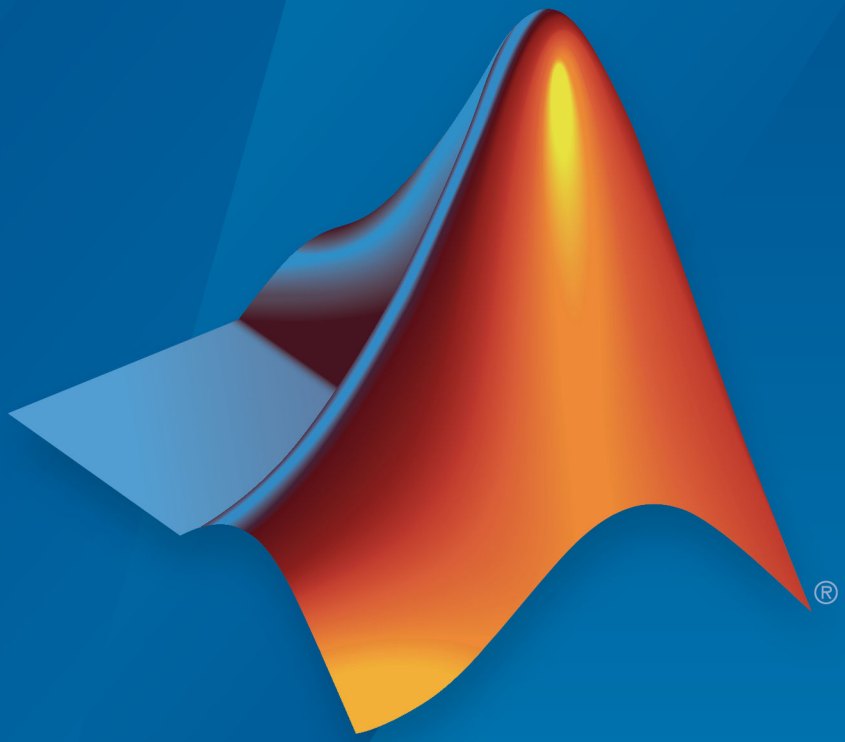
5G Toolbox™

Getting Started Guide



MATLAB®

R2018b

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

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

Phone: 508-647-7000

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

5G Toolbox™ Getting Started Guide

Revision History

September 2018 Online only New for Version 1.0 (Release 2018b)

Contents

1

Getting Started with 5G Toolbox Software

5G Toolbox Product Description

Simulate, analyze, and test the physical layer of 5G communications systems

5G Toolbox provides standard-compliant functions and reference examples for the modeling, simulation, and verification of 5G communications systems. The toolbox supports link-level simulation, golden reference verification and conformance testing, and test waveform generation.

With the toolbox you can configure, simulate, measure, and analyze end-to-end communications links. You can modify or customize the toolbox functions and use them as reference models for implementing 5G systems and devices.

The toolbox provides reference examples to help you explore baseband specifications and simulate the effects of RF designs and interference sources on system performance. You can generate waveforms and customize test benches to verify that your designs, prototypes, and implementations comply with the 3GPP 5G New Radio (NR) standard.

Key Features

- Standard-compliant models for 3GPP 5G NR Release 15
- Link-level simulation with reference examples, including 5G NR PDSCH throughput simulation
- OFDM waveform generation with NR subcarrier spacings and frame numerologies
- TR 38.901 propagation channel models, including tapped delay line (TDL) and clustered delay line (CDL)
- Downlink transport and physical channels (shared, control, broadcast); synchronization and demodulation reference signals
- Signal processing functions, including channel coding (LDPC and polar codes), channel estimation, synchronization, and equalization
- C and C++ code generation support

About 5G

What Is 5G New Radio?

New Radio (NR) is the air interface supporting the next generation of mobile communication, commonly referred to as fifth generation or 5G.

The predecessors of 5G NR are GSM, UMTS, and LTE, also referred to as second generation (2G), third generation (3G), and fourth generation (4G) technologies, respectively. GSM primarily enabled voice calls. The redesigned interfaces of UMTS and LTE enabled and gradually improved mobile broadband connectivity with high data rates and high efficiency.

5G NR continues on the path of LTE by enabling much higher data rates and much higher efficiency for mobile broadband. However, as a response to the demands of *networked society*, the scope of 5G NR goes beyond mobile broadband connectivity. The main requirement of 5G NR is to enable wireless connectivity everywhere, at any time to anyone and anything.

The wide range of use cases that drive 5G NR are classified by three main scenarios.

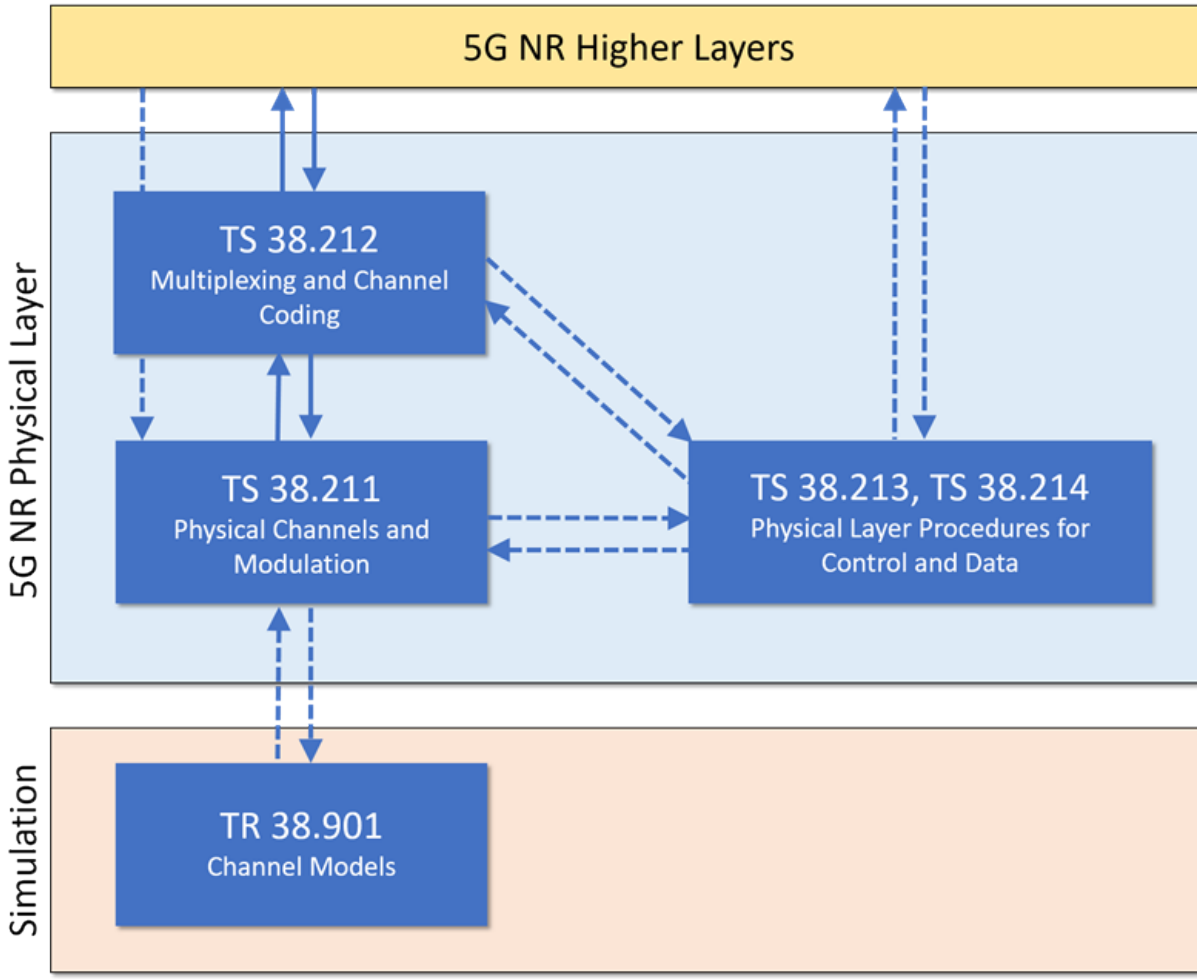
- *Enhanced mobile broadband* (eMBB) — This scenario is still the most important usage scenario that addresses human-centric communications. eMBB use cases have various challenges. For example, hot spots require higher data rates, higher user density, and a need for high capacity. Wide area coverage stresses mobility and seamless user experience with lower requirements on data rate and user density.
- *Massive machine type communications* (mMTC) — This scenario addresses pure machine-centric use cases characterized by a large number of connected devices. Typically, the data rate requirement of mMTC applications is low. However, the use cases demand a high connection density locally, low cost, and long battery life.
- *Ultra reliable and low latency communications* (URLLC) — This scenario covers both human-centric communication and critical machine-type communication (C-MTC) that demand low latency, reliability, and high availability. Typical URLLC use cases include 3-D gaming, self driving cars, mission-critical applications, remote medical surgery, and wireless control of industrial equipment.

This classification is based on presently foreseen use cases and identifies key capabilities of 5G NR. Based on these capabilities, the 5G NR interface is designed to easily adapt to unforeseen use cases that will evolve and emerge over time.

Scope of 5G Toolbox

The 5G NR specification is developed by the Third Generation Partnership Project (3GPP). The first release of the standard was frozen in mid-2018 as 3GPP 5G NR Release 15.

5G Toolbox provides implementations for a subset of the 5G NR physical layer specification and channel model specifications. The following diagram highlights the scope of 5G Toolbox in terms of the addressed specifications and their connectivity.



References

[1] Dalman, E., S. Parkvall, and J. Sköld. *4G, LTE-Advanced Pro and The Road to 5G*. Kidlington, Oxford: Academic Press, 2016.

See Also

More About

- “5G Toolbox and the 5G NR Protocol Layers” on page 2-5
- “What Is LTE?” (LTE Toolbox)

