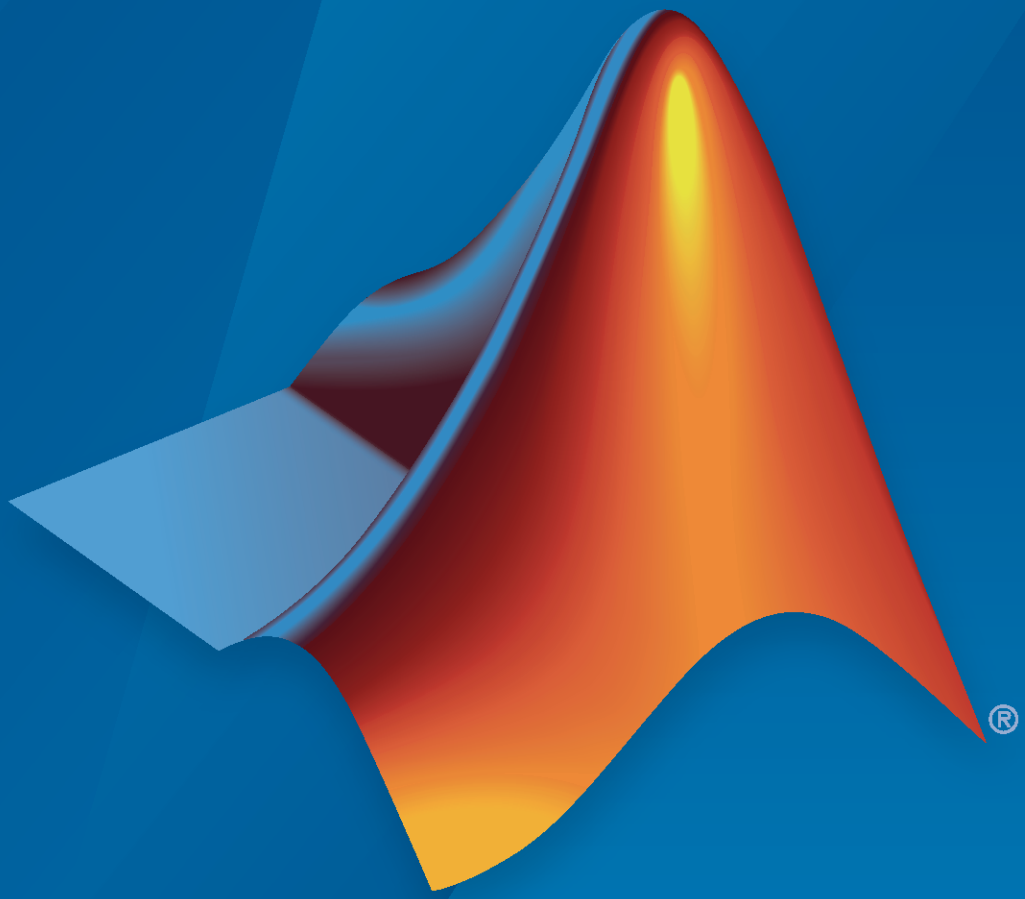
WLAN Toolbox™

Reference



MATLAB®

R2022b

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

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

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

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

Phone: 508-647-7000

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

WLAN Toolbox™ Reference

© COPYRIGHT 2015–2022 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` 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` for more information.

Revision History

October 2015	Online only	New for Version 1.0 (Release 2015b)
March 2016	Online only	Revised for Version 1.1 (Release 2016a)
September 2016	Online only	Revised for Version 1.2 (Release 2016b)
March 2017	Online only	Revised for Version 1.3 (Release 2017a)
September 2017	Online only	Revised for Version 1.4 (Release 2017b)
March 2018	Online only	Revised for Version 1.5 (Release 2018a)
September 2018	Online only	Revised for Version 2.0 (Release 2018b)
March 2019	Online only	Revised for Version 2.1 (Release 2019a)
September 2019	Online only	Revised for Version 2.2 (Release 2019b)
March 2020	Online only	Revised for Version 3.0 (Release 2020a)
September 2020	Online only	Revised for Version 3.1 (Release 2020b)
March 2021	Online only	Revised for Version 3.2 (Release 2021a)
September 2021	Online only	Revised for Version 3.3 (Release 2021b)
March 2022	Online only	Revised for Version 3.4 (Release 2022a)
September 2022	Online only	Revised for Version 3.5 (Release 2022b)

Contents

1 Apps

2 Blocks

3 Functions

4 Classes

Apps

WLAN Waveform Generator

Create, impair, visualize, and export WLAN waveforms

Description

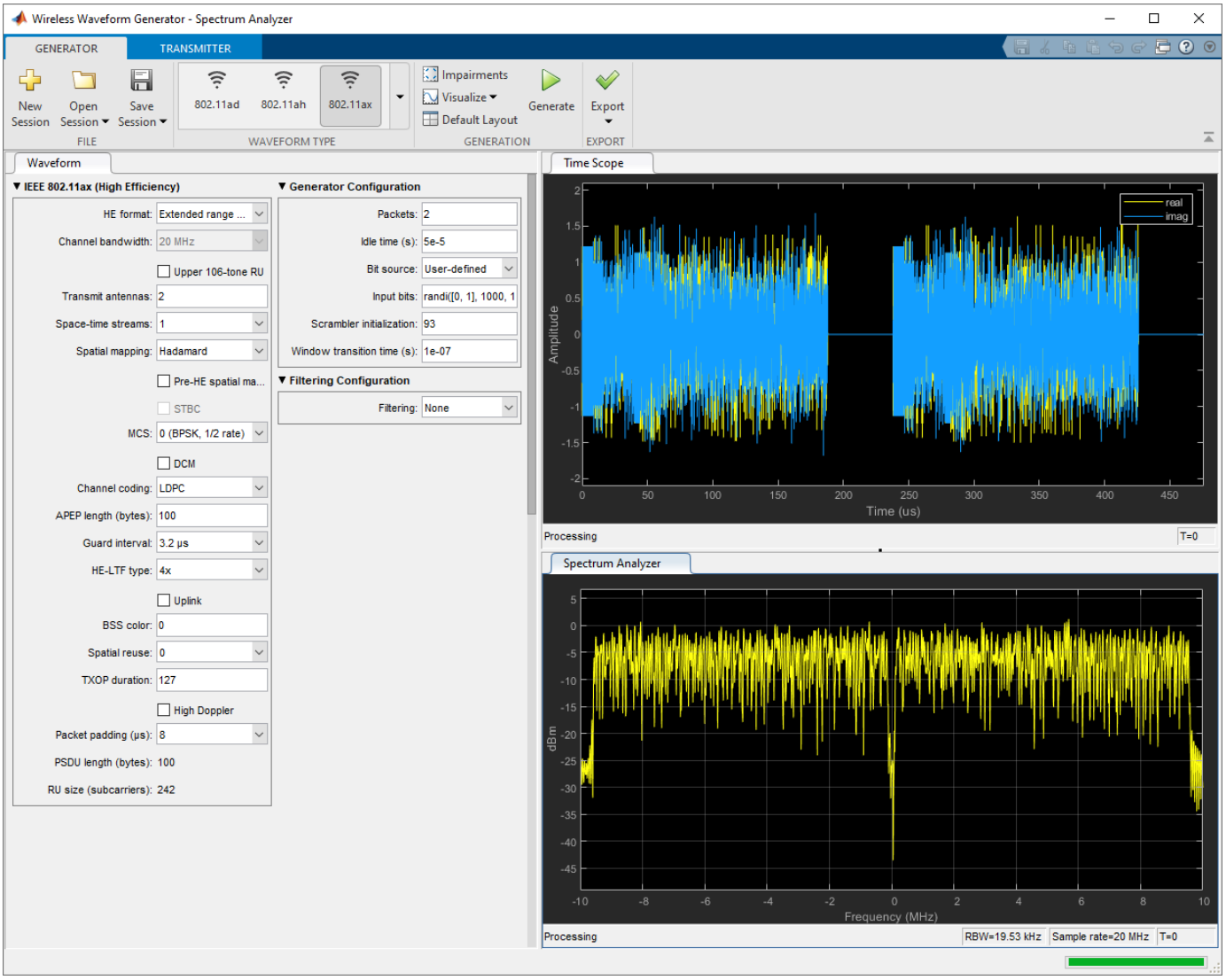
The **WLAN Waveform Generator** app enables you to create, impair, visualize, and export IEEE® 802.11™ waveforms.

The app provides these capabilities by using the **Wireless Waveform Generator** app configured for WLAN waveform generation. Using the app, you can:

- Generate IEEE 802.11ax™ waveforms, as specified in [1].
- Generate IEEE 802.11ac™, 802.11ad™, 802.11n™, 802.11ah™, 802.11p™, 802.11a™, 802.11g™, 802.11j™, and 802.11b™ waveforms, as specified in [2].
- Export the WLAN waveform to your workspace or to a `.mat` or a `.bb` file.
- Export WLAN waveform generation parameters to a runnable MATLAB® script or a Simulink® block.
  - Use the exported script to generate your waveform without the app from the command line.
  - Use the exported block as a waveform source in a Simulink model. For more information, see Waveform From Wireless Waveform Generator App.
- Visualize the WLAN waveform in time scope, spectrum analyzer, constellation diagram, and complementary cumulative distribution function (CCDF) plots.
- Visualize the resource unit (RU) and subcarrier assignment in an IEEE 802.11ax waveform.
- Distort the WLAN waveform by adding RF impairments, such as AWGN, phase offset, frequency offset, DC offset, IQ imbalance, and memoryless cubic nonlinearity.
- Generate a WLAN waveform that you can transmit using a connected radio or lab test instrument.
  - To transmit a waveform by using an SDR, connect one of the supported SDRs (ADALM-Pluto, USRP™, USRP embedded series, and Xilinx® Zynq-based radios) to your computer and have the associated add-on installed. For more information, see “Transmit Using SDR”.
  - To transmit a waveform by using lab test instrument, connect one of the instruments supported by the `rfsiggen` function to your computer. For more information, see “Quick-Control RF Signal Generator Requirements” (Instrument Control Toolbox). This feature requires “Instrument Control Toolbox”.
  - To transmit your waveforms over the air at full radio device rates, use the Wireless Testbench™ software and connect a supported radio to your computer. For a list of radios that support full device rates, see “Supported Radio Devices” (Wireless Testbench). This feature requires “Wireless Testbench”. For an example, see “Transmit App-Generated Wireless Waveform Using Radio Transmitters” on page 1-5.

To create, impair, visualize, and export waveforms other than WLAN waveforms, you must reconfigure the app. For a full list of features, see the **Wireless Waveform Generator** app.

For more information, see “Create Waveforms Using Wireless Waveform Generator App”.



Open the WLAN Waveform Generator App

MATLAB Toolstrip: On the **Apps** tab, under **Signal Processing and Communications**, click the app icon.

MATLAB Command Prompt: Enter `wirelessWaveformGenerator`. This command opens the **Wireless Waveform Generator** app. To configure the app for WLAN waveform generation, in the **Waveform Type** section, select one of the formats under **WLAN (IEEE 802.11)**.

Examples

App-Based WLAN Waveform Generation

This example shows how to generate IEEE® 802.11™ waveforms by using the **WLAN Waveform Generator** app.

Open WLAN Waveform Generator App

On the **Apps** tab of the MATLAB® toolstrip, select the **WLAN Waveform Generator** app icon under **Signal Processing and Communications**. This selection opens the **Wireless Waveform Generator** app configured for WLAN waveform generation.

Select IEEE 802.11 PHY Format

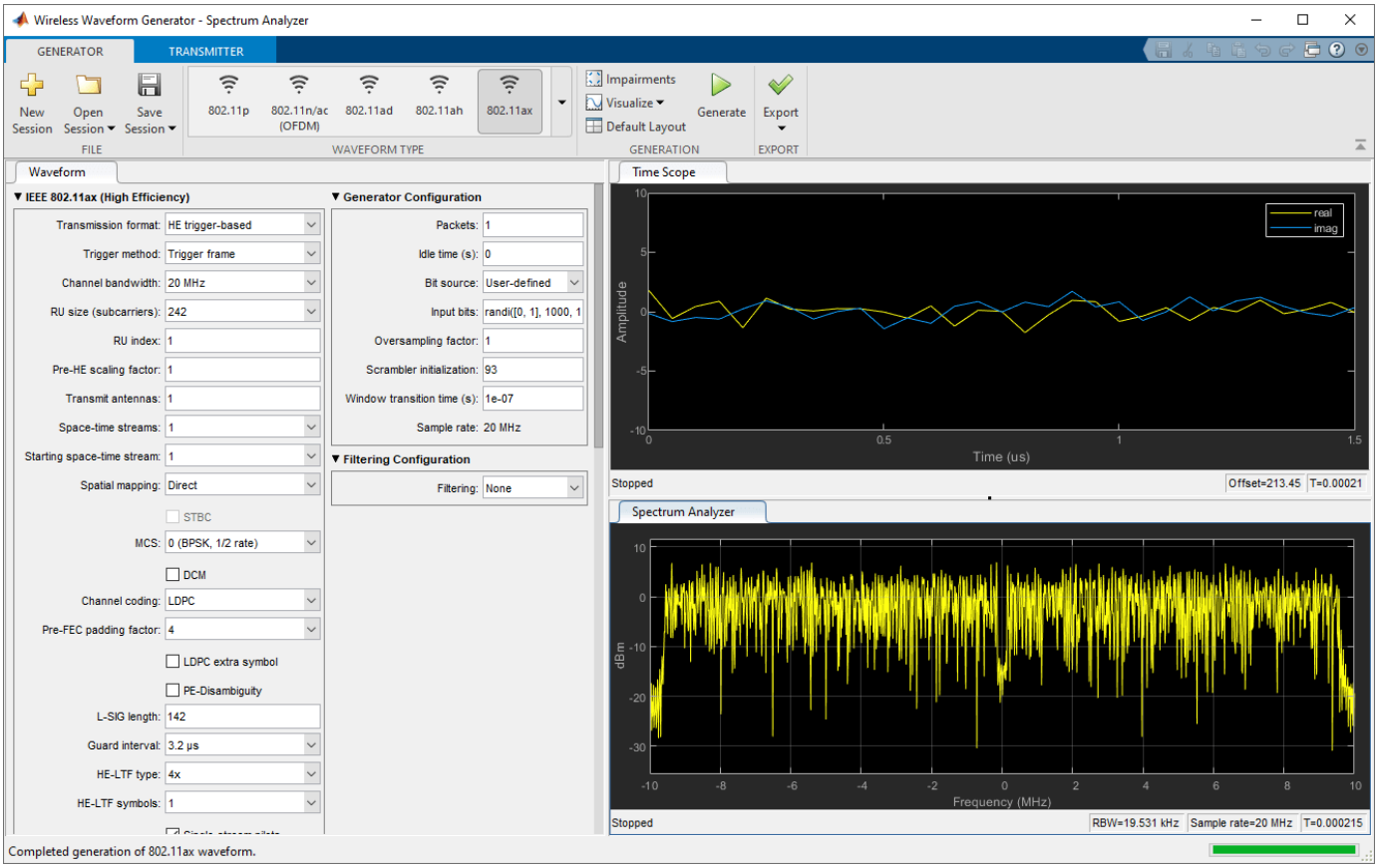
Choose the PHY format of the waveform you want to generate by selecting one of the formats under **WLAN (IEEE 802.11)** in the **Waveform Type** section of the app toolstrip. The app supports these IEEE 802.11 PHY formats.

- `802.11ax`
- `802.11ah`
- `802.11ad`
- `802.11n/ac`
- `802.11p`
- `802.11b/g`
- `802.11a/g/j`

Generate WLAN Waveform

Set transmission and configuration parameters by specifying options in the **Waveform** tab on the left pane of the app. Add impairments and select visualization tools by specifying options in the **Generation** section of the app toolstrip. To visualize the waveform, click **Generate**.

For example, this figure shows the **Time Scope** and **Spectrum Analyzer** visualization results for a high-efficiency trigger-based (HE TB) waveform with default parameters.



Export Generated Waveform

You can export the generated waveform and its parameters by clicking **Export**. You can export the waveform to:

- A MATLAB script with a `.m` extension, which you can run to generate the waveform without the app
- A file with a `.bb` or `.mat` extension
- Your MATLAB workspace as a structure
- A Simulink® block, which you can use to generate the waveform in a Simulink model without the app

Transmit WLAN Waveform

This feature requires “Instrument Control Toolbox”™ software. To transmit a generated waveform, click the **Transmitter** tab on the app toolstrip and configure the instruments. You can use any instrument supported by the `rfsiggen` (Instrument Control Toolbox) function.

Transmit App-Generated Wireless Waveform Using Radio Transmitters

This example shows how to use the NI™ USRP™ N310, USRP N320, USRP N321, and USRP X310 radio transmitters available in the **Wireless Waveform Generator** app to transmit an app-generated

waveform over the air (requires Wireless Testbench™). These radio transmitters enable you to transmit up to 2 GB of contiguous data over the air at full radio device rate.

Introduction

The Wireless Waveform Generator app is an interactive tool for creating, impairing, visualizing, and transmitting waveforms. Using the USRP N310, USRP N320, USRP N321, and USRP X310 radio transmitters available in the app, you can transmit your generated waveform repeatedly over the air. You can also export the waveform generation and transmission parameters to a runnable MATLAB® script. This example shows how to configure these radio transmitters.

Although this example shows how to transmit an OFDM waveform, the same process applies for all waveform types that you can generate with the app.

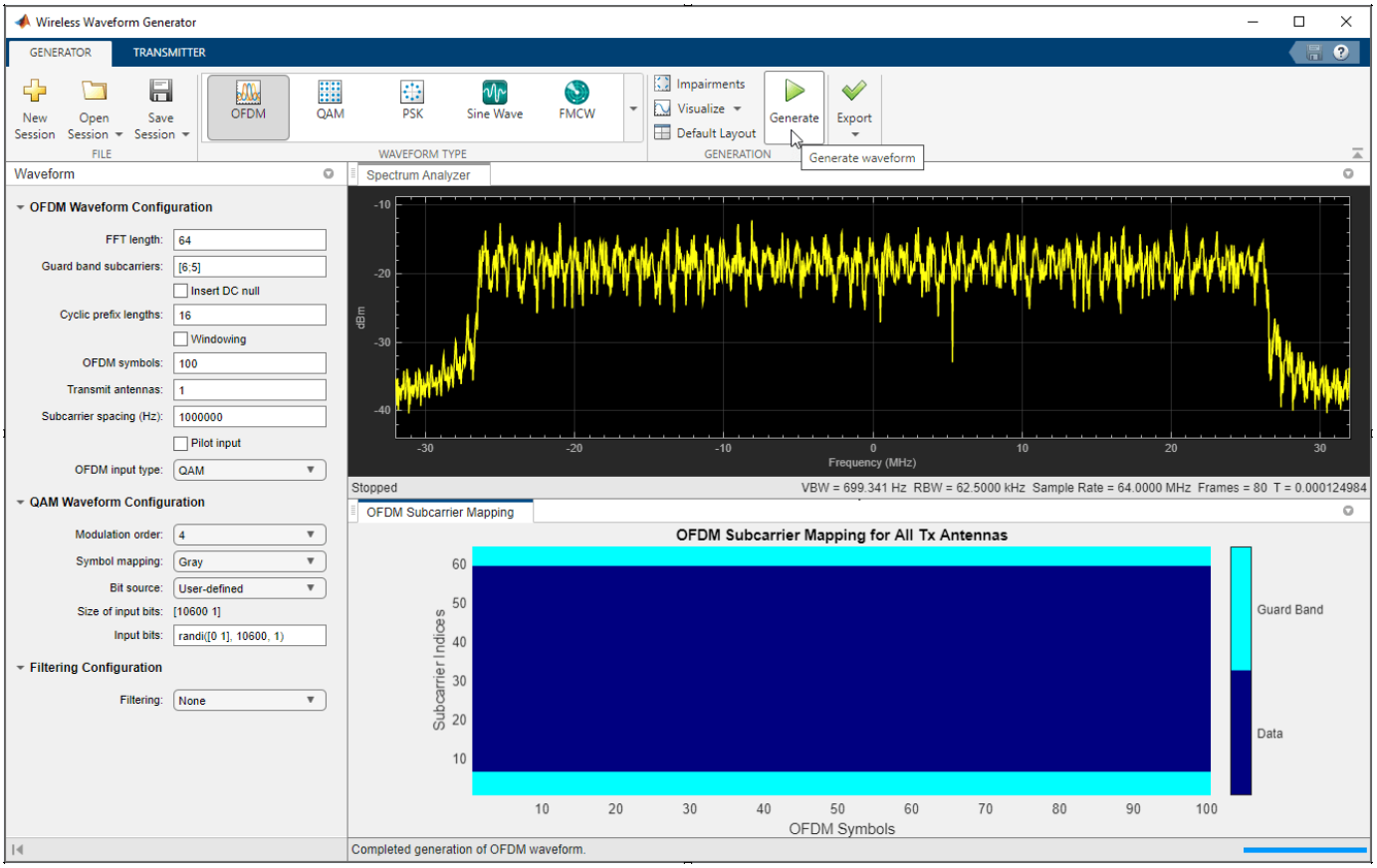
Set Up for Radio Transmission

To use the radio transmitters in the app, you need to install the Wireless Testbench Support Package for NI USRP Radios add-on and set up your radio outside the app. For more information, see “Connect and Set Up NI USRP Radios” (Wireless Testbench).

Generate Waveform for Transmission

Open the **Wireless Waveform Generator** app by clicking the app icon on the **Apps** tab, under **Signal Processing and Communications**. Alternatively, enter `wirelessWaveformGenerator` at the MATLAB command prompt.

In the **Waveform Type** section, select an OFDM waveform by clicking **OFDM**. In the leftmost pane of the app, adjust any configuration parameters for the selected waveform. Then generate the configuration by clicking **Generate** in the app toolstrip.



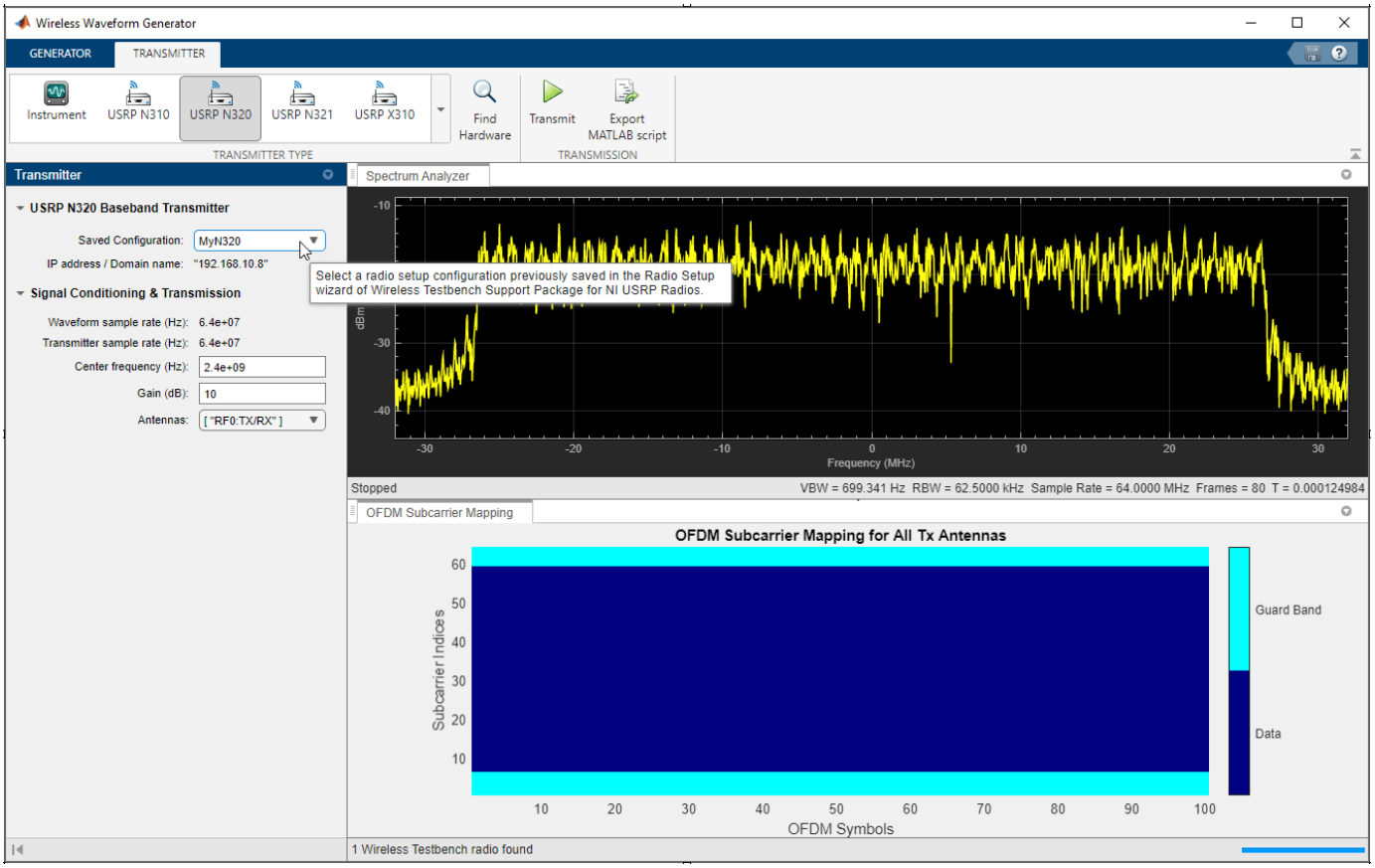
Configure Radio Transmitter

Select the **Transmitter** tab from the app toolstrip. In the transmitter gallery, select the USRP N310, USRP N320, USRP N321, or USRP X310 radio transmitter.

In the leftmost pane of the app, select the name of a radio setup configuration that you saved using the Radio Setup wizard. For more information, see “Connect and Set Up NI USRP Radios” (Wireless Testbench).

Set the center frequency, gain, and antennas configuration parameters. The app automatically sets the waveform sample rate based on the waveform that you generated earlier. The radio transmitter uses onboard data buffering to ensure contiguous data transmission at up to the full hardware sample rate. If necessary, to achieve the specified sample rate, the radio uses a Farrow rate converter. Use this list as a reference when setting the sample rate:

- **USRP N310** — 120,945 Hz to 76.8 MHz, or one of: 122.88 MHz, 125 MHz, or 153.6 MHz
- **USRP N320** — 196,851 Hz to 125 MHz, or one of: 200 MHz, 245.76 MHz or 250 MHz
- **USRP N321** — 196,851 Hz to 125 MHz, or one of: 200 MHz, 245.76 MHz or 250 MHz
- **USRP X310** — 181,418 Hz to 100 MHz, or one of: 184.32 MHz or 200 MHz



Transmit Waveform

To transmit the waveform continuously, click **Transmit**. To end the continuous transmission, click **Stop transmission**. To export the waveform generation and transmission parameters to a runnable MATLAB script, click **Export MATLAB script**.

Version History

Introduced in R2018b

References

[1] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

[2] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

See Also

Apps
Wireless Waveform Generator

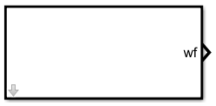
Functions
`wlanWaveformGenerator`

Topics
“Create Waveforms Using Wireless Waveform Generator App”

Blocks

Waveform From Wireless Waveform Generator App

Wireless waveform source exported to Simulink
Library: None



Description

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

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

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

To generate a block:

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

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

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

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

Limitations

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

Ports

Output

`wf` — Time-domain wireless waveform

complex matrix

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

Data Types: `double`

Parameters

Read-Only Waveform Parameters

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

`Waveform sample rate (Fs)` — Waveform sample rate

numeric scalar

This parameter is read-only.

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

`Waveform length` — Waveform length

positive integer

This parameter is read-only.

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

Simulation Parameters

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

`Samples per frame` — Samples per frame

1 (default) | positive integer

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

`Form output after final data value by` — Output values after last waveform sample

`Cyclic repetition` (default) | `Setting to zero`

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

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

Version History

Introduced in R2021b

Extended Capabilities

C/C++ Code Generation

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

See Also

Apps

WLAN Waveform Generator

Functions

addIE

Update MAC management frame with IE

Syntax

```
cfgUpdated = addIE(cfgMgmt,id,information)
```

Description

`cfgUpdated = addIE(cfgMgmt,id,information)` updates WLAN MAC management frame-body configuration `cfgMgmt` by appending an information element (IE). Specify the IE by its identifier `id` and information field `information`. The function returns `cfgUpdated`, an MAC management frame-body configuration whose `InformationElements` property contains the appended IE.

Examples

Add Information Element to WLAN MAC Management Frame-Body Configuration Object

Add the DSSS Parameter Set information element to a WLAN MAC management frame-body configuration object by using the `addIE` object function. The element ID for this information element is 3. The information is `'0b'`, representing the current channel (11) in hexadecimal format.

```
config = wlanMACManagementConfig('FrameType','Beacon');
id = 3;
information = '0b'
```

```
information =
'0b'
```

```
config = addIE(config,id,information);
```

Display the information elements of the frame-body configuration object by using the `displayIEs` object function.

```
displayIEs(config)
```

```
Element ID: 0, Information: 0x64656661756C742053534944
Element ID: 1, Information: 0x8C98B0
Element ID: 3, Information: 0x0B
```

Input Arguments

`cfgMgmt` — MAC management frame configuration

`wlanMACManagementConfig` object

MAC management frame configuration, specified as a `wlanMACManagementConfig` object.

`id` — IE ID field

nonnegative integer

IE ID field, specified as a nonnegative integer in the interval [0,255]. For more information, see Table 9-92—Element IDs in [1].

Data Types: `double`

`information` — Value of information field
character vector | string scalar

Value of information field in hexadecimal format, specified as a character vector or string scalar of hexadecimal octets. For more information, see Table 9-92—Element IDs in [1].

Data Types: `char` | `string`

Output Arguments

`cfgUpdated` — Updated MAC management frame-body configuration
`wlanMACManagementConfig` object

Updated MAC management frame-body configuration, returned as a `wlanMACManagementConfig` object. The `InformationElements` property of this output contains the appended IE.

Version History

Introduced in R2018b

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

Extended Capabilities

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

See Also

`wlanMACManagementConfig`

addUserInfo

Add User Info field to WLAN MAC trigger frame

Syntax

```
cfgUpdated = addUserInfo(cfgTrigger,cfgUser)
```

Description

`cfgUpdated = addUserInfo(cfgTrigger,cfgUser)` adds the User Info field specified by configuration object `cfgUser` to the MAC trigger frame parameterized by configuration object `cfgTrigger` by updating the `UserInfo` property of `cfgTrigger`. The updated MAC trigger frame configuration object, `cfgUpdated`, contains the updated `UserInfo` property and all other property values of `cfgTrigger`.

When you first create a `wlanMACTriggerConfig` object, the `UserInfo` property contains a single User Info field corresponding to a default `wlanMACTriggerUserConfig` object. The first User Info field you add by using this object function overwrites the default User Info field. The function appends subsequent User Info fields that you add to the `UserInfo` property.

For more information on the trigger frame format, see section 9.3.1.22 of [1]

Examples

Create Basic MAC Trigger Frame

Create a basic MAC trigger frame to carry information for two users.

Create a MAC trigger frame-body configuration object, specifying a channel bandwidth of 40 MHz.

```
cfgTrigger = wlanMACTriggerConfig(ChannelBandwidth="CBW40");
```

Create configuration objects for the User Info fields of the trigger frame.

```
cfgUser1 = wlanMACTriggerUserConfig(AID12=1, ...
    RUSize=242,RUIndex=1);
cfgUser2 = wlanMACTriggerUserConfig(AID12=2, ...
    RUSize=242,RUIndex=2);
```

Add the User Info fields to the trigger frame.

```
cfgTrigger = addUserInfo(cfgTrigger,cfgUser1);
cfgTrigger = addUserInfo(cfgTrigger,cfgUser2);
```

Configure the trigger frame by creating a MAC frame-body configuration object, specifying the frame type and the trigger frame-body configuration.

```
cfgMAC = wlanMACFrameConfig(FrameType="Trigger", ...
    TriggerConfig=cfgTrigger);
```

Specify a non-HT PHY configuration by creating a default non-HT configuration object.

```
cfgPHY = wlanNonHTConfig;
```

Create the MAC trigger frame and display its length.

```
[frame,frameLength] = wlanMACFrame(cfgMAC,cfgPHY);
disp(frameLength)
```

```
    40
```

Input Arguments

`cfgTrigger` — WLAN MAC trigger frame configuration

`wlanMACTriggerConfig` object

WLAN MAC trigger frame configuration, specified as a `wlanMACTriggerConfig` object.

`cfgUser` — User Info field configuration

`wlanMACTriggerUserConfig` object

User Info field configuration, specified as a `wlanMACTriggerUserConfig` object.

Output Arguments

`cfgUpdated` — Updated WLAN MAC trigger frame configuration

`wlanMACTriggerConfig` object

WLAN MAC trigger frame configuration, returned as a `wlanMACTriggerConfig` object. This output contains all property values of the `cfgTrigger` input but with the `UserInfo` property updated to contain the User Info field specified by the `cfgUser` input.

Version History

Introduced in R2021a

References

[1] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`wlanMACFrame`

Objects
`wlanMACFrameConfig` | `wlanMACTriggerConfig`

Topics
“Generate and Parse WLAN MAC Frames”

displayIEs

Display list of IEs in MAC management frame

Syntax

```
displayIEs(cfg)
```

Description

`displayIEs(cfg)` displays the list of information elements (IEs) in WLAN MAC management frame parameterized by frame-body configuration object `cfg`. Each row consists of the element ID, element extension ID (if present), and the information. The element ID and element extension ID are integers in the interval [0, 255]. The IEs are displayed in hexadecimal format with the prefix `0x`.

Examples

Display IEs of `wlanMACManagementConfig` Object

Create a WLAN MAC management frame-body configuration object with default settings. Display the information elements of the object using the `displayIEs` object function.

```
cfg = wlanMACManagementConfig;
displayIEs(cfg);
```

```
Element ID: 0, Information: 0x64656661756C742053534944
Element ID: 1, Information: 0x8C98B0
```

Input Arguments

`cfg` — WLAN MAC management frame-body configuration

`wlanMACManagementConfig` object

WLAN MAC management frame-body configuration, specified as a `wlanMACManagementConfig` object.

Version History

Introduced in R2019a

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects
`wlanMACManagementConfig`

Functions
`addIE`

generate

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

Syntax

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

Description

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

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

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

Examples

Generate VoIP Application Traffic Pattern Without Jitter

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

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

Generate a VoIP application traffic pattern.

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

Input Arguments

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

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

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

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

nonnegative scalar

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

Data Types: `double`

Output Arguments

`dt` — Time remaining to generate next packet

nonnegative scalar

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

Data Types: `double`

`packetSize` — Size of current packet

positive scalar

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

Data Types: `double`

`packet` — Application data packet

column vector of integers in the range [0, 255]

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

Dependencies

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

Data Types: `double`

Version History

Introduced in R2020b

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

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

getActiveSubchannelIndex

Active subchannel indices

Syntax

```
idx = getActiveSubchannelIndex(cfgWUR)
```

Description

`idx = getActiveSubchannelIndex(cfgWUR)` returns indices of the active 20 MHz subchannels in a WLAN wake-up radio (WUR) transmission parameterized by `cfgWUR`.

Examples

Get Active Subchannel Indices of WUR Transmission

Create a WUR configuration object for a transmission with two subchannels and four transmit antennas.

```
numSubchannels = 2;
cfgWUR = wlanWURConfig(numSubchannels,NumTransmitAntennas=4);
```

Get the active subchannel indices for the transmission.

```
idx = getActiveSubchannelIndex(cfgWUR)
```

```
idx = 1×2

     1     2
```

Input Arguments

`cfgWUR` — WUR transmission parameters

`wlanWURConfig` object

WUR transmission parameters, specified as a `wlanWURConfig` object.

Output Arguments

`idx` — Active subchannel indices

integer-valued row vector

Active subchannel indices, returned as an integer-valued row vector of length equal to the number of subchannels in the transmission.

Data Types: `double`

Version History

Introduced in R2021b

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`wlanWURConfig` | `wlanWURSubchannel`

getNDPFeedbackConfiguration

Valid HE TB feedback NDP PHY configuration

Syntax

```
cfgFeedback = getNDPFeedbackConfiguration(cfgHETB)
```

Description

`cfgFeedback = getNDPFeedbackConfiguration(cfgHETB)` generates a valid physical layer (PHY) configuration for a high-efficiency trigger-based (HE TB) feedback null data packet (NDP). The function sets properties of `cfgFeedback` by changing a subset of the properties of HE TB configuration `cfgHETB`. You can parameterize an HE TB feedback NDP by using the `cfgFeedback` output. For more information about the HE TB feedback NDP, see section 27.3.18 of [1].

Examples

Recover Feedback Status from HE TB Feedback NDP

Configure an uplink HE TB feedback NDP transmission with four stations (STAs), a channel bandwidth of 20 MHz, and a signal-to-noise ratio (SNR) of 20 dB.

```
numSTA = 4;
cbw = 'CBW20';
snr = 20;
cfgSTA = cell(1,numSTA);
```

Specify the resource unit (RU) tone set index, starting space-time stream, and feedback status for all STAs.

```
ruToneSetIndex = repmat([1 2],1,round(numSTA/2));
startingSTS = repmat([1 2],1,round(numSTA/2));
feedbackStatus = repmat([1 0],1,round(numSTA/2));
```

Create a valid HE TB feedback NDP configuration.

```
cfg = wlanHETBConfig;
cfg = getNDPFeedbackConfiguration(cfg);
```

Configure the channel for transmission, assuming no variation across STAs.

```
tgax = wlanTGaxChannel('ChannelBandwidth',cbw, ...
    'TransmissionDirection','Uplink', ...
    'SampleRate',wlanSampleRate(cfg));
chanInfo = info(tgax);
awgn = comm.AWGNChannel('NoiseMethod','Signal to noise ratio (SNR)', ...
    'SignalPower',1/tgax.NumReceiveAntennas);
```

Configure STAs and generate an HE TB feedback NDP waveform.

```
rx = 0;
for idx = 1:numSTA
```

```
    % Configure STAs

    cfg.RUToneSetIndex = ruToneSetIndex(idx);
    cfg.StartingSpaceTimeStream = startingSTS(idx);
    cfg.FeedbackStatus = feedbackStatus(idx);
    cfgSTA{idx} = cfg;

    % Generate transmit waveform

    waveform = wlanWaveformGenerator([],cfg);

    % Pass waveform through TGax channel

    rx = rx + tgax([waveform; zeros(15,size(waveform,2))]);
end
```

Pass the waveform through the AWGN channel, accounting for the noise energy in nulls to ensure the SNR is defined per active and complementary subcarrier.

```
field = 'HE-LTF';
ofdmInfo = wlanHEOFDMInfo(field,cbw,cfg.GuardInterval);
awgn.SNR = snr - 10*log10(ofdmInfo.FFTLength/12);
rx = awgn(rx);
```

Get the field indices and extract the HE-LTF.

```
ind = wlanFieldIndices(cfgSTA{1});
offset = chanInfo.ChannelFilterDelay;
heltf = rx(offset+(ind.HELTF(1):ind.HELTF(2)),:);
```

Demodulate the HE-LTF.

```
rxSym = wlanHEDemodulate(heltf,field,cbw,cfg.GuardInterval,cfg.HELTFType);
```

Recover the feedback status for the STAs.

```
status = zeros(1,numSTA);
for n = 1:numSTA
    status(n) = wlanHETBNDPFeedbackStatus(rxSym,cfgSTA{n});
end
```

Compare the transmitted and received feedback status for the STAs.

```
disp(isequal(feedbackStatus(1:numSTA),status))
```

```
   1
```

Input Arguments

`cfgHETB` — HE TB PHY configuration

`wlanHETBConfig` object

HE TB PHY configuration, specified as a `wlanHETBConfig` object.

Output Arguments

`cfgFeedback` — Valid HE TB feedback NDP PHY configuration
`wlanHETBConfig` object

Valid HE TB feedback NDP PHY configuration, returned as a `wlanHETBConfig` object. The function sets property values such that this object can parameterize a valid HE TB feedback NDP.

Version History

Introduced in R2021a

References

[1] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

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

See Also

Functions
`wlanHETBNDPFeedbackStatus`

Objects
`wlanHETBConfig`

Topics
“802.11ax Feedback Status Misdetection Simulation for Uplink Trigger-Based Feedback NDP”

getNumPostFECPaddingBits

Calculate required number of post-FEC padding bits

Syntax

```
n = getNumPostFECPaddingBits(cfg)
```

Description

`n = getNumPostFECPaddingBits(cfg)` calculates the required number of post-FEC padding bits `n` in a high-efficiency (HE) transmission parameterized by `cfg`.

Examples

Calculate Number of Post-FEC Padding Bits for HE Transmissions

This example shows how to calculate the required number of post-forward-error-correction (post-FEC) padding bits for high-efficiency (HE) WLAN transmissions.

Calculate Number of Post-FEC Padding Bits for HE SU Transmission

Parameterize an HE single-user (HE SU) transmission by creating a `wlanHESUConfig` object.

```
cfgHESU = wlanHESUConfig('ChannelBandwidth','CBW80','MCS',7);
```

Calculate the number of post-FEC padding bits required for waveform generation.

```
n = getNumPostFECPaddingBits(cfgHESU)
```

```
n = 3000
```

Calculate Number of Post-FEC Padding Bits for HE MU Transmission

Parameterize an HE multi-user (HE MU) transmission by creating a `wlanHEMUConfig` object.

```
cfgHEMU = wlanHEMUConfig(40);
```

Calculate the number of post-FEC padding bits required for waveform generation.

```
n = getNumPostFECPaddingBits(cfgHEMU)
```

```
n = 1×5

    18    18    36    18    78
```

Calculate Number of Post-FEC Padding Bits for HE TB Transmission

Parameterize an HE trigger-based (HE TB) transmission by creating a `wlanHETBConfig` object.

```
cfgHETB = wlanHETBConfig('ChannelBandwidth','CBW40','MCS',6, ...
    'NumTransmitAntennas',4,'PreFECPaddingFactor',3);
```

Calculate the number of post-FEC padding bits required for waveform generation.

```
n = getNumPostFECPaddingBits(cfgHETB)
```

```
n = 324
```

Input Arguments

`cfg` — HE transmission parameters

`wlanHEMUConfig` object | `wlanHESUConfig` object | `wlanHETBConfig` object

HE transmission parameters specified as one of these objects.

- `wlanHEMUConfig` — HE MU transmission
- `wlanHESUConfig` — HE SU transmission
- `wlanHETBConfig` — HE TB transmission

Output Arguments

`n` — Required number of post-FEC padding bits

binary-valued scalar | binary-valued column vector

Required number of post-FEC padding bits, returned as one of these values.

- A binary-valued scalar for single-user transmissions
- A binary-valued column vector of length N_{users} for multi-user transmissions, where N_{users} is the number of users in the transmission. In this case, the kth entry is the required number of post-FEC padding bits for the kth user in the transmission.

Data Types: `double` | `int8`

Version History

Introduced in R2020b

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`wlanHEMUConfig` | `wlanHEMUUser` | `wlanHESUConfig` | `wlanHETBConfig`

getPSDULength

Calculate HE or WUR PSDU length

Syntax

```
psduLength = getPSDULength(cfg)
```

Description

`psduLength = getPSDULength(cfg)` calculates the PSDU length for high-efficiency (HE) or wake-up radio (WUR) transmission parameters `cfg`.

Examples

Get PSDU Length for HE Configuration Objects

Get the PSDU length of HE single-user (HE SU) and HE multi-user (HE MU) configuration objects.

Get HE SU PSDU Length

Create an HE SU configuration object. Get and display the PSDU length for the configured transmission.

```
cfgHESU = wlanHESUConfig;
psduLength = getPSDULength(cfgHESU)
```

```
psduLength = 100
```

Get HE MU PSDU Lengths

Create an HE MU configuration object with the allocation index set to `5`, which configures a transmission with seven users. Get and display the PSDU lengths for the configured transmission.

```
cfgHEMU = wlanHEMUConfig(5);
psduLength = getPSDULength(cfgHEMU)
```

```
psduLength = 1×7

   100   100   202   100   100   100   202
```

Get PSDU Length for WUR Configuration

Create a wake-up radio (WUR) configuration object for a transmission with four subchannels and four transmit antennas.

```
cfgWUR = wlanWURConfig(4,NumTransmitAntennas=4);
```

Calculate the PSDU lengths for the transmission and display the result.

```
psduLength = getPSDULength(cfgWUR)

psduLength = 1×4

     8     8     8     8
```

Input Arguments

`cfg` — HE or WUR transmission parameters

`wlanHESUConfig` object | `wlanHEMUConfig` object | `wlanHETBConfig` object | `wlanHERecoveryConfig` object | `wlanWURConfig` object

HE or WUR transmission parameters, specified as one of these objects: `wlanHEMUConfig`, `wlanHESUConfig`, `wlanHETBConfig`, `wlanHERecoveryConfig`, or `wlanWURConfig`.

Output Arguments

`psduLength` — PSDU length

positive integer | vector

PSDU length, in bytes, returned as a positive integer or row vector of size 1-by-*N*, where *N* is the number of users in the transmission. For an HE transmission, *N* is an integer in the interval [1, 74]. For WUR transmissions, *N* is 1, 2, 3, or 4.

For more information about *N* in HE transmissions, see the `ruInfo` function. The `NumUsers` field of the `info` output corresponds to *N*.

Data Types: `double`

Version History

Introduced in R2018b

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`wlanHEMUConfig` | `wlanHESUConfig` | `wlanHETBConfig` | `wlanHERecoveryConfig` | `wlanWURConfig`

Functions

`packetFormat` | `ruInfo` | `showAllocation` | `wlanWaveformGenerator`

getSIGBLength

Return information relevant to HE-SIG-B field length

Syntax

```
info = getSIGBLength(cfg)
```

Description

`info = getSIGBLength(cfg)` returns a structure, `info`, which contains information relevant to the HE-SIG-B field length of the high-efficiency (HE) recovery configuration object `cfg`. The input `cfg` contains the parameters recovered from decoding the signaling fields of an HE-format waveform.

Examples

Return HE-SIG-B Field Length Information

Create a WLAN HE-MU-format configuration object, specifying the allocation index.

```
cfgHEMU = wlanHEMUConfig(0);
```

Generate a WLAN waveform for the specified configuration and return the PPDU field indices.

```
waveform = wlanWaveformGenerator(1,cfgHEMU);
ind = wlanFieldIndices(cfgHEMU);
```

Decode the L-SIG field and obtain the OFDM information. This information is required to obtain the L-SIG length, which is used in the recovery configuration object.

```
lsig = waveform(ind.LSIG(1):ind.LSIG(2),:);
lsigDemod = wlanHEDemodulate(lsig,'L-SIG',cfgHEMU.ChannelBandwidth);
preHEInfo = wlanHEOFDMInfo('L-SIG',cfgHEMU.ChannelBandwidth);
```

Recover the L-SIG information bits and related information, making sure that the bits pass the parity check. For this example, we assume a noiseless channel. For more realistic results you can pass the waveform through an 802.11ax™ channel model by using the `wlanTGaxChannel` System object™ and work with the received waveform.

```
csi = ones(52,1);
[lsigBits,failCheck,lsigInfo] = wlanLSIGBitRecover(lsigDemod(preHEInfo.DataIndices,:,:),0,csi);
```

Decode the HE-SIG-A field and recover the HE-SIG-A information bits, ensuring that the bits pass the cyclic redundancy check (CRC).

```
siga = waveform(ind.HESIGA(1):ind.HESIGA(2),:);
sigaDemod = wlanHEDemodulate(siga,'HE-SIG-A',cfgHEMU.ChannelBandwidth);
preHEInfo = wlanHEOFDMInfo('HE-SIG-A',cfgHEMU.ChannelBandwidth);
[bits,failCRC] = wlanHESIGABitRecover(sigaDemod(preHEInfo.DataIndices,:,:),0,csi);
```

Create a WLAN recovery configuration object, specifying an HE-MU-format packet and the length of the L-SIG field.

```
cfg = wlanHERecoveryConfig('PacketFormat','HE-MU','LSIGLength',lsigInfo.Length);
```

Update the recovery configuration object with the recovered HE-SIG-A bits.

```
cfgUpdated = interpretHESIGABits(cfg,bits);
```

Return and display the HE-SIG-B information.

```
info = getSIGBLength(cfgUpdated);
disp(info);
```

```
    NumSIGBCommonFieldSamples: 80
                NumSIGBSymbols: 10
```

Input Arguments

`cfg` — HE recovery configuration object

`wlanHERecoveryConfig` object

HE recovery configuration object, specified as a `wlanHERecoveryConfig` object.

Output Arguments

`info` — Information relevant to HE-SIG-B field length

structure

Information relevant to the HE-SIG-B field length, returned as a structure containing these fields.

`NumHESIGBCommonFieldSamples` — Number of samples in HE-SIG-B common field

nonnegative integer

Number of samples in HE-SIG-B common field, returned as a nonnegative integer.

Data Types: `double`

`NumHESIGBSymbols` — Total number of symbols in HE-SIG-B field

nonnegative integer

Total number of symbols in HE-SIG-B field, returned as a nonnegative integer.

Data Types: `double`

Data Types: `struct`

Version History

Introduced in R2019a

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanHERecoveryConfig`

getTRSConfiguration

Valid HE TB PHY configuration in response to triggering frame containing TRS Control subfield

Syntax

```
cfgTRS = getTRSConfiguration(cfgHETB)
```

Description

`cfgTRS = getTRSConfiguration(cfgHETB)` generates `cfgTRS`, a valid PHY configuration for the IEEE 802.11 high-efficiency trigger-based (HE TB) packet format. The function sets properties of `cfgTRS` by changing a subset of the properties of input PHY configuration `cfgHETB`. You can parameterize an HE TB uplink PPDU transmitted in response to a frame containing a triggered response scheduling (TRS) Control subfield by using the `cfgTRS` output. For a detailed description of the HE WLAN formats, see [1].

Examples

Generate HE TB Waveform in Response to Frame Containing TRS Control Subfield

Configure and generate a WLAN HE TB waveform to be transmitted in response to a frame containing a TRS Control subfield.

Create an HE TB configuration object, specifying the triggering frame type.

```
cfgHETB = wlanHETBConfig('TriggerMethod','TRS');
```

Generate a valid configuration by using the `getTRSConfiguration` object function, displaying the result.

```
cfgTRS = getTRSConfiguration(cfgHETB)
```

```
cfgTRS =
  wlanHETBConfig with properties:

               FeedbackNDP: 0
             TriggerMethod: 'TRS'
          ChannelBandwidth: 'CBW20'
                    RUSize: 242
                   RUIndex: 1
    PreHEPowerScalingFactor: 1
        NumTransmitAntennas: 1
        NumSpaceTimeStreams: 1
    StartingSpaceTimeStream: 1
             SpatialMapping: 'Direct'
                       STBC: 0
                        MCS: 0
                        DCM: 0
             ChannelCoding: 'BCC'
         PreFECPaddingFactor: 4
              NumDataSymbols: 10
```

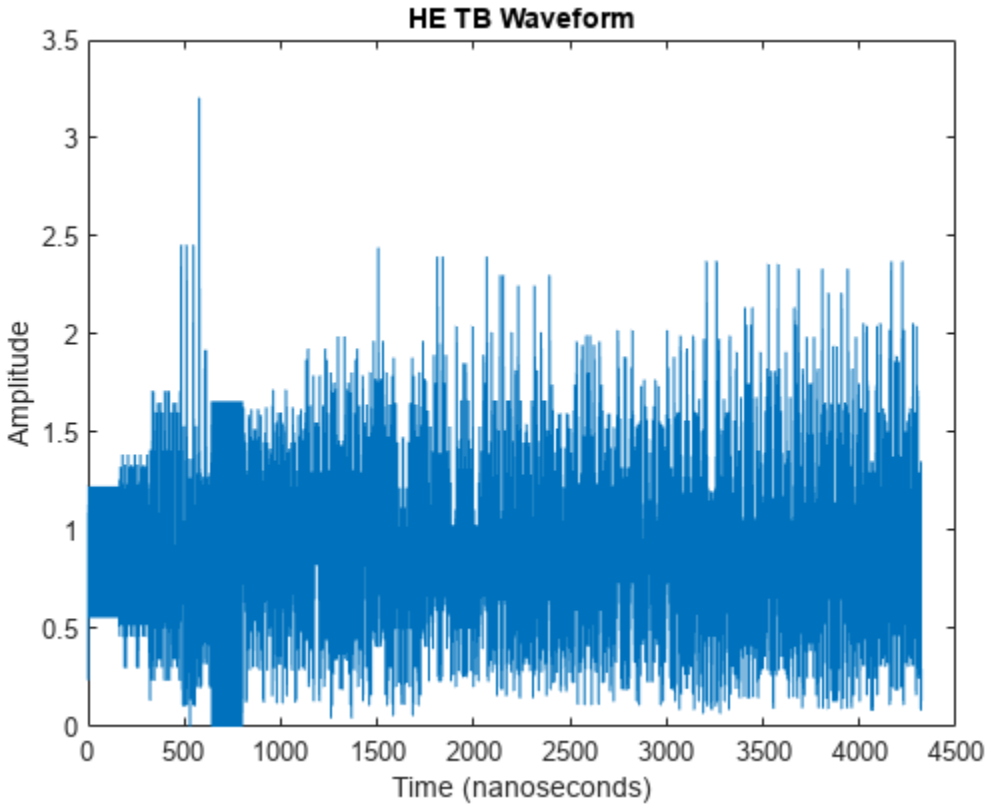
```
      DefaultPEDuration: 0
          GuardInterval: 3.2000
              HELTFType: 4
         NumHELTFSymbols: 1
       SingleStreamPilots: 1
                 BSSColor: 0
          SpatialReuse1: 15
          SpatialReuse2: 15
          SpatialReuse3: 15
          SpatialReuse4: 15
           TXOPDuration: 127
            HighDoppler: 0
      HESIGAReservedBits: [9x1 double]
    PostFECPaddingSource: 'mt19937ar with seed'
      PostFECPaddingSeed: 73
```

Get the PSDU length in bytes and generate a PSDU for transmission.

```
psduLength = getPSDULength(cfgTRS);
psdu = randi([0 1],8*psduLength,1);
```

Generate and plot the waveform.

```
waveform = wlanWaveformGenerator(psdu,cfgTRS);
figure;
plot(abs(waveform));
title('HE TB Waveform');
xlabel('Time (nanoseconds)');
ylabel('Amplitude');
```



Input Arguments

`cfgHETB` — HE TB PHY configuration

`wlanHETBConfig` object

HE TB PHY configuration, specified as a `wlanHETBConfig` object.

Output Arguments

`cfgTRS` — Valid HE TB PHY configuration

`wlanHETBConfig` object

Valid HE TB PHY configuration, returned as a `wlanHETBConfig` object. The `getTRSConfiguration` function sets property values such that this object can parameterize an HE TB uplink PPDU in response to a frame containing a TRS Control subfield.

More About

PPDU

The physical layer (PHY) protocol data unit (PPDU) is the complete physical layer convergence procedure (PLCP) frame, including PLCP headers, MAC headers, the MAC data field, and the MAC and PLCP trailers.

Version History

Introduced in R2020a

References

[1] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`wlanHETBConfig`

info

Characteristic information about multipath fading channels

Syntax

```
s = info(chan)
```

Description

`s = info(chan)` returns a structure containing characteristic information about the specified multipath fading channel System object™.

Examples

Return Characteristic Information of WLAN TGax Channel

Create a WLAN TGax multipath fading channel System object with default property values.

```
chan = wlanTGaxChannel;
```

Return and display the characteristic information of the TGax channel.

```
s = info(chan)
```

```
s = struct with fields:
           ChannelFilterDelay: 7
    ChannelFilterCoefficients: [17x22 double]
                   PathDelays: [0 5.0000e-09 1.0000e-08 1.5000e-08 ... ]
             AveragePathGains: [0 -2.7143 -5.4287 -8.1431 -2.5162 ... ]
                     Pathloss: 0
```

Return Characteristic Information of WLAN TGay Channel

Create a WLAN TGay multipath fading channel System object with default property values.

```
chan = wlanTGayChannel;
```

Return and display the characteristic information of the TGay channel.

```
s = info(chan)
```

```
s = struct with fields:
     ChannelFilterDelay: 7
    NumSamplesProcessed: 0
           NumTxStreams: 1
           NumRxStreams: 1
          NumTxElements: 4
```

```
NumRxElements: 4
```

Input Arguments

chan — Multipath fading channel

`wlanTGnChannel` object | `wlanTGacChannel` object | `wlanTGahChannel` object | `wlanTGaxChannel` object | `wlanTGayChannel` object

Multipath fading channel, specified as a `wlanTGnChannel`, `wlanTGacChannel`, `wlanTGahChannel`, `wlanTGaxChannel`, or `wlanTGayChannel` object.

Output Arguments

s — Characteristic information

structure

Characteristic information, returned as a structure whose fields depend on the multipath fading channel you specify in the `chan` input.

This field applies to all channel objects.

Field	Values	Description	Data Types
`ChannelFilterDelay`	Positive integer	Channel filter delay in samples	`double`
`PathDelays`	Nonnegative-valued row vector	Multipath delay, in seconds, of the discrete paths	

These fields do not apply to the `wlanTGayChannel` object.

Field	Values	Description	Data Types
`ChannelFilterCoefficients`	Real-valued matrix	Channel fractional delay filter coefficients	`double`
`AveragePathGains`	Nonpositive-valued row vector	Average gains, in dB, of the discrete paths	
`PathLoss`	Nonnegative scalar	Path loss, in dB, between the transmitter and receiver	

These fields apply only to the `wlanTGayChannel` object.

Field	Values	Description	Data Types
`NumSamplesProcessed`	Nonnegative integer	Number of samples the channel processed since its last reset	`double`
`NumTxStreams`	Positive integer	Number of transmitted data streams	

Field	Values	Description	Data Types
`NumRxStreams`		Number of received data streams	
`NumTxElements`		Number of elements in each transmit antenna array	
`NumRxElements`		Number of elements in each receive antenna array	
`AnglesOfDeparture`	Real-valued matrix	Azimuth and elevation angles of departure in degrees. Each element of this matrix contains the angles of departure for the corresponding for each pair of transmit and receive antennas. To enable this field, lock the TGay channel.	
`AnglesOfArrival`		Azimuth and elevation angles of arrival in degrees. Each element of this matrix contains the angles of arrival for the corresponding pair of transmit and receive antenna elements. To enable this field, lock the TGay channel.	
`DopplerShifts`	Real-valued row vector	Doppler shifts in Hz. Each element of this vector contains the Doppler shift for the corresponding pair of transmit and receive antenna elements. To enable this field, lock the TGay channel.	

Data Types: `struct`

Version History

Introduced in R2015b

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanTGnChannel` | `wlanTGacChannel` | `wlanTGahChannel` | `wlanTGaxChannel` | `wlanTGayChannel`

interpretHESIGABits

Update transmission parameters with HE-SIG-A field bits

Syntax

```
cfgUpdated = interpretHESIGABits(cfg,bits)
[cfgUpdated,failInterpretation] = interpretHESIGABits(cfg,bits)
```

Description

`cfgUpdated = interpretHESIGABits(cfg,bits)` updates high-efficiency (HE) transmission parameters `cfg` by interpreting recovered HE-SIG-A field bits. The function populates the properties of `cfg` that are relevant to the HE-SIG-A field and returns updated HE transmission parameters `cfgUpdated`. If you use this syntax and the function cannot interpret the recovered HE-SIG-A field bits, the function does not return an output and issues an error message.

`[cfgUpdated,failInterpretation] = interpretHESIGABits(cfg,bits)` returns the result of HE-SIG-A field interpretation. If you use this syntax and the function cannot interpret the recovered HE-SIG-A field bits, the function returns the `failInterpretation` output as `1` and `cfgUpdated` as the `cfg` input without updating any property values.

Examples

Recover HE-Data Field from HE SU Transmission

Recover bits from the HE-Data field of an HE SU transmission.

Configure an HE SU transmission by creating a configuration object with the specified modulation and coding scheme (MCS). Extract the channel bandwidth.

```
cfgHESU = wlanHESUConfig('MCS',0);
cbw = cfgHESU.ChannelBandwidth;       % Channel bandwidth of transmission
```

Create a sequence of data bits and generate an HE SU waveform.

```
bits = randi([0 1],8*getPSDULength(cfgHESU),1,'int8');
waveform = wlanWaveformGenerator(bits,cfgHESU);
```

Create a WLAN recovery configuration object, specifying the known channel bandwidth and packet format.

```
cfgRX = wlanHERecoveryConfig('ChannelBandwidth',cbw,'PacketFormat','HE-SU');
```

Recover the HE signaling fields by retrieving the field indices and performing the relevant demodulation operations.

```
ind = wlanFieldIndices(cfgRX);
heLSIGandRLSIG = waveform(ind.LSIG(1):ind.RLSIG(2),:);
symLSIG = wlanHEDemodulate(heLSIGandRLSIG,'L-SIG',cbw);
info = wlanHEOFDMInfo('L-SIG',cbw);
```

Merge the L-SIG and RL-SIG fields for diversity and obtain the data subcarriers.

```
symLSIG = mean(symLSIG,2);
lsig = symLSIG(info.DataIndices,:);
```

Decode the L-SIG field, assuming a noiseless channel, and use the length field to update the recovery object.

```
noiseVarEst = 0;
[~,~,lsigInfo] = wlanLSIGBitRecover(lsig,noiseVarEst);
cfgRX.LSIGLength = lsigInfo.Length;
```

Recover and demodulate the HE-SIG-A field, obtain the data subcarriers, and recover the HE-SIG-A bits.

```
heSIGA = waveform(ind.HESIGA(1):ind.HESIGA(2),:);
symSIGA = wlanHEDemodulate(heSIGA,'HE-SIG-A',cbw);
siga = symSIGA(info.DataIndices,:);
[sigaBits,failCRC] = wlanHESIGABitRecover(siga,0);
```

Update the recovery configuration object with the recovered HE-SIG-A bits and obtain the updated field indices.

```
cfgHE = interpretHESIGABits(cfgRX,sigaBits);
ind = wlanFieldIndices(cfgHE);
```

Retrieve and decode the HE-Data field.

```
heData = waveform(ind.HEData(1):ind.HEData(2),:);
symData = wlanHEDemodulate(heData,'HE-Data', ...
    cbw,cfgHE.GuardInterval,[cfgHE.RUSize cfgHE.RUIndex]);
infoData = wlanHEOFDMInfo('HE-Data',cbw,cfgHE.GuardInterval,[cfgHE.RUSize cfgHE.RUIndex]);
rxDataSym = symData(infoData.DataIndices,:,:);
dataBits = wlanHEDataBitRecover(rxDataSym,noiseVarEst,cfgHE);
```

Confirm that the recovered bits match the transmitted bits.

```
isequal(bits,dataBits)
```

```
ans = logical
   1
```

Update HE MU Recovery Configuration Object

Update a WLAN HE recovery configuration object by interpreting recovered HE-SIG-A and HE-SIG-B information bits.

