) waveforms for frequency range 1 (FR1) and frequency range 2 (FR2) [ 1 ]. For the NR-TM waveform generation, you can specify the NR-TM name, the channel bandwidth, the subcarrier spacing (SCS), the duplexing mode, the cell identity, and the TS 38.141 version using the Waveform Parameters block.

Similarly, the LTE-TM Transmission block transmits standard-compliant LTE-TM waveforms [ 2 ]. You can also specify the TM name, the channel bandwidth, the duplexing mode and the cell identity. This model resamples, if necessary, the LTE waveform to match the sampling rate of the NR waveform. The Waveform Parameter block does not accept an LTE bandwidth wider than the NR bandwidth.

Additionally, the Waveform Parameters block provides the option to enable or disable the ACLR test. When the ACLR measurement is enabled, the waveform is oversampled to visualize the spectral regrowth. The Waveform Parameters block also includes a parameter called **Interferer Gain** which allows control the linear gain of the LTE interference. To cancel the transmission of the LTE interference, set the **Interferer Gain** parameter to 0. The Interferer Gain block is connected between the Baseband Generation and the RF Reception stages.

**Block Parameters: Waveform Parameters**

**NR Waveform Parameters**

NR-TM: NR-FR1-TM3.1a (Full band uniform 256QAM)

NR bandwidth: 5MHz (FR1)

SCS: 30kHz (FR1)

NR duplexing: FDD

NR cell identity: 1

TS 38.141 version: 15.2.0

**LTE Waveform Parameters**

LTE-TM: 3.1

LTE bandwidth: 3MHz

LTE duplexing: FDD

LTE cell identity: 1

ACLR test (waveform oversampling is applied)

Interferer gain: 1

OK Cancel Help Apply

For more information on how to generate NR-TMs and LTE-TMs, see “5G NR-TM and FRC Waveform Generation” on page 6-2 and “LTE Downlink Test Model (E-TM) Waveform Generation” (LTE Toolbox), respectively.

As the example works on a subframe by subframe basis, the NR-TM and LTE-TM Transmission blocks generate one subframe at a time. Transmitting ten subframes, corresponding to one frame in the case of FDD duplexing mode, lasts 10 ms. If the simulation time is longer than 10 ms, both blocks transmit the same frame cyclically. The Subframe Counter block stores the number of the currently transmitted subframe. If the simulation time is longer than a frame period, the Subframe Counter block resets to 0.

### RF Reception

The RF Receiver block is based on a superheterodyne receiver architecture. This architecture applies passband filtering and amplification and downconverts the received waveform to an intermediate frequency. The RF components of this superheterodyne receiver are:

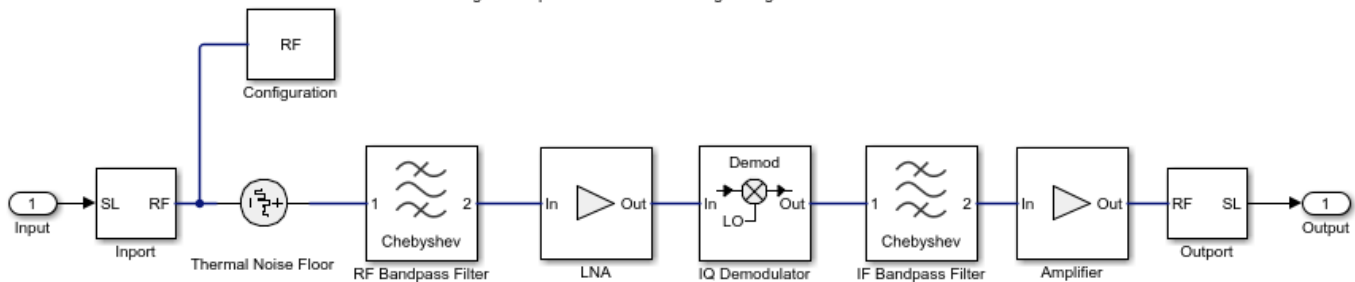
- Bandpass filters

- Low noise and power amplifiers
- IQ demodulator consisting of mixers, phase shifter, and local oscillator

```
set_param(modelName, 'Open', 'off');
set_param([modelName '/RF Receiver'], 'Open', 'on');
```

## RF Superheterodyne Receiver

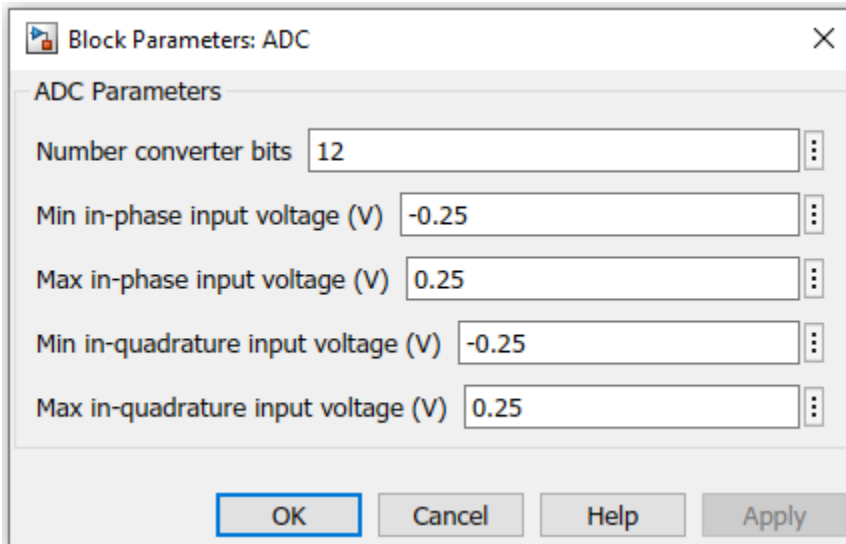
This architecture downconverts the waveform to the intermediate frequency 70MHz (default) and applies RF filtering and amplification before decoding the signal.



To send both waveforms to the RF Receiver, insert a Vector Concatenate block between the Baseband Reception and the RF Reception stages. The Vector Concatenate block concatenates both waveforms horizontally, one column per waveform. Then, the Inport block inside the RF Receiver converts the two concatenated complex baseband waveforms into RF signals considering the center frequencies chosen in the **Carrier frequencies** parameter of this block (every frequency selected in **Carrier frequencies** is assigned to a different concatenated waveform). The Outport block converts the RF signal back into complex baseband.

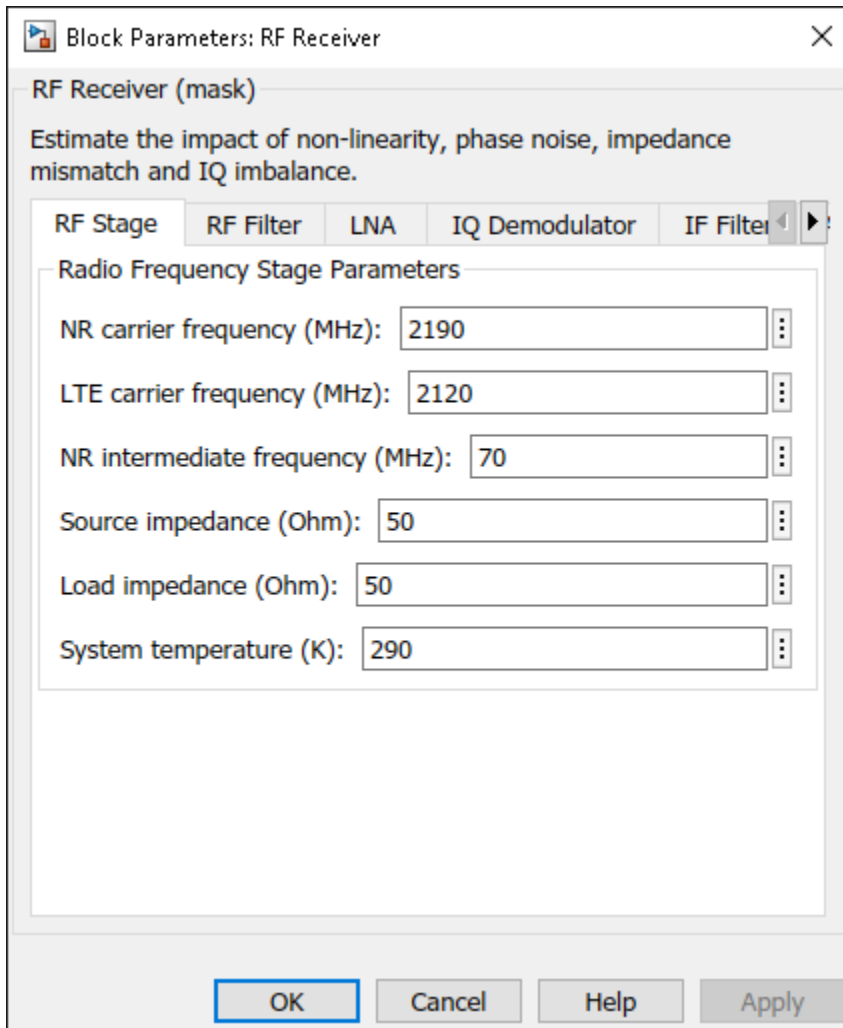
Due to the fact that the RF Receiver accepts a maximum of 1024 samples per subframe, the Input Buffer, before the RF Receiver block, reduces the number of samples sent to the RF Receiver. In the current configuration, the Input Buffer sends 1024 samples at a time.