External Websites

- http://www.3gpp.org

5G Toolbox and the 5G NR Protocol Layers

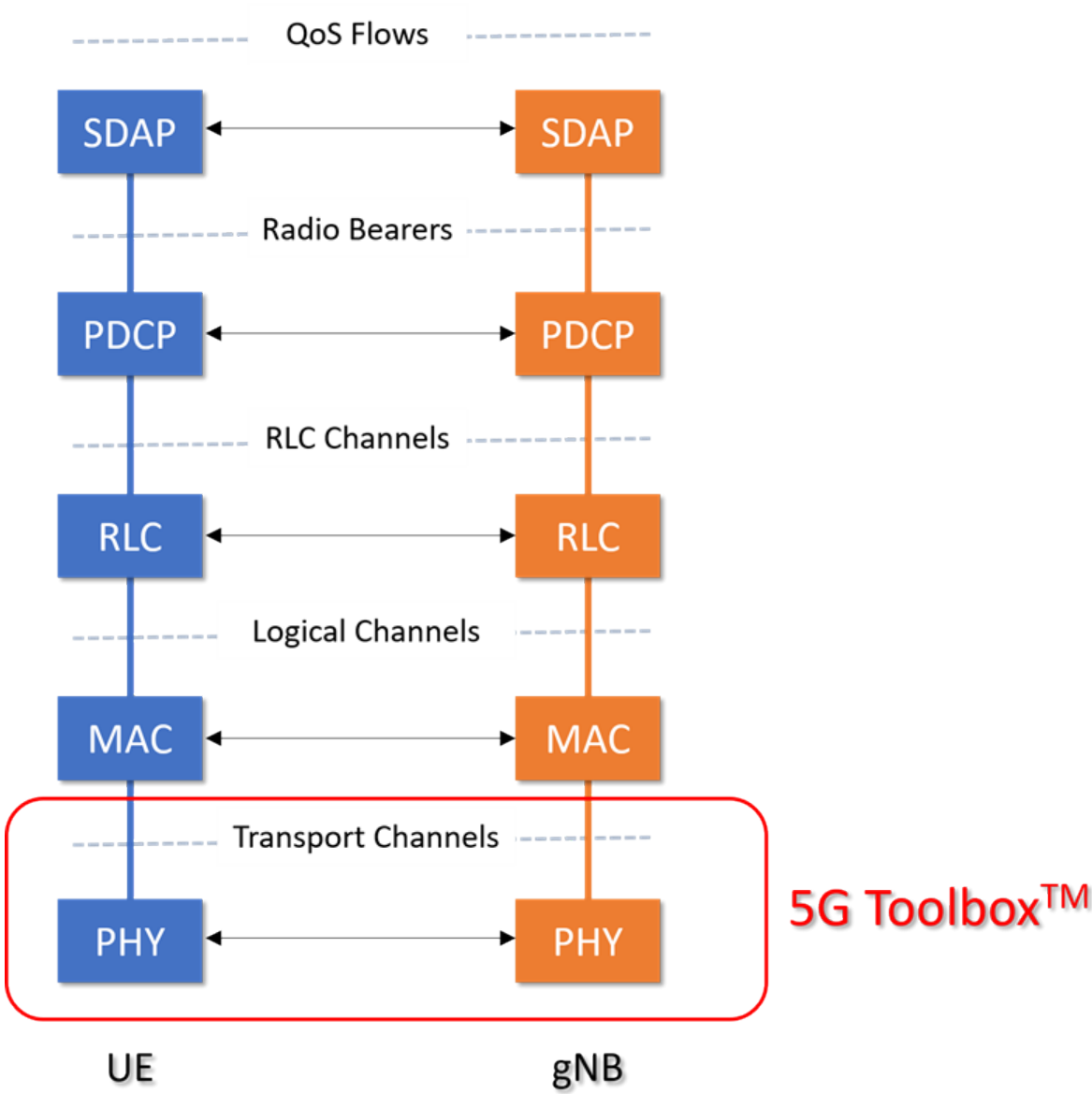
The 5G NR radio access network is comprised of these protocol entities:

- Service data adaptation protocol (SDAP)
- Packet data convergence protocol (PDCP)
- Radio link control (RLC)
- Medium access control (MAC)
- Physical layer (PHY)

The SDAP protocol is new in 5G NR compared to the LTE protocol stack. SDAP handles the new QoS framework of the 5G System (in the 5G Core). SDAP applies also to LTE when connected to the 5G Core. The introduction of SDAP enables end-to-end QoS framework that works in both directions.

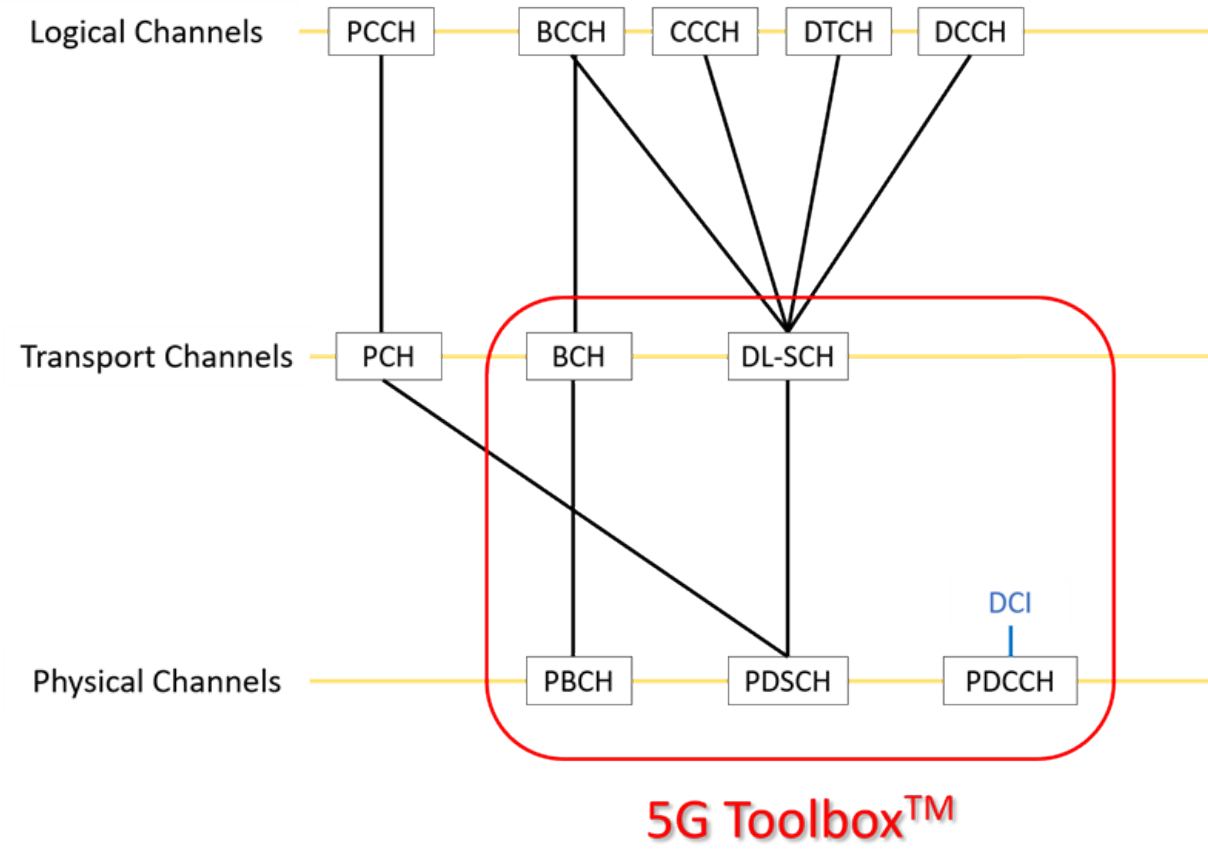
To meet the desired key capabilities of 5G NR, the other layers of the stack provide various enhancements over their LTE counterparts. The PDCP, RLC, and MAC protocols handle tasks such as header compression, ciphering, segmentation and concatenation, and multiplexing and demultiplexing. PHY handles coding and decoding, modulation and demodulation, and antenna mapping.

This figure shows the 5G NR user plane protocol stack for user equipment (UE) and the NR radio access network node (gNB). 5G Toolbox supports the 5G NR physical layer, including physical channels and signals. The toolbox also supports interfacing with portions of the RLC and MAC layers, including transport channels and logical channels.



Downlink Channel Mapping

5G NR system downlink data follows the mapping between logical channels, transport channels, and physical channels, as indicated in the diagram. 5G Toolbox provides the red-highlighted downlink functionality for physical channels, transport channels, and control information.



For more details, see “Downlink Channels” or the specific channel category of interest:

- “Physical Signals”
- “Physical Channels”
- “Transport Channels”

- “Control Information”

References

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

[2] 3GPP TS 38.300. “NR; NR and NG-RAN Overall Description.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

More About

- “What Is 5G New Radio?” on page 2-2

External Websites

- http://www.3gpp.org

3

Tutorials

Downlink Carrier Waveform Generation

This example implements a 5G NR downlink carrier waveform generator using 5G Toolbox™.

Introduction

This example shows how to parameterize and generate a 5G New Radio (NR) downlink waveform. The following channels and signals are generated;

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

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

Carrier Configuration

This section sets the overall carrier bandwidth in resource blocks, the cell ID, and the length of the generated waveform in subframes. You can visualize the generated resource grids by setting the `DisplayGrids` field to 1.

```
carrier = [];
carrier.NDLRB = 200;        % Carrier width in 15kHz numerology
carrier.NCellID = 0;        % Cell identity
carrier.NumSubframes = 10;  % Number of 1ms subframes in generated waveform (1,2,4,8 sl
carrier.DisplayGrids = 1;   % Display the resource grids after signal generation
```

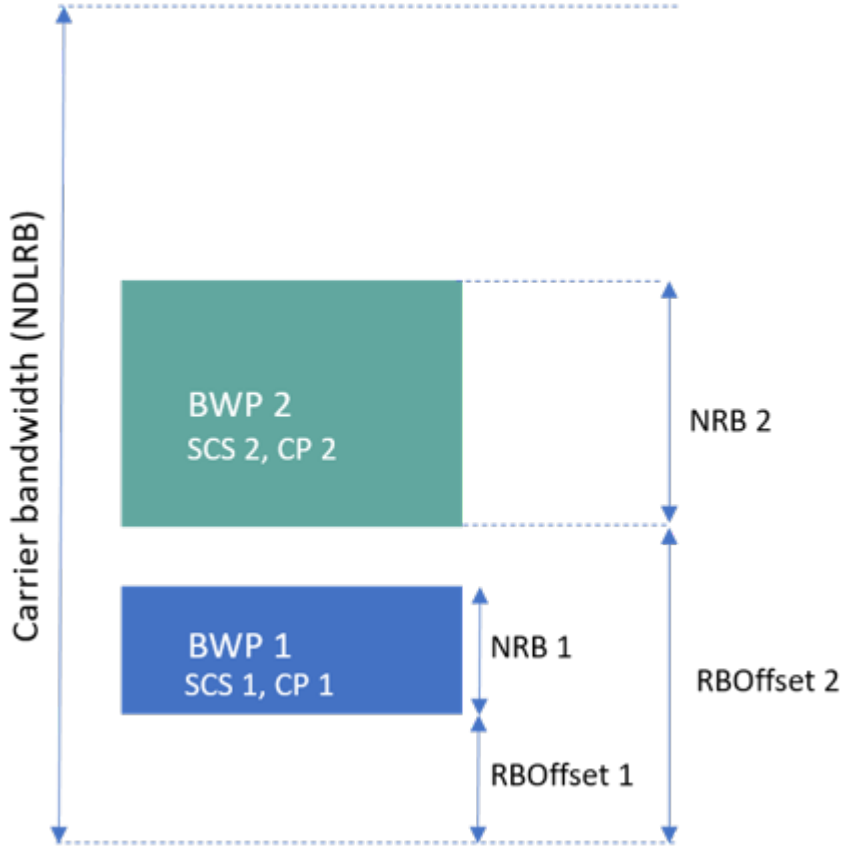
SS Burst

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 TS38.213 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 `hSSBurst`.