Generate HE MU Waveform

Create a WLAN HE MU configuration object, setting the allocation index to `0`.

```
cfgHEMU = wlanHEMUConfig(0);
```

Generate a WLAN waveform and PPDU field indices for the specified configuration.

```
waveform = wlanWaveformGenerator(1,cfgHEMU);
ind = wlanFieldIndices(cfgHEMU);
```

Recover L-SIG Bits

Create a WLAN recovery configuration object, specifying an HE MU packet format and the length of the L-SIG field.

```
cfg = wlanHERecoveryConfig('PacketFormat','HE-MU','ChannelBandwidth','CBW20');
```

Decode the L-SIG field and obtain the orthogonal frequency-division multiplexing (OFDM) information. The recovery configuration object requires this information to obtain the L-SIG length.

```
lsig = waveform(ind.LSIG(1):ind.LSIG(2));
lsigDemod = wlanHEDemodulate(lsig,'L-SIG',cfg.ChannelBandwidth);
info = wlanHEOFDMInfo('L-SIG',cfg.ChannelBandwidth);
lsigDemod = lsigDemod(info.DataIndices,:);
```

Recover the L-SIG bits and related information, making sure that the bits pass the parity check, and update the recovery configuration object with the L-SIG length. For this example we assume a noiseless channel. For more realistic results you can pass the waveform through an 802.11ax™ channel model by using the `wlanTGaxChannel` System object™ and work with the received waveform.

```
csi = ones(52,1);
[lsigBits,failCheck,lsigInfo] = wlanLSIGBitRecover(lsigDemod,0,csi);
cfg.LSIGLength = lsigInfo.Length;
```

Update Recovery Configuration Object with HE-SIG-A Bits

Decode the HE-SIG-A field and recover the HE-SIG-A bits, ensuring that the bits pass the cyclic redundancy check (CRC).

```
siga = waveform(ind.HESIGA(1):ind.HESIGA(2));
sigaDemod = wlanHEDemodulate(siga,'HE-SIG-A',cfg.ChannelBandwidth);
sigaDemod = sigaDemod(info.DataIndices,:);
[sigaBits,failCRC] = wlanHESIGABitRecover(sigaDemod,0,csi);
disp(failCRC)
```

```
   0
```

Update the recovery configuration object with the recovered HE-SIG-A bits. Display the updated object. A property value of `-1` or `'Unknown'` indicates an unknown or undefined property, which can be updated after decoding the HE-SIG-B common and user fields of the HE MU packet.

```
[cfg,failInterpretation] = interpretHESIGABits(cfg,sigaBits)
```

```
cfg =
  wlanHERecoveryConfig with properties:

                    PacketFormat: 'HE-MU'
                ChannelBandwidth: 'CBW20'
                       LSIGLength: 878
                 SIGBCompression: 0
                         SIGBMCS: 0
                         SIGBDCM: 0
          NumSIGBSymbolsSignaled: 10
                            STBC: 0
```

```
                LDPCExtraSymbol: 1
             PreFECPaddingFactor: 1
                  PEDisambiguity: 0
                   GuardInterval: 3.2000
                       HELTFType: 4
                  NumHELTFSymbols: 1
                 UplinkIndication: 0
                          BSSColor: 0
                     SpatialReuse: 0
                     TXOPDuration: 127
                      HighDoppler: 0
                 AllocationIndex: -1
       NumUsersPerContentChannel: -1
         RUTotalSpaceTimeStreams: -1
                          RUSize: -1
                         RUIndex: -1
                           STAID: -1
                             MCS: -1
                             DCM: -1
                  ChannelCoding: 'Unknown'
                    Beamforming: -1
            NumSpaceTimeStreams: -1
    SpaceTimeStreamStartingIndex: -1


failInterpretation = logical
   0
```

Update Recovery Configuration Object with HE-SIG-B Common Field Bits

Decode the HE-SIG-B common field, ensuring that all content channels pass the CRC.

```
len = getSIGBLength(cfg);
sigbCommon = waveform(double(ind.HESIGA(2))+(1:len.NumSIGBCommonFieldSamples),:);
sigbCommonDemod = wlanHEDemodulate(sigbCommon,'HE-SIG-B',cfgHEMU.ChannelBandwidth);
sigbCommonDemod = sigbCommonDemod(info.DataIndices);
[sigbCommonBits,status,~] = wlanHESIGBCommonBitRecover(sigbCommonDemod,0,csi,cfg);
disp(status)
```

```
Success
```

Update the recovery configuration object with the recovered HE-SIG-B common field bits and display the updated object. A field returned as `-1` or `'Unknown'` indicates an unknown or undefined property value, which can be updated after decoding the HE-SIG-B user field of the HE MU packet.

```
[cfg,failInterpretation] = interpretHESIGBCommonBits(cfg,sigbCommonBits,status)
```

```
cfg =
  wlanHERecoveryConfig with properties:

                   PacketFormat: 'HE-MU'
                ChannelBandwidth: 'CBW20'
                      LSIGLength: 878
                SIGBCompression: 0
                         SIGBMCS: 0
                         SIGBDCM: 0
          NumSIGBSymbolsSignaled: 10
                            STBC: 0
```

```
                LDPCExtraSymbol: 1
             PreFECPaddingFactor: 1
                  PEDisambiguity: 0
                   GuardInterval: 3.2000
                       HELTFType: 4
                  NumHELTFSymbols: 1
                 UplinkIndication: 0
                          BSSColor: 0
                     SpatialReuse: 0
                     TXOPDuration: 127
                      HighDoppler: 0
                 AllocationIndex: 0
       NumUsersPerContentChannel: 9
         RUTotalSpaceTimeStreams: -1
                          RUSize: -1
                         RUIndex: -1
                           STAID: -1
                             MCS: -1
                             DCM: -1
                   ChannelCoding: 'Unknown'
                     Beamforming: -1
             NumSpaceTimeStreams: -1
    SpaceTimeStreamStartingIndex: -1


failInterpretation = logical
   0
```

Update Recovery Configuration Object with HE-SIG-B User Field Bits

Decode the HE-SIG-B user field, ensuring that all users pass the CRC.

```
sigbUser = waveform(ind.HESIGB(1):ind.HESIGB(2));
sigbUserDemod = wlanHEDemodulate(sigbUser,'HE-SIG-B',cfgHEMU.ChannelBandwidth);
sigbUserDemod = sigbUserDemod(info.DataIndices,:);
[sigbUserBits,failCRC,~] = wlanHESIGBUserBitRecover(sigbUserDemod,0,csi,cfg);
disp(failCRC)
```

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

Update the recovery configuration object with the recovered HE-SIG-B user field bits.

```
[user,failInterpretation] = interpretHESIGBUserBits(cfg,sigbUserBits,failCRC);
```

Display the results of interpretation and the third element of the `user` output.

```
disp(failInterpretation)
```

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

```
disp(user{3})
```

```
  wlanHERecoveryConfig with properties:

                  PacketFormat: 'HE-MU'
               ChannelBandwidth: 'CBW20'
                     LSIGLength: 878
                SIGBCompression: 0
```

```
                    SIGBMCS: 0
                   SIGBDCM: 0
      NumSIGBSymbolsSignaled: 10
                       STBC: 0
             LDPCExtraSymbol: 1
          PreFECPaddingFactor: 1
               PEDisambiguity: 0
                GuardInterval: 3.2000
                    HELTFType: 4
               NumHELTFSymbols: 1
              UplinkIndication: 0
                      BSSColor: 0
                 SpatialReuse: 0
                 TXOPDuration: 127
                  HighDoppler: 0
              AllocationIndex: 0
  NumUsersPerContentChannel: 9
    RUTotalSpaceTimeStreams: 1
                     RUSize: 26
                    RUIndex: 3
                      STAID: 0
                        MCS: 0
                        DCM: 0
                ChannelCoding: 'LDPC'
                  Beamforming: 0
          NumSpaceTimeStreams: 1
SpaceTimeStreamStartingIndex: 1
```

Input Arguments

`cfg` — HE transmission parameters before interpretation of HE-SIG-A field bits

`wlanHERecoveryConfig` object

HE transmission parameters before interpretation of HE-SIG-A field bits, specified as a `wlanHERecoveryConfig` object.

`bits` — Recovered HE-SIG-A field bits

binary-valued column vector

Recovered HE-SIG-A field bits, specified as a binary-valued column vector of length 52.

Data Types: `double` | `int8`

Output Arguments

`cfgUpdated` — Updated HE transmission parameters

`wlanHERecoveryConfig` object

Updated He transmission parameters, returned as a `wlanHERecoveryConfig` object. The function updates the properties of this object in accordance with the recovered HE-SIG-A field bits.

For information about the contents of the HE-SIG-A field, see [2].

`failInterpretation` — Result of HE-SIG-A field interpretation

`0` | `1`

Result of HE-SIG-A field interpretation, returned as a logical `0` or `1`. A value of `1` indicates that the function cannot interpret the recovered HE-SIG-A field bits. A value of `0` indicates that the function successfully interprets the HE-SIG-A field bits.

Data Types: `logical`

Version History

Introduced in R2019a

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

[2] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`wlanHERecoveryConfig`

Functions

`wlanHESIGABitRecover`

interpretHESIGBCommonBits

Update HE MU transmission parameters with HE-SIG-B common field bits

Syntax

```
cfgUpdated = interpretHESIGBCommonBits(cfg,bits,status)
[cfgUpdated,failInterpretation] = interpretHESIGBCommonBits(cfg,bits,status)
```

Description

`cfgUpdated = interpretHESIGBCommonBits(cfg,bits,status)` updates high-efficiency multi-user (HE MU) transmission parameters `cfg` by interpreting `bits`, the recovered HE-SIG-B common field bits, and `status`, the result of content channel decoding. The function populates the properties of `cfg` that are relevant to the HE-SIG-B common field and returns updated HE MU transmission parameters `cfgUpdated`. If you use this syntax and the function cannot interpret the recovered HE-SIG-B common field bits, the function does not return an output and issues an error message.

`[cfgUpdated,failInterpretation] = interpretHESIGBCommonBits(cfg,bits,status)` returns the result of HE-SIG-B common field interpretation. If you use this syntax and the function cannot interpret the recovered HE-SIG-B common field bits, the function returns the `failInterpretation` output as `1` and `cfgUpdated` as `cfg` without updating any property values.

Examples

Update HE MU Recovery Configuration Object

Update a WLAN HE recovery configuration object by interpreting recovered HE-SIG-A and HE-SIG-B information bits.

Generate HE MU Waveform

Create a WLAN HE MU configuration object, setting the allocation index to `0`.

```
cfgHEMU = wlanHEMUConfig(0);
```

Generate a WLAN waveform and PPDU field indices for the specified configuration.

```
waveform = wlanWaveformGenerator(1,cfgHEMU);
ind = wlanFieldIndices(cfgHEMU);
```

Recover L-SIG Bits

Create a WLAN recovery configuration object, specifying an HE MU packet format and the length of the L-SIG field.

```
cfg = wlanHERecoveryConfig('PacketFormat','HE-MU','ChannelBandwidth','CBW20');
```

Decode the L-SIG field and obtain the orthogonal frequency-division multiplexing (OFDM) information. The recovery configuration object requires this information to obtain the L-SIG length.

```
lsig = waveform(ind.LSIG(1):ind.LSIG(2));
lsigDemod = wlanHEDemodulate(lsig,'L-SIG',cfg.ChannelBandwidth);
info = wlanHEOFDMInfo('L-SIG',cfg.ChannelBandwidth);
lsigDemod = lsigDemod(info.DataIndices,:);
```

Recover the L-SIG bits and related information, making sure that the bits pass the parity check, and update the recovery configuration object with the L-SIG length. For this example we assume a noiseless channel. For more realistic results you can pass the waveform through an 802.11ax™ channel model by using the `wlanTGaxChannel` System object™ and work with the received waveform.

```
csi = ones(52,1);
[lsigBits,failCheck,lsigInfo] = wlanLSIGBitRecover(lsigDemod,0,csi);
cfg.LSIGLength = lsigInfo.Length;
```

Update Recovery Configuration Object with HE-SIG-A Bits

Decode the HE-SIG-A field and recover the HE-SIG-A bits, ensuring that the bits pass the cyclic redundancy check (CRC).

```
siga = waveform(ind.HESIGA(1):ind.HESIGA(2));
sigaDemod = wlanHEDemodulate(siga,'HE-SIG-A',cfg.ChannelBandwidth);
sigaDemod = sigaDemod(info.DataIndices,:);
[sigaBits,failCRC] = wlanHESIGABitRecover(sigaDemod,0,csi);
disp(failCRC)
```

```
   0
```

Update the recovery configuration object with the recovered HE-SIG-A bits. Display the updated object. A property value of `-1` or `'Unknown'` indicates an unknown or undefined property, which can be updated after decoding the HE-SIG-B common and user fields of the HE MU packet.

```
[cfg,failInterpretation] = interpretHESIGABits(cfg,sigaBits)
```

```
cfg =
  wlanHERecoveryConfig with properties:

                   PacketFormat: 'HE-MU'
              ChannelBandwidth: 'CBW20'
                     LSIGLength: 878
                SIGBCompression: 0
                        SIGBMCS: 0
                        SIGBDCM: 0
        NumSIGBSymbolsSignaled: 10
                          STBC: 0
               LDPCExtraSymbol: 1
          PreFECPaddingFactor: 1
               PEDisambiguity: 0
                GuardInterval: 3.2000
                    HELTFType: 4
              NumHELTFSymbols: 1
             UplinkIndication: 0
                      BSSColor: 0
                 SpatialReuse: 0
                 TXOPDuration: 127
                  HighDoppler: 0
              AllocationIndex: -1
    NumUsersPerContentChannel: -1
```

```
         RUTotalSpaceTimeStreams: -1
                          RUSize: -1
                         RUIndex: -1
                           STAID: -1
                             MCS: -1
                             DCM: -1
                  ChannelCoding: 'Unknown'
                    Beamforming: -1
              NumSpaceTimeStreams: -1
    SpaceTimeStreamStartingIndex: -1


failInterpretation = logical
   0
```

Update Recovery Configuration Object with HE-SIG-B Common Field Bits

Decode the HE-SIG-B common field, ensuring that all content channels pass the CRC.

```
len = getSIGBLength(cfg);
sigbCommon = waveform(double(ind.HESIGA(2))+(1:len.NumSIGBCommonFieldSamples),:);
sigbCommonDemod = wlanHEDemodulate(sigbCommon,'HE-SIG-B',cfgHEMU.ChannelBandwidth);
sigbCommonDemod = sigbCommonDemod(info.DataIndices);
[sigbCommonBits,status,~] = wlanHESIGBCommonBitRecover(sigbCommonDemod,0,csi,cfg);
disp(status)
```

```
Success
```

Update the recovery configuration object with the recovered HE-SIG-B common field bits and display the updated object. A field returned as `-1` or `'Unknown'` indicates an unknown or undefined property value, which can be updated after decoding the HE-SIG-B user field of the HE MU packet.

```
[cfg,failInterpretation] = interpretHESIGBCommonBits(cfg,sigbCommonBits,status)
```

```
cfg =
  wlanHERecoveryConfig with properties:

                    PacketFormat: 'HE-MU'
                ChannelBandwidth: 'CBW20'
                       LSIGLength: 878
                 SIGBCompression: 0
                         SIGBMCS: 0
                         SIGBDCM: 0
           NumSIGBSymbolsSignaled: 10
                             STBC: 0
                  LDPCExtraSymbol: 1
              PreFECPaddingFactor: 1
                   PEDisambiguity: 0
                    GuardInterval: 3.2000
                        HELTFType: 4
                  NumHELTFSymbols: 1
                 UplinkIndication: 0
                          BSSColor: 0
                     SpatialReuse: 0
                     TXOPDuration: 127
                      HighDoppler: 0
                  AllocationIndex: 0
        NumUsersPerContentChannel: 9
```

```
         RUTotalSpaceTimeStreams: -1
                          RUSize: -1
                         RUIndex: -1
                           STAID: -1
                             MCS: -1
                             DCM: -1
                  ChannelCoding: 'Unknown'
                    Beamforming: -1
              NumSpaceTimeStreams: -1
    SpaceTimeStreamStartingIndex: -1


failInterpretation = logical
   0
```

Update Recovery Configuration Object with HE-SIG-B User Field Bits

Decode the HE-SIG-B user field, ensuring that all users pass the CRC.

```
sigbUser = waveform(ind.HESIGB(1):ind.HESIGB(2));
sigbUserDemod = wlanHEDemodulate(sigbUser,'HE-SIG-B',cfgHEMU.ChannelBandwidth);
sigbUserDemod = sigbUserDemod(info.DataIndices,:);
[sigbUserBits,failCRC,~] = wlanHESIGBUserBitRecover(sigbUserDemod,0,csi,cfg);
disp(failCRC)
```

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

Update the recovery configuration object with the recovered HE-SIG-B user field bits.

```
[user,failInterpretation] = interpretHESIGBUserBits(cfg,sigbUserBits,failCRC);
```

Display the results of interpretation and the third element of the `user` output.

```
disp(failInterpretation)
```

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

```
disp(user{3})
```

```
  wlanHERecoveryConfig with properties:

                  PacketFormat: 'HE-MU'
               ChannelBandwidth: 'CBW20'
                     LSIGLength: 878
                SIGBCompression: 0
                        SIGBMCS: 0
                        SIGBDCM: 0
         NumSIGBSymbolsSignaled: 10
                           STBC: 0
                LDPCExtraSymbol: 1
           PreFECPaddingFactor: 1
                PEDisambiguity: 0
                 GuardInterval: 3.2000
                     HELTFType: 4
                NumHELTFSymbols: 1
               UplinkIndication: 0
                        BSSColor: 0
                   SpatialReuse: 0
```

```
                TXOPDuration: 127
                  HighDoppler: 0
             AllocationIndex: 0
    NumUsersPerContentChannel: 9
      RUTotalSpaceTimeStreams: 1
                       RUSize: 26
                      RUIndex: 3
                        STAID: 0
                          MCS: 0
                          DCM: 0
                ChannelCoding: 'LDPC'
                  Beamforming: 0
          NumSpaceTimeStreams: 1
SpaceTimeStreamStartingIndex: 1
```

Input Arguments

`cfg` — HE MU transmission parameters before interpretation of HE-SIG-B common field bits

`wlanHERecoveryConfig` object

HE MU transmission parameters before interpretation of HE-SIG-B common field bits, specified as a `wlanHERecoveryConfig` object.

`bits` — Recovered HE-SIG-B common field bits

binary-valued matrix

Recovered HE-SIG-B common field bits, specified as binary-valued matrix whose size depends on the channel bandwidth of the transmission.

Channel Bandwidth	Size of `bits`
20 MHz	18-by-1
40 MHz	18-by-2
80 MHz	27-by-2
160 MHz	43-by-2

Data Types: `int8`

`status` — Result of content channel decoding

character vector

Result of content channel decoding, specified as one of these values.

- `'Success'` – All content channels passed the cyclic redundancy check (CRC).
- `'ContentChannel1Fail'` – Content channel 1 failed the CRC, and the number of HE-SIG-B symbols signaled in the HE-SIG-A field is less than 16.
- `'ContentChannel2Fail'` – Content channel 2 failed the CRC, and the number of HE-SIG-B symbols signaled in the HE-SIG-A field is less than 16.
- `'UnknownNumUsersContentChannel1'` – Content channel 1 failed the CRC, and the number of HE-SIG-B symbols signaled in the HE-SIG-B field is 16.
- `'UnknownNumUsersContentChannel2'` – Content channel 2 failed the CRC, and the number of HE-SIG-B symbols signaled in the HE-SIG-B field is 16.

- `'AllContentChannelCRCFail'` – All content channels failed the CRC.

The value of this input depends on the result of the cyclic redundancy check (CRC) per content channel and the number of HE-SIG-B symbols signaled in the HE-SIG-A field. for more information, see `wlanHESIGBCommonBitRecover`.

Data Types: `char` | `string`

Output Arguments

`cfgUpdated` — Updated HE MU transmission parameters

`wlanHERecoveryConfig` object

Updated HE MU transmission parameters, returned as a `wlanHERecoveryConfig` object. The function updates the properties of this object in accordance with the recovered HE-SIG-B common field bits `bits`. For information on the contents of the HE-SIG-B field, see [1].

`failInterpretation` — Result of HE-SIG-B common field interpretation

`0` | `1`

Result of HE-SIG-B common field interpretation, returned as a logical `0` or `1`. A value of `1` indicates that the function cannot interpret the recovered HE-SIG-B common field bits. A value of `0` indicates that the function successfully interprets the HE-SIG-B common field bits.

Data Types: `logical`

Version History

Introduced in R2020b

References

[1] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`wlanHERecoveryConfig`

Functions

`wlanHESIGBCommonBitRecover`

interpretHESIGBUserBits

Update HE MU transmission parameters with HE-SIG-B user field bits

Syntax

```
user = interpretHESIGBUserBits(cfg,bits,failCRC)
[user,failInterpretation] = interpretHESIGBUserBits(cfg,bits,failCRC)
```

Description

`user = interpretHESIGBUserBits(cfg,bits,failCRC)` updates high-efficiency multi-user (HE MU) transmission parameters `cfg` by interpreting recovered HE-SIG-B user field bits. The function populates the properties of `cfg` that are relevant to the HE-SIG-B user field and returns updated HE MU transmission parameters `user`, a cell array of `wlanHERecoveryConfig` objects. If you use this syntax and the function cannot interpret the recovered HE-SIG-B user field bits, the function does not return an output and issues an error message.

`[user,failInterpretation] = interpretHESIGBUserBits(cfg,bits,failCRC)` returns the result of HE-SIG-B user field interpretation. If you use this syntax and the function cannot interpret the recovered HE-SIG-B user field bits for the *k*th user, the function returns the *k*th element of the `failInterpretation` output as `1` and does not return the *k*th element of the `user` output.

Examples

Update HE MU Recovery Configuration Object

Update a WLAN HE recovery configuration object by interpreting recovered HE-SIG-A and HE-SIG-B information bits.

Generate HE MU Waveform

Create a WLAN HE MU configuration object, setting the allocation index to `0`.

```
cfgHEMU = wlanHEMUConfig(0);
```

Generate a WLAN waveform and PPDU field indices for the specified configuration.

```
waveform = wlanWaveformGenerator(1,cfgHEMU);
ind = wlanFieldIndices(cfgHEMU);
```

Recover L-SIG Bits

Create a WLAN recovery configuration object, specifying an HE MU packet format and the length of the L-SIG field.

```
cfg = wlanHERecoveryConfig('PacketFormat','HE-MU','ChannelBandwidth','CBW20');
```

Decode the L-SIG field and obtain the orthogonal frequency-division multiplexing (OFDM) information. The recovery configuration object requires this information to obtain the L-SIG length.

```
lsig = waveform(ind.LSIG(1):ind.LSIG(2));
lsigDemod = wlanHEDemodulate(lsig,'L-SIG',cfg.ChannelBandwidth);
info = wlanHEOFDMInfo('L-SIG',cfg.ChannelBandwidth);
lsigDemod = lsigDemod(info.DataIndices,:);
```

Recover the L-SIG bits and related information, making sure that the bits pass the parity check, and update the recovery configuration object with the L-SIG length. For this example we assume a noiseless channel. For more realistic results you can pass the waveform through an 802.11ax™ channel model by using the `wlanTGaxChannel` System object™ and work with the received waveform.

```
csi = ones(52,1);
[lsigBits,failCheck,lsigInfo] = wlanLSIGBitRecover(lsigDemod,0,csi);
cfg.LSIGLength = lsigInfo.Length;
```

Update Recovery Configuration Object with HE-SIG-A Bits

Decode the HE-SIG-A field and recover the HE-SIG-A bits, ensuring that the bits pass the cyclic redundancy check (CRC).

```
siga = waveform(ind.HESIGA(1):ind.HESIGA(2));
sigaDemod = wlanHEDemodulate(siga,'HE-SIG-A',cfg.ChannelBandwidth);
sigaDemod = sigaDemod(info.DataIndices,:);
[sigaBits,failCRC] = wlanHESIGABitRecover(sigaDemod,0,csi);
disp(failCRC)
```

```
   0
```

Update the recovery configuration object with the recovered HE-SIG-A bits. Display the updated object. A property value of `-1` or `'Unknown'` indicates an unknown or undefined property, which can be updated after decoding the HE-SIG-B common and user fields of the HE MU packet.

```
[cfg,failInterpretation] = interpretHESIGABits(cfg,sigaBits)
```

```
cfg =
  wlanHERecoveryConfig with properties:

                    PacketFormat: 'HE-MU'
               ChannelBandwidth: 'CBW20'
                     LSIGLength: 878
                 SIGBCompression: 0
                        SIGBMCS: 0
                        SIGBDCM: 0
          NumSIGBSymbolsSignaled: 10
                           STBC: 0
                 LDPCExtraSymbol: 1
             PreFECPaddingFactor: 1
                  PEDisambiguity: 0
                   GuardInterval: 3.2000
                       HELTFType: 4
                NumHELTFSymbols: 1
               UplinkIndication: 0
                        BSSColor: 0
                   SpatialReuse: 0
                   TXOPDuration: 127
                    HighDoppler: 0
                AllocationIndex: -1
       NumUsersPerContentChannel: -1
```

```
         RUTotalSpaceTimeStreams: -1
                          RUSize: -1
                         RUIndex: -1
                           STAID: -1
                             MCS: -1
                             DCM: -1
                  ChannelCoding: 'Unknown'
                    Beamforming: -1
              NumSpaceTimeStreams: -1
    SpaceTimeStreamStartingIndex: -1


failInterpretation = logical
   0
```

Update Recovery Configuration Object with HE-SIG-B Common Field Bits

Decode the HE-SIG-B common field, ensuring that all content channels pass the CRC.

```
len = getSIGBLength(cfg);
sigbCommon = waveform(double(ind.HESIGA(2))+(1:len.NumSIGBCommonFieldSamples),:);
sigbCommonDemod = wlanHEDemodulate(sigbCommon,'HE-SIG-B',cfgHEMU.ChannelBandwidth);
sigbCommonDemod = sigbCommonDemod(info.DataIndices);
[sigbCommonBits,status,~] = wlanHESIGBCommonBitRecover(sigbCommonDemod,0,csi,cfg);
disp(status)
```

```
Success
```

Update the recovery configuration object with the recovered HE-SIG-B common field bits and display the updated object. A field returned as `-1` or `'Unknown'` indicates an unknown or undefined property value, which can be updated after decoding the HE-SIG-B user field of the HE MU packet.

```
[cfg,failInterpretation] = interpretHESIGBCommonBits(cfg,sigbCommonBits,status)
```

```
cfg =
  wlanHERecoveryConfig with properties:

                    PacketFormat: 'HE-MU'
                ChannelBandwidth: 'CBW20'
                       LSIGLength: 878
                 SIGBCompression: 0
                         SIGBMCS: 0
                         SIGBDCM: 0
           NumSIGBSymbolsSignaled: 10
                             STBC: 0
                  LDPCExtraSymbol: 1
              PreFECPaddingFactor: 1
                   PEDisambiguity: 0
                    GuardInterval: 3.2000
                        HELTFType: 4
                  NumHELTFSymbols: 1
                 UplinkIndication: 0
                          BSSColor: 0
                     SpatialReuse: 0
                     TXOPDuration: 127
                      HighDoppler: 0
                  AllocationIndex: 0
        NumUsersPerContentChannel: 9
```

```
         RUTotalSpaceTimeStreams: -1
                          RUSize: -1
                         RUIndex: -1
                           STAID: -1
                             MCS: -1
                             DCM: -1
                  ChannelCoding: 'Unknown'
                    Beamforming: -1
              NumSpaceTimeStreams: -1
    SpaceTimeStreamStartingIndex: -1


failInterpretation = logical
   0
```

Update Recovery Configuration Object with HE-SIG-B User Field Bits

Decode the HE-SIG-B user field, ensuring that all users pass the CRC.

```
sigbUser = waveform(ind.HESIGB(1):ind.HESIGB(2));
sigbUserDemod = wlanHEDemodulate(sigbUser,'HE-SIG-B',cfgHEMU.ChannelBandwidth);
sigbUserDemod = sigbUserDemod(info.DataIndices,:);
[sigbUserBits,failCRC,~] = wlanHESIGBUserBitRecover(sigbUserDemod,0,csi,cfg);
disp(failCRC)
```

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

Update the recovery configuration object with the recovered HE-SIG-B user field bits.

```
[user,failInterpretation] = interpretHESIGBUserBits(cfg,sigbUserBits,failCRC);
```

Display the results of interpretation and the third element of the `user` output.

```
disp(failInterpretation)
```

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

```
disp(user{3})
```

```
  wlanHERecoveryConfig with properties:

                  PacketFormat: 'HE-MU'
               ChannelBandwidth: 'CBW20'
                     LSIGLength: 878
                SIGBCompression: 0
                        SIGBMCS: 0
                        SIGBDCM: 0
         NumSIGBSymbolsSignaled: 10
                           STBC: 0
                LDPCExtraSymbol: 1
           PreFECPaddingFactor: 1
                PEDisambiguity: 0
                 GuardInterval: 3.2000
                     HELTFType: 4
               NumHELTFSymbols: 1
              UplinkIndication: 0
                       BSSColor: 0
                  SpatialReuse: 0
```

```
                TXOPDuration: 127
                  HighDoppler: 0
             AllocationIndex: 0
    NumUsersPerContentChannel: 9
      RUTotalSpaceTimeStreams: 1
                       RUSize: 26
                      RUIndex: 3
                        STAID: 0
                          MCS: 0
                          DCM: 0
               ChannelCoding: 'LDPC'
                  Beamforming: 0
          NumSpaceTimeStreams: 1
SpaceTimeStreamStartingIndex: 1
```

Input Arguments

`cfg` — HE MU transmission parameters before interpretation of HE-SIG-B user bits

`wlanHERecoveryConfig` object

HE MU transmission parameters before interpretation of HE-SIG-A bits, specified as a `wlanHERecoveryConfig` object.

`bits` — Recovered HE-SIG-B user field bits

binary-valued matrix

Recovered HE-SIG-B user field bits, specified as a binary-valued matrix of size 21-by-N_{users}, where N_{users} is the number of users in the transmission.

Data Types: `double` | `int8`

`failCRC` — CRC result for each user

logical-valued row vector

CRC result for each user, specified as a logical-valued row vector of length N_{users}, where N_{users} is the number of users in the transmission. Each element of this argument represents the result of the CRC for the corresponding user. A value of `1` in the kth entry indicates that the Kth user's bits failed the CRC. A value of `0` in the kth entry indicates that the kth user's bits passed the CRC.

Data Types: `logical`

Output Arguments

`user` — Updated HE MU transmission parameters

cell array of `wlanHERecoveryConfig` objects

Updated HE MU transmission parameters, returned as a cell array of `wlanHERecoveryConfig` objects. The function updates the properties of each object in accordance with the recovered HE-SIG-B user field bits. For information on the contents of the HE-SIG-B field, see [1].

Data Types: `cell`

`failInterpretation` — Result of HE-SIG-B user field interpretation

logical-valued row vector

Result of HE-SIG-B user field interpretation, returned as a logical-valued row vector of length N_{users}, where N_{users} is the number of users in the transmission. A value of `1` in the kth entry indicates that the

function cannot interpret the recovered HE-SIG-B user field bits for user k. A value of `0` in the kth entry indicates that the function can successfully interpret the HE-SIG-B user field bits for user k.

Data Types: `logical`

Version History

Introduced in R2020b

References

[1] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`wlanHERecoveryConfig`

Functions

`wlanHESIGBUserBitRecover`

numPostFECPaddingBits

Required number of post-FEC padding bits

Syntax

```
n = numPostFECPaddingBits(cfg)
```

Description

`n = numPostFECPaddingBits(cfg)` calculates the required number of post-FEC padding bits in an extremely high-throughput (EHT) transmission parameterized by `cfg`.

Examples

Calculate Number of Post-FEC Padding Bits for EHT Transmission

Parameterize an extremely high-throughput multi-user (EHT MU) transmission by creating a non-OFDMA `wlanEHTMUConfig` object. Set the channel bandwidth to 320 MHz.

```
cfgEHTMU = wlanEHTMUConfig("CBW320");
```

Calculate the number of post-FEC padding bits required for the user.

```
n = numPostFECPaddingBits(cfgEHTMU)
```

```
n = 968
```

Input Arguments

`cfg` — EHT transmission parameters

`wlanEHTMUConfig` object

EHT transmission parameters, specified as a `wlanEHTMUConfig` object.

Output Arguments

`n` — Required number of post-FEC padding bits

vector of integers

Required number of post-FEC padding bits, returned as a vector of integers. The length of `n` is equal to the number of users. The vector's values are equal to the required number of bits for each user.

Version History

Introduced in R2022b

See Also

Objects
`wlanEHTMUConfig` | `wlanEHTUser`

packetFormat

WLAN packet format

Syntax

```
format = packetFormat(cfg)
```

Description

`format = packetFormat(cfg)` returns the packet format of the WLAN transmission parameterized by physical layer (PHY) format configuration `cfg`.

Examples

Create WLAN S1G Configuration Object and Return Packet Format

Create an S1G configuration object with default property values.

```
cfgS1G = wlanS1GConfig;
```

Compute and display the packet format. The default properties specify a transmission with short preamble.

```
format = packetFormat(cfgS1G);
disp(format)
```

```
S1G-Short
```

Now create an S1G configuration object, specifying a long preamble.

```
cfgS1GLongPreamble = wlanS1GConfig('Preamble','Long');
```

Compute and display the packet format.

```
format = packetFormat(cfgS1GLongPreamble);
disp(format)
```

```
S1G-Long
```

Input Arguments

`cfg` — PHY format configuration

`wlanEHTMUConfig` object | `wlanHESUConfig` object | `wlanHEMUConfig` object | `wlanHETBConfig` object | `wlanHERecoveryConfig` object | `wlanS1GConfig` object | `wlanWURConfig` object

PHY format configuration, specified as one of these objects: `wlanEHTMUConfig`, `wlanHESUConfig`, `wlanHEMUConfig`, `wlanHETBConfig`, `wlanHERecoveryConfig`, `wlanS1GConfig`, or `wlanWURConfig`.

Output Arguments

`format` — WLAN packet format
character vector

WLAN packet format, returned as a character vector.

- When you specify the `cfg` input as a `wlanEHTMUConfig` object, the function returns `'EHT-MU'`.
- When you specify the `cfg` input as a `wlanHESUConfig` object, the function returns `'HE-EXT-SU'` or `'HE-SU'`.
- When you specify the `cfg` input as a `wlanHEMUConfig` object, the function returns `'HE-MU'`.
- When you specify the `cfg` input as a `wlanHETBConfig` object, the function returns `'HE-TB'`.
- When you specify the `cfg` input as a `wlanHERecoveryConfig` object, the function returns `'HE-EXT-SU'`, `'HE-SU'`, or `'HE-MU'`.
- When you specify the `cfg` input as a `wlanS1GConfig` object, the function returns `'S1G-1M'`, `'S1G-Short'`, or `'S1G-Long'`.
- When you specify the `cfg` input as a `wlanWURConfig` object, the function returns `'WUR'`.

Version History

Introduced in R2017b

EHT MU packet format

You can specify `cfg` as an object of type `wlanEHTMUConfig`.

Extended Capabilities

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

See Also

Objects
`wlanS1GConfig` | `wlanEHTMUConfig` | `wlanHESUConfig` | `wlanHEMUConfig` | `wlanHETBConfig` | `wlanWURConfig`

Functions
`getPSDULength` | `ruInfo` | `showAllocation` | `wlanWaveformGenerator`

phyType

Return DMG PHY modulation type

Syntax

```
type = phyType(cfg)
```

Description

`type = phyType(cfg)` returns the DMG physical layer (PHY) modulation type, based on the configuration of the DMG object.

Examples

Create DMG Configuration Object and Return DMG PHY Type

Create DMG configuration objects and change the default property settings by using dot notation. Use the `phyType` object function to access the DMG PHY modulation type.

Create a DMG configuration object and return the DMG PHY modulation type. By default, the configuration object creates properties to model the DMG control PHY.

```
dmg = wlanDMGConfig;
phyType(dmg)
```

```
ans =
'Control'
```

Model the SC PHY by modifying the defaults by using the dot notation to specify an MCS of 5.

```
dmg.MCS = 5;
phyType(dmg)
```

```
ans =
'SC'
```

Input Arguments

`cfg` — DMG PPDU configuration

`wlanDMGConfig` object

DMG PPDU configuration, specified as a `wlanDMGConfig` object.

Output Arguments

`type` — DMG PHY modulation type

`'Control'` | `'SC'` | `'OFDM'`

DMG PHY modulation type, specified as '`Control`', '`SC`', or '`OFDM`'.

Data Types: `char` | `string`

Version History

Introduced in R2017b

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`wlanDMGConfig`

ruInfo

Resource unit allocation information

Syntax

```
info = ruInfo(cfg)
```

Description

`info = ruInfo(cfg)` returns `info`, resource unit (RU) allocation information for a format configuration object `cfg`.

Examples

Get RU Allocation Information for HE Configuration Objects

Use the `ruInfo` function to get the resource unit information of single user and multi-user HE configuration objects.

Get Single User RU Allocation Information

Create a single user HE configuration object. Get and display the RU allocation information for the configured object.

```
hesu = wlanHESUConfig;
ru = ruInfo(hesu)
```

```
ru = struct with fields:
                    NumUsers: 1
                      NumRUs: 1
                   RUIndices: 1
                     RUSizes: 242
                NumUsersPerRU: 1
    NumSpaceTimeStreamsPerRU: 1
       PowerBoostFactorPerRU: 1
                   RUNumbers: 1
```

Get Multiuser RU Allocation Information

Create a multiuser HE configuration object with the allocation index set to 5, which configures the object with seven users. Get and display the RU allocation information for the configured object.

```
hemu = wlanHEMUConfig(5);
ru = ruInfo(hemu)
```

```
ru = struct with fields:
                    NumUsers: 7
                      NumRUs: 7
                   RUIndices: [1 2 2 5 6 7 4]
                     RUSizes: [26 26 52 26 26 26 52]
```

```
              NumUsersPerRU: [1 1 1 1 1 1 1]
    NumSpaceTimeStreamsPerRU: [1 1 1 1 1 1 1]
       PowerBoostFactorPerRU: [1 1 1 1 1 1 1]
                   RUNumbers: [1 2 3 4 5 6 7]
```

Inactivate RU for Second User in Multiuser HE

Create a two user HE configuration object. Make the RU for the second user inactive by setting the station identity to 2046.

Create a multiuser HE configuration object with the allocation index set to 96, which configures an object for two users. The resource information shows that RUs are active for two users.

```
hemu = wlanHEMUConfig(96);
ruInfo(hemu)
```

```
ans = struct with fields:
                     NumUsers: 2
                       NumRUs: 2
                    RUIndices: [1 2]
                      RUSizes: [106 106]
                NumUsersPerRU: [1 1]
    NumSpaceTimeStreamsPerRU: [1 1]
       PowerBoostFactorPerRU: [1 1]
                   RUNumbers: [1 2]
```

Set the station identity to 2046 for the second user. The RU allocation information now shows that RUs are active only for RU index 1.

```
hemu.User{2}.STAID = 2046;
ruInfo(hemu)
```

```
ans = struct with fields:
                     NumUsers: 2
                       NumRUs: 1
                    RUIndices: 1
                      RUSizes: 106
                NumUsersPerRU: 1
    NumSpaceTimeStreamsPerRU: 1
       PowerBoostFactorPerRU: 1
                   RUNumbers: 1
```

Input Arguments

`cfg` — Format configuration object

`wlanEHTMUConfig` object | `wlanHESUConfig` object | `wlanHEMUConfig` object | `wlanHETBConfig` object

Format configuration object, specified as an object of type `wlanEHTMUConfig`, `wlanHEMUConfig`, `wlanHESUConfig`, or `wlanHETBConfig`.

Output Arguments

`info` — Information about RU properties of object

structure

Information about the RU properties of the input object, returned as a structure.

`NumUsers` — Number of users

integer in the range [1, 144]

Number of users, returned as an integer in the range [1, 144]. When you specify `cfg` as an object of type `wlanHEMUConfig`, `wlanHESUConfig`, or `wlanHETBConfig`, the maximum number of users is 74. When you specify `cfg` as an object of type `wlanEHTMUConfig`, the maximum number of users is 144.

Data Types: `double`

`NumRUs` — Number of RUs

integer in the range [1, 144]

Number of RUs, returned as an integer in the range [1, 144]. When you specify `cfg` as an object of type `wlanHEMUConfig`, `wlanHESUConfig`, or `wlanHETBConfig`, the maximum number of RUs is 74. When you specify `cfg` as an object of type `wlanEHTMUConfig`, the maximum number of RUs is 144.

Data Types: `double`

`RUIndices` — RU indices

integer | vector | cell array

RU indices, returned as:

- An integer or a 1-by-`NumRUs` vector with elements that have integer values in the range [1, 74]. This applies when you specify `cfg` as an object of type `wlanHEMUConfig`, `wlanHESUConfig`, or `wlanHETBConfig`.
- A 1-by-`NumRUs` cell array of vectors with elements that have integer values in the range [1, 144]. This applies when you specify `cfg` as an object of type `wlanEHTMUConfig`.

Data Types: `double` | `cell`

`RUSizes` — Resource unit sizes

integer | vector | cell array

Resource unit sizes, returned as:

- An integer or a 1-by-`NumRUs` vector with elements that have integer values of `26`, `52`, `106`, `242`, `484`, `968`, `996`, or `1992`. This applies when you specify `cfg` as an object of type `wlanHEMUConfig`, `wlanHESUConfig`, or `wlanHETBConfig`.
- A 1-by-`NumRUs` cell array of vectors with elements that have integer values of `26`, `52`, `106`, `242`, `484`, `968`, `996`, `1992`, or `3984`. This applies when you specify `cfg` as an object of type `wlanEHTMUConfig`.

Data Types: `double`

`NumUsersPerRU` — Number of users per RU

integer | vector

Number of users per RU, returned as an integer or a 1-by-`NumRUs` vector with elements that have integer values in the range [1, 8].

Data Types: `double`

`NumSpaceTimeStreamsPerRU` — Number of space-time streams per RU
integer | vector

Number of space-time streams per RU, returned as an integer or a 1-by-`NumRUs`vector with elements that have integer values in the range [1, 8].

Data Types: `double`

`PowerBoostFactorPerRU` — Power boost factor per RU
integer | vector

Power boost factor per RU, returned as an integer or a 1-by-`NumRUs` vector with elements that have scalar values in the range [0.5, 2].

Data Types: `double`

`RUNumbers` — RU numbers
integer | vector

RU numbers, returned as an integer or a 1-by-`NumRUs` vector with elements that have integer values in the range [1, 144]. `RUNumbers` correspond to the indices for each active RU configured in the `cfg`.RU object. An RU is inactive when it contains a single station with its station identifier set to 2046.

Data Types: `double`

Data Types: `struct`

Version History

Introduced in R2018b

EHT MU packet format

You can specify `cfg` as an object of type `wlanEHTMUConfig`.

Extended Capabilities

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

See Also

Objects
`wlanEHTMUConfig` | `wlanHEMUConfig` | `wlanHESUConfig` | `wlanHETBConfig`

Functions
`getPSDULength` | `packetFormat` | `showAllocation` | `wlanWaveformGenerator`

scramblerRange

Get scrambler initialization range

Syntax

```
[range,numBits] = scramblerRange(cfg)
```

Description

`[range,numBits] = scramblerRange(cfg)` returns `range`, the scrambler initialization range, and `numBits`, the number of pseudorandom bits required for scrambler initialization in a non-high-throughput (non-HT) transmission with parameters `cfg`.

Examples

Generate Data Field Signal of Non-HT Transmission

Configure transmission parameters by creating a `wlanNonHTConfig` object, specifying a channel bandwidth of 80 MHz and static bandwidth operation.

```
cfg = wlanNonHTConfig('ChannelBandwidth','CBW80','SignalChannelBandwidth',true, ...
     'BandwidthOperation','Static');
```

Generate a random PSDU of the appropriate length.

```
psdu = randi([0 1],8*cfg.PSDULength,1,'int8');
```

Generate the initial pseudorandom scrambler sequence.

```
[range,numBits] = scramblerRange(cfg);
scramInit = randi(range);
```

Generate the non-HT Data field signal.

```
y = wlanNonHTData(psdu,cfg,scramInit);
```

Input Arguments

`cfg` — Non-HT transmission parameters

`wlanNonHTConfig` object

Non-HT transmission parameters, specified as a `wlanNonHTConfig` object.

Output Arguments

`range` — Scrambler initialization range

integer-valued row vector

Scrambler initialization range, returned as an integer-valued row vector of the form [*min max*]. The values of *min* and *max* represent the minimum and maximum values, respectively, of pseudorandom bits required for scrambler initialization in a non-HT transmission.

Data Types: `double`

`numBits` — Number of bits required for scrambler initialization
4 | 5 | 7

Number of bits required by the `wlanWaveformGenerator` or `wlanNonHTData` function for scrambler initialization in a non-HT transmission, returned as `4`, `5`, or `7`. The value of this output depends on whether the transmission signals bandwidth operation in accordance with the `SignalChannelBandwidth` and `BandwidthOperation` property values of the `cfg` input. For more information, see Table 17-7 of [1].

Data Types: `double`

Version History

Introduced in R2020b

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`wlanInterpretScramblerState` | `wlanScramble` | `wlanWaveformGenerator`

Objects

`wlanNonHTConfig`

showAllocation

Resource unit (RU) allocation

Syntax

```
showAllocation(cfg)
showAllocation(cfg,ax)
```

Description

`showAllocation(cfg)` shows the RU allocation in a WLAN transmission parameterized by format configuration `cfg`. You can get more information about an RU by clicking on it.

`showAllocation(cfg,ax)` specifies `ax`, the axes that the function uses to plot the allocation.

Examples

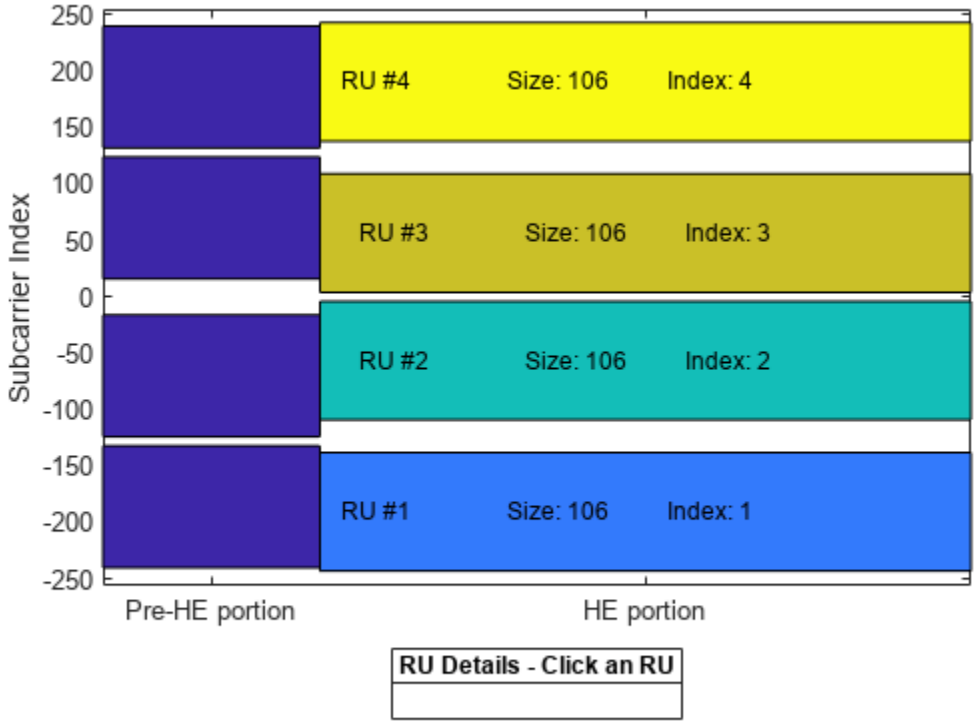
Show RU Allocation in HE MU Transmission

Create a configuration object for an HE MU transmission with a channel bandwidth of 40 MHz.

```
AllocationIndex = [100 98];
cfgHE = wlanHEMUConfig(AllocationIndex);
```

Show the RU allocation in the resultant transmission. This configuration object specifies seven users across four RUs.

```
showAllocation(cfgHE)
```



Input Arguments

`cfg` — Format configuration

`wlanEHTMUConfig` object | `wlanHEMUConfig` object | `wlanHESUConfig` object | `wlanHETBConfig` object

Format configuration, specified as an object of type `wlanEHTMUConfig`, `wlanHEMUConfig`, `wlanHESUConfig`, or `wlanHETBConfig`.

- To show the RU allocation for an EHT multi-user (EHT MU) transmission, specify this input as a `wlanEHTMUConfig` object.
- To show the RU allocation for an HE single-user (HE SU) or HE extended-range SU (HE ER SU) transmission, specify this input as a `wlanHESUConfig` object.
- To show the RU allocation for an HE multi-user (HE MU) transmission, specify this input as a `wlanHEMUConfig` object.
- To show the RU allocation for an HE trigger-based (HE TB) transmission, specify this input as a `wlanHETBConfig` object.

`ax` — Plot axes

`Axes` object

Plot axes, specified as an `Axes` object. For more information, see Axes Properties. If you do not specify this input, the `showAllocation` object function shows the RU allocation plotted on the default axes.

Version History

Introduced in R2019b

EHT MU packet format

You can specify `cfg` as an object of type `wlanEHTMUConfig`.

See Also

Objects
`wlanEHTMUConfig` | `wlanHEMUConfig` | `wlanHESUConfig` | `wlanHETBConfig`

Functions
`getPSDULength` | `packetFormat` | `ruInfo`

showEnvironment

Display channel environment with D-Rays from ray tracing

Syntax

```
showEnvironment(tgay)
showEnvironment(tgay,envOnly)
```

Description

`showEnvironment(tgay)` displays a three-dimensional figure for the IEEE 802.11ay™ (TGay) channel environment determined by the input `wlanTGayChannel` object, `tgay`. The figure shows a schematic depiction of the channel environment, locations of the transmit and receive antenna arrays, and quasi-deterministic strong rays (D-rays) between the arrays determined by ray tracing.

`showEnvironment(tgay,envOnly)` displays the channel environment with the option of turning off ray tracing.

Examples

Filter Dual-Polarized Signal Through 802.11ay Channel

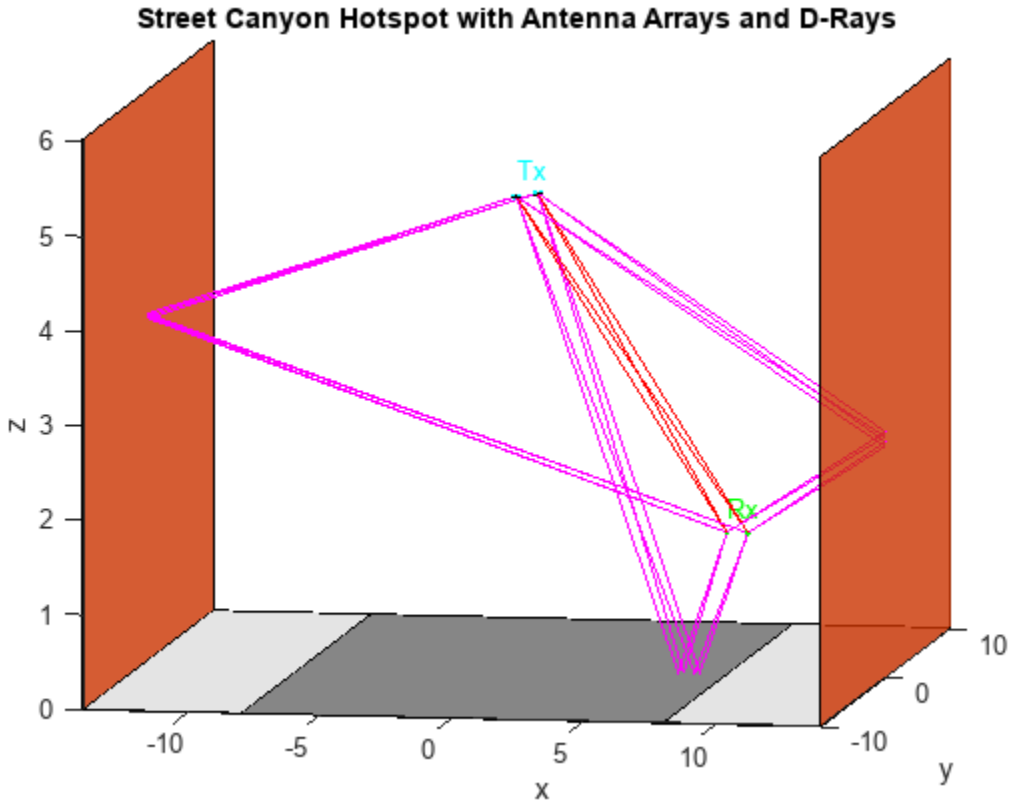
Filter a dual-polarized signal through a WLAN 802.11ay™ channel, specifying a street canyon environment.

Configure a TGay channel System object for a street canyon environment, specifying a user configuration of single-user multiple-input/multiple-output (SU-MIMO) with two transmit antenna arrays and two receive antenna arrays. Specify the transmit antenna arrays as two-element uniform linear arrays (ULAs) and the receive antenna arrays as single isotropic elements. Use a custom beamforming method to specify the transmit and receive beamforming vectors, and specify the source of the random number stream.

```
tgay = wlanTGayChannel('SampleRate',2e9,'Environment','Street canyon hotspot', ...
    'UserConfiguration','SU-MIMO 2x2','ArraySeparation',[0.8 0.8],'ArrayPolarization','Dual, Dua
    'TransmitArray',wlanURAConfig('Size',[1 2]),'TransmitArrayOrientation',[10; 10; 10], ...
    'ReceiveArray',wlanURAConfig('Size',[1 1]),'BeamformingMethod','Custom','NormalizeImpulseRes
    'RandomStream','mt19937ar with seed','Seed',100);
```

Display the environment of the TGay channel.

```
showEnvironment(tgay);
title('Street Canyon Hotspot with Antenna Arrays and D-Rays');
```



Retrieve channel characteristics by using the `info` object function.

```
tgayInfo = tgay.info;
```

Formulate the beamforming vectors in terms of the number of transmit elements, receive elements, transmit streams, and receive streams obtained from `tgayInfo`.

```
NTE = tgayInfo.NumTxElements;
NTS = tgayInfo.NumTxStreams;
NRE = tgayInfo.NumRxElements;
NRS = tgayInfo.NumRxStreams;
tgay.TransmitBeamformingVectors = ones(NTE,NTS)/sqrt(NTE);
tgay.ReceiveBeamformingVectors = ones(NRE,NRS)/sqrt(NRE);
```

Create a random input signal and filter it through the TGay channel.

```
txSignal = complex(rand(100,NTS),rand(100,NTS));
rxSignal = tgay(txSignal);
```

Display Open Area Hotspot Environment

Display the open area hotspot environment for a WLAN TGay channel model.

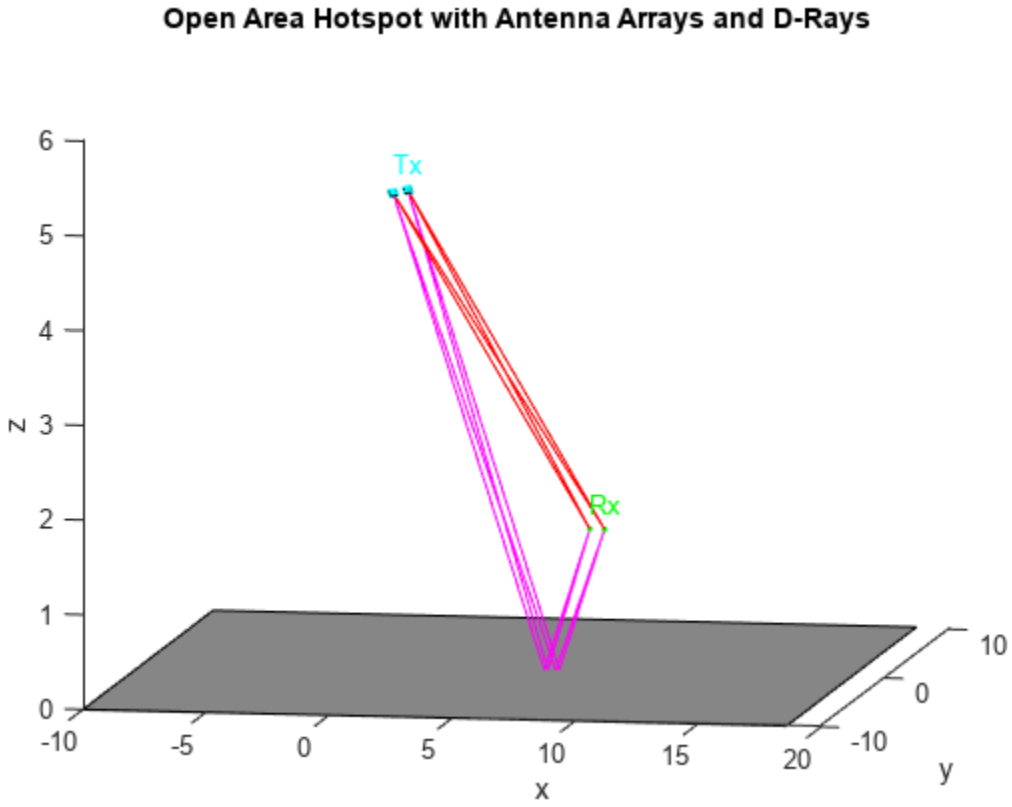
Configure a TGay channel System object for an open area hotspot environment, specifying a user configuration of single-user multiple-input/multiple-output (SU-MIMO) with two transmit antenna

arrays and two receive antenna arrays. Specify the transmit antenna array as a 2x2 uniform rectangular array (URA) and the receive antenna arrays as single isotropic elements. Use a custom beamforming method to specify the transmit and receive beamforming vectors, and specify the source of the random number stream.

```
tgay = wlanTGayChannel('SampleRate',1e9,'Environment','Open area hotspot', ...
    'UserConfiguration','SU-MIMO 2x2','ArraySeparation',[0.6 0.6],'ArrayPolarization','Dual, Dua
    'TransmitArray',wlanURAConfig('Size',[2 2]),'TransmitArrayOrientation',[20; 10; 10], ...
    'ReceiveArray',wlanURAConfig('Size',[1 1]),'BeamformingMethod','Custom', ...
    'NormalizeImpulseResponses',false,'RandomStream','mt19937ar with seed','Seed',150);
```

Display the environment of the TGay channel.

```
envOnly = false;
showEnvironment(tgay,envOnly);
title('Open Area Hotspot with Antenna Arrays and D-Rays');
```



Input Arguments

`tgay` — TGay multipath fading channel

`wlanTGayChannel` System object

TGay multipath fading channel, specified as a `wlanTGayChannel` System object.

`envOnly` — Display environment without D-Rays

`false` (default) | `true`

Display environment without D-Rays, specified as a logical value of `true` or `false`. To display a schematic of the channel environment without D-Rays, set this property to `true`. To display a schematic of the channel environment with D-Rays, set this property to `false`.

Data Types: `logical`

Version History

Introduced in R2019a

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`wlanTGayChannel`

transmitTime

Packet transmission time

Syntax

```
t = transmitTime(cfg)
t = transmitTime(cfg,unit)
```

Description

`t = transmitTime(cfg)` returns the packet transmission time in seconds for the physical layer (PHY) format configuration specified in `cfg`.

`t = transmitTime(cfg,unit)` specifies `unit`, the unit in which the function returns the packet transmission time.

Examples

Create HE SU Configuration Object and Calculate Packet Transmission Time

Create an HE SU configuration object for a 40 MHz transmission.

```
cfgHE = wlanHESUConfig;
cfgHE.ChannelBandwidth = 'CBW40';
```

Calculate and display the packet transmission times in seconds and milliseconds.

```
t = transmitTime(cfgHE)
```

```
t = 1.1600e-04
```

```
t_milli = transmitTime(cfgHE,'milliseconds')
```

```
t_milli = 0.1160
```

Input Arguments

`cfg` — PHY format configuration

`wlanDMGconfig` object | `wlanEHTMUConfig` object | `wlanHEMUconfig` object | `wlanHESUconfig` object | `wlanHETBconfig` object | `wlanHTconfig` object | `wlanNonHTconfig` object | `wlanS1Gconfig` object | `wlanVHTconfig` object | `wlanWURconfig` object

PHY format configuration, specified as one of these objects: `wlanDMGConfig`, `wlanEHTMUConfig`, `wlanHEMUConfig`, `wlanHESUConfig`, `wlanHETBConfig`, `wlanHTConfig`, `wlanNonHTConfig`, `wlanS1GConfig`, `wlanVHTConfig`, or `wlanWURConfig`.

`unit` — Unit of the packet transmission time

`'seconds'` (default) | `'milliseconds'` | `'microseconds'` | `'nanoseconds'`

The unit of the returned packet transmission time, specified as one of `'seconds'`, `'milliseconds'`, `'microseconds'`, or `'nanoseconds'`.

Data Types: `char` | `string`

Output Arguments

`t` — Packet transmission time

positive scalar

Packet transmission time, returned as a positive scalar.

- When you do not specify `unit`, or specify `unit` as `'seconds'`, the time is returned in seconds.
- When you specify `unit` as `'milliseconds'`, the time is returned in milliseconds.
- When you specify `unit` as `'microseconds'`, the time is returned in microseconds.
- When you specify `unit` as `'nanoseconds'`, the time is returned in nanoseconds.

Version History

Introduced in R2022b

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`wlanDMGConfig` | `wlanEHTMUConfig` | `wlanHEMUConfig` | `wlanHESUConfig` | `wlanHETBConfig` | `wlanHTConfig` | `wlanNonHTConfig` | `wlanS1GConfig` | `wlanVHTConfig` | `wlanWURConfig`

wlanAMPDUDeaggregate

Deaggregate A-MPDU and extract MPDUs

Syntax

```
[mpduList,failCRC,status] = wlanAMPDUDeaggregate(ampdu,phyFormat)
[mpduList,failCRC,status] = wlanAMPDUDeaggregate(ampdu,cfgPHY)
[mpduList,failCRC,status] = wlanAMPDUDeaggregate( ___ ,Name,Value)
```

Description

`[mpduList,failCRC,status] = wlanAMPDUDeaggregate(ampdu,phyFormat)` recovers `mpduList`, a list of medium access control (MAC) protocol data units (MPDUs), by deaggregating `ampdu`, an aggregate MPDU (A-MPDU). The function deaggregates the A-MPDU by using parameters appropriate for `phyFormat`, the physical layer (PHY) format.

The function also returns `failCRC`, the delimiter cyclic redundancy check (CRC) status for subframes found in `ampdu`, and `status`, the status of A-MPDU deaggregation.

`[mpduList,failCRC,status] = wlanAMPDUDeaggregate(ampdu,cfgPHY)` deaggreagates the A-MPDU by using PHY transmission parameters `cfgPHY`.

`[mpduList,failCRC,status] = wlanAMPDUDeaggregate( ___ ,Name,Value)` specifies options using one or more name-value pair arguments in addition to any input argument combination from previous syntaxes.

Examples

Deaggregate HT A-MPDU

Create a WLAN MAC frame configuration object, specifying the frame type, frame format, and MPDU aggregation.

```
cfgMAC = wlanMACFrameConfig('FrameType','QoS Data', ...
    'FrameFormat','HT-Mixed','MPDUAggregation',1);
```

Create an HT configuration object, specifying MPDU aggregation.

```
cfgPHY = wlanHTConfig('AggregatedMPDU',1);
```

Create a random payload of eight MSDUs, and then use it generate an A-MPDU in bit form.

```
payload = repmat({randi([0 255],1,40)},1,8);
ampdu = wlanMACFrame(payload,cfgMAC,cfgPHY,'OutputFormat','bits');
```

Return the list of MPDUs by deaggregating the A-MPDU.

```
phyFormat = 'HT';
[mpduList,failCRC,status] = wlanAMPDUDeaggregate(ampdu,phyFormat);
```

Confirm successful deaggregation by displaying the result of the delimiter CRC and the decoding status.

```
disp(failCRC)
```

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

```
disp(status)
```

```
    Success
```

Deaggregate VHT A-MPDU

Create a WLAN MAC frame configuration object, specifying the frame type and frame format.

```
cfgMAC =  wlanMACFrameConfig('FrameType','QoS Data','FrameFormat','VHT');
```

Create a VHT configuration object with default settings.

```
cfgPHY = wlanVHTConfig;
```

Create a random payload of eight MSDUs, and then use it to generate an A-MPDU in bit form.

```
payload = repmat({randi([0 255],1,40)},1,8);
ampdu = wlanMACFrame(payload,cfgMAC,cfgPHY,'Outputformat','bits');
```

Deaggregate the A-MPDU.

```
[mpduList,failCRC,status] = wlanAMPDUDeaggregate(ampdu,cfgPHY);
```

Confirm successful deaggregation by displaying the result of the delimiter CRC and the decoding status.

```
disp(failCRC)
```

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

```
disp(status)
```

```
    Success
```

Decode MPDUs Extracted from A-MPDU

Deaggregate a VHT A-MPDU and decode the extracted MPDUs.

Create a WLAN MAC frame configuration object for a VHT A-MPDU.

```
txCfgMAC = wlanMACFrameConfig('FrameType','QoS Data', ...
    'FrameFormat','VHT');
```

Create a VHT-format configuration object with default settings.

```
cfgPHY = wlanVHTConfig;
```

Generate a random payload of eight MSDUs.

```
txPayload =  repmat({randi([0 255],1,40)},1,8);
```

Generate the A-MPDU containing eight MPDUs for the specified MAC and PHY configurations.

```
ampdu = wlanMACFrame(txPayload,txCfgMAC,cfgPHY);
```

Extract the list of MPDUs by deaggregating the A-MPDU. Display the status of the deaggregation and the delimiter CRC.

```
[mpduList,failCRC,status] = wlanAMPDUDeaggregate(ampdu,cfgPHY, ...
    'DataFormat','octets');
disp(status)
```

```
    Success
```

```
disp(failCRC)
```

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

Decode all of the MPDUs in the extracted. Confirm successful decoding by displaying the status.

```
if strcmp(status,'Success')
    for i = 1:numel(mpduList)
        if ~failCRC(i)
            [cfgMAC,payload,status(i)] = ...
                wlanMPDUDecode(mpduList{i},cfgPHY, ...
                'DataFormat','octets');
        end
    end
end
disp(status)
```

```
  Columns 1 through 7

    Success    Success    Success    Success    Success    Success    Success

  Column 8

    Success
```

Input Arguments

ampdu — A-MPDU to be deaggregated

binary-valued vector | vector of integers in the interval [0, 255] | string scalar | character array

A-MPDU to be deaggregated, specified as one of these values.

- A binary-valued vector representing the A-MPDU in bit format
- A vector of integers in the interval [0, 255] representing octets in decimal format
- A string scalar representing the A-MPDU as octets in hexadecimal format
- A character vector representing the A-MPDU as octets in hexadecimal format
- A character array, where each row represents an octet in hexadecimal format

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `char` | `string`

phyFormat — PHY format
'HE-SU' | 'HE-EXT-SU' | 'HE-MU | 'HE-TB' | 'VHT' | 'HT'

PHY format, specified as one of these values.

- 'HE-SU' — High-efficiency single-user (HE SU) format
- 'HE-EXT-SU' — HE extended-range SU (HE ER SU) format
- 'HE-MU' — HE multi-user (HE MU) format
- 'HE-TB' — HE trigger-based (HE TB) format
- 'VHT' — Very-high-throughput (VHT) format
- 'HT' — High-throughput (HT) format

Data Types: char | string

cfgPHY — PHY format and transmission parameters
wlanHESUConfig object | wlanHEMUConfig object | wlanHETBConfig object | wlanHERecoveryConfig object | wlanVHTConfig object | wlanHTConfig object

PHY format and transmission parameters, specified as one of these objects.

- wlanHESUConfig — HE SU or HE ER SU format
- wlanHEMUConfig — HE MU format
- wlanHETBConfig — HE TB format
- wlanHERecoveryConfig — Recovered HE transmission in HE SU, HE ER SU, or HE MU format
- wlanVHTConfig — VHT format
- wlanHTConfig — HT format

Name-Value Pair Arguments

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

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

Example: 'DataFormat','octets'

DataFormat — Format of input A-MPDU
'bits' (default) | 'octets'

Format of input A-MPDU, specified as the comma-separated pair consisting of 'DataFormat' and one of these values.

- 'bits' — Specify the ampdu input in bit format
- 'octets' — Specify the ampdu input in octet format

Data Types: char | string

SuppressWarnings — Suppress warning messages
false or 0 (default) | true or 1

Suppress warning messages, specified as the comma-separated pair consisting of `'SuppressWarnings'` and one of these values.

- `false` or `0` — Allow warning messages.
- `true` or `1` — Suppress warning messages.

Data Types: `logical`

Output Arguments

`mpduList` — List of MPDUs

cell array of character arrays

List of MPDUs, returned as a cell array of character arrays, where each character array corresponds to one MPDU. In these character arrays, each row is the hexadecimal representation of an octet.