Before sending the samples onto the Decode Subframe block, the Output Buffer (after the RF Receiver) buffers all samples within a subframe and the ADC block digitizes the signal. You can modify the ADC block parameters using its mask.



The Delay block accounts for buffer-induced delays. As the duration of the delay is equivalent to the transmission of a subframe, the Decode Subframe block does not demodulate the first information received during a subframe period.

You can configure the RF Receiver components using the RF Receiver block mask.



The RF Receiver block exhibits typical impairments, including:

- I/Q imbalance as a result of gain or phase mismatches between the parallel sections of the receiver chain dealing with the IQ signal paths.
- Phase noise as a secondary effect directly related to the thermal noise within the active devices of the oscillator.
- PA nonlinearities due to DC power limitation when the amplifier works in saturation region.

### NR Baseband Reception and Measurements

The Decode Subframe block performs OFDM demodulation of the received subframe, channel estimation, and equalization to recover and plot the PDSCH symbols in the Constellation Diagram. This block also averages the EVM over time and frequency and plots these values:

- EVM per OFDM symbol: EVM averaged over each OFDM symbol.
- EVM per slot: EVM averaged over the allocated PDSCH symbols within a slot.
- EVM per subcarrier: EVM averaged over the allocated PDSCH symbols within a subcarrier.
- Overall EVM: EVM averaged over all the allocated PDSCH symbols transmitted.

According to TS 38.141-1 [ 1 ], not all PDSCH symbols are considered for the EVM evaluation. Using the RNTI, the helper function `hListTargetPDSCHs` selects the target PDSCH symbols to analyze.

The Spectrum Analyzer block provides frequency-domain measurements such as ACLR (referred to as ACPR), occupied bandwidth, channel power, and CCDF. To visualize the spectral regrowth, the ACLR test oversamples the waveform. The oversampling factor depends on the waveform configuration and should be set such that the generated signal is capable of representing first and second adjacent channels. The ACLR evaluation follows the specifications in TS 38.141-1 [ 1 ].

### Model Performance

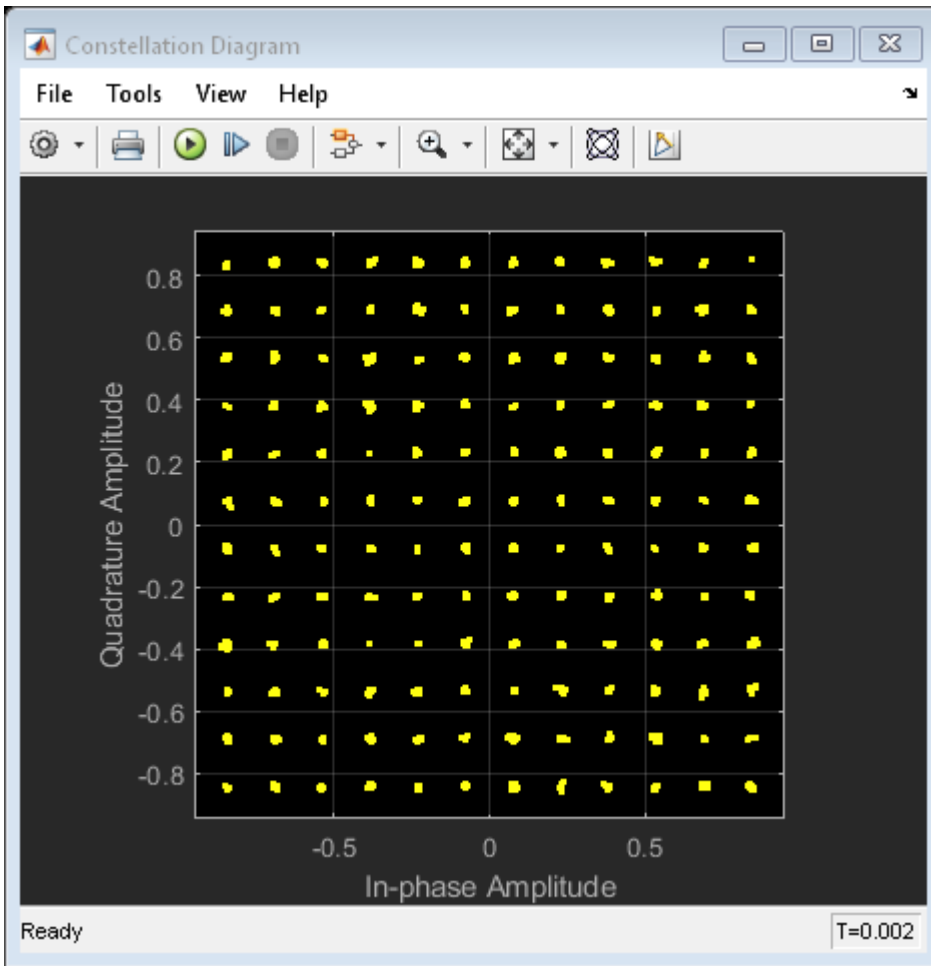
To characterize the impact of the LTE interference on the NR reception you can compare the EVM and ACLR results for two different cases: 1) there is no interference, for example only the NR waveform; and 2) there is interference, you transmit both waveforms, NR and LTE.

- *Without LTE interference (Interferer gain = 0).* To eliminate the LTE interference, set the **Interferer gain** parameter of the Waveform Parameters block to 0. To simulate a whole frame, run the simulation long enough to capture 10 subframes (10 ms). During simulation, the model displays the EVM and ACLR measurements and the constellation diagram. These are the results when transmitting 2 subframes.

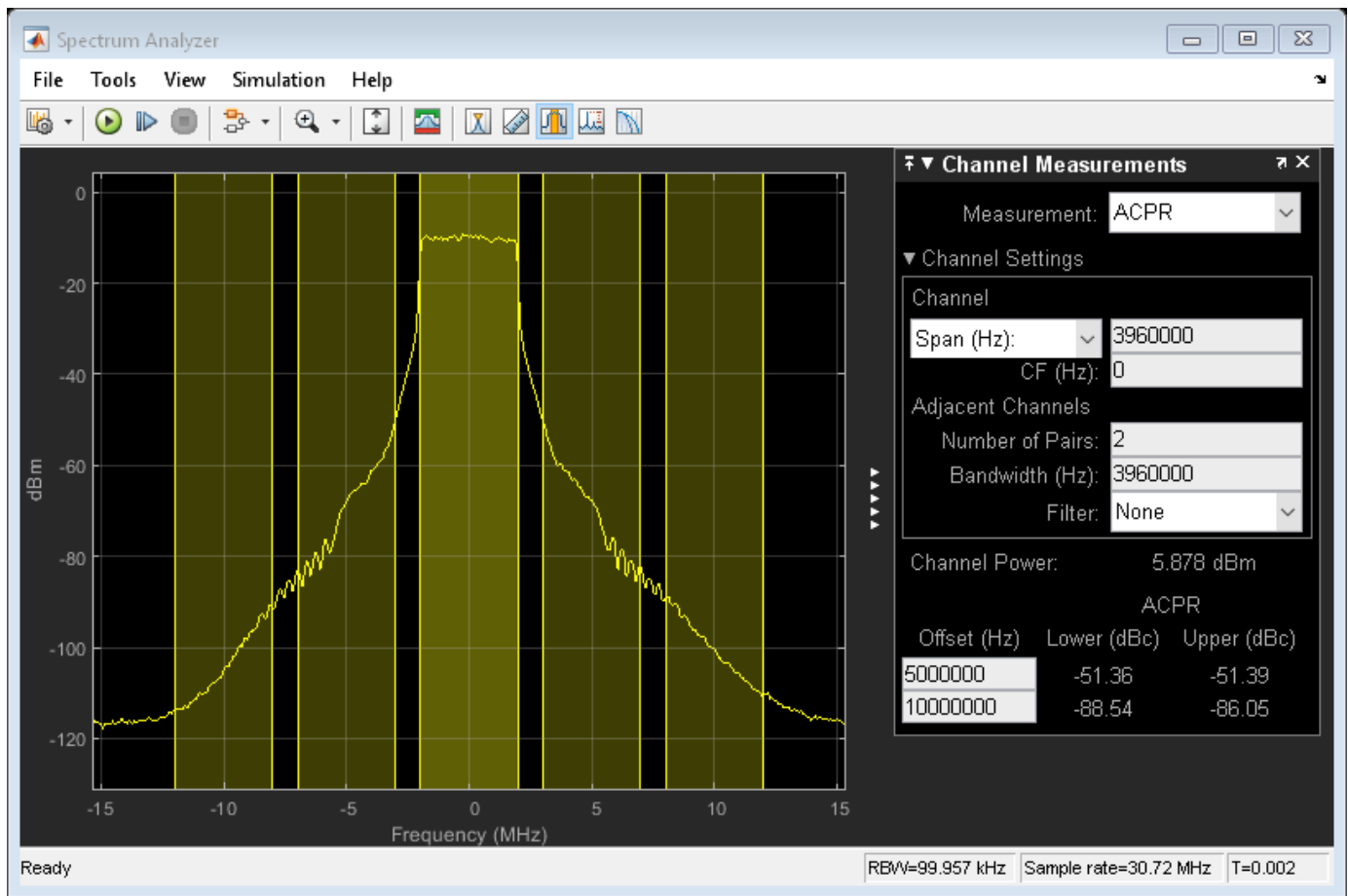
```
set_param([modelName '/Waveform Parameters'], 'InterfererGain', '0');
sim(modelName);
```