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

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 struct array. Each entry in the array represents a BWP. Each BWP can have different subcarrier spacings (SCS), use different cyclic prefix (CP) lengths and span different bandwidths. 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 Subcarrier Spacing
bwp(1).CyclicPrefix = 'Normal';         % BWP Cyclic prefix for 15 kHz
bwp(1).NRB = 25;                        % Size of BWP
bwp(1).RBOffset = 10;                   % Position of BWP in carrier

bwp(2).SubcarrierSpacing = 30;          % BWP Subcarrier Spacing
bwp(2).CyclicPrefix = 'Normal';         % BWP Cyclic prefix for 30 kHz
bwp(2).NRB = 50;                        % Size of BWP
bwp(2).RBOffset = 50;                   % Position of BWP in carrier
```

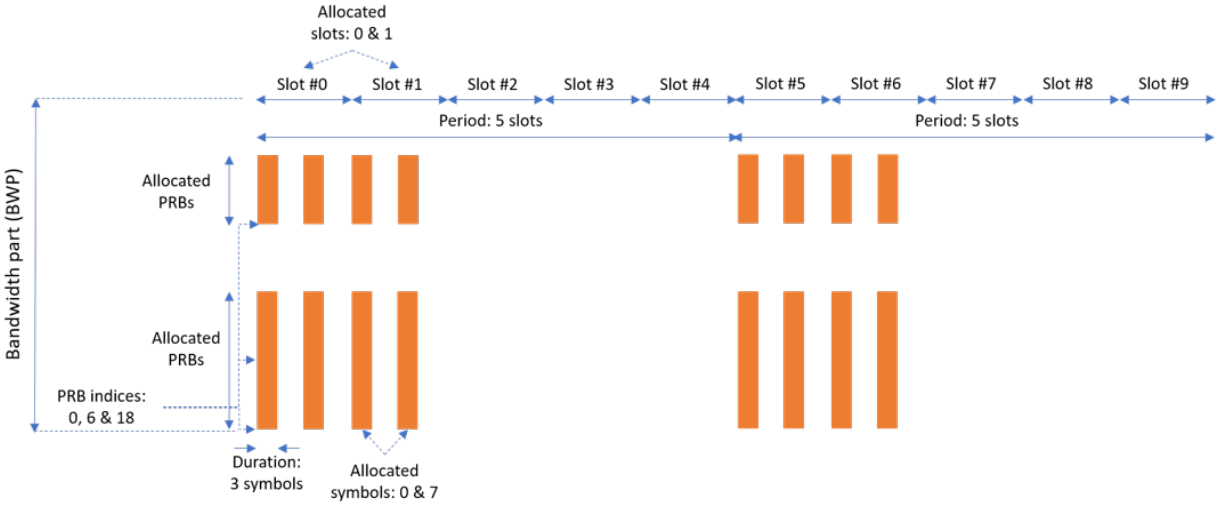
CORESET and Search Space Configuration

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

- Allocated OFDM symbols: specifies the first symbol of each CORESET monitoring opportunity in a slot
- The allocated slots within a period
- Periodicity of the allocation. If this is set to empty it indicates no repetition
- CORESET duration in symbols, either 1, 2 or 3.
- The first PRB of the allocation. Note that the allocation is in blocks of 6 PRBs.

Note that this example only supports non-interleaved CCE-to-REG mapped CORESETs.

The figure below shows the meaning of the CORESET parameters.




```
% CORESET/search configurations
coreset = [];
coreset(1).AllocatedSymbols = [0,7];    % First symbol of each CORESET monitoring oppor
coreset(1).AllocatedSlots = [0,1];      % Allocated slots within a period
coreset(1).AllocatedPeriod = 5;         % Allocated slot period (empty implies no repet
```

```
coreset(1).Duration = 3;                % CORESET symbol duration (1,2,3)
coreset(1).AllocatedPRB = 6*[0,1,3];    % 6 REG sized indices, (RRC - frequencyAllocati
```

PDCCH Instances Configuration

This section specifies the parameters for the different PDCCH instances. Each entry in the struct array defines an instance of the PDCCH. The following parameters can be set:

- Enable/disable the PDCCH instance
- Specify the BWP the PDCCH instance refers to
- PDCCH instance power in dB
- Allocated search spaces: indices within the corresponding CORESET where to map the PDCCH
- CORESET which carries the PDCCH instance
- Periodicity of the allocation. If this is set to empty it indicates no repetition
- Number of control channel elements (CCEs) in this PDCCH
- `NumCCE` and `StartCCE` specify the elements used for the transmission of this 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 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 initialised with all ones

```
pdcch = [];
pdcch(1).Enable = 1;                    % Enable PDCCH config
pdcch(1).BWP = 1;                       % Bandwidth part
pdcch(1).Power = 1.1;                   % Power scaling in dB
pdcch(1).AllocatedSearchSpaces = [0,1]; % Index within the CORESET
pdcch(1).CORESET = 1;                   % Control resource set ID which carries this PD
pdcch(1).AllocatedPeriod = [4];         % Allocation slot period (empty implies no repe
pdcch(1).NumCCE = 30;                   % Number of CCE (in PRBs) in PDCCH
pdcch(1).StartCCE = 10;                 % Starting CCE of PDCCH
pdcch(1).RNTI = 0;                      % RNTI
pdcch(1).NID = 1;                       % PDCCH and DM-RS scrambling NID
pdcch(1).PowerDMRS = 0;                 % Additional power boosting in dB
```

```
pdcch(1).DataBlkSize = 20;              % DCI payload size
pdcch(1).DataSource = 'PN9';            % DCI data source
```

PDSCH Instances Configuration

This section specifies PDSCH instances using a struct array. This example sets two PDSCH instances.

General Parameters

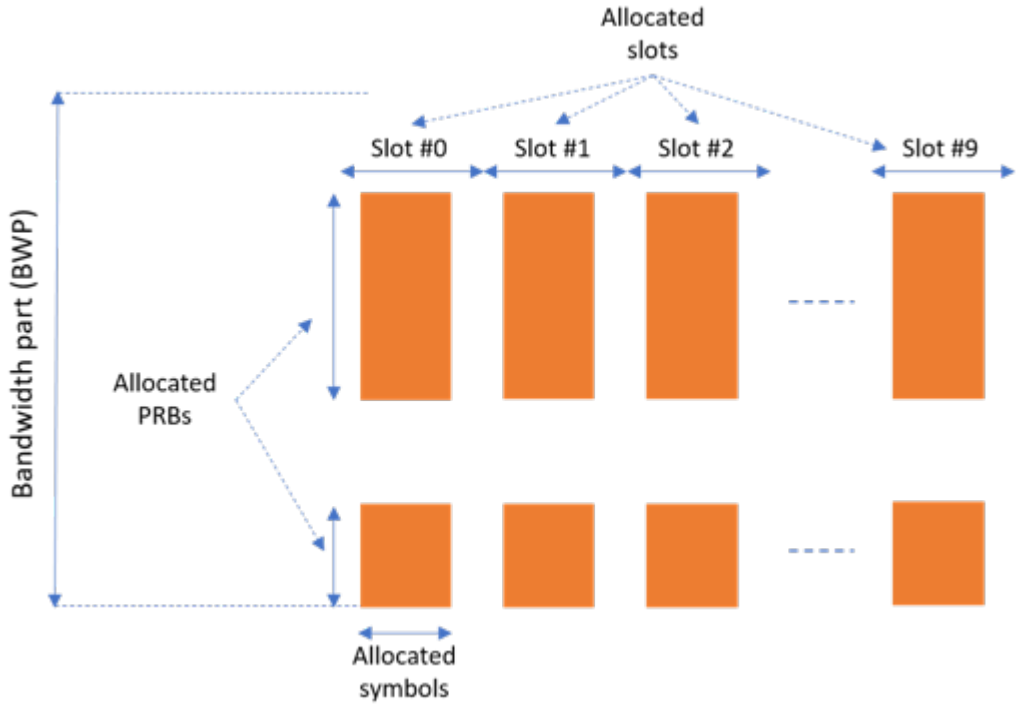
The following parameters are set for each instance:

- Enable/disable this PDSCH instance
- Specify the BWP this PDSCH maps to. The PDSCH will use the SCS specified for this BWP
- Power scaling in dB
- 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 initialised with all ones
- Target code rate used to calculate the transport block sizes
- Overhead parameter
- Symbol modulation
- Number of layers
- Redundancy version (RV) sequence

```
pdsch = [];
pdsch(1).Enable = 1;                    % Enable PDSCH config
pdsch(1).BWP = 1;                       % Bandwidth part
pdsch(1).Power = 0;                     % Power scaling in dB
pdsch(1).DataSource = 'PN9';            % Transport block data source
pdsch(1).TargetCodeRate = 0.4785;       % Code rate used to calculate transport block s
pdsch(1).Xoh_PDSCH = 0;                 % Overhead
pdsch(1).Modulation = 'QPSK';           % 'QPSK', '16QAM', '64QAM', '256QAM'
pdsch(1).NLayers = 2;                   % Number of PDSCH layers
pdsch(1).RVSequence = [0,1,2,3];        % RV sequence to be applied cyclically across t
```

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 where the PDSCH is mapped to. It does not need to be a contiguous allocation
- Slots in a frame used for the 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 links the PDSCH to an instance of the PDCCH
- NID for scrambling the PDSCH bits

```
pdsch(1).AllocatedSymbols = [2:10];     % Range of symbols in a slot
pdsch(1).AllocatedSlots = [0:9];        % Allocated slots indices
pdsch(1).AllocatedPeriod = 15;          % Allocation period in slots (empty implies no
```

```
pdsch(1).AllocatedPRB = [0:5, 10:20];   % PRB allocation
pdsch(1).RNTI = 0;                      % RNTI
pdsch(1).NID = 1;                       % Scrambling for data part
```

The generator in this example does not check for inter-channel conflict. However, additional parameters can be specified for rate matching around other allocations

- The PDSCH can be rate matched around several CORESETs
- The PDSCH can be rate matched around other PDSCH allocations

```
pdsch(1).RateMatch(1).CORESET = [1];                % Rate matching pattern, defined
pdsch(1).RateMatch(1).Pattern.AllocatedPRB = [];    % Rate matching pattern, defined
pdsch(1).RateMatch(1).Pattern.AllocatedSymbols = [];
pdsch(1).RateMatch(1).Pattern.AllocatedSlots = [];
pdsch(1).RateMatch(1).Pattern.AllocatedPeriod = [];
```

DM-RS Configuration

The following DM-RS parameters can be set