If no MPDU delimiter is found in the input A-MPDU, the function returns `mpduList` as an empty cell array.

Data Types: `cell`

`failCRC` — Delimiter CRC failure indicator

row vector of logical values

Delimiter CRC failure indicator, returned as a row vector of logical values. Each element of this vector indicates the delimiter CRC failure status for an A-MPDU subframe.

A value of `1` for the *k*th element of this vector indicates that the delimiter CRC failed for the *k*th A-MPDU subframe. In this case, the *k*th element of `mpduList` contains an MPDU that might be invalid.

A value of `0` for the *k*th element of this vector indicates that the delimiter CRC passed for the *k*th subframe. In this case, the *k*th element of `mpduList` contains a valid MPDU.

Data Types: `logical`

`status` — Status of A-MPDU deaggregation

nonpositive integer

Status of A-MPDU deaggregation, returned as a nonpositive integer in the interval [–20, 0]. Each enumeration value of `status` corresponds to a member of the `wlanMACDecodeStatus` enumeration class, which indicates the status of MAC frame parsing according to this table.

Enumeration Value	Member of Enumeration Class	Decoding Status
`0`	`Success`	MAC frame successfully decoded
`-1`	`FCSFailed`	Frame check sequence (FCS) failed
`-2`	`InvalidProtocolVersion`	Invalid protocol version
`-3`	`UnsupportedFrameType`	Unsupported frame type
`-4`	`UnsupportedFrameSubtype`	Unsupported frame subtype

-5	NotEnoughData	Insufficient data to decode frame
-6	UnsupportedBAVariant	Unsupported variant of Block Ack frame
-7	UnknownBitmapSize	Unknown bitmap size
-8	UnknownAddressExtMode	Unknown address extension mode
-9	MalformedAMSDULength	Malformed aggregate MAC service data unit (A-MSDU) with invalid length
-10	MalformedSSID	Malformed service set identifier (SSID) information element (IE)
-11	MalformedSupportedRatesIE	Malformed supported rates IE
-12	MalformedIELength	Malformed IE length field
-13	MissingMandatoryIEs	Mandatory IEs missing
-14	NoMPDUFound	No MPDU found in A-MPDU
-15	CorruptedAMPDU	All the delimiters in received A-MPDU failed cyclic redundancy check (CRC)
-16	InvalidDelimiterLength	Invalid length field in MPDU delimiter
-17	MaxAMSDULenthExceeded	A-MSDU exceeds maximum length limit
-18	MaxMPDULengthExceeded	MPDU exceeds maximum length limit
-19	MaxMMPDULengthExceeded	MAC management frame exceeds maximum length limit
-20	MaxMSDULengthExceeded	MSDU exceeds maximum length limit
-21	UnexpectedProtectedFrame	Invalid value of protected bit for this frame type
-22	UnsupportedTriggerType	Unsupported trigger frame type
-23	UnknownHELTFTypeAndGI	Unknown guard interval (GI) and high-efficiency long training field (HE-LTF) type
-24	UnknownAPTxPower	Unknown value for AP Tx Power subfield of Common Info field
-25	UnknownAID12Value	Unknown value for AID12 subfield of User Info field
-26	UnknownRUAllocation	Unknown value for B7-B1 in RU Allocation subfield of User Info field

-27	UnknownULMCS	Unknown value for UL MCS subfield of User Info field
-28	UnknownTargetRSSI	Unknown value for UL Target RSSI subfield of User Info field
-29	UnsupportedBARType	Unsupported value for BAR Type subfield of BAR Control field
-30	MissingUserInfo	Received trigger frame contains invalid User Info field
-31	InvalidLSIGLength	Invalid value for UL Length subfield of Common Info field, corresponding to length of legacy signal (L-SIG) field.

An enumeration value other than `0` means that A-MPDU deaggregation stopped because the input A-MPDU is corrupt or malformed.

Data Types: `int16`

Version History

Introduced in R2019a

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`wlanMACFrame` | `wlanMPDUDecode`

Objects

`wlanMACFrameConfig` | `wlanMACManagementConfig`

Topics

“Generate and Parse WLAN MAC Frames”

wlanAPEPLength

Calculate APEP length in octets

Syntax

```
APEPLength = wlanAPEPLength(cfgPHY,unit,value)
```

Description

`APEPLength = wlanAPEPLength(cfgPHY,unit,value)` returns `APEPLength`, the aggregate MAC protocol data unit (A-MPDU) pre-EOF padding (APEP) length, in octets from the given `value` and the physical layer configuration, `cfgPHY`. The units of `value` can be in terms of physical layer conformance procedure (PLCP) protocol data unit (PPDU) transmission time or number of data symbols, specified by the `unit` input argument.

Examples

Generate VHT Waveform with Specified Transmission Time

Create a `wlanVHTConfig` object, `'cfgPHY'`, and specify the transmission time, `'txTime'`, in microseconds.

```
cfgPHY = wlanVHTConfig;
txTime = 200;
```

Calculate the APEP length in octets.

```
apepLength = wlanAPEPLength(cfgPHY,'TxTime',txTime)
```

```
apepLength = 580
```

Set the number of bytes carried in the user payload for the configuration object, `'cfgPHY'`, to this APEP length.

```
cfgPHY.APEPLength = apepLength;
```

Create a `wlanMACFrameConfig` object, `'cfgMAC'`. Use this object to generate a VHT-format QoS data frame.

```
cfgMAC = wlanMACFrameConfig('FrameType','QoS Data', ...
    'FrameFormat','VHT');
```

Calculate the MAC service data unit (MSDU) lengths required to generate a MAC frame of size `'APEPLength'`.

```
msduLengths = wlanMSDULengths(apepLength,cfgMAC,cfgPHY);
```

Create random MSDUs using `'msduLengths'`.

```
msduList = cell(1, numel(msduLengths));
for i = 1:numel(msduLengths)
```

```
    msduList{i} = randi([0 255],1,msduLengths(i));
end
```

Generate MAC frame bits using the MSDUs, `'msduList'`.

```
macFrameBits = wlanMACFrame(msduList,cfgMAC,cfgPHY,'OutputFormat','bits');
```

Generate a VHT waveform using `'cfgPHY'` and the generated MAC frame bits, `'macFrameBits'`.

```
waveform = wlanWaveformGenerator(macFrameBits,cfgPHY);
```

Generate HE-SU Waveform with Specified Number of Data Symbols

Create a `wlanHESUConfig` object, `'cfgPHY'`.

```
cfgPHY = wlanHESUConfig;
```

Calculate the APEP length in octets, specifying 200 data symbols.

```
apepLength = wlanAPEPLength(cfgPHY,'NumDataSymbols',200)
```

```
apepLength = 2916
```

Set the number of bytes carried in the user payload for the configuration object, `'cfgPHY'`, to this APEP length. Calculate the PSDU length.

```
cfgPHY.APEPLength = apepLength;
psduLength = getPSDULength(cfgPHY)*8;
```

Create a random PSDU, `'psdu'`, using the calculated PSDU length.

```
psdu = randi([0 1],getPSDULength(cfgPHY)*8,1);
```

Generate an HE-SU waveform using `'cfgPHY'` and `'psdu'`.

```
waveform = wlanWaveformGenerator(psdu,cfgPHY);
```

Input Arguments

`cfgPHY` — PHY format configuration

`wlanHESUConfig` object | `wlanVHTConfig` object

PHY format configuration, specified a `wlanHESUConfig` ,or `wlanVHTConfig` object. This object defines a PHY format configuration and its applicable properties.

`unit` — Units of argument `value`

`'TxTime'` | `'NumDataSymbols'`

Units of argument `value`, specified as one of `'TxTime'` or `'NumDataSymbols'`. This value indicates the units of `value` from which the APEP length is calculated.

Data Types: `char` | `string`

`value` — Value from which APEP length is calculated

numeric scalar

Value from which APEP length is calculated, specified as a numeric scalar. Input argument `unit` specifies the unit of `value`. This table describes how the function interprets `value` based on `unit`.

`unit` Value	`value` Description
`'TxTime'`	Scalar number specifying the time in microseconds
`'NumDataSymbols'`	Scalar number specifying the number of data symbols

Data Types: `double`

Output Arguments

`APEPLength` — Length of APEP

numeric scalar

Length of APEP, in octets, returned as a numeric scalar. This value returns the maximum APEP length that fits into the specified value of `'TxTime'` or `'NumDataSymbols'`.

Data Types: `double`

Version History

Introduced in R2019b

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`wlanMSDULengths` | `wlanPSDULength`

wlanBCCDecode

Convolutionally decode input data

Syntax

```
y = wlanBCCDecode(sym,rate)
y = wlanBCCDecode(sym,rate,decType)
y = wlanBCCDecode(sym,rate,tDepth)
y = wlanBCCDecode(sym,rate,decType,tDepth)
```

Description

`y = wlanBCCDecode(sym,rate)` returns decoded bits `y` by convolutionally decoding symbols `sym` the decoding rate specified by `rate`.

`y = wlanBCCDecode(sym,rate,decType)` specifies the decoding type of the Viterbi decoding algorithm.

`y = wlanBCCDecode(sym,rate,tDepth)` specifies the traceback depth of the Viterbi decoding algorithm.

`y = wlanBCCDecode(sym,rate,decType,tDepth)` specifies the decoding type and the traceback depth. You can specify the `decType` and `tDepth` inputs in any order after `rate`.

Examples

BCC-Decode Two Encoded Streams

Decode two encoded streams of soft bits by using a BCC of rate 1/2.

Create the sequence of data bits.

```
dataBits = randi([0 1],100,1,"int8");
```

Parse the data bits.

```
numES = 2;
parsedData = reshape(dataBits,numES,[]).';
```

BCC-encode the parsed sequence.

```
rate = "1/2"
```

```
rate =
"1/2"
```

```
encodedData = wlanBCCEncode(parsedData,rate);
```

Convert the encoded bits to soft bits.

```
demodData = double(1-2*encodedData);
```

BCC-decode the demodulated data.

```
decodedData = wlanBCCDecode(demodData,rate);
```

Deparse the decoded data.

```
deparsedData = reshape(decodedData.',[],1);
```

Verify that the decoded data matches the original data.

```
isequal(dataBits,deparsedData)
```

```
ans = logical
   1
```

BCC-Decode Soft Bits

Decode a sequence of soft bits by using a BCC of rate 3/4 and a traceback depth of 60.

Create the sequence of data bits.

```
dataBits = randi([0 1],300,1);
```

BCC-encode the sequence of bits.

```
encodedData = wlanBCCEncode(dataBits,3/4);
```

Convert the encoded bits to soft bits (i.e. LLR demodulation).

```
demodData = 1-2*encodedData;
```

BCC-decode the demodulated bits.

```
tDepth = 60;
decodedData = wlanBCCDecode(demodData,3/4,tDepth);
```

Verify that the decoded data matches the original data.

```
isequal(dataBits,decodedData)
```

```
ans = logical
   1
```

BCC-Decode Hard Bits

Decode a sequence of hard bits by using a BCC of rate 3/4 and a traceback depth of 45.

Create the sequence of data bits.

```
dataBits = randi([0 1],300,1,'int8');
```

BCC-encode the sequence of bits.

```
encodedData = wlanBCCEncode(dataBits,'2/3');
```

Perform hard BCC decoding on the encoded bits. Specify a traceback depth 45.

```
tDepth = 45;
decodedBits = wlanBCCDecode(encodedData,'2/3','hard',tDepth);
```

Verify that the decoded bits match the original bits.

```
isequal(dataBits,decodedBits)
```

```
ans = logical
   1
```

Input Arguments

`sym` — Symbols to decode

matrix of integers

Symbols to decode, specified as a matrix of integers. The number of columns must be the number of encoded streams. Each stream is encoded separately. When you specify the `decType` input as `'soft'`, this input must be a real matrix with log-likelihood ratios. Positive values represent a logical `0` and negative values represent a logical `1`.

Data Types: `double` | `int8`

`rate` — Code rate

numeric scalar | character vector | string scalar

Code rate of the binary convolutional code (BCC), specified as a numeric scalar, character vector, or string scalar. To select a code rate, specify this input as a value in accordance with the table.

Code Rate	Scalar	Character Vector	String
1/2	`1/2`	`'1/2'`	`"1/2"`
2/3	`2/3`	`'2/3'`	`"2/3"`
3/4	`3/4`	`'3/4'`	`"3/4"`
5/6	`5/6`	`'5/6'`	`"5/6"`

Example: `'3/4'`

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

`decType` — Decoding type

`'soft'` (default) | `'hard'`

Decoding type of the binary convolutional code (BCC), specified as a character vector or a string scalar. To specify a hard input Viterbi algorithm, specify this input as `'hard'`. To specify a soft input Viterbi algorithm without any quantization, specify this input as`'soft'`.

For more information on BCC, see sections 17.3.5.6 and 19.3.11.6 in [1].

Data Types: `char` | `string`

`tDepth` — Traceback depth
positive integer

Traceback depth of the Viterbi decoding algorithm, specified as a positive integer less than or equal to the number of input symbols in `sym`.

Data Types: `double`

Output Arguments

`y` — Binary convolutionally decoded bits
binary matrix

Binary convolutionally decoded bits, returned as a binary matrix. The number of rows of `y` is equal to the number of rows of input `sym` multiplied by `rate`, rounded to the next integer. The number of columns of `y` is equal to the number of columns of `sym`.

Data Types: `int8`

Version History

Introduced in R2017b

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

Extended Capabilities

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

See Also

`wlanBCCEncode` | `vitdec`

wlanBCCDeinterleave

Deinterleave binary convolutionally interleaved input

Syntax

```
y = wlanBCCDeinterleave(bits,type,numCBPSSI,cbw)
y = wlanBCCDeinterleave(bits,type,numCBPSSI)
```

Description

`y = wlanBCCDeinterleave(bits,type,numCBPSSI,cbw)` outputs the binary convolutionally deinterleaved input `bits` for a specified interleaver `type`, as defined in IEEE 802.11-2012 Section 18.3.5.7, IEEE 802.11ac-2013 Section 22.3.10.8, and IEEE 802.11ah Section 24.3.9.8. `numCBPSSI` specifies the number of coded bits per OFDM symbol per spatial stream per interleaver block and `cbw` specifies the channel bandwidth.

`y = wlanBCCDeinterleave(bits,type,numCBPSSI)` outputs the deinterleaved input `bits` for the non-HT interleaver `type`.

Examples

Interleave and Deinterleave VHT Data Field

Perform BCC interleaving and deinterleaving for the VHT interleaving type.

Define the input parameters. Set the number of coded bits per OFDM symbol per spatial stream per interleaver block to 52, the channel bandwidth to 20Mhz and the number of spatial streams, named as numSS, to 4.

```
numCBPSSI = 52;
chanBW = 'CBW20';
numSS = 4;
```

Create a sequence of bits for two OFDM symbols, four spatial streams, and one segment.

```
bits = randi([0 1],(2*numCBPSSI),numSS,1);
```

Perform BCC interleaving on the bits.

```
intBits = wlanBCCInterleave(bits,'VHT',numCBPSSI,chanBW);
```

Perform BCC deinterleaving on the interleaved bits.

```
out = wlanBCCDeinterleave(intBits,'VHT',numCBPSSI,chanBW);
```

Verify that the deinterleaved data matches the original data.

```
isequal(bits,out)
```

```
ans = logical
   1
```

Interleave and Deinterleave Non-HT Data Field

Perform BCC interleaving and deinterleaving for the non-HT interleaving type.

Define the input parameters. Set the number of coded bits per OFDM symbol per spatial stream per interleaver block to 48.

```
numCBPSSI = 48;
```

Create a sequence of random bits for one OFDM symbol, one spatial stream, and one segment.

```
bits = randi([0 1],numCBPSSI,1);
```

Perform BCC interleaving on the bits.

```
intBits = wlanBCCInterleave(bits,'Non-HT',numCBPSSI);
```

Perform BCC deinterleaving on the interleaved bits.

```
out = wlanBCCDeinterleave(intBits,'Non-HT',numCBPSSI);
```

Verify that the deinterleaved data matches the original data.

```
isequal(bits,out)
```

```
ans = logical
   1
```

Input Arguments

`bits` — Input sequence

matrix | 3-D array

Input sequence containing binary convolutionally interleaved data, specified as an (N_{CBPSSI}×N_{SYM})-by-N_{SS}-by-N_{SEG} array, where:

- N_{CBPSSI} is the number of coded bits per OFDM symbol per spatial stream per interleaver block.
- N_{SYM} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.
  - If `type= 'Non-HT'`, then N_{SS} must be 1.
  - If `type= 'VHT'`, then N_{SS} must be from 1 to 8.
- N_{SEG} is the number of segments.

Data Types: `double`

`type` — Type of interleaving

`'VHT'` | `'Non-HT'`

The type of interleaving, specified as `'VHT'` or `'Non-HT'`.

Data Types: `char` | `string`

`numCBPSSI` — Number of coded bits per OFDM symbol per spatial stream per interleaver block
positive integer

Number of coded bits per OFDM symbol per spatial stream per interleaver block specified as a positive integer. As defined in IEEE 802.11ac-2013 Table 22-6, the value of `numCBPSSI` depends on the interleaving type:

`'Non-HT'`	$N_{SD} \times N_{BPSCS}$
`'VHT'`	$N_{SD} \times N_{BPSCS} / N_{SEG}$

where:

- N_{SD} is the number of data subcarriers.
- N_{BPSCS} is the number of coded bits per subcarrier per spatial stream, specified as 1, 2, 4, 6, or 8.
- N_{SEG} is the number of segments.

When `type= 'Non-HT'`, `numCBPSSI` can be 48, 96, 192, 288, and 384, since $N_{CBPSSI} = 48 \times N_{BPSCS}$.

When `type= 'VHT'`, `numCBPSSI` can be 24, 48, 96, 144, and 192, since $N_{CBPSSI} = 24 \times N_{BPSCS}$.

Data Types: `double`

`cbw` — Channel bandwidth
`'CBW1'` | `'CBW2'` | `'CBW4'` | `'CBW8'` | `'CBW10'` | `'CBW16` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth in MHz, specified as `'CBW1'`,`'CBW2'`, `'CBW4'`,`'CBW8'`, `'CBW10'`, `'CBW16'`, `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`. When the interleaver type is set to `'Non-HT'`, then `cbw` is optional.

Data Types: `char` | `string`

Output Arguments

`y` — Deinterleaved output
matrix | 3-D array

Deinterleaved output, returned as an ($N_{CBPSSI} \times N_{SYM}$)-by-$N_{SS}$-by-$N_{SEG}$ array, where:

- N_{CBPSSI} is the number of coded bits per OFDM symbol per spatial stream per interleaver block.
- N_{SYM} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.
- N_{SEG} is the number of segments.

Version History

Introduced in R2017b

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanBCCInterleave` | `convdeintrlv`

wlanBCCEncode

Convolutionally encode binary data

Syntax

```
y = wlanBCCEncode(bits,rate)
```

Description

`y = wlanBCCEncode(bits,rate)` convolutionally encodes the binary input `bits` using a binary convolutional code (BCC) at the specified `rate`.

Examples

BCC-Encode Bits

Encode a sequence of data bits by using a BCC of rate 3/4.

Create the sequence of data bits.

```
dataBits = randi([0 1],300,1);
```

BCC-encode the data bits.

```
encodedData = wlanBCCEncode(dataBits,'3/4');
size(encodedData)
```

```
ans = 1×2

   400     1
```

BCC-Encode Two Streams

Encode two streams of data bits by using a BCC of rate 1/2.

Create the sequence of data bits.

```
dataBits = randi([0 1],100,1,"int8");
```

Parse the sequence of bits for the specified number of encoded streams.

```
numES = 2;
parsedData = reshape(dataBits,numES,[]).';
```

BCC-encode the parsed sequence.

```
rate = 1/2;
encodedData = wlanBCCEncode(parsedData,rate);
size(encodedData)
```

```
ans = 1×2

   100     2
```

Input Arguments

`bits` — Input sequence

matrix

Input sequence with data bits to encode, specified as a binary matrix. The number of columns must equal the number of encoded streams. Each stream is encoded separately.

For more information on BCC, see sections 17.3.5.6 and 19.3.11.6 in [1].

Data Types: `double` | `int8`

`rate` — Code rate

numeric scalar | character vector | string scalar

Code rate of the binary convolutional code (BCC), specified as a numeric scalar, character vector, or string scalar. To select a code rate, specify this input as a value in accordance with the table.

Code Rate	Scalar	Character Vector	String
1/2	`1/2`	`'1/2'`	`"1/2"`
2/3	`2/3`	`'2/3'`	`"2/3"`
3/4	`3/4`	`'3/4'`	`"3/4"`
5/6	`5/6`	`'5/6'`	`"5/6"`

Example: `'3/4'`

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

Output Arguments

`y` — Binary convolutionally encoded output

matrix

Binary convolutionally encoded output, returned as a binary matrix of the same type as the `bits` input. The number of rows of `y` is the result of dividing the number of rows of input `bits` by `rate`, rounded to the next integer. The number of columns of `y` is equal to the number of columns of `bits`.

Data Types: `double` | `int8`

Version History

Introduced in R2017b

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanBCCDecode` | `convenc`

wlanBCCInterleave

Interleave binary convolutionally encoded input

Syntax

```
y = wlanBCCInterleave(bits,type,numCBPSSI,cbw)
y = wlanBCCInterleave(bits,type,numCBPSSI)
```

Description

`y = wlanBCCInterleave(bits,type,numCBPSSI,cbw)` outputs the interleaved binary convolutionally encoded (BCC) input `bits` for a specified interleaver `type`, as defined in IEEE 802.11-2012 Section 18.3.5.7, IEEE 802.11ac-2013 Section 22.3.10.8, and IEEE 802.11ah Section 24.3.9.8. `numCBPSSI` specifies the number of coded bits per OFDM symbol per spatial stream per interleaver block and `cbw` specifies the channel bandwidth.

`y = wlanBCCInterleave(bits,type,numCBPSSI)` outputs the interleaved input `bits` for the non-HT interleaver `type`.

Examples

Interleave VHT Data Field

Perform BCC interleaving for the VHT interleaving type.

Define the input parameters. Set the number of coded bits per OFDM symbol per spatial stream per interleaver block to 52, the channel bandwidth to 20 MHz, and the number of spatial streams to 4.

```
numCBPSSI = 52;
cbw = 'CBW20';
numSS = 4;
```

Create a sequence of bits for two OFDM symbols, four spatial streams, and one segment.

```
inBits = randi([0 1],2*numCBPSSI,numSS,1,'int8');
```

Perform BCC interleaving on the bits.

```
out = wlanBCCInterleave(inBits,'VHT',numCBPSSI,cbw);
```

Interleave Non-HT Data Field

Perform BCC interleaving for the non-HT interleaving type.

Define the input parameters. Set the number of coded bits per OFDM symbol per spatial stream per interleaver block to 48.

```
numCBPSSI = 48;
```

Create a sequence of random bits for one OFDM symbol, one spatial stream, and one segment.

```
inBits = randi([0 1],numCBPSSI,1);
```

Perform BCC interleaving on the bits.

```
out = wlanBCCInterleave(inBits,'Non-HT',numCBPSSI);
```

Compare the original sequence with the interleaved one.

```
[inBits out]
```

```
ans = 48×2

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

Interleave Sequence

Get the interleaving sequence of a non-HT interleaver type.

Define the input parameters. Set the number of coded bits per OFDM symbol per spatial stream per interleaver block to 192.

```
numCBPSSI = 192;
```

Create a numeric sequence from 1 to `numCBPSSI`.

```
seq = (1:numCBPSSI).';
```

Perform BCC interleaving on the numeric sequence.

```
intSeq = wlanBCCInterleave(seq,'Non-HT',numCBPSSI);
intSeq(1:10)
```

```
ans = 10×1

     1
    17
    33
    49
    65
    81
    97
   113
   129
```

145

Input Arguments

`bits` — Input sequence

matrix | 3-D array

Input sequence containing binary convolutionally encoded (BCC) data, specified as an (N_{CBPSSI}×N_{SYM})-by-N_{SS}-by-N_{SEG} array, where:

- N_{CBPSSI} is the number of coded bits per OFDM symbol per spatial stream per interleaver block.
- N_{SYM} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.
  - If `type= 'Non-HT'`, then N_{SS} must be 1.
  - If `type= 'VHT'`, then N_{SS} must be from 1 to 8.
- N_{SEG} is the number of segments.

Data Types: `double` | `int8`

`type` — Type of interleaving

`'VHT'` | `'Non-HT'`

The type of interleaving, specified as `'VHT'` or `'Non-HT'`.

Data Types: `char` | `string`

`numCBPSSI` — Number of coded bits per OFDM symbol per spatial stream per interleaver block

positive integer

Number of coded bits per OFDM symbol per spatial stream per interleaver block specified as a positive integer. As defined in IEEE 802.11ac-2013 Table 22-6, the value of `numCBPSSI` depends on the interleaving type:

`'Non-HT'`	N_{SD}×N_{BPSCS}
`'VHT'`	N_{SD}×N_{BPSCS}/N_{SEG}

where:

- N_{SD} is the number of data subcarriers.
- N_{BPSCS} is the number of coded bits per subcarrier per spatial stream, specified as 1, 2, 4, 6, or 8.
- N_{SEG} is the number of segments.

When `type= 'Non-HT'`, `numCBPSSI` can be 48, 96, 192, 288, and 384, since $N_{CBPSSI} = 48 \times N_{BPSCS}$.

When `type= 'VHT'`, `numCBPSSI` can be 24, 48, 96, 144, and 192, since $N_{CBPSSI} = 24 \times N_{BPSCS}$.

Data Types: `double`

cbw — Channel bandwidth
`'CBW1'` | `'CBW2'` | `'CBW4'` | `'CBW8'` | `'CBW10'` | `'CBW16` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth in MHz, specified as `'CBW1'`,`'CBW2'`, `'CBW4'`,`'CBW8'`, `'CBW10'`, `'CBW16'`, `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`. When the interleaver type is set to `'Non-HT'`, then `cbw` is optional.

Data Types: `char` | `string`

Output Arguments

y — Interleaved output
matrix | 3-D array

Interleaved output, returned as an (N_{CBPSSI}×N_{SYM})-by-N_{SS}-by-N_{SEG} array, where:

- N_{CBPSSI} is the number of coded bits per OFDM symbol per spatial stream per interleaver block.
- N_{SYM} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.
- N_{SEG} is the number of segments.

Version History

Introduced in R2017b

Extended Capabilities

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

See Also

`wlanBCCDeinterleave` | `convintrlv`

wlanChannelFrequency

Determine channel center frequency

Syntax

```
fc = wlanChannelFrequency(channel,band)
```

Description

`fc = wlanChannelFrequency(channel,band)` calculates the center frequency of the WLAN channel number `channel` in the operating band `band`. For more information about IEEE 802.11 channel designations, see “WLAN Radio Frequency Channels”. For a detailed description of country-specific information, operating classes, and behavior limits, see Annex E of [1] and [2].

Examples

Calculate WLAN Channel Center Frequencies

Calculate the center frequency of channel 6 in the 2.4 GHz band.

```
channel = 6;
band = 2.4;
fc = wlanChannelFrequency(channel,band)
```

```
fc = 2.4370e+09
```

Calculate the center frequency of channels 37, 42, and 91 in the 6 GHz band.

```
channel = [37 42 91];
band = 6;
fc = wlanChannelFrequency(channel,band)
```

```
fc = 1×3
10^9 ×

    6.1350    6.1600    6.4050
```

Input Arguments

`channel` — Channel number

integer in the interval [1, 233] | array of integers in the interval [1, 233]

Channel number, specified as one of these values.

- An integer or array of integers in the interval [1, 14] when you specify the `band` input as `2.4`
- An integer or array of integers in the interval [1, 200] when you specify the `band` input as `5`
- An integer or array of integers in the interval [1, 233] when you specify the `band` input as `6`

- An array of integers in the interval [1,233] when you specify the `band` input as an array of equal dimensions with entries equal to `2.4`, `5`, or `6`

Data Types: `double`

band — Operating band
`2.4` | `5` | `6` | array with entries equal to `2.4`, `5`, or `6`

Operating band in GHz, specified as one of these values.

- One of the scalars `2.4`, `5`, or `6`
- An array of the scalars above when you specify the `channel` input as an array of equal dimensions with entries in the interval [1,233]

Data Types: `double`

Output Arguments

`fc` — Channel center frequency
positive scalar | array of positive values

Channel center frequency, in Hz, returned as one of these values.

- A positive scalar when you specify the `channel` and `band` inputs as scalars
- An array of positive values when you specify the `channel` input as an array and the `band` input as a scalar. The entries of this output are the center frequencies of the channel numbers specified in the corresponding entries of the `channel` input. The frequency for each channel number is calculated in the operating band specified in the `band` input
- An array of positive values when you specify the `channel` and `band` inputs as arrays with equal dimensions. The entries of this output are the center frequencies of the channel numbers specified in the corresponding entries of the `channel` input. The frequency for each channel number is calculated in the operating band specified in the corresponding entry of the `band` input

Data Types: `double`

Version History

Introduced in R2022a

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

[2] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

Topics

“WLAN Radio Frequency Channels”

wlanClosestReferenceSymbol

Find closest constellation points

Syntax

```
refSym = wlanClosestReferenceSymbol(sym,mod)
refSym = wlanClosestReferenceSymbol(sym,mod,phase)

refSym = wlanClosestReferenceSymbol(sym,cfg)
refSym = wlanClosestReferenceSymbol(sym,cfg,userNumber)
```

Description

`refSym = wlanClosestReferenceSymbol(sym,mod)` returns the constellation points closest to equalized symbols `sym` for modulation scheme `mod`.

`refSym = wlanClosestReferenceSymbol(sym,mod,phase)` returns the constellation points closest to the equalized symbols with counterclockwise rotation `phase`.

`refSym = wlanClosestReferenceSymbol(sym,cfg)` returns the constellation points closest to the equalized symbols for `cfg`, a configuration object that parameterizes a single-user (SU) transmission or a recovered high-efficiency (HE) transmission.

`refSym = wlanClosestReferenceSymbol(sym,cfg,userNumber)` returns the constellation points closest to the equalized symbols for the user specified by `userNumber` in a multiuser (MU) transmission and MU-format configuration object `cfg`.

Examples

Find Closest Reference Symbols for 64-QAM

Find the closest reference symbols to a received set of noisy symbols for 64-QAM.

Create a high-efficiency single-user-format (HE-SU-format) configuration object, specifying a modulation and coding scheme (MCS) with 64-QAM.

```
cfg = wlanHESUConfig('MCS',6);
```

Obtain the PSDU length.

```
psduLength = getPSDULength(cfg);
```

Generate a waveform for a payload of randomly generated bits and the specified configuration.

```
bits = randi([0 1],8*psduLength,1,'int8');
waveform = wlanWaveformGenerator(bits,cfg);
```

Generate noise to be added to the signal, specifying the signal-to-noise ratio (SNR).

```
SNR = 10;
rxWaveform = awgn(waveform,SNR);
```

Get the field indices and extract the HE-Data field.

```
ind =  wlanFieldIndices(cfg);
sym = rxWaveform(ind.HEData(1):ind.HEData(2));
```

Find the closest reference symbols for the specified modulation scheme.

```
mod = '64QAM';
refSym = wlanClosestReferenceSymbol(sym,mod);
```

Find Nearest Reference Symbols for DMG-Format Configuration

Find the closest reference symbols to a received set of noisy symbols in a DMG-format configuration.

Create a DMG-format configuration object, specifying the MCS.

```
cfg = wlanDMGConfig('MCS',10);
```

Generate a waveform for a payload of randomly generated bits and the specified configuration.

```
bits = randi([0 1],8*cfg.PSDULength,1,'int8');
waveform = wlanWaveformGenerator(bits,cfg);
```

Generate noise to be added to the signal, specifying the SNR.

```
SNR = 10;
rxWaveform = awgn(waveform,SNR);
```

Get the field indices and extract the DMG-Data field.

```
ind =  wlanFieldIndices(cfg);
sym = rxWaveform(ind.DMGData(1):ind.DMGData(2));
```

Find the closest reference symbols for the specified configuration.

```
refSym = wlanClosestReferenceSymbol(sym,cfg);
```

Input Arguments

`sym` — Equalized symbols

complex-valued array

Equalized symbols, specified as a complex-valued array.

Data Types: `double`
Complex Number Support: Yes

`mod` — Modulation scheme

`'BPSK'` | `'pi/2-BPSK'` | `'QPSK'` | `'pi/2-QPSK'` | `'16QAM'` | `'pi/2-16QAM'` | `'64QAM'` | `'pi/2-64QAM'` | `'256QAM'` | `'1024QAM'` | `'4096QAM'`

Modulation scheme, specified as one of these values:

- `'BPSK'` — Binary phase-shift keying (BPSK)

- `'pi/2-BPSK'` — π/2-BPSK
- `'QPSK'` — Quadrature phase-shift keying (QPSK)
- `'pi/2-QPSK'` — π/2-QPSK
- `'16QAM'` — 16-point quadrature amplitude modulation (16-QAM)
- `'pi/2-16QAM'` — π/2-16-QAM
- `'64QAM'` — 64-QAM
- `'pi/2-64QAM'` — π/2-64-QAM
- `'256QAM'` — 256-QAM
- `'1024QAM'` — 1024-QAM
- `'4096QAM'` — 4096-QAM

Data Types: `char` | `string`

`phase` — Counterclockwise rotation
real-valued scalar (default) | real-valued row vector

Counterclockwise rotation, in radians, specified as a real-valued scalar or real-valued row vector. To return reference symbols for different phases, specify `phase` as a row vector in which each element represents a chosen phase.

Note The rotations you specify in `phase` apply only to the constellation points returned in `refSym`, and not to the equalized symbols specified in `sym`.

Data Types: `double`

`cfg` — PHY format configuration
`wlanHESUConfig` object | `wlanHEMUConfig` object | `wlanHERecoveryConfig` object | `wlanHETBConfig` object | `wlanVHTConfig` object | `wlanHTConfig` object | `wlanNonHTConfig` object | `wlanDMGConfig` object | `wlanS1GConfig` object

Physical layer (PHY) format configuration, specified as one of these objects: `wlanHESUConfig`, `wlanHEMUConfig`, `wlanHERecoveryConfig`, `wlanHETBConfig`, `wlanVHTConfig`, `wlanHTConfig`, `wlanNonHTConfig`, `wlanDMGConfig`, or `wlanS1GConfig`.

`userNumber` — Number assigned to user of interest
positive integer

Number assigned to user of interest, specified as a positive integer in the interval [1, N_u], where N_u is the number users in the transmission.

This argument is required when you specify the `cfg` input as an object of type `wlanHEMUConfig`, `wlanS1GConfig`, or `wlanVHTConfig`.

If `cfg` is a `wlanHEMUConfig` object, N_u is equal to the number of elements in the value of its `User` property. If `cfg` is a `wlanS1GConfig` or `wlanVHTConfig` object, N_u is equal to the value of its `NumUsers`.

Dependencies

This argument applies only when the `cfg` input is an object of type `wlanHEMUConfig`, `wlanS1GConfig`, or `wlanVHTConfig`.

Data Types: `double`

Output Arguments

`refSym` — Constellation points closest to input symbols

complex-valued column vector

Constellation points closest to input symbols, returned as a complex-valued column vector. Each entry of `refSym` is the constellation point closest to the corresponding entry of the `sym` input; `refSym` is the same size as `sym`.

Data Types: `double`

Version History

Introduced in R2019a

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`wlanReferenceSymbols`

Objects

`comm.EVM`

wlanCoarseCFOEstimate

Perform coarse CFO estimation

Syntax

```
fOffset = wlanCoarseCFOEstimate(rxSig,cbw)
fOffset = wlanCoarseCFOEstimate(rxSig,cbw,corrOffset)
```

Description

`fOffset = wlanCoarseCFOEstimate(rxSig,cbw)` performs coarse carrier frequency offset (CFO) estimation on received time-domain "L-STF" on page 3-108[1] samples `rxSig` for channel bandwidth `cbw`. The function can estimate a maximum CFO of 625 kHz, or twice the subcarrier spacing.

`fOffset = wlanCoarseCFOEstimate(rxSig,cbw,corrOffset)` specifies the correlation offset as a fraction of a short training symbol.

Examples

Coarse Estimate of CFO for Non-HT Waveform

Create a non-HT configuration object.

```
nht = wlanNonHTConfig;
```

Generate a non-HT waveform.

```
txSig = wlanWaveformGenerator([1;0;0;1],nht);
```

Create a phase and frequency offset object and introduce a 2 kHz frequency offset.

```
pfOffset = comm.PhaseFrequencyOffset('SampleRate',20e6,'FrequencyOffset',2000);
rxSig = pfOffset(txSig);
```

Extract the L-STF.

```
ind = wlanFieldIndices(nht,'L-STF');
rxLSTF = rxSig(ind(1):ind(2),:);
```

Estimate the frequency offset from the L-STF.

```
freqOffsetEst = wlanCoarseCFOEstimate(rxLSTF,'CBW20')
```

```
freqOffsetEst = 2.0000e+03
```

1 IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.

Estimate and Correct CFO for VHT Waveform with Correlation Offset

Estimate the frequency offset for a VHT signal passing through a noisy, TGac channel. Correct for the frequency offset.

Create a VHT configuration object and create the L-STF.

```
vht = wlanVHTConfig;
txstf = wlanLSTF(vht);
```

Set the channel bandwidth and sample rate.

```
cbw = 'CBW80';
fs = 80e6;
```

Create TGac and thermal noise channel objects. Set the delay profile of the TGac channel to `'Model-C'`. Set the noise figure of the thermal noise channel to 9 dB.

```
tgacChan = wlanTGacChannel('SampleRate',fs,'ChannelBandwidth',cbw, ...
    'DelayProfile','Model-C','LargeScaleFadingEffect','Pathloss');

noise = comm.ThermalNoise('SampleRate',fs,'NoiseMethod','Noise figure', ...
    'NoiseFigure',9);
```

Pass the L-STF through the noisy TGac channel.

```
rxstfNoNoise = tgacChan(txstf);
rxstf = noise(rxstfNoNoise);
```

Create a phase and frequency offset object and introduce a 750 Hz frequency offset.

```
pfOffset = comm.PhaseFrequencyOffset('SampleRate',fs, ...
    'FrequencyOffsetSource','Input port');
rxstf = pfOffset(rxstf,750);
```

For the model-C delay profile, the RMS delay spread is 30 ns, which is 3/8 of the 80 ns short training symbol duration. As such, set the correlation offset to 0.375.

```
corrOffset = 0.375;
```

Estimate the frequency offset. Your results may differ slightly.

```
fOffsetEst = wlanCoarseCFOEstimate(rxstf,cbw,corrOffset)
```

```
fOffsetEst = 746.2700
```

The estimate is very close to the introduced CFO of 750 Hz.

Change the delay profile to `'Model-E'`, which has an RMS delay spread of 100 ns.

```
release(tgacChan)
tgacChan.DelayProfile = 'Model-E';
```

Pass the transmitted signal through the modified channel and apply the 750 Hz CFO.

```
rxstfNoNoise = tgacChan(txstf);
rxstf = noise(rxstfNoNoise);
rxstf = pfOffset(rxstf,750);
```

Estimate the frequency offset.

```
fOffsetEst = wlanCoarseCFOEstimate(rxstf,cbw,corrOffset)
```

```
fOffsetEst = 947.7234
```

The estimate is inaccurate because the RMS delay spread is greater than the duration of the training symbol.

Set the correlation offset to the maximum value of 1 and estimate the CFO.

```
corrOffset = 1;
fOffsetEst = wlanCoarseCFOEstimate(rxstf,cbw,corrOffset)
```

```
fOffsetEst = 745.3640
```

The estimate is accurate because the autocorrelation does not use the first training symbol. The channel delay renders this symbol useless.

Correct for the estimated frequency offset.

```
rxstfCorrected = pfOffset(rxstf,-fOffsetEst);
```

Estimate the frequency offset of the corrected signal.

```
fOffsetEstCorr = wlanCoarseCFOEstimate(rxstfCorrected,cbw,corrOffset)
```

```
fOffsetEstCorr = 2.7402e-11
```

The corrected signal has negligible frequency offset.

Two-Step CFO Estimation and Correction

Estimate and correct for a significant carrier frequency offset in two steps. Estimate the frequency offset after all corrections have been made.

Set the channel bandwidth and the corresponding sample rate.

```
cbw = 'CBW40';
fs = 40e6;
```

Coarse Frequency Correction

Generate an HT format configuration object.

```
cfg = wlanHTConfig('ChannelBandwidth',cbw);
```

Generate the transmit waveform.

```
txSig = wlanWaveformGenerator([1;0;0;1],cfg);
```

Create TGn and thermal noise channel objects. Set the noise figure of the receiver to 9 dB.

```
tgnChan = wlanTGnChannel('SampleRate',fs,'DelayProfile','Model-D', ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
noise = comm.ThermalNoise('SampleRate',fs, ...
```

```
    'NoiseMethod','Noise figure', ...
    'NoiseFigure',9);
```

Pass the waveform through the TGn channel and add noise.

```
rxSigNoNoise = tgnChan(txSig);
rxSig = noise(rxSigNoNoise);
```

Create a phase and frequency offset object to introduce a carrier frequency offset. Introduce a 2 kHz frequency offset.

```
pfOffset = comm.PhaseFrequencyOffset('SampleRate',fs,'FrequencyOffsetSource','Input port');
rxSig = pfOffset(rxSig,2e3);
```

Extract the L-STF signal for coarse frequency offset estimation.

```
istf = wlanFieldIndices(cfg,'L-STF');
rxstf = rxSig(istf(1):istf(2),:);
```

Perform a coarse estimate of the frequency offset. Your results may differ.

```
foffset1 = wlanCoarseCFOEstimate(rxstf,cbw)
```

```
foffset1 = 2.0221e+03
```

Correct for the estimated offset.

```
rxSigCorr1 = pfOffset(rxSig,-foffset1);
```

Fine Frequency Correction

Extract the L-LTF signal for fine offset estimation.

```
iltf = wlanFieldIndices(cfg,'L-LTF');
rxltf1 = rxSigCorr1(iltf(1):iltf(2),:);
```

Perform a fine estimate of the corrected signal.

```
foffset2 = wlanFineCFOEstimate(rxltf1,cbw)
```

```
foffset2 = -11.0795
```

The corrected signal offset is reduced from 2000 Hz to approximately 7 Hz.

Correct for the remaining offset.

```
rxSigCorr2 = pfOffset(rxSigCorr1,-foffset2);
```

Determine the frequency offset of the twice corrected signal.

```
rxltf2 = rxSigCorr2(iltf(1):iltf(2),:);
deltaFreq = wlanFineCFOEstimate(rxltf2,cbw)
```

```
deltaFreq = -2.0374e-11
```

The CFO is zero.

Input Arguments

`rxSig` — Received L-STF samples

complex-valued matrix

Received L-STF samples, specified as a complex-valued matrix of size N_S-by-N_R. N_S is the number of samples in the L-STF and N_R is the number of receive antennas.

Note If the number of samples in this input is greater than the number of samples in the L-STF, the function estimates the CFO by using only the first N_S samples.

Data Types: `single` | `double`

`cbw` — Channel bandwidth

`'CBW5'` | `'CBW10'` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'` | `'CBW320'`

Channel bandwidth, specified as one of these values.

- `'CBW5'` – Channel bandwidth of 5 MHz
- `'CBW10'` – Channel bandwidth of 10 MHz
- `'CBW20'` – Channel bandwidth of 20 MHz
- `'CBW40'` – Channel bandwidth of 40 MHz
- `'CBW80'` – Channel bandwidth of 80 MHz
- `'CBW160'` – Channel bandwidth of 160 MHz
- `'CBW320'` – Channel bandwidth of 320 MHz

Data Types: `char` | `string`

`corrOffset` — Correlation offset

`0.75` (default) | scalar in the interval [0, 1]

Correlation offset as a fraction of a short training symbol, specified as a scalar in the interval [0, 1]. The duration of the short training symbol varies with bandwidth. For more information, see “L-STF” on page 3-108.

Data Types: `single` | `double`

Output Arguments

`fOffset` — Frequency offset

real-valued scalar

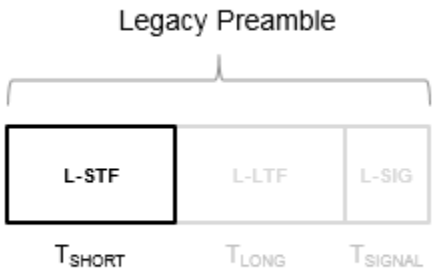
Frequency offset, in Hz, returned as a real-valued scalar. The function can estimate a maximum CFO of 625 kHz, or twice the subcarrier spacing.

Data Types: `double`

More About

L-STF

The legacy short training field (L-STF) is the first field of the 802.11 OFDM PLCP legacy preamble. The L-STF is a component of VHT, HT, and non-HT PPDUs.



The L-STF duration varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period ($T_{FFT} = 1 / \Delta_F$)	L-STF Duration ($T_{SHORT} = 10 \times T_{FFT} / 4$)
20, 40, 80, 160, and 320	312.5	3.2 µs	8 µs
10	156.25	6.4 µs	16 µs
5	78.125	12.8 µs	32 µs

Because the sequence has good correlation properties, it is used for start-of-packet detection, for coarse frequency correction, and for setting the AGC. The sequence uses 12 of the 52 subcarriers that are available per 20 MHz channel bandwidth segment. For 5 MHz, 10 MHz, and 20 MHz bandwidths, the number of channel bandwidths segments is 1.

Version History

Introduced in R2015b

References

[1] IEEE Std 802.11™-2016 (Revision of IEEE Std 802.11-2012). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements.

[2] Li, Jian. “Carrier Frequency Offset Estimation for OFDM-Based WLANs.” *IEEE Signal Processing Letters*. Vol. 8, Issue 3, Mar 2001, pp. 80–82.

[3] Moose, P. H. “A technique for orthogonal frequency division multiplexing frequency offset correction.” *IEEE Transactions on Communications*. Vol. 42, Issue 10, Oct 1994, pp. 2908–2914.

[4] Perahia, E. and R. Stacey. *Next Generation Wireless LANs: 802.11n and 802.11ac*. 2nd Edition. United Kingdom: Cambridge University Press, 2013.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanFineCFOEstimate` | `comm.PhaseFrequencyOffset` | `wlanLSTF`

wlanConstellationDemap

Constellation demapping

Syntax

```
y = wlanConstellationDemap(sym,noiseVarEst,numBPSCS)
y = wlanConstellationDemap(sym,noiseVarEst,numBPSCS,demapType)
y = wlanConstellationDemap(sym,noiseVarEst,numBPSCS,phase)
y = wlanConstellationDemap(sym,noiseVarEst,numBPSCS,demapType,phase)
```

Description

`y = wlanConstellationDemap(sym,noiseVarEst,numBPSCS)` demaps received symbols `sym` using the soft-decision approximate log-likelihood-ratio (LLR) method for `numBPSCS`, the number of coded bits per subcarrier per spatial stream.

`y = wlanConstellationDemap(sym,noiseVarEst,numBPSCS,demapType)` also specifies the demapping type.

`y = wlanConstellationDemap(sym,noiseVarEst,numBPSCS,phase)` derotates the symbols clockwise before demapping by the number of radians specified in `phase`.

`y = wlanConstellationDemap(sym,noiseVarEst,numBPSCS,demapType,phase)` specifies the demapping type and the phase rotation.

Examples

4096-QAM Demapping

Perform a 4096-QAM demapping on a sequence of data bits.

Create the sequence of data bits.

```
bits = randi([0 1],49152,1,'int8');
```

Perform constellation mapping on the data bits by using 4096-QAM.

```
numBPSCS = 12;
sym = wlanConstellationMap(bits,numBPSCS);
size(sym)
```

```
ans = 1×2

        4096           1
```

Perform 4096-QAM constellation demapping. Because the default demapping type is soft, the output is a vector of soft bits.

```
noiseVarEst = 0;
y = wlanConstellationDemap(sym,noiseVarEst,numBPSCS);
size(y)
```

```
ans = 1×2

       49152           1
```

Constellation Demapping with Hard Demodulation

Perform a 256-QAM demapping by using hard demodulation.

Create the sequence of data bits.

```
 bits = randi([0 1],416,1);
```

Perform the constellation mapping on the data bits by using a 256-QAM constellation.

```
numBPSCS = 8;
sym = wlanConstellationMap(bits,numBPSCS);
```

Perform the hard 256-QAM constellation demapping.

```
noiseVarEst = 0;
demapType = 'hard';
y = wlanConstellationDemap(sym,noiseVarEst,numBPSCS,demapType);
```

Verify that the demapped data matches the original data.

```
isequal(bits,y)
```

```
ans = logical
   1
```

BPSK and QBPSK Demapping for VHT-SIG-A Field

BPSK and QBPSK demapping for different OFDM symbols for the VHT-SIG-A field by using a soft demodulation. The demapping is defined in IEEE® 802.11ac™-2013 Section 22.3.8.3.3

Create the sequence of data bits. Specify the two OFDM symbols in columns.

```
 bits = randi([0 1],48,2,'int8');
```

Perform constellation mapping on the data bits. Specify the size of the constellation rotation as the number in columns of the input sequence. The first column is mapped with a BPSK modulation. The second column is modulated with a QBPSK modulation.

```
numBPSCS = 1;
phase = [0 pi/2];
mappedData = wlanConstellationMap(bits,numBPSCS,phase);
```

Perform the constellation demapping with an estimated variance noise equal to zero (no added noise). To derotate the constellation, specify the same phase as in the mapping function. The output is a vector of soft bits ready to be the input of a convolutional decoder.

```
noiseVar = 0;
demappedData = wlanConstellationDemap(mappedData,noiseVar,numBPSCS,phase);
```

Verify that the demapped data matches the original data. Because no noise is present, you can recover the original data without errors by assigning the negative values to a logical 1 and the positive values to a logical 0. In other words, you can convert the soft bits into hard bits.

```
demappedBits = int8((demappedData<=0));
isequal(bits,demappedBits)
```

```
ans = logical
   1
```

Demap 4-D Array

Perform BPSK demapping on a four-dimensional array by using hard demodulation.

Create the sequence of data bits as an array of four dimensions, with 416 coded bits per subcarrier per spatial stream per interleaver block, four OFDM symbols, two spatial streams, and two segments.

```
numCBPSSI = 416;
numSym = 4;
numSS = 2;
numSeg = 2;
bits = randi([0 1],numCBPSSI,numSym,numSS,numSeg);
size(bits)
```

```
ans = 1×4

   416     4     2     2
```

Perform BPSK constellation mapping on the data bits with a rotation of $\frac{\pi}{2}$ radians.

```
numBPSCS = 1;
phase = pi/2;
sym = wlanConstellationMap(bits,numBPSCS,phase);
size(sym)
```

```
ans = 1×4

   416     4     2     2
```

Perform hard QBPSK constellation demapping. To derotate the constellation, specify the same phase as in the mapping function.

```
noiseVarEst = 0;
demapType = 'hard';
y = wlanConstellationDemap(sym,noiseVarEst,numBPSCS,demapType);
```

Verify that the demapped data matches the original data.

```
isequal(bits,y)
```

```
ans = logical
   1
```

Input Arguments

`sym` — Input sequence

vector | matrix | multidimensional array

Received symbols, specified as a complex-valued vector, matrix, or multidimensional array. If you specify this input as a matrix or array, the function performs constellation demapping column-wise.

Data Types: `double`
Complex Number Support: Yes

`noiseVarEst` — Noise variance estimate

nonnegative scalar

Noise variance estimate, specified as a nonnegative scalar. When you specify the `demapType` input as `'hard'`, the function does not use this input.

Example: `0.7071`

Data Types: `double`

`numBPSCS` — Number of coded bits per subcarrier per spatial stream

`1` | `2` | `4` | `6` | `8` | `10` | `12`

Number of coded bits per subcarrier per spatial stream, specified as log2(M), where M is the modulation order. Therefore, `numBPSCS` must be one of these values.

- `1` for binary phase-shift keying (BPSK) modulation, as specified in section 17.3.5.8 of [1]
- `2` for quadrature phase-shift keying (QPSK) modulation, as specified in section 17.3.5.8 of [1]
- `4` for 16-point quadrature amplitude modulation (16-QAM), as specified in section 17.3.5.8 of [1]
- `6` for 64-QAM, as specified in section 17.3.5.8 of [1]
- `8` for 256-QAM, as specified in section 21.3.12.9 of [2]
- `10` for 1024-QAM, as specified in section 27.3.12.9 of [2]
- `12` for 4096-QAM, as specified in section 36.3.12.8 of [3]

Data Types: `double`

`demapType` — Demapping type

`'soft'` (default) | `'hard'`

Demapping type, specified as one of these values.

- `'hard'` — hard-decision demapping
- `'soft'` — soft-decision approximate LLR method

Data Types: `double`

phase — Constellation rotation
scalar | vector | multidimensional array

Constellation rotation, in radians, specified as a scalar, vector, or multidimensional array. The size of this input must be compatible with the size of the `sym` input. This input and `sym` have compatible sizes if, for each corresponding dimension, the dimension sizes are either equal or one of them is 1. When one of the dimensions of `sym` is equal to 1 and the corresponding dimension of `phase` is larger than 1, then the output dimensions have the same size as the dimensions of `phase`.

Example: `pi*(0:size(bits,1)/numBPSCS-1).'/2;`

Data Types: `double`

Output Arguments

y — Demapped symbols
vector | matrix | multidimensional array

Demapped symbols, returned as an integer-valued vector, matrix, or multidimensional array . This output has the same size as `sym` except for the number of rows, which is equal to the number of rows of `sym` multiplied by `numBPSCS`.

Version History

Introduced in R2017b

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

[2] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

[3] IEEE P802.11be™/D1.2. “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 8: Enhancements for Extremely High Throughput (EHT).” Draft Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

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

See Also

`wlanConstellationMap`

wlanConstellationMap

Constellation mapping

Syntax

```
y = wlanConstellationMap(bits,numBPSCS)
y = wlanConstellationMap(bits,numBPSCS,phase)
```

Description

`y = wlanConstellationMap(bits,numBPSCS)` maps input sequence `bits` using `numBPSCS`, the number of coded bits per subcarrier per spatial stream.

`y = wlanConstellationMap(bits,numBPSCS,phase)` rotates the constellation points counterclockwise by the number of radians specified in `phase`.

Examples

4096-QAM Mapping

Perform a 4096-QAM mapping.

Create the sequence of data bits.

```
bits = randi([0 1],49152,1,'int8');
```

Perform the constellation mapping on the data bits with 4096-QAM.

```
numBPSCS = 12;
y = wlanConstellationMap(bits,numBPSCS);
```

The size of the output returned by this modulation is the size of the input sequence divided by 12.

```
size(y)
```

```
ans = 1×2

        4096           1
```

π/2-BPSK Mapping

Perform $\frac{\pi}{2}$-BPSK mapping on a sequence of data bits.

Create the sequence of data bits.

```
bits = randi([0 1],512,1);
```

Perform the BPSK mapping on the data bits with a rotation of $\frac{\pi}{2}$ radians.

```
numBPSCS = 1;
phase = pi*(0:size(bits,1)/numBPSCS-1).'/2;
y = wlanConstellationMap(bits,numBPSCS,phase);
```

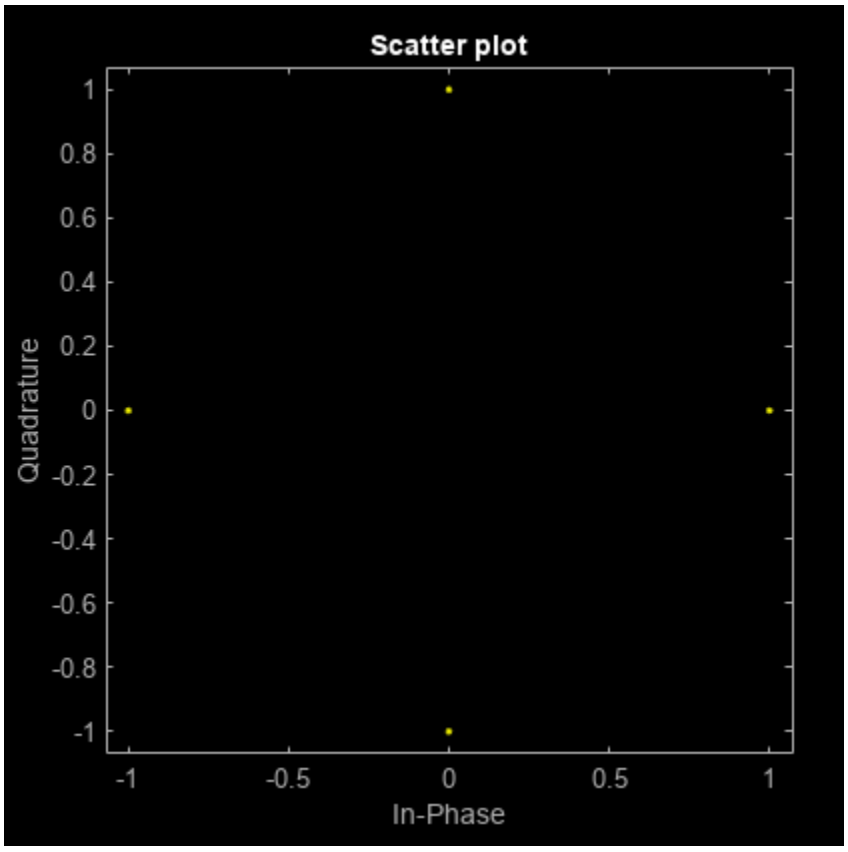
The size of the output is equal to the size of the original sequence.

```
size(y)
```

```
ans = 1×2

   512     1
```

Display the modulated signal constellation.

```
scatterplot(y)
```



BPSK Mapping for VHT-SIG-A field

Perform BPSK mapping of OFDM symbols for the VHT-SIG-A field.

Create the sequence of data bits. Place the two OFDM symbols in columns.

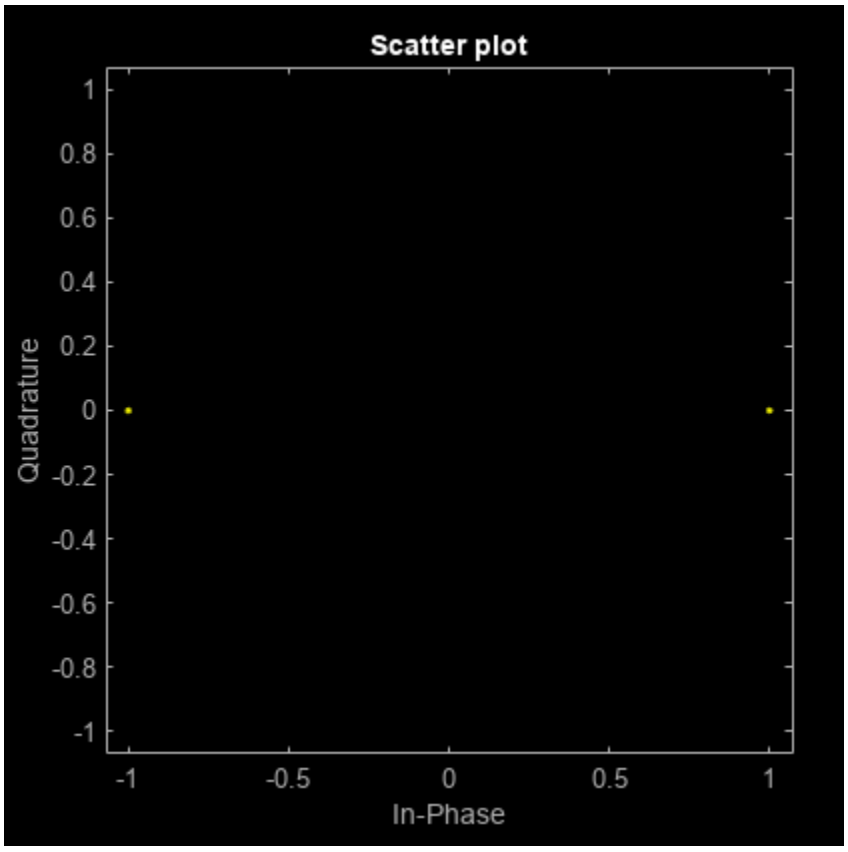
```
bits = randi([0 1],48,2,'int8');
```

Perform constellation mapping on the data bits. Specify the size of constellation rotation `phase` as the number of columns in the input sequence.

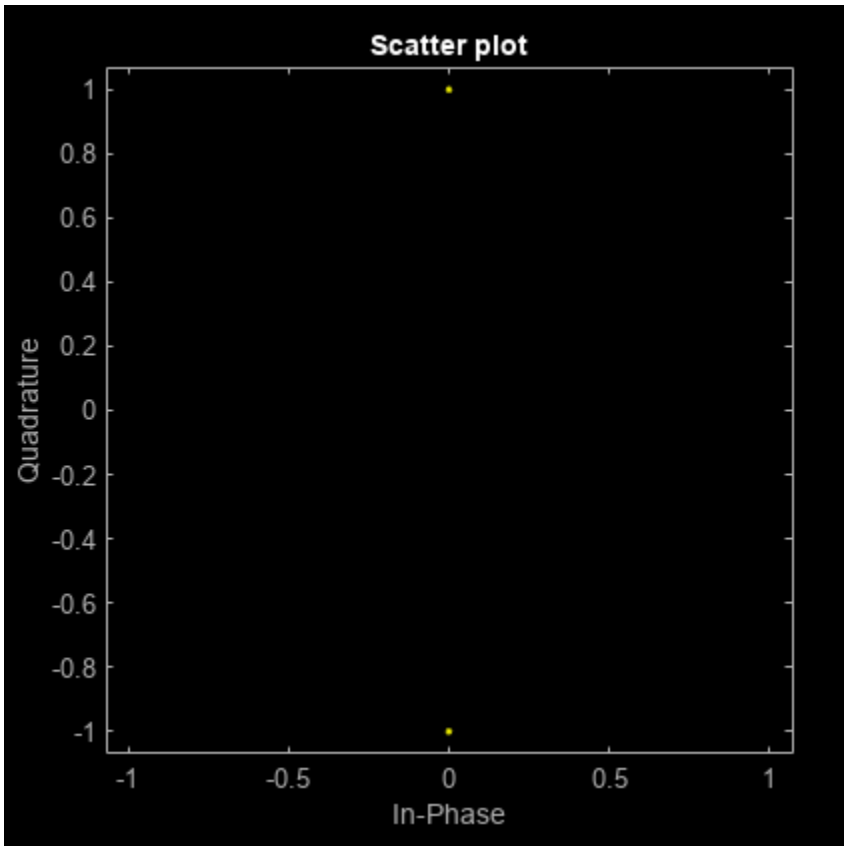
```
numBPSCS = 1;
phase = [0 pi/2];
y = wlanConstellationMap(bits,numBPSCS,phase);
```

Display the modulated signal constellation.

```
scatterplot(y(:,1))
```




```
scatterplot(y(:,2))
```



Input Arguments

`bits` — Input sequence

vector | matrix | multidimensional array

Input sequence of bits to map into symbols, specified as a binary-valued vector, matrix, or multidimensional array.

Data Types: `double` | `int8`

`numBPSCS` — Number of coded bits per subcarrier per spatial stream

`1` | `2` | `4` | `6` | `8` | `10` | `12`

Number of coded bits per subcarrier per spatial stream, specified as log2(*M*), where *M* is the modulation order. Therefore, `numBPSCS` must be one of these values.

- `1` for binary phase-shift keying (BPSK) modulation, as specified in section 17.3.5.8 of [1]
- `2` for quadrature phase-shift keying (QPSK) modulation, as specified in section 17.3.5.8 of [1]
- `4` for 16-point quadrature amplitude modulation (16-QAM), as specified in section 17.3.5.8 of [1]
- `6` for 64-QAM, as specified in section 17.3.5.8 of [1]
- `8` for 256-QAM, as specified in section 21.3.12.9 of [2]
- `10` for 1024-QAM, as specified in section 27.3.12.9 of [2]

- `12` for 4096-QAM, as specified in section 36.3.12.8 of [3]

Example: `4`

Data Types: `double`

phase — Constellation rotation
scalar | vector | multidimensional array

Constellation rotation in radians, specified as a scalar, vector, or multidimensional array. The size of `phase` must be compatible with the size of the `bits` input. This input and `bits` have compatible sizes if, for each corresponding dimension, the dimension sizes are either equal or one of them is 1. When one of the dimensions of `bits` is equal to 1 and the corresponding dimension of `phase` is larger than 1, then the output dimensions have the same size as the dimensions of `phase`.

Example: `pi*(0:size(bits,1)/numBPSCS-1).'/2;`

Data Types: `double`

Output Arguments

y — Mapped symbols
vector | matrix | multidimensional array

Mapped symbols, returned as a complex-valued vector, matrix, or multidimensional array. This output has the same size as `bits`, except for the number of rows, which is equal to the number of rows of `bits` divided by `numBPSCS`.

Data Types: `double`
Complex Number Support: Yes

Version History

Introduced in R2017b

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

[2] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

[3] IEEE P802.11be/D1.2. “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 8: Enhancements for Extremely High Throughput (EHT).” Draft Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanConstellationDemap`

wlanDMGDataBitRecover

Recover bits from DMG Data field

Syntax

```
dataBits = wlanDMGDataBitRecover(rxData,noiseVarEst,cfgDMG)
dataBits = wlanDMGDataBitRecover(rxData,noiseVarEst,csi,cfgDMG)
dataBits = wlanDMGDataBitRecover( ___ ,Name,Value)
```

Description

`dataBits = wlanDMGDataBitRecover(rxData,noiseVarEst,cfgDMG)` recovers `dataBits`, a column vector of bits, from `rxData`, the DMG Data field of a directional multi-gigabit (DMG) transmission. The function recovers `dataBits` by using noise variance estimate `noiseVarEst` and DMG transmission parameters `cfgDMG`.

`dataBits = wlanDMGDataBitRecover(rxData,noiseVarEst,csi,cfgDMG)` enhances the demapping of OFDM subcarriers by using channel state information `csi`. Use this syntax for DMG transmissions that use an orthogonal frequency-division multiplexing (OFDM) PHY configuration.

`dataBits = wlanDMGDataBitRecover( ___ ,Name,Value)` specifies algorithm options by using one or more name-value pair arguments, in addition to any input argument combination from previous syntaxes. For example, `'LDPCDecodingMethod','layered-bp'` specifies the layered belief propagation low-density parity-check (LDPC) decoding algorithm.

Examples

Recover DMG Data Field from Control Transmission

Recover bits from the DMG Data field in a control transmission.

Create a DMG configuration object with a modulation and coding scheme (MCS) for a control PHY configuration.