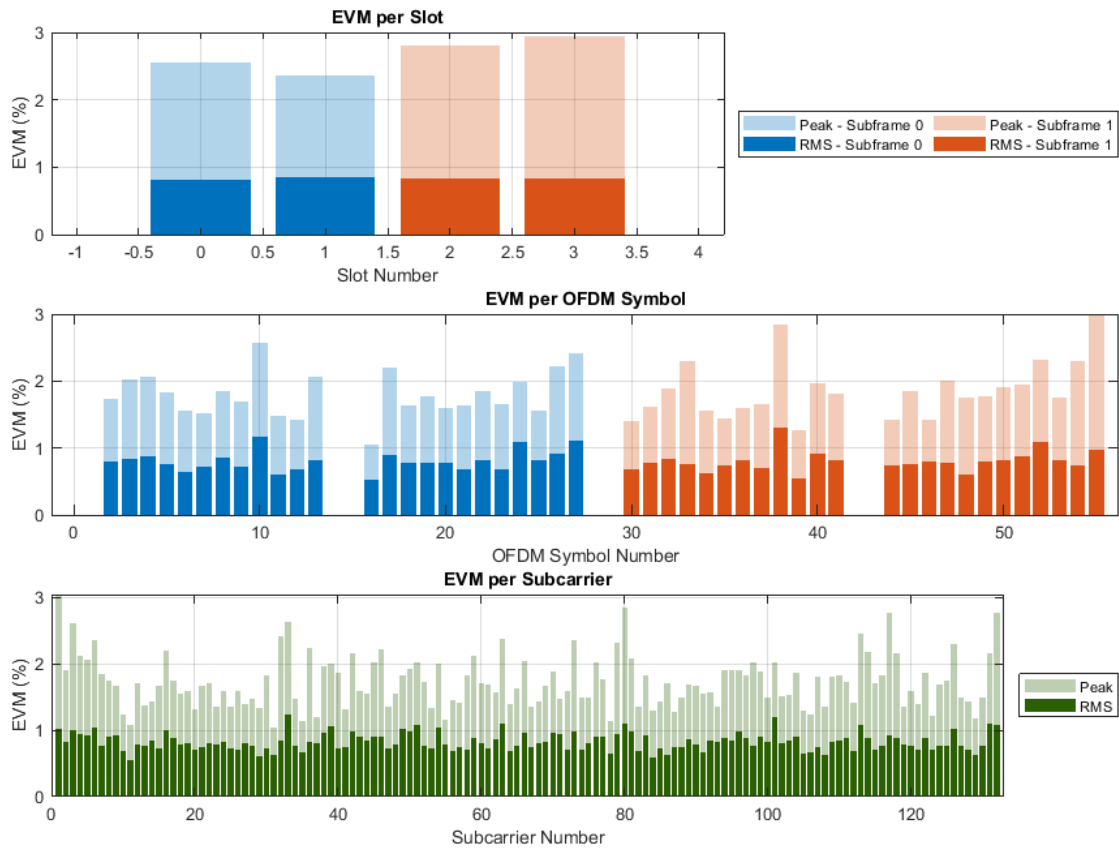
```
--- Starting simulation ---
  Transmitting subframe 0 ...
  Transmitting subframe 1 ...

--- End of simulation ---
```









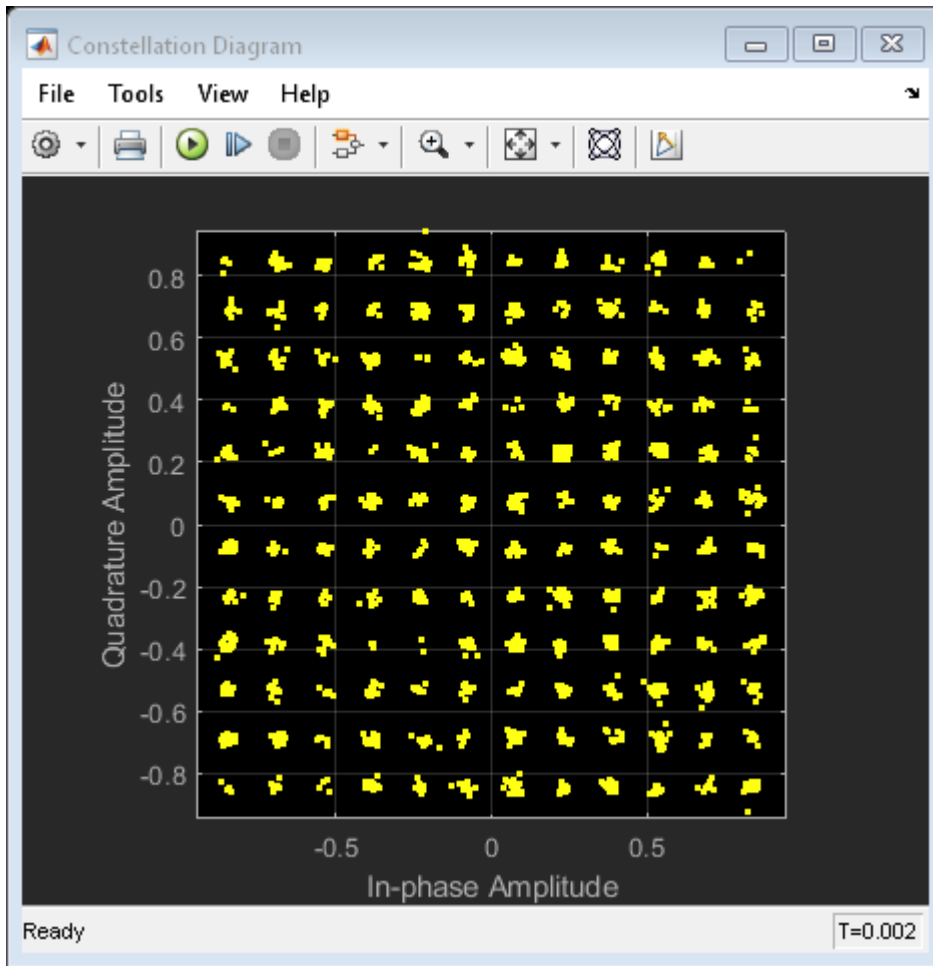
According to TS 38.104 [ 3 ], the minimum required ACLR for conducted measurements is 45 dB and the maximum required EVM when the constellation is 256-QAM is 3.5%. As the ACLR values, around 53 dB, are higher than 45 dB, and the overall EVM, around 0.8%, is lower than 3.5%, both measurements fall within the requirements.