```
% DM-RS configuration (TS 38.211 section 7.4.1.1)
pdsch(1).PortSet = 0:pdsch(1).NLayers-1; % DM-RS ports to use for the layers
pdsch(1).PDSCHMappingType = 'A';       % PDSCH mapping type ('A'(slot-wise),'B'(non slo
pdsch(1).DL_DMRS_typeA_pos = 2;        % Mapping type A only. First DM-RS symbol positi
pdsch(1).DL_DMRS_max_len = 1;          % Number of front-loaded DM-RS symbols (1(single
pdsch(1).DL_DMRS_add_pos = 0;          % Additional DM-RS symbol positions (max range (
pdsch(1).DL_DMRS_config_type = 2;      % DM-RS configuration type (1,2)
pdsch(1).NIDNSCID = 1;                 % Scrambling identity (0...65535)
pdsch(1).NSCID = 0;                    % Scrambling initialisation (0,1)
pdsch(1).PowerDMRS = 0;                % Additional power boosting in dB
```

Specifying Multiple PDSCH Instances

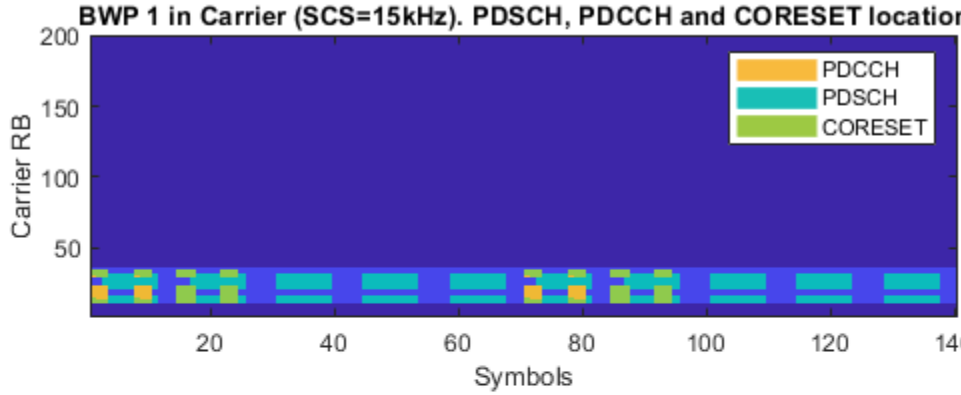
A second PDSCH instance is specified next using the second BWP.

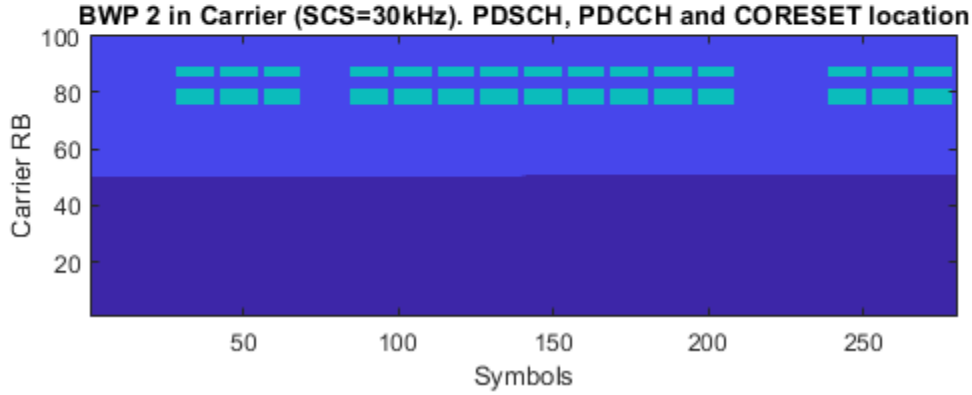
```
pdsch(2) = pdsch(1);
pdsch(2).Enable = 1;
pdsch(2).BWP = 2;                           % PDSCH mapped to 2nd BWP
pdsch(2).AllocatedSymbols = [0:11];
pdsch(2).AllocatedSlots = [2:4,6:20];
pdsch(2).AllocatedPRB = [25:30, 35:38];     % PRB allocation
```

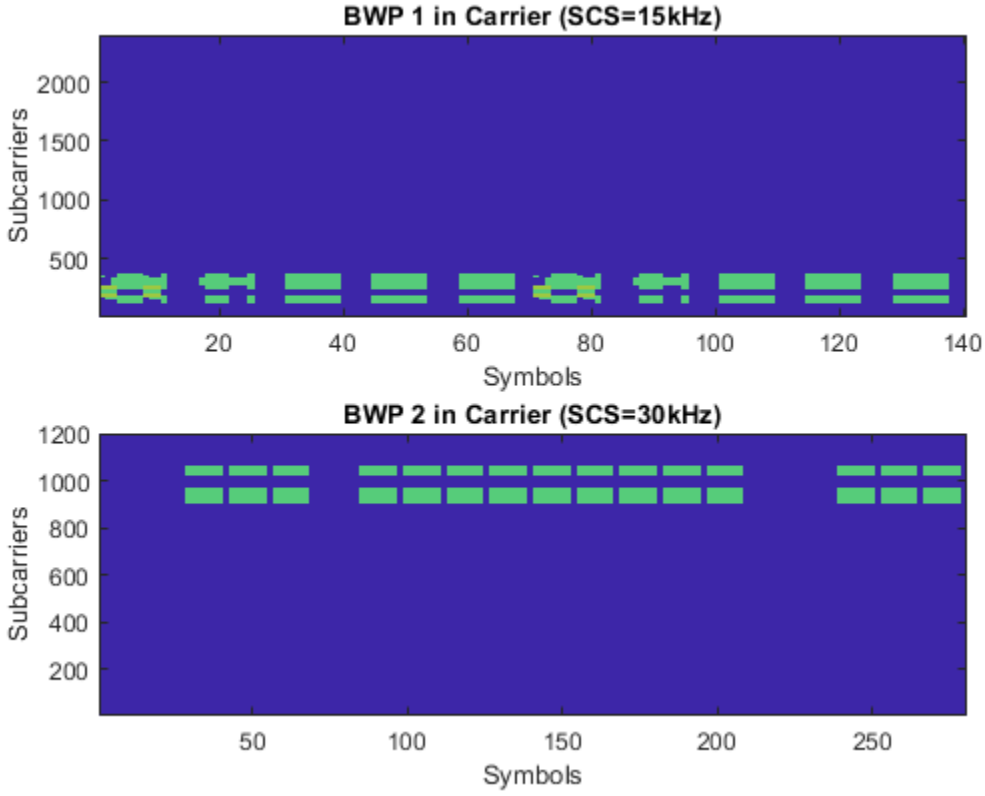
Waveform Generation

This section collects all the parameters into the carrier configuration and generates the waveform.

```
% Collect together channel oriented parameter sets into a single
% configuration
carrier.SSBurst = ssburst;
carrier.BWP = bwp;
carrier.CORESET = coreset;
carrier.PDCCH = pdcch;
carrier.PDSCH = pdsch;

% Generate complex baseband waveform
[waveform,bwpset] = hNRDownlinkWaveformGenerator(carrier);
```


BWP 1 in Carrier (SCS=15kHz). PDSCH, PDCCH and CORESET location
Carrier RB
200
150
100
50
20
40
60
80
100
120
140
Symbols
PDCCH
PDSCH
CORESET


BWP 2 in Carrier (SCS=30kHz). PDSCH, PDCCH and CORESET location
Carrier RB
100
80
60
40
20
50
100
150
200
250
Symbols



The waveform generator also plots 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 (PDCCH, PDSCH, and CORESET) in each BWP. This does not plot the power of the signals, just their location in the grid.
- Generated waveform in the frequency domain for each BWP. This includes the PDCCH and PDSCH instances.

Note that none of these resource grids include the SS burst, this is independent of the BWPs.

The waveform generator function returns the time domain waveform and a struct array `bwpset`, 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(bwpset(1).Info)
```

```
Information associated to BWP 1:
           SamplingRate: 61440000
                   Nfft: 4096
              Windowing: 10
    CyclicPrefixLengths: [1x14 double]
          SymbolLengths: [1x14 double]
           NSubcarriers: 2400
      SubcarrierSpacing: 15
         SymbolsPerSlot: 14
        SlotsPerSubframe: 1
     SymbolsPerSubframe: 14
     SamplesPerSubframe: 61440
         SubframePeriod: 1.0000e-03
              Midpoints: [1x141 double]
          WindowOverlap: [10 10 10 10 10 10 10 10 10 10 10 10 10 10]
```

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`

Synchronization Signal Blocks and Bursts

This example shows how to generate a synchronization signal block (SSB) and generate multiple SSBs to form a synchronization signal burst (SS burst). The channels and signals that form a synchronization signal block (primary and secondary synchronization signals, physical broadcast channel) are created and mapped into a matrix representing the block. Finally a matrix representing a synchronization signal burst is created, and each synchronization signal block in the burst is created and mapped into the matrix.

SS/PBCH block

TS 38.211 Section 7.4.3.1 defines the Synchronization Signal / Physical Broadcast Channel (SS/PBCH) block as 240 subcarriers and 4 OFDM symbols containing the following channels and signals:

- Primary synchronization signal (PSS)
- Secondary synchronization signal (SSS)
- Physical broadcast channel (PBCH)
- PBCH demodulation reference signal (PBCH DM-RS)

In other documents, for example TS 38.331, the SS/PBCH is termed "synchronization signal block" or "SS block".

Create a 240-by-4 matrix representing the SS/PBCH block:

```
ssblock = zeros([240 4])
```

```
ssblock = 240×4

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

Create the PSS for a given cell identity:

```
ncellid = 17;
pssSymbols = nrPSS(ncellid)
```

```
pssSymbols = 127×1

    -1
    -1
    -1
    -1
    -1
    -1
     1
     1
     1
    -1
      ⋮
```

The variable `pssSymbols` is a column vector containing the 127 BPSK symbols of the PSS.

Create the PSS indices:

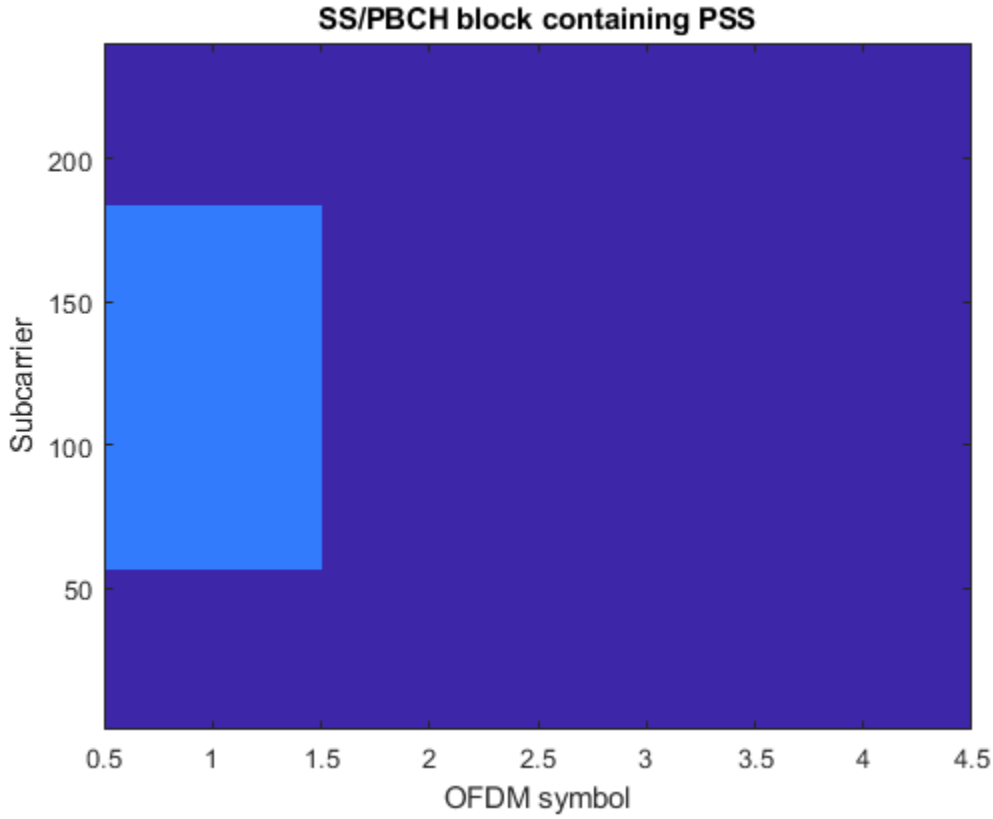
```
pssIndices = nrPSSIndices;
```

The variable `pssIndices` is a column vector of the same size as `pssSymbols`. The value in each element of `pssIndices` is the linear index of the location in the SS/PBCH block to which the corresponding symbols in `pssSymbols` should be mapped. Therefore the mapping of the PSS symbols to the SS/PBCH block can be performed with a simple MATLAB assignment, using linear indexing to select the correct elements of the SS/PBCH block matrix:

```
ssblock(pssIndices) = pssSymbols;
```

Plot the SS/PBCH block matrix to show the location of the PSS:

```
imagesc(abs(ssblock));
caxis([0 4]);
axis xy;
xlabel('OFDM symbol');
ylabel('Subcarrier');
title('SS/PBCH block containing PSS');
```



Create the SSS for the same cell identity as configured for the PSS:

```
sssSymbols = nrSSS(ncellid)
```

```
sssSymbols = 127×1

    -1
     1
    -1
    -1
    -1
     1
    -1
     1
```

```
    -1
     1
      ⋮
```

Create the SSS indices and map the SSS symbols to the SS/PBCH block, following the same pattern used for the PSS. Note that a scaling factor of 2 is applied to the SSS symbols, to represent β_{SSS} in TS 38.211 Section 7.4.3.1.2:

```
sssIndices = nrSSSIndices;
ssblock(sssIndices) = 2 * sssSymbols;
```

The default form of the indices is 1-based linear indices, suitable for linear indexing of MATLAB matrices like `ssblock` as already shown. However, the NR standard documents describe the OFDM resources in terms of OFDM subcarrier and symbol subscripts, using 0-based numbering. For convenient cross-checking with the NR standard, the indices functions accept options to allow the indexing style (linear index versus subscript) and base (0-based versus 1-based) to be selected:

```
sssSubscripts = nrSSSIndices('IndexStyle','subscript','IndexBase','0based')
```

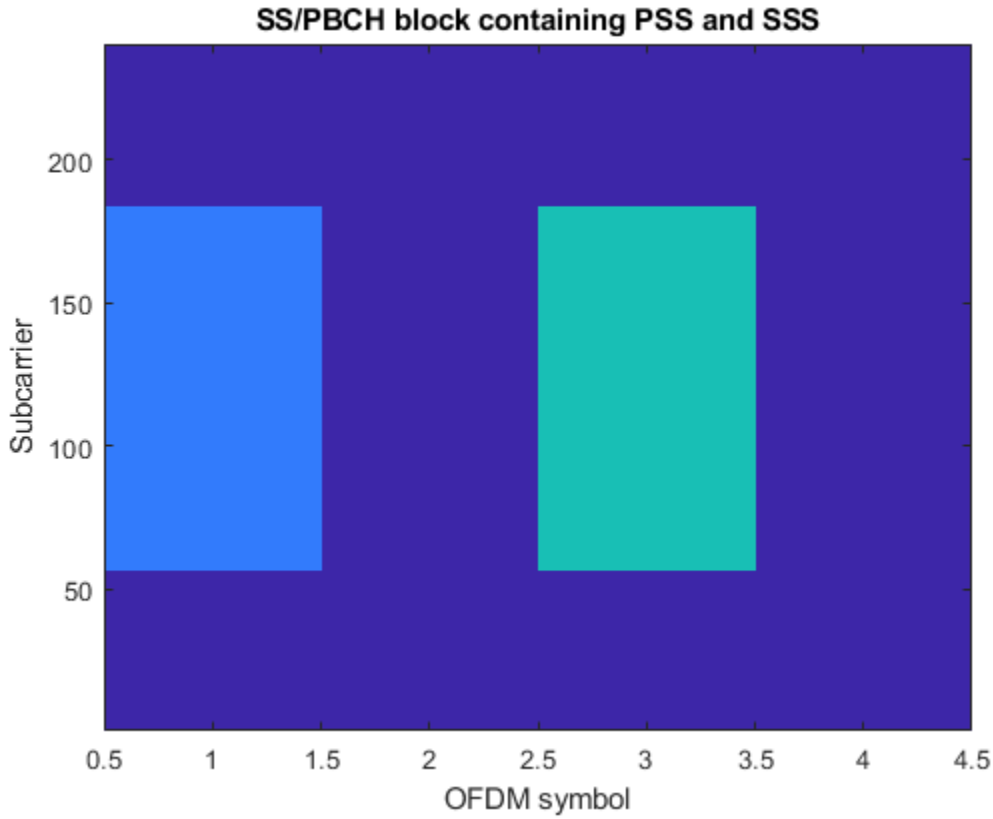
```
sssSubscripts = 127x3 uint32 matrix

   56    2    0
   57    2    0
   58    2    0
   59    2    0
   60    2    0
   61    2    0
   62    2    0
   63    2    0
   64    2    0
   65    2    0
      ⋮
```

It can be seen from the subscripts that the SSS is located in OFDM symbol 2 (0-based) of the SS/PBCH block, starting at subcarrier 56 (0-based).

Plot the SS/PBCH block matrix again to show the locations of the PSS and SSS:

```
imagesc(abs(ssblock));
caxis([0 4]);
axis xy;
```

```
xlabel('OFDM symbol');
ylabel('Subcarrier');
title('SS/PBCH block containing PSS and SSS');
```



The PBCH carries a codeword of length 864 bits, created by performing BCH encoding of the master information block (MIB). For more information on BCH coding, see the functions `nrBCH` and `nrBCHDecode` and their use in the “NR Synchronization Procedures” example. Here a PBCH codeword consisting of 864 random bits is used:

```
cw = randi([0 1],864,1);
```

The PBCH modulation consists of the following steps as described in TS 38.211 Section 7.3.3:

- Scrambling
- Modulation
- Mapping to physical resources

Multiple SS/PBCH blocks are transmitted across half a frame, as described in the cell search procedure in TS 38.213 Section 4.1. Each SS/PBCH block is given an index from $0 \ldots L-1$, where L is the number SS/PBCH blocks in the half frame. The scrambling sequence for the PBCH is initialized according to the cell identity `ncellid`, and the subsequence used to scramble the PBCH codeword depends on the value v, 2 or 3 LSBs of SS/PBCH block index, as described in TS 38.211 Section 7.3.3.1. In this example, $v = 0$ is used. The function `nrPBCH` creates the appropriate subsequence of the scrambling sequence, performs scrambling and then performs QPSK modulation:

```
v = 0;
pbchSymbols = nrPBCH(cw,ncellid,v)
```

```
pbchSymbols = 432×1 complex

  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
  -0.7071 - 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
   0.7071 - 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
      ⋮
```

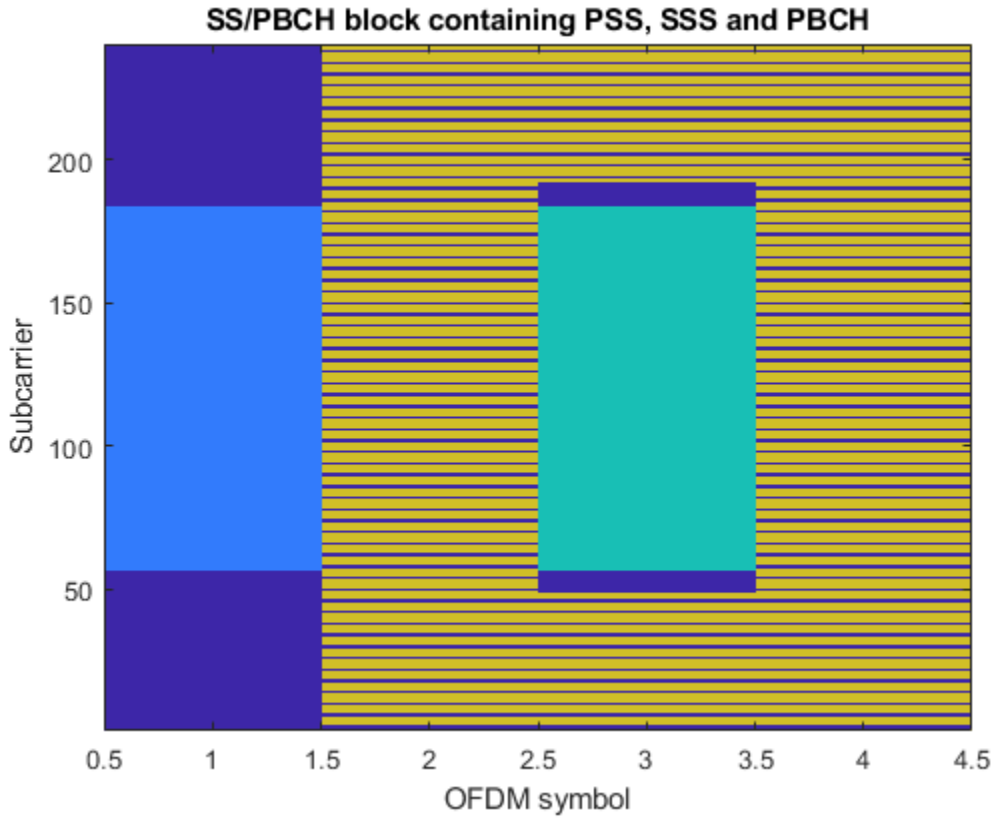
Create the PBCH indices and map the PBCH symbols to the SS/PBCH block:

```
pbchIndices = nrPBCHIndices(ncellid);
ssblock(pbchIndices) = 3 * pbchSymbols;
```

Plot the SS/PBCH block matrix again to show the locations of the PSS, SSS and PBCH:

```
imagesc(abs(ssblock));
caxis([0 4]);
axis xy;
xlabel('OFDM symbol');
```

```
ylabel('Subcarrier');
title('SS/PBCH block containing PSS, SSS and PBCH');
```



The final component of the SS/PBCH block is the DM-RS associated with the PBCH. Similar to the PBCH, the DM-RS sequence used derives from the SS/PBCH block index and is configured using the variable $\bar{i}_{\mathrm{SSB}}$ described in TS 38.211 Section 7.4.1.4.1. Here $\bar{i}_{\mathrm{SSB}} = 0$ is used:

```
ibar_SSB = 0;
dmrsSymbols = nrPBCHDMRS(ncellid,ibar_SSB)
```

```
dmrsSymbols = 144×1 complex
```

```
   0.7071 - 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
   0.7071 - 0.7071i
   0.7071 + 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
   0.7071 + 0.7071i
      ⋮
```

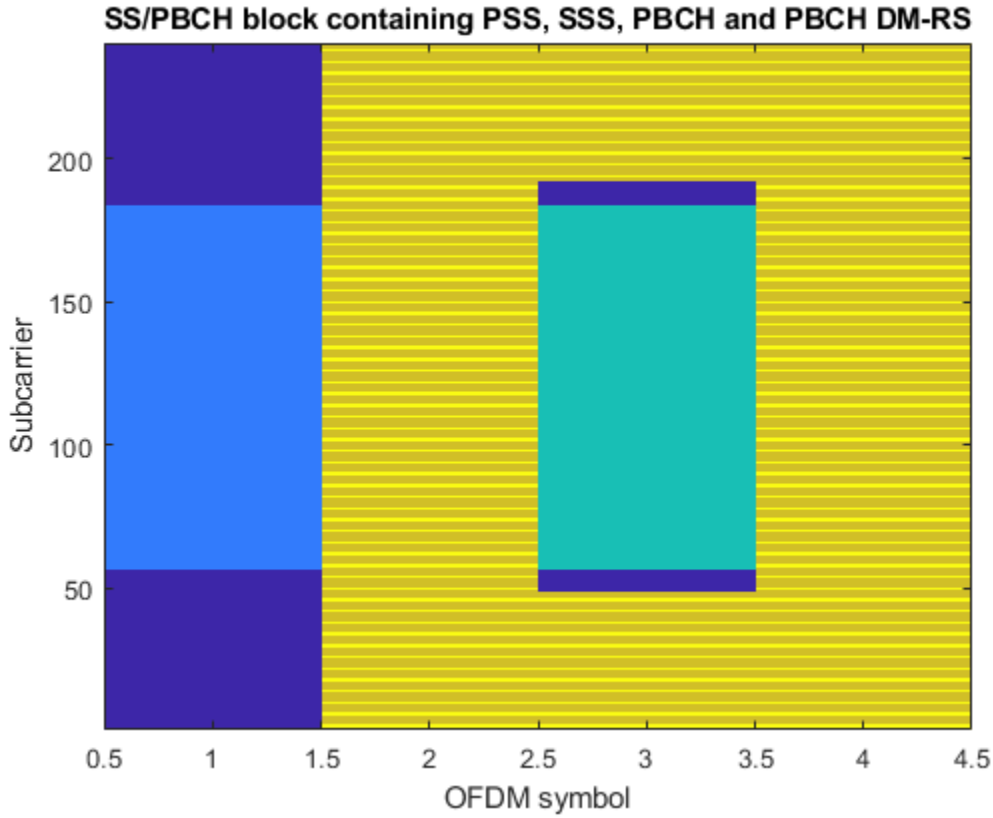
Note that TS 38.211 Section 7.4.1.4.1 defines an intermediate variable i_{SSB} which is defined identically to ν described previously for the PBCH.

Create the PBCH DM-RS indices and map the PBCH DM-RS symbols to the SS/PBCH block:

```
dmrsIndices = nrPBCHDMRSIndices(ncellid);
ssblock(dmrsIndices) = 4 * dmrsSymbols;
```

Plot the SS/PBCH block matrix again to show the locations of the PSS, SSS, PBCH and PBCH DM-RS:

```
imagesc(abs(ssblock));
caxis([0 4]);
axis xy;
xlabel('OFDM symbol');
ylabel('Subcarrier');
title('SS/PBCH block containing PSS, SSS, PBCH and PBCH DM-RS');
```



Generating an SS burst

An SS burst, consisting of multiple SS/PBCH blocks, can be generated by creating a larger grid and mapping SS/PBCH blocks into the appropriate locations, with each SS/PBCH block having the correct parameters according to the location.

In the NR standard, OFDM symbols are grouped into slots, subframes and frames. As defined in TS 38.211 Section 4.3.1, there are 10 subframes in a frame, and each subframe has a fixed duration of 1ms. Each SS burst has a duration of half a frame, and therefore spans 5 subframes:

```
nSubframes = 5
```

```
nSubframes = 5
```

TS 38.211 Section 4.3.2 defines each slot as having 14 OFDM symbols (for normal cyclic prefix length) and this is fixed:

```
symbolsPerSlot = 14
```

```
symbolsPerSlot = 14
```

However, the number of slots per subframe varies and is a function of the subcarrier spacing. As the subcarrier spacing increases, the OFDM symbol duration decreases and therefore more OFDM symbols can be fitted into the fixed subframe duration of 1ms.

There are 5 subcarrier spacing configurations $\mu = 0 \ldots 4$, with the corresponding subcarrier spacing being $15 \cdot 2^{\mu}$ kHz. In this example we shall use $\mu = 1$, corresponding to 30 kHz subcarrier spacing:

```
mu = 1
```

```
mu = 1
```

The number of slots per subframe is 2^{μ}, as doubling the subcarrier spacing halves the OFDM symbol duration. Note that definition of a slot in NR is different from LTE: a subframe in LTE consists of 2 slots of 7 symbols (for normal cyclic prefix) whereas in NR, a subframe using the LTE subcarrier spacing ($\mu = 0$, 15 kHz) consists of 1 slot of 14 symbols.

Calculate the total number of OFDM symbols in an SS burst:

```
nSymbols = symbolsPerSlot * 2^mu * nSubframes
```

```
nSymbols = 140
```

Create an empty grid for the whole SS burst :

```
ssburst = zeros([240 nSymbols])
```

```
ssburst = 240×140
```

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

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

The pattern of SS/PBCH blocks within an SS burst is indirectly specified by the cell search procedure in TS 38.213, which describes the locations in which the UE may detect an SS/PBCH block. There are 5 block patterns, Case A - Case E, which have different subcarrier spacings and are applicable for different carrier frequencies.

Create the indices of the first symbols in the candidate SS/PBCH blocks for block pattern Case B, which has $L = 8$ blocks per burst:

```
n = [0, 1];
firstSymbolIndex = [4; 8; 16; 20] + 28*n;
firstSymbolIndex = firstSymbolIndex(:).'
```

```
firstSymbolIndex = 1×8

     4     8    16    20    32    36    44    48
```

Now a loop can be created which generates each SS block and assigns it into the appropriate location of the SS burst. Note the following:

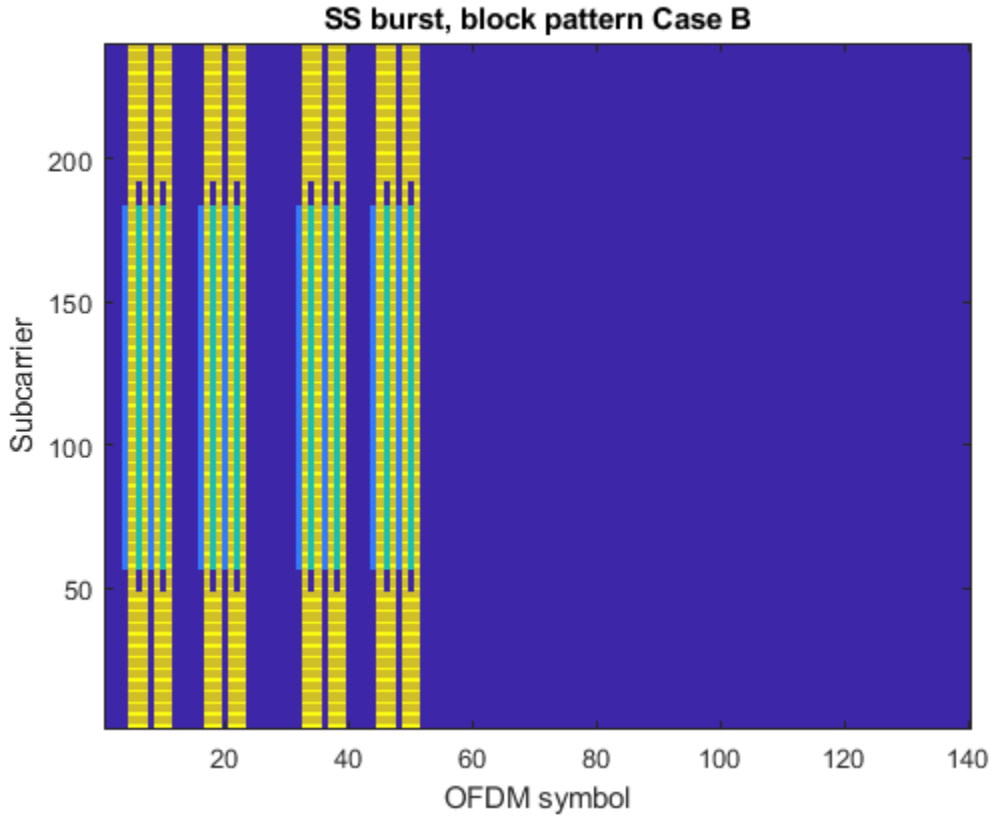
- The code re-uses various variables created earlier in this example (PSS, SSS, and 4 sets of indices)
- The PSS and SSS are independent of the SS/PBCH block index, so can be mapped into the SS block before the loop
- The PBCH indices and PBCH DM-RS indices are independent of the SS/PBCH block index, so do not need updated in the loop
- i_{SSB}, $\bar{i}_{\mathrm{SSB}}$ and v are set up according to the rules in TS 38.211 Sections 7.3.3.1 and 7.4.1.4.1 for the case of $L = 8$.
- Each channel / signal has been scaled in order to give them different colors in the final plot

```
ssblock = zeros([240 4]);
ssblock(pssIndices) = pssSymbols;
```

```
ssblock(sssIndices) = 2 * sssSymbols;

for ssbIndex = 1:length(firstSymbolIndex)

    i_SSB = mod(ssbIndex,8);
    ibar_SSB = i_SSB;
    v = i_SSB;

    pbchSymbols = nrPBCH(cw,ncellid,v);
    ssblock(pbchIndices) = 3 * pbchSymbols;

    dmrsSymbols = nrPBCHDMRS(ncellid,ibar_SSB);
    ssblock(dmrsIndices) = 4 * dmrsSymbols;

    ssburst(:,firstSymbolIndex(ssbIndex) + (0:3)) = ssblock;

end
```

Finally, plot the SS burst content:

```
imagesc(abs(ssburst));
caxis([0 4]);
axis xy;
xlabel('OFDM symbol');
ylabel('Subcarrier');
title('SS burst, block pattern Case B');
```



See Also

Functions

`nrPBCH` | `nrPBCHDMRS` | `nrPBCHDMRSIndices` | `nrPBCHIndices` | `nrPSS` | `nrPSSIndices` | `nrSSS` | `nrSSSIndices`

More About

- “NR Synchronization Procedures”

Modeling Downlink Control Information

This example describes the downlink control information (DCI) processing for the 5G New Radio communications system. Starting from a random DCI message, it models the message encoding followed by the physical downlink control channel (PDCCH) processing on the transmit end. Corresponding receiver components recover the transmitted control information elements.

System Parameters

Set parameters for a UE-specific search space.

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

nID = 23;           % pdcch-DMRS-ScramblingID
rnti = 100;         % C-RNTI for PDCCH in a UE-specific search space
K = 64;             % Number of DCI message bits
E = 288;            % Number of bits for PDCCH resources
```

DCI Encoding

The DCI message bits based on a downlink format are encoded using the `nrDCIEncode` function, which includes the stages of CRC attachment, polar encoding and rate matching.

```
dciBits = randi([0 1],K,1,'int8');
dciCW = nrDCIEncode(dciBits,rnti,E);
```

PDCCH Symbol Generation

The encoded DCI bits (a codeword) are mapped onto the physical downlink control channel (PDCCH) using the `nrPDCCH` function which generates the scrambled, QPSK-modulated symbols. The scrambling accounts for the user-specific parameters.

```
sym = nrPDCCH(dciCW,nID,rnti);
```

For NR, the PDCCH symbols are then mapped to the resource elements of an OFDM grid which also has PDSCH, PBCH and other reference signal elements. These are followed by OFDM modulation and transmission over a channel. For simplicity, we directly pass the PDCCH symbols over an AWGN channel next.

Channel

The PDCCH symbols are transmitted over an AWGN channel with a specified SNR, accounting for the coding rate and QPSK modulation.

```
EbNo = 3;                       % in dB
bps = 2;                        % bits per symbol, 2 for QPSK
EsNo = EbNo + 10*log10(bps);
snrdB = EsNo + 10*log10(K/E);

rxSym = awgn(sym,snrdB,'measured');
```

PDCCH Decoding

The received symbols are demodulated with known user-specific parameters and channel noise variance using the `nrPDCCHDecode` function. The soft output is the log-likelihood ratio for each bit in the codeword.

```
noiseVar = 10.^(-snrdB/10);     % assumes unit signal power
rxCW = nrPDCCHDecode(rxSym,nID,rnti,noiseVar);
```

DCI Decoding

An instance of the received PDCCH codeword is then decoded by the `nrDCIDecode` function. This includes the stages of rate recovery, polar decoding and CRC decoding to recover the transmitted information bits.

```
listLen = 8;                    % polar decoding list length
[decDCIBits,mask] = nrDCIDecode(rxCW,K,listLen);

isequal(mask,rnti)
isequal(decDCIBits,dciBits)
```

```
ans =

  logical

   1


ans =

  logical
```

```
1
```

The output mask matches the C-RNTI used for the UE, thereby confirming the message recipient. For the chosen system parameters, the decoded information matches the transmitted information bits.

See Also

Functions

`nrDCIDecode` | `nrDCIEncode` | `nrPDCCH` | `nrPDCCHDecode`

More About

- “Downlink Control Processing and Procedures”

5G New Radio Polar Coding

This example highlights the new polar channel coding technique chosen for 5G New Radio (NR) communications system. Of the two main types of code constructions specified by 3GPP, this example models the CRC-Aided Polar (CA-Polar) coding scheme. This example describes the main components of the polar coding scheme with individual components for code construction, encoding and decoding along-with rate-matching. It models a polar-coded QPSK-modulated link over AWGN and presents Block-Error-Rate results for different message lengths and code rates for the coding scheme.

Introduction

The selection of polar codes as the channel coding technique for control channels for 5G NR communications system has proven the merits of Arikan's [1] discovery and will establish their application in commercial systems [6]. Based on the concept of channel polarization, this new coding family is capacity achieving as opposed to just capacity approaching. With better or comparable performance than LDPC and turbo codes, it supersedes the tail-biting convolutional codes used in LTE systems for control channels. It is applied for downlink and uplink control information (DCI/UCI) for the enhanced mobile broadband (eMBB) use case, as well as the broadcast channel (BCH). Alternatively, the channel coding scheme for data channels for eMBB is specified to be flexible LDPC for all block sizes.

This example highlights the components to enable a polar coding downlink simulation using QPSK modulation over an AWGN channel. In the following sections, the individual polar coding components are further detailed.