```
cfgDMG = wlanDMGConfig('MCS',0);
```

Create a sequence of data bits and generate a DMG waveform.

```
bits = randi([0 1],cfgDMG.PSDULength*8,1,'int8');
waveform = wlanWaveformGenerator(bits,cfgDMG);
```

Pass the waveform through a noiseless channel.

```
noiseVarEst = 0;
```

Extract the DMG Header and the DMG Data fields by using the `wlanFieldIndices` function.

```
ind = wlanFieldIndices(cfgDMG);
rxSym = waveform(ind.DMGHeader(1):ind.DMGData(2));
```

Rotate the received signal by 90 degrees.

```
rxSymRotated = rxSym.*exp(-1i*(pi/2)*(0:size(rxSym,1) - 1).');
```

Generate a Golay sequence of length 32 by using the `wlanGolaySequence` function.

```
len = 32;
Ga = wlanGolaySequence(len);
```

Despread the signal with a factor equal to the Golay sequence length

```
rxData = (reshape(rxSymRotated,len,length(rxSymRotated)/len)'*Ga)/len;
```

Recover the PSDU from the DMG Data field.

```
dataBits = wlanDMGDataBitRecover(rxData,noiseVarEst,cfgDMG);
```

Confirm that the decoded bits match the transmitted bits.

```
disp(isequal(bits,dataBits))
```

```
   1
```

Recover DMG Data Field from OFDM Transmission

Recover bits from the DMG Data field of an OFDM transmission.

Configure an OFDM transmission by creating a DMG configuration object with an MCS of 14.

```
cfgDMG = wlanDMGConfig('MCS',14);
```

Create a sequence of data bits and generate a DMG waveform.

```
bits = randi([0 1],8*cfgDMG.PSDULength,1,'int8');
waveform = wlanWaveformGenerator(bits,cfgDMG);
```

Pass the waveform through a channel, assuming additive white Gaussian noise (AWGN) for the specified signal-to-noise ratio (SNR).

```
snr = 10;                       % SNR, in dB
noiseVarEst = 10^(-snr/10);     % Noise variance
rxSig = awgn(waveform,snr);
```

Extract the DMG Data field from the received signal.

```
ind = wlanFieldIndices(cfgDMG);
rxSym = rxSig(ind.DMGData(1):ind.DMGData(2));
```

Perform OFDM demodulation on the received waveform and extract the data subcarriers.

```
demod = wlanDMGOFDMDemodulate(rxSym);
info = wlanDMGOFDMInfo;
rxData = demod(info.DataIndices,:);
```

Recover the PSDU from the DMG Data field, assuming a CSI estimate of all ones.

```
csi = ones(length(info.DataIndices),1);
dataBits = wlanDMGDataBitRecover(rxData,noiseVarEst,csi,cfgDMG);
```

Confirm that the decoded bits match the transmitted bits.

```
disp(isequal(bits,dataBits))
```

```
   1
```

Recover DMG Data Field from SC Transmission

Recover bits from the DMG Data field of a single-carrier (SC) transmission.

Configure an SC transmission by creating a DMG configuration object with an MCS of `10`.

```
cfgDMG = wlanDMGConfig('MCS',10);
```

Create a sequence of data bits and generate a DMG waveform.

```
bits = randi([0 1],8*cfgDMG.PSDULength,1,'int8');
waveform = wlanWaveformGenerator(bits,cfgDMG);
```

Pass the waveform through a channel, assuming AWGN with an SNR of 10 dB.

```
snr = 10;                       % SNR, in dB
noiseVarEst = 10^(-snr/10);     % Noise variance
rxSig = awgn(waveform,snr);
```

Extract the DMG Data field from the received signal.

```
ind = wlanFieldIndices(cfgDMG);
rxSym = rxSig(ind.DMGData(1):ind.DMGData(2));
```

Reshape the received data waveform into blocks. Set the data block size to 512 and the guard interval (GI) length to 64. Remove the last GI from the received waveform. The resulting waveform is a 512-by-`Nblks` matrix, where `Nblks` is the number of DMG data blocks.

```
blkSize = 512;                                  % Block size
Ngi = 64;                                       % GI length
rxSymNoGI = rxSym(1:end-Ngi);                   % Remove GI
rxSymReshaped = reshape(rxSymNoGI,blkSize,[]);  % Reshape received data
```

Remove the GI from each block. Confirm that the resulting signal is a 448-by-`Nblks` matrix, as expected for the time-domain DMG Data field signal in an SC PHY configuration.

```
rxData = rxSymReshaped(Ngi+1:end,:);
disp(size(rxData))
```

```
   448     9
```

Recover the PSDU from the DMG Data field, specifying layered belief propagation LDPC decoding.

```
dataBits = wlanDMGDataBitRecover(rxData,noiseVarEst,cfgDMG,'LDPCDecodingMethod','norm-min-sum');
```

Confirm that the decoded bits match the original information bits.

```
disp(isequal(bits,dataBits))
```

```
   1
```

Input Arguments

`rxData` — Received DMG Data field

column vector | matrix

Received DMG Data field, specified as a column vector or matrix. The contents and size of this input depend on the PHY configuration you specify in the `cfgDMG` input.

- SC PHY — This input contains the time-domain DMG Data field signal in a 448-by-N_{blks} matrix. The value 448 is the number of symbols in a DMG Data block, and N_{blks} is the number of DMG Data blocks. For more information about block transmission, see section 21.6.3.2.5 of [1]
- OFDM PHY — This input contains the demodulated DMG Data field OFDM symbols in a 336-by-N_{sym} matrix. The value 336 is the number of data subcarriers in the DMG Data field and N_{sym} is the number of OFDM symbols.
- Control PHY — This input contains the time-domain DMG Header and DMG Data fields in a column vector of length N_b, where N_b is the number of despread symbols.

Data Types: `double`
Complex Number Support: Yes

`noiseVarEst` — Noise variance estimate

nonnegative scalar

Noise variance estimate, specified as a nonnegative scalar.

Data Types: `double`

`cfgDMG` — DMG transmission configuration

`wlanDMGConfig` object

DMG transmission configuration, specified as a `wlanDMGConfig` object.

`csi` — Channel state information

real-valued column vector

Channel state information, specified as a real-valued column vector of length 336. The value 336 specifies the number of data subcarriers in the DMG Data field.

Dependencies

To enable this input, specify an OFDM PHY configuration in the `cfgDMG` input.

Data Types: `double`

Name-Value Pair Arguments

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

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

Example: `'MaximumLDPCIterationCount','12','EarlyTermination','false'` specifies a maximum of 12 LDPC decoding iterations and disables early termination so that the decoder completes the 12 iterations.

`LDPCDecodingMethod` — LDPC decoding algorithm
`'bp'` (default) | `'layered-bp'` | `'norm-min-sum'` | `'offset-min-sum'`

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

- `'bp'` — Use the belief propagation (BP) decoding algorithm. For more information, see “Belief Propagation Decoding” on page 3-128.
- `'layered-bp'` — Use the layered BP decoding algorithm, suitable for quasi-cyclic parity check matrices (PCMs). For more information, see “Layered Belief Propagation Decoding” on page 3-129.
- `'norm-min-sum'` — Use the layered BP decoding algorithm with the normalized min-sum approximation. For more information, see “Normalized Min-Sum Decoding” on page 3-129.
- `'offset-min-sum'` — Use the layered BP decoding algorithm with the offset min-sum approximation. For more information, see “Offset Min-Sum Decoding” on page 3-129.

Data Types: `char` | `string`

`MinSumScalingFactor` — Scaling factor for normalized min-sum LDPC decoding
`0.75` (default) | scalar in the interval (0, 1]

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

Dependencies

To enable this argument, specify the `'LDPCDecodingMethod'` name-value pair argument as `'norm-min-sum'`.

Data Types: `double`

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

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

Dependencies

To enable this argument, specify the `'LDPCDecodingMethod'` name-value pair argument as `'offset-min-sum'`.

Data Types: `double`

`MaximumLDPCIterationCount` — Maximum number of LDPC decoding iterations
`12` (default) | positive integer

Maximum number of LDPC decoding iterations, specified as the comma-separated pair consisting of `'MaximumLDPCIterationCount'` and a positive integer.

Data Types: `double`

`EarlyTermination` — Enable early termination of LDPC decoding
`false` or `0` (default) | `true` or `1`

Enable early termination of LDPC decoding, specified as the comma-separated pair consisting of `'EarlyTermination'` and `1` (`true`) or `0` (`false`).

- When you set this value to `0` (`false`), LDPC decoding completes the number of iterations specified in the `'MaximumLDPCIterationCount'` name-value pair argument regardless of parity check status.
- When you set this value to `1` (`true`), LDPC decoding terminates when all parity checks are satisfied.

Data Types: `logical`

Output Arguments

`dataBits` — Bits recovered from DMG Data field

`1` | `0` | binary-valued column vector

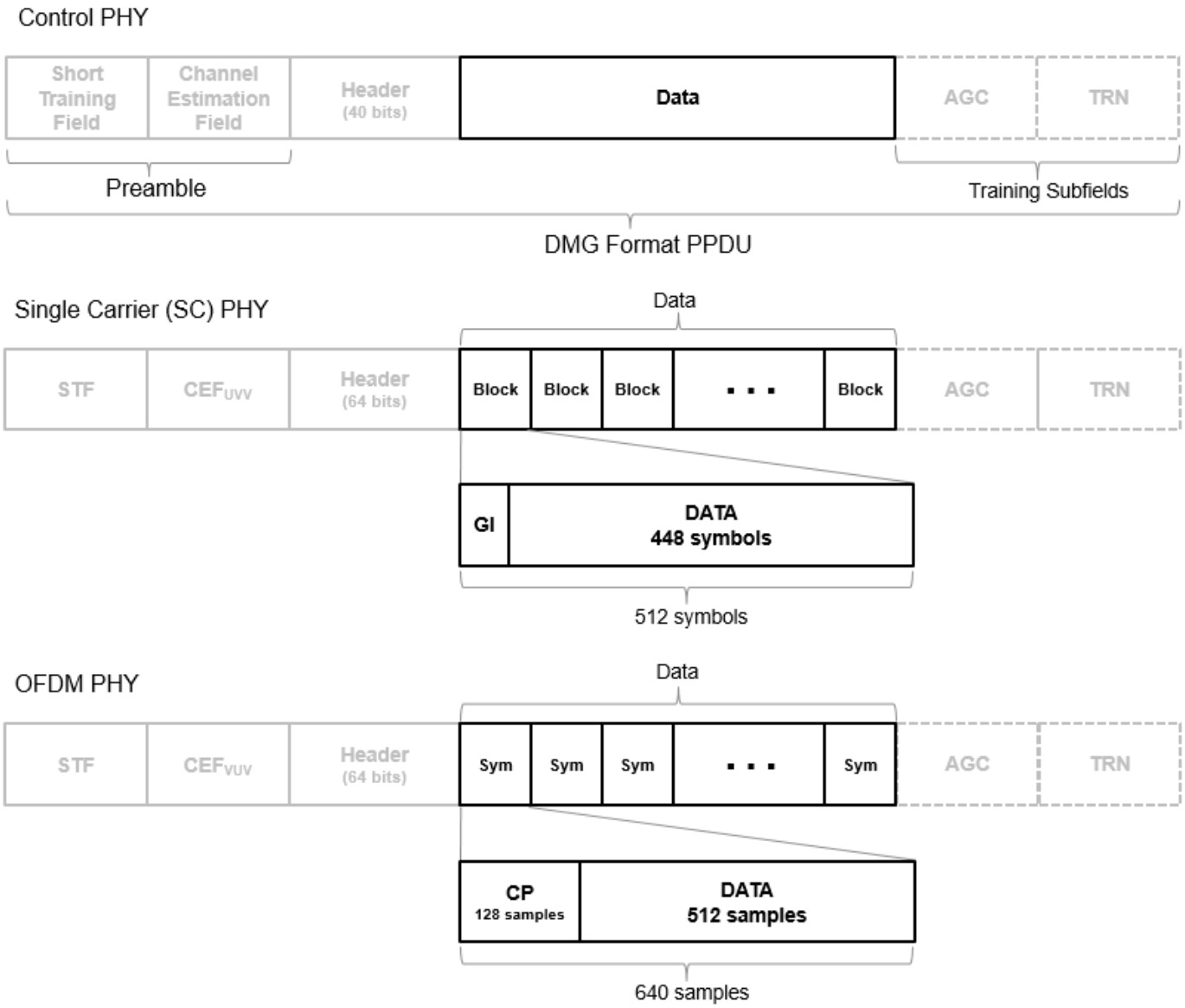
Bits recovered from the DMG Data field, returned as `1`, `0`, or a binary-valued column vector of length $8 \times L$, where L is the PSDU length in bytes.

Data Types: `int8`

More About

DMG Data Field

The DMG format supports three PHY modulation schemes: control, single-carrier (SC), and OFDM. The DMG Data field serves the same function for all three PHY types and carries the user data payload. The length of the DMG Data field differs between PHY types.



For SC PHY, each block in the data field is 512 symbols long and has a guard interval (GI) of 64 symbols with the Golay Sequence. For OFDM, each OFDM symbol in the data field is 640 samples long and has a cyclic prefix (CP) of 128 samples to prevent intersymbol interference.

IEEE 802.11ad-2012 specifies the common aspects of the DMG PPDU packet structure in Section 21.3. The PHY modulation-specific aspects of the data field structure are specified in these sections.

- The DMG control PHY packet structure is specified in Section 21.4.
- The DMG OFDM PHY packet structure is specified in Section 21.5.
- The DMG SC PHY packet structure is specified in Section 21.6.

Algorithms

This function supports these four LDPC decoding algorithms.

Belief Propagation Decoding

The function implements the BP algorithm based on the decoding algorithm presented in [2]. For transmitted LDPC-encoded codeword $c = (c_0, c_1, \ldots, c_{n-1})$, the input to the LDPC decoder is the log-likelihood ratio (LLR) given by

$$L(c_i) = \log\left(\frac{\Pr(c_i = 0 \,|\, \text{channel output for } c_i)}{\Pr(c_i = 0 \,|\, \text{channel output for } c_i)}\right).$$

In each iteration, the function updates the key components of the algorithm based on these equations:

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

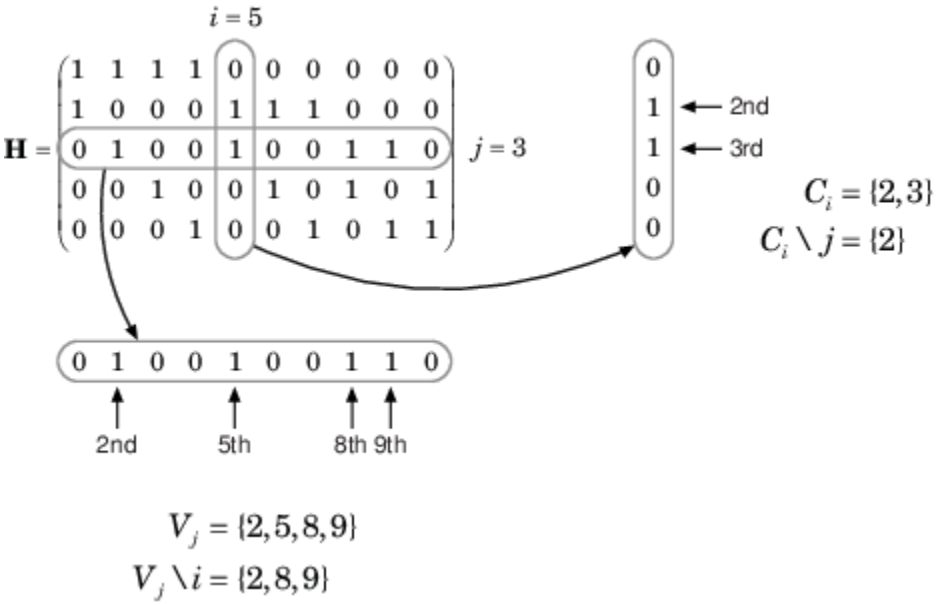
$L(q_{ij}) = L(c_i) + \sum_{j' \in C_i \backslash \{j\}} L(r_{j'i})$, initialized as $L(q_{ij}) = L(c_i)$ before the first iteration, and

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

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

Index sets $C_i \backslash \{j\}$ and $V_j \backslash \{i\}$ are based on the PCM such that the sets C_i and V_j correspond to all nonzero elements in column i and row j of the PCM, respectively.

This figure demonstrates how to compute these index sets for PCM **H** for the case $i = 5$ and $j = 3$.



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

When you specify the `'EarlyTermination'` name-value pair argument as `0` (`false`), the decoding terminates after the number of iterations specified by the `'MaximumLDPCIterationCount'` name-value pair argument. When you specify the `'EarlyTermination'` name-value pair argument as `1` (`true`), the decoding terminates when all parity checks are satisfied ($\mathbf{Hc}^T = 0$) or after the number of iterations specified by the `'MaximumLDPCIterationCount'` name-value pair argument.

Layered Belief Propagation Decoding

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

(1) $L\left(q_{mj}\right) = L\left(q_j\right) - R_{mj}$

(2) $\Psi(x) = \log(|\tanh(x/2)|)$

(3) $A_{mj} = \sum_{n \in N(m)\backslash\{j\}} \Psi(L(q_{mn}))$

(4) $s_{mj} = \prod_{n \in N(m)\backslash\{j\}} \mathrm{sgn}(L(q_{mn}))$

(5) $R_{mj} = -s_{mj}\Psi\left(A_{mj}\right)$

(6) $L\left(q_j\right) = L\left(q_{mj}\right) + R_{mj}$

For each layer, the decoding equation (6) works on the combined input obtained from the current LLR inputs, $L\left(q_{mj}\right)$, and the previous layer updates, R_{mj}.

Because the layered BP algorithm updates only a subset of the nodes in a layer, this algorithm is faster than the BP algorithm. To achieve the same error rate as attained with BP decoding, use half the number of decoding iterations when using the layered BP algorithm.

Normalized Min-Sum Decoding

The function implements the normalized min-sum decoding algorithm by following the layered BP algorithm with equation (3) replaced by

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

where α is the scaling factor specified by the `'MinSumScalingFactor'` name-value pair argument. This equation is an adaptation of equation (4) presented in [4].

Offset Min-Sum Decoding

The function implements the offset min-sum decoding algorithm by following the layered BP algorithm with equation (3) replaced by

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

where β is the offset specified by the `'MinSumOffset'` name-value pair argument. This equation is an adaptation of equation (5) presented in [4].

Version History

Introduced in R2017b

References

[1] IEEE STD 802.11ad-2012 (Amendment to IEEE Std 802.11-2012, as amended by IEEE Std 802.11ae™-2012 and IEEE Std 802.11a™-2012). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 4: Enhancements for Very High Throughput Operation in Bands below 6 GHz.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

[2] Gallager, Robert G. *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.

[3] Hocevar, D.E. "A Reduced Complexity Decoder Architecture via Layered Decoding of LDPC Codes." In *IEEE Workshop on Signal Processing Systems, 2004. SIPS 2004.*, 107-12. Austin, Texas, USA: IEEE, 2004. https://doi.org/10.1109/SIPS.2004.1363033.

[4] Jinghu Chen, R.M. Tanner, C. Jones, and Yan Li. "Improved Min-Sum Decoding Algorithms for Irregular LDPC Codes." In *Proceedings. International Symposium on Information Theory, 2005. ISIT 2005.*, 449-53, 2005. https://doi.org/10.1109/ISIT.2005.1523374.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanDMGHeaderBitRecover` | `wlanDMGConfig`

wlanDMGHeaderBitRecover

Recover bits from DMG Header field

Syntax

```
[headerBits,failHCS] = wlanDMGHeaderBitRecover(rxHeader,noiseVarEst,cfgDMG)
[headerBits,failHCS] = wlanDMGHeaderBitRecover(rxHeader,noiseVarEst,csi,
cfgDMG)
[headerBits,failHCS] = wlanDMGHeaderBitRecover( ___ ,Name,Value)
```

Description

`[headerBits,failHCS] = wlanDMGHeaderBitRecover(rxHeader,noiseVarEst,cfgDMG)` recovers `headerBits`, a column vector of bits, from `rxHeader`, the DMG Header field of a directional multi-gigaqbit (DMG) transmission. The function recovers `headerBits` by using noise variance estimate `noiseVarEst` and DMG transmission parameters `cfgDMG`.

The function also returns `failHCS`, the result of the header check sequence (HCS) on the recovered bits.

`[headerBits,failHCS] = wlanDMGHeaderBitRecover(rxHeader,noiseVarEst,csi, cfgDMG)` enhances the demapping of OFDM subcarriers by using channel state information `csi`. Use this syntax for DMG transmissions that use an orthogonal frequency-division multiplexing (OFDM) PHY configuration.

`[headerBits,failHCS] = wlanDMGHeaderBitRecover( ___ ,Name,Value)` specifies algorithm options by using one or more name-value pair arguments, in addition to any input argument combination from previous syntaxes. For example, `'LDPCDecodingMethod','layered-bp'` specifies the layered belief propagation low-density parity-check (LDPC) decoding algorithm.

Examples

Recover DMG Header Field from Control Transmission

Recover bits from the DMG Header field in a control transmission.

Create a DMG configuration object with a modulation and coding scheme (MCS) for a control PHY configuration.

```
cfgDMG = wlanDMGConfig('MCS',0);
```

Create a sequence of data bits and generate a DMG waveform.

```
bits = randi([0 1],8*cfgDMG.PSDULength,1,'int8');
waveform = wlanWaveformGenerator(bits,cfgDMG);
```

Pass the waveform through a noiseless channel.

```
noiseVarEst = 0;
```

Extract the DMG Header field by using the `wlanFieldIndices` function.

```
ind = wlanFieldIndices(cfgDMG);
rxSym = waveform(ind.DMGHeader(1):ind.DMGHeader(2));
```

Rotate the received signal by 90 degrees.

```
rxSymRotated = rxSym.*exp(-1i*(pi/2)*(0:size(rxSym,1) - 1).');
```

Generate a Golay sequence of length 32 by using the `wlanGolaySequence` function.

```
len = 32;
Ga = wlanGolaySequence(len);
```

Despread the signal with a factor equal to the golay sequence length.

```
rxHeader = reshape(rxSymRotated,len,length(rxSymRotated)/len)'*Ga/len;
```

Recover the bits from the DMG Header field.

```
[headerBits,failHCS] = wlanDMGHeaderBitRecover(rxHeader,noiseVarEst,cfgDMG);
```

Display the result of the HCS check.

```
disp(failHCS);
```

```
   0
```

Recover DMG Header Field from OFDM Transmission

Recover bits from the DMG Header field of an OFDM transmission.

Configure an OFDM transmission by creating a DMG configuration object with an MCS of 14.

```
cfgDMG = wlanDMGConfig('MCS',14);
```

Create a sequence of data bits and generate a DMG waveform.

```
bits = randi([0 1],8*cfgDMG.PSDULength,1,'int8');
waveform = wlanWaveformGenerator(bits,cfgDMG);
```

Pass the waveform through a channel, assuming additive white Gaussian noise (AWGN) for the specified signal-to-noise ratio (SNR).

```
snr = 10;                       % SNR, in dB
noiseVarEst = 10^(-snr/10);     % Noise variance
rxSig = awgn(waveform,snr);
```

Extract the DMG Header field from the received signal.

```
ind = wlanFieldIndices(cfgDMG);
rxSym = rxSig(ind.DMGHeader(1):ind.DMGHeader(2));
```

Perform OFDM demodulation on the received header and extract the data subcarriers.

```
demod = wlanDMGOFDMDemodulate(rxSym);
info = wlanDMGOFDMInfo;
rxHeader = demod(info.DataIndices,:);
```

Recover the bits from the DMG Header field, assuming a CSI estimate of all ones.

```
csi = ones(length(info.DataIndices),1);
[headerBits,failHCS] = wlanDMGHeaderBitRecover(rxHeader,noiseVarEst,csi,cfgDMG);
```

Confirm that the recovered bits pass the HCS.

```
disp(failHCS)
```

```
   0
```

Recover Header Bits from DMG SC Transmission

Recover information bits from the DMG header in a single-carrier (SC) transmission.

Transmitter

Create a DMG configuration object with an MCS for an SC PHY configuration.

```
cfg = wlanDMGConfig('MCS',10);
```

Create the input sequence of data bits and generate a DMG waveform.

```
txBits = randi([0 1],8*cfg.PSDULength,1,'int8');
tx = wlanWaveformGenerator(txBits,cfg);
```

AWGN Channel

Set an SNR of 10 dB, calculate the noise power (noise variance), and add AWGN to the waveform by using the `awgn` function.

```
snr = 10;
nVar = 10^(-snr/10);
rx = awgn(tx,snr);
```

Receiver

Extract the header field.

```
ind = wlanFieldIndices(cfg);
rxHeader = rx(ind.DMGHeader(1):ind.DMGHeader(2));
```

Reshape the received waveform into blocks. Set the data block size to 512 and the guard interval length to 64. Remove the last guard interval from the received data waveform.

```
blkSize = 512;
Ngi = 64;
rxHeader = reshape(rxHeader,blkSize,[]);
rxSym = rxHeader(Ngi+1:end,:);
disp(size(rxSym))
```

```
   448     2
```

Recover the header bits from the DMG header, specifying layered belief propagation LDPC decoding.

```
[rxBits,failHCS] = wlanDMGHeaderBitRecover(rxSym,nVar,cfg,'LDPCDecodingMethod','norm-min-sum');
```

Confirm that the recovered bits pass the HCS.

```
disp(failHCS)
```

```
   0
```

Input Arguments

`rxHeader` — Received DMG Header field

column vector | matrix

Received DMG Header field signal, specified as a column vector or matrix. The contents and size of this input depend on the PHY configuration you specify in the `cfgDMG` input.

- SC PHY — This input contains the time-domain DMG Header field signal in a 448-by-N_{blks} matrix. The value 448 is the number of symbols in a DMG Header block, and N_{blks} is the number of DMG Header blocks.
- OFDM PHY — This input contains the demodulated DMG Data field OFDM symbols in a column vector of length 336. The value 336 is the number of data subcarriers in the DMG Header field.
- Control PHY — This input contains the time-domain DMG Header field in a column vector of length N_b, where N_b is the number of despread symbols.

Data Types: `double`
Complex Number Support: Yes

`noiseVarEst` — Noise variance estimate

nonnegative scalar

Noise variance estimate, specified as a nonnegative scalar.

Data Types: `double`

`cfgDMG` — DMG transmission configuration

`wlanDMGConfig` object

DMG transmission configuration, specified as a `wlanDMGConfig` object.

`csi` — Channel State Information

real-valued column vector

Channel state information, specified as a real-valued column vector of length 336. The value 336 specifies the number of data subcarriers in the DMG Header field.

Dependencies

To enable this input, specify an OFDM PHY configuration in the `cfgDMG` input.

Data Types: `double`

Name-Value Pair Arguments

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

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

Example: `'MaximumLDPCIterationCount','12','EarlyTermination','false'` specifies a maximum of 12 LDPC decoding iterations and disables early termination so that the decoder completes the 12 iterations.

`LDPCDecodingMethod` — LDPC decoding algorithm

`'bp'` (default) | `'layered-bp'` | `'norm-min-sum'` | `'offset-min-sum'`

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

- `'bp'` — Use the belief propagation (BP) decoding algorithm. For more information, see “Belief Propagation Decoding” on page 3-138.
- `'layered-bp'` — Use the layered BP decoding algorithm, suitable for quasi-cyclic parity check matrices (PCMs). For more information, see “Layered Belief Propagation Decoding” on page 3-139.
- `'norm-min-sum'` — Use the layered BP decoding algorithm with the normalized min-sum approximation. for more information, see “Normalized Min-Sum Decoding” on page 3-139.
- `'offset-min-sum'` — Use the layered BP decoding algorithm with the offset min-sum approximation. For more information, see “Offset Min-Sum Decoding” on page 3-139.

Data Types: `char` | `string`

`MinSumScalingFactor` — Scaling factor for normalized min-sum LDPC decoding

`0.75` (default) | scalar in the interval (0, 1]

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

Dependencies

To enable this argument, specify the `'LDPCDecodingMethod'` name-value pair argument as `'norm-min-sum'`.

Data Types: `double`

`MinSumOffset` — Offset for offset min-sum LDPC decoding

`0.5` (default) | nonnegative scalar

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

Dependencies

To enable this argument, specify the `'LDPCDecodingMethod'` name-value pair argument as `'offset-min-sum'`.

Data Types: `double`

`MaximumLDPCIterationCount` — Maximum number of LDPC decoding iterations
`12` (default) | positive integer

Maximum number of LDPC decoding iterations, specified as the comma-separated pair consisting of `'MaximumLDPCIterationCount'` and a positive integer.

Data Types: `double`

`EarlyTermination` — Enable early termination of LDPC decoding
`false` or `0` (default) | `true` or `1`

Enable early termination of LDPC decoding, specified as the comma-separated pair consisting of `'EarlyTermination'` and `1` (`true`) or `0` (`false`).

- When you set this value to `0` (`false`), LDPC decoding completes the number of iterations specified in the `'MaximumLDPCIterationCount'` name-value pair argument regardless of parity check status.
- When you set this value to `1` (`true`), LDPC decoding terminates when all parity checks are satisfied.

Data Types: `logical`

Output Arguments

`headerBits` — Bits recovered from DMG Header field
`1` | `0` | binary-valued column vector

Bits recovered from the DMG Header field, returned as `1`, `0`, or a binary-valued column vector.

- If you specify an OFDM or SC PHY configuration in the `cfgDMG` input, this output contains 64 elements.
- If you specify a control PHY configuration in the `cfgDMG` input, this output contains 40 elements.

Data Types: `int8`

`failHCS` — HCS check result
`0` | `1`

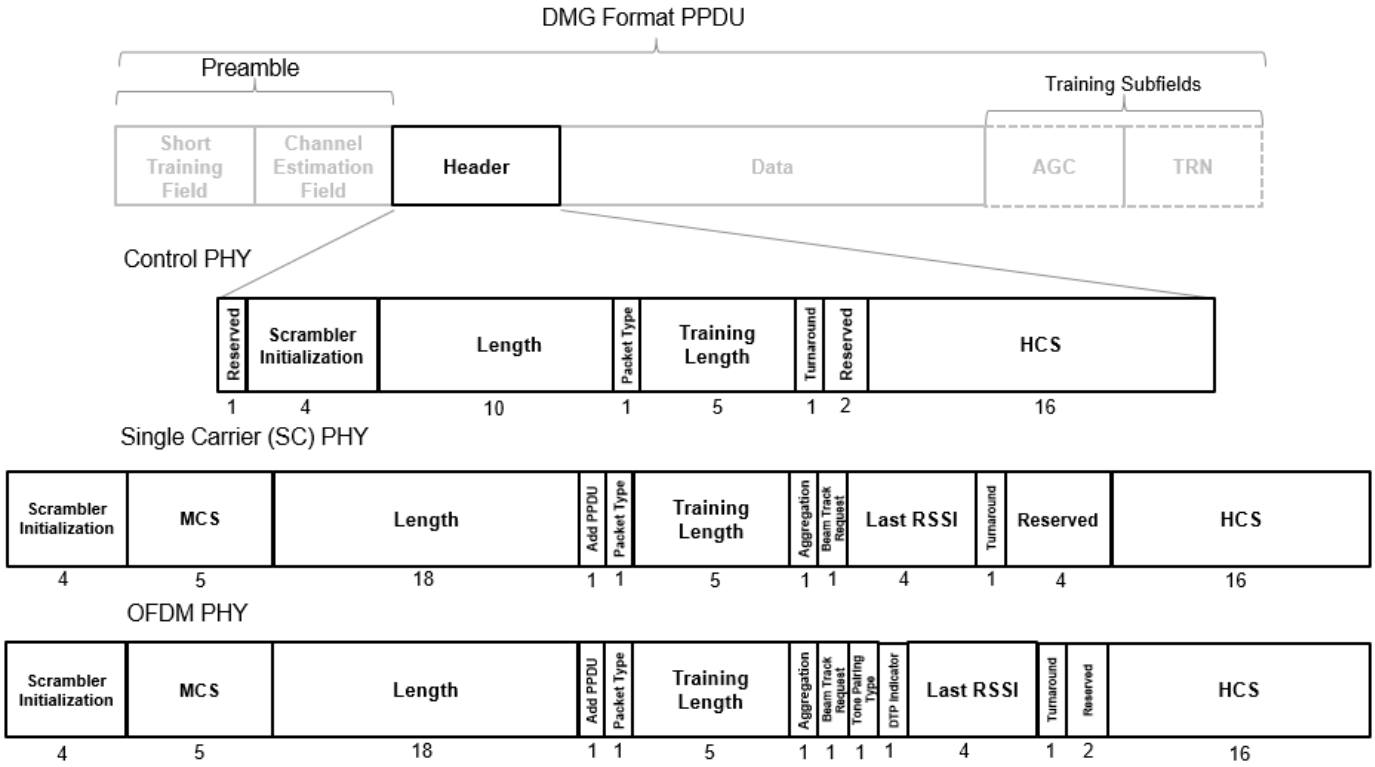
HCS check result, returned as `0` or `1`. When the bits recovered from the DMG Header fail the HCS check, the function returns this output as `1`. Otherwise, the function returns this output as `0`.

Data Types: `logical`

More About

DMG Header Field

In the DMG format, the DMG Header field is different in size and content for each supported PHY modulation scheme. This field contains additional information for the receiver.



The total size of the DMG Header field is 40 bits for control PHY configurations and 64 bits for SC and OFDM PHY configurations.

These fields are common for the three PHY modes.

- *Scrambler initialization* — Specifies the initial state for the scrambler
- *MCS* — Specifies the MCS for the DMG Data field (not present in control PHY)
- *Length* — Specifies the length of the data field
- *Packet Type* — Specifies whether the beamforming training field is intended for the receiver or the transmitter
- *Training Length* — Specifies the presence of a beamforming training field, and, if present, the length of the field
- *HCS* — Provides a checksum per CRC for the header.

IEEE 802.11ad-2012 specifies the detailed aspects of the DMG header field structure. In particular, the PHY modulation-specific aspects of the header field are specified in these sections.

- The DMG control PHY header structure is specified in Section 21.4.3.2.
- The DMG OFDM PHY header structure is specified in Section 21.5.3.1.
- The DMG SC PHY header structure is specified in Section 21.6.3.1.

Algorithms

This function supports these four LDPC decoding algorithms.

Belief Propagation Decoding

The function implements the BP algorithm based on the decoding algorithm presented in [2]. For transmitted LDPC-encoded codeword $c = (c_0, c_1, \ldots, c_{n-1})$, the input to the LDPC decoder is the log-likelihood ratio (LLR) given by

$$L(c_i) = \log\left(\frac{\Pr(c_i = 0 \,|\, \text{channel output for } c_i)}{\Pr(c_i = 0 \,|\, \text{channel output for } c_i)}\right).$$

In each iteration, the function updates the key components of the algorithm based on these equations:

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

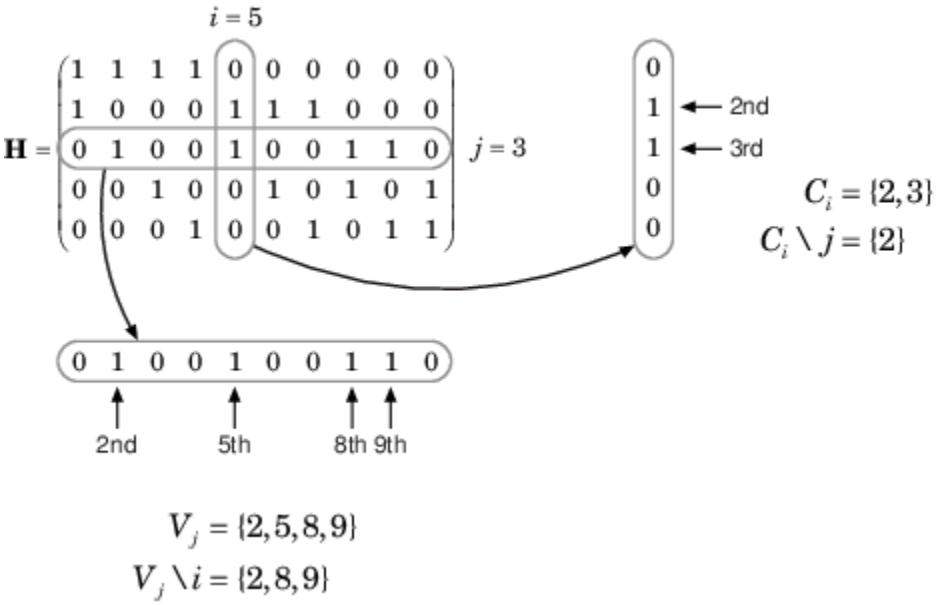
$L(q_{ij}) = L(c_i) + \sum_{j' \in C_i \backslash \{j\}} L(r_{j'i})$, initialized as $L(q_{ij}) = L(c_i)$ before the first iteration, and

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

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

Index sets $C_i \backslash \{j\}$ and $V_j \backslash \{i\}$ are based on the PCM such that the sets C_i and V_j correspond to all nonzero elements in column i and row j of the PCM, respectively.

This figure demonstrates how to compute these index sets for PCM **H** for the case $i = 5$ and $j = 3$.



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

When you specify the `'EarlyTermination'` name-value pair argument as `0` (`false`), the decoding terminates after the number of iterations specified by the `'MaximumLDPCIterationCount'` name-value pair argument. When you specify the `'EarlyTermination'` name-value pair argument as `1` (`true`), the decoding terminates when all parity checks are satisfied ($\mathbf{Hc}^T = 0$) or after the number of iterations specified by the `'MaximumLDPCIterationCount'` name-value pair argument.

Layered Belief Propagation Decoding

The function implements the layered BP algorithm based on the decoding algorithm presented in Section II.A of [3]. The decoding loop iterates over subsets of rows (layers) of the PCM.

For each row, m, in a layer and each bit index, j, the implementation updates the key components of the algorithm based on these equations.

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

(2) $\Psi(x) = \log(|\tanh(x/2)|)$

(3) $A_{mj} = \sum_{n \in N(m)\backslash\{j\}} \Psi(L(q_{mn}))$

(4) $s_{mj} = \prod_{n \in N(m)\backslash\{j\}} \mathrm{sgn}(L(q_{mn}))$

(5) $R_{mj} = -s_{mj}\Psi(A_{mj})$

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

For each layer, the decoding equation (6) works on the combined input obtained from the current LLR inputs, $L(q_{mj})$, and the previous layer updates, R_{mj}.

Because the layered BP algorithm updates only a subset of the nodes in a layer, this algorithm is faster than the BP algorithm. To achieve the same error rate as attained with BP decoding, use half the number of decoding iterations when using the layered BP algorithm.

Normalized Min-Sum Decoding

The function implements the normalized min-sum decoding algorithm by following the layered BP algorithm with equation (3) replaced by

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

where α is the scaling factor specified by the `'MinSumScalingFactor'` name-value pair argument. This equation is an adaptation of equation (4) presented in [4].

Offset Min-Sum Decoding

The function implements the offset min-sum decoding algorithm by following the layered BP algorithm with equation (3) replaced by

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

where β is the offset specified by the `'MinSumOffset'` name-value pair argument. This equation is an adaptation of equation (5) presented in [4].

Version History

Introduced in R2017b

References

[1] IEEE STD 802.11ad-2012 (Amendment to IEEE Std 802.11-2012, as amended by IEEE Std 802.11ae™-2012 and IEEE Std 802.11a™-2012). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 4: Enhancements for Very High Throughput Operation in Bands below 6 GHz.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

[2] Gallager, Robert G. *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.

[3] Hocevar, D.E. "A Reduced Complexity Decoder Architecture via Layered Decoding of LDPC Codes." In *IEEE Workshop on Signal Processing Systems, 2004. SIPS 2004.*, 107-12. Austin, Texas, USA: IEEE, 2004. https://doi.org/10.1109/SIPS.2004.1363033.

[4] Jinghu Chen, R.M. Tanner, C. Jones, and Yan Li. "Improved Min-Sum Decoding Algorithms for Irregular LDPC Codes." In *Proceedings. International Symposium on Information Theory, 2005. ISIT 2005.*, 449-53, 2005. https://doi.org/10.1109/ISIT.2005.1523374.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanDMGDataBitRecover` | `wlanDMGConfig`

wlanDMGOFDMDemodulate

Demodulate fields of DMG waveform

Syntax

```
sym = wlanDMGOFDMDemodulate(rx)
sym = wlanDMGOFDMDemodulate(rx,'OFDMSymbolOffset',symOffset)
```

Description

`sym = wlanDMGOFDMDemodulate(rx)` recovers a demodulated frequency-domain signal by orthogonal frequency-division multiplexing (OFDM) demodulating a directional multi-gigabit (DMG) time-domain waveform.

`sym = wlanDMGOFDMDemodulate(rx,'OFDMSymbolOffset',symOffset)` specifies OFDM symbol sampling offset as a fraction of the cyclic prefix length.

Examples

Demodulate DMG-Data Field and Get OFDM Information

Perform OFDM demodulation on the DMG-Data field, then extract the data and pilot subcarriers.

Generate a WLAN waveform for a DMG transmission, specifying the modulation and coding scheme (MCS).

```
cfg = wlanDMGConfig('MCS','15');
bits = [1; 0; 0; 1];
waveform = wlanWaveformGenerator(bits,cfg);
```

Obtain the field indices and extract the DMG-Data field.

```
ind = wlanFieldIndices(cfg);
rx = waveform(ind.DMGData(1):ind.DMGData(2),:);
```

Perform OFDM demodulation on the DMG-Data field.

```
sym = wlanDMGOFDMDemodulate(rx);
```

Get the OFDM information, then extract the data and pilot subcarriers.

```
info = wlanDMGOFDMInfo;
data = sym(info.DataIndices,:,:);
pilots =  sym(info.PilotIndices,:,:);
```

Demodulate DMG-Data field for OFDM Symbol Offset

Perform OFDM demodulation on the DMG-Data field for an OFDM symbol offset, specified as a fraction of the cyclic prefix length.

Generate a WLAN waveform for a DMG transmisison, specifying the modulation and coding scheme (MCS).

```
cfg = wlanDMGConfig('MCS','12');
bits = [0; 0; 0; 1];
waveform = wlanWaveformGenerator(bits,cfg);
```

Obtain the field indices and extract the DMG-Data field.

```
ind = wlanFieldIndices(cfg);
rx = waveform(ind.DMGData(1):ind.DMGData(2),:);
```

Perform OFDM demodulation on the DMG-Data field, specifying an OFDM symbol offset of `0.5`.

```
sym = wlanDMGOFDMDemodulate(rx,'OFDMSymbolOffset',0.5);
```

Input Arguments

`rx` — Received time-domain signal

complex-valued matrix

Received time-domain signal, specified as a complex-valued matrix of size N_s-by-N_r.

- N_s is the number of time-domain samples. If N_s is not an integer multiple of the OFDM symbol length, L_s, for the specified field,then the function ignores the remaining `mod(`N_s`,`L_s`)` symbols.
- N_r is the number of receive antennas.

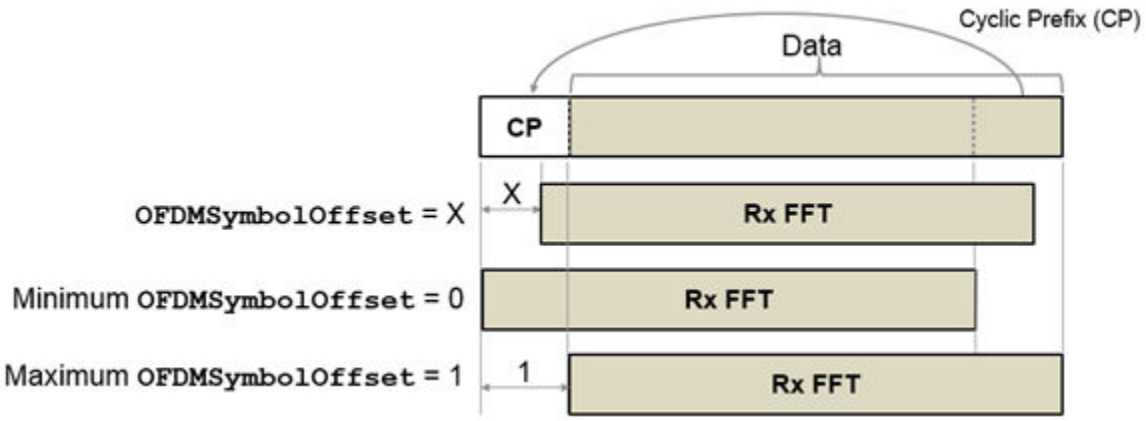
Data Types: `double`
Complex Number Support: Yes

`symOffset` — OFDM symbol sampling offset

`0.75` (default) | scalar in the interval [0, 1]

OFDM symbol sampling offset, as a fraction of the cyclic prefix length, specified as a scalar in the interval [0, 1].

The value that you specify indicates the start location for OFDM demodulation relative to the beginning of the cyclic prefix.



Example: `0.45`

Data Types: `double`

Output Arguments

`sym` — Demodulated frequency-domain signal

complex-valued array

Demodulated frequency-domain signal, returned as a complex-valued array of size N_{sc}-by-N_{sym}-by-N_r.

- N_{sc} is the number of active occupied subcarriers in the demodulated field.
- N_{sym} is the number of OFDM symbols.
- N_r is the number of receive antennas.

Data Types: `double`

Version History

Introduced in R2019a

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`wlanDMGOFDMInfo` | `wlanHEDemodulate` | `wlanS1GDemodulate`

Objects

`wlanDMGConfig`

wlanDMGOFDMInfo

OFDM information for DMG transmission

Syntax

```
info = wlanDMGOFDMInfo
```

Description

`info = wlanDMGOFDMInfo` returns `info`, a structure containing orthogonal frequency-division multiplexing (OFDM) information for the DMG-Data or DMG-Header fields in a directional multi-gigabit transmission.

Examples

Get OFDM Information for DMG-Data Field

Get the OFDM information for the DMG-Data field and display the fast Fourier transform (FFT) length.

```
info = wlanDMGOFDMInfo;
disp(info.FFTLength)
```

```
   512
```

Demodulate DMG-Data Field and Get OFDM Information

Perform OFDM demodulation on the DMG-Data field, then extract the data and pilot subcarriers.

Generate a WLAN waveform for a DMG transmission, specifying the modulation and coding scheme (MCS).

```
cfg = wlanDMGConfig('MCS','15');
bits = [1; 0; 0; 1];
waveform = wlanWaveformGenerator(bits,cfg);
```

Obtain the field indices and extract the DMG-Data field.

```
ind = wlanFieldIndices(cfg);
rx = waveform(ind.DMGData(1):ind.DMGData(2),:);
```

Perform OFDM demodulation on the DMG-Data field.

```
sym = wlanDMGOFDMDemodulate(rx);
```

Get the OFDM information, then extract the data and pilot subcarriers.

```
info = wlanDMGOFDMInfo;
data = sym(info.DataIndices,:,:);
pilots =  sym(info.PilotIndices,:,:);
```

Output Arguments

`info` — OFDM information

structure

OFDM information, returned as a structure containing these fields.

Name	Values	Description	Data Types
`FFTLength`	Positive integer	Length of the fast Fourier transform (FFT)	`double`
`SampleRate`	Positive scalar	Sample rate of the waveform	`double`
`CPLength`	Positive integer	Cyclic prefix length, in samples	`double`
`NumTones`	Nonnegative integer	Number of active subcarriers	`double`
`NumSubchannels`	Positive integer	Number of 20-MHz subchannels	`double`
`ActiveFrequencyIndices`	Column vector of integers in the interval [-`FFTLength`/2, (`FFTLength`/2 – 1)]	Indices of active subcarriers. Each element of this field is the index of an active subcarrier, such that the direct current (DC) or null subcarrier is at the center of the frequency band.	`double`
`ActiveFFTIndices`	Column vector of integers in the interval [1, `FFTLength`]	Indices of active subcarriers within the FFT	`double`
`DataIndices`	Column vector of integers in the interval [1, `NumTones`]	Indices of data within the active subcarriers	`double`
`PilotIndices`	Column vector of integers in the interval [1, `NumTones`]	Indices of pilots within the active subcarriers	`double`

Data Types: `struct`

Version History

Introduced in R2019a

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`wlanDMGOFDMDemodulate`

Objects

`wlanDMGConfig`

wlanFieldIndices

Generate PPDU field indices

Syntax

```
ind = wlanFieldIndices(cfg)
ind = wlanFieldIndices(cfg,field)
ind = wlanFieldIndices( ___ ,OversamplingFactor=osf)
```

Description

`ind = wlanFieldIndices(cfg)` returns `ind`, a structure containing the start and stop indices of the individual component fields that comprise the baseband physical layer convergence procedure protocol data unit (PPDU) waveform.

Note For non-high-throughput (non-HT) format, this function supports generation of field indices only for OFDM modulation.

`ind = wlanFieldIndices(cfg,field)` returns the start and stop indices for the specified field type.

`ind = wlanFieldIndices( ___ ,OversamplingFactor=osf)` returns field indices for an oversampled transmission with the specified oversampling factor. For more information about oversampling, see “FFT-Based Oversampling” on page 3-154.

Examples

Recover Information Bits in HE-SIG-A Field

Recover the information bits in the HE-SIG-A field of a WLAN HE single-user (HE-SU) waveform.

Create a WLAN HE-SU-format configuration object with default settings and use it to generate an HE-SU waveform.

```
cfgHE = wlanHESUConfig;
cbw = cfgHE.ChannelBandwidth;
waveform = wlanWaveformGenerator(1,cfgHE);
```

Obtain the WLAN field indices, which contain the HE-SIG-A field.

```
ind = wlanFieldIndices(cfgHE);
rxSIGA = waveform(ind.HESIGA(1):ind.HESIGA(2),:);
```

Perform orthogonal frequency-division multiplexing (OFDM) demodulation to extract the HE-SIG-A field.

```
sigaDemod = wlanHEDemodulate(rxSIGA,'HE-SIG-A',cbw);
```

Return the pre-HE OFDM information and extract the demodulated HE-SIG-A symbols.

```
preHEInfo = wlanHEOFDMInfo('HE-SIG-A',cbw);
siga = sigaDemod(preHEInfo.DataIndices,:);
```

Recover the HE-SIG-A information bits and other information, assuming no channel noise. Display the parity check result.

```
noiseVarEst = 0;
[bits,failCRC] = wlanHESIGABitRecover(siga,noiseVarEst);
disp(failCRC);
```

```
   0
```

Extract VHT-STF From VHT Waveform

Extract the very-high-throughput short training field (VHT-STF) from a VHT waveform.

Create a VHT-format configuration object for a multiple-input/multiple-output (MIMO) transmission using a 160-MHz channel bandwidth. Generate the corresponding VHT waveform.

```
cfg = wlanVHTConfig('MCS',8,'ChannelBandwidth','CBW160', ...
    'NumTransmitAntennas',2,'NumSpaceTimeStreams',2);
txSig = wlanWaveformGenerator([1;0;0;1],cfg);
```

Determine the component PPDU field indices for the VHT format.

```
ind = wlanFieldIndices(cfg)
```

```
ind = struct with fields:
       LSTF: [1 1280]
       LLTF: [1281 2560]
       LSIG: [2561 3200]
    VHTSIGA: [3201 4480]
     VHTSTF: [4481 5120]
     VHTLTF: [5121 6400]
    VHTSIGB: [6401 7040]
    VHTData: [7041 8320]
```

The VHT PPDU waveform is comprised of eight fields, including seven preamble fields and one data field.

Extract the VHT-STF from the transmitted waveform.

```
stf = txSig(ind.VHTSTF(1):ind.VHTSTF(2),:);
```

Verify that the VHT-STF has dimension 640-by-2, corresponding to the number of samples (80 for each 20-MHz bandwidth segment) and the number of transmit antennas.

```
disp(size(stf))
```

```
   640     2
```

Extract VHT-LTF and Recover VHT Data

Generate a VHT waveform. Extract and demodulate the VHT long training field (VHT-LTF) to estimate the channel coefficients. Recover the data field by using the channel estimate and use this field to determine the number of bit errors.

Configure a VHT-format configuration object with two paths.

```
vht = wlanVHTConfig('NumTransmitAntennas',2,'NumSpaceTimeStreams',2);
```

Generate a random PSDU and create the corresponding VHT waveform.

```
txPSDU = randi([0 1],8*vht.PSDULength,1);
txSig = wlanWaveformGenerator(txPSDU,vht);
```

Pass the signal through a TGac 2x2 MIMO channel.

```
tgacChan = wlanTGacChannel('NumTransmitAntennas',2,'NumReceiveAntennas',2, ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
rxSigNoNoise = tgacChan(txSig);
```

Add AWGN to the received signal. Set the noise variance for the case in which the receiver has a 9-dB noise figure.

```
nVar = 10^((-228.6+10*log10(290)+10*log10(80e6)+9)/10);
awgnChan = comm.AWGNChannel('NoiseMethod','Variance','Variance',nVar);
rxSig = awgnChan(rxSigNoNoise);
```

Determine the indices for the VHT-LTF and extract the field from the received signal.

```
indVHT = wlanFieldIndices(vht,'VHT-LTF');
rxLTF = rxSig(indVHT(1):indVHT(2),:);
```

Demodulate the VHT-LTF and estimate the channel coefficients.

```
dLTF = wlanVHTLTFDemodulate(rxLTF,vht);
chEst = wlanVHTLTFChannelEstimate(dLTF,vht);
```

Extract the VHT-Data field and recover the information bits.

```
indData = wlanFieldIndices(vht,'VHT-Data');
rxData = rxSig(indData(1):indData(2),:);
rxPSDU = wlanVHTDataRecover(rxData,chEst,nVar,vht);
```

Determine the number of bit errors.

```
numErrs = biterr(txPSDU,rxPSDU)
```

```
numErrs = 0
```

Return Field Indices for Oversampled HE MU Waveform

Create a WLAN HE MU configuration object and use it to generate an HE MU waveform with packet extension and an oversampling factor.

```
cfg = wlanHEMUConfig(192);
cfg.User{1}.NominalPacketPadding = 16;
```

```
bits = [1; 0; 0; 1];
osf = 3;
waveform = wlanWaveformGenerator(bits,cfg,OversamplingFactor=osf);
```

Return and display the PPDU field indices.

```
ind = wlanFieldIndices(cfg,OversamplingFactor=osf);
disp(ind)
```

```
      LSTF: [1 480]
      LLTF: [481 960]
      LSIG: [961 1200]
     RLSIG: [1201 1440]
    HESIGA: [1441 1920]
    HESIGB: [1921 2400]
     HESTF: [2401 2640]
     HELTF: [2641 3600]
    HEData: [3601 11280]
      HEPE: [11281 11520]
```

Input Arguments

`cfg` — Transmission format

`wlanHESUConfig` object | `wlanHEMUConfig` object | `wlanHERecoveryConfig` object | `wlanHETBConfig` object | `wlanWURConfig` | `wlanVHTConfig` object | `wlanHTConfig` object | `wlanNonHTConfig` object | `wlanDMGConfig` object | `wlanS1GConfig` object

Transmission format, specified as one of these configuration objects: `wlanHESUConfig`, `wlanHEMUConfig`, `wlanHERecoveryConfig`, `wlanHETBConfig`, `wlanWURConfig`, `wlanVHTConfig`, `wlanHTConfig`, `wlanNonHTConfig`, `wlanDMGConfig`, or `wlanS1GConfig`.

Example: `cfg = wlanVHTConfig`

`field` — PPDU field name

character vector

PPDU field name, specified as a character vector. The valid set of values for this input depends on the transmission format you specify in the `cfg` input.

Transmission Format (`cfg`)	Valid Field Name Values (`field`)
`wlanHESUConfig`, `wlanHEMUConfig`, `wlanHERecoveryConfig`, or `wlanHETBConfig`	`'L-STF'`, `'L-LTF'`, `'L-SIG'`, `'RL-SIG'`, `'HE-SIG-A'`, `'HE-SIG-B'`, `'HE-STF'`, `'HE-LTF'`, `'HE-Data'`, or `'HE-PE'`
`wlanWURConfig`	`'L-STF'`, `'L-LTF'`, `'L-SIG'`, `'BPSK-Mark1'`, `'BPSK-Mark2'`, `'WUR-Data'`, or `'WUR-Sync'`
`wlanDMGConfig`	`'DMG-STF'`, `'DMG-CE'`, `'DMG-Header'`, and `'DMG-Data'` are common for all directional multi-gigabit (DMG) physical layer (PHY) configurations.

Transmission Format (`cfg`)	Valid Field Name Values (`field`)
	When the `TrainingLength` of the `wlanDMGConfig` is positive, additional valid fields are `'DMG-AGC'`, `'DMG-AGCSubfields'`, `'DMG-TRN'`, `'DMG-TRNCE'`, and `'DMG-TRNSubfields'`.
`wlanS1GConfig`	`'S1G-STF'`, `'S1G-LTF1'`, and `'S1G-Data'` are common for all sub-one-gigahertz (S1G) configurations.
	For a 1-MHz or greater than 2-MHz short preamble configuration, additional valid fields are `'S1G-SIG'` and `'S1G-LTF2N'`.
	For a greater than 2-MHz long preamble configuration, additional valid fields are `'S1G-SIG-A'`, `'S1G-DSTF'`, `'S1G-DLTF'`, and `'S1G-SIG-B'`.
`wlanVHTConfig`	`'L-STF'`, `'L-LTF'`, `'L-SIG'`, `'VHT-SIG-A'`, `'VHT-STF'`, `'VHT-LTF'`, `'VHT-SIG-B'`, or `'VHT-Data'`
`wlanHTConfig`	`'L-STF'`, `'L-LTF'`, `'L-SIG'`, `'HT-SIG'`, `'HT-STF'`, `'HT-LTF'`, or `'HT-Data'`
`wlanNonHTConfig`	`'L-STF'`, `'L-LTF'`, `'L-SIG'`, or `'NonHT-Data'`

Data Types: `char` | `string`

`osf` — Oversampling factor

1 (default) | scalar greater than or equal to 1

Oversampling factor, specified as a scalar greater than or equal to 1. The oversampled field indices must be integer-valued.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`ind` — Start and stop indices

structure | integer-valued matrix

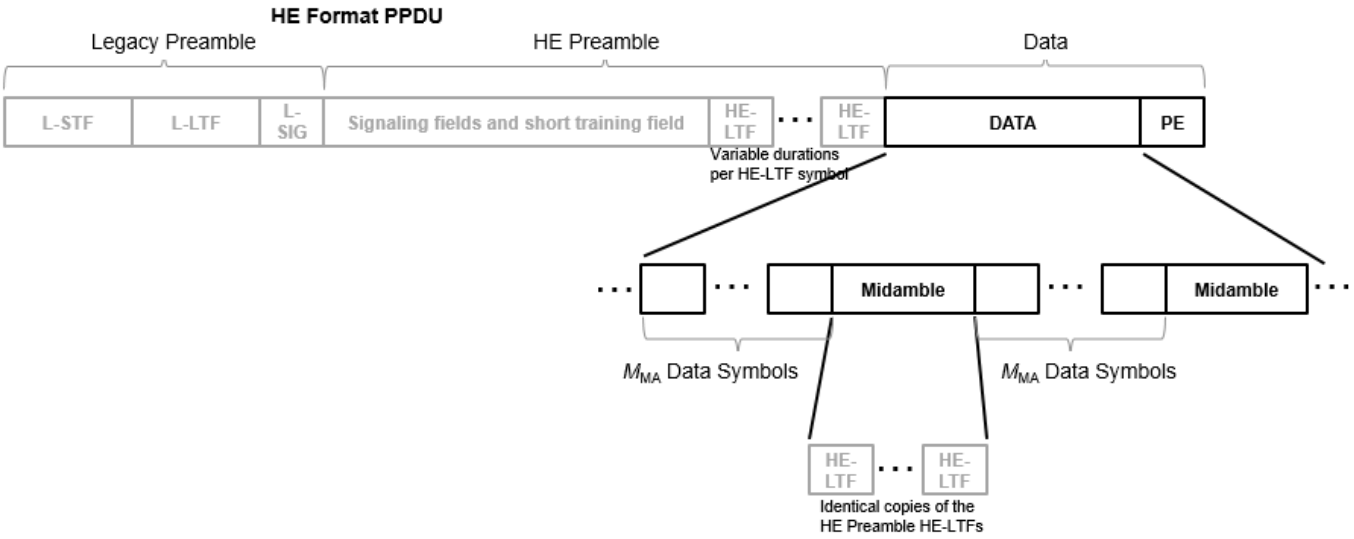
Start and stop indices, returned as a structure or an integer-valued matrix. The indices correspond to the start and stop indices of fields included in the baseband waveform defined by the `cfg` input.

If you specify the `field` input, the function returns `ind` as an *N*-by-2 integer-valued matrix consisting of the start and stop indices of the specified PPDU field. This table outlines the *N* dimension of the *N*-by-2 matrix that is returned based on the specific format and configuration.

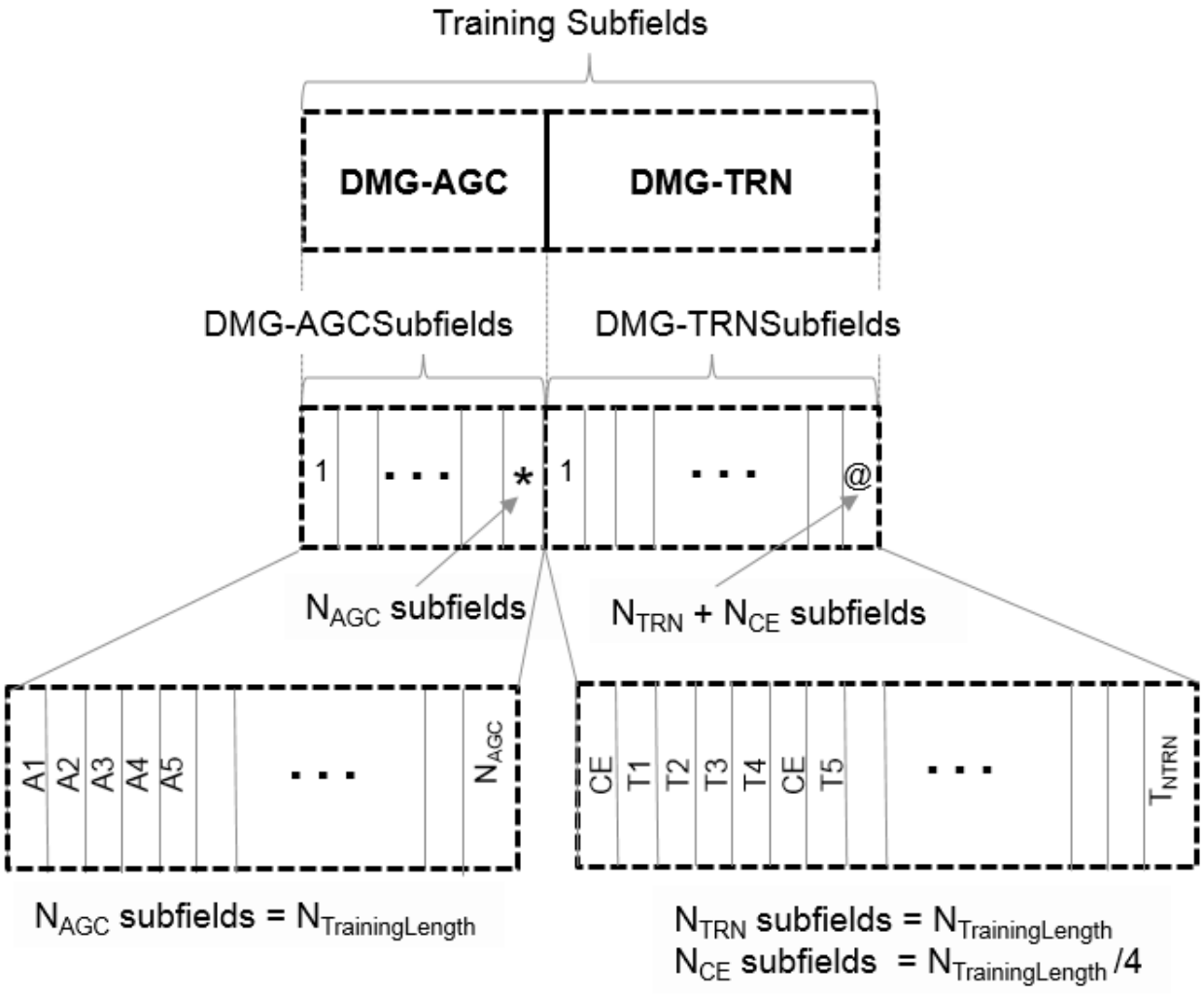
Format	Configuration	`ind` or Specific Field Dimension
non-HT	—	1-by-2 matrix for each field
HT	—	1-by-2 matrix for each field

Format	Configuration	`ind` or Specific Field Dimension
	Null data packet (NDP) mode, if the `PSDULength` property of `wlanHTConfig` object is `0`	Empty matrix
VHT and S1G	—	1-by-2 matrix for each field
	NDP mode, if the `APEPLength` property of the `wlanVHTConfig` or `wlanS1GConfig` object is `0`	Empty matrix
WUR	—	`cfg.NumUsers`-by-2 matrix when you specify the `field` input as `'WUR-Sync'` or `'WUR-Data'`. Otherwise, 1-by-2 matrix for each field.
HE[1]	—	1-by-2 matrix for each field
	NDP mode, if the `APEPLength` property of the `wlanHESUConfig` or `wlanHESUConfig` object is `0`	Empty matrix
	When a midamble is added to the HE-Data field to improve channel estimates for high-Doppler scenarios	*R*-by-2 matrix when you specify the `field` input as `'HE-Data'`, where *R* is the number of data blocks separated by midamble periods
DMG[2]	—	1-by-2 matrix for each field
	When the `TrainingLength` property of `wlanDMGConfig` object is positive	1-by-2 matrix when you specify the `field` input as `'DMG-AGC'` or `'DMG-TRN'`
		`'DMG-AGCSubfields'` is a `TrainingLength`-by-2 matrix.
		`TrainingLength`-by-2 matrix when you specify the `field` input as `'DMG-TRNSubfields'`
		(`TrainingLength`/4)-by-2 matrix when you specify the `field` input as `'DMG-TRNCE'`
	When the `TrainingLength` property of `wlanDMGConfig` object is `0`	Empty matrix when you specify the `field` input as `'DMG-AGC'`, `'DMG-TRN'`, `'DMG-AGCSubfields'`, `'DMG-TRNSubfields'`, or `'DMG-TRNCE'`.

1 As described in section 27.3.12.16 of [1], you can add a midamble to the HE-Data field to improve the channel estimates for high-Doppler scenarios.



2 For DMG, the `'DMG-AGC'` field contains $N_{\text{TrainingLength}}$ subfields, where $N_{\text{TrainingLength}}$ is 0–64 subfields. The `'DMG-TRN'` field contains $N_{\text{TrainingLength}} + (N_{\text{TrainingLength}}/4)$ subfields. As shown in this figure, the indices for `'DMG-AGC'` and `'DMG-TRN'` overlap with the indices of their respective subfields, `'DMG-AGCSubfields'` and `'DMG-TRNSubfields'`.



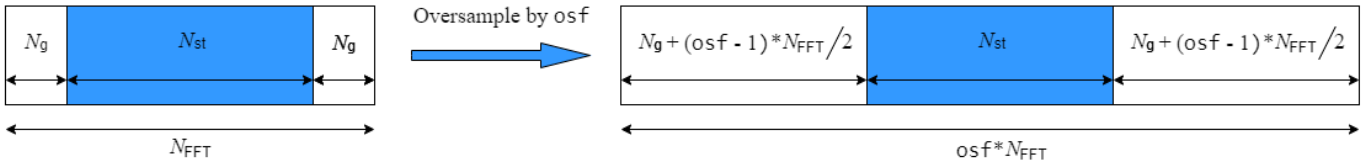
Data Types: `uint32` | `struct`

Algorithms

FFT-Based Oversampling

An oversampled signal is a signal sampled at a frequency that is higher than the Nyquist rate. WLAN signals maximize occupied bandwidth by using small guardbands, which can pose problems for anti-imaging and anti-aliasing filters. Oversampling increases guardband width relative to the total signal bandwidth, thereby increasing the number of samples in the signal.

This function performs oversampling by using a larger IFFT and zero pad when generating an OFDM waveform. This diagram shows the oversampling process for an OFDM waveform with N_{FFT} subcarriers comprising N_{g} guardband subcarriers on either side of N_{st} occupied bandwidth subcarriers.



Version History

Introduced in R2015b

References

[1] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

[2] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`wlanHESUConfig` | `wlanHEMUConfig` | `wlanHETBConfig` | `wlanHERecoveryConfig` | `wlanWURConfig` | `wlanDMGConfig` | `wlanS1GConfig` | `wlanVHTConfig` | `wlanHTConfig` | `wlanNonHTConfig`

wlanFineCFOEstimate

Perform fine CFO estimation

Syntax

```
fOffset = wlanFineCFOEstimate(rxSig,cbw)
fOffset = wlanFineCFOEstimate(rxSig,cbw,corrOffset)
```

Description

`fOffset = wlanFineCFOEstimate(rxSig,cbw)` performs fine carrier frequency offset (CFO) estimation on received time-domain "L-LTF" on page 3-160[2] samples `rxSig` for channel bandwidth `cbw`. The function can estimate a maximum CFO of 156.25 kHz, or half of the subcarrier spacing.

`fOffset = wlanFineCFOEstimate(rxSig,cbw,corrOffset)` specifies the correlation offset as a fraction of the L-LTF cyclic prefix.

Examples

Fine Estimate of Carrier Frequency Offset

Create non-HT configuration object.