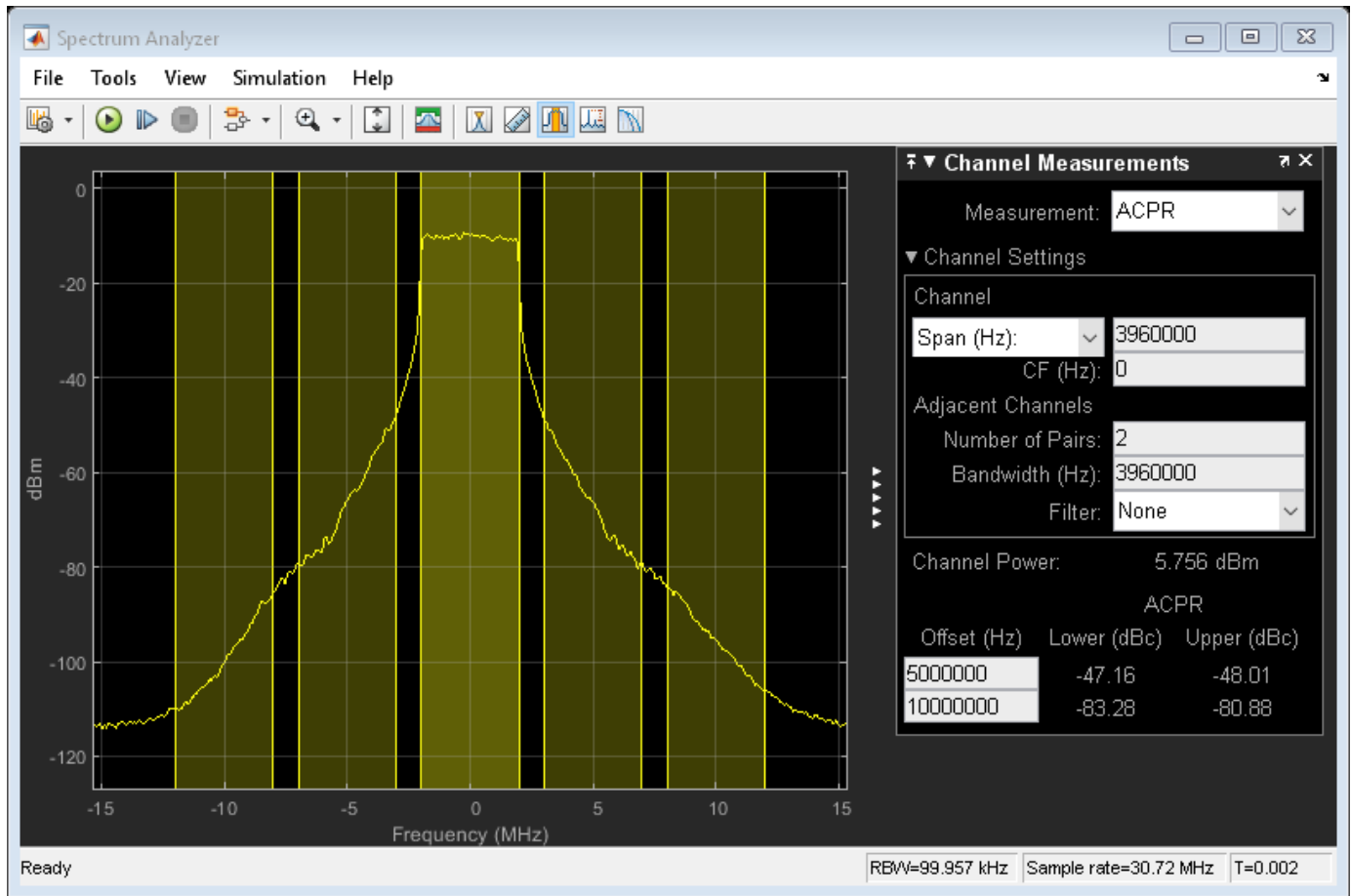
- *With LTE interference (Interferer gain = 1).* To activate the LTE interference, set the **Interferer gain** parameter of the Waveform Parameters block to any available value different from 0. For example, choose value 1.

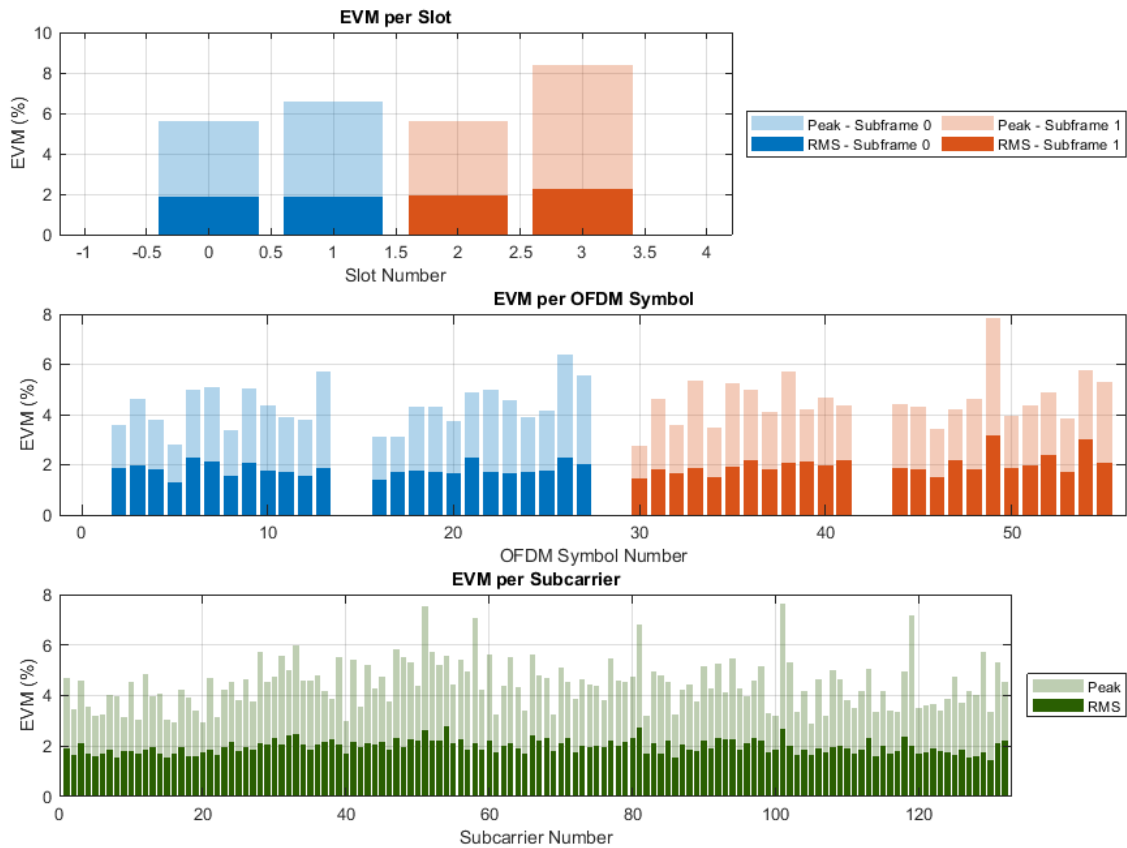
```
set_param([modelName '/Waveform Parameters'], 'InterfererGain', '1');
sim(modelName);
slmsgviewer.DeleteInstance();

--- Starting simulation ---
Transmitting subframe 0 ...
Transmitting subframe 1 ...

--- End of simulation ---
```







Compared to the previous case, the constellation diagram is more distorted and the spectral regrowth is higher. In terms of the measurements, the ACLR values, around 48 dB, and the overall EVM, around 2%, still fall within the requirements of TS 38.104 [ 3 ].

The RF Receiver is configured to work with the default values of the Waveform Parameters block and with the NR and LTE carriers centered at 2190 MHz and 2120 MHz, respectively. These carriers are within the NR operating band n65 [ 4 ] and the E-UTRA operating band 1 [ 5 ]. If you change the carrier frequency or the values in the Waveform Parameters block, you may need to update the parameters of the RF Receiver components as these parameters have been selected to work for the default configuration of the example. For more information, see the Summary and Further Exploration section of this example.

### Summary and Further Exploration

This example demonstrates how to model and test the reception of an NR waveform when coexisting with an LTE waveform. The RF receiver consists of bandpass filters, amplifiers and an IQ demodulator. To evaluate the impact of the LTE interference, the example modifies the gain of the LTE waveform and performs ACLR and EVM measurements. You can explore the impact of altering the RF impairments as well. For example:

- Increase I/Q imbalance by using the **I/Q gain mismatch (dB)** and **I/Q phase mismatch (Deg)** parameters on the **IQ Demodulator** tab of the RF Receiver block.

- Increase the phase noise by using **Phase noise offset (Hz)** and **Phase noise level (dBc/Hz)** parameters on the **IQ Demodulator** tab of the RF Receiver block.
- Reduce the input back-off of the two amplifiers (PA I and PA Q) increasing the **Gain (dB)** parameter on the **LNA** tab of the RF Receiver block.