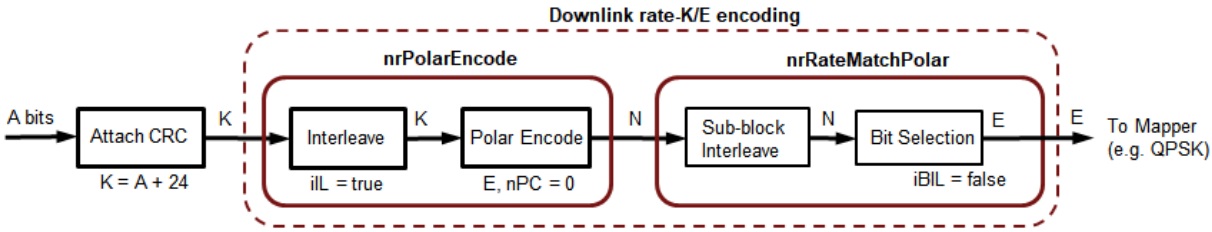
```
s = rng(611);       % Seed the RNG for repeatability
```

Specify the code parameters used for a simulation.

```
% Code parameters
K = 54;             % Message length in bits, including CRC, K > 30
E = 124;            % Rate matched output length, E <= 8192

EbNo = 0.8;         % EbNo in dB
L = 8;              % List length, a power of two, [1 2 4 8]
numFrames = 10;     % Number of frames to simulate
linkDir = 'DL';     % Link direction: downlink ('DL') OR uplink ('UL')
```

Polar Encoding

The following schematic details the transmit-end processing for the downlink, with relevant components and their parameters highlighted.



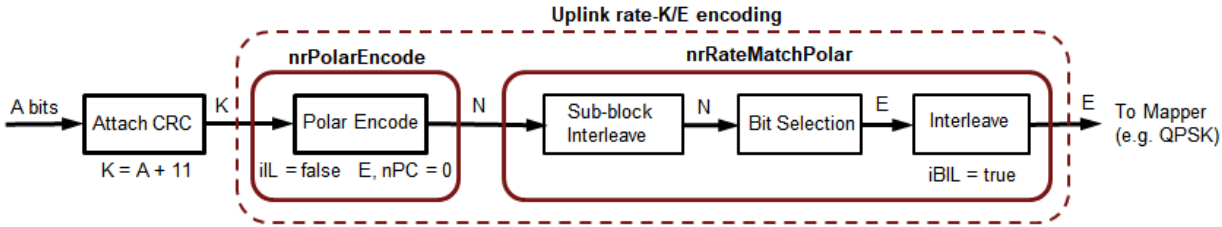
For the downlink, the input bits are interleaved prior to polar encoding. The CRC bits appended at the end of the information bits are thus distributed for the CA-Polar scheme. This interleaving is not specified for the uplink.

The polar encoding uses an SNR-independent method where the reliability of each subchannel is computed offline and the ordered sequence stored for a maximum code length [6]. The nested property of polar codes allows this sequence to be used for any code rate and all code lengths smaller than the maximum code length.

This sequence is computed for given rate-matched output length, `E`, and information length, `K`, by the function `nrPolarEncode`, which implements the non-systematic encoding of the input `K` bits.

```
if strcmp(linkDir,'DL')
    % Downlink scenario (K >= 36, including CRC bits)
    crcLen = 24;      % Number of CRC bits for DL, Section 5.1, [6]
    poly = '24C';     % CRC polynomial
    nPC = 0;          % Number of parity check bits, Section 5.3.1.2, [6]
    nMax = 9;         % Maximum value of n, for 2^n, Section 7.3.3, [6]
    iIL = true;       % Interleave input, Section 5.3.1.1, [6]
    iBIL = false;     % Interleave coded bits, Section 5.4.1.3, [6]
else
    % Uplink scenario (K > 30, including CRC bits)
    crcLen = 11;
    poly = '11';
    nPC = 0;
    nMax = 10;
    iIL = false;
    iBIL = true;
end
```

The following schematic details the transmit-end processing for the uplink, for a payload size greater than 19 bits and no code-block segmentation, with relevant components and their parameters highlighted.



Rate Matching and Rate Recovery

The polar encoded set of bits (`N`) are rate-matched to output the specified number of bits (`E`) for resource element mapping [7]. The coded bits are sub-block interleaved and passed to a circular buffer of length `N`. Depending on the desired code rate and selected values of `K`, `E`, and `N`, either of repetition (`E >= N`), and puncturing or shortening (`E < N`) is realized by reading the output bits from the buffer.

- For puncturing, `E` bits are taken from the end
- For shortening, `E` bits are taken from the start
- For repetition, `E` bits are repeated modulo `N`.

For the downlink, the selected bits are passed on to the modulation mapper, while for the uplink, they are further interleaved prior to mapping. The rate-matching processing is implemented by the function `nrRateMatchPolar`.

At the receiver end, rate recovery is accomplished for each of the cases

- For puncturing, corresponding LLRs for the bits removed are set to zero
- For shortening, corresponding LLRs for the bits removed are set to a large value
- For repetition, the set of LLRs corresponding to first `N` bits are selected.

The rate-recovery processing is implemented by the function `nrRateRecoverPolar`.

```
R = K/E;                          % Effective code rate
bps = 2;                          % bits per symbol, 1 for BPSK, 2 for QPSK
EsNo = EbNo + 10*log10(bps);
snrdB = EsNo + 10*log10(R);       % in dB
noiseVar = 1./(10.^(snrdB/10));
```

```
% Modulator, Channel, Demodulator
qpskMod = comm.QPSKModulator('BitInput',true);
chan = comm.AWGNChannel('NoiseMethod','Variance','Variance',noiseVar);
qpskDemod = comm.QPSKDemodulator('BitOutput',true,'DecisionMethod', ...
    'Approximate log-likelihood ratio','Variance',noiseVar);
```

Polar Decoding

The implicit CRC encoding of the downlink (DCI or BCH) message bits dictates the use of the CRC-Aided Successive Cancellation List Decoding (CA-SCL) [3] as the channel decoder algorithm. It is well known that CA-SCL decoding can outperform turbo or LDPC codes and this was one of the major factors in the adoption of polar codes by 3GPP.

Tal & Vardy [2] describe the SCL decoding algorithm in terms of likelihoods (probabilities). However, due to underflow, the inherent computations are numerically unstable. To overcome this issue, Stimming et.al. [5] offer the SCL decoding solely in the log-likelihood ratio (LLR) domain. The list decoding is characterized by the L parameter, which represents the number of most likely decoding paths retained. At the end of the decoding, the most likely code-path among the L paths is the decoder output. As L is increased, the decoder performance also improves, however, with a diminishing-returns effect.

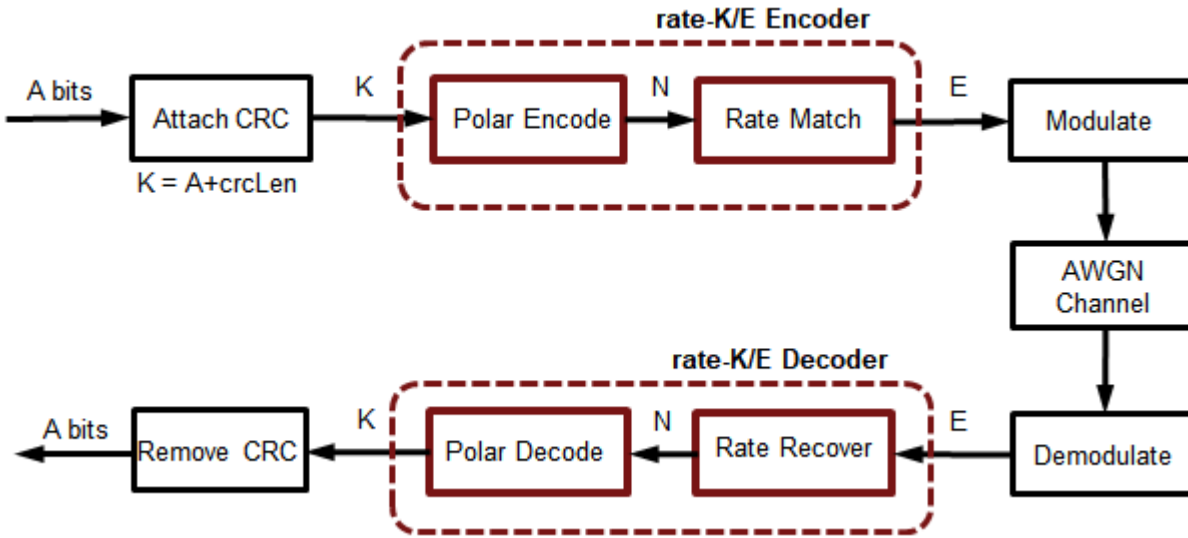
For an input message which is concatenated with a CRC, CA-SCL decoding prunes out any of the paths for which the CRC is invalid, if at least one path has the correct CRC. This additional insight in the final path selection improves the performance further, when compared to SCL decoding. For the downlink, a CRC of 24 bits is used, while for the uplink CRCs of 6 and 11 bits are specified, which vary on the value of K.

The decoder is implemented by the function `nrPolarDecode`, which supports CRC lengths of 11 and 24 bits. The decoder function also accounts for the input bit interleaving specified at the transmitter for the downlink, prior to outputting the decoded bits.

```
% Error meter
ber = comm.ErrorRate;
```

Frame Processing Loop

This section shows how the prior described components for polar coding are used in a Block Error Rate (BLER) simulation. The simulation link is highlighted in the following schematic.



For each frame processed, the following steps are performed:

- `K-crcLen` random bits are generated,
- A CRC is computed and appended to these bits
- The CRC appended bits are polar encoded to the mother code block length
- Rate-matching is performed to transmit `E` bits
- The `E` bits are QPSK modulated
- White Gaussian Noise of specified power is added
- The noisy signal is soft QPSK demodulated to output LLR values
- Rate recovery is performed accounting for either of puncturing, shortening or repetition
- The recovered LLR values are polar decoded using the CA-SCL algorithm, including deinterleaving.
- Off the decoded `K` bits, the first `K-crcLen` bits are compared with those transmitted to update the BLER and bit-error-rate (BER) metrics.

At the end of the simulation, the two performance indicators, BLER and BER, are reported.

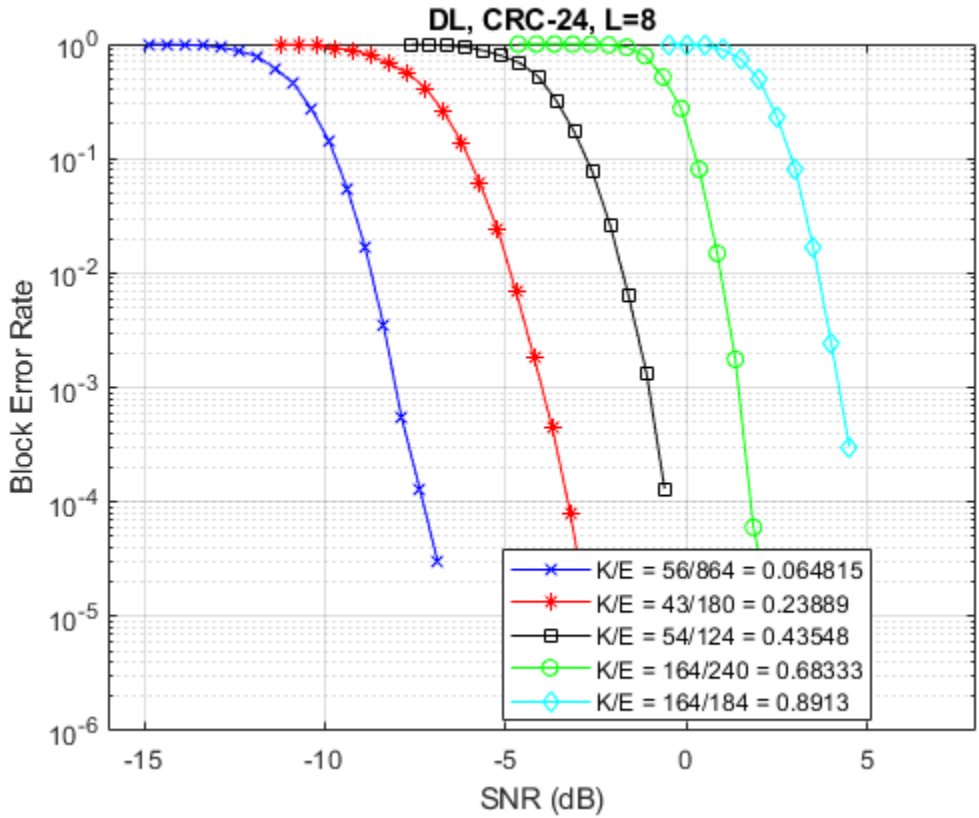
```
numferr = 0;
for i = 1:numFrames
```