```
nht = wlanNonHTConfig;
```

Generate a non-HT waveform.

```
txSig = wlanWaveformGenerator([1;0;0;1],nht);
```

Create a phase and frequency offset object and introduce a 2 Hz frequency offset.

```
pfOffset = comm.PhaseFrequencyOffset('SampleRate',20e6,'FrequencyOffset',2);
rxSig = pfOffset(txSig);
```

Extract the L-LTF and estimate the frequency offset.

```
ind = wlanFieldIndices(nht,'L-LTF');
rxlltf = rxSig(ind(1):ind(2),:);
freqOffsetEst = wlanFineCFOEstimate(rxlltf,'CBW20')
```

```
freqOffsetEst = 2.0000
```

Estimate and Correct CFO for VHT Waveform

Estimate the frequency offset for a VHT signal passing through a noisy, TGac channel. Correct for the frequency offset.

2 IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.

Create a VHT configuration object and create the L-LTF.

```
vht = wlanVHTConfig;
txltf = wlanLLTF(vht);
```

Set the sample rate to correspond to the default bandwidth of the VHT configuration object.

```
fs = 80e6;
```

Create TGac and thermal noise channel objects. Set the noise figure of the AWGN channel to 10 dB.

```
tgacChan = wlanTGacChannel('SampleRate',fs, ...
    'ChannelBandwidth',vht.ChannelBandwidth, ...
    'DelayProfile','Model-C','LargeScaleFadingEffect','Pathloss');

noise = comm.ThermalNoise('SampleRate',fs, ...
    'NoiseMethod','Noise figure', ...
    'NoiseFigure',10);
```

Pass the L-LTF through the noisy TGac channel.

```
rxltfNoNoise = tgacChan(txltf);
rxltf = noise(rxltfNoNoise);
```

Create a phase and frequency offset object and introduce a 25 Hz frequency offset.

```
pfoffset = comm.PhaseFrequencyOffset('SampleRate',fs,'FrequencyOffsetSource','Input port');
rxltf = pfoffset(rxltf,25);
```

Perform a fine estimate the frequency offset using a correlation offset of 0.6. Your results may differ slightly.

```
fOffsetEst = wlanFineCFOEstimate(rxltf,vht.ChannelBandwidth,0.6)
```

```
fOffsetEst = 28.0773
```

Correct for the estimated frequency offset.

```
rxltfCorr = pfoffset(rxltf,-fOffsetEst);
```

Estimate the frequency offset of the corrected signal.

```
fOffsetEstCorr = wlanFineCFOEstimate(rxltfCorr,vht.ChannelBandwidth,0.6)
```

```
fOffsetEstCorr = 2.5029e-13
```

The corrected signal has negligible frequency offset.

Two-Step CFO Estimation and Correction

Estimate and correct for a significant carrier frequency offset in two steps. Estimate the frequency offset after all corrections have been made.

Set the channel bandwidth and the corresponding sample rate.

```
cbw = 'CBW40';
fs = 40e6;
```

Coarse Frequency Correction

Generate an HT format configuration object.

```
cfg = wlanHTConfig('ChannelBandwidth',cbw);
```

Generate the transmit waveform.

```
txSig = wlanWaveformGenerator([1;0;0;1],cfg);
```

Create TGn and thermal noise channel objects. Set the noise figure of the receiver to 9 dB.

```
tgnChan = wlanTGnChannel('SampleRate',fs,'DelayProfile','Model-D', ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
noise = comm.ThermalNoise('SampleRate',fs, ...
    'NoiseMethod','Noise figure', ...
    'NoiseFigure',9);
```

Pass the waveform through the TGn channel and add noise.

```
rxSigNoNoise = tgnChan(txSig);
rxSig = noise(rxSigNoNoise);
```

Create a phase and frequency offset object to introduce a carrier frequency offset. Introduce a 2 kHz frequency offset.

```
pfOffset = comm.PhaseFrequencyOffset('SampleRate',fs,'FrequencyOffsetSource','Input port');
rxSig = pfOffset(rxSig,2e3);
```

Extract the L-STF signal for coarse frequency offset estimation.

```
istf = wlanFieldIndices(cfg,'L-STF');
rxstf = rxSig(istf(1):istf(2),:);
```

Perform a coarse estimate of the frequency offset. Your results may differ.

```
foffset1 = wlanCoarseCFOEstimate(rxstf,cbw)
```

```
foffset1 = 2.0221e+03
```

Correct for the estimated offset.

```
rxSigCorr1 = pfOffset(rxSig,-foffset1);
```

Fine Frequency Correction

Extract the L-LTF signal for fine offset estimation.

```
iltf = wlanFieldIndices(cfg,'L-LTF');
rxltf1 = rxSigCorr1(iltf(1):iltf(2),:);
```

Perform a fine estimate of the corrected signal.

```
foffset2 = wlanFineCFOEstimate(rxltf1,cbw)
```

```
foffset2 = -11.0795
```

The corrected signal offset is reduced from 2000 Hz to approximately 7 Hz.

Correct for the remaining offset.

```
rxSigCorr2 = pfOffset(rxSigCorr1,-foffset2);
```

Determine the frequency offset of the twice corrected signal.

```
rxltf2 = rxSigCorr2(iltf(1):iltf(2),:);
deltaFreq = wlanFineCFOEstimate(rxltf2,cbw)
```

```
deltaFreq = -2.0374e-11
```

The CFO is zero.

Input Arguments

`rxSig` — Received L-LTF samples

complex-valued matrix

Received L-LTF samples, specified as a complex-valued matrix of size N_S-by-N_R. N_S is the number of samples in the L-LTF and N_R is the number of receive antennas.

Note If the number of samples in this input is greater than the number of samples in the L-LTF, the function estimates the CFO by using only the first N_S samples.

Data Types: `single` | `double`

`cbw` — Channel bandwidth

`'CBW5'` | `'CBW10'` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'` | `'CBW320'`

Channel bandwidth, specified as one of these values.

- `'CBW5'` – Channel bandwidth of 5 MHz
- `'CBW10'` – Channel bandwidth of 10 MHz
- `'CBW20'` – Channel bandwidth of 20 MHz
- `'CBW40'` – Channel bandwidth of 40 MHz
- `'CBW80'` – Channel bandwidth of 80 MHz
- `'CBW160'` – Channel bandwidth of 160 MHz
- `'CBW320'` – Channel bandwidth of 320 MHz

Data Types: `char` | `string`

`corrOffset` — Correlation offset

`0.75` (default) | scalar in the interval [0, 1]

Correlation offset as a fraction of the L-LTF cyclic prefix, specified as a scalar in the interval [0, 1]. The duration of the long training symbol varies with bandwidth. For more information, see “L-LTF” on page 3-160.

Data Types: `single` | `double`

Output Arguments

`f0ffset` — Frequency offset

real-valued scalar

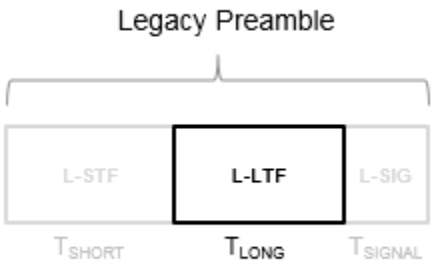
Frequency offset, in Hz, returned as a real-valued scalar. The function can estimate a maximum CFO of 156.25 kHz, or half of the subcarrier spacing.

Data Types: `double`

More About

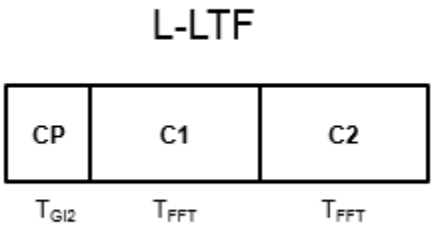
L-LTF

The L-LTF is the second field in the 802.11 OFDM PLCP legacy preamble. The L-LTF is a component of HE, VHT, HT, and non-HT PPDUs.



Channel estimation, fine frequency offset estimation, and fine symbol timing offset estimation rely on the L-LTF.

The L-LTF is composed of a cyclic prefix (CP) followed by two identical long training symbols (C1 and C2). The CP consists of the second half of the long training symbol.



The L-LTF duration varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period ($T_{FFT} = 1 / \Delta_F$)	Cyclic Prefix or Training Symbol Guard Interval (GI2) Duration ($T_{GI2} = T_{FFT} / 2$)	L-LTF Duration ($T_{LONG} = T_{GI2} + 2 \times T_{FFT}$)
20, 40, 80, 160, and 320	312.5	3.2 µs	1.6 µs	8 µs
10	156.25	6.4 µs	3.2 µs	16 µs
5	78.125	12.8 µs	6.4 µs	32 µs

Version History

Introduced in R2015b

References

[1] IEEE Std 802.11™-2016 (Revision of IEEE Std 802.11-2012). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements.

[2] Li, Jian. “Carrier Frequency Offset Estimation for OFDM-Based WLANs.” *IEEE Signal Processing Letters*. Vol. 8, Issue 3, Mar 2001, pp. 80–82.

[3] Moose, P. H. “A technique for orthogonal frequency division multiplexing frequency offset correction.” *IEEE Transactions on Communications*. Vol. 42, Issue 10, Oct 1994, pp. 2908–2914.

[4] Perahia, E., and R. Stacey. *Next Generation Wireless LANs: 802.11n and 802.11ac*. 2nd Edition. United Kingdom: Cambridge University Press, 2013.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanCoarseCFOEstimate` | `comm.PhaseFrequencyOffset` | `wlanLLTF`

wlanFormatDetect

Detect packet format

Syntax

```
format = wlanFormatDetect(rxSig,chEst,noiseVarEst,cbw)
format = wlanFormatDetect(rxSig,chEst,noiseVarEst,cbw,Name,Value)
```

Description

`format = wlanFormatDetect(rxSig,chEst,noiseVarEst,cbw)` detects and returns `format`, the packet format of a received time-domain signal. The function detects the packet format by performing a series of checks on `rxSig`, a portion of the signal whose contents uniquely determine the packet format. For more information, see “Format Detection Processing” on page 3-167. To perform these checks, the function also requires estimated channel characteristics `chEst`, estimated noise variance `noiseVarEst`, and channel bandwidth `cbw`.

`format = wlanFormatDetect(rxSig,chEst,noiseVarEst,cbw,Name,Value)` specifies algorithm options for information bit recovery by using one or more name-value pair arguments.

Examples

Detect HE-SU-Format Waveform

Detect the format of a WLAN HE SU waveform

Generate an HE SU waveform and obtain a received signal by adding noise.

```
cbw = 'CBW20'; % Specify channel bandwidth of 20MHz
cfgHESU = wlanHESUConfig('ChannelBandwidth',cbw); % Create configuration object for HE SU format
bits = [1;0;0;1];
tx = wlanWaveformGenerator(bits,cfgHESU); % Generate HE SU waveform
snr = 10; % Specify signal-to-noise ratio (SNR)
rx = awgn(tx,snr); % Create receive waveform
```

Specify the sample rate and durations of the relevant PPDU fields.

```
sr = 20e6; % Sample rate in samples per second
tLSTF = 8e-6; % Duration of legacy short training field (L-STF)
tLLTF = tLSTF; % Duration of legacy long training field (L-LTF)
```

Determine the field indices and estimate the channel using the L-LTF.

```
ind = tLSTF*sr+(1:tLLTF*sr);
y = wlanLLTFDemodulate(rx(ind,:),cbw);
chEst = wlanLLTFChannelEstimate(y,cbw);
```

Specify the channel noise variance estimate and detect the format of the waveform.

```
noiseVarEst = 10^(-snr/20);
rxSig = rx((tLSTF+tLLTF)*sr+(1:sr*(16e-6)),:); % 16 microseconds corresponding to four OFDM symbo
```

```
format = wlanFormatDetect(rxSig,chEst,noiseVarEst,cbw);
disp(format)
```

```
HE-SU
```

Detect HT-MF Format Waveform

Perform format detection on a WLAN high throughput mixed format (HT-MF) waveform.

Generate an HT-MF waveform and add noise to the transmitted waveform.

```
cbw = 'CBW20';
cfgTx = wlanHTConfig('ChannelBandwidth',cbw);
tx = wlanWaveformGenerator([1;0;0;1],cfgTx);
snr = 10;
rxSig = awgn(tx,snr);
```

Demodulate Received Signal and Perform Channel Estimation

- Determine indices for the L-LTF for the 20 MHz bandwidth waveform. For this calculation, define local variables for the sample rate and duration of the L-STF and L-LTF fields in seconds.
- Demodulate the L-LTF.
- Perform channel estimation using the L-LTF.
- Estimate the noise variance.

```
sr = 20e6;
Tlstf = 8e-6;
Tlltf = 8e-6;

idxlltf = Tlstf*sr+(1:Tlltf*sr);

lltfDemod = wlanLLTFDemodulate(rxSig(idxlltf,:),cbw);
chEst = wlanLLTFChannelEstimate(lltfDemod,cbw);
noiseVarEst = 10^(-snr/20);
```

Detect Signal Format

- Determine indices for the three symbols following the L-LTF. For a 20 MHz bandwidth waveform, the duration for three symbols is 12 μs.
- Perform format detection.

```
idxDetectionSymbols = (Tlstf+Tlltf)*sr+(1:12e-6*sr);

in = rxSig(idxDetectionSymbols,:);
format = wlanFormatDetect(in,chEst,noiseVarEst,cbw)
```

```
format =
'HT-MF'
```

Detect VHT Waveform After Adjusting Recovery Algorithm

Detect the format of a WLAN very high throughput (VHT) waveform, adjusting the default recovery algorithm settings.

Generate a VHT waveform and add white Gaussian noise to the transmitted waveform.

```
cbw = 'CBW80';
cfgTx = wlanVHTConfig('ChannelBandwidth',cbw);
tx = wlanWaveformGenerator([1;0;0;1],cfgTx);
snr = 10;
rxSig = awgn(tx,snr);
```

Received signal demodulation and channel estimation

- Determine indices for the L-LTF for the 80 MHz bandwidth waveform. For this calculation, define local variables for the sample rate and duration of the L-STF and L-LTF in seconds.
- Demodulate the L-LTF.
- Perform channel estimation using the L-LTF.
- Estimate the noise variance.

```
sr = 80e6;
Tlstf = 8e-6;
Tlltf = 8e-6;
idxlltf = Tlstf*sr+(1:Tlltf*sr);
lltfDemod = wlanLLTFDemodulate(rxSig(idxlltf,:),cbw);
chEst = wlanLLTFChannelEstimate(lltfDemod,cbw);
noiseVarEst = 10^(-snr/20);
```

Format detection

- Determine indices for the three symbols following the L-LTF. For an 80 MHz bandwidth waveform, the duration for three symbols is 12 μs.
- Detect format detection using modified recovery settings.

```
TdetectionSymbols = 12e-6;
idxDetectionSymbols = (Tlstf+Tlltf)*sr+(1:TdetectionSymbols*sr);
in = rxSig(idxDetectionSymbols,:);
format = wlanFormatDetect(in,chEst,noiseVarEst,cbw,...
    'OFDMSymbolOffset',0.5,'PilotPhaseTracking','None')
```

```
format =
'VHT'
```

Input Arguments

`rxSig` — Post-LTF portion of received time-domain signal

complex-valued matrix

Post-long-training-field (post-LTF) portion of the received time-domain signal, specified as a complex-valued N_S-by-N_R matrix, where:

- N_S is the number of time-domain samples.
- N_R is the number of receive antennas.

To successfully detect the format of an HE packet, this input must contain all time-domain samples in the four OFDM symbols immediately following the L-LTF: *sym1*, *sym2*, *sym3*, and *sym4*. To successfully detect other packet formats, this signal must contain all time-domain samples in the three OFDM symbols immediately following the relevant LTF: *sym1*, *sym2*, *sym3*. The first entry in each column of this input must be the first time-domain sample of the symbol received by the

corresponding antenna. For more information about how the `wlanFormatDetect` function uses this input for format detection, see “Format Detection Processing” on page 3-167.

Note If the number of received OFDM symbols is greater than four, the function ignores additional samples after *sym4*.

Data Types: `double`
Complex Number Support: Yes

`chEst` — Channel estimate

numeric matrix | 3-D numeric array

Channel estimate for data and pilot subcarriers based on the L-LTF, specified as a numeric matrix or array of size N_{ST}-by-1-by-N_R, where:

- N_{ST} is the number of occupied subcarriers.
- N_R is the number of receive antennas.

The second dimension corresponds to the single transmitted stream in the L-LTF. If the transmission uses multiple antennas, the single transmitted stream includes the combined cyclic shifts.

Data Types: `double`
Complex Number Support: Yes

`noiseVarEst` — Noise variance estimate

nonnegative scalar

Noise variance estimate, specified as a nonnegative scalar.

Data Types: `double`

`cbw` — Channel bandwidth

`'CBW5'` | `'CBW10'` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth, in MHz, specified as one of these values.

- `'CBW5'` – Channel bandwidth of 5 MHz
- `'CBW10'` – Channel bandwidth of 10 MHz
- `'CBW20'` – Channel bandwidth of 20 MHz
- `'CBW40'` – Channel bandwidth of 40 MHz
- `'CBW80'` – Channel bandwidth of 80 MHz
- `'CBW160'` – Channel bandwidth of 160 MHz

Data Types: `char`

Name-Value Pair Arguments

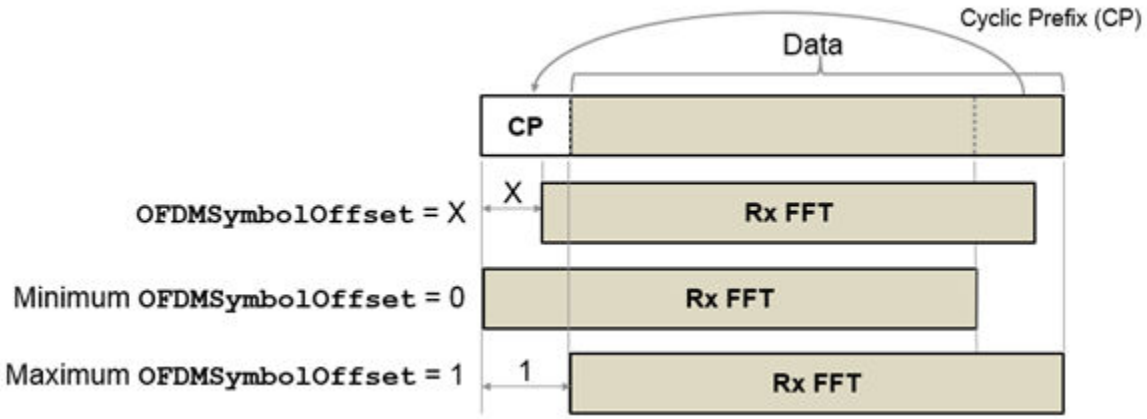
Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

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

Example: `'PilotPhaseTracking','None'` disables pilot phase tracking.

`OFDMSymbolOffset` — OFDM symbol sampling offset
`0.75` (default) | scalar in the interval [0, 1]

OFDM symbol sampling offset represented as a fraction of the cyclic prefix (CP) length, specified as the comma-separated pair consisting of `'OFDMSymbolOffset'` and a scalar in the interval [0, 1]. The value you specify indicates the start location for OFDM demodulation relative to the beginning of the CP. The value `0` represents the start of the CP, and the value `1` represents the end of the CP.



Data Types: `double`

`EqualizationMethod` — Equalization method
`'MMSE'` (default) | `'ZF'`

Equalization method, specified as one of these values.

- `'MMSE'` — The receiver uses a minimum mean-square error equalizer.
- `'ZF'` — The receiver uses a zero-forcing equalizer.

When the received signal has multiple receive antennas, the function exploits receiver diversity during equalization. When the number of transmitted space-time streams is one and you specify this argument as `'ZF'`, the function performs maximal-ratio combining.

Data Types: `char` | `string`

`PilotPhaseTracking` — Pilot phase tracking
`'PreEQ'` (default) | `'None'`

Pilot phase tracking, specified as the comma-separated pair consisting of `'PilotPhaseTracking'` and one of these values.

- `'PreEQ'` — Enable pilot phase tracking, which the function performs before any equalization operation.
- `'None'` — Disable pilot phase tracking.

Data Types: `char` | `string`

Output Arguments

`format` — Packet format

`'Non-HT'` | `'HT-MF'` | `'HT-GF'` | `'VHT'` | `'HE-SU'` | `'HE-EXT-SU'` | `'HE-MU'` | `'HE-TB'`

Packet format, returned as one of these values:

- `'Non-HT'` – Non-high-throughput (non-HT) format
- `'HT-MF'` – High-throughput mixed format (HT MF)
- `'HT-GF'` – High-throughput greenfield format (HT GF)
- `'VHT'` – Very-high-throughput (VHT) format
- `'HE-SU'` – High-efficiency single-user (HE SU) format
- `'HE-EXT-SU'` – HE extended-range single-user (HE ER SU) format
- `'HE-MU'` – HE multiuser (HE MU) format
- `'HE-TB'` – HE trigger-based (HE TB) format

Algorithms

Format Detection Processing

The format detection processing algorithm determines packet format by checking relevant attributes of the `rxSig` input.

- To successfully detect the format of an HE packet, `rxSig` must contain all time-domain samples in the four OFDM symbols following the L-LTF.
- To successfully detect the format of a non-HT, HT-MF, or VHT packet, `rxSig` must contain all time-domain samples in the three OFDM symbols following the L-LTF.
- To successfully detect the format of an HT-GF packet, `rxSig` must contain all time-domain samples in the three OFDM symbols following the HT-LTF1.

The first entry in each column of `rxSig` must be the first time-domain sample of the first OFDM symbol following the relevant LTF received by the corresponding antenna. The function does not use additional samples after the last sample of the fourth OFDM symbol.

Before demodulating any packet symbols, the `wlanFormatDetect` function checks the number of OFDM symbols in the `rxSig` input. If the function detects four or more symbols, it determines the packet format by following the steps outlined in “HE Format Detection” on page 3-167. If the function detects three symbols, it determines the packet format by following the steps outlined in “Legacy Format Detection” on page 3-169.

HE Format Detection

1. Check for the repeated L-SIG (RL-SIG) field. If RL-SIG field is detected, begin detection for HE formats.
    a. Check the validity of the L-SIG field by computing the parity and data rate. If the L-SIG field is valid, follow these steps.
        i. Check the length of the L-SIG field *modulo* 3.
            A. If the length *modulo* 3 is 1, the packet format is either HE ER SU or HE MU. Demodulate the HE-SIG-A field and check the modulation scheme of *sym4*.

I If the modulation method of *sym4* is QBPSK, the format is HE ER SU.

II If the modulation method of *sym4* is BPSK, the format is HE MU.

B If the length *modulo* 3 is 2, the packet format is either HE SU or HE TB. Recover the information bits in the HE-SIG-A field by using the `wlanHESIGABitRecover` function.

I If the first bit is 0, the format is HE TB.

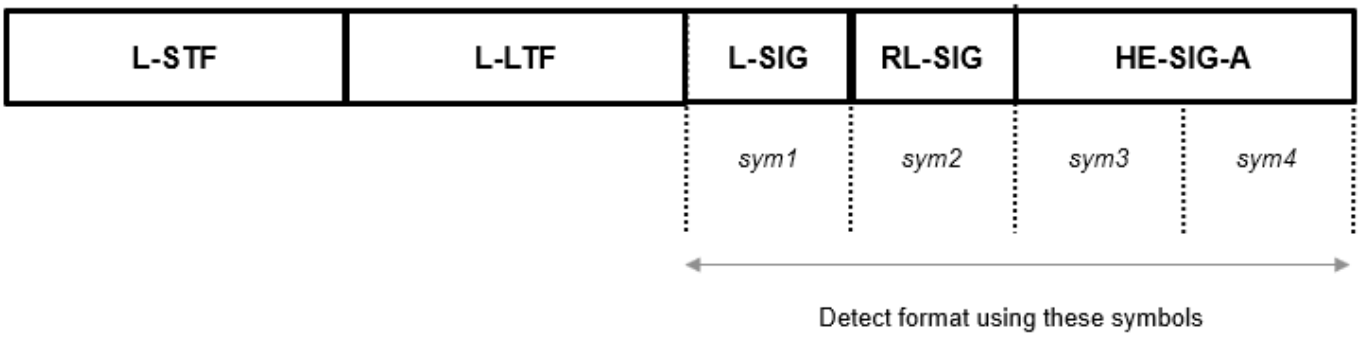
II If the first bit is 1, the format is HE SU.

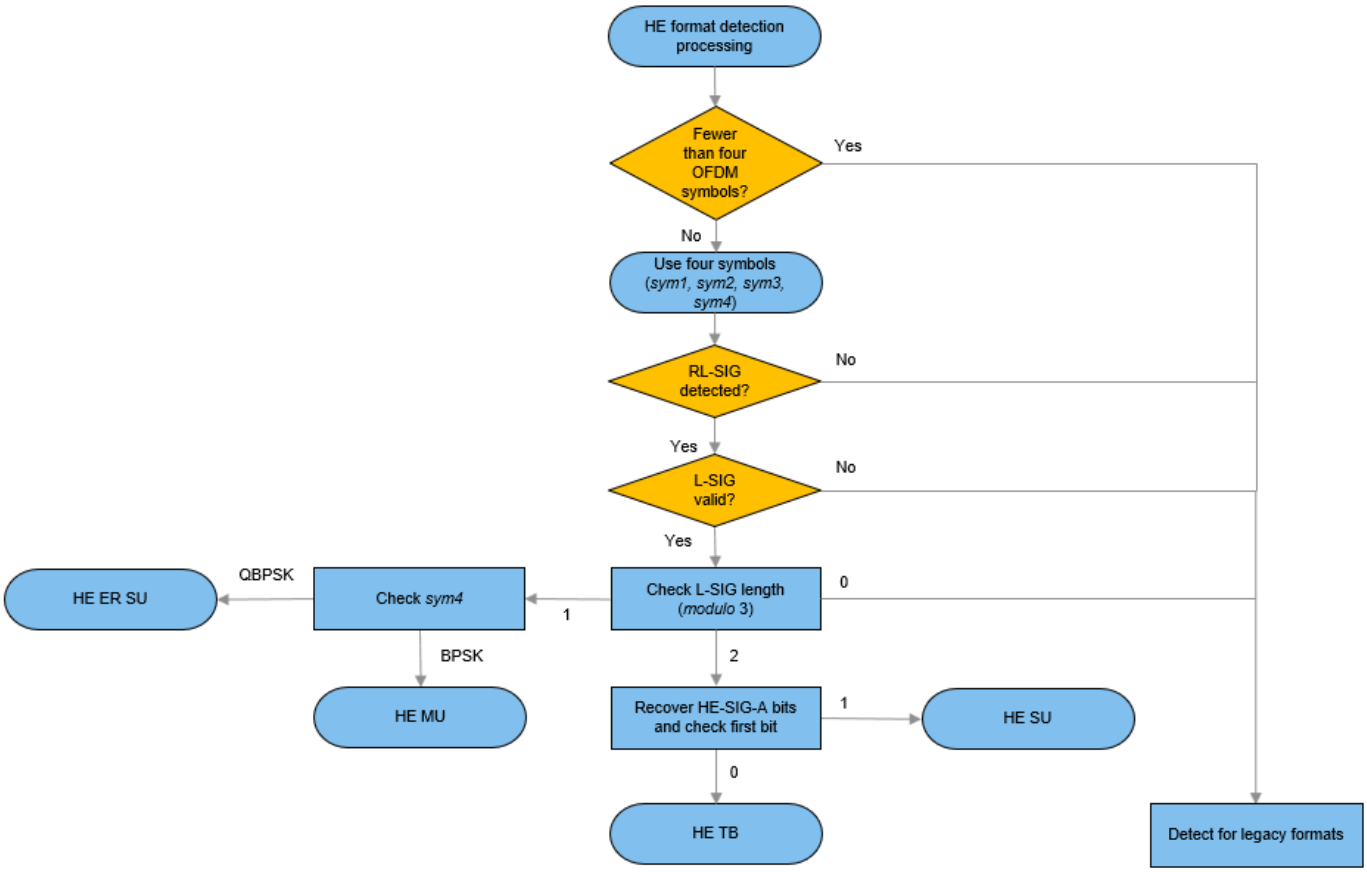
III If the HE-SIG-A field fails the cyclic redundancy check (CRC), detect the format by following the steps in “Legacy Format Detection” on page 3-169.

C If the length *modulo* 3 is `0`, detect the format by following the steps in “Legacy Format Detection” on page 3-169.

b If the L-SIG field is invalid, detect the format by following the steps in “Legacy Format Detection” on page 3-169.

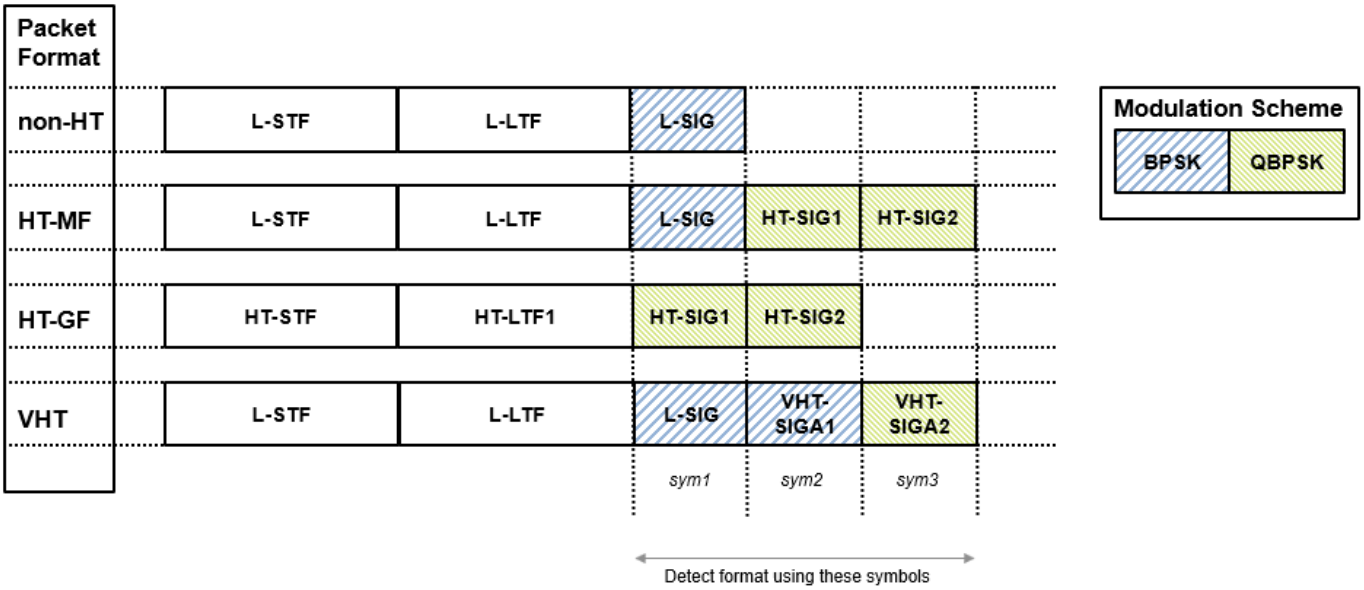
2 If RL-SIG field is not detected, detect the format by following the steps outlined in “Legacy Format Detection” on page 3-169.

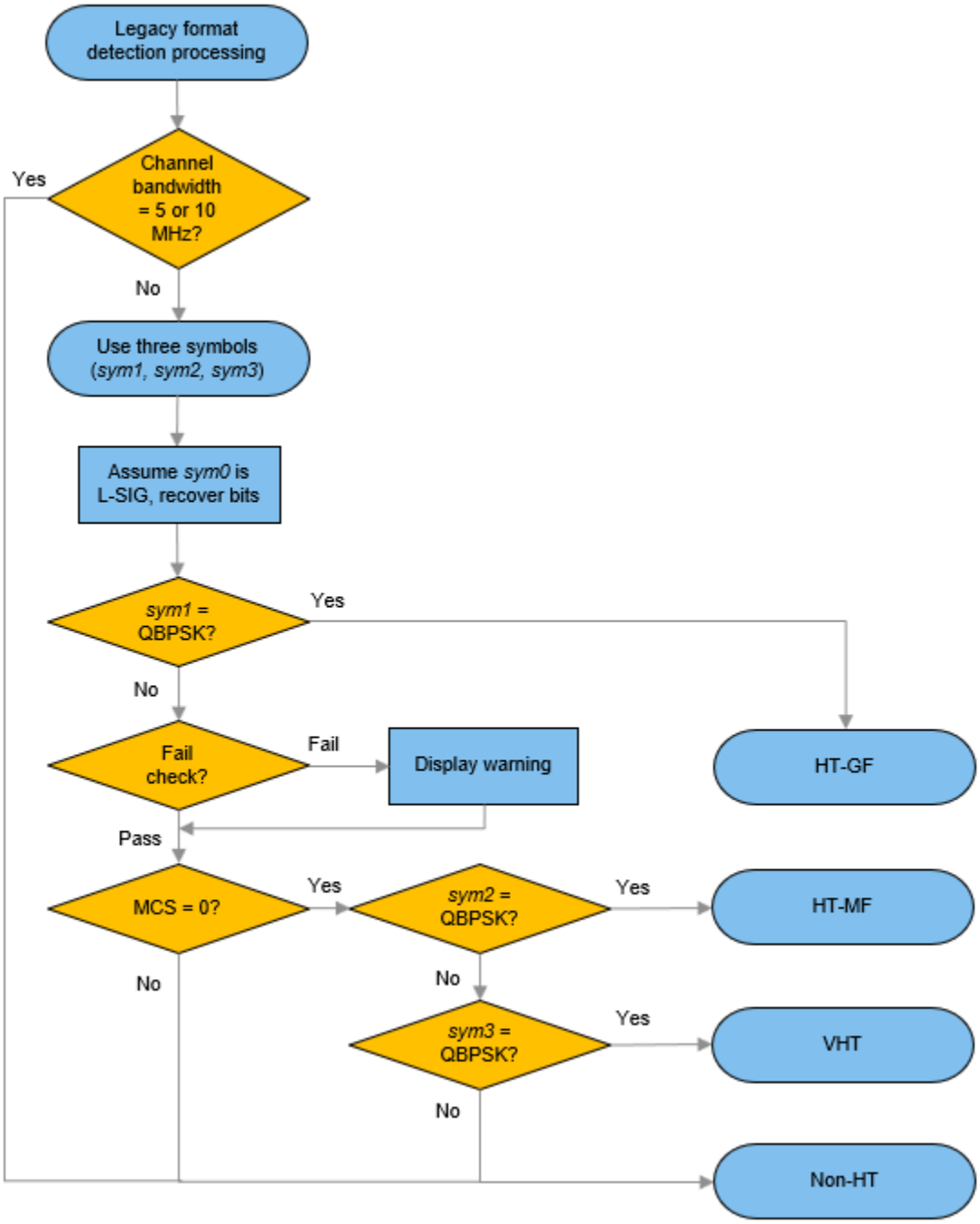




Legacy Format Detection

1. If the modulation scheme of *sym1* is QBPSK, the packet format is HT-GF.
2. If the modulation scheme of *sym1* is BPSK and the L-SIG parity check fails, the function returns a warning. The format detection processing continues because the L-SIG parity check does not conclusively indicate an error in the modulation and coding scheme (MCS) determination.
    a. If the MCS is not 0, the packet format is non-HT.
    b. If the MCS is 0, check the modulation scheme of *sym2*.
        i. If the modulation scheme of *sym2* is QBPSK, the format is HT MF.
        ii. If the modulation scheme of *sym2* is BPSK, detect the modulation scheme of *sym3*.
            A. If the modulation scheme of *sym3* is QBPSK, the format is VHT.
            B. If the modulation scheme of *sym3* is BPSK, the format is non-HT.


Packet Format
non-HT
L-STF
L-LTF
L-SIG
HT-MF
L-STF
L-LTF
L-SIG
HT-SIG1
HT-SIG2
HT-GF
HT-STF
HT-LTF1
HT-SIG1
HT-SIG2
VHT
L-STF
L-LTF
L-SIG
VHT-SIGA1
VHT-SIGA2
sym1
sym2
sym3
Detect format using these symbols
Modulation Scheme
BPSK
QBPSK



Version History

Introduced in R2016b

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`wlanLLTFChannelEstimate` | `wlanLSIGRecover` | `wlanHTSIGRecover` | `wlanHESIGABitRecover` | `wlanVHTSIGARecover`

Objects

`wlanHERecoveryConfig`

wlanGolaySequence

Generate Golay sequence

Syntax

```
[Ga,Gb] = wlanGolaySequence(len)
```

Description

`[Ga,Gb] = wlanGolaySequence(len)` returns the Golay sequences `Ga` and `Gb` for a specified sequence length. The sequences are defined in IEEE 802.11ad-2012, Section 21.11.

Examples

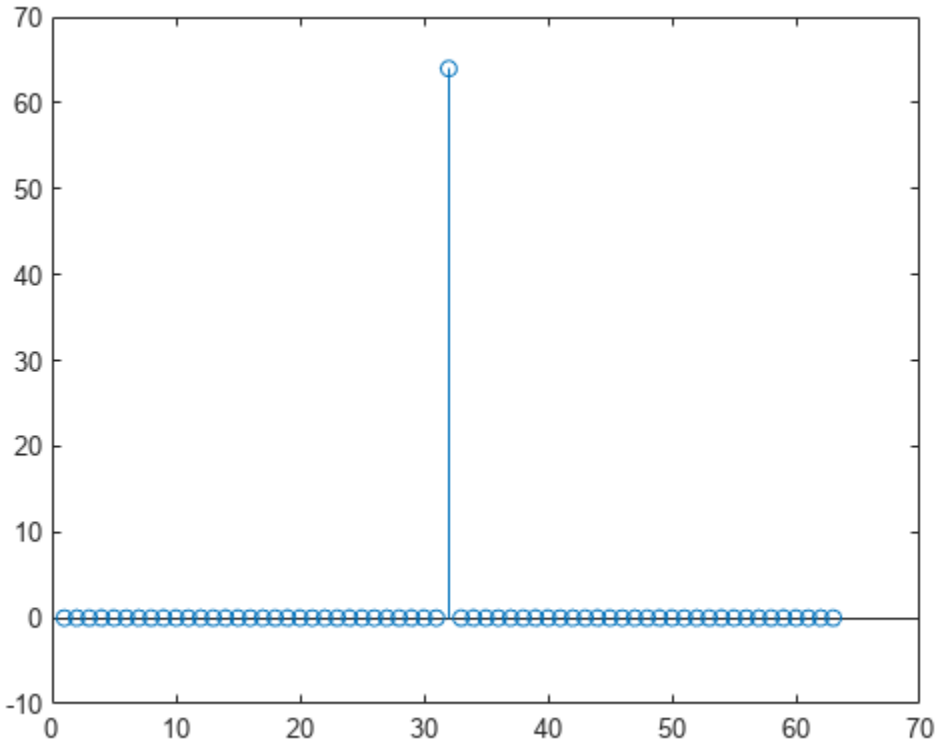
Generate Golay Sequences

Generate complementary 32-length Golay sequences.

```
[Ga,Gb] = wlanGolaySequence(32);
```

The sum of the autocorrelations is a dirac delta function.

```
figure
stem(xcorr(Ga)+xcorr(Gb))
```



Input Arguments

`len` — Sequence length
32 | 64 | 128

Sequence length, specified as 32, 64, or 128.

Data Types: `double`

Output Arguments

`Ga` — Golay sequence
column vector of integers

Golay sequence, returned as a column vector of integers of length `len`.

`Gb` — Complementary Golay sequence
column vector of integers

Complementary Golay sequence, returned as a column vector of integers of length `len`.

Version History

Introduced in R2017b

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanDMGDataBitRecover` | `wlanDMGConfig`

wlanHEDataBitRecover

Recover bits from HE-Data field

Syntax

```
dataBits = wlanHEDataBitRecover(rxDataSym,noiseVarEst,cfgHE)
dataBits = wlanHEDataBitRecover(rxDataSym,noiseVarEst,csi,cfgHE)

dataBits = wlanHEDataBitRecover(rxDataSym,noiseVarEst,cfgHE,userIdx)
dataBits = wlanHEDataBitRecover(rxDataSym,noiseVarEst,csi,cfgHE,userIdx)

dataBits = wlanHEDataBitRecover( ___ ,Name,Value)
```

Description

`dataBits = wlanHEDataBitRecover(rxDataSym,noiseVarEst,cfgHE)` recovers `dataBits`, a column vector of bits, from `rxDataSym`, the equalized OFDM symbols that comprise the HE-Data field of a high-efficiency single-user (HE SU) transmission. The function recovers `dataBits` by using noise variance estimate `noiseVarEst` and HE transmission parameters `cfgHE`.

`dataBits = wlanHEDataBitRecover(rxDataSym,noiseVarEst,csi,cfgHE)` enhances the demapping of OFDM subcarriers by using `csi`, a vector that contains channel state information (CSI).

`dataBits = wlanHEDataBitRecover(rxDataSym,noiseVarEst,cfgHE,userIdx)` recovers `dataBits` for one user, specified by user index `userIdx`, in a high-efficiency multi-user (HE MU) transmission.

`dataBits = wlanHEDataBitRecover(rxDataSym,noiseVarEst,csi,cfgHE,userIdx)` recovers `dataBits` for the specified user in an HE MU transmission and enhances the demapping of OFDM subcarriers by using channel state information

`dataBits = wlanHEDataBitRecover( ___ ,Name,Value)` specifies algorithm options by using one or more name-value pair arguments, in addition to any input argument combination from previous syntaxes. For example, `'LDPCDecodingMethod','layered-bp'` specifies the layered belief propagation low-density parity-check (LDPC) decoding algorithm.

Examples

Recover HE-Data Field from HE SU Transmission

Recover bits from the HE-Data field of an HE SU transmission.

Configure an HE SU transmission by creating a configuration object with the specified modulation and coding scheme (MCS). Extract the channel bandwidth.

```
cfgHESU = wlanHESUConfig('MCS',0);
cbw = cfgHESU.ChannelBandwidth;       % Channel bandwidth of transmission
```

Create a sequence of data bits and generate an HE SU waveform.

```
bits = randi([0 1],8*getPSDULength(cfgHESU),1,'int8');
waveform = wlanWaveformGenerator(bits,cfgHESU);
```

Create a WLAN recovery configuration object, specifying the known channel bandwidth and packet format.

```
cfgRX = wlanHERecoveryConfig('ChannelBandwidth',cbw,'PacketFormat','HE-SU');
```

Recover the HE signaling fields by retrieving the field indices and performing the relevant demodulation operations.

```
ind = wlanFieldIndices(cfgRX);
heLSIGandRLSIG = waveform(ind.LSIG(1):ind.RLSIG(2),:);
symLSIG = wlanHEDemodulate(heLSIGandRLSIG,'L-SIG',cbw);
info = wlanHEOFDMInfo('L-SIG',cbw);
```

Merge the L-SIG and RL-SIG fields for diversity and obtain the data subcarriers.

```
symLSIG = mean(symLSIG,2);
lsig = symLSIG(info.DataIndices,:);
```

Decode the L-SIG field, assuming a noiseless channel, and use the length field to update the recovery object.

```
noiseVarEst = 0;
[~,~,lsigInfo] = wlanLSIGBitRecover(lsig,noiseVarEst);
cfgRX.LSIGLength = lsigInfo.Length;
```

Recover and demodulate the HE-SIG-A field, obtain the data subcarriers, and recover the HE-SIG-A bits.

```
heSIGA = waveform(ind.HESIGA(1):ind.HESIGA(2),:);
symSIGA = wlanHEDemodulate(heSIGA,'HE-SIG-A',cbw);
siga = symSIGA(info.DataIndices,:);
[sigaBits,failCRC] = wlanHESIGABitRecover(siga,0);
```

Update the recovery configuration object with the recovered HE-SIG-A bits and obtain the updated field indices.

```
cfgHE = interpretHESIGABits(cfgRX,sigaBits);
ind = wlanFieldIndices(cfgHE);
```

Retrieve and decode the HE-Data field.

```
heData = waveform(ind.HEData(1):ind.HEData(2),:);
symData = wlanHEDemodulate(heData,'HE-Data', ...
    cbw,cfgHE.GuardInterval,[cfgHE.RUSize cfgHE.RUIndex]);
infoData = wlanHEOFDMInfo('HE-Data',cbw,cfgHE.GuardInterval,[cfgHE.RUSize cfgHE.RUIndex]);
rxDataSym = symData(infoData.DataIndices,:,:);
dataBits = wlanHEDataBitRecover(rxDataSym,noiseVarEst,cfgHE);
```

Confirm that the recovered bits match the transmitted bits.

```
isequal(bits,dataBits)
```

```
ans = logical
   1
```

Recover HE-Data Field from HE SU Transmission in AWGN Channel

Recover bits from the HE-Data field of an HE SU waveform transmitted though an additive white Gaussian noise (AWGN) channel.

```
cfgHE = wlanHESUConfig('MCS',11);
```

Generate a transmit waveform containing eight data packets.

```
psduLength = 8*getPSDULength(cfgHE);
bits = randi([0 1],psduLength,1,'int8');
waveform = wlanWaveformGenerator(bits,cfgHE);
```

Set a signal-to-noise ratio (SNR) of 30 dB, and pass the waveform through an AWGN channel.

```
snr = 30;
rx = awgn(waveform,snr);
```

Extract the HE-Data field from the received waveform.

```
ind = wlanFieldIndices(cfgHE);
rxData = rx(ind.HEData(1):ind.HEData(2),:);
```

Perform OFDM demodulation on the received HE-Data field.

```
sym = wlanHEDemodulate(rxData,'HE-Data',cfgHE);
```

Obtain the data subcarriers from the received symbols.

```
info = wlanHEOFDMInfo('HE-Data',cfgHE);
rxDataSym = sym(info.DataIndices);
```

Recover the bits from the HE-Data field for the appropriate noise variance estimate, assuming a CSI estimate of ones.

```
csi = ones(length(rxDataSym),1);    % Assume CSI estimate of all ones
noiseVarEst = 10^(-snr/10);         % Noise variance estimate
dataBits = wlanHEDataBitRecover(rxDataSym,noiseVarEst,csi,cfgHE);
```

Confirm that the recovered bits match the transmitted bits.

```
isequal(bits,dataBits)
```

```
ans = logical
   1
```

Recover Bits from HE-Data Field of HE MU Transmission

Recover bits from the HE-Data field of an HE MU waveform transmitted through an AWGN channel.

Configure an HE MU transmission for two users, specifying a channel bandwidth of 20 MHz and two 106-tone resource units (RUs).

```
AllocationIndex = 96;
cfgHE = wlanHEMUConfig(AllocationIndex);
```

Specify the MCS for both users and an APEP length for the second user.

```
cfgHE.User{1}.MCS = 4;
cfgHE.User{2}.APEPLength = 1e3;
cfgHE.User{2}.MCS = 7;
```

Generate a random PSDU for each user.

```
numUsers = numel(cfgHE.User);
psduLength = getPSDULength(cfgHE);
bits = cell(1,numUsers);
for i = 1:numUsers
   bits{i} = randi([0 1],8*psduLength(i),1);
end
```

Generate an OFDMA waveform and transmit through an AWGN channel for the specified SNR.

```
waveform = wlanWaveformGenerator(bits,cfgHE);
snr = 25;
noiseVarEst = 10^(-snr/10);
rx = awgn(waveform,snr);
```

Extract the HE-Data field from the received waveform.

```
ind = wlanFieldIndices(cfgHE);
rxData = rx(ind.HEData(1):ind.HEData(2),:);
```

Perform OFDM demodulation on the received HE-Data field for each RU, obtain the data subcarriers, and recover the bits for each user.

```
allocationInfo = ruInfo(cfgHE);
for userIdx = 1:allocationInfo.NumUsers
    ruNumber = allocationInfo.RUNumbers(userIdx);
    sym = wlanHEDemodulate(rxData,'HE-Data',cfgHE,ruNumber);
```

Because this example does not use equalization, we must scale the received symbols by a scaling factor equal to the ratio of the total number of tones to the number of tones in the RU of interest.

```
    sf = sqrt(sum(allocationInfo.RUSizes)/allocationInfo.RUSizes(userIdx));
    symScaled = sf*sym;
```

Extract the data subcarriers.

```
    ofdmInfo = wlanHEOFDMInfo('HE-Data',cfgHE,ruNumber);
    rxDataSym = symScaled(ofdmInfo.DataIndices,:,:);
```

Assume a CSI estimate of all ones and recover the bits, confirming that the recovered bits match the transmitted bits.

```
    csi = ones(length(rxDataSym),1);
    dataBits = wlanHEDataBitRecover(rxDataSym,noiseVarEst,csi,cfgHE,userIdx);
    disp(isequal(dataBits,bits{userIdx}))
end
```

```
   1

   1
```

Recover Bits from HE-Data Field from HE TB Transmission

Recover bits from the HE-Data field of an HE TB waveform transmitted through an AWGN channel.

Generate a WLAN HE TB waveform in response to a frame containing the TRS control subfield.

```
cfgHE = getTRSConfiguration(wlanHETBConfig);
psduLength = 8*getPSDULength(cfgHE);
bits = randi([0,1],psduLength,1,'int8');
waveform = wlanWaveformGenerator(bits,cfgHE);
```

Pass the waveform through an AWGN channel with an SNR of 30 dB.

```
snr = 30;
rx = awgn(waveform,snr);
```

Extract the HE-Data field from the received waveform.

```
ind = wlanFieldIndices(cfgHE);
rxData = rx(ind.HEData(1):ind.HEData(2),:);
```

Demodulate the waveform and extract the data subcarriers.

```
demod = wlanHEDemodulate(rxData,'HE-Data',cfgHE);
info = wlanHEOFDMInfo('HE-Data',cfgHE);
rxDataSym = demod(info.DataIndices,:,:);
```

Recover the data bits subject to the specified estimates for CSI and noise variance, implementing normalized min-sum low-density parity-check (LDPC) decoding.

```
csi = ones(length(rxDataSym),1);
noiseVarEst = 10^(-snr/10);
dataBits = wlanHEDataBitRecover(rxDataSym,noiseVarEst,csi,cfgHE, ...
    'LDPCDecodingMethod','norm-min-sum');
```

Confirm that the recovered information bits match the transmitted PSDU.

```
isequal(dataBits,bits)
```

```
ans = logical
   1
```

Input Arguments

`rxDataSym` — Demodulated HE-Data field for a user

complex-valued array

Demodulated HE-Data field for a user, specified as a complex-valued array of size N_{SD}-by-N_{Sym}-by-N_{SS}.

- N_{SD} is the number of data subcarriers in the HE-Data field.
- N_{Sym} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.

The contents and size of this input depend on the HE format specified in the `cfgHE` input.

Data Types: `double`
Complex Number Support: Yes

`noiseVarEst` — Noise variance estimate
nonnegative scalar

Noise variance estimate, specified as a nonnegative scalar.

Data Types: `double`

`csi` — Channel state information
real-valued array

Channel state information, specified as a real-valued array of size N_{SD}-by-N_{SS}.

- N_{SD} is the number of data subcarriers in the HE-Data field.
- N_{SS} is the number of spatial streams.

Data Types: `double`

`cfgHE` — HE transmission configuration
`wlanHESUConfig` object | `wlanHEMUConfig` object | `wlanHETBConfig` object | `wlanHERecoveryConfig` object

HE transmission configuration, specified as an object of type `wlanHESUConfig`, `wlanHEMUConfig`, `wlanHETBConfig`, or `wlanHERecoveryConfig`.

`userIdx` — User index
integer in the interval [1, 8]

User index, specified as an integer in the interval [1, 8].

Data Types: `double`

Name-Value Pair Arguments

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

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

Example: `'MaximumLDPCIterationCount','12','EarlyTermination','false'` specifies a maximum of 12 LDPC decoding iterations and disables early termination so that the decoder completes the 12 iterations.

`LDPCDecodingMethod` — LDPC decoding algorithm
`'bp'` (default) | `'layered-bp'` | `'norm-min-sum'` | `'offset-min-sum'`

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

- `'bp'` — Use the belief propagation (BP) decoding algorithm. For more information, see “Belief Propagation Decoding” on page 3-184.
- `'layered-bp'` — Use the layered BP decoding algorithm, suitable for quasi-cyclic parity check matrices (PCMs). For more information, see “Layered Belief Propagation Decoding” on page 3-185.
- `'norm-min-sum'` — Use the layered BP decoding algorithm with the normalized min-sum approximation. For more information, see “Normalized Min-Sum Decoding” on page 3-186.
- `'offset-min-sum'` — Use the layered BP decoding algorithm with the offset min-sum approximation. For more information, see “Offset Min-Sum Decoding” on page 3-186.

Note When you specify this input as `'norm-min-sum'` or `'offset-min-sum'`, the function sets input log-likelihood ratio (LLR) values that are greater than `1e10` or less than `-1e10` to `1e10` and `-1e10`, respectively. The function then uses these values when executing the LDPC decoding algorithm.

Dependencies

To enable this argument, specify the `ChannelCoding` property of the `cfgHE` input as `'LDPC'` for the user corresponding to the `userIdx` input.

Data Types: `char` | `string`

`MinSumScalingFactor` — Scaling factor for normalized min-sum LDPC decoding

`0.75` (default) | scalar in the interval (0, 1]

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

Dependencies

To enable this argument, specify the `'LDPCDecodingMethod'` name-value pair argument as `'norm-min-sum'`.

Data Types: `double`

`MinSumOffset` — Offset for offset min-sum LDPC decoding

`0.5` (default) | nonnegative scalar

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

Dependencies

To enable this argument, specify the `'LDPCDecodingMethod'` name-value pair argument as `'offset-min-sum'`.

Data Types: `double`

`MaximumLDPCIterationCount` — Maximum number of LDPC decoding iterations

12 (default) | positive integer

Maximum number of LDPC decoding iterations, specified as the comma-separated pair consisting of `'MaximumLDPCIterationCount'` and a positive integer.

Dependencies

To enable this argument, set the `ChannelCoding` property of the `cfgHE` input to `'LDPC'` for the user corresponding to the `userIdx` input.

Data Types: `double`

`EarlyTermination` — Enable early termination of LDPC decoding

`false` or `0` (default) | `true` or `1`

Enable early termination of LDPC decoding, specified as the comma-separated pair consisting of `'EarlyTermination'` and `1` (`true`) or `0` (`false`).

- When you set this value to `0` (`false`), LDPC decoding completes the number of iterations specified in the `'MaximumLDPCIterationCount''` name-value pair argument regardless of parity check status.
- When you set this value to `1` (`true`), LDPC decoding terminates when all parity checks are satisfied.

Dependencies

To enable this argument, set the `ChannelCoding` property of the `cfgHE` input to `'LDPC'` for the user corresponding to the `userIdx` input.

Data Types: `logical`

Output Arguments

`dataBits` — Bits recovered from HE-Data field

binary-valued column vector

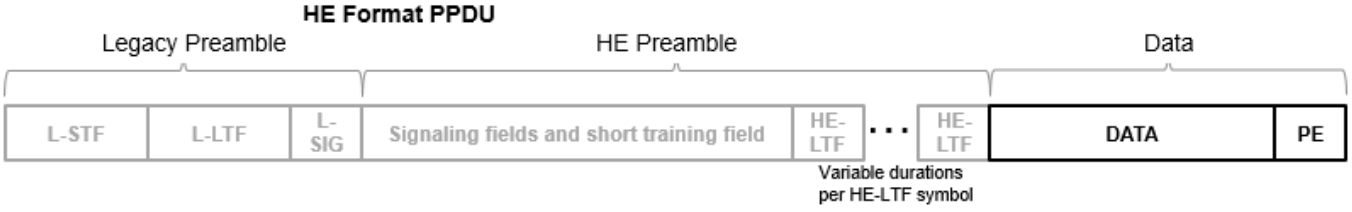
Bits recovered from the HE-Data field, returned as a binary valued column vector of length $8 \times L_{\text{PSDU}}$, where L_{PSDU} is the PSDU length in bytes. Calculate the PSDU length by using the `getPSDULength` object function with the `cfgHE` input.

Data Types: `int8`

More About

HE-Data Field

The HE-Data field of the HE PPDU contains data for one or more users.



As described in [1], the number of OFDM symbols in the HE-Data field depends on the length value of the legacy signal (L-SIG) field in accordance with equation (27-11), the preamble duration, and the settings of the GI+LTF Size, Pre-FEC Padding Factor, and PE Disambiguity subfields of the HE-SIG-A field in accordance with section 27.3.10.7.

- Data symbols in an HE PPDU use a discrete Fourier transform (DFT) period of 12.8 µs and subcarrier spacing of 78.125 kHz.
- Data symbols in an HE PPDU support GI durations of 0.8 µs, 1.6 µs, and 3.2 µs.
- HE PPDUs have single-stream pilots in the HE-Data field.

When the transmission uses BCC encoding, the HE-Data field consists of the SERVICE field, the PSDU, the pre-FEC padding bits, the tail bits, and the post-FEC padding bits.

When the transmission uses LDPC encoding, the HE-Data field consists of the SERVICE field, the PSDU, the pre-FEC padding bits, the post-FEC padding bits, and the packet extension (PE) field.

For more information, see “WLAN PPDU Structure” and “802.11ax Waveform Generation”.

Algorithms

This function supports these four LDPC decoding algorithms.

Belief Propagation Decoding

The function implements the BP algorithm based on the decoding algorithm presented in [2]. For transmitted LDPC-encoded codeword $c = (c_0, c_1, \ldots, c_{n-1})$, the input to the LDPC decoder is the LLR given by

$$L(c_i) = \log\left(\frac{\Pr(c_i = 0 \,|\, \text{channel output for } c_i)}{\Pr(c_i = 0 \,|\, \text{channel output for } c_i)}\right).$$

In each iteration, the function updates the key components of the algorithm based on these equations:

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

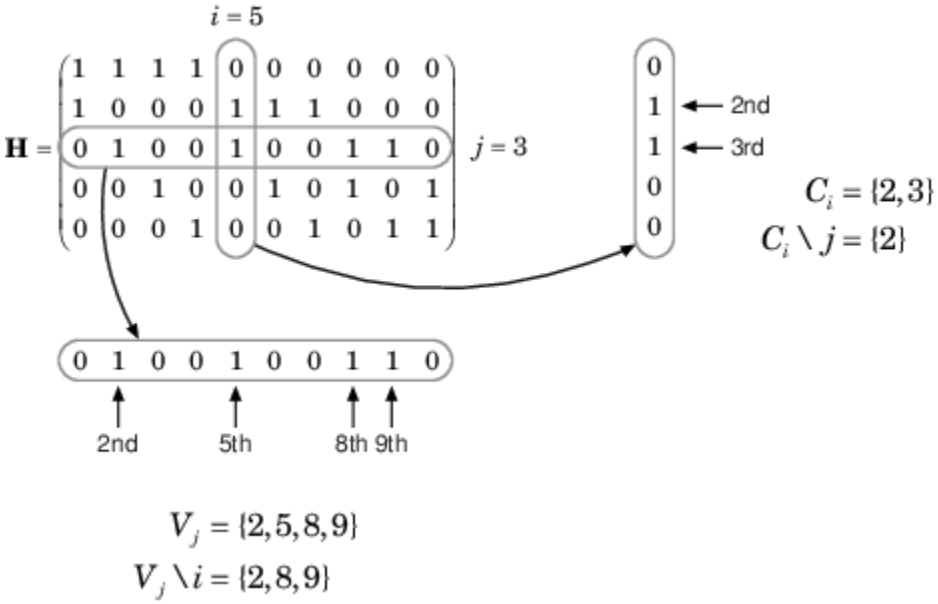
$L(q_{ij}) = L(c_i) + \sum_{j' \in C_i \backslash \{j\}} L(r_{j'i})$, initialized as $L(q_{ij}) = L(c_i)$ before the first iteration, and

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

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

Index sets $C_i \backslash \{j\}$ and $V_j \backslash \{i\}$ are based on the PCM such that the sets C_i and V_j correspond to all nonzero elements in column i and row j of the PCM, respectively.

This figure demonstrates how to compute these index sets for PCM **H** for the case $i = 5$ and $j = 3$.



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

When you specify the `'EarlyTermination'` name-value pair argument as `0` (`false`), the decoding terminates after the number of iterations specified by the `'MaximumLDPCIterationCount'` name-value pair argument. When you specify the `'EarlyTermination'` name-value pair argument as `1` (`true`), the decoding terminates when all parity checks are satisfied ($\mathbf{Hc}^T = 0$) or after the number of iterations specified by the `'MaximumLDPCIterationCount'` name-value pair argument.

Layered Belief Propagation Decoding

The function implements the layered BP algorithm based on the decoding algorithm presented in Section II.A of [3]. The decoding loop iterates over subsets of rows (layers) of the PCM.

For each row, m, in a layer and each bit index, j, the implementation updates the key components of the algorithm based on these equations.

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

(2) $\Psi(x) = \log(|\tanh(x/2)|)$

(3) $A_{mj} = \sum_{n \in N(m)\setminus\{j\}} \Psi(L(q_{mn}))$

(4) $s_{mj} = \prod_{n \in N(m)\setminus\{j\}} \mathrm{sgn}(L(q_{mn}))$

(5) $R_{mj} = -s_{mj}\Psi(A_{mj})$

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

For each layer, the decoding equation (6) works on the combined input obtained from the current LLR inputs, $L(q_{mj})$, and the previous layer updates, R_{mj}.

Because the layered BP algorithm updates only a subset of the nodes in a layer, this algorithm is faster than the BP algorithm. To achieve the same error rate as attained with BP decoding, use half the number of decoding iterations when using the layered BP algorithm.

Normalized Min-Sum Decoding

The function implements the normalized min-sum decoding algorithm by following the layered BP algorithm with equation (3) replaced by

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

where α is the scaling factor specified by the `'MinSumScalingFactor'` name-value pair argument. This equation is an adaptation of equation (4) presented in [4].

Offset Min-Sum Decoding

The function implements the offset min-sum decoding algorithm by following the layered BP algorithm with equation (3) replaced by

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

where β is the offset specified by the `'MinSumOffset'` name-value pair argument. This equation is an adaptation of equation (5) presented in [4].

Version History

Introduced in R2018b

References

[1] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

[2] Gallager, Robert G. *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.

[3] Hocevar, D.E. "A Reduced Complexity Decoder Architecture via Layered Decoding of LDPC Codes." In *IEEE Workshop on Signal Processing Systems, 2004. SIPS 2004.*, 107-12. Austin, Texas, USA: IEEE, 2004. https://doi.org/10.1109/SIPS.2004.1363033.

[4] Jinghu Chen, R.M. Tanner, C. Jones, and Yan Li. "Improved Min-Sum Decoding Algorithms for Irregular LDPC Codes." In *Proceedings. International Symposium on Information Theory, 2005. ISIT 2005.*, 449-53, 2005. https://doi.org/10.1109/ISIT.2005.1523374.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanHESUConfig` | `wlanHEMUConfig` | `wlanHERecoveryConfig` | `wlanHETBConfig`

wlanHEDemodulate

Demodulate fields of HE waveform

Syntax

```
sym = wlanHEDemodulate(rx,field,cfg)
sym = wlanHEDemodulate(rx,field,cfg,ruNumber)

sym = wlanHEDemodulate(rx,field,cbw,hegi,ru)
sym = wlanHEDemodulate(rx,field,cbw,hegi,ltfType,ru)
sym = wlanHEDemodulate(rx,field,cbw)

sym = wlanHEDemodulate(___,'OFDMSymbolOffset',symOffset)
```

Description

`sym = wlanHEDemodulate(rx,field,cfg)` recovers a demodulated frequency-domain signal by orthogonal frequency-division multiplexing (OFDM) demodulating received high-efficiency (HE) time-domain signal `rx`. The function demodulates `rx` by using HE transmission parameters `cfg` and signal field value `field`.

`sym = wlanHEDemodulate(rx,field,cfg,ruNumber)` specifies the number of a resource unit. To demodulate either the HE-Data field or the HE long training field (HE-LTF), use this syntax.

`sym = wlanHEDemodulate(rx,field,cbw,hegi,ru)` specifies channel bandwidth `cbw`, guard interval `hegi`, and resource unit determined by the size and index specified in `ru`. If `ru` is not specified, the function returns the demodulated signal assuming a full band configuration. To demodulate the HE-Data field when the PHY format is unknown, use this syntax.

`sym = wlanHEDemodulate(rx,field,cbw,hegi,ltfType,ru)` specifies the HE-LTF type. If `ru` is not specified, `wlanHEDemodulate` returns the demodulated signal assuming a full band configuration. To demodulate the HE-LTF when the PHY format is unknown, use this syntax.

`sym = wlanHEDemodulate(rx,field,cbw)` recovers the frequency-domain signal for the specified field and channel bandwidth. To demodulate the L-LTF, L-SIG, RL-SIG, HE-SIG-A, or HE-SIG-B field when the PHY format configuration is unknown, use this syntax.

`sym = wlanHEDemodulate(___,'OFDMSymbolOffset',symOffset)` specifies the OFDM symbol sampling offset as a fraction of the cyclic prefix length in addition to any combination of arguments from the previous syntaxes.

Examples

Demodulate HE-SIG-A Field and Get OFDM Information

Perform OFDM demodulation on the HE-SIG-A field and extract the data and pilot subcarriers.

Generate a WLAN waveform for an HE SU transmission.