Additionally, you can check the occupied bandwidth, the channel power, and the CCDF measurements by using the Spectrum Analyzer block.

If you change the carrier frequencies or the values in the Waveform Parameters block, you may need to update the parameters of the RF Receiver components as these parameters have been selected to work for the default configuration of the example. For instance, a change in the carrier frequencies requires revising the bandwidth of the filters. If you select an NR bandwidth wider than 20MHz, you may need to update the **Impulse response duration** and **Phase noise frequency offset (Hz)** parameters of the IQ Modulator block. The phase noise offset determines the lower limit of the impulse response duration. If the phase noise frequency offset resolution is too high for a given impulse response duration, a warning message appears, specifying the minimum duration suitable for the required resolution. For more information, see IQ Modulator (RF Blockset).

This example could be the basis for testing the coexistence between NR-TM and LTE-TM waveforms for different RF configurations. You can try replacing the RF Receiver block by another RF subsystem of your choice and configuring the model accordingly.

## 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 36.141 "E-UTRA; Base Station (BS) conformance testing" *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 38.101-1. "NR; User Equipment (UE) radio transmission and reception." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
- 5 3GPP TS 36.101. "E-UTRA; User Equipment (UE) radio transmission and reception." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

## See Also

### Related Examples

- "5G NR-TM and FRC Waveform Generation" on page 6-2
- "Modeling and Testing an NR RF Transmitter" on page 6-32

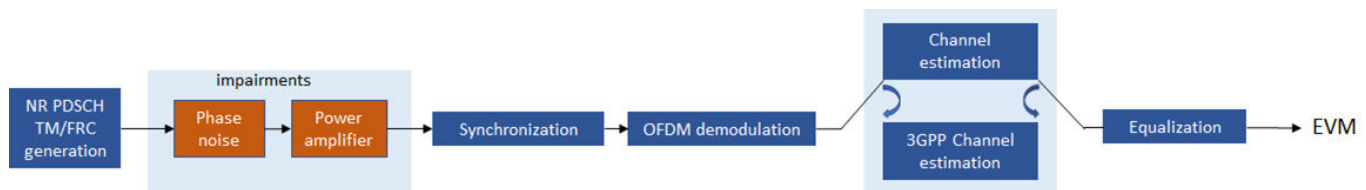
## EVM Measurement of 5G NR PDSCH Waveforms

This example measures the error vector magnitude (EVM) of NR test model (NR-TM) or fixed reference channel (FRC) waveforms. The example adds impairments including phase noise and memoryless nonlinearity.

### Introduction

For base station RF testing, the 3GPP 5G NR standard defines a set of downlink test model (NR-TM) waveforms. For user equipment (UE) testing, the standard defines a set of fixed reference channel (FRC) waveforms. The NR-TMs and FRCs for frequency range 1 (FR1) are defined in TS 38.141-1 while the NR-TMs and FRCs for frequency range 2 (FR2) are defined in TS 38.141-2.

This example shows how to generate an NR waveform (TM or FRC) and add impairments. Here we consider phase noise and memoryless nonlinearities. We then calculate the EVM of the resulting signal. We plot the RMS and peak EVMs per OFDM symbol, slot, and subcarrier. We also calculate the overall EVM (RMS EVM averaged over the complete waveform). Annex B and Annex C of TS 38.104 define an alternative method for computing the EVM in FR1 and FR2, respectively. The figure below shows the processing chain implemented in this example



### Simulation parameters

Each NR-TM or FRC waveform is defined by a combination of:

- NR-TM/FRC name
- Channel bandwidth
- Subcarrier spacing
- Duplexing mode

% Select one of the Release 15 NR-TMs for FR1 and FR2 among:

```

% "NR-FR1-TM1.1", "NR-FR1-TM1.2", "NR-FR1-TM2",
% "NR-FR1-TM2a", "NR-FR1-TM3.1", "NR-FR1-TM3.1a",
% "NR-FR1-TM3.2", "NR-FR1-TM3.3", "NR-FR2-TM1.1",
% "NR-FR2-TM2", "NR-FR2-TM3.1"
  
```

% or

% Select one of the Release 15 FRCs for FR1 and FR2 among:

```

% "DL-FRC-FR1-QPSK", "DL-FRC-FR1-64QAM",
% "DL-FRC-FR1-256QAM", "DL-FRC-FR2-QPSK",
% "DL-FRC-FR2-16QAM", "DL-FRC-FR2-64QAM"
  
```

```
rc = "NR-FR1-TM3.2"; % Reference channel(NR-TM or FRC)
```

```
% Select the NR waveform parameters
```

```
bw = "10MHz"; % Channel bandwidth
scs = "30kHz"; % Subcarrier spacing
dm = "FDD"; % Duplexing mode
```

For TMs, the generated waveform may contain more than one PDSCH. The chosen PDSCH to analyse is based on the RNTI. By default, the following RNTIs are considered for EVM calculation:

- NR-FR1-TM2: RNTI = 2 (64QAM EVM)
- NR-FR1-TM2a: RNTI = 2 (256QAM EVM)
- NR-FR1-TM3.1: RNTI = 0 and 2 (64QAM EVM)
- NR-FR1-TM3.1a: RNTI = 0 and 2 (256QAM EVM)
- NR-FR1-TM3.2: RNTI = 1 (16QAM EVM)
- NR-FR1-TM3.3: RNTI = 1 (QPSK EVM)
- NR-FR2-TM2: RNTI = 2 (64QAM EVM)
- NR-FR2-TM3.1: RNTI = 0 and 2 (64QAM EVM)

As per the specifications (TS 38.141-1, TS 38.141-2), these TMs are not designed to perform EVM measurements: NR-FR1-TM1.1, NR-FR1-TM1.2, NR-FR2-TM1.1. However, if you generate these TMs, the example measures the EVM for the following RNTIs.

- NR-FR1-TM1.1: RNTI = 0 (QPSK EVM)
- NR-FR1-TM1.2: RNTI = 2 (QPSK EVM)
- NR-FR2-TM1.1: RNTI = 0 (QPSK EVM)

For FRCs, by default, RNTI 0 is considered for EVM calculation.

The example calculates the EVM for the RNTIs listed above. To override the default RNTIs, specify the `targetRNTIs` vector

```
targetRNTIs = [];
```

To print EVM statistics on the command window, set `displayEVM` to `true`. To disable the prints, set `displayEVM` to `false`.

```
displayEVM = true;
```

To plot EVM statistics, set `plotEVM` to `true`. To disable the plot, set `plotEVM` to `false`.

```
plotEVM = true;
```

```
if displayEVM
    fprintf('Reference Channel = %s\n', rc);
end
```

```
Reference Channel = NR-FR1-TM3.2
```

To measure EVM as defined in TS 38.104, Annex B(FR1) / Annex C(FR2), set `evm3GPP` to `true`.

```
evm3GPP = false;
```

Create waveform generator object and generate the waveform

```
tmwavegen = hNRReferenceWaveformGenerator(rc,bw,scs,dm);
[txWaveform,tmwaveinfo,resourcesinfo] = generateWaveform(tmwavegen,tmwavegen.Config.NumSubframes
```

### Impairment: Phase Noise and Nonlinearity

This example considers two impairments: phase noise and memoryless nonlinearity. Enable or disable impairments by toggling the flags `phaseNoiseOn` and `nonLinearityModelOn`.



```
phaseNoiseOn = true;
nonLinearityModelOn = true;
```

Normalize the waveform to fit the dynamic range of the nonlinearity.

```
txWaveform = txWaveform/max(abs(txWaveform));
```

The waveform consists of one frame for FDD and two for TDD. Repeat the signal twice. We will remove the first half of the resulting waveform to avoid the transient introduced by the phase noise model.

```
txWaveform = repmat(txWaveform,2,1);
```