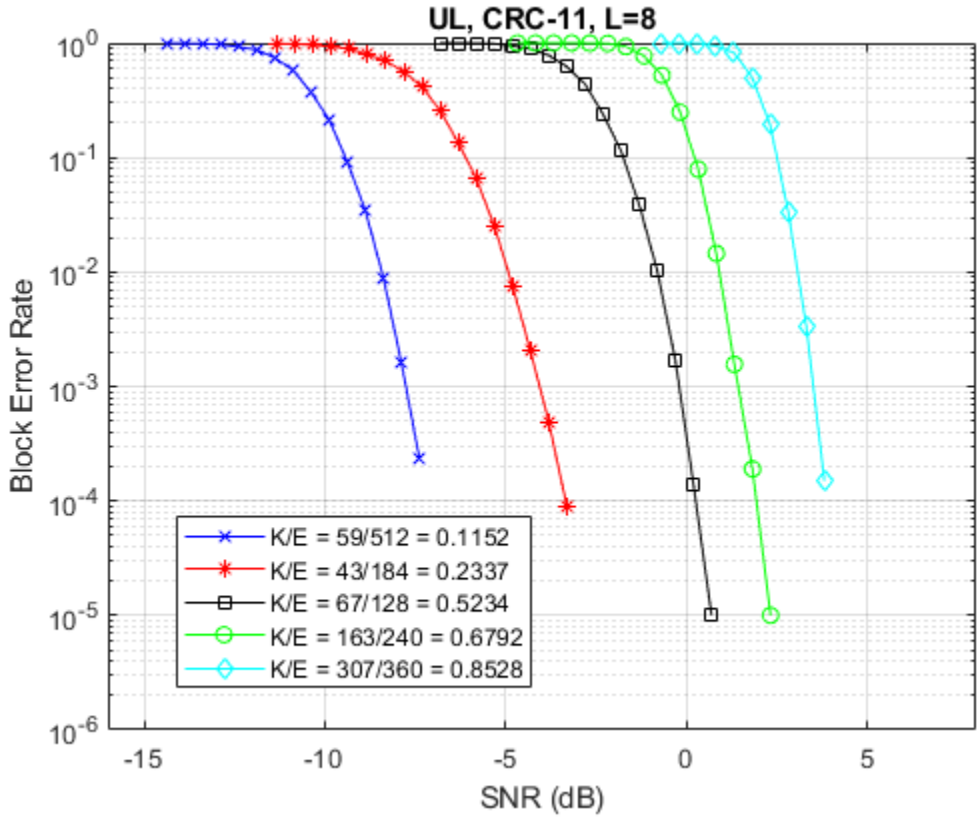
```
    % Generate a random message
    msg = randi([0 1],K-crcLen,1);

    % Attach CRC
    msgcrc = nrCRCEncode(msg,poly);

    % Polar encode
    encOut = nrPolarEncode(msgcrc,E,nMax,iIL);
    N = length(encOut);

    % Rate match
    modIn = nrRateMatchPolar(encOut,K,E,iBIL);

    % Modulate
    modOut = qpskMod(modIn);

    % Add White Gaussian noise
    rSig = chan(modOut);

    % Soft demodulate
    rxLLR = qpskDemod(rSig);

    % Rate recover
    decIn = nrRateRecoverPolar(rxLLR,K,N,iBIL);

    % Polar decode
    decBits = nrPolarDecode(decIn,K,E,L,nMax,iIL,crcLen);

    % Compare msg and decoded bits
    errStats = ber(double(decBits(1:K-crcLen)), msg);
    numferr = numferr + any(decBits(1:K-crcLen)~=msg);

end

disp(['Block Error Rate: ' num2str(numferr/numFrames) ...
      ', Bit Error Rate: ' num2str(errStats(1)) ...
      ', at SNR = ' num2str(snrdB) ' dB'])

rng(s);     % Restore RNG
```

```
Block Error Rate: 0, Bit Error Rate: 0, at SNR = 0.20002 dB
```

Results

To get meaningful results, simulations have to be run for a longer duration. Using scripts which encapsulate the above processing into a function that supports C-code generation, the following results for different code rates and message lengths are presented for both link directions.





The above results were generated by simulating, for each SNR point, up to 1000 frame errors or a maximum of 100e3 frames, whichever occurred first.

The BLER performance results indicate the suitability of polar codes in a communication link and their implicit support for rate-compatibility at the bit-level granularity.

The use of C-code generation tools for the components reduces the execution time, a key concern for simulations. The C-code generation is enabled by MATLAB Coder™.

Summary and Further Exploration

This example highlights one of the polar coding schemes (CRC-Aided Polar) specified by 3GPP for New Radio control channel information (DCI, UCI) and broadcast channel (BCH). It shows the use of components for all stages of the processing (encoding, rate-

matching, rate-recovery and decoding) and uses them in a link with QPSK over an AWGN channel. Highlighted performance results for different code rates and message lengths show agreement to published trends, within parametric and simulation assumption variations.

Explore simple parameter variations (`K`, `E`, `L`) and their effect on BLER performance. The polar coding functions are implemented as open MATLAB® code to enable their application for both downlink/uplink control information and broadcast channel. The CA-Polar scheme is applicable for both

- Downlink, for all message lengths, and
- Uplink, for `K > 30`, with `crcLen = 11`, `nPC = 0`, `nMax = 10`, `iIL = false`, and `iBIL = true`.

Refer to “Modeling Downlink Control Information” on page 3-27 and “NR Synchronization Procedures” examples, for the use of polar coding functions within the DCI and BCH functions respectively.

Selected References

1. Arikan, E., "Channel Polarization: A Method for constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels," IEEE Transactions on Information Theory, vol. 55, No. 7, pp. 3051-3073, July 2009.
2. Tal, I, and Vardy, A., "List decoding of Polar Codes", IEEE Transactions on Information Theory, vol. 61, No. 5, pp. 2213-2226, May 2015.
3. Niu, K., and Chen, K., "CRC-Aided Decoding of Polar Codes," IEEE Communications Letters, vol. 16, No. 10, pp. 1668-1671, Oct. 2012.
4. Niu, K., Chen, K., and Lin, J.R., "Beyond turbo codes: rate compatible punctured polar codes", IEEE International Conference on Communications, pp. 3423-3427, 2013.
5. Stimming, A. B., Parizi, M. B., and Burg, A., "LLR-Based Successive Cancellation List Decoding of Polar Codes", IEEE Transaction on Signal Processing, vol. 63, No. 19, pp. 5165-5179, 2015.
6. 3GPP TS 38.212. "NR; Multiplexing and channel coding (Release 15)." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.

7 R1-1711729. "WF on circular buffer of Polar Code", 3GPP TSG RAN WG1 meeting NR Ad-Hoc#2, Ericsson, Qualcomm, MediaTek, LGE. June 2017.

See Also

Functions

`nrPolarDecode` | `nrPolarEncode` | `nrRateMatchPolar` | `nrRateRecoverPolar`

More About

- “Modeling Downlink Control Information” on page 3-27
- “NR Synchronization Procedures”

LDPC Processing for DL-SCH

This example highlights the low-density parity-check (LDPC) coding chain for the 5G NR downlink shared transport channel (DL-SCH).

DL-SCH Parameters

Set parameters for downlink shared (DL-SCH) channel.

```
rng(210);               % Set RNG state for repeatability

A = 10000;              % Transport block length, positive integer
rate = 449/1024;        % Target code rate, 0<R<1
rv = 0;                 % Redundancy version, 0-3
modulation = 'QPSK';    % Modulation scheme, QPSK, 16QAM, 64QAM, 256QAM
nlayers = 1;            % Number of layers, 1-4
```

Based on the selected transport block length and target coding rate, DL-SCH coding parameters are determined using the `nrDLSCHInfo` function.

```
% DL-SCH coding parameters
cbsInfo = nrDLSCHInfo(A,rate);
disp('DL-SCH coding parameters')
disp(cbsInfo)
```

```
DL-SCH coding parameters
    CRC: '24A'
      L: 24
    BGN: 1
      C: 2
    Lcb: 24
      F: 244
     Zc: 240
      K: 5280
      N: 15840
```

Transport Block Processing using LDPC Coding

Data delivered from the MAC layer to the physical layer is termed as a transport block. For the downlink shared channel (DL-SCH), a transport block goes through the processing stages of:

- CRC attachment,
- Code block segmentation and code block CRC attachment,
- Channel coding using LDPC,
- Rate matching and code block concatenation

before being passed on to the physical downlink shared channel (PDSCH) for scrambling, modulation, layer mapping and resource/antenna mapping. Each of these stages is performed by a function as shown next.

```
% Random transport block data generation
in = randi([0 1],A,1,'int8');

% Transport block CRC attachment
tbIn = nrCRCEncode(in,cbsInfo.CRC);

% Code block segmentation and CRC attachment
cbsIn = nrCodeBlockSegmentLDPC(tbIn,cbsInfo.BGN);

% LDPC encoding
enc = nrLDPCEncode(cbsIn,cbsInfo.BGN);

% Rate matching and code block concatenation
punctLen = 200;     % puncturing length, if non-zero, repLen must be zero
repLen = 0;         % repetition length, if non-zero, punctLen must be zero
outlen = numel(enc)-punctLen+repLen;
chIn = nrRateMatchLDPC(enc,outlen,rv,modulation,nlayers);
```

The output number of bits from the rate matching and code block concatenation process must match the bit capacity of the PDSCH, based on the available resources. In this example, as the PDSCH is not modeled, one can select either of puncturing or repetition or full coded block transmission using the `punctLen`, and `repLen` parameters to specify the respective lengths.

Channel

A simple bipolar channel with no noise is used for this example. With the full PDSCH processing, one can consider fading channels, AWGN and other RF impairments as well.

```
chOut = double(1-2*(chIn));
```

Receive Processing using LDPC Decoding

The receive end processing for the DL-SCH channel comprises of the corresponding dual operations to the transmit end that include

- Rate recovery
- LDPC decoding
- Code block desegmentation and CRC decoding
- Transport block CRC decoding

Each of these stages is performed by a function as shown next.

```
% Rate recovery
raterec = nrRateRecoverLDPC(chOut,A,rate,rv,modulation,nlayers);

% LDPC decoding
decBits = nrLDPCDecode(raterec,cbsInfo.BGN,25);

% Code block desegmentation and CRC decoding
[blk,blkErr] = nrCodeBlockDesegmentLDPC(decBits,cbsInfo.BGN,A+cbsInfo.L);

disp(['CRC error per code-block: [' num2str(blkErr) ']'])

% Transport block CRC decoding
[out,tbErr] = nrCRCDecode(blk,cbsInfo.CRC);

disp(['Transport block CRC error: ' num2str(tbErr)])
disp(['Recovered transport block with no error: ' num2str(isequal(out,in))])
```

```
CRC error per code-block: [0  0]
Transport block CRC error: 0
Recovered transport block with no error: 1
```

As the displays indicate, there are no CRC errors at both the code-block and transport block levels. This leads to the transport block being recovered and decoded with no errors, as expected, for a noiseless channel.

See Also

Functions

`nrCRCDecode` | `nrCRCEncode` | `nrCodeBlockDesegmentLDPC` | `nrCodeBlockSegmentLDPC` | `nrDLSCHInfo` | `nrLDPCDecode` | `nrLDPCEncode` | `nrRateMatchLDPC` | `nrRateRecoverLDPC`

More About

- “NR PDSCH Throughput”