```
cfg = wlanHESUConfig;
bits = [1; 0; 0; 1];
waveform = wlanWaveformGenerator(bits,cfg);
```

Obtain the field indices and extract the HE-SIG-A field.

```
ind = wlanFieldIndices(cfg);
rx = waveform(ind.HESIGA(1):ind.HESIGA(2),:);
```

Perform OFDM demodulation on the HE-SIG-A field.

```
sym = wlanHEDemodulate(rx,'HE-SIG-A',cfg);
```

Get the OFDM information, then extract the data and pilot subcarriers.

```
info = wlanHEOFDMInfo('HE-SIG-A',cfg);
data = sym(info.DataIndices,:,:);
pilots =  sym(info.PilotIndices,:,:);
```

Demodulate HE-LTF for RUs in HE MU Waveform

Demodulate the HE-LTF for each RU in an HE MU waveform.

Create a WLAN HE-MU-format configuration object, specifying the allocation index, HE-LTF type, and guard interval.

```
AllocationIndex = 16;
cfg = wlanHEMUConfig(16,'HELTFType',2,'GuardInterval',1.6);
```

Generate a waveform for the specified information bits and format configuration object.

```
bits = [1; 0; 0; 1];
waveform = wlanWaveformGenerator(bits,cfg);
```

Generate field indices and extract the HE-LTF.

```
ind = wlanFieldIndices(cfg);
rx = waveform(ind.HELTF(1):ind.HELTF(2),:);
```

Demodulate the HE-LTF for each RU and display the size of the array containing the demodulated symbols in each case.

```
info = ruInfo(cfg);
allRUs = info.NumRUs;
for ruNumber = 1:allRUs
    sym = wlanHEDemodulate(rx,'HE-LTF',cfg,ruNumber);
    disp(size(sym));
end
```

```
    52     1

    52     1

   106     1
```

Demodulate L-LTF

Perform OFDM demodulation on the legacy long training field (L-LTF) of a received signal, specifying a channel bandwidth of 80 MHz.

Retrieve the L-LTF from a very-high-throughput (VHT) waveform with a channel bandwidth of 80 MHz.

```
cbw = 'CBW80'; % Specify the channel bandwidth
rx = wlanLLTF(wlanVHTConfig('ChannelBandwidth',cbw));
```

Get the frequency-domain signal by demodulating the L-LTF.

```
sym = wlanHEDemodulate(rx,'L-LTF',cbw);
```

Input Arguments

`rx` — Received time-domain signal

complex-valued matrix

Received time-domain signal, specified as a complex-valued matrix of size N_s-by-N_r.

- N_s is the number of time-domain samples. If N_s is not an integer multiple of the OFDM symbol length, L_s, for the specified field,then the function ignores the remaining $\mathtt{mod}(N_s, L_s)$ symbols.
- N_r is the number of receive antennas.

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

`field` — Field to be demodulated

`'L-LTF'` | `'L-SIG'` | `'RL-SIG'` | `'HE-SIG-A'` | `'HE-SIG-B'` | `'HE-LTF'` | `'HE-Data'`

Field to be demodulated, specified as one of these values:

- `'L-LTF'` – Demodulate the legacy long training field (L-LTF).
- `'L-SIG'` – Demodulate the legacy signal (L-SIG) field.
- `'RL-SIG'` – Demodulate the repeated legacy signal (RL-SIG) field.
- `'HE-SIG-A'` – Demodulate the HE signal A (HE-SIG-A) field.
- `'HE-SIG-B'` – Demodulate the HE signal B (HE-SIG-B) field.
- `'HE-LTF'` – Demodulate the HE long training field (HE-LTF).
- `'HE-Data'` – Demodulate the HE-Data field.

Data Types: `char` | `string`

`cfg` — PHY format configuration

`wlanHESUConfig` object | `wlanHEMUConfig` object | `wlanHETBConfig` object | `wlanHERecoveryConfig` object

Physical layer (PHY) format configuration, specified as an object of type `wlanHESUConfig`, `wlanHEMUConfig`, `wlanHETBConfig`, or `wlanHERecoveryConfig`.

When you specify this input as a `wlanHETBConfig` object with the `FeedbackNDP` property set to `1` (`true`), the function interleaves the symbols for active and complementary tone sets for the value of the `RUToneSetIndex` property in accordance with Table 27-32 of [1].

`ruNumber` — Number of RU of interest
positive integer

Number of the RU of interest, specified as a positive integer. The RU number specifies the location of the RU within the channel. For example, consider an 80-MHz transmission with two 242-tone RUs and one 484-tone RU, in order of absolute frequency. For this allocation:

- RU number 1 corresponds to the 242-tone RU in the 20-MHz subchannel at the lowest absolute frequency (size 242, index 1).
- RU number 2 corresponds to the 242-tone RU in the 20-MHz subchannel at the next lowest absolute frequency (size 242, index 2).
- RU number 3 corresponds to the 484-tone RU in the 40-MHz subchannel at the highest absolute frequency (size 484, index 2).

Data Types: `single` | `double`

`cbw` — Channel bandwidth
`'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth, specified as one of these values.

- `'CBW20'` – Channel bandwidth of 20 MHz
- `'CBW40'` – Channel bandwidth of 40 MHz
- `'CBW80'` – Channel bandwidth of 80 MHz
- `'CBW160'` – Channel bandwidth of 160 MHz

Data Types: `char` | `string`

`hegi` — Guard interval duration
`0.8` | `1.6` | `3.2`

Guard interval duration, in microseconds, specified as `0.8`, `1.6`, or `3.2`.

Data Types: `single` | `double`

`ltfType` — HE-LTF type
`1` | `2` | `4`

HE-LTF type, specified as `1`, `2`, or `4`.

Data Types: `single` | `double`

`ru` — RU size and index
1-by-2 vector of positive scalars

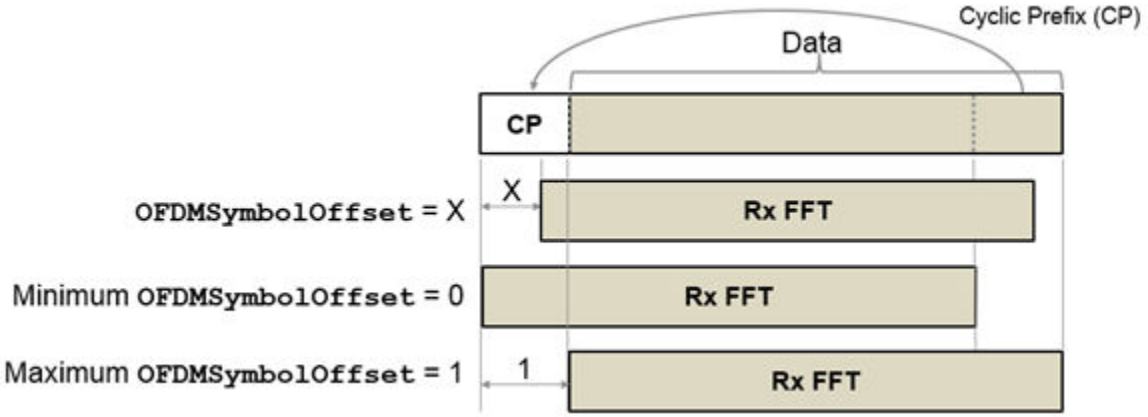
RU size and index, specified as a 1-by-2 vector of positive scalars. Specify `ru` in the form [*size*,*index*], where *size* must be 26, 52, 106, 242, 484, 996, or 1992 in accordance with the specified channel bandwidth. For example, an 80-MHz transmission has four possible 242-tone RUs (one for each 20-MHz subchannel). RU number 242-1 (*size* = 242 and *index* = 1) is the lowest absolute frequency within the 80-MHz channel. RU number 242-4 is the highest absolute frequency.

Data Types: `single` | `double`

`symOffset` — OFDM symbol sampling offset
`0.75` (default) | scalar in the interval [0, 1]

OFDM symbol sampling offset, as a fraction of the cyclic prefix length, specified as a scalar in the interval [0, 1].

The value that you specify indicates the start location for OFDM demodulation relative to the beginning of the cyclic prefix.



Example: `0.45`

Data Types: `single` | `double`

Output Arguments

`sym` — Demodulated frequency-domain signal
complex-valued array

Demodulated frequency-domain signal, returned as a complex-valued array of size N_{sc}-by-N_{sym}-by-N_r.

- N_{sc} is the number of active occupied subcarriers in the demodulated field.
- N_{sym} is the number of OFDM symbols.
- N_r is the number of receive antennas.

Data Types: `double`

Version History

Introduced in R2019a

Single precision support

You can specify the function's numeric inputs as single precision values.

References

[1] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`wlanDMGOFDMDemodulate` | `wlanHEOFDMInfo` | `wlanS1GDemodulate`

Objects

`wlanHESUConfig` | `wlanHEMUConfig` | `wlanHETBConfig`

wlanHELTFChannelEstimate

Channel estimation using HE-LTF

Syntax

```
chEst = wlanHELTFChannelEstimate(demodSig,cfg)
chEst = wlanHELTFChannelEstimate(demodSig,cfg,ruNumber)
[chEst,chEstPilots] = wlanHELTFChannelEstimate( ___ )
[chEst,chEstPilots] = wlanHELTFChannelEstimate( ___ ,FrequencySmoothingSpan=
span)
```

Description

`chEst = wlanHELTFChannelEstimate(demodSig,cfg)` returns the channel estimate at the high efficiency long training field (HE-LTF) using the demodulated HE-LTF signal, `demodSig`, given the parameters specified in the configuration object `cfg`. For more information about this field, see “HE-LTF” on page 3-198.

`chEst = wlanHELTFChannelEstimate(demodSig,cfg,ruNumber)` returns the channel estimate for the resource unit (RU) of interest for a multi-user (MU) configuration. Here, `cfg` must be specified as a `wlanHEMUConfig` object.

`[chEst,chEstPilots] = wlanHELTFChannelEstimate( ___ )` returns the channel estimate at each pilot subcarrier location for each demodulated HE-LTF OFDM symbol in addition to any input argument combination from the previous syntaxes.

`[chEst,chEstPilots] = wlanHELTFChannelEstimate( ___ ,FrequencySmoothingSpan= span)` specifies frequency smoothing using a name-value pair argument.

Examples

Channel Estimation Using HE-LTF

Calculate and plot the channel estimate of an HE SU format channel by using the high efficiency long training field.

Create an HE SU format configuration object. Generate a time-domain waveform for an 802.11ax packet.

```
cfg = wlanHESUConfig;
txSig = wlanWaveformGenerator([1;0;0;1],cfg);
```

Multiply the transmitted signal by 0.1 + 0.3i and pass it through an AWGN channel with a signal-to-noise ratio of 25 dB.

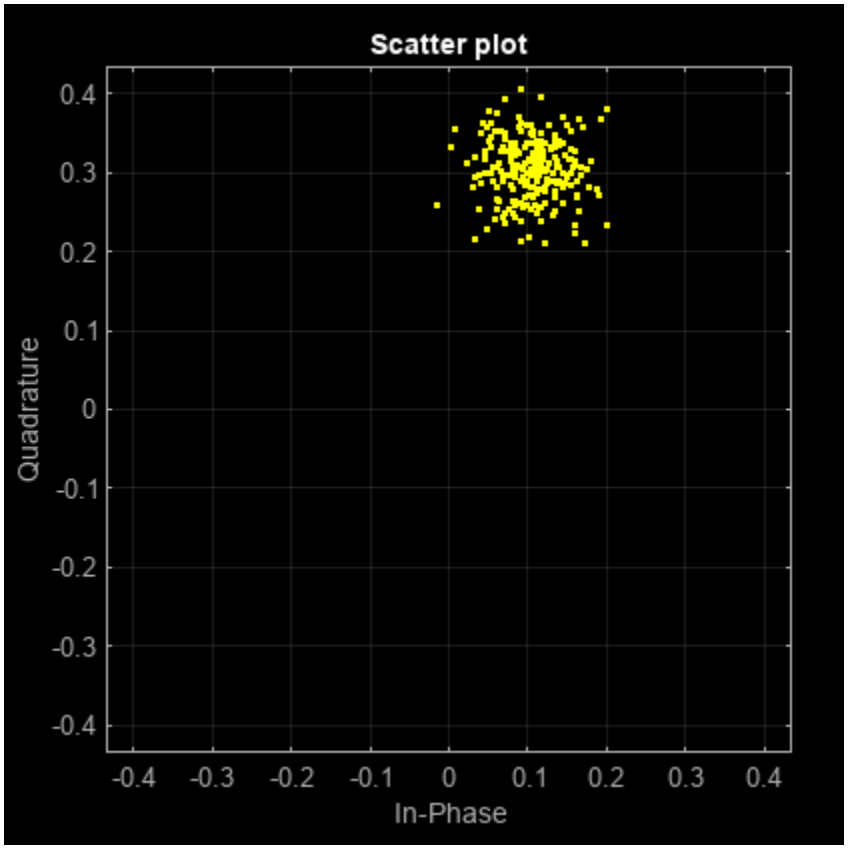
```
rxSig = awgn((0.1 + 0.3i)*txSig,25);
```

Extract the HE-LTF field indices and demodulate the HE-LTF. Perform channel estimation without frequency smoothing.

```
idxHELTF = wlanFieldIndices(cfg,"HE-LTF");
demodSig = wlanHEDemodulate(rxSig(idxHELTF(1):idxHELTF(2),:),"HE-LTF",cfg);
est = wlanHELTFChannelEstimate(demodSig,cfg);
```

Plot the channel estimate.

```
scatterplot(est)
grid
```



The channel estimate matches the complex channel multiplier.

MIMO Channel Estimation Using HE-LTF

Calculate and plot the channel estimate of an HE SU format MIMO channel by using the high efficiency long training field.

Create an HE SU format configuration format object, with a channel bandwidth of 160 MHz and two transmit antennas and two space-time streams. Generate a time-domain waveform for an 802.11ax packet.

```
cfg = wlanHESUConfig(NumSpaceTimeStreams=2, NumTransmitAntennas=2);
cfg.ChannelBandwidth = "CBW160";
txSig = wlanWaveformGenerator([1;0;0;1],cfg);
```

Create a TGax channel system object with a channel bandwidth of 160 MHz and two transmit and receive antennas. Pass the transmitted signal through the TGax MIMO channel.

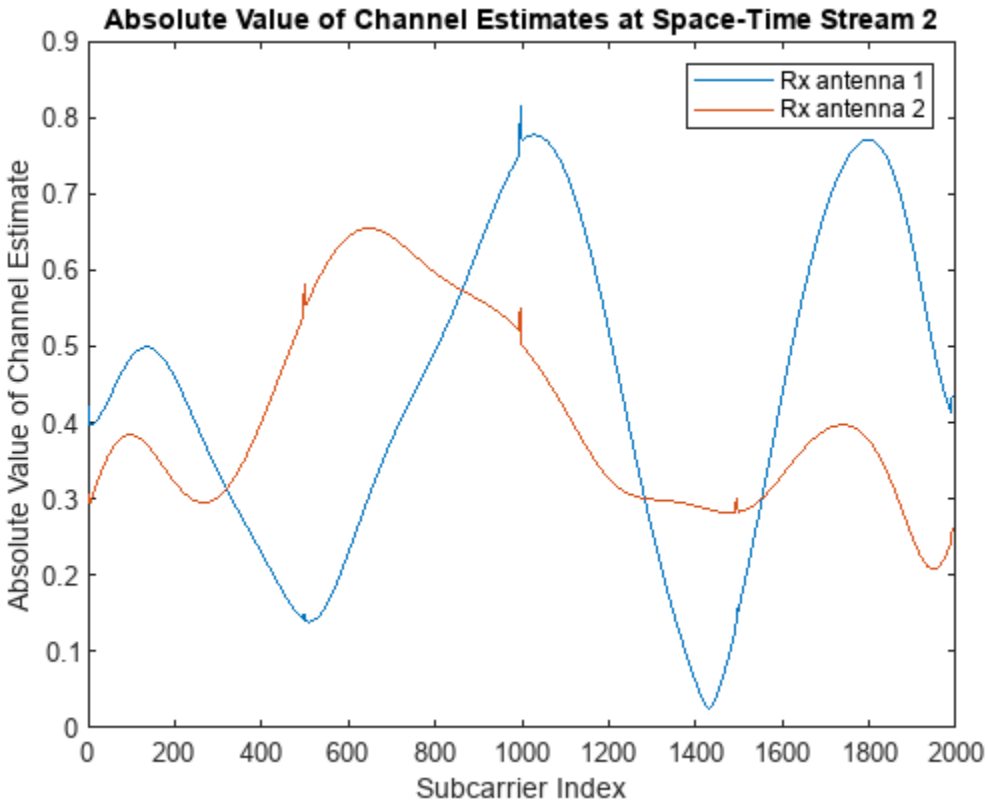
```
tgax = wlanTGaxChannel;
tgax.ChannelBandwidth = "CBW160";
tgax.NumReceiveAntennas = 2;
tgax.NumTransmitAntennas = 2;
rxSig =  tgax(txSig);
```

Extract the HE-LTF field indices and demodulate the HE-LTF. Using a filter span of 7, perform channel estimation at each pilot subcarrier location with frequency smoothing.

```
idxHELTF = wlanFieldIndices(cfg,"HE-LTF");
demodSig = wlanHEDemodulate(rxSig(idxHELTF(1):idxHELTF(2),:),"HE-LTF",cfg);
[est,estPilots] = wlanHELTFChannelEstimate(demodSig,cfg,FrequencySmoothingSpan=7);
```

Plot the absolute value of the channel estimate for the second of the two space-time streams at both receive antennas.

```
plot(abs(est(:,2,1)))
hold on
plot(abs(est(:,2,2)))
xlabel("Subcarrier Index")
title("Absolute Value of Channel Estimates at Space-Time Stream 2")
ylabel("Absolute Value of Channel Estimate")
legend("Rx antenna 1", "Rx antenna 2")
```



Input Arguments

`demodSig` — Demodulated HE-LTF signal

3-D array

Demodulated HE-LTF signal, specified as an N_{ST}-by-N_{SYM}-by-N_R array. N_{ST} is the number of occupied subcarriers, N_{SYM} is the number of demodulated HE-LTF OFDM symbols, and N_R is the number of receive antennas.

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

`cfg` — Format configuration

`wlanHEMUConfig` object | `wlanHERecoveryConfig` object | `wlanHESUConfig` object | `wlanHETBConfig` object

Format configuration, specified as one of these objects: `wlanHEMUConfig`, `wlanHESUConfig`, `wlanHERecoveryConfig`, or `wlanHETBConfig`.

`ruNumber` — RU number in HE MU transmission

1 (default) | positive integer

RU number in an HE MU transmission, specified as a positive integer. The RU number specifies the location of the RU within the channel. For example, consider an 80 MHz transmission with two 242-tone RUs and one 484-tone RU, in order of absolute frequency. For this allocation:

- RU number 1 corresponds to the 242-tone RU in the 20 MHz subchannel at the lowest absolute frequency (size 242, index 1).
- RU number 2 corresponds to the 242-tone RU in the 20 MHz subchannel at the next lowest absolute frequency (size 242, index 2).
- RU number 3 corresponds to the 484-tone RU in the 40 MHz subchannel at the highest absolute frequency (size 484, index 2).

Data Types: `single` | `double`

`span` — Frequency smoothing span

positive odd integer

Span of the frequency smoothing filter, specified as a positive odd integer. The span is expressed as a number of subcarriers. The function applies frequency smoothing when `span` is greater than one. For more information on when to specify this input, see “Frequency Smoothing” on page 3-198.

Output Arguments

`chEst` — Channel estimate

3-D array

Channel estimate between all combinations of space-time streams and receive antennas, returned as an N_{ST}-by-$N_{STS,total}$-by-N_R array. N_{ST} is the number of occupied subcarriers. $N_{STS,total}$ is the total number of space-time streams for all users. In the single-user case, $N_{STS,total}$=N_{STS}. In the multi-user case, $N_{STS,total}$ is the sum of the values of N_{STS} for each user of the RU of interest. N_R is the number of receive antennas.

Data Types: `single` | `double`

`chEstPilots` — Channel estimate for pilot subcarriers
3-D array

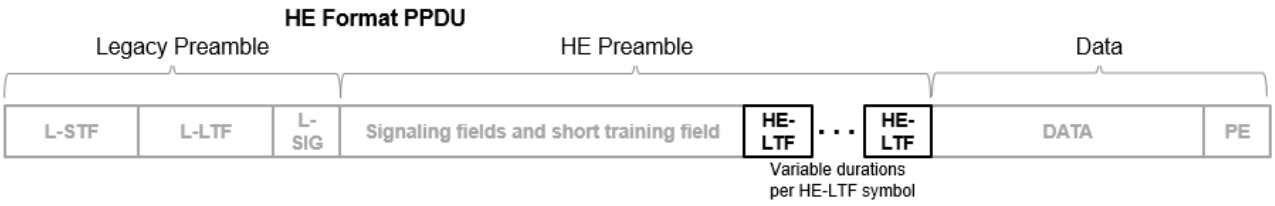
Channel estimate at each pilot subcarrier location for each HE-LTF symbol, returned as an N_{SP}-by-N_{SYM}-by-N_R array. N_{SP} is the number of pilot subcarriers, N_{SYM} is the number of demodulated HE-LTF OFDM symbols, and N_R is the number of receive antennas. The function performs this estimate assuming one space-time stream at the transmitter.

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

More About

HE-LTF

The HE-LTF is located between the HE-STF and data field of an HE packet.



As described in Section 27.3.11.10 of IEEE Std 802.11ax-2021, the receiver can use the HE-LTF to estimate the MIMO channel between the set of constellation mapper outputs (or, if STBC is applied, the STBC encoder outputs) and the receive chains. An HE PPDU supports three HE-LTF types: 1x HE-LTF, 2x HE-LTF, and 4x HE-LTF, which determine symbol durations of 3.2, 6.4, and 12.8 µs respectively. In these durations, guard intervals are omitted.

The number of HE-LTF OFDM symbols transmitted can be one, two, four, six, or eight. The following table, adapted from Table 21-13 in IEEE Std 802.11-2020, shows how the number of HE-LTF OFDM symbols depends on the number of space-time streams.

$N_{STS,total}$	N_{SYM}
1	1
2	2
3	4
4	4
5	6
6	6
7	8
8	8

Frequency Smoothing

Frequency smoothing can improve channel estimation by averaging out noise.

Frequency smoothing is recommended only for cases in which a single transmit antenna is used. Frequency smoothing consists of applying a moving-average filter that spans multiple adjacent subcarriers. Channel conditions dictate whether frequency smoothing is beneficial.

- If adjacent subcarriers are highly correlated, frequency smoothing results in significant noise reduction.
- In a highly frequency-selective channel, smoothing can degrade the quality of the channel estimate.

Version History

Introduced in R2022b

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

[2] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`wlanHEMUConfig` | `wlanHERecoveryConfig` | `wlanHESUConfig` | `wlanHETBConfig`

Functions

`wlanHEDataBitRecover` | `wlanHEDemodulate` | `wlanFieldIndices` | `wlanPreHEChannelEstimate`

wlanHEOFDMInfo

OFDM information for HE transmission

Syntax

```
info = wlanHEOFDMInfo(field,cfg)
info = wlanHEOFDMInfo(field,cfg,ruNumber)
info = wlanHEOFDMInfo(field,cbw,hegi,ru)
info = wlanHEOFDMInfo(field,cbw)
info = wlanHEOFDMInfo( ___ ,OversamplingFactor=osf)
```

Description

`info = wlanHEOFDMInfo(field,cfg)` returns `info`, a structure containing orthogonal frequency-division multiplexing (OFDM) information for the input field in a high-efficiency (HE) transmission parameterized by configuration object `cfg`.

`info = wlanHEOFDMInfo(field,cfg,ruNumber)` returns OFDM information for the resource unit (RU) of interest determined by its number `ruNumber`, when you specify `cfg` as an HE multi-user (HE MU) configuration. To return OFDM information for either the HE-Data field or the HE long training field (HE-LTF) in an HE MU transmission, use this syntax.

`info = wlanHEOFDMInfo(field,cbw,hegi,ru)` returns OFDM information for channel bandwidth `cbw`, guard interval `hegi`, and RU of interest determined by its size and index specified in `ru`. If you do not specify `ru`, then `wlanHEOFDMInfo` returns information assuming a full band configuration. To return OFDM information for either the HE-Data field or the HE-LTF when the PHY format configuration is unknown, use this syntax.

`info = wlanHEOFDMInfo(field,cbw)` returns OFDM information for the specified field and channel bandwidth. To return OFDM information for one of the L-LTF, L-SIG, RL-SIG, HE-SIG-A, or HE-SIG-B fields when the PHY format configuration is unknown, use this syntax.

`info = wlanHEOFDMInfo( ___ ,OversamplingFactor=osf)` returns OFDM information for the specified oversampling factor. For more information about oversampling, see “FFT-Based Oversampling” on page 3-204.

Examples

Demodulate HE-SIG-A Field and Get OFDM Information

Perform OFDM demodulation on the HE-SIG-A field and extract the data and pilot subcarriers.

Generate a WLAN waveform for an HE SU transmission.

```
cfg = wlanHESUConfig;
bits = [1; 0; 0; 1];
waveform = wlanWaveformGenerator(bits,cfg);
```

Obtain the field indices and extract the HE-SIG-A field.

```
ind = wlanFieldIndices(cfg);
rx = waveform(ind.HESIGA(1):ind.HESIGA(2),:);
```

Perform OFDM demodulation on the HE-SIG-A field.

```
sym = wlanHEDemodulate(rx,'HE-SIG-A',cfg);
```

Get the OFDM information, then extract the data and pilot subcarriers.

```
info = wlanHEOFDMInfo('HE-SIG-A',cfg);
data = sym(info.DataIndices,:,:);
pilots =  sym(info.PilotIndices,:,:);
```

Get OFDM Information for RUs in HE MU Waveform

Get OFDM information for each RU in an HE MU waveform.

Create a WLAN HE-MU-format configuration object with the allocation index set to 16.

```
AllocationIndex = 16;
cfg = wlanHEMUConfig(AllocationIndex);
```

Get the OFDM information pertaining to the HE-Data field for each RU and extract the number of active subcarriers in each case.

```
NumTones =  zeros(numel(cfg.RU),1);
for ruNumber = 1:numel(cfg.RU)
    info = wlanHEOFDMInfo('HE-Data',cfg,ruNumber);
    NumTones(ruNumber) = info.NumTones;
end
```

Display the number of active subcarriers for each RU.

```
disp(NumTones');
```

```
    52    52   106
```

Get OFDM Information for L-LTF

Get OFDM information for the legacy long training field (L-LTF) for a specified channel bandwidth.

Specify a channel bandwidth of 40 MHz.

```
cbw = 'CBW40';
```

Get the OFDM information for the L-LTF and display the fast Fourier transform (FFT) length.

```
info = wlanHEOFDMInfo('L-LTF',cbw);
disp(info.FFTLength);
```

```
   128
```

Input Arguments

`field` — Field for which to return OFDM information

`'L-LTF'` | `'L-SIG'` | `'RL-SIG'` | `'HE-SIG-A'` | `'HE-SIG-B'` | `'HE-LTF'` | `'HE-Data'`

Field for which to return OFDM information, specified as one of these values.

- `'L-LTF'` – Return OFDM information for the legacy long training field (L-LTF).
- `'L-SIG'` – Return OFDM information for the legacy signal (L-SIG) field.
- `'RL-SIG'` – Return OFDM information for the repeated legacy signal (RL-SIG) field.
- `'HE-SIG-A'` – Return OFDM information for the HE signal A (HE-SIG-A) field.
- `'HE-SIG-B'` – Return OFDM information for the HE signal B (HE-SIG-B) field.
- `'HE-LTF'` – Return OFDM information for the HE long training field (HE-LTF).
- `'HE-Data'` – Return OFDM information for the HE-Data field.

Data Types: `char` | `string`

`cfg` — PHY format configuration

`wlanHESUConfig` object | `wlanHEMUConfig` object | `wlanHETBConfig` object | `wlanHERecoveryConfig` object

Physical layer (PHY) format configuration, specified as an object of type `wlanHESUConfig`, `wlanHEMUConfig`, `wlanHETBConfig`, or `wlanHERecoveryConfig`.

When you specify this input as a `wlanHETBConfig` object with the `FeedbackNDP` property set to `1` (`true`), the function interleaves the symbols for active and complementary tone sets for the value of the `RUToneSetIndex` property in accordance with Table 27-32 of [1].

`ruNumber` — Number of RU of interest

positive integer

Number of the RU of interest, specified as a positive integer. The RU number specifies the location of the RU within the channel. For example, consider an 80-MHz transmission with two 242-tone RUs and one 484-tone RU, in order of absolute frequency. For this allocation:

- RU number 1 corresponds to the 242-tone RU in the 20-MHz subchannel at the lowest absolute frequency (size 242, index 1).
- RU number 2 corresponds to the 242-tone RU in the 20-MHz subchannel at the next lowest absolute frequency (size 242, index 2).
- RU number 3 corresponds to the 484-tone RU in the 40-MHz subchannel at the highest absolute frequency (size 484, index 2).

Data Types: `double`

`cbw` — Channel bandwidth

`'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth, specified as one of these values.

- `'CBW20'` – Channel bandwidth of 20 MHz
- `'CBW40'` – Channel bandwidth of 40 MHz

- `'CBW80'` – Channel bandwidth of 80 MHz
- `'CBW160'` – Channel bandwidth of 160 MHz

Data Types: `char` | `string`

`hegi` — Guard interval duration
`0.8` | `1.6` | `3.2`

Guard interval duration, in microseconds, specified as `0.8`, `1.6`, or `3.2`.

Data Types: `double`

`ru` — RU size and index
1-by-2 vector of positive scalars

RU size and index, specified as a 1-by-2 vector of positive scalars. Specify `ru` in the form [*size*,*index*], where *size* must be 26, 52, 106, 242, 484, 996, or 1992 in accordance with the specified channel bandwidth. For example, an 80-MHz transmission has four possible 242-tone RUs (one for each 20-MHz subchannel). RU number 242-1 (*size* = 242 and *index* = 1) is the lowest absolute frequency within the 80-MHz channel. RU number 242-4 is the highest absolute frequency.

Data Types: `double`

`osf` — Oversampling factor
1 (default) | scalar greater than or equal to 1

Oversampling factor, specified as a scalar greater than or equal to 1. The oversampled cyclic prefix length must be an integer number of samples.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`info` — OFDM information
structure

OFDM information, returned as a structure containing these fields.

Name	Values	Description	Data Types
`FFTLength`	Positive integer	Length of the fast Fourier transform (FFT)	`double`
`SampleRate`	Positive scalar	Sample rate of the waveform	`double`
`CPLength`	Positive integer	Cyclic prefix length, in samples	`double`
`NumTones`	Nonnegative integer	Number of active subcarriers	`double`
`NumSubchannels`	Positive integer	Number of 20-MHz subchannels	`double`

Name	Values	Description	Data Types
ActiveFrequencyIndices	Column vector of integers in the interval [-FFTLength/2, (FFTLength/2 – 1)]	Indices of active subcarriers. Each element of this field is the index of an active subcarrier, such that the direct current (DC) or null subcarrier is at the center of the frequency band.	double
ActiveFFTIndices	Column vector of integers in the interval [1, FFTLength]	Indices of active subcarriers within the FFT	double
DataIndices	Column vector of integers in the interval [1, NumTones]	Indices of data within the active subcarriers	double
PilotIndices	Column vector of integers in the interval [1, NumTones]	Indices of pilots within the active subcarriers	double

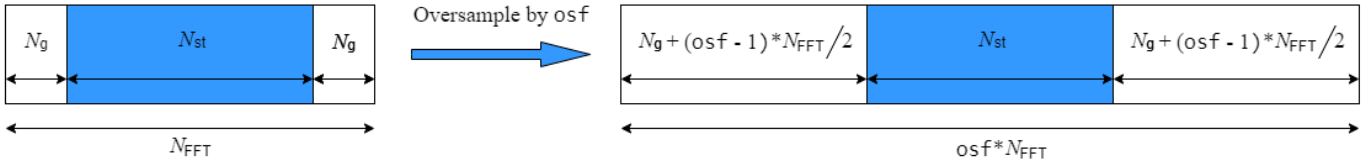
Data Types: struct

Algorithms

FFT-Based Oversampling

An oversampled signal is a signal sampled at a frequency that is higher than the Nyquist rate. WLAN signals maximize occupied bandwidth by using small guardbands, which can pose problems for anti-imaging and anti-aliasing filters. Oversampling increases guardband width relative to the total signal bandwidth, thereby increasing the number of samples in the signal.

This function performs oversampling by using a larger IFFT and zero pad when generating an OFDM waveform. This diagram shows the oversampling process for an OFDM waveform with N_{FFT} subcarriers comprising N_g guardband subcarriers on either side of N_{st} occupied bandwidth subcarriers.



Version History

Introduced in R2019a

References

[1] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements

for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`wlanHEDemodulate`

Objects

`wlanHESUConfig` | `wlanHEMUConfig` | `wlanHETBConfig`

wlanHESIGABitRecover

Recover information bits in HE-SIG-A field

Syntax

```
[bits,failCRC] = wlanHESIGABitRecover(siga,noiseVarEst)
[bits,failCRC] = wlanHESIGABitRecover(siga,noiseVarEst,csi)
```

Description

`[bits,failCRC] = wlanHESIGABitRecover(siga,noiseVarEst)` recovers `bits`, the information bits contained in `siga`, the HE-SIG-A field of an IEEE 802.11 high-efficiency transmission subject to channel noise with estimated variance `noiseVarEst`. The function also returns `failCRC`, the result of the cyclic redundancy check (CRC) on `bits`.

For more information on 802.11ax signal recovery, see “Recovery Procedure for an 802.11ax Packet”.

`[bits,failCRC] = wlanHESIGABitRecover(siga,noiseVarEst,csi)` also enhances the demapping of orthogonal frequency-division multiplexing (OFDM) subcarriers by using channel state information `csi`.

Examples

Recover Information Bits in HE-SIG-A Field

Recover the information bits in the HE-SIG-A field of a WLAN HE single-user (HE-SU) waveform.

Create a WLAN HE-SU-format configuration object with default settings and use it to generate an HE-SU waveform.

```
cfgHE = wlanHESUConfig;
cbw = cfgHE.ChannelBandwidth;
waveform = wlanWaveformGenerator(1,cfgHE);
```

Obtain the WLAN field indices, which contain the HE-SIG-A field.

```
ind = wlanFieldIndices(cfgHE);
rxSIGA = waveform(ind.HESIGA(1):ind.HESIGA(2),:);
```

Perform orthogonal frequency-division multiplexing (OFDM) demodulation to extract the HE-SIG-A field.

```
sigaDemod = wlanHEDemodulate(rxSIGA,'HE-SIG-A',cbw);
```

Return the pre-HE OFDM information and extract the demodulated HE-SIG-A symbols.

```
preHEInfo = wlanHEOFDMInfo('HE-SIG-A',cbw);
siga = sigaDemod(preHEInfo.DataIndices,:);
```

Recover the HE-SIG-A information bits and other information, assuming no channel noise. Display the parity check result.

```
noiseVarEst = 0;
[bits,failCRC] = wlanHESIGABitRecover(siga,noiseVarEst);
disp(failCRC);
```

```
   0
```

Recover HE-SIG-A Information Bits with Channel State Information

Recover the information bits in the HE-SIG-A field of a WLAN HE multiuser (HE-MU) waveform with specified channel state information.

Create a WLAN HE-MU-format configuration object with default settings and use it to generate an HE-MU waveform.

```
cfgHE = wlanHEMUConfig(0);
cbw = cfgHE.ChannelBandwidth;
waveform = wlanWaveformGenerator(1,cfgHE);
```

Obtain the WLAN field indices, which contain the modulated HE-SIG-A symbols.

```
ind = wlanFieldIndices(cfgHE);
rxSIGA = waveform(ind.HESIGA(1):ind.HESIGA(2),:);
```

Perform OFDM demodulation to extract the HE-SIG-A field.

```
sigaDemod = wlanHEDemodulate(rxSIGA,'HE-SIG-A',cbw);
```

Return the pre-HE OFDM information and extract the demodulated HE-SIG-A symbols.

```
preHEInfo = wlanHEOFDMInfo('HE-SIG-A',cbw);
siga = sigaDemod(preHEInfo.DataIndices,:);
```

Specify the channel state information and assume no channel noise.

```
csi = ones(52,1);
noiseVarEst = 0;
```

Recover the HE-SIG-A information bits and other information. Display the CRC result.

```
[bits,failCRC] = wlanHESIGABitRecover(siga,noiseVarEst,csi);
disp(failCRC);
```

```
   0
```

Update HE MU Recovery Configuration Object

Update a WLAN HE recovery configuration object by interpreting recovered HE-SIG-A and HE-SIG-B information bits.

Generate HE MU Waveform

Create a WLAN HE MU configuration object, setting the allocation index to `0`.

```
cfgHEMU = wlanHEMUConfig(0);
```

Generate a WLAN waveform and PPDU field indices for the specified configuration.

```
waveform = wlanWaveformGenerator(1,cfgHEMU);
ind = wlanFieldIndices(cfgHEMU);
```

Recover L-SIG Bits

Create a WLAN recovery configuration object, specifying an HE MU packet format and the length of the L-SIG field.

```
cfg = wlanHERecoveryConfig('PacketFormat','HE-MU','ChannelBandwidth','CBW20');
```

Decode the L-SIG field and obtain the orthogonal frequency-division multiplexing (OFDM) information. The recovery configuration object requires this information to obtain the L-SIG length.

```
lsig = waveform(ind.LSIG(1):ind.LSIG(2));
lsigDemod = wlanHEDemodulate(lsig,'L-SIG',cfg.ChannelBandwidth);
info = wlanHEOFDMInfo('L-SIG',cfg.ChannelBandwidth);
lsigDemod = lsigDemod(info.DataIndices,:);
```

Recover the L-SIG bits and related information, making sure that the bits pass the parity check, and update the recovery configuration object with the L-SIG length. For this example we assume a noiseless channel. For more realistic results you can pass the waveform through an 802.11ax™ channel model by using the `wlanTGaxChannel` System object™ and work with the received waveform.

```
csi = ones(52,1);
[lsigBits,failCheck,lsigInfo] = wlanLSIGBitRecover(lsigDemod,0,csi);
cfg.LSIGLength = lsigInfo.Length;
```

Update Recovery Configuration Object with HE-SIG-A Bits

Decode the HE-SIG-A field and recover the HE-SIG-A bits, ensuring that the bits pass the cyclic redundancy check (CRC).

```
siga = waveform(ind.HESIGA(1):ind.HESIGA(2));
sigaDemod = wlanHEDemodulate(siga,'HE-SIG-A',cfg.ChannelBandwidth);
sigaDemod = sigaDemod(info.DataIndices,:);
[sigaBits,failCRC] = wlanHESIGABitRecover(sigaDemod,0,csi);
disp(failCRC)
```

```
   0
```

Update the recovery configuration object with the recovered HE-SIG-A bits. Display the updated object. A property value of `-1` or `'Unknown'` indicates an unknown or undefined property, which can be updated after decoding the HE-SIG-B common and user fields of the HE MU packet.

```
[cfg,failInterpretation] = interpretHESIGABits(cfg,sigaBits)
```

```
cfg =
  wlanHERecoveryConfig with properties:

                    PacketFormat: 'HE-MU'
                ChannelBandwidth: 'CBW20'
                       LSIGLength: 878
                 SIGBCompression: 0
                         SIGBMCS: 0
                         SIGBDCM: 0
```

```
          NumSIGBSymbolsSignaled: 10
                            STBC: 0
                  LDPCExtraSymbol: 1
               PreFECPaddingFactor: 1
                    PEDisambiguity: 0
                     GuardInterval: 3.2000
                         HELTFType: 4
                    NumHELTFSymbols: 1
                   UplinkIndication: 0
                           BSSColor: 0
                       SpatialReuse: 0
                       TXOPDuration: 127
                        HighDoppler: 0
                   AllocationIndex: -1
        NumUsersPerContentChannel: -1
          RUTotalSpaceTimeStreams: -1
                           RUSize: -1
                          RUIndex: -1
                            STAID: -1
                              MCS: -1
                              DCM: -1
                    ChannelCoding: 'Unknown'
                      Beamforming: -1
              NumSpaceTimeStreams: -1
     SpaceTimeStreamStartingIndex: -1


failInterpretation = logical
   0
```

Update Recovery Configuration Object with HE-SIG-B Common Field Bits

Decode the HE-SIG-B common field, ensuring that all content channels pass the CRC.

```
len = getSIGBLength(cfg);
sigbCommon = waveform(double(ind.HESIGA(2))+(1:len.NumSIGBCommonFieldSamples),:);
sigbCommonDemod = wlanHEDemodulate(sigbCommon,'HE-SIG-B',cfgHEMU.ChannelBandwidth);
sigbCommonDemod = sigbCommonDemod(info.DataIndices);
[sigbCommonBits,status,~] = wlanHESIGBCommonBitRecover(sigbCommonDemod,0,csi,cfg);
disp(status)
```

```
Success
```

Update the recovery configuration object with the recovered HE-SIG-B common field bits and display the updated object. A field returned as `-1` or `'Unknown'` indicates an unknown or undefined property value, which can be updated after decoding the HE-SIG-B user field of the HE MU packet.

```
[cfg,failInterpretation] = interpretHESIGBCommonBits(cfg,sigbCommonBits,status)
```

```
cfg =
  wlanHERecoveryConfig with properties:

                    PacketFormat: 'HE-MU'
                ChannelBandwidth: 'CBW20'
                      LSIGLength: 878
                 SIGBCompression: 0
                         SIGBMCS: 0
                         SIGBDCM: 0
```

```
          NumSIGBSymbolsSignaled: 10
                            STBC: 0
                  LDPCExtraSymbol: 1
               PreFECPaddingFactor: 1
                    PEDisambiguity: 0
                     GuardInterval: 3.2000
                         HELTFType: 4
                   NumHELTFSymbols: 1
                  UplinkIndication: 0
                           BSSColor: 0
                      SpatialReuse: 0
                      TXOPDuration: 127
                       HighDoppler: 0
                  AllocationIndex: 0
        NumUsersPerContentChannel: 9
          RUTotalSpaceTimeStreams: -1
                           RUSize: -1
                          RUIndex: -1
                            STAID: -1
                              MCS: -1
                              DCM: -1
                    ChannelCoding: 'Unknown'
                      Beamforming: -1
              NumSpaceTimeStreams: -1
     SpaceTimeStreamStartingIndex: -1


failInterpretation = logical
   0
```

Update Recovery Configuration Object with HE-SIG-B User Field Bits

Decode the HE-SIG-B user field, ensuring that all users pass the CRC.

```
sigbUser = waveform(ind.HESIGB(1):ind.HESIGB(2));
sigbUserDemod = wlanHEDemodulate(sigbUser,'HE-SIG-B',cfgHEMU.ChannelBandwidth);
sigbUserDemod = sigbUserDemod(info.DataIndices,:);
[sigbUserBits,failCRC,~] = wlanHESIGBUserBitRecover(sigbUserDemod,0,csi,cfg);
disp(failCRC)
```

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

Update the recovery configuration object with the recovered HE-SIG-B user field bits.

```
[user,failInterpretation] = interpretHESIGBUserBits(cfg,sigbUserBits,failCRC);
```

Display the results of interpretation and the third element of the `user` output.

```
disp(failInterpretation)
```

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

```
disp(user{3})
```

```
  wlanHERecoveryConfig with properties:

                  PacketFormat: 'HE-MU'
             ChannelBandwidth: 'CBW20'
```

```
                  LSIGLength: 878
             SIGBCompression: 0
                     SIGBMCS: 0
                     SIGBDCM: 0
       NumSIGBSymbolsSignaled: 10
                        STBC: 0
              LDPCExtraSymbol: 1
           PreFECPaddingFactor: 1
                PEDisambiguity: 0
                 GuardInterval: 3.2000
                     HELTFType: 4
              NumHELTFSymbols: 1
             UplinkIndication: 0
                     BSSColor: 0
                SpatialReuse: 0
                TXOPDuration: 127
                 HighDoppler: 0
              AllocationIndex: 0
    NumUsersPerContentChannel: 9
      RUTotalSpaceTimeStreams: 1
                       RUSize: 26
                      RUIndex: 3
                        STAID: 0
                          MCS: 0
                          DCM: 0
                ChannelCoding: 'LDPC'
                  Beamforming: 0
           NumSpaceTimeStreams: 1
SpaceTimeStreamStartingIndex: 1
```

Input Arguments

`siga` — Demodulated HE-SIG-A symbols

complex-valued matrix

Demodulated HE-SIG-A symbols, specified as a complex-valued matrix. The size of `siga` depends on the packet format.

- For high-efficiency single-user (HE SU) or high-efficiency multiuser (HE MU) packets, specify a 52-by-2 matrix.
- For high-efficiency extended-range single-user (HE ER SU) packets, specify a 52-by-4 matrix.

Data Types: `double`
Complex Number Support: Yes

`noiseVarEst` — Channel noise variance estimate

nonnegative scalar

Channel noise variance estimate, specified as a nonnegative scalar.

Data Types: `double`

`csi` — Channel state information

52-by-1 real-valued vector

Channel state information, specified as a 52-by-1 real-valued vector. To use the channel state information for enhanced demapping of the orthogonal frequency-division multiplexing (OFDM) symbols, specify this argument.

Data Types: `double`

Output Arguments

`bits` — Information bits recovered from HE-SIG-A field

52-by-1 binary column vector

Information bits recovered from HE-SIG-A field, returned as a 52-by-1 binary column vector.

Data Types: `int8`

`failCRC` — CRC result

`1` (`true`) | `0` (`false`)

CRC result, returned as a logical value of `1` (`true`) or `0` (`false`). The function returns this argument as `1` (`true`) if the recovered bits fail the CRC. The function returns this argument as `0` (`false`) if the recovered bits pass the CRC.

Data Types: `logical`

Version History

Introduced in R2019a

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

[2] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`wlanFieldIndices` | `wlanHEDataBitRecover` | `wlanHESIGBCommonBitRecover` | `wlanHESIGBUserBitRecover` | `wlanLSIGBitRecover`

Objects
`wlanHERecoveryConfig`

wlanHESIGBCommonBitRecover

Recover common field bits in HE-SIG-B field

Syntax

```
[bits,status,cfgUpdated] = wlanHESIGBCommonBitRecover(sym,noiseVarEst,cfg)
[bits,status,cfgUpdated] = wlanHESIGBCommonBitRecover(sym,noiseVarEst,csi,
cfg)
```

Description

`[bits,status,cfgUpdated] = wlanHESIGBCommonBitRecover(sym,noiseVarEst,cfg)` recovers `bits`, the HE-SIG-B common field bits in an IEEE 802.11ax high-efficiency multi-user (HE MU) transmission.

The function recovers `bits` from `sym`, the demodulated and equalized HE-SIG-B common field symbols. The `cfg` input parameterizes the transmission, which is subject to channel noise with estimated variance `noiseVarEst`.

The function also returns `status`, the result of content channel decoding, and `cfgUpdated`, the updated transmission parameters recovered from the decoded HE-SIG-B field.

For more information on 802.11ax signal recovery, see the “Recovery Procedure for an 802.11ax Packet” example.

`[bits,status,cfgUpdated] = wlanHESIGBCommonBitRecover(sym,noiseVarEst,csi, cfg)` also enhances the demapping of orthogonal frequency-division multiplexing (OFDM) subcarriers by using channel state information `csi`.

Examples

Recover Information Bits in HE-SIG-B Common Field

Recover the information bits in the HE-SIG-B common field of a WLAN HE MU waveform.

Generate an HE MU waveform for the specified information bits and HE-MU-format configuration object.

```
AllocationIndex = 192;
cfgHE = wlanHEMUConfig(AllocationIndex,'SIGBCompression',false);
bits = [1;0];
waveform = wlanWaveformGenerator(bits,cfgHE);
```

Extract the L-SIG and HE-SIG-A portions from the waveform.

```
ind = wlanFieldIndices(cfgHE);
rxLSIG = waveform(ind.LSIG(1):ind.LSIG(2),:);
rxSIGA = waveform(ind.HESIGA(1):ind.HESIGA(2),:);
```

Create an HE recovery configuration object for an HE-MU-format packet, specifying the channel bandwidth and packet format.

```
cbw = cfgHE.ChannelBandwidth;
cfgRecovery = wlanHERecoveryConfig('ChannelBandwidth',cbw,'PacketFormat','HE-MU');
```

Perform OFDM demodulation to extract the L-SIG and HE-SIG-A fields, assuming no channel noise.

```
noiseVarEst = 0;
info = wlanHEOFDMInfo('L-SIG',cbw);
lsigDemod = wlanHEDemodulate(rxLSIG,'L-SIG',cbw);
sigaDemod = wlanHEDemodulate(rxSIGA,'HE-SIG-A',cbw);
[~,~,lsigInfo] = wlanLSIGBitRecover(lsigDemod(info.DataIndices,:),noiseVarEst);
cfgRecovery.LSIGLength = lsigInfo.Length;
siga = wlanHESIGABitRecover(sigaDemod(info.DataIndices,:),noiseVarEst);
```

Update the HE recovery configuration object with the HE-SIG-A information bits.

```
cfg = interpretHESIGABits(cfgRecovery,siga);
```

Extract the HE-SIG-B field.

```
s = getSIGBLength(cfg);
rxSIGB = waveform(double(ind.HESIGA(2))+(1:s.NumSIGBCommonFieldSamples),:);
```

Demodulate and decode the HE-SIG-B common field, displaying the result.

```
sigbDemod = wlanHEDemodulate(rxSIGB,'HE-SIG-B',cbw);
sigb = sigbDemod(info.DataIndices,:,:);
[bits,status,cfgUpdated] = wlanHESIGBCommonBitRecover(sigb,noiseVarEst,cfg);
disp(bits')
```

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

```
disp(status)
```

```
Success
```

Update HE MU Recovery Configuration Object

Update a WLAN HE recovery configuration object by interpreting recovered HE-SIG-A and HE-SIG-B information bits.

Generate HE MU Waveform

Create a WLAN HE MU configuration object, setting the allocation index to `0`.

```
cfgHEMU = wlanHEMUConfig(0);
```

Generate a WLAN waveform and PPDU field indices for the specified configuration.

```
waveform = wlanWaveformGenerator(1,cfgHEMU);
ind = wlanFieldIndices(cfgHEMU);
```

Recover L-SIG Bits

Create a WLAN recovery configuration object, specifying an HE MU packet format and the length of the L-SIG field.

```
cfg = wlanHERecoveryConfig('PacketFormat','HE-MU','ChannelBandwidth','CBW20');
```

Decode the L-SIG field and obtain the orthogonal frequency-division multiplexing (OFDM) information. The recovery configuration object requires this information to obtain the L-SIG length.

```
lsig = waveform(ind.LSIG(1):ind.LSIG(2));
lsigDemod = wlanHEDemodulate(lsig,'L-SIG',cfg.ChannelBandwidth);
info = wlanHEOFDMInfo('L-SIG',cfg.ChannelBandwidth);
lsigDemod = lsigDemod(info.DataIndices,:);
```

Recover the L-SIG bits and related information, making sure that the bits pass the parity check, and update the recovery configuration object with the L-SIG length. For this example we assume a noiseless channel. For more realistic results you can pass the waveform through an 802.11ax™ channel model by using the `wlanTGaxChannel` System object™ and work with the received waveform.

```
csi = ones(52,1);
[lsigBits,failCheck,lsigInfo] = wlanLSIGBitRecover(lsigDemod,0,csi);
cfg.LSIGLength = lsigInfo.Length;
```

Update Recovery Configuration Object with HE-SIG-A Bits

Decode the HE-SIG-A field and recover the HE-SIG-A bits, ensuring that the bits pass the cyclic redundancy check (CRC).

```
siga = waveform(ind.HESIGA(1):ind.HESIGA(2));
sigaDemod = wlanHEDemodulate(siga,'HE-SIG-A',cfg.ChannelBandwidth);
sigaDemod = sigaDemod(info.DataIndices,:);
[sigaBits,failCRC] = wlanHESIGABitRecover(sigaDemod,0,csi);
disp(failCRC)
```

```
   0
```

Update the recovery configuration object with the recovered HE-SIG-A bits. Display the updated object. A property value of `-1` or `'Unknown'` indicates an unknown or undefined property, which can be updated after decoding the HE-SIG-B common and user fields of the HE MU packet.

```
[cfg,failInterpretation] = interpretHESIGABits(cfg,sigaBits)
```

```
cfg =
  wlanHERecoveryConfig with properties:

                   PacketFormat: 'HE-MU'
              ChannelBandwidth: 'CBW20'
                     LSIGLength: 878
               SIGBCompression: 0
                       SIGBMCS: 0
                       SIGBDCM: 0
        NumSIGBSymbolsSignaled: 10
                          STBC: 0
                 LDPCExtraSymbol: 1
            PreFECPaddingFactor: 1
                 PEDisambiguity: 0
```

```
                  GuardInterval: 3.2000
                      HELTFType: 4
                NumHELTFSymbols: 1
               UplinkIndication: 0
                        BSSColor: 0
                   SpatialReuse: 0
                   TXOPDuration: 127
                    HighDoppler: 0
              AllocationIndex: -1
      NumUsersPerContentChannel: -1
        RUTotalSpaceTimeStreams: -1
                         RUSize: -1
                        RUIndex: -1
                          STAID: -1
                            MCS: -1
                            DCM: -1
                 ChannelCoding: 'Unknown'
                    Beamforming: -1
           NumSpaceTimeStreams: -1
   SpaceTimeStreamStartingIndex: -1


failInterpretation = logical
   0
```

Update Recovery Configuration Object with HE-SIG-B Common Field Bits

Decode the HE-SIG-B common field, ensuring that all content channels pass the CRC.

```
len = getSIGBLength(cfg);
sigbCommon = waveform(double(ind.HESIGA(2))+(1:len.NumSIGBCommonFieldSamples),:);
sigbCommonDemod = wlanHEDemodulate(sigbCommon,'HE-SIG-B',cfgHEMU.ChannelBandwidth);
sigbCommonDemod = sigbCommonDemod(info.DataIndices);
[sigbCommonBits,status,~] = wlanHESIGBCommonBitRecover(sigbCommonDemod,0,csi,cfg);
disp(status)
```

```
Success
```

Update the recovery configuration object with the recovered HE-SIG-B common field bits and display the updated object. A field returned as `-1` or `'Unknown'` indicates an unknown or undefined property value, which can be updated after decoding the HE-SIG-B user field of the HE MU packet.

```
[cfg,failInterpretation] = interpretHESIGBCommonBits(cfg,sigbCommonBits,status)
```

```
cfg =
  wlanHERecoveryConfig with properties:

                   PacketFormat: 'HE-MU'
                ChannelBandwidth: 'CBW20'
                      LSIGLength: 878
                 SIGBCompression: 0
                         SIGBMCS: 0
                         SIGBDCM: 0
           NumSIGBSymbolsSignaled: 10
                            STBC: 0
                  LDPCExtraSymbol: 1
              PreFECPaddingFactor: 1
                   PEDisambiguity: 0
```

```
                  GuardInterval: 3.2000
                      HELTFType: 4
                NumHELTFSymbols: 1
               UplinkIndication: 0
                        BSSColor: 0
                   SpatialReuse: 0
                   TXOPDuration: 127
                    HighDoppler: 0
              AllocationIndex: 0
      NumUsersPerContentChannel: 9
        RUTotalSpaceTimeStreams: -1
                         RUSize: -1
                        RUIndex: -1
                          STAID: -1
                            MCS: -1
                            DCM: -1
                  ChannelCoding: 'Unknown'
                    Beamforming: -1
            NumSpaceTimeStreams: -1
  SpaceTimeStreamStartingIndex: -1


failInterpretation = logical
   0
```

Update Recovery Configuration Object with HE-SIG-B User Field Bits

Decode the HE-SIG-B user field, ensuring that all users pass the CRC.

```
sigbUser = waveform(ind.HESIGB(1):ind.HESIGB(2));
sigbUserDemod = wlanHEDemodulate(sigbUser,'HE-SIG-B',cfgHEMU.ChannelBandwidth);
sigbUserDemod = sigbUserDemod(info.DataIndices,:);
[sigbUserBits,failCRC,~] = wlanHESIGBUserBitRecover(sigbUserDemod,0,csi,cfg);
disp(failCRC)
```

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

Update the recovery configuration object with the recovered HE-SIG-B user field bits.

```
[user,failInterpretation] = interpretHESIGBUserBits(cfg,sigbUserBits,failCRC);
```

Display the results of interpretation and the third element of the `user` output.

```
disp(failInterpretation)
```

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

```
disp(user{3})
```

```
  wlanHERecoveryConfig with properties:

                   PacketFormat: 'HE-MU'
                ChannelBandwidth: 'CBW20'
                      LSIGLength: 878
                SIGBCompression: 0
                        SIGBMCS: 0
                        SIGBDCM: 0
          NumSIGBSymbolsSignaled: 10
```

```
                        STBC: 0
                LDPCExtraSymbol: 1
            PreFECPaddingFactor: 1
                 PEDisambiguity: 0
                  GuardInterval: 3.2000
                      HELTFType: 4
                NumHELTFSymbols: 1
               UplinkIndication: 0
                        BSSColor: 0
                   SpatialReuse: 0
                   TXOPDuration: 127
                    HighDoppler: 0
                AllocationIndex: 0
    NumUsersPerContentChannel: 9
      RUTotalSpaceTimeStreams: 1
                       RUSize: 26
                      RUIndex: 3
                        STAID: 0
                          MCS: 0
                          DCM: 0
                ChannelCoding: 'LDPC'
                  Beamforming: 0
            NumSpaceTimeStreams: 1
SpaceTimeStreamStartingIndex: 1
```

Input Arguments

`sym` — Demodulated and equalized HE-SIG-B symbols

52-by-1 complex-valued vector | 104-by-1 complex-valued vector

Demodulated and equalized HE-SIG-B symbols, specified as a complex-valued column vector. The length of the vector is equal to the number of active subcarriers, which depends on the channel bandwidth of the transmission.

- If the channel bandwidth is 20 MHz, specify this argument as a 52-by-1 vector.
- If the channel bandwidth is 40 MHz, 80 MHz, or 160 MHz, specify this argument as a 104-by-1 vector. This vector must contain the combined 20 MHz subchannel repetitions.

The `ChannelBandwidth` property of the `cfg` input determines the channel bandwidth.

Data Types: `double`
Complex Number Support: Yes

`noiseVarEst` — Noise variance estimate

nonnegative scalar

Noise variance estimate, specified as a nonnegative scalar.

Data Types: `double`

`cfg` — HE MU transmission parameters

`wlanHERecoveryConfig` object

HE MU transmission parameters, specified as a `wlanHERecoveryConfig` object.

`csi` — Channel state information

real-valued column vector

Channel state information, specified as an N_{SD}-by-1 real-valued vector, where N_{SD} is the number of data subcarriers in the HE-SIG-B field.

Data Types: `double`

Output Arguments

`bits` — Recovered HE-SIG-B common field bits

binary column vector | binary matrix

Recovered HE-SIG-B common field bits for each content channel of the HE-SIG-B field, returned as a binary column vector or matrix. The size of this output depends on the channel bandwidth of the transmission according to this table.

Channel Bandwidth/MHz	Size of `bits`
20	18-by-1
40	18-by-2
80	27-by-2
160	43-by-2

Data Types: `int8`

`status` — Result of content channel decoding

character vector

Result of content channel decoding, returned as one of these values.

- `'Success'` – All content channels passed the cyclic redundancy check (CRC).
- `'ContentChannel1Fail'` – Content channel 1 failed the CRC, and the number of HE-SIG-B symbols signaled in the HE-SIG-A field is less than 16.
- `'ContentChannel2Fail'` – Content channel 2 failed the CRC, and the number of HE-SIG-B symbols signaled in the HE-SIG-A field is less than 16.
- `'UnknownNumUsersContentChannel1'` – Content channel 1 failed the CRC, and the number of HE-SIG-B symbols signaled in the HE-SIG-B field is 16.
- `'UnknownNumUsersContentChannel2'` – Content channel 2 failed the CRC, and the number of HE-SIG-B symbols signaled in the HE-SIG-B field is 16.
- `'AllContentChannelCRCFail'` – All content channels failed the CRC.

If the number of HE-SIG-B symbols signaled in the HE-SIG-A field is less than 16 and any content channel fails the CRC, the HE-SIG-A field determines the length of the HE-SIG-B field. If the number of HE-SIG-B symbols signaled in the HE-SIG-A field is 16 and any content channel fails the CRC, the length of the HE-SIG-B field is undetermined.

Data Types: `char`

`cfgUpdated` — Updated HE MU transmission parameters

`wlanHERecoveryConfig` object

Updated HE MU transmission parameters recovered from the decoded HE-SIG-B field, returned as a `wlanHERecoveryConfig` object.

Version History

Introduced in R2019b

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

[2] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`wlanFieldIndices` | `wlanHEDataBitRecover` | `wlanHESIGABitRecover` | `wlanHESIGBUserBitRecover` | `wlanLSIGBitRecover`

Objects

`wlanHERecoveryConfig`

Topics

“HE MU Transmission”

wlanHESIGBUserBitRecover

Recover user field bits in HE-SIG-B field

Syntax

```
[bits,failCRC,cfgUpdated] = wlanHESIGBUserBitRecover(sym,noiseVarEst,cfg)
[bits,failCRC,cfgUpdated] = wlanHESIGBUserBitRecover(sym,noiseVarEst,csi,cfg)
```

Description

`[bits,failCRC,cfgUpdated] = wlanHESIGBUserBitRecover(sym,noiseVarEst,cfg)` recovers `bits`, the HE-SIG-B user field bits contained in an IEEE 802.11ax high-efficiency multi-user (HE MU) transmission.

The function recovers `bits` from `sym`, the demodulated and equalized HE-SIG-B user field symbols. The `cfg` input parameterizes the transmission, which is subject to channel noise with estimated variance `noiseVarEst`.

The function also returns `failCRC`, the result of the cyclic redundancy check (CRC) on `bits`, and `cfgUpdated`, the updated transmission parameters recovered from the decoded HE-SIG-B field.

For more information on 802.11ax signal recovery, see the “Recovery Procedure for an 802.11ax Packet” example.

`[bits,failCRC,cfgUpdated] = wlanHESIGBUserBitRecover(sym,noiseVarEst,csi,cfg)` also enhances the demapping of orthogonal frequency-division multiplexing (OFDM) subcarriers by using channel state information `csi`.

Examples

Recover Information Bits in HE-SIG-B User Field

Recover the information bits in the HE-SIG-B user field of a WLAN HE MU waveform.

Generate an HE MU waveform for the specified information bits and HE-MU-format configuration object.

```
cfgHE = wlanHEMUConfig(192);
bits = [1;0;1;1];
waveform = wlanWaveformGenerator(bits,cfgHE);
```

Extract the L-SIG and HE-SIG-A portions of the waveform.

```
ind = wlanFieldIndices(cfgHE);
rxLSIG = waveform(ind.LSIG(1):ind.LSIG(2),:);
rxSIGA = waveform(ind.HESIGA(1):ind.HESIGA(2),:);
```

Create an HE recovery configuration object for an HE-MU-format packet, specifying the channel bandwidth and packet format.