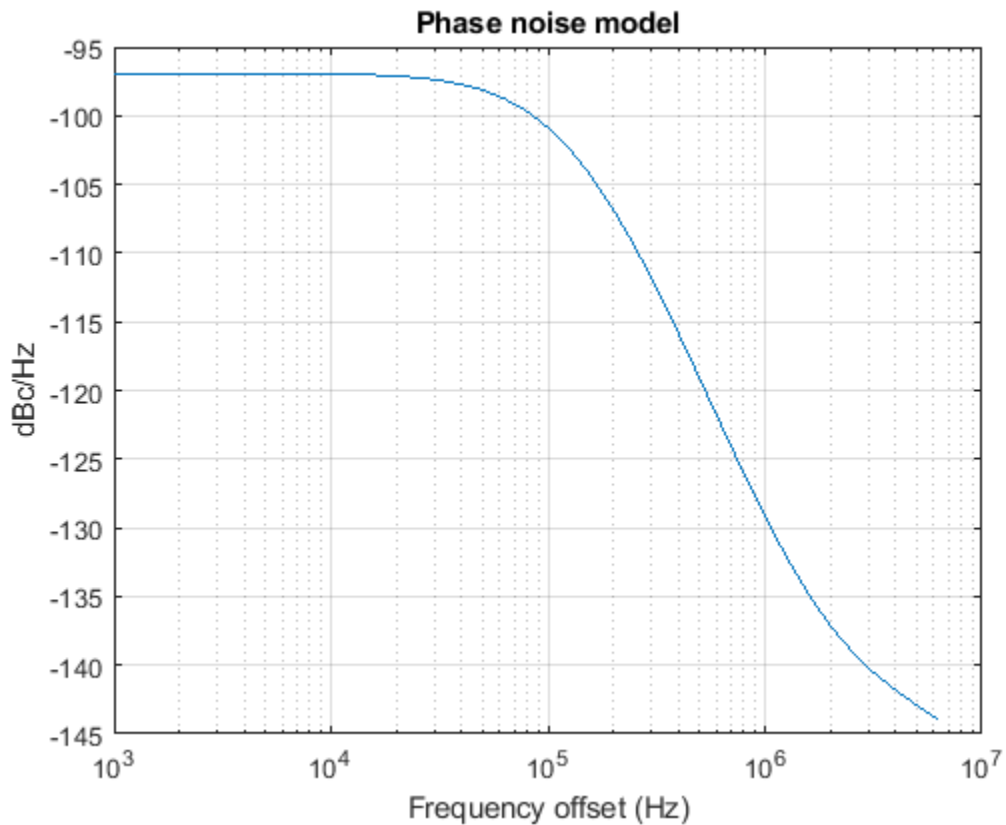
Introduce phase noise distortion. The generated figure shows the phase noise characteristic. The carrier frequency considered depends on the frequency range. We use values of 4 GHz and 28 GHz for FR1 and FR2 respectively. The phase noise characteristic is generated with the pole/zero model described in R1-163984, "Discussion on phase noise modeling".

```
if phaseNoiseOn
    sr = tmwaveinfo.Info.SamplingRate;

    % Carrier frequency
    if tmwavegen.Config.FrequencyRange == "FR1" % carrier frequency for FR1
        fc = 4e9;
    else % carrier frequency for FR2
        fc = 28e9;
    end

    % Calculate the phase noise level.
    foffsetLog = (3:0.1:log10(sr/2)); % model offset from 1e3Hz to sr/2
    foffset = 10.^foffsetLog; % linear freq offset
    PN_dBc_Hz = hPhaseNoisePoleZeroModel(foffset,fc,'A');
    figure; semilogx(foffset,PN_dBc_Hz); xlabel('Frequency offset (Hz)'); ylabel('dBc/Hz'); title

    % Apply phase noise to waveform
    pnoise = comm.PhaseNoise('FrequencyOffset',foffset,'Level',PN_dBc_Hz,'SampleRate',sr);
    rxWaveform = pnoise(txWaveform);
else
    rxWaveform = txWaveform; %#ok<*UNRCH>
end
```



Introduce nonlinear distortion. We use the Rapp model in this example. The figure shows the nonlinearity introduced. Set the parameters for the Rapp model to match the characteristics of the memoryless model from TR 38.803 (Memoryless polynomial model - Annex A.1).

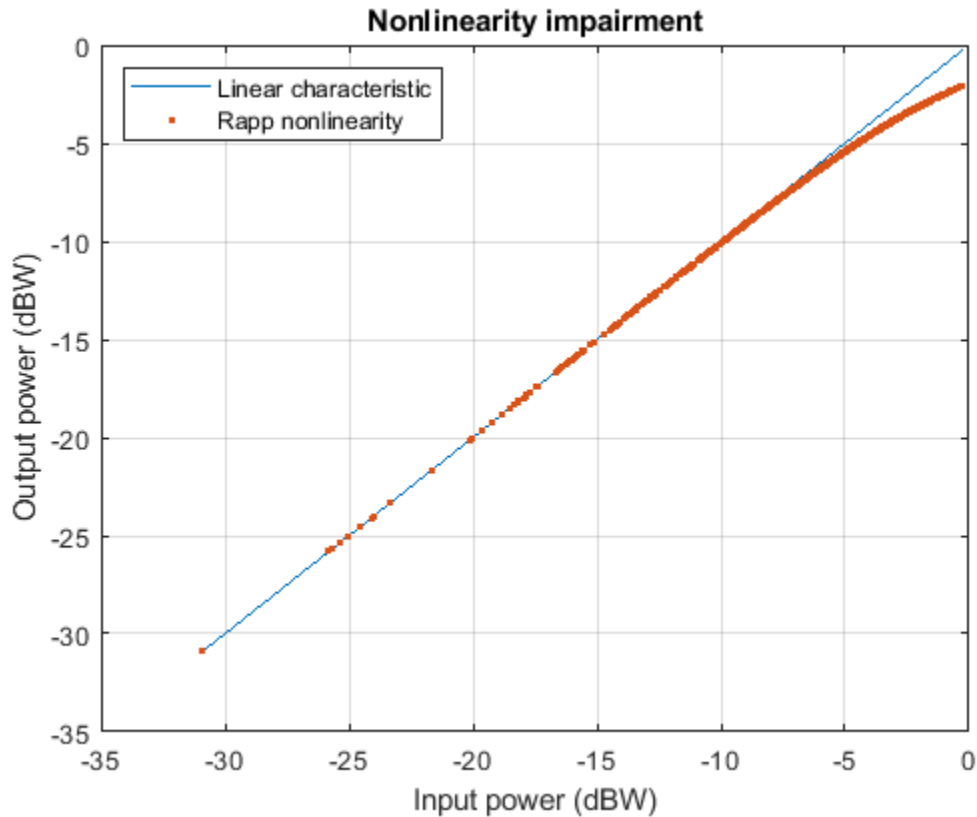
```

if nonLinearityModelOn
    rapp = comm.MemorylessNonlinearity('Method','Rapp model');
    rapp.Smoothness = 1.55;
    rapp.OutputSaturationLevel = 1;

    % Plot non-linear characteristic
    plotNonLinearCharacteristic(rapp);

    % Apply nonlinearity
    rxWaveform = rapp(rxWaveform);
end

```



The signal was previously repeated twice. We now remove the first half of this signal. This avoids any transient introduced by the impairment models.

```
if dm == "FDD"
    nFrames = 1;
else % TDD
    nFrames = 2;
end
rxWaveform(1:nFrames*tmwaveinfo.Info.SamplesPerSubframe*10) = [];
```

### Perform Measurements

The function, `hNRPDSCEVM`, performs these steps to decode and analyze the waveform:

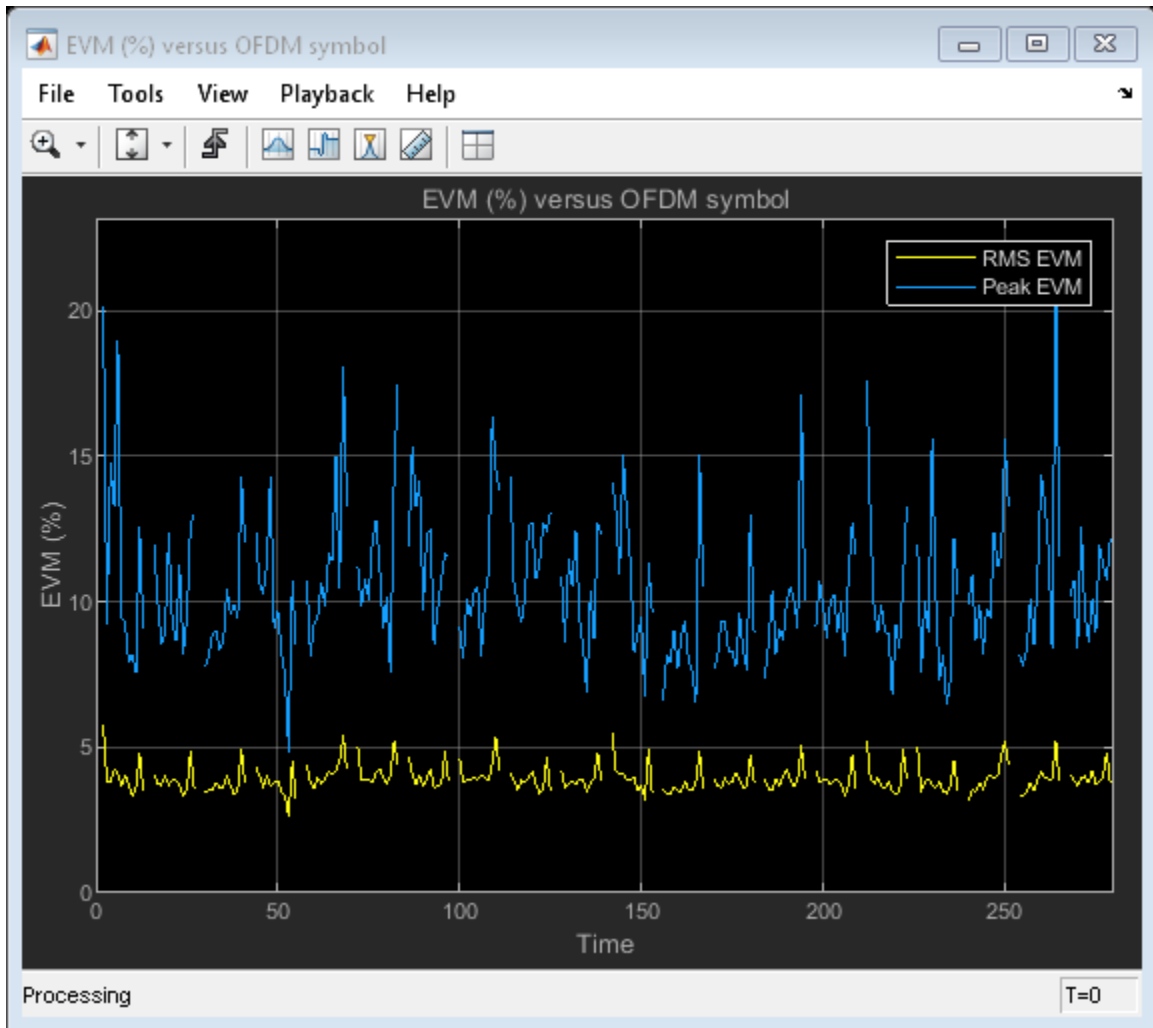
- Synchronization using the DM-RS over one frame for FDD (two frames for TDD)
- OFDM demodulation of the received waveform
- Channel estimation
- Equalization
- PDSCH EVM computation (enable the switch `evm3GPP`, to process according to the EVM measurement requirements specified in TS 38.104, Annex B(FR1) / Annex C(FR2))

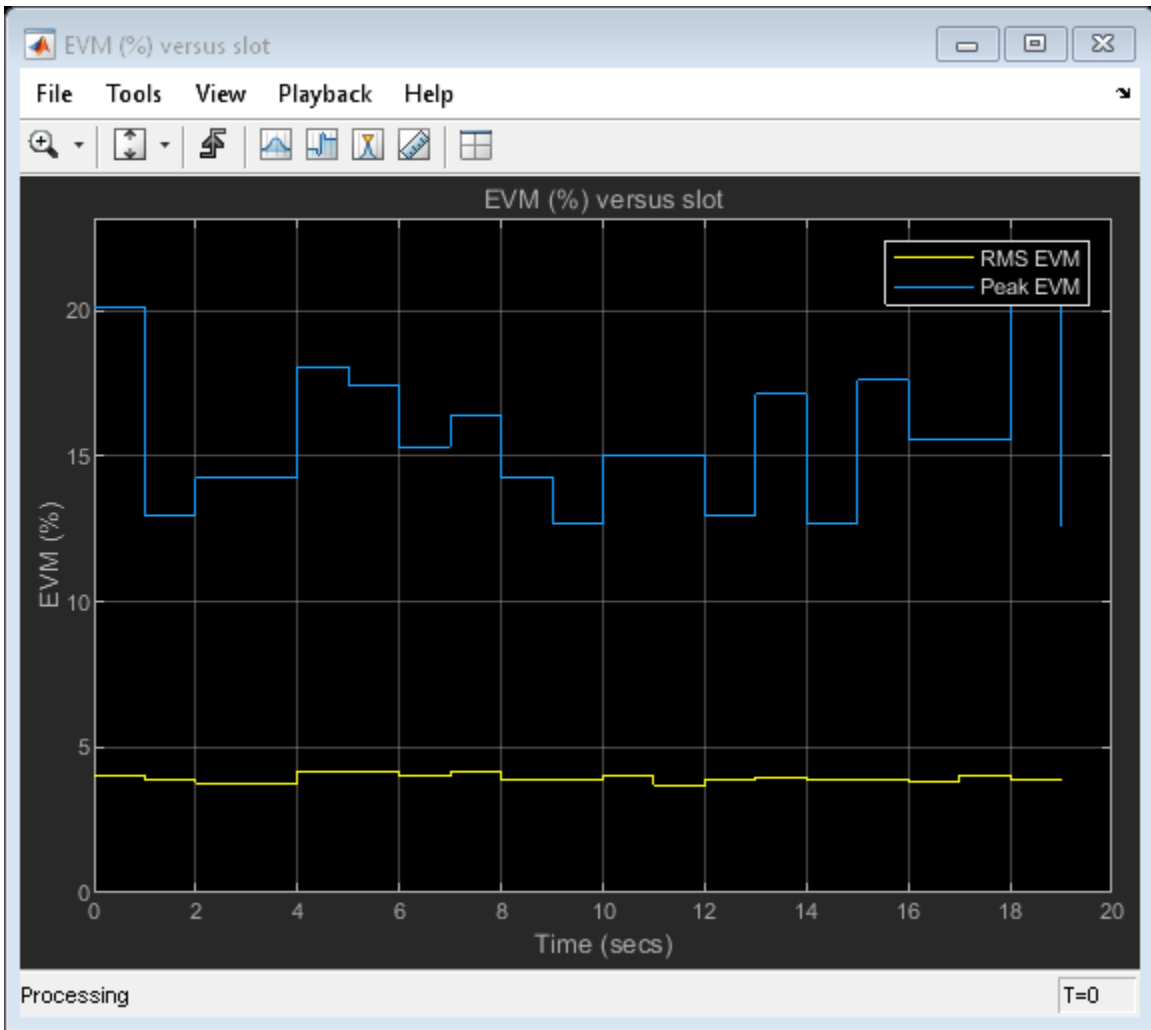
The example measures and outputs various EVM related statistics (i.e. per symbol, per slot, and per frame peak EVM and RMS EVM). The example displays EVM for each slot and frame on the command window. It also displays the overall EVM averaged over the entire input waveform. The example produces a number of plots: EVM vs per OFDM symbol, slot, subcarrier, and overall EVM. Each plot displays the peak vs RMS EVM.