```
cbw = cfgHE.ChannelBandwidth;
cfgRecovery = wlanHERecoveryConfig('ChannelBandwidth',cbw,'PacketFormat','HE-MU');
```

Perform OFDM demodulation to extract the L-SIG and HE-SIG-A fields, assuming no channel noise.

```
noiseVarEst = 0;
info = wlanHEOFDMInfo('L-SIG',cbw);
lsigDemod = wlanHEDemodulate(rxLSIG,'L-SIG',cbw);
sigaDemod = wlanHEDemodulate(rxSIGA,'HE-SIG-A',cbw);
[~,~,lsigInfo] = wlanLSIGBitRecover(lsigDemod(info.DataIndices,:),noiseVarEst);
cfgRecovery.LSIGLength = lsigInfo.Length;
siga = wlanHESIGABitRecover(sigaDemod(info.DataIndices,:),noiseVarEst);
```

Update the HE recovery configuration object with the HE-SIG-A information bits.

```
cfg = interpretHESIGABits(cfgRecovery,siga);
```

Extract the HE-SIG-B field.

```
rxSIGB = waveform(ind.HESIGB(1):ind.HESIGB(2),:);
```

Demodulate and decode the HE-SIG-B user field, displaying the result.

```
sigbDemod = wlanHEDemodulate(rxSIGB,'HE-SIG-B',cbw);
sigb = sigbDemod(info.DataIndices,:,:);
[bits,failCRC,cfgUpdated] = wlanHESIGBUserBitRecover(sigb,noiseVarEst,cfg);
disp(bits')
```

```
  Columns 1 through 19

   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0

  Columns 20 through 21

   0   1
```

```
disp(failCRC)
```

```
   0
```

Update HE MU Recovery Configuration Object

Update a WLAN HE recovery configuration object by interpreting recovered HE-SIG-A and HE-SIG-B information bits.

Generate HE MU Waveform

Create a WLAN HE MU configuration object, setting the allocation index to `0`.

```
cfgHEMU = wlanHEMUConfig(0);
```

Generate a WLAN waveform and PPDU field indices for the specified configuration.

```
waveform = wlanWaveformGenerator(1,cfgHEMU);
ind = wlanFieldIndices(cfgHEMU);
```

Recover L-SIG Bits

Create a WLAN recovery configuration object, specifying an HE MU packet format and the length of the L-SIG field.

```
cfg = wlanHERecoveryConfig('PacketFormat','HE-MU','ChannelBandwidth','CBW20');
```

Decode the L-SIG field and obtain the orthogonal frequency-division multiplexing (OFDM) information. The recovery configuration object requires this information to obtain the L-SIG length.

```
lsig = waveform(ind.LSIG(1):ind.LSIG(2));
lsigDemod = wlanHEDemodulate(lsig,'L-SIG',cfg.ChannelBandwidth);
info = wlanHEOFDMInfo('L-SIG',cfg.ChannelBandwidth);
lsigDemod = lsigDemod(info.DataIndices,:);
```

Recover the L-SIG bits and related information, making sure that the bits pass the parity check, and update the recovery configuration object with the L-SIG length. For this example we assume a noiseless channel. For more realistic results you can pass the waveform through an 802.11ax™ channel model by using the `wlanTGaxChannel` System object™ and work with the received waveform.

```
csi = ones(52,1);
[lsigBits,failCheck,lsigInfo] = wlanLSIGBitRecover(lsigDemod,0,csi);
cfg.LSIGLength = lsigInfo.Length;
```

Update Recovery Configuration Object with HE-SIG-A Bits

Decode the HE-SIG-A field and recover the HE-SIG-A bits, ensuring that the bits pass the cyclic redundancy check (CRC).

```
siga = waveform(ind.HESIGA(1):ind.HESIGA(2));
sigaDemod = wlanHEDemodulate(siga,'HE-SIG-A',cfg.ChannelBandwidth);
sigaDemod = sigaDemod(info.DataIndices,:);
[sigaBits,failCRC] = wlanHESIGABitRecover(sigaDemod,0,csi);
disp(failCRC)
```

```
   0
```

Update the recovery configuration object with the recovered HE-SIG-A bits. Display the updated object. A property value of `-1` or `'Unknown'` indicates an unknown or undefined property, which can be updated after decoding the HE-SIG-B common and user fields of the HE MU packet.

```
[cfg,failInterpretation] = interpretHESIGABits(cfg,sigaBits)
```

```
cfg =
  wlanHERecoveryConfig with properties:

                   PacketFormat: 'HE-MU'
              ChannelBandwidth: 'CBW20'
                     LSIGLength: 878
               SIGBCompression: 0
                       SIGBMCS: 0
                       SIGBDCM: 0
        NumSIGBSymbolsSignaled: 10
                          STBC: 0
                 LDPCExtraSymbol: 1
            PreFECPaddingFactor: 1
                 PEDisambiguity: 0
```

```
                  GuardInterval: 3.2000
                      HELTFType: 4
                NumHELTFSymbols: 1
               UplinkIndication: 0
                        BSSColor: 0
                   SpatialReuse: 0
                   TXOPDuration: 127
                    HighDoppler: 0
              AllocationIndex: -1
      NumUsersPerContentChannel: -1
        RUTotalSpaceTimeStreams: -1
                         RUSize: -1
                        RUIndex: -1
                          STAID: -1
                            MCS: -1
                            DCM: -1
                 ChannelCoding: 'Unknown'
                    Beamforming: -1
           NumSpaceTimeStreams: -1
  SpaceTimeStreamStartingIndex: -1


failInterpretation = logical
   0
```

Update Recovery Configuration Object with HE-SIG-B Common Field Bits

Decode the HE-SIG-B common field, ensuring that all content channels pass the CRC.

```
len = getSIGBLength(cfg);
sigbCommon = waveform(double(ind.HESIGA(2))+(1:len.NumSIGBCommonFieldSamples),:);
sigbCommonDemod = wlanHEDemodulate(sigbCommon,'HE-SIG-B',cfgHEMU.ChannelBandwidth);
sigbCommonDemod = sigbCommonDemod(info.DataIndices);
[sigbCommonBits,status,~] = wlanHESIGBCommonBitRecover(sigbCommonDemod,0,csi,cfg);
disp(status)
```

```
Success
```

Update the recovery configuration object with the recovered HE-SIG-B common field bits and display the updated object. A field returned as `-1` or `'Unknown'` indicates an unknown or undefined property value, which can be updated after decoding the HE-SIG-B user field of the HE MU packet.

```
[cfg,failInterpretation] = interpretHESIGBCommonBits(cfg,sigbCommonBits,status)
```

```
cfg =
  wlanHERecoveryConfig with properties:

                   PacketFormat: 'HE-MU'
              ChannelBandwidth: 'CBW20'
                    LSIGLength: 878
               SIGBCompression: 0
                       SIGBMCS: 0
                       SIGBDCM: 0
        NumSIGBSymbolsSignaled: 10
                          STBC: 0
                LDPCExtraSymbol: 1
           PreFECPaddingFactor: 1
                PEDisambiguity: 0
```

```
                  GuardInterval: 3.2000
                      HELTFType: 4
                NumHELTFSymbols: 1
               UplinkIndication: 0
                        BSSColor: 0
                   SpatialReuse: 0
                   TXOPDuration: 127
                    HighDoppler: 0
              AllocationIndex: 0
      NumUsersPerContentChannel: 9
        RUTotalSpaceTimeStreams: -1
                         RUSize: -1
                        RUIndex: -1
                          STAID: -1
                            MCS: -1
                            DCM: -1
                  ChannelCoding: 'Unknown'
                    Beamforming: -1
            NumSpaceTimeStreams: -1
  SpaceTimeStreamStartingIndex: -1


failInterpretation = logical
   0
```

Update Recovery Configuration Object with HE-SIG-B User Field Bits

Decode the HE-SIG-B user field, ensuring that all users pass the CRC.

```
sigbUser = waveform(ind.HESIGB(1):ind.HESIGB(2));
sigbUserDemod = wlanHEDemodulate(sigbUser,'HE-SIG-B',cfgHEMU.ChannelBandwidth);
sigbUserDemod = sigbUserDemod(info.DataIndices,:);
[sigbUserBits,failCRC,~] = wlanHESIGBUserBitRecover(sigbUserDemod,0,csi,cfg);
disp(failCRC)
```

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

Update the recovery configuration object with the recovered HE-SIG-B user field bits.

```
[user,failInterpretation] = interpretHESIGBUserBits(cfg,sigbUserBits,failCRC);
```

Display the results of interpretation and the third element of the `user` output.

```
disp(failInterpretation)
```

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

```
disp(user{3})
```

```
  wlanHERecoveryConfig with properties:

                  PacketFormat: 'HE-MU'
               ChannelBandwidth: 'CBW20'
                     LSIGLength: 878
              SIGBCompression: 0
                       SIGBMCS: 0
                       SIGBDCM: 0
         NumSIGBSymbolsSignaled: 10
```

```
                        STBC: 0
                LDPCExtraSymbol: 1
           PreFECPaddingFactor: 1
                PEDisambiguity: 0
                 GuardInterval: 3.2000
                     HELTFType: 4
                NumHELTFSymbols: 1
               UplinkIndication: 0
                        BSSColor: 0
                   SpatialReuse: 0
                   TXOPDuration: 127
                    HighDoppler: 0
              AllocationIndex: 0
    NumUsersPerContentChannel: 9
      RUTotalSpaceTimeStreams: 1
                       RUSize: 26
                      RUIndex: 3
                        STAID: 0
                          MCS: 0
                          DCM: 0
                ChannelCoding: 'LDPC'
                  Beamforming: 0
          NumSpaceTimeStreams: 1
SpaceTimeStreamStartingIndex: 1
```

Input Arguments

`sym` — Demodulated and equalized HE-SIG-B symbols

52-by-1 complex-valued vector | 104-by-1 complex-valued vector

Demodulated and equalized HE-SIG-B symbols, specified as a complex-valued vector. The length of the vector is equal to the number of active subcarriers, which depends on the channel bandwidth of the transmission.

- If the channel bandwidth is 20 MHz, specify this argument as a 52-by-1 vector.
- If the channel bandwidth is 40 MHz, 80 MHz, or 160 MHz, specify this argument as a 104-by-1 vector. This vector must contain the combined 20 MHz subchannel repetitions.

Data Types: `double`
Complex Number Support: Yes

`noiseVarEst` — Channel noise variance estimate

nonnegative scalar

Channel noise variance estimate, specified as a nonnegative scalar.

Data Types: `double`

`cfg` — HE MU transmission parameters

`wlanHERecoveryConfig` object

HE MU transmission parameters, specified as a `wlanHERecoveryConfig` object.

`csi` — Channel state information

real-valued column vector

Channel state information, specified as an N_{SD}-by-1 real-valued vector, where N_{SD} is the number of data subcarriers in the HE-SIG-B field.

Data Types: `double`

Output Arguments

`bits` — Recovered HE-SIG-B user field bits

binary matrix

Recovered HE-SIG-B user field bits, returned as a 21-by-N_{users} binary matrix, where N_{users} is the number of users in the transmission. Each column of this argument contains the recovered user field bits for the corresponding user.

Data Types: `int8`

`failCRC` — CRC result for each user

logical row vector

CRC result for each user, returned as a 1-by-N_{users} logical vector, where N_{users} is the number of users in the transmission. Each element of this argument represents the result of the CRC for the corresponding user. A value of `1` indicates that the user bits failed the CRC. A value of `0` indicates that the user bits passed the CRC.

Data Types: `logical`

`cfgUpdated` — Updated HE MU transmission parameters

cell array of `wlanHERecoveryConfig` objects

Updated HE MU transmission parameters recovered from the decoded HE-SIG-B field, returned as a 1-by-N_{users} cell array of `wlanHERecoveryConfig` objects. N_{users} is the number of users in the transmission.

Note The `wlanHESIGBUserBitRecover` function does not return the updated transmission parameters for users whose `FailCRC` result is `1`.

Data Types: `cell`

Limitations

The `wlanHESIGBUserBitRecover` function returns output arguments `bits`, `failCRC`, and `cfgUpdated` only for users in a valid HE-SIG-B content channel. If the function cannot decode an HE-SIG-B content channel, then it does not return any output arguments for users in that content channel. For more information on allocating users across HE-SIG-B content channels in HE MU transmissions, see “HE MU Transmission”.

Version History

Introduced in R2019b

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

[2] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`wlanFieldIndices` | `wlanHEDataBitRecover` | `wlanHESIGABitRecover` | `wlanHESIGBCommonBitRecover` | `wlanLSIGBitRecover`

Objects

`wlanHERecoveryConfig`

Topics

“HE MU Transmission”

wlanHETBNDPFeedbackStatus

Recover feedback status from HE TB feedback NDP

Syntax

```
status = wlanHETBNDPFeedbackStatus(rxSym,cfg)
```

Description

`status = wlanHETBNDPFeedbackStatus(rxSym,cfg)` recovers the feedback status of an HE trigger-based (HE TB) feedback null data packet (NDP) transmission parameterized by `cfg`. The function recovers the feedback status by using `rxSym`, the demodulated high-efficiency long training field (HE-LTF) of the transmission. The function estimates the feedback status by using the algorithm defined in [1].

The recovered feedback status indicates the value of the bit used for tone modulation in each tone set specified by the `RUToneSetIndex` property of the `cfg` input.

For more information about the HE TB feedback NDP, see section 27.3.18 of [2].

Examples

Recover Feedback Status from HE TB Feedback NDP

Configure an uplink HE TB feedback NDP transmission with four stations (STAs), a channel bandwidth of 20 MHz, and a signal-to-noise ratio (SNR) of 20 dB.

```
numSTA = 4;
cbw = 'CBW20';
snr = 20;
cfgSTA = cell(1,numSTA);
```

Specify the resource unit (RU) tone set index, starting space-time stream, and feedback status for all STAs.

```
ruToneSetIndex = repmat([1 2],1,round(numSTA/2));
startingSTS = repmat([1 2],1,round(numSTA/2));
feedbackStatus = repmat([1 0],1,round(numSTA/2));
```

Create a valid HE TB feedback NDP configuration.

```
cfg = wlanHETBConfig;
cfg = getNDPFeedbackConfiguration(cfg);
```

Configure the channel for transmission, assuming no variation across STAs.

```
tgax = wlanTGaxChannel('ChannelBandwidth',cbw, ...
    'TransmissionDirection','Uplink', ...
    'SampleRate',wlanSampleRate(cfg));
chanInfo = info(tgax);
```

```
awgn = comm.AWGNChannel('NoiseMethod','Signal to noise ratio (SNR)', ...
    'SignalPower',1/tgax.NumReceiveAntennas);
```

Configure STAs and generate an HE TB feedback NDP waveform.

```
rx = 0;
for idx = 1:numSTA

    % Configure STAs

    cfg.RUToneSetIndex = ruToneSetIndex(idx);
    cfg.StartingSpaceTimeStream = startingSTS(idx);
    cfg.FeedbackStatus = feedbackStatus(idx);
    cfgSTA{idx} = cfg;

    % Generate transmit waveform

    waveform = wlanWaveformGenerator([],cfg);

    % Pass waveform through TGax channel

    rx = rx + tgax([waveform; zeros(15,size(waveform,2))]);
end
```

Pass the waveform through the AWGN channel, accounting for the noise energy in nulls to ensure the SNR is defined per active and complementary subcarrier.

```
field = 'HE-LTF';
ofdmInfo = wlanHEOFDMInfo(field,cbw,cfg.GuardInterval);
awgn.SNR = snr - 10*log10(ofdmInfo.FFTLength/12);
rx = awgn(rx);
```

Get the field indices and extract the HE-LTF.

```
ind = wlanFieldIndices(cfgSTA{1});
offset = chanInfo.ChannelFilterDelay;
heltf = rx(offset+(ind.HELTF(1):ind.HELTF(2)),:);
```

Demodulate the HE-LTF.

```
rxSym = wlanHEDemodulate(heltf,field,cbw,cfg.GuardInterval,cfg.HELTFType);
```

Recover the feedback status for the STAs.

```
status = zeros(1,numSTA);
for n = 1:numSTA
    status(n) = wlanHETBNDPFeedbackStatus(rxSym,cfgSTA{n});
end
```

Compare the transmitted and received feedback status for the STAs.

```
disp(isequal(feedbackStatus(1:numSTA),status))
```

```
   1
```

Input Arguments

`rxSym` — Demodulated HE-LTF of received HE TB feedback NDP

complex-valued array

Demodulated HE-LTF of the received HE TB feedback NDP, specified as a complex-valued array of size N_{st}-by-2-by-N_r.

- N_{st} is the number of subcarriers.
  - For transmissions that include active and complementary subcarriers, N_{st} must be 12.
  - For all other transmissions, N_{st} must be 242, 484, 996, or 1992.
- N_r is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

`cfg` — HE TB feedback NDP transmission configuration

`wlanHETBConfig` object

HE TB feedback NDP transmission configuration, specified as a `wlanHETBConfig` object.

Output Arguments

`status` — Feedback status

`1` | `0` | `-1`

Feedback status, returned as one of these values.

- `1` — Transmission detected on the first tone set
- `0` — Transmission detected on the second tone set
- `-1` — Transmission not detected on either tone set

This output indicates the value of the bit used for tone modulation in each tone set specified by the `RUToneSetIndex` property of the `cfg` input. The feedback status and RU tone set index determine the HE-LTF subcarrier mapping in accordance with Table 27-32 of [2].

Data Types: `double`

Version History

Introduced in R2021a

References

[1] Montreuil, Leo *et al. NDP Short Feedback Design*. IEEE 802.11-17/0044r4 (May 2017).

[2] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`wlanHEDemodulate` | `wlanWaveformGenerator`

Objects

`wlanHETBConfig`

Topics

“802.11ax Feedback Status Misdetection Simulation for Uplink Trigger-Based Feedback NDP”

wlanHETrackPilotError

Track errors using pilot subcarriers of HE waveform

Syntax

```
[y,cpe,ae] = wlanHETrackPilotError(x,chanEst,cfg,field)
[y,cpe,ae] = wlanHETrackPilotError(x,chanEst,cfg,field,ruNumber)
[y,cpe,ae] = wlanHETrackPilotError(x,chanEst,cbw,prehefield)
[y,cpe,ae] = wlanHETrackPilotError( ___ ,Name=Value)
```

Description

`[y,cpe,ae] = wlanHETrackPilotError(x,chanEst,cfg,field)` tracks errors using the pilot subcarriers of a high-efficiency (HE) WLAN waveform. The function tracks errors using orthogonal frequency-division multiplexing (OFDM) symbols `x` of the field specified in `field` and channel estimate `chanEst`. The function returns the pilot-error-tracked symbols `y`, common phase error `cpe`, and amplitude error `ae`. To track errors in HE single-user (HE SU) waveforms, use this syntax.

`[y,cpe,ae] = wlanHETrackPilotError(x,chanEst,cfg,field,ruNumber)` also specifies the resource unit (RU) number. To track errors in HE multi-user (HE MU) waveforms, use this syntax.

`[y,cpe,ae] = wlanHETrackPilotError(x,chanEst,cbw,prehefield)` returns HE pilot-error-tracked OFDM symbols for pre-HE fields. The pre-HE field is specified in `prehefield`, and the channel bandwidth is specified in `chanbw`.

`[y,cpe,ae] = wlanHETrackPilotError( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to either input argument combination from the previous syntaxes.

Examples

Track Pilot Error in HE SU Waveform

This example shows how to perform pilot phase and amplitude tracking on the L-SIG field of a distorted HE SU waveform.

Generate HE SU Waveform

Configure an HE SU waveform with default transmission parameters and get the nominal sample rate and field indices.

```
cfgHE = wlanHESUConfig;
cbw = cfgHE.ChannelBandwidth;
fs = wlanSampleRate(cfgHE);
ind = wlanFieldIndices(cfgHE);
```

Generate the waveform, specifying PSDU bits to transmit.

```
bits = [1; 0; 1; 0];
waveform = wlanWaveformGenerator(bits,cfgHE);
```

Impair Waveform

Model a step power amplifier droop impairment by applying a step gain change at the start of the L-SIG field.

Specify the droop gain and length.

```
dbGain = 1; % Droop gain in dB
linearGain = db2mag(dbGain); % Magnitude droop gain
len = double(ind.LSIG(1));
```

Apply the magnitude droop gain to each time-domain sample of the waveform.

```
numSamples = size(waveform,1);
droopPerSample = ones(numSamples,1);
droopPerSample(1:len,:) = linearGain*droopPerSample(1:len,:);
rx = droopPerSample.*waveform;
```

Perform Pilot Amplitude and Phase Tracking

Estimate the channel using the L-LTF.

```
lltf = rx(ind.LLTF(1):ind.LLTF(2),:);
lltfDemod = wlanHEDemodulate(lltf,"L-LTF",cbw);
chanEst = wlanLLTFChannelEstimate(lltfDemod,cbw);
```

Demodulate the L-SIG field.

```
lsig = rx(ind.LSIG(1):ind.LSIG(2),:);
field = "L-SIG";
x = wlanHEDemodulate(lsig,field,cbw);
```

Track pilot errors using the L-SIG field and display the errors.

```
[y,cpe,ae] = wlanHETrackPilotError(x, ...
    chanEst,cfgHE,field,TrackAmplitude=true);
disp([cpe ae])
```

```
   -0.0000   -1.0000
```

Input Arguments

x — Received OFDM symbols

complex-valued 3-D array

Received OFDM symbols, specified as a complex-valued array of size N_{st}-by-N_{sym}-by-N_r.

- N_{st} is the number of active subcarriers, which comprises data and pilot subcarriers.
- N_{sym} is the number of OFDM symbols.
- N_r is the number of receive antennas.

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

chanEst — Channel estimates

complex-valued 3-D array

Channel estimates at the data and pilot subcarriers or channel estimates at the pilot subcarriers only, specified as a complex-valued array of size N_{st}-by-N_{sts}-by-N_r or N_{sp}-by-N_{sts}-by-N_r, respectively.

- N_{st} is the number of occupied subcarriers.
- N_{sts} is the number of space-time streams.
- N_r is the number of receive antennas.
- N_{sp} is the number of pilot subcarriers.

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

`cfg` — HE transmission parameters
`wlanHESUConfig` object | `wlanHEMUConfig` object | `wlanHERecoveryConfig` object

HE transmission parameters, specified as one of these objects.

- `wlanHESUConfig` — Set parameters for an HE SU transmission.
- `wlanHEMUConfig` — Set parameters for an HE MU transmission.
- `wlanHERecoveryConfig` — Set or recover parameters for an HE SU or HE MU transmission.

`field` — Field of received OFDM symbols
`"L-SIG"` | `"RL-SIG"` | `"HE-SIG-A"` | `"HE-SIG-B"` | `"HE-LTF"` | `"HE-Data"`

Field of received OFDM symbols, specified as one of these values.

- `"L-SIG"` — Legacy signal field
- `"RL-SIG"` — Repeated legacy signal field
- `"HE-SIG-A"` — HE SIGNAL A field
- `"HE-SIG-B"` — HE SIGNAL B field
- `"HE-LTF"`— HE long training field
- `"HE-Data"` — HE data field

Data Types: `char` | `string`

`cbw` — Channel bandwidth
`'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth in MHz, specified as `'CBW20'`,`'CBW40'`,`'CBW80'`, or `'CBW160'`.

Data Types: `char` | `string`

`prehefield` — Pre-HE field of received OFDM symbols
`"L-SIG"` | `"RL-SIG"` | `"HE-SIG-A"` | `"HE-SIG-B"`

Pre-HE field of received OFDM symbols, specified as one of these values.

- `"L-SIG"` — Legacy signal field
- `"RL-SIG"` — Repeated legacy signal field
- `"HE-SIG-A"` — HE SIGNAL A field
- `"HE-SIG-B"` — HE SIGNAL B field

Data Types: `char` | `string`

`ruNumber` — RU number in HE MU transmission
`1` (default) | positive integer

RU number in HE MU transmission, specified as a positive integer. The RU number specifies the location of the RU within the channel. For example, consider an 80 MHz transmission with two 242-tone RUs and one 484-tone RU, in order of absolute frequency. For this allocation:

- RU number 1 corresponds to the 242-tone RU in the 20 MHz subchannel at the lowest absolute frequency (size 242, index 1).
- RU number 2 corresponds to the 242-tone RU in the 20 MHz subchannel at the next lowest absolute frequency (size 242, index 2).
- RU number 3 corresponds to the 484-tone RU in the 40 MHz subchannel at the highest absolute frequency (size 484, index 2).

Data Types: `double`

Name-Value Arguments

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

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

Example: `TrackAmplitude=true` enables pilot amplitude tracking.

`TrackPhase` — Pilot phase tracking
`1` or `true` (default) | `0` or `false`

Enable pilot phase tracking, specified as a numeric or logical `1` (`true`) or `0` (`false`). To estimate and correct a common phase offset across all subcarriers and receive antennas, set this property to `1` (`true`). Otherwise, set this property to `0` (`false`).

Data Types: `logical`

`TrackAmplitude` — Pilot amplitude tracking
`0` or `false` (default) | `1` or `true`

Enable pilot amplitude tracking, specified as a numeric or logical `1` (`true`) or `0` (`false`). To estimate and correct an average amplitude error across all subcarriers for each OFDM symbol and receive antenna, set this property to `1` (`true`). Otherwise, set this property to `0` (`false`).

Note Due to the limitations of the algorithm used, disable pilot amplitude tracking when filtering a waveform through a MIMO fading channel.

Data Types: `logical`

Output Arguments

y — Pilot-error-tracked OFDM symbols
complex-valued 3-D array

Pilot-error-tracked OFDM symbols, returned as a complex-valued array of size N_{st}-by-N_{sym}-by-N_r.

- N_{st} is the number of occupied subcarriers.
- N_{sym} is the number of OFDM symbols.
- N_r is the number of receive antennas.

Data Types: `single` | `double`

cpe — Common phase error
real-valued row vector

Common phase error, in radians, averaged over all receive antennas, returned as a real-valued row vector of length equal to the number of OFDM symbols. Each element of this vector contains the common phase error for the corresponding OFDM symbol.

Data Types: `single` | `double`

ae — Average amplitude error
real-valued matrix

Average amplitude error, in dB, returned as a real-valued matrix of size N_{sym}-by-N_r.

- N_{sym} is the number of OFDM symbols.
- N_r is the number of receive antennas.

Each element of this matrix contains the amplitude error for all subcarriers with respect to the estimated received pilots for the corresponding OFDM symbol and receive antenna.

Data Types: `single` | `double`

Version History

Introduced in R2022a

Extended Capabilities

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

See Also

Functions
`wlanFieldIndices` | `wlanHEDemodulate` | `wlanHEDataBitRecover` | `wlanLLTFChannelEstimate`

wlanHTData

Generate HT-Data field waveform

Syntax

```
y = wlanHTData(psdu,cfg)
y = wlanHTData(psdu,cfg,scramInit)
y = wlanHTData( ___ ,OversamplingFactor=osf)
```

Description

`y = wlanHTData(psdu,cfg)` generates an "HT-Data field" on page 3-242[3] time-domain waveform for PLCP service data unit `psdu` and specified transmission parameters `cfg`. See "HT-Data Field Processing" on page 3-242 for waveform generation details.

`y = wlanHTData(psdu,cfg,scramInit)` uses `scramInit` for the scrambler initialization state.

`y = wlanHTData( ___ ,OversamplingFactor=osf)` generates an HT-Data field waveform with an oversampling factor. For more information about oversampling, see "FFT-Based Oversampling" on page 3-244.

Examples

Generate HT-Data Waveform

Generate the waveform signal for a 40 MHz HT-mixed data field with multiple transmit antennas. Create an HT format configuration object. Specify 40 MHz channel bandwidth, two transmit antennas, and two space-time streams.

```
cfgHT = wlanHTConfig('ChannelBandwidth','CBW40','NumTransmitAntennas',2,'NumSpaceTimeStreams', 2
```

```
cfgHT =
  wlanHTConfig with properties:

       ChannelBandwidth: 'CBW40'
    NumTransmitAntennas: 2
    NumSpaceTimeStreams: 2
         SpatialMapping: 'Direct'
                    MCS: 12
          GuardInterval: 'Long'
          ChannelCoding: 'BCC'
             PSDULength: 1024
         AggregatedMPDU: 0
     RecommendSmoothing: 1
```

Assign `PSDULength` bytes of random data to a bit stream and generate the HT data waveform.

3 IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.

```
PSDU =  randi([0 1],cfgHT.PSDULength*8,1);
y = wlanHTData(PSDU,cfgHT);
```

Determine the size of the waveform.

```
size(y)
```

```
ans = 1×2

        2080           2
```

The function returns a complex two-column time-domain waveform. Each column contains 2080 samples, corresponding to the HT-Data field for each transmit antenna.

Input Arguments

psdu — PLCP Service Data Unit

vector

PLCP Service Data Unit (“PSDU” on page 3-242), specified as an N_b-by-1 vector. N_b is the number of bits and equals `PSDULength` × 8.

Data Types: `double`

cfg — Transmission parameters

`wlanHTConfig` object

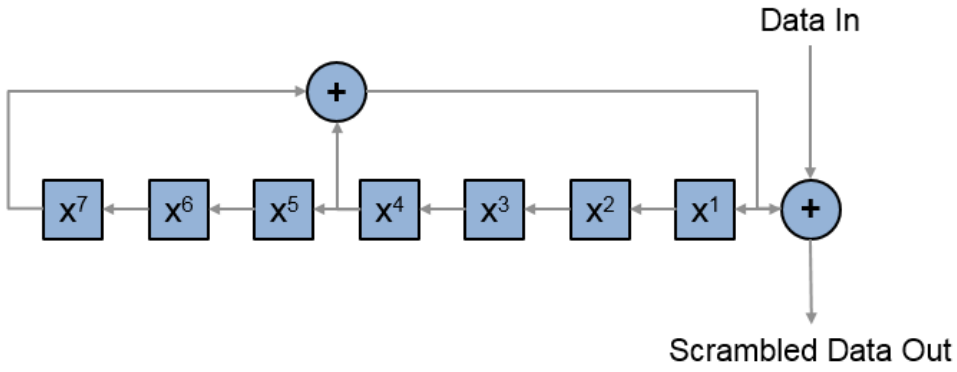
Transmission parameters, specified as a `wlanHTConfig` object.

scramInit — Scrambler initialization state

93 (default) | integer in the interval [1, 127] | binary vector

Scrambler initialization state for each packet generated, specified as an integer in the interval [1, 127] or as the corresponding binary vector of length seven. The default value of 93 is the example state given in IEEE Std 802.11-2012, Section L.1.5.2.

The scrambler initialization used on the transmission data follows the process described in IEEE Std 802.11-2012, Section 18.3.5.5 and IEEE Std 802.11ad-2012, Section 21.3.9. The header and data fields that follow the scrambler initialization field (including data padding bits) are scrambled by XORing each bit with a length-127 periodic sequence generated by the polynomial $S(x) = x^7+x^4+1$. The octets of the PSDU are placed into a bit stream and, within each octet, bit 0 (LSB) is first and bit 7 (MSB) is last. This figure shows the generation of the sequence and the XOR operation.



Conversion from integer to bits uses left-MSB orientation. For example, initializing the scrambler with decimal 1, the bits map to these elements.

Element	X^7	X^6	X^5	X^4	X^3	X^2	X^1
Bit Value	0	0	0	0	0	0	1

To generate the bit stream equivalent to a decimal, use the `int2bit` function. For example, for decimal 1:

```
int2bit(1,7)'
ans =

     0     0     0     0     0     0     1
```

Example: `[1; 0; 1; 1; 1; 0; 1]` conveys the scrambler initialization state of 93 as a binary vector.

Data Types: `double` | `int8`

`osf` — Oversampling factor
1 (default) | scalar greater than or equal to 1

Oversampling factor, specified as a scalar greater than or equal to 1. The oversampled cyclic prefix length must be an integer number of samples.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

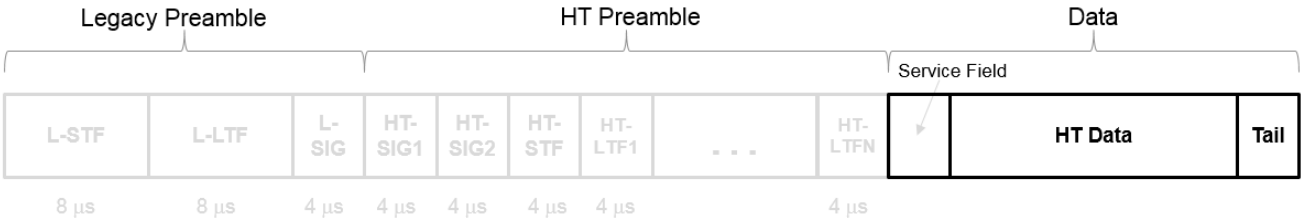
`y` — HT-Data field time-domain waveform
matrix

"HT-Data field" on page 3-242 time-domain waveform for HT-mixed format, returned as an N_S-by-N_T matrix. N_S is the number of time domain samples, and N_T is the number of transmit antennas.

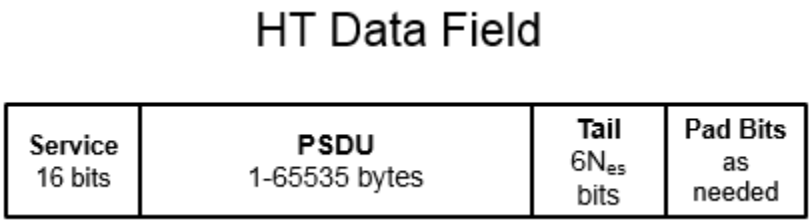
More About

HT-Data field

The HT-Data field follows the last HT-long training field (HT-LTF) of an HT-mixed packet.



The HT-Data field is carries one or more frames from the medium access control (MAC) layer and consists of four subfields.



- **Service** — Contains 16 zeros to initialize the data scrambler
- **PSDU** — Variable-length field containing a PLCP service data unit (PSDU)
- **Tail** — Contains six zeros for each encoding stream, required to terminate a convolutional code
- **Pad Bits** — Variable-length field required to ensure that the HT-Data field consists of an integer number of symbols

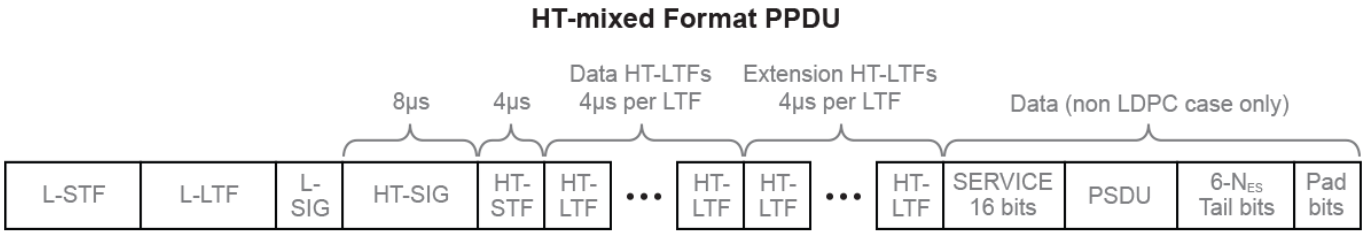
PSDU

Physical layer (PHY) service data unit (PSDU). This field is composed of a variable number of octets. The minimum is 0 (zero) and the maximum is 2500. For more information, see IEEE Std 802.11™-2012, Section 15.3.5.7.

Algorithms

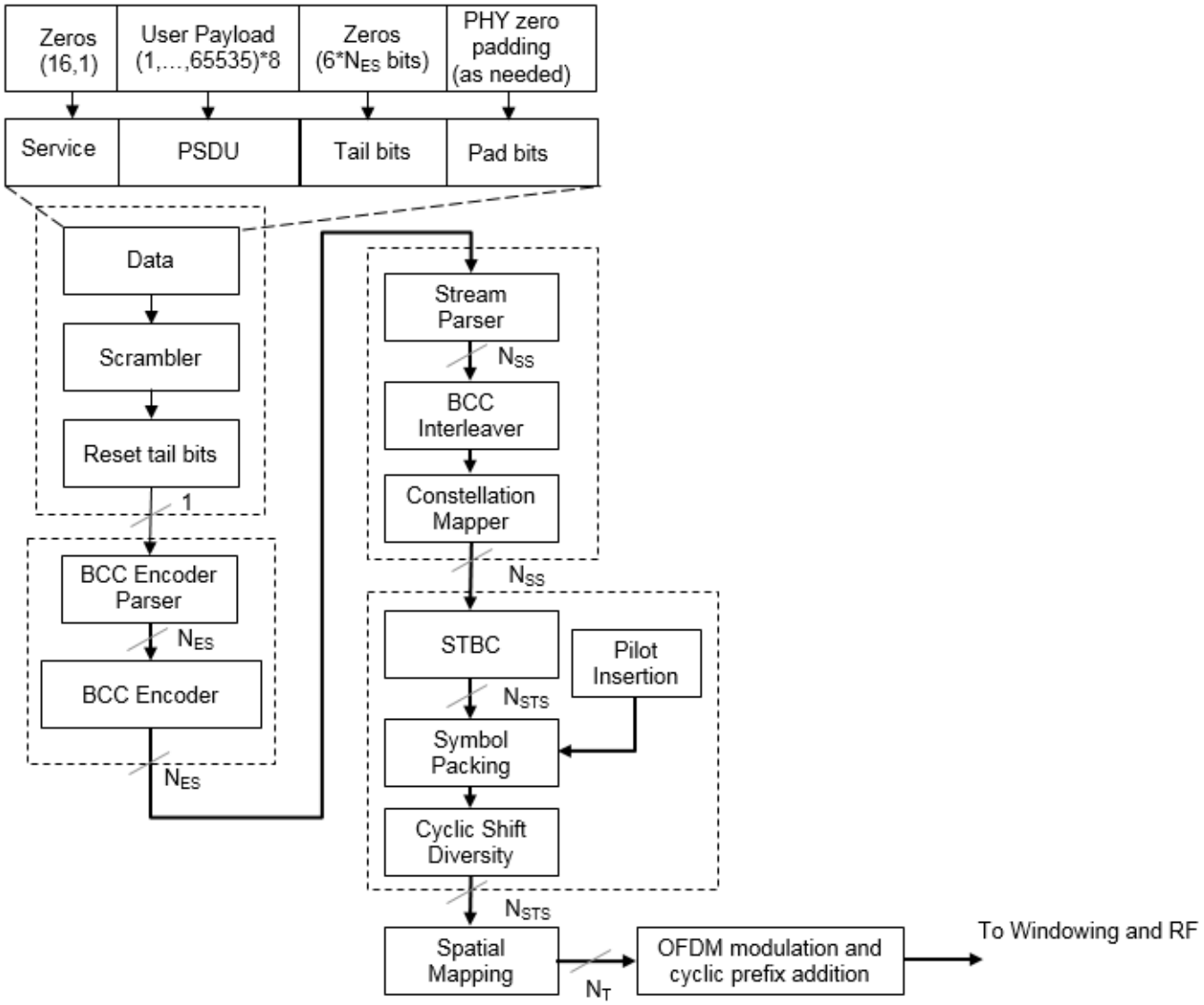
HT-Data Field Processing

The "HT-Data field" on page 3-242 follows the last HT-LTF in the packet structure.



The “HT-Data field” on page 3-242 includes the user payload in the PSDU, plus 16 service bits, $6 \times N_{ES}$ tail bits, and additional padding bits as required to fill out the last OFDM symbol.

For algorithm details, refer to IEEE Std 802.11™-2012 [1], Section 20.3.11. The `wlanHTData` function performs transmitter processing on the “HT-Data field” on page 3-242 and outputs the time-domain waveform for N_T transmit antennas.



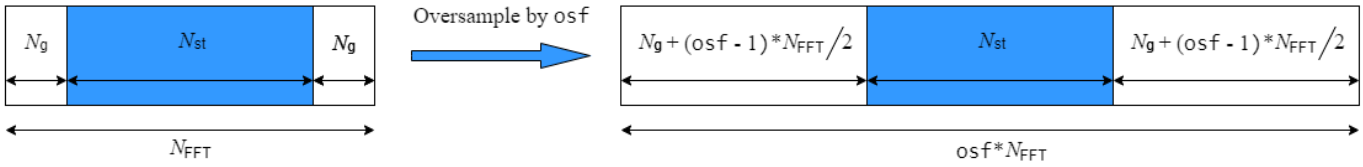
N_{ES} is the number of BCC encoders.
N_{SS} is the number of spatial streams.
N_{STS} is the number of space-time streams.
N_T is the number of transmit antennas.

BCC channel coding is shown. STBC and spatial mapping are optional modes for HT format.

FFT-Based Oversampling

An oversampled signal is a signal sampled at a frequency that is higher than the Nyquist rate. WLAN signals maximize occupied bandwidth by using small guardbands, which can pose problems for anti-imaging and anti-aliasing filters. Oversampling increases guardband width relative to the total signal bandwidth, thereby increasing the number of samples in the signal.

This function performs oversampling by using a larger IFFT and zero pad when generating an OFDM waveform. This diagram shows the oversampling process for an OFDM waveform with N_{FFT} subcarriers comprising N_g guardband subcarriers on either side of N_{st} occupied bandwidth subcarriers.



Version History

Introduced in R2015b

References

[1] IEEE Std 802.11™-2012 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanHTConfig` | `wlanHTDataRecover` | `wlanHTLTF` | `wlanWaveformGenerator`

wlanHTDataRecover

Recover bits from HT-Data field

Syntax

```
dataBits = wlanHTDataRecover(rxDataSig,chEst,noiseVarEst,cfgHT)
dataBits = wlanHTDataRecover(rxDataSig,chEst,noiseVarEst,cfgHT,Name,Value)
[dataBits,eqSym] = wlanHTDataRecover( ___ )
[dataBits,eqSym,cpe] = wlanHTDataRecover( ___ )
[dataBits,eqSym,cpe,ae] = wlanHTDataRecover( ___ )
```

Description

`dataBits = wlanHTDataRecover(rxDataSig,chEst,noiseVarEst,cfgHT)` recovers `dataBits`, a column vector of bits, from `rxDataSig`, the received HT-Data field of a high-throughput-mixed (HT-mixed) transmission. The function recovers `dataBits` by using `chEst`, a channel estimate for the occupied subcarriers, `noiseVarEst`, an estimate of noise variance, and `cfgHT`, a configuration object that contains HT transmission parameters.

For more information about the HT-Data field, see “HT-Data Field” on page 3-252. For more information about the HT-mixed format, see “HT-Mixed Format” on page 3-253.

`dataBits = wlanHTDataRecover(rxDataSig,chEst,noiseVarEst,cfgHT,Name,Value)` specifies algorithm options by using one or more name-value pair arguments. For example, `'LDPCDecodingMethod','layered-bp'` specifies the layered belief propagation low-density parity-check (LDPC) decoding algorithm.

`[dataBits,eqSym] = wlanHTDataRecover( ___ )` returns `eqSym`, the equalized OFDM symbols that comprise the data subcarriers of the HT-Data field, using any input argument combination from the previous syntaxes.

`[dataBits,eqSym,cpe] = wlanHTDataRecover( ___ )` returns `cpe`, the common phase error between the received and expected OFDM symbols.

`[dataBits,eqSym,cpe,ae] = wlanHTDataRecover( ___ )` returns `ae`, the average amplitude error with respect to the estimated received pilots.

Examples

Recover Bits from HT-Data Field

Recover bits from the HT-Data field of an HT-mixed waveform transmitted through an additive white Gaussian noise (AWGN) channel.

Configure an HT-mixed transmission and generate the corresponding HT-Data field.

```
cfgHT = wlanHTConfig('PSDULength',1024);
psduLength = 8*cfgHT.PSDULength;
```

```
bits = randi([0 1],psduLength,1);
txDataSig = wlanHTData(bits,cfgHT);
```

Transmit the signal through an AWGN channel with a signal-to-noise ratio (SNR) of 10 dB.

```
snr = 10;
noiseVarEst = 10^(-snr/10);
rxDataSig = awgn(txDataSig,snr);
```

Specify a channel estimate. Because the signal does not pass through a fading channel, a vector of ones is a perfect estimate. For a channel bandwidth of 20 MHz, the HT-SIG field contains 52 data subcarriers and 4 pilot subcarriers.

```
chEst = ones(56,1);
```

Recover the bits from the received HT-Data field and confirm that the recovered bits match the transmitted bits.

```
dataBits = wlanHTDataRecover(rxDataSig,chEst,noiseVarEst,cfgHT);
isequal(dataBits,bits)
```

```
ans = logical
   1
```

Recover HT-Data Field Using Zero-Forcing Algorithm

Recover the HT-Data field and calculate the common phase error of an HT-mixed signal recovered from an AWGN channel by using zero-forcing equalization at the receiver.

Configure an HT-mixed transmission with a channel bandwidth of 40 MHz and a PSDU length of 1024 bytes, then generate the corresponding HT-Data field.

```
psduLength = 1024;
cfgHT = wlanHTConfig('ChannelBandwidth','CBW40','PSDULength',psduLength);
bits = randi([0 1],8*psduLength,1);
txDataSig = wlanHTData(bits,cfgHT);
```

Pass the signal through an AWGN channel with an SNR of 7 dB.

```
snr = 7;
noiseVarEst = 10^(-snr/10);
rxDataSig = awgn(txDataSig,7);
```

Specify a channel estimate.

```
chEst = ones(114,1);
```

Recover the bits from the received HT-Data field and confirm that the recovered bits match the transmitted bits.

```
[dataBits,eqSym,cpe] = wlanHTDataRecover(rxDataSig,chEst,noiseVarEst,cfgHT,'EqualizationMethod',
isequal(bits,dataBits)
```

```
ans = logical
   1
```

Calculate and display the maximum common phase error.

```
max(abs(cpe))
```

```
ans = 0.4709
```

Recover HT-Data Field and Calculate Amplitude Error

Investigate how applying an amplitude droop affects the amplitude error of the HT-Data field generated from an HT-mixed signal.

Configure an HT-mixed transmission with a channel bandwidth of 40 MHz and a PSDU length of 1024 bytes, then generate the corresponding HT-Data field.

```
psduLength = 1024;
cfgHT = wlanHTConfig('ChannelBandwidth','CBW40','PSDULength',psduLength);
bits = randi([0 1],8*psduLength,1);
tx = wlanHTData(bits,cfgHT);
```

Modify the signal by applying an amplitude droop of 10 dB, starting at the halfway point.

```
signalLength = size(tx,1);
droopGain = 10;
droopGainLinear = 10^(droopGain/20);
txDroop = [ones(signalLength/2,1); droopGainLinear*ones(signalLength/2,1)].*tx;
```

Specify a channel estimate.

```
chEst = ones(114,1);
```

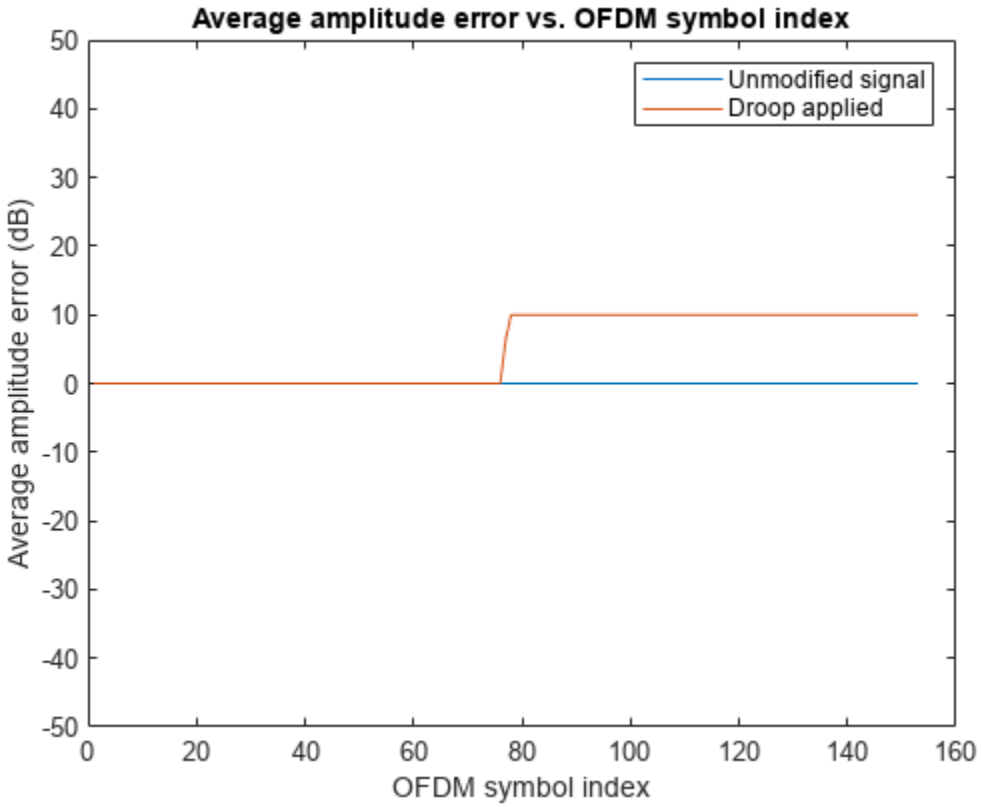
Recover the bits from the ideal and impaired HT-Data fields and confirm that the recovered bits match the transmitted bits.

```
[databits_1, eqSym_1, cpe_1, ae_1] = wlanHTDataRecover(tx, chEst, 0, cfgHT, EqualizationMethod="Z
[databits_2, eqSym_2, cpe_2, ae_2] = wlanHTDataRecover(txDroop, chEst, 0, cfgHT, EqualizationMetl
isequal(databits_1, databits_2, bits)
```

```
ans = logical
   1
```

Plot the absolute value of the measured amplitude errors for the ideal and impaired HT-Data fields.

```
plot(abs(ae_1))
title('Average amplitude error vs. OFDM symbol index')
ylabel('Average amplitude error (dB)')
xlabel('OFDM symbol index')
ylim([-50 50])
hold on
plot(abs(ae_2))
legend('Unmodified signal', 'Droop applied')
```



Input Arguments

`rxDataSig` — Received HT-Data field

complex-valued array

Received HT-Data field, specified as a complex-valued array of size N_S-by-N_R.

- N_S is the number of time-domain samples.
- N_R is the number of receive antennas.

Data Types: `double`

`chEst` — Channel estimate

complex-valued array

Channel estimate, specified as a complex-valued array of size N_{ST}-by-N_{STS}-by-N_R.

- N_{ST} is the number of occupied subcarriers.
- N_{STS} is the number of space-time streams.
- N_R is the number of receive antennas.

Data Types: `double`

`noiseVarEst` — Noise variance estimate

nonnegative scalar

Noise variance estimate, specified as a nonnegative scalar.

Data Types: `double`

`cfgHT` — HT transmission parameters

`wlanHTConfig` object

HT transmission parameters, specified as a `wlanHTConfig` object.

Name-Value Pair Arguments

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

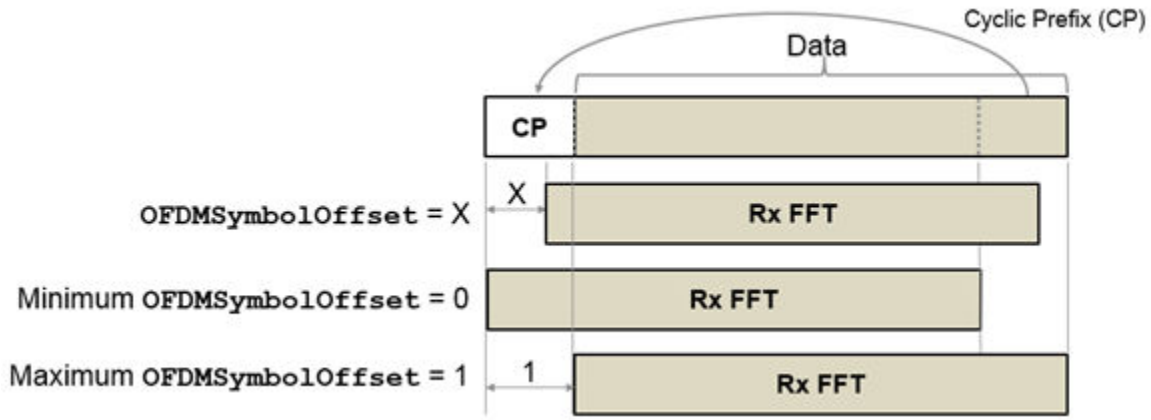
Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'PilotPhaseTracking','None'` disables pilot phase tracking.

`OFDMSymbolOffset` — OFDM symbol sampling offset

`0.75` (default) | scalar in the interval [0, 1]

OFDM symbol sampling offset represented as a fraction of the cyclic prefix (CP) length, specified as the comma-separated pair consisting of `'OFDMSymbolOffset'` and a scalar in the interval [0, 1]. The value you specify indicates the start location for OFDM demodulation relative to the beginning of the CP. The value `0` represents the start of the CP, and the value `1` represents the end of the CP.



Data Types: `double`

`EqualizationMethod` — Equalization method

`'MMSE'` (default) | `'ZF'`

Equalization method, specified as one of these values.

- `'MMSE'` — The receiver uses a minimum mean-square error equalizer.
- `'ZF'` — The receiver uses a zero-forcing equalizer.

When the received signal has multiple receive antennas, the function exploits receiver diversity during equalization. When the number of transmitted space-time streams is one and you specify this argument as `'ZF'`, the function performs maximal-ratio combining.

Data Types: `char` | `string`

`PilotPhaseTracking` — Pilot phase tracking
`'PreEQ'` (default) | `'None'`

Pilot phase tracking, specified as the comma-separated pair consisting of `'PilotPhaseTracking'` and one of these values.

- `'PreEQ'` — Enable pilot phase tracking, which the function performs before any equalization operation.
- `'None'` — Disable pilot phase tracking.

Data Types: `char` | `string`

`PilotAmplitudeTracking` — Pilot amplitude tracking
`'None'` (default) | `'PreEQ'`

Pilot amplitude tracking, specified as the comma-separated pair consisting of `'PilotAmplitudeTracking'` and one of these values.

- `'None'` — Disable pilot amplitude tracking.
- `'PreEQ'` — Enable pilot amplitude tracking, which the function performs before any equalization operation.

Note Due to the limitations of the algorithm used, disable pilot amplitude tracking when filtering a waveform through a MIMO fading channel.

Data Types: `char` | `string`

`LDPCDecodingMethod` — LDPC decoding algorithm
`'bp'` (default) | `'layered-bp'` | `'norm-min-sum'` | `'offset-min-sum'`

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

- `'bp'` — Use the belief propagation (BP) decoding algorithm. For more information, see “Belief Propagation Decoding” on page 3-253.
- `'layered-bp'` — Use the layered BP decoding algorithm, suitable for quasi-cyclic parity check matrices (PCMs). For more information, see “Layered Belief Propagation Decoding” on page 3-254.
- `'norm-min-sum'` — Use the layered BP decoding algorithm with the normalized min-sum approximation. for more information, see “Normalized Min-Sum Decoding” on page 3-255.
- `'offset-min-sum'` — Use the layered BP decoding algorithm with the offset min-sum approximation. For more information, see “Offset Min-Sum Decoding” on page 3-255.

Note When you specify this input as `'norm-min-sum'` or `'offset-min-sum'`, the function sets input log-likelihood ratio (LLR) values that are greater than `1e10` or less than `-1e10` to `1e10` and `-1e10`, respectively. The function then uses these values when executing the LDPC decoding algorithm.

Dependencies

To enable this argument, set the `ChannelCoding` property of the `cfgHT` input to `'LDPC'`.

Data Types: `char` | `string`

`MinSumScalingFactor` — Scaling factor for normalized min-sum LDPC decoding

`0.75` (default) | scalar in the interval (0, 1]

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

Dependencies

To enable this argument, specify the `'LDPCDecodingMethod'` name-value pair argument as `'norm-min-sum'`.

Data Types: `double`

`MinSumOffset` — Offset for offset min-sum LDPC decoding

`0.5` (default) | nonnegative scalar

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

Dependencies

To enable this argument, specify the `'LDPCDecodingMethod'` name-value pair argument as `'offset-min-sum'`.

Data Types: `double`

`MaximumLDPCIterationCount` — Maximum number of LDPC decoding iterations

`12` (default) | positive integer

Maximum number of LDPC decoding iterations, specified as the comma-separated pair consisting of `'MaximumLDPCIterationCount'` and a positive integer.

Dependencies

To enable this argument, set the `ChannelCoding` property of the `cfgHT` input to `'LDPC'`.

Data Types: `double`

`EarlyTermination` — Enable early termination of LDPC decoding

`false` or `0` (default) | `true` or `1`

Enable early termination of LDPC decoding, specified as the comma-separated pair consisting of `'EarlyTermination'` and `1` (`true`) or `0` (`false`).

- When you set this value to `0` (`false`), LDPC decoding completes the number of iterations specified in the `'MaximumLDPCIterationCount'` name-value pair argument regardless of parity check status.
- When you set this value to `1` (`true`), LDPC decoding terminates when all parity checks are satisfied.

Dependencies

To enable this argument, set the `ChannelCoding` property of the `cfgHT` input to `'LDPC'`.

Data Types: `logical`

Output Arguments

`dataBits` — Bits recovered from HT-Data field

binary-valued column vector

Bits recovered from HT-Data field, returned as a binary-valued column vector of length $8 \times L_{\mathrm{PSDU}}$, where L_{PSDU} is the length of the PSDU in bytes.

Data Types: `int8`

`eqSym` — Equalized OFDM symbols

complex-valued array

Equalized OFDM symbols comprising the HT-Data field, returned as a complex-valued array of size N_{SD}-by-N_{Sym}-by-N_{SS}.

- N_{SD} is the number of data subcarriers.
- N_{Sym} is the number of OFDM symbols in the HT-Data field.
- N_{SS} is the number of spatial streams.

Data Types: `double`

`cpe` — Common phase error

real-valued column vector

Common phase error between the received and expected OFDM symbols, in radians, returned as a real-valued column vector. The length of this output is N_{Sym}, the number of OFDM symbols in the HT-Data field. This output is averaged over the receive antennas.

Data Types: `double`

`ae` — Average amplitude error

real-valued array

Average amplitude error, in dB, returned as a real-valued array of size N_{Sym}-by- N_{R}.

- N_{Sym} is the number of OFDM symbols in the HT-Data field.
- N_{R} is the number of receive antennas.

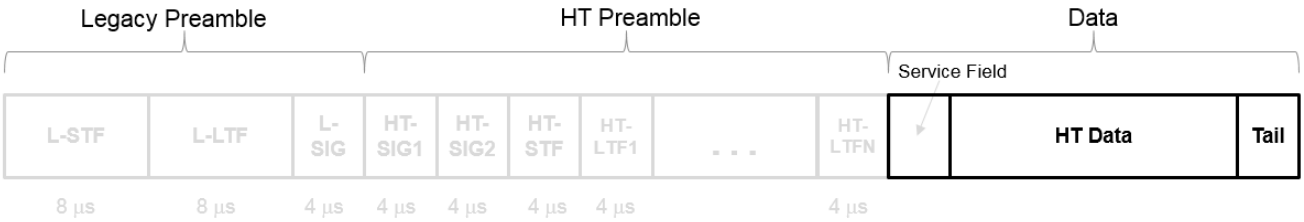
Each element of this matrix contains the amplitude error for all subcarriers with respect to the estimated received pilots for the corresponding OFDM symbol and receive antenna.

Data Types: `double`

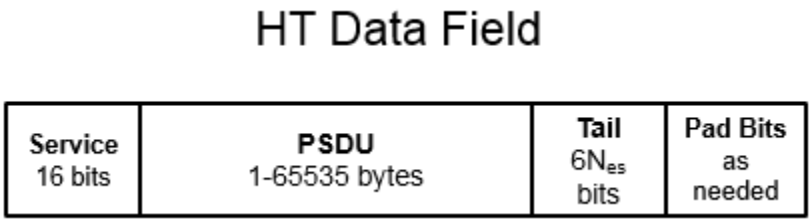
More About

HT-Data Field

The HT-Data field follows the last HT-long training field (HT-LTF) of an HT-mixed packet.



The HT-Data field is carries one or more frames from the medium access control (MAC) layer and consists of four subfields.



- **Service** — Contains 16 zeros to initialize the data scrambler
- **PSDU** — Variable-length field containing a PLCP service data unit (PSDU)
- **Tail** — Contains six zeros for each encoding stream, required to terminate a convolutional code
- **Pad Bits** — Variable-length field required to ensure that the HT-Data field consists of an integer number of symbols

HT-Mixed Format

HT-mixed transmissions contain a PLCP header such that devices operating in HT and non-HT modes can decode them.

Algorithms

This function supports these four LDPC decoding algorithms.

Belief Propagation Decoding

The function implements the BP algorithm based on the decoding algorithm presented in [2]. For transmitted LDPC-encoded codeword $c = (c_0, c_1, \ldots, c_{n-1})$, the input to the LDPC decoder is the LLR given by

$$L(c_i) = \log\left(\frac{\Pr(c_i = 0 \,|\, \text{channel output for } c_i)}{\Pr(c_i = 0 \,|\, \text{channel output for } c_i)}\right).$$

In each iteration, the function updates the key components of the algorithm based on these equations:

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

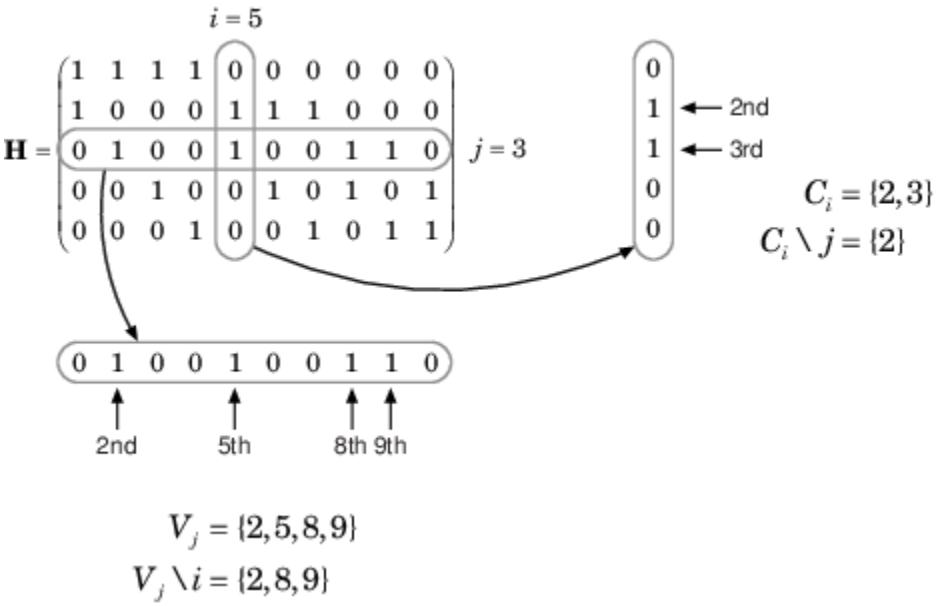
$L(q_{ij}) = L(c_i) + \sum_{j' \in C_i \backslash \{j\}} L(r_{j'i})$, initialized as $L(q_{ij}) = L(c_i)$ before the first iteration, and

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

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

Index sets $C_i \backslash \{j\}$ and $V_j \backslash \{i\}$ are based on the PCM such that the sets C_i and V_j correspond to all nonzero elements in column i and row j of the PCM, respectively.

This figure demonstrates how to compute these index sets for PCM **H** for the case $i = 5$ and $j = 3$.



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

When you specify the `'EarlyTermination'` name-value pair argument as `0` (`false`), the decoding terminates after the number of iterations specified by the `'MaximumLDPCIterationCount'` name-value pair argument. When you specify the `'EarlyTermination'` name-value pair argument as `1` (`true`), the decoding terminates when all parity checks are satisfied ($\mathbf{Hc}^T = 0$) or after the number of iterations specified by the `'MaximumLDPCIterationCount'` name-value pair argument.

Layered Belief Propagation Decoding

The function implements the layered BP algorithm based on the decoding algorithm presented in Section II.A of [3]. The decoding loop iterates over subsets of rows (layers) of the PCM.

For each row, m, in a layer and each bit index, j, the implementation updates the key components of the algorithm based on these equations.

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

(2) $\Psi(x) = \log(|\tanh(x/2)|)$

(3) $A_{mj} = \sum_{n \in N(m)\backslash\{j\}} \Psi(L(q_{mn}))$

(4) $s_{mj} = \prod_{n \in N(m)\backslash\{j\}} \mathrm{sgn}(L(q_{mn}))$

(5) $R_{mj} = -s_{mj}\Psi(A_{mj})$

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

For each layer, the decoding equation (6) works on the combined input obtained from the current LLR inputs, $L(q_{mj})$, and the previous layer updates, R_{mj}.

Because the layered BP algorithm updates only a subset of the nodes in a layer, this algorithm is faster than the BP algorithm. To achieve the same error rate as attained with BP decoding, use half the number of decoding iterations when using the layered BP algorithm.

Normalized Min-Sum Decoding

The function implements the normalized min-sum decoding algorithm by following the layered BP algorithm with equation (3) replaced by

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

where α is the scaling factor specified by the `'MinSumScalingFactor'` name-value pair argument. This equation is an adaptation of equation (4) presented in [4].

Offset Min-Sum Decoding

The function implements the offset min-sum decoding algorithm by following the layered BP algorithm with equation (3) replaced by

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

where β is the offset specified by the `'MinSumOffset'` name-value pair argument. This equation is an adaptation of equation (5) presented in [4].

Version History

Introduced in R2015b

Pilot amplitude tracking

You can perform pilot amplitude tracking by setting the `PilotAmplitudeTracking` input argument to `'PreEQ'`. The average amplitude error is returned in the output argument `ae`.

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

[2] Gallager, Robert G. *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.

[3] Hocevar, D.E. "A Reduced Complexity Decoder Architecture via Layered Decoding of LDPC Codes." In *IEEE Workshop on Signal Processing Systems, 2004. SIPS 2004.*, 107-12. Austin, Texas, USA: IEEE, 2004. https://doi.org/10.1109/SIPS.2004.1363033.

[4] Jinghu Chen, R.M. Tanner, C. Jones, and Yan Li. "Improved Min-Sum Decoding Algorithms for Irregular LDPC Codes." In *Proceedings. International Symposium on Information Theory, 2005. ISIT 2005.*, 449-53, 2005. https://doi.org/10.1109/ISIT.2005.1523374.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanHTConfig` | `wlanHTSIGRecover`

wlanHTLTF

Generate HT-LTF waveform

Syntax

```
y = wlanHTLTF(cfg)
y = wlanHTLTF(cfg,OversamplingFactor=osf)
```

Description

`y = wlanHTLTF(cfg)` generates an “HT-LTF” on page 3-259[4] time-domain waveform for “HT-mixed” on page 3-260 transmissions with parameters specified in `cfg`.

`y = wlanHTLTF(cfg,OversamplingFactor=osf)` generates an oversampled HT-LTF waveform with the specified oversampling factor.

Examples

Generate Single-Stream HT-LTF Waveform

Create a `wlanHTConfig` object having a channel bandwidth of 40 MHz.