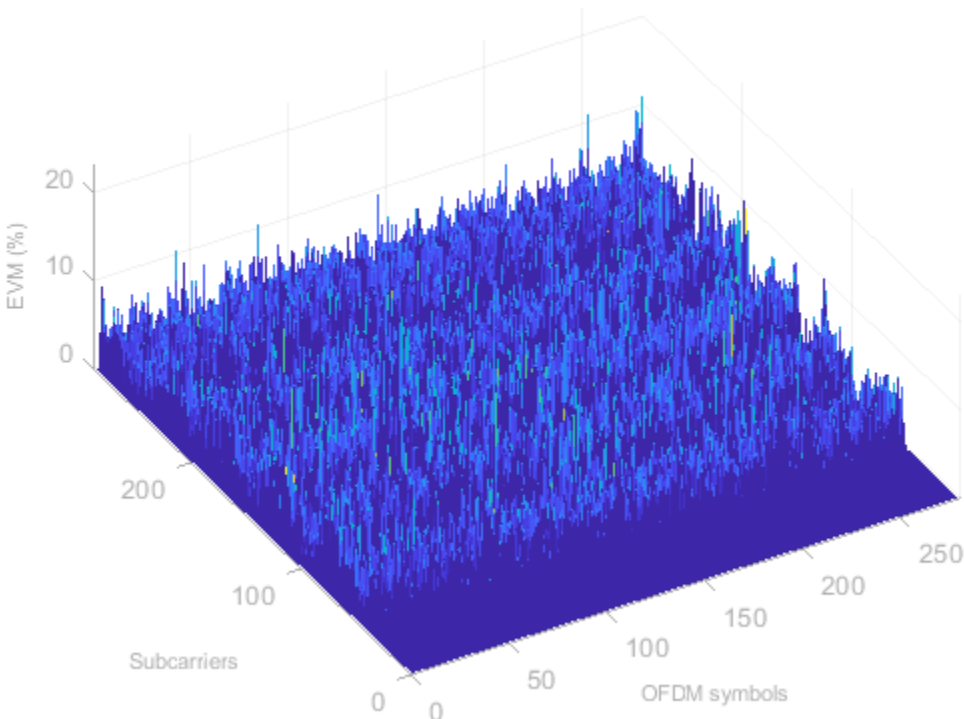
```
% Compute and display EVM measurements
cfg = struct();
cfg.Evm3GPP = evm3GPP;
cfg.TargetRNTIs = targetRNTIs;
cfg.PlotEVM = plotEVM;
cfg.DisplayEVM = displayEVM;
cfg.Label = tmwavegen.ConfiguredModel{1};
[evmGrid,eqSym,refSym,plots] = hNRPDSCEVM(tmwavegen.Config,rxWaveform,cfg,resourcesinfo.Waveform);

RMS EVM, Peak EVM, slot 0: 4.007 20.148%
RMS EVM, Peak EVM, slot 1: 3.854 12.988%
RMS EVM, Peak EVM, slot 2: 3.782 14.258%
RMS EVM, Peak EVM, slot 3: 3.748 14.297%
RMS EVM, Peak EVM, slot 4: 4.158 18.071%
RMS EVM, Peak EVM, slot 5: 4.176 17.451%
RMS EVM, Peak EVM, slot 6: 3.999 15.321%
RMS EVM, Peak EVM, slot 7: 4.141 16.396%
RMS EVM, Peak EVM, slot 8: 3.859 14.277%
RMS EVM, Peak EVM, slot 9: 3.896 12.688%
RMS EVM, Peak EVM, slot 10: 4.039 15.034%
RMS EVM, Peak EVM, slot 11: 3.701 15.019%
RMS EVM, Peak EVM, slot 12: 3.897 12.996%
RMS EVM, Peak EVM, slot 13: 3.959 17.131%
RMS EVM, Peak EVM, slot 14: 3.866 12.698%
RMS EVM, Peak EVM, slot 15: 3.917 17.618%
RMS EVM, Peak EVM, slot 16: 3.808 15.598%
RMS EVM, Peak EVM, slot 17: 4.022 15.583%
RMS EVM, Peak EVM, slot 18: 3.905 21.075%
RMS EVM, Peak EVM, slot 19: 3.976 12.555%
Averaged overall RMS EVM: 3.938%
Peak EVM = 21.0751%
```

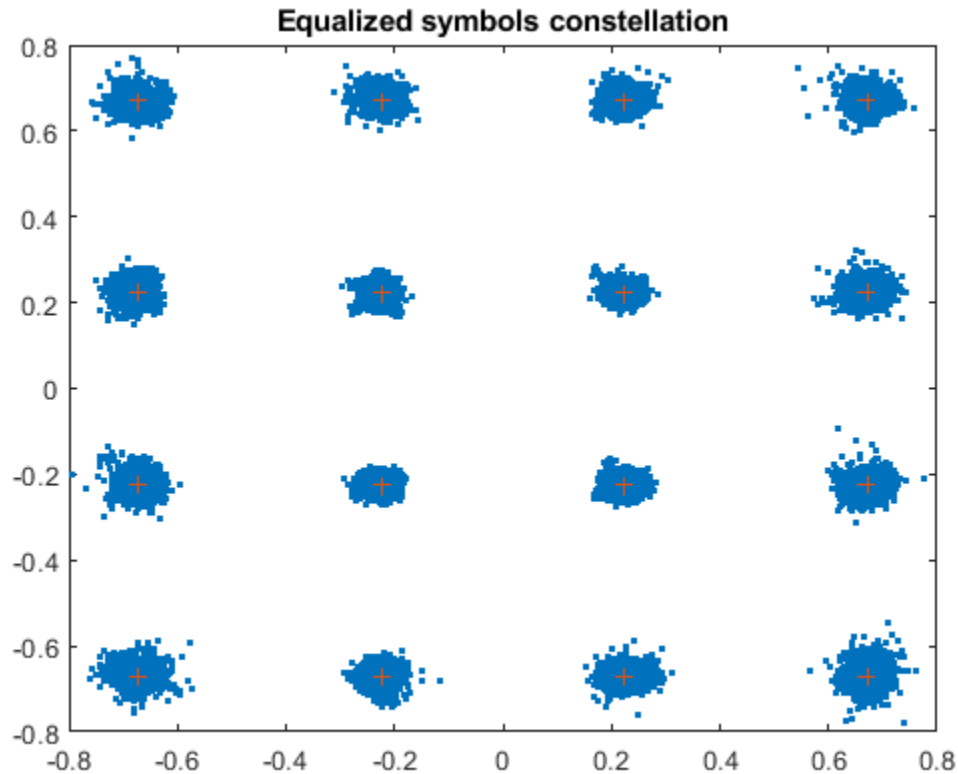












### Local functions

```
function plotNonLinearCharacteristic(memoryLessNonlinearity)
% Plot the non linear characteristic of the power amplifier (PA) impairment
% represented by memoryLessNonlinearity system object. This input parameter
% is a comm.MemorylessNonlinearity system object.

% Input samples
x = complex((1/sqrt(2))*(-1+2*rand(1000,1)), (1/sqrt(2))*(-1+2*rand(1000,1)));

% Non linearity
yRapp = memoryLessNonlinearity(x);
% Release object so we can feed it different number of samples
release(memoryLessNonlinearity);

% Plot characteristic
figure; plot(10*log10(abs(x).^2), 10*log10(abs(x).^2)); hold on; grid on
plot(10*log10(abs(x).^2), 10*log10(abs(yRapp).^2), '.');
xlabel('Input power (dBW)'); ylabel('Output power (dBW)'); title('Nonlinearity impairment')
legend('Linear characteristic', 'Rapp nonlinearity', 'Location', 'Northwest');
```

end

## **See Also**

### **Related Examples**

- “5G NR-TM and FRC Waveform Generation” on page 6-2
- “NR Phase Noise Modeling and Compensation” on page 3-16

# Code Generation and Deployment

---

## What is C Code Generation from MATLAB?

You can use 5G Toolbox together with MATLAB Coder™ to:

- Create a MEX file to speed up your MATLAB application.
- Generate ANSI®/ISO® compliant C/C++ source code that implements your MATLAB functions and models.
- Generate a standalone executable that runs independently of MATLAB on your computer or another platform.

In general, the code you generate using the toolbox is portable ANSI C code. In order to use code generation, you need a MATLAB Coder license. For more information, see “Get Started with MATLAB Coder” (MATLAB Coder).

### Using MATLAB Coder

Creating a MATLAB Coder MEX file can substantially accelerate your MATLAB code. It is also a convenient first step in a workflow that ultimately leads to completely standalone code. When you create a MEX file, it runs in the MATLAB environment. Its inputs and outputs are available for inspection just like any other MATLAB variable. You can then use MATLAB tools for visualization, verification, and analysis.

The simplest way to generate MEX files from your MATLAB code is by using the `codegen` function at the command line. For example, if you have an existing function, `myfunction.m`, you can type the commands at the command line to compile and run the MEX function. `codegen` adds a platform-specific extension to this name. In this case, the “`mex`” suffix is added.

```
codegen myfunction.m  
myfunction_mex;
```

Within your code, you can run specific commands either as generated C code or by using the MATLAB engine. In cases where an isolated command does not yet have code generation support, you can use the `coder.extrinsic` command to embed the command in your code. This means that the generated code reenters the MATLAB environment when it needs to run that particular command. This is also useful if you want to embed commands that cannot generate code (such as plotting functions).

To generate standalone executables that run independently of the MATLAB environment, create a MATLAB Coder project inside the MATLAB Coder Integrated Development Environment (IDE). Alternatively, you can call the `codegen` command in the command line environment with appropriate configuration parameters. A standalone executable requires you to write your own `main.c` or `main.cpp` function. See “Generating Standalone C/C++ Executables from MATLAB Code” (MATLAB Coder) for more information.

### C/C++ Compiler Setup

Before using `codegen` to compile your code, you must set up your C/C++ compiler. For 32-bit Windows platforms, MathWorks® supplies a default compiler with MATLAB. If your installation does not include a default compiler, you can supply your own compiler. For the current list of supported compilers, see Supported and Compatible Compilers on the MathWorks website. Install a compiler that is suitable for your platform, then read “Setting Up the C or C++ Compiler” (MATLAB Coder).

After installation, at the MATLAB command prompt, run `mex -setup`. You can then use the `codegen` function to compile your code.

## Functions and System Objects That Support Code Generation

For an alphabetized list of features supporting C/C++ code generation, see [5G Toolbox - Functions Filtered by C/C++ Code Generation](#).

### See Also

#### Functions

`codegen` | `mex`

### More About

- [“Code Generation Workflow” \(MATLAB Coder\)](#)
- [Generate C Code from MATLAB Code Video](#)