```
cfg = wlanHTConfig('ChannelBandwidth','CBW40');
```

Generate the corresponding HT-LTF.

```
hltfOut = wlanHTLTF(cfg);
size(hltfOut)
```

```
ans = 1×2

   160     1
```

The `cfg` parameters result in a 160-sample waveform having only one column corresponding to a single stream transmission.

Generate Oversampled HT-LTF with Four Space-Time Streams

Generate an oversampled HT-LTF waveform with four transmit antennas and four space-time streams.

Create a `wlanHTConfig` configuration with MCS index `31`, four transmit antennas, and four space-time streams.

```
cfg = wlanHTConfig('MCS',31,'NumTransmitAntennas',4,'NumSpaceTimeStreams',4)
```

4 IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.

```
cfg =
  wlanHTConfig with properties:

       ChannelBandwidth: 'CBW20'
    NumTransmitAntennas: 4
    NumSpaceTimeStreams: 4
         SpatialMapping: 'Direct'
                    MCS: 31
          GuardInterval: 'Long'
          ChannelCoding: 'BCC'
             PSDULength: 1024
         AggregatedMPDU: 0
     RecommendSmoothing: 1
```

Specify an oversampling factor and generate the corresponding HT-LTF waveform.

```
osf = 4;
y = wlanHTLTF(cfg,OversamplingFactor=osf);
```

Verify that the waveform output consists of four streams (one for each antenna). Because the channel bandwidth is 20 MHz and the waveform is oversampled and has four space-time streams, the waveform has four HT-LTF and 1280 time-domain samples.

```
size(y)
```

```
ans = 1×2

        1280           4
```

Input Arguments

`cfg` — Transmission parameters

`wlanHTConfig` object

Transmission parameters, specified as a `wlanHTConfig` object.

`osf` — Oversampling factor

1 (default) | scalar greater than or equal to 1

Oversampling factor, specified as a scalar greater than or equal to 1. The oversampled cyclic prefix length must be an integer number of samples.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`y` — HT-LTF waveform

matrix

HT-LTF waveform, returned as an ($N_S \times N_{HTLTF}$)-by-N_T matrix. N_S is the number of time domain samples per N_{HTLTF}, where N_{HTLTF} is the number of OFDM symbols in the “HT-LTF” on page 3-259. N_T is the number of transmit antennas.

N_S is proportional to the channel bandwidth. Each symbol contains 80 time samples per 20 MHz channel.

ChannelBandwidth	N_S
'CBW20'	80
'CBW40'	160

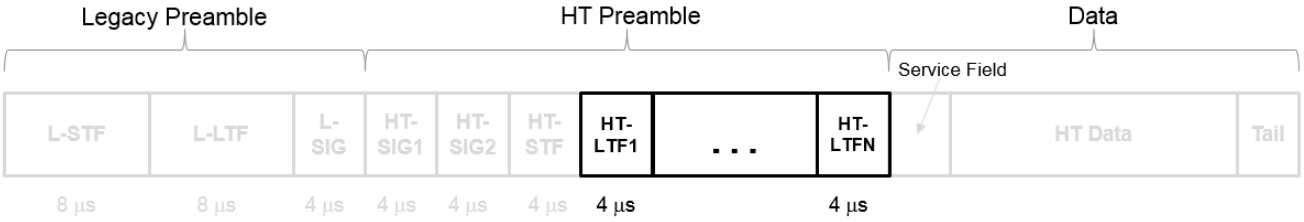
Determination of the number of N_{HTLTF} is described in “HT-LTF” on page 3-259.

Data Types: double

More About

HT-LTF

The high throughput long training field (HT-LTF) is located between the HT-STF and data field of an HT-mixed packet.



As described in Section 19.3.9.4.6 of IEEE Std 802.11-2016, the receiver can use the HT-LTF to estimate the MIMO channel between the set of QAM mapper outputs (or, if STBC is applied, the STBC encoder outputs) and the receive chains. The HT-LTF portion has one or two parts. The first part consists of one, two, or four HT-LTFs that are necessary for demodulation of the HT-Data portion of the PPDU. These HT-LTFs are referred to as HT-DLTFs. The optional second part consists of zero, one, two, or four HT-LTFs that can be used to sound extra spatial dimensions of the MIMO channel not utilized by the HT-Data portion of the PPDU. These HT-LTFs are referred to as HT-ELTFs. Each HT long training symbol is 4 µs. The number of space-time streams and the number of extension streams determines the number of HT-LTF symbols transmitted.

Tables 19-12, 19-13 and 90-14 from IEEE Std 802.11-2012 are reproduced here.

N_{STS} Determination	N_{HTDLTF} Determination	N_{HTELTF} Determination
Table 19-12 defines the number of space-time streams (N_{STS}) based on the number of spatial streams (N_{SS}) from the MCS and the STBC field.	Table 19-13 defines the number of HT-DLTFs required for the N_{STS}.	Table 19-14 defines the number of HT-ELTFs required for the number of extension spatial streams (N_{ESS}). N_{ESS} is defined in HT-SIG$_2$.

N_{STS} Determination		
N_{SS} from MCS	STBC field	N_{STS}
1	0	1
1	1	2
2	0	2
2	1	3
2	2	4
3	0	3
3	1	4
4	0	4

N_{HTDLTF} Determination	
N_{STS}	N_{HTDLTF}
1	1
2	2
3	4
4	4

N_{HTELTF} Determination	
N_{ESS}	N_{HTELTF}
0	0
1	1
2	2
3	4

Additional constraints include:

- $N_{HTLTF} = N_{HTDLTF} + N_{HTELTF} \leq 5$.
- $N_{STS} + N_{ESS} \leq 4$.
  - When $N_{STS} = 3$, N_{ESS} cannot exceed one.
  - If $N_{ESS} = 1$ when $N_{STS} = 3$ then $N_{HTLTF} = 5$.

HT-mixed

As described in IEEE Std 802.11-2012, Section 20.1.4, high throughput mixed (HT-mixed) format packets contain a preamble compatible with IEEE Std 802.11-2012, Section 18 and Section 19 receivers. Non-HT (Section 18 and Section19) STAs can decode the non-HT fields (L-STF, L-LTF, and L-SIG). The remaining preamble fields (HT-SIG, HT-STF, and HT-LTF) are for HT transmission, so the Section 18 and Section 19 STAs cannot decode them. The HT portion of the packet is described in IEEE Std 802.11-2012, Section 20.3.9.4. Support for the HT-mixed format is mandatory.

PPDU

The physical layer (PHY) protocol data unit (PPDU) is the complete physical layer convergence procedure (PLCP) frame, including PLCP headers, MAC headers, the MAC data field, and the MAC and PLCP trailers.

Version History

Introduced in R2015b

References

[1] IEEE Std 802.11™-2012 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanHTConfig` | `wlanLLTF` | `wlanHTData` | `wlanHTLTFDemodulate` | `wlanHTLTFChannelEstimate`

wlanHTLTFChannelEstimate

Channel estimation using HT-LTF

Syntax

```
chEst = wlanHTLTFChannelEstimate(demodSig,cfg)
chEst = wlanHTLTFChannelEstimate(demodSig,cfg,span)
```

Description

`chEst = wlanHTLTFChannelEstimate(demodSig,cfg)` returns the channel estimate using the demodulated "HT-LTF" on page 3-265[5] signal, `demodSig`, given the parameters specified in configuration object `cfg`.

`chEst = wlanHTLTFChannelEstimate(demodSig,cfg,span)` returns the channel estimate and specifies the span of a moving-average filter used to perform frequency smoothing.

Examples

Estimate SISO Channel Using HT-LTF

Estimate and plot the channel coefficients of an HT-mixed format channel by using the high throughput long training field.

Create an HT format configuration object. Generate the corresponding HT-LTF based on the object.

```
cfg = wlanHTConfig;
txSig = wlanHTLTF(cfg);
```

Multiply the transmitted HT-LTF signal by 0.2 + 0.1i and pass it through an AWGN channel. Demodulate the received signal.

```
rxSig = awgn(txSig*(0.2+0.1i),30);
demodSig = wlanHTLTFDemodulate(rxSig,cfg);
```

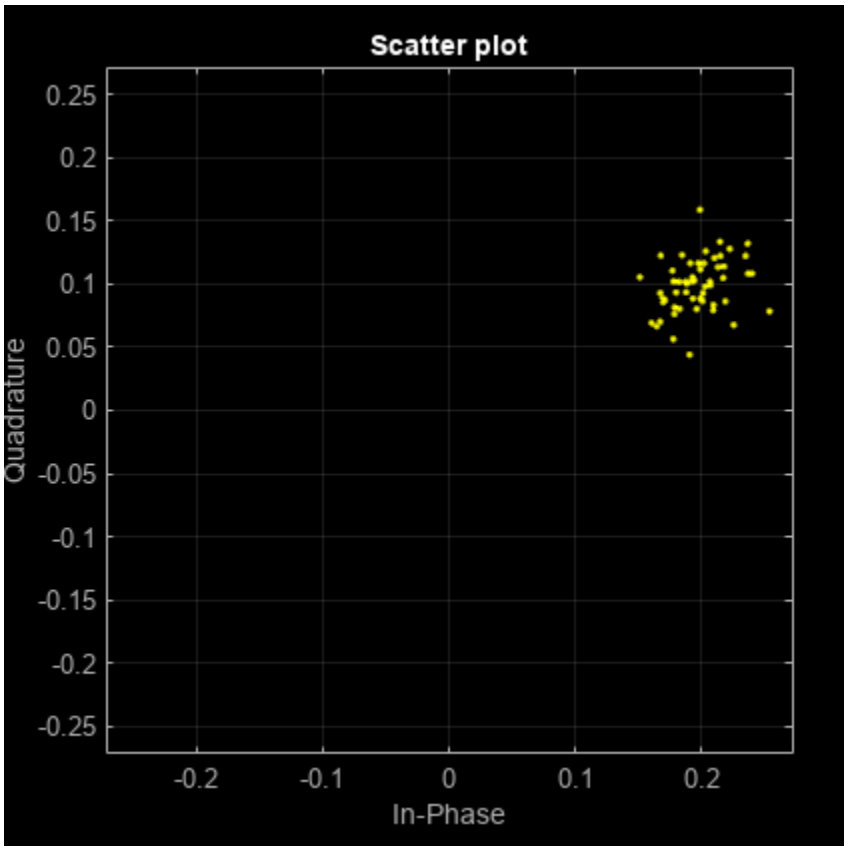
Estimate the channel response using the demodulated HT-LTF.

```
est = wlanHTLTFChannelEstimate(demodSig,cfg);
```

Plot the channel estimate.

```
scatterplot(est)
grid
```

5 IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.



The channel estimate matches the complex channel multiplier.

Estimate MIMO Channel Using HT-LTF

Estimate the channel coefficients of a 2x2 MIMO channel by using the high throughput long training field. Recover the HT-data field and determine the number of bit errors.

Create an HT-mixed format configuration object for a channel having two spatial streams and four transmit antennas. Transmit a complete HT waveform.

```
cfg = wlanHTConfig('NumTransmitAntennas',2, ...
    'NumSpaceTimeStreams',2,'MCS',11);
txPSDU = randi([0 1],8*cfg.PSDULength,1);
txWaveform = wlanWaveformGenerator(txPSDU,cfg);
```

Pass the transmitted waveform through a 2x2 TGn channel.

```
tgnChan = wlanTGnChannel('SampleRate',20e6, ...
    'NumTransmitAntennas',2, ...
    'NumReceiveAntennas',2, ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
rxWaveformNoNoise = tgnChan(txWaveform);
```

Create an AWGN channel with noise power, `nVar`, corresponding to a receiver having a 9 dB noise figure. The noise power is equal to *kTBF*, where *k* is Boltzmann's constant, *T* is the ambient noise temperature (290K), *B* is the bandwidth (20 MHz), and *F* is the noise figure (9 dB).

```
nVar = 10^((-228.6 + 10*log10(290) + 10*log10(20e6) + 9)/10);
awgnChan = comm.AWGNChannel('NoiseMethod','Variance', ...
    'Variance',nVar);
```

Pass the signal through the AWGN channel.

```
rxWaveform = awgnChan(rxWaveformNoNoise);
```

Determine the indices for the HT-LTF. Extract the HT-LTF from the received waveform. Demodulate the HT-LTF.

```
indLTF  = wlanFieldIndices(cfg,'HT-LTF');
rxLTF = rxWaveform(indLTF(1):indLTF(2),:);
ltfDemodSig = wlanHTLTFDemodulate(rxLTF,cfg);
```

Generate the channel estimate by using the demodulated HT-LTF signal. Specify a smoothing filter span of three subcarriers.

```
chEst = wlanHTLTFChannelEstimate(ltfDemodSig,cfg,3);
```

Extract the HT-data field from the received waveform.

```
indData = wlanFieldIndices(cfg,'HT-Data');
rxDataField = rxWaveform(indData(1):indData(2),:);
```

Recover the data and verify that there no bit errors occurred.

```
rxPSDU = wlanHTDataRecover(rxDataField,chEst,nVar,cfg);

numErrs = biterr(txPSDU,rxPSDU)
```

```
numErrs = 0
```

Input Arguments

`demodSig` — Demodulated HT-LTF signal

3-D array

Demodulated HT-LTF signal, specified as an N_{ST}-by-N_{SYM}-by-N_R array. N_{ST} is the number of occupied subcarriers, N_{SYM} is the number of HT-LTF OFDM symbols, and N_R is the number of receive antennas.

Data Types: `double`

`cfg` — Configuration information

`wlanHTConfig` object

Configuration information, specified as a `wlanHTConfig` object.

`span` — Filter span

positive odd integer

Filter span of the frequency smoothing filter, specified as a positive odd integer and expressed as a number of subcarriers. Frequency smoothing is applied only when `span` is specified and greater than one. See “Frequency Smoothing” on page 3-266.

Data Types: `double`

Output Arguments

`chEst` — Channel estimate

3-D array

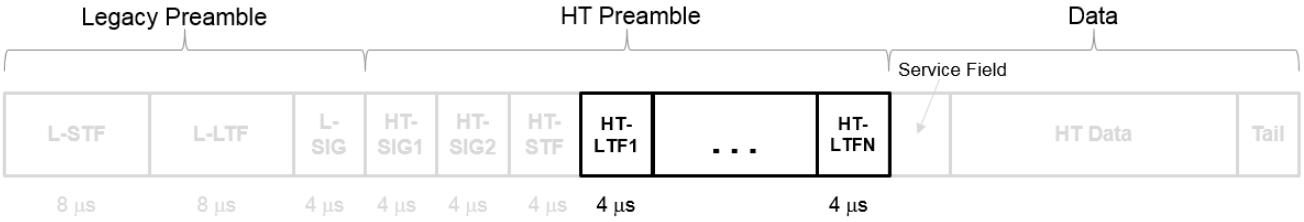
Channel estimate between all combinations of space-time streams and receive antennas, returned as an N_{ST}-by-(N_{STS}+N_{ESS})-by-N_R array. N_{ST} is the number of occupied subcarriers, N_{STS} is the number of space-time streams. N_{ESS} is the number of extension spatial streams. N_R is the number of receive antennas. Data and pilot subcarriers are included in the channel estimate.

Data Types: `double`

More About

HT-LTF

The high throughput long training field (HT-LTF) is located between the HT-STF and data field of an HT-mixed packet.



As described in Section 19.3.9.4.6 of IEEE Std 802.11-2016, the receiver can use the HT-LTF to estimate the MIMO channel between the set of QAM mapper outputs (or, if STBC is applied, the STBC encoder outputs) and the receive chains. The HT-LTF portion has one or two parts. The first part consists of one, two, or four HT-LTFs that are necessary for demodulation of the HT-Data portion of the PPDU. These HT-LTFs are referred to as HT-DLTFs. The optional second part consists of zero, one, two, or four HT-LTFs that can be used to sound extra spatial dimensions of the MIMO channel not utilized by the HT-Data portion of the PPDU. These HT-LTFs are referred to as HT-ELTFs. Each HT long training symbol is 4 µs. The number of space-time streams and the number of extension streams determines the number of HT-LTF symbols transmitted.

Tables 19-12, 19-13 and 90-14 from IEEE Std 802.11-2012 are reproduced here.

N_{STS} Determination

Table 19-12 defines the number of space-time streams (N_{STS}) based on the number of spatial streams (N_{SS}) from the MCS and the STBC field.

N_{SS} from MCS	STBC field	N_{STS}
1	0	1
1	1	2
2	0	2
2	1	3
2	2	4
3	0	3
3	1	4
4	0	4

N_{HTDLTF} Determination

Table 19-13 defines the number of HT-DLTFs required for the N_{STS}.

N_{STS}	N_{HTDLTF}
1	1
2	2
3	4
4	4

N_{HTELTF} Determination

Table 19-14 defines the number of HT-ELTFs required for the number of extension spatial streams (N_{ESS}). N_{ESS} is defined in HT-SIG$_2$.

N_{ESS}	N_{HTELTF}
0	0
1	1
2	2
3	4

Additional constraints include:

- $N_{HTLTF} = N_{HTDLTF} + N_{HTELTF} \leq 5$.
- $N_{STS} + N_{ESS} \leq 4$.
  - When $N_{STS} = 3$, N_{ESS} cannot exceed one.
  - If $N_{ESS} = 1$ when $N_{STS} = 3$ then $N_{HTLTF} = 5$.

Frequency Smoothing

Frequency smoothing can improve channel estimation by averaging out noise.

Frequency smoothing is recommended only for cases in which a single transmit antenna is used. Frequency smoothing consists of applying a moving-average filter that spans multiple adjacent subcarriers. Channel conditions dictate whether frequency smoothing is beneficial.

- If adjacent subcarriers are highly correlated, frequency smoothing results in significant noise reduction.
- In a highly frequency-selective channel, smoothing can degrade the quality of the channel estimate.

Version History

Introduced in R2015b

References

[1] IEEE Std 802.11™-2012 IEEE Standard for Information technology — Telecommunications and information exchange between systems, Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

[2] Perahia, E., and R. Stacey. *Next Generation Wireless LANs: 802.11n and 802.11ac* . 2nd Edition, United Kingdom: Cambridge University Press, 2013.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanHTConfig` | `wlanHTLTFDemodulate` | `wlanHTLTF`

wlanHTLTFDemodulate

Demodulate HT-LTF waveform

Syntax

```
sym = wlanHTLTFDemodulate(rx,cfg)
sym = wlanHTLTFDemodulate(rx,cfg,symOffset)
```

Description

`sym = wlanHTLTFDemodulate(rx,cfg)` returns the demodulated "HT-LTF" on page 3-271[6] by demodulating received time-domain HT-LTF signal `rx`. The input signal is a component of the "HT-mixed" on page 3-272 format "PPDU" on page 3-272. The function demodulates the signal by using the transmission parameters `cfg`.

`sym = wlanHTLTFDemodulate(rx,cfg,symOffset)` specifies the OFDM symbol sampling offset as a fraction of the cyclic prefix length.

Examples

Demodulate HT-LTF in AWGN

Create an HT configuration object.

```
cfg = wlanHTConfig;
```

Generate an HT-LTF signal based on the object.

```
x = wlanHTLTF(cfg);
```

Pass the HT-LTF signal through an AWGN channel.

```
y = awgn(x,20);
```

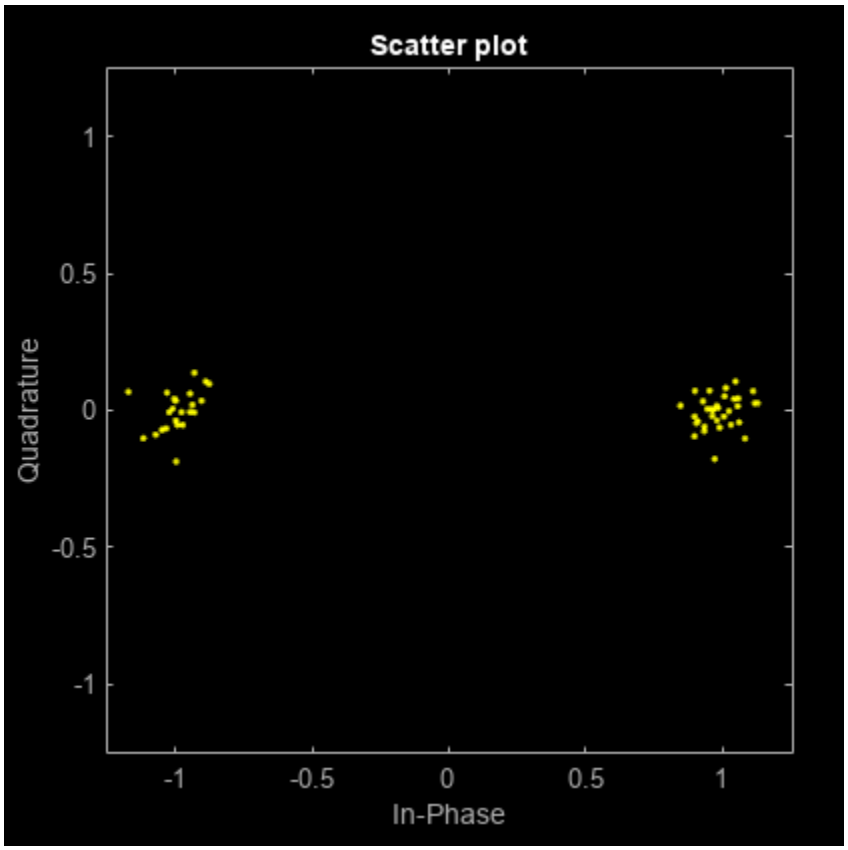
Demodulate the received signal.

```
z = wlanHTLTFDemodulate(y,cfg);
```

Display the scatter plot of the demodulated signal.

```
scatterplot(z)
```

6 IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.



Demodulate 2x2 HT-LTF with OFDM Symbol Offset

Create an HT configuration object having two transmit antennas and two space-time streams.

```
cfg = wlanHTConfig('NumTransmitAntennas',2,'NumSpaceTimeStreams',2, ...
    'MCS',8);
```

Generate the HT-LTF based on the configuration object.

```
x = wlanHTLTF(cfg);
```

Pass the HT-LTF signal through an AWGN channel.

```
y = awgn(x,10);
```

Demodulate the received signal. Set the OFDM symbol offset to `0.5`, which corresponds to 1/2 of the cyclic prefix length.

```
z = wlanHTLTFDemodulate(y,cfg,0.5);
```

Input Arguments

`rx` — Received time-domain signal

complex-valued matrix

Received time-domain signal, specified as a complex-valued matrix of size N_s-by-N_r.

- N_s is the number of time-domain samples. If N_s is not an integer multiple of the OFDM symbol length, L_s, for the specified field,then the function ignores the remaining `mod(`N_s`,`L_s`)` symbols.
- N_r is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

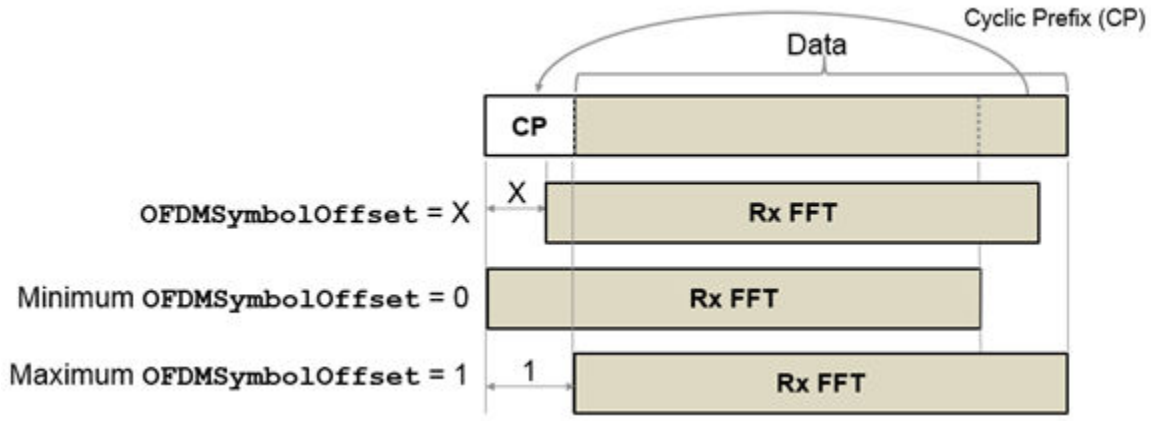
`cfg` — HT format configuration
`wlanHTConfig` object

HT format configuration, specified as a `wlanHTConfig` object.

`symOffset` — OFDM symbol sampling offset
`0.75` (default) | scalar in the interval [0, 1]

OFDM symbol sampling offset, as a fraction of the cyclic prefix length, specified as a scalar in the interval [0, 1].

The value that you specify indicates the start location for OFDM demodulation relative to the beginning of the cyclic prefix.



Example: `0.45`

Data Types: `double`

Output Arguments

`sym` — Demodulated frequency-domain signal
complex-valued array

Demodulated frequency-domain signal, returned as a complex-valued array of size N_{sc}-by-N_{sym}-by-N_r.

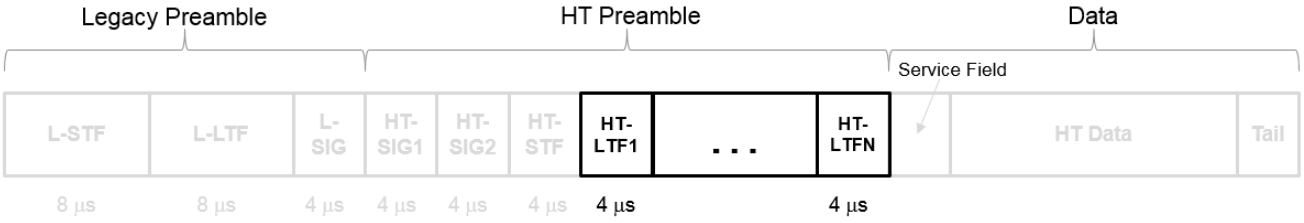
- N_{sc} is the number of active occupied subcarriers in the demodulated field.
- N_{sym} is the number of OFDM symbols.
- N_r is the number of receive antennas.

Data Types: `double`

More About

HT-LTF

The high throughput long training field (HT-LTF) is located between the HT-STF and data field of an HT-mixed packet.



As described in Section 19.3.9.4.6 of IEEE Std 802.11-2016, the receiver can use the HT-LTF to estimate the MIMO channel between the set of QAM mapper outputs (or, if STBC is applied, the STBC encoder outputs) and the receive chains. The HT-LTF portion has one or two parts. The first part consists of one, two, or four HT-LTFs that are necessary for demodulation of the HT-Data portion of the PPDU. These HT-LTFs are referred to as HT-DLTFs. The optional second part consists of zero, one, two, or four HT-LTFs that can be used to sound extra spatial dimensions of the MIMO channel not utilized by the HT-Data portion of the PPDU. These HT-LTFs are referred to as HT-ELTFs. Each HT long training symbol is 4 µs. The number of space-time streams and the number of extension streams determines the number of HT-LTF symbols transmitted.

Tables 19-12, 19-13 and 90-14 from IEEE Std 802.11-2012 are reproduced here.

N_{STS} Determination

Table 19-12 defines the number of space-time streams (N_{STS}) based on the number of spatial streams (N_{SS}) from the MCS and the STBC field.

N_{SS} from MCS	STBC field	N_{STS}
1	0	1
1	1	2
2	0	2
2	1	3
2	2	4
3	0	3
3	1	4
4	0	4

N_{HTDLTF} Determination

Table 19-13 defines the number of HT-DLTFs required for the N_{STS}.

N_{STS}	N_{HTDLTF}
1	1
2	2
3	4
4	4

N_{HTELTF} Determination

Table 19-14 defines the number of HT-ELTFs required for the number of extension spatial streams (N_{ESS}). N_{ESS} is defined in HT-SIG$_2$.

N_{ESS}	N_{HTELTF}
0	0
1	1
2	2
3	4

Additional constraints include:

- $N_{HTLTF} = N_{HTDLTF} + N_{HTELTF} \leq 5$.

- $N_{STS} + N_{ESS} \leq 4$.
  - When $N_{STS} = 3$, N_{ESS} cannot exceed one.
  - If $N_{ESS} = 1$ when $N_{STS} = 3$ then $N_{HTLTF} = 5$.

HT-mixed

High throughput mixed (HT-mixed) format devices support a mixed mode in which the PLCP header is compatible with HT and non-HT modes.

PPDU

The physical layer convergence procedure (PLCP) protocol data unit (PPDU) is the complete PLCP frame, including PLCP headers, MAC headers, the MAC data field, and the MAC and PLCP trailers.

Version History

Introduced in R2015b

References

[1] IEEE Std 802.11™-2012 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanHTLTF` | `wlanHTConfig` | `wlanHTLTFChannelEstimate`

wlanHTOFDMInfo

OFDM information for HT transmission

Syntax

```
info = wlanHTOFDMInfo(field,cfg)
info = wlanHTOFDMInfo(field,cbw,gi)
info = wlanHTOFDMInfo(field,cbw)
info = wlanHTOFDMInfo(___,OversamplingFactor=osf)
```

Description

`info = wlanHTOFDMInfo(field,cfg)` returns `info`, a structure containing orthogonal frequency-division multiplexing (OFDM) information for the input field in a high-throughput (HT) transmission parameterized by `cfg`.

`info = wlanHTOFDMInfo(field,cbw,gi)` returns OFDM information for channel bandwidth `cbw` and guard interval `gi`. To return OFDM information for the HT-Data field when the format configuration is unknown, use this syntax.

`info = wlanHTOFDMInfo(field,cbw)` returns OFDM information for the specified field and channel bandwidth. To return OFDM information for any field other than HT-Data when the format configuration is unknown, use this syntax.

`info = wlanHTOFDMInfo(___,OversamplingFactor=osf)` returns OFDM information for the specified oversampling factor. For more information about oversampling, see “FFT-Based Oversampling” on page 3-276.

Examples

Demodulate HT-LTF and Get OFDM Information

Perform OFDM demodulation on the HT-LTF, then extract the data and pilot subcarriers.

Generate a WLAN waveform for an HT transmission.

```
cfg = wlanHTConfig;
bits = [1; 0; 0; 1];
waveform = wlanWaveformGenerator(bits,cfg);
```

Obtain the field indices and extract the HT-LTF.

```
ind = wlanFieldIndices(cfg);
rx = waveform(ind.HTLTF(1):ind.HTLTF(2),:);
```

Perform OFDM demodulation on the HT-LTF.

```
sym = wlanHTLTFDemodulate(rx,cfg);
```

Get the OFDM information, then extract the data and pilot subcarriers.

```
info = wlanHTOFDMInfo('HT-LTF',cfg);
data = sym(info.DataIndices,:,:);
pilots =  sym(info.PilotIndices,:,:);
```

Get OFDM Information for HT-Data Field

Get OFDM information for the HT-Data field.

Specify the channel bandwidth and guard interval duration.

```
cbw = 'CBW40';
gi = 'Long';
```

Return and display the OFDM information for the HT-Data field.

```
info = wlanHTOFDMInfo('HT-Data',cbw,gi);
disp(info);
```

```
                FFTLength: 128
               SampleRate: 40000000
                 CPLength: 32
           NumSubchannels: 2
                 NumTones: 114
    ActiveFrequencyIndices: [114x1 double]
          ActiveFFTIndices: [114x1 double]
               DataIndices: [108x1 double]
              PilotIndices: [6x1 double]
```

Get OFDM Information for HT L-LTF

Get OFDM information for the L-LTF in a transmission with specified channel bandwidth.

Specify a channel bandwidth of 40 MHz.

```
cbw = 'CBW40';
```

Return and display the OFDM information for the L-LTF.

```
info = wlanHTOFDMInfo('L-LTF',cbw);
disp(info);
```

```
                FFTLength: 128
               SampleRate: 40000000
                 CPLength: [64 0]
           NumSubchannels: 2
                 NumTones: 104
    ActiveFrequencyIndices: [104x1 double]
          ActiveFFTIndices: [104x1 double]
               DataIndices: [96x1 double]
              PilotIndices: [8x1 double]
```

Input Arguments

`field` — Field for which to return OFDM information

`'L-LTF'` | `'L-SIG'` | `'HT-SIG'` | `'HT-LTF'` | `'HT-Data'`

Field for which to return OFDM information, specified as one of these values.

- `'L-LTF'` – Return OFDM information for the legacy long training field (L-LTF).
- `'L-SIG'` – Return OFDM information for the legacy signal (L-SIG) field.
- `'HT-SIG'` – Return OFDM information for the HT signal (HT-SIG) field.
- `'HT-LTF'` – Return OFDM information for the HT long training field (HT-LTF).
- `'HT-Data'` – Return OFDM information for the HT-Data field.

Data Types: `char` | `string`

`cfg` — Transmission parameters

`wlanHTConfig` object

Transmission parameters, specified as a `wlanHTConfig` object.

`cbw` — Channel bandwidth

`'CBW20'` | `'CBW40'`

Channel bandwidth, specified as one of these values.

- `'CBW20'` – Channel bandwidth of 20 MHz
- `'CBW40'` – Channel bandwidth of 40 MHz

Data Types: `char` | `string`

`gi` — Guard interval duration

`'Short'` | `'Long'`

Guard interval duration, specified as `'Short'` or `'Long'`.

Data Types: `char` | `string`

`osf` — Oversampling factor

1 (default) | scalar greater than or equal to 1

Oversampling factor, specified as a scalar greater than or equal to 1. The oversampled cyclic prefix length must be an integer number of samples.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`info` — OFDM information

structure

OFDM information, returned as a structure containing these fields.

Name	Values	Description	Data Types
FFTLength	Positive integer	Length of the fast Fourier transform (FFT)	double
SampleRate	Positive scalar	Sample rate of the waveform	double
CPLength	Positive integer	Cyclic prefix length, in samples	double
NumTones	Nonnegative integer	Number of active subcarriers	double
NumSubchannels	Positive integer	Number of 20-MHz subchannels	double
ActiveFrequencyIndices	Column vector of integers in the interval [-FFTLength/2, (FFTLength/2 – 1)]	Indices of active subcarriers. Each element of this field is the index of an active subcarrier, such that the direct current (DC) or null subcarrier is at the center of the frequency band.	double
ActiveFFTIndices	Column vector of integers in the interval [1, FFTLength]	Indices of active subcarriers within the FFT	double
DataIndices	Column vector of integers in the interval [1, NumTones]	Indices of data within the active subcarriers	double
PilotIndices	Column vector of integers in the interval [1, NumTones]	Indices of pilots within the active subcarriers	double

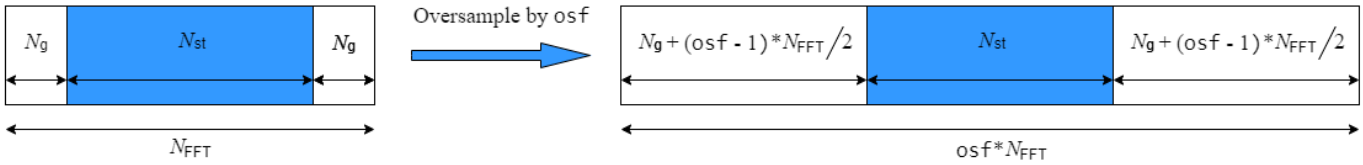
Data Types: struct

Algorithms

FFT-Based Oversampling

An oversampled signal is a signal sampled at a frequency that is higher than the Nyquist rate. WLAN signals maximize occupied bandwidth by using small guardbands, which can pose problems for anti-imaging and anti-aliasing filters. Oversampling increases guardband width relative to the total signal bandwidth, thereby increasing the number of samples in the signal.

This function performs oversampling by using a larger IFFT and zero pad when generating an OFDM waveform. This diagram shows the oversampling process for an OFDM waveform with N_{FFT} subcarriers comprising N_{g} guardband subcarriers on either side of N_{st} occupied bandwidth subcarriers.



Version History

Introduced in R2019a

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`wlanHTLTFDemodulate` | `wlanLLTFDemodulate`

Objects

`wlanHTConfig`

wlanHTSIG

Generate HT-SIG waveform

Syntax

```
y = wlanHTSIG(cfg)
[y,bits] = wlanHTSIG(cfg)
[ ___ ] = wlanHTSIG(cfg,OversamplingFactor=osf)
```

Description

`y = wlanHTSIG(cfg)` generates an "HT-SIG" on page 3-280[7] time-domain waveform for an "HT-mixed" on page 3-281 transmission parameterized by `cfg`.

`[y,bits] = wlanHTSIG(cfg)` also returns the information bits that comprise the HT-SIG field.

`[ ___ ] = wlanHTSIG(cfg,OversamplingFactor=osf)` generates an oversampled HT-SIG waveform with the specified oversampling factor. For more information about oversampling, see "FFT-Based Oversampling" on page 3-281.

Examples

Generate HT-SIG Waveform

Generate an HT-SIG waveform for a single transmit antenna.

Create an HT configuration object. Specify a 40 MHz channel bandwidth.

```
cfg = wlanHTConfig;
cfg.ChannelBandwidth = 'CBW40'
```

```
cfg =
  wlanHTConfig with properties:

       ChannelBandwidth: 'CBW40'
    NumTransmitAntennas: 1
    NumSpaceTimeStreams: 1
         SpatialMapping: 'Direct'
                    MCS: 0
          GuardInterval: 'Long'
          ChannelCoding: 'BCC'
             PSDULength: 1024
         AggregatedMPDU: 0
     RecommendSmoothing: 1
```

Generate the HT-SIG waveform. Determine the size of the waveform.

7 IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.

```
y = wlanHTSIG(cfg);
size(y)
```

```
ans = 1×2

   320     1
```

The function returns a waveform having a complex output of 320 samples corresponding to two 160-sample OFDM symbols.

Display MCS Information from HT-SIG

Generate an HT-SIG waveform and display the MCS information. Change the MCS and display the updated information.

Create a `wlanHTConfig` object having two spatial streams and two transmit antennas. Specify an MCS value of 8, corresponding to BPSK modulation and a coding rate of 1/2.

```
cfg = wlanHTConfig('NumSpaceTimeStreams',2,'NumTransmitAntennas',2,'MCS',8);
```

Generate the information bits from the HT-SIG waveform.

```
[~,sigBits] = wlanHTSIG(cfg);
```

Extract the MCS field from `sigBits` and convert it to its decimal equivalent. The MCS information is contained in bits 1-7.

```
 mcsBits = sigBits(1:7);
 bi2de(mcsBits')
```

```
ans = int8
    8
```

The MCS matches the specified value.

Change the MCS to 13, which corresponds to 64-QAM modulation with a 2/3 coding rate. Generate the HT-SIG waveform.

```
cfg.MCS = 13;
[~,sigBits] = wlanHTSIG(cfg);
```

Verify that the MCS bits are the binary equivalent of 13.

```
mcsBits = sigBits(1:7);
bi2de(mcsBits')
```

```
ans = int8
    13
```

Input Arguments

`cfg` — Transmission parameters

`wlanHTConfig` object

Transmission parameters, specified as a `wlanHTConfig` object.

`osf` — Oversampling factor
1 (default) | scalar greater than or equal to 1

Oversampling factor, specified as a scalar greater than or equal to 1. The oversampled cyclic prefix length must be an integer number of samples.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`y` — HT-SIG waveform
matrix

HT-SIG waveform, returned as an N_S-by-N_T matrix. N_S is the number of time-domain samples, and N_T is the number of transmit antennas.

Data Types: `double`

`bits` — HT-SIG information bits
vector

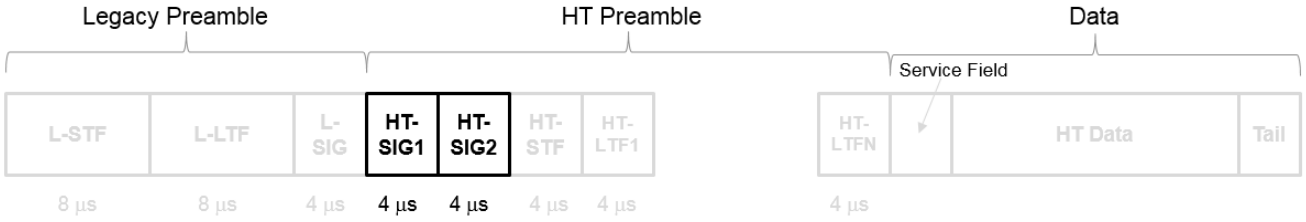
HT-SIG information bits, returned as a 48-by-1 vector.

Data Types: `int8`

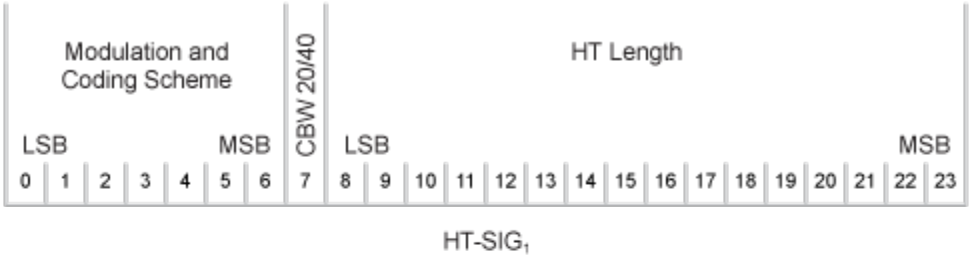
More About

HT-SIG

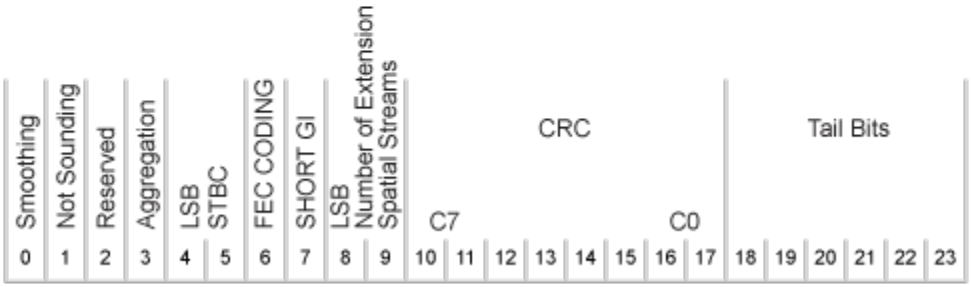
The high throughput signal (HT-SIG) field is located between the L-SIG field and HT-STF and is part of the HT-mixed format preamble. It is composed of two symbols, HT-SIG$_1$ and HT-SIG$_2$.



HT-SIG carries information used to decode the HT packet, including the MCS, packet length, FEC coding type, guard interval, number of extension spatial streams, and whether there is payload aggregation. The HT-SIG symbols are also used for auto-detection between HT-mixed format and legacy OFDM packets.



HT-SIG$_1$



HT-SIG$_2$

For a detailed description of the HT-SIG field, see section 19.3.9.4.3 of IEEE Std 802.11-2020.

HT-mixed

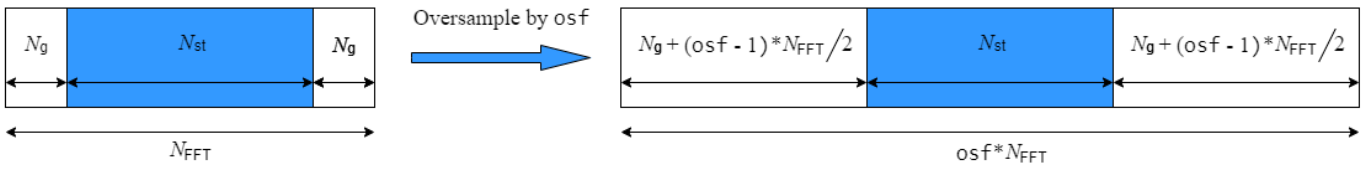
As described in IEEE Std 802.11-2012, Section 20.1.4, high throughput mixed (HT-mixed) format packets contain a preamble compatible with IEEE Std 802.11-2012, Section 18 and Section 19 receivers. Non-HT (Section 18 and Section19) STAs can decode the non-HT fields (L-STF, L-LTF, and L-SIG). The remaining preamble fields (HT-SIG, HT-STF, and HT-LTF) are for HT transmission, so the Section 18 and Section 19 STAs cannot decode them. The HT portion of the packet is described in IEEE Std 802.11-2012, Section 20.3.9.4. Support for the HT-mixed format is mandatory.

Algorithms

FFT-Based Oversampling

An oversampled signal is a signal sampled at a frequency that is higher than the Nyquist rate. WLAN signals maximize occupied bandwidth by using small guardbands, which can pose problems for anti-imaging and anti-aliasing filters. Oversampling increases guardband width relative to the total signal bandwidth, thereby increasing the number of samples in the signal.

This function performs oversampling by using a larger IFFT and zero pad when generating an OFDM waveform. This diagram shows the oversampling process for an OFDM waveform with N_{FFT} subcarriers comprising N_{g} guardband subcarriers on either side of N_{st} occupied bandwidth subcarriers.



Version History

Introduced in R2015b

References

[1] IEEE Std 802.11™-2012 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanHTConfig` | `wlanLSIG` | `wlanHTSTF` | `wlanHTSIGRecover`

wlanHTSIGRecover

Recover HT-SIG information bits

Syntax

```
recBits = wlanHTSIGRecover(rxSig,chEst,noiseVarEst,cbw)
recBits = wlanHTSIGRecover(rxSig,chEst,noiseVarEst,cbw,Name,Value)
[recBits,failCRC] = wlanHTSIGRecover( ___ )
[recBits,failCRC,eqSym] = wlanHTSIGRecover( ___ )
[recBits,failCRC,eqSym,cpe] = wlanHTSIGRecover( ___ )
```

Description

`recBits = wlanHTSIGRecover(rxSig,chEst,noiseVarEst,cbw)` returns the recovered information bits from the "HT-SIG" on page 3-288[8] field and performs a CRC check. Inputs include the channel estimate data `chEst`, noise variance estimate `noiseVarEst`, and channel bandwidth `cbw`.

`recBits = wlanHTSIGRecover(rxSig,chEst,noiseVarEst,cbw,Name,Value)` specifies algorithm parameters by using one or more name-value pair arguments.

`[recBits,failCRC] = wlanHTSIGRecover( ___ )` returns the result of the CRC check, `failCRC`, using any of the arguments from the previous syntaxes.

`[recBits,failCRC,eqSym] = wlanHTSIGRecover( ___ )` returns the equalized symbols, `eqSym`.

`[recBits,failCRC,eqSym,cpe] = wlanHTSIGRecover( ___ )` returns the common phase error, `cpe`.

Examples

Recover HT-SIG Information Bits in Perfect Channel

Create a `wlanHTConfig` object having a channel bandwidth of 40 MHz. Use the object to create an HT-SIG field.

```
cfg = wlanHTConfig('ChannelBandwidth','CBW40');
[txSig,txBits] = wlanHTSIG(cfg);
```

Because a perfect channel is assumed, specify the channel estimate as a column vector of ones and the noise variance estimate as zero.

```
chEst = ones(104,1);
noiseVarEst = 0;
```

Recover the HT-SIG information bits. Verify that the received information bits are identical to the transmitted bits.

8 IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.

```
rxBits = wlanHTSIGRecover(txSig,chEst,noiseVarEst,'CBW40');
numerr = biterr(txBits,rxBits)
```

```
numerr = 0
```

Recover HT-SIG Bits Using Zero-Forcing Equalizer

Configure an HT transmission with a channel bandwidth of 40 MHz by creating a `wlanHTConfig` object. Generate the corresponding HT-SIG field.

```
cfg = wlanHTConfig('ChannelBandwidth','CBW40');
[txSig,txBits] = wlanHTSIG(cfg);
```

Pass the transmitted HT-SIG waveform through an additive white Gaussian noise (AWGN) channel.

```
awgnChan = comm.AWGNChannel('NoiseMethod','Variance','Variance',0.1);
rxSig = awgnChan(txSig);
```

Recover the HT-SIG field assuming a perfect channel and a noise variance estimate of `0.1`, specifying zero-force equalization. Verify that the received information has no bit errors.

```
recBits = wlanHTSIGRecover(rxSig,ones(104,1),0.1,'CBW40','EqualizationMethod','ZF');
biterr(txBits,recBits)
```

```
ans = 0
```

Recover HT-SIG in 2x2 MIMO Channel

Recover HT-SIG in a 2x2 MIMO channel with AWGN. Confirm that the `CRC` check passes.

Configure a 2x2 MIMO TGn channel.

```
chanBW = 'CBW20';
cfg = wlanHTConfig( ...
    'ChannelBandwidth',chanBW, ...
    'NumTransmitAntennas',2, ...
    'NumSpaceTimeStreams',2);
```

Generate L-LTF and HT-SIG waveforms.

```
txLLTF  = wlanLLTF(cfg);
txHTSIG = wlanHTSIG(cfg);
```

Set the sample rate to correspond to the channel bandwidth. Create a TGn 2x2 MIMO channel without large scale fading effects.

```
fsamp = 20e6;
tgnChan = wlanTGnChannel('SampleRate',fsamp, ...
    'LargeScaleFadingEffect','None', ...
    'NumTransmitAntennas',2, ...
    'NumReceiveAntennas',2);
```

Pass the L-LTF and HT-SIG waveforms through a TGn channel with white noise.

```
rxLLTF = awgn(tgnChan(txLLTF),20);
rxHTSIG = awgn(tgnChan(txHTSIG),20);
```

Demodulate the L-LTF signal. Generate a channel estimate by using the demodulated L-LTF.

```
demodLLTF = wlanLLTFDemodulate(rxLLTF,chanBW,1);
chanEst = wlanLLTFChannelEstimate(demodLLTF,chanBW);
```

Recover the information bits, the CRC failure status, and the equalized symbols from the received HT-SIG field.

```
[recHTSIGBits,failCRC,eqSym] = wlanHTSIGRecover(rxHTSIG, ...
    chanEst,0.01,chanBW);
```

Verify that HT-SIG passed a CRC check by examining the status of `failCRC`.

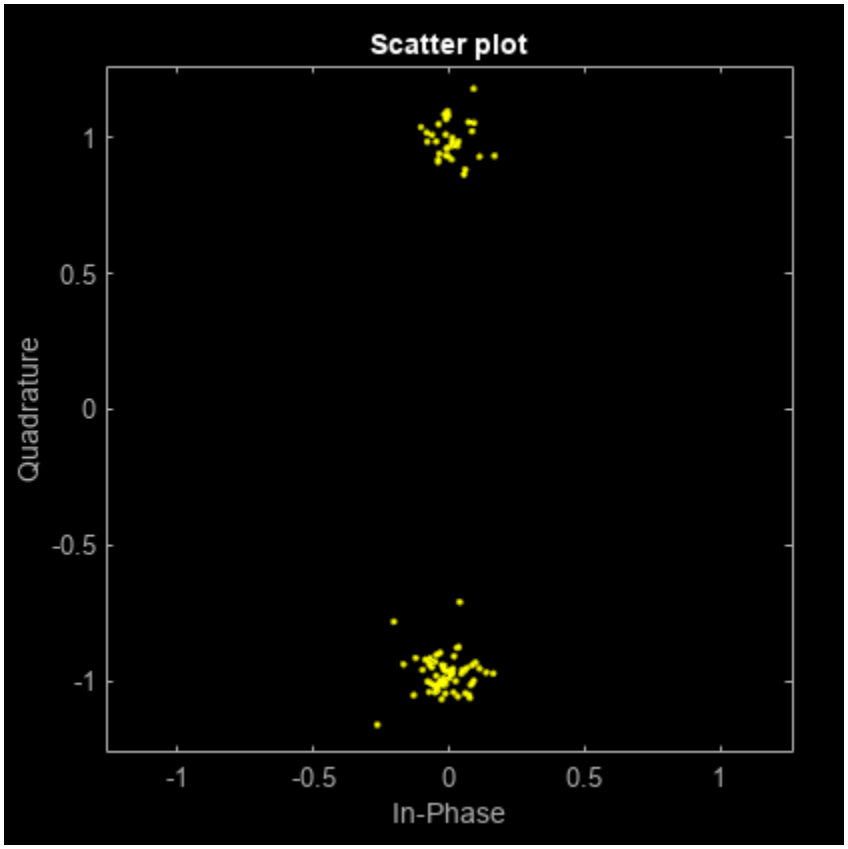
```
failCRC
```

```
failCRC = logical
   0
```

Because `failCRC` is `0`, HT-SIG passed the CRC check.

Visualize the scatter plot of the equalized symbols, `eqSym`.

```
scatterplot(eqSym(:))
```



Input Arguments

`rxSig` — Received HT-SIG field

matrix

Received HT-SIG field, specified as an N_S-by-N_R matrix. N_S is the number of samples and increases with channel bandwidth.

Channel Bandwidth	N_S
`'CBW20'`	160
`'CBW40'`	320

N_R is the number of receive antennas.

Data Types: `double`

`chEst` — Channel estimate

vector | 3-D array

Channel estimate, specified as an N_{ST}-by-1-by-N_R array. N_{ST} is the number of occupied subcarriers and increases with channel bandwidth.

Channel Bandwidth	N_{ST}
`'CBW20'`	52
`'CBW40'`	104

N_R is the number of receive antennas.

The channel estimate is based on the “L-LTF” on page 3-289.

`noiseVarEst` — Noise variance estimate

nonnegative scalar

Noise variance estimate, specified as a nonnegative scalar.

Data Types: `double`

`cbw` — Channel bandwidth

`'CBW20'` | `'CBW40'`

Channel bandwidth in MHz, specified as `'CBW20'` or `'CBW40'`.

Data Types: `char` | `string`

Name-Value Pair Arguments

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

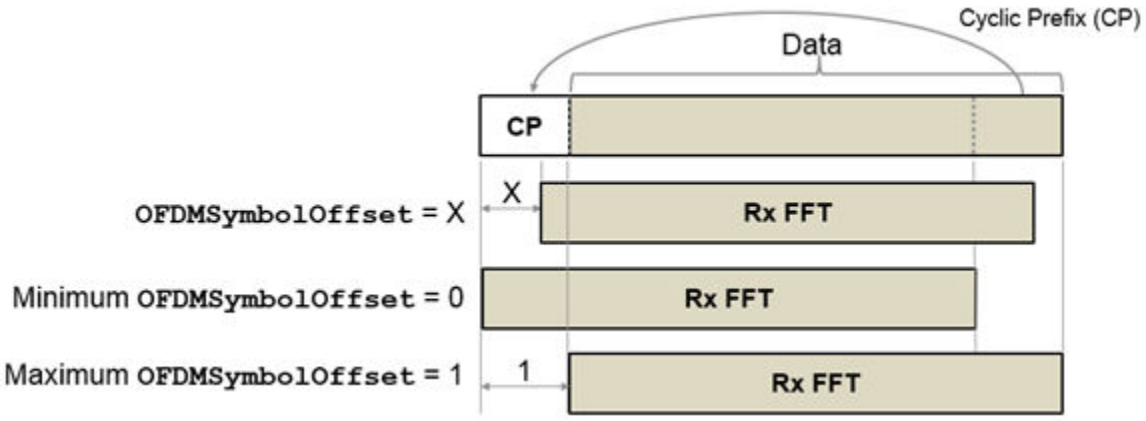
Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'PilotPhaseTracking','None'` disables pilot phase tracking.

`OFDMSymbolOffset` — OFDM symbol sampling offset

`0.75` (default) | scalar in the interval [0, 1]

OFDM symbol sampling offset represented as a fraction of the cyclic prefix (CP) length, specified as the comma-separated pair consisting of `'OFDMSymbolOffset'` and a scalar in the interval [0, 1]. The value you specify indicates the start location for OFDM demodulation relative to the beginning of the CP. The value `0` represents the start of the CP, and the value `1` represents the end of the CP.



Data Types: `double`

`EqualizationMethod` — Equalization method

`'MMSE'` (default) | `'ZF'`

Equalization method, specified as one of these values.

- `'MMSE'` — The receiver uses a minimum mean-square error equalizer.
- `'ZF'` — The receiver uses a zero-forcing equalizer.

When the received signal has multiple receive antennas, the function exploits receiver diversity during equalization. When the number of transmitted space-time streams is one and you specify this argument as `'ZF'`, the function performs maximal-ratio combining.

Data Types: `char` | `string`

`PilotPhaseTracking` — Pilot phase tracking

`'PreEQ'` (default) | `'None'`

Pilot phase tracking, specified as the comma-separated pair consisting of `'PilotPhaseTracking'` and one of these values.

- `'PreEQ'` — Enable pilot phase tracking, which the function performs before any equalization operation.
- `'None'` — Disable pilot phase tracking.

Data Types: `char` | `string`

Output Arguments

`recBits` — Recovered HT-SIG information

vector

Recovered HT-SIG information bits, returned as a 48-element column vector. The number of elements corresponds to the length of the HT-SIG field.

`failCRC` — CRC failure status

`true` | `false`

CRC failure status, returned as a logical scalar. If HT-SIG fails the CRC check, `failCRC` is `true`.

`eqSym` — Equalized symbols

matrix

Equalized symbols, returned as a 48-by-2 matrix corresponding to 48 data subcarriers and 2 OFDM symbols.

`cpe` — Common phase error

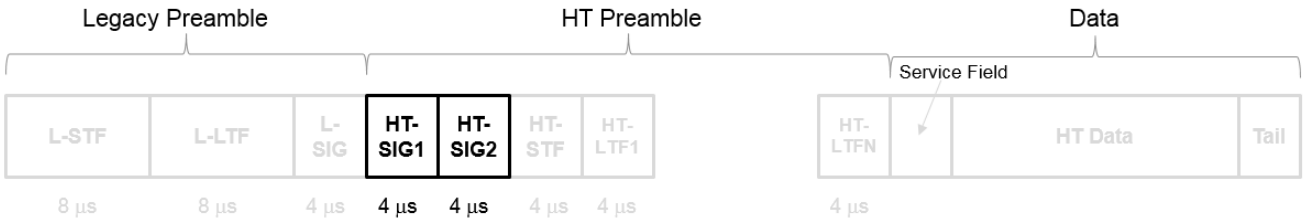
column vector

Common phase error in radians, returned as a 2-by-1 column vector.

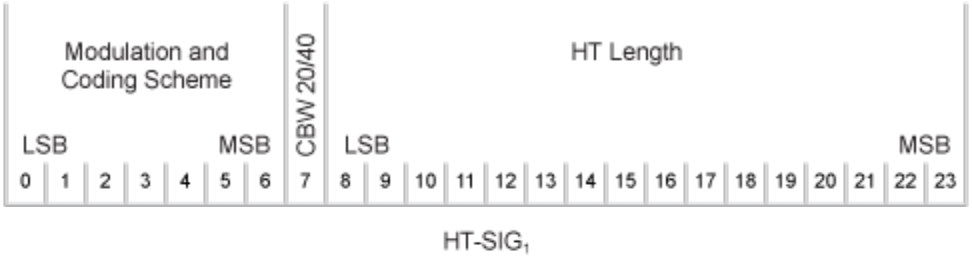
More About

HT-SIG

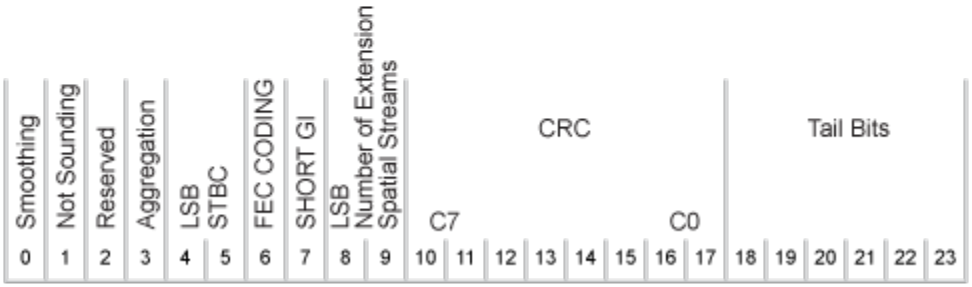
The high throughput signal (HT-SIG) field is located between the L-SIG field and HT-STF and is part of the HT-mixed format preamble. It is composed of two symbols, HT-SIG_1 and HT-SIG_2.



HT-SIG carries information used to decode the HT packet, including the MCS, packet length, FEC coding type, guard interval, number of extension spatial streams, and whether there is payload aggregation. The HT-SIG symbols are also used for auto-detection between HT-mixed format and legacy OFDM packets.



$HT\text{-}SIG_1$

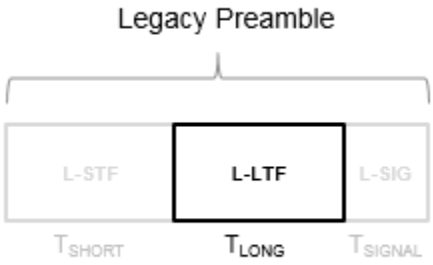


$HT\text{-}SIG_2$

For a detailed description of the HT-SIG field, see section 19.3.9.4.3 of IEEE Std 802.11-2020.

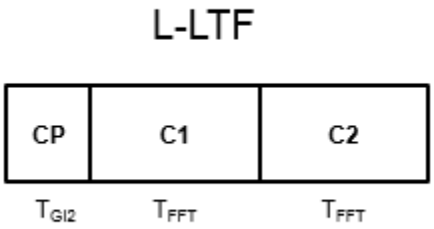
L-LTF

The L-LTF is the second field in the 802.11 OFDM PLCP legacy preamble. The L-LTF is a component of HE, VHT, HT, and non-HT PPDUs.



Channel estimation, fine frequency offset estimation, and fine symbol timing offset estimation rely on the L-LTF.

The L-LTF is composed of a cyclic prefix (CP) followed by two identical long training symbols (C1 and C2). The CP consists of the second half of the long training symbol.



The L-LTF duration varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period ($T_{FFT} = 1 / \Delta_F$)	Cyclic Prefix or Training Symbol Guard Interval (GI2) Duration ($T_{GI2} = T_{FFT} / 2$)	L-LTF Duration ($T_{LONG} = T_{GI2} + 2 \times T_{FFT}$)
20, 40, 80, 160, and 320	312.5	3.2 µs	1.6 µs	8 µs
10	156.25	6.4 µs	3.2 µs	16 µs
5	78.125	12.8 µs	6.4 µs	32 µs

Version History

Introduced in R2015b

References

[1] IEEE Std 802.11™-2012 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanHTSIG` | `wlanHTConfig` | `wlanHTDataRecover`

wlanHTSTF

Generate HT-STF waveform

Syntax

```
y = wlanHTSTF(cfg)
y = wlanHTSTF(cfg,OversamplingFactor=osf)
```

Description

`y = wlanHTSTF(cfg)` generates an "HT-STF" on page 3-292[9] time-domain waveform for an "HT-mixed" on page 3-292 transmission parameterized by `cfg`.

`y = wlanHTSTF(cfg,OversamplingFactor=osf)` generates an oversampled HT-mixed waveform with the specified oversampling rate. For more information about oversampling, see "FFT-Based Oversampling" on page 3-293.

Examples

Generate HT Short Training Field

Create a `wlanHTConfig` object with a 40 MHz bandwidth.

```
cfg = wlanHTConfig('ChannelBandwidth','CBW40');
```

Generate an HT-STF. The function returns a complex output of 160 samples.

```
stf = wlanHTSTF(cfg);
size(stf)
```

```
ans = 1×2

   160     1
```

Change the channel bandwidth to 20 MHz and create a new HT-STF.

```
cfg.ChannelBandwidth = 'CBW20';
stf = wlanHTSTF(cfg);
```

Verify that the number of samples has been halved due to the bandwidth reduction.

```
size(stf)
```

```
ans = 1×2

    80     1
```

9 IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.

Input Arguments

`cfg` — Transmission parameters

`wlanHTConfig` object

Transmission parameters, specified as a `wlanHTConfig` object.

`osf` — Oversampling factor

1 (default) | scalar greater than or equal to 1

Oversampling factor, specified as a scalar greater than or equal to 1. The oversampled cyclic prefix length must be an integer number of samples. The resultant inverse fast Fourier transform (IFFT) length must be even.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

y — HT-STF waveform

matrix

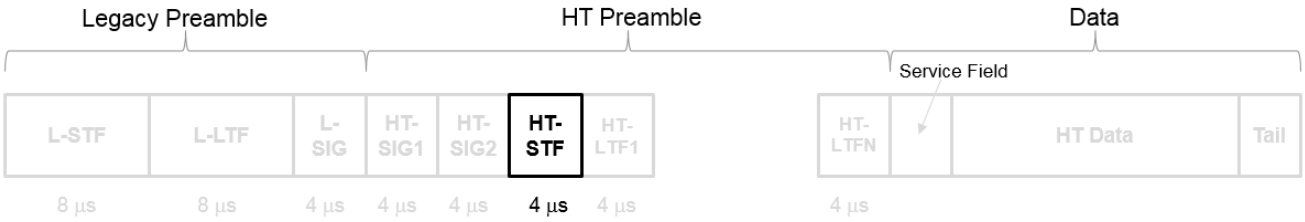
HT-STF waveform, returned as an N_S-by-N_T matrix. N_S is the number of samples, and N_T is the number of transmit antennas.

Data Types: `double`

More About

HT-STF

The high throughput short training field (HT-STF) is located between the HT-SIG and HT-LTF fields of an HT-mixed packet. The HT-STF is 4 µs in length and is used to improve automatic gain control estimation for a MIMO system. For a 20 MHz transmission, the frequency sequence used to construct the HT-STF is identical to that of the L-STF. For a 40 MHz transmission, the upper subcarriers of the HT-STF are constructed from a frequency-shifted and phase-rotated version of the L-STF.



HT-mixed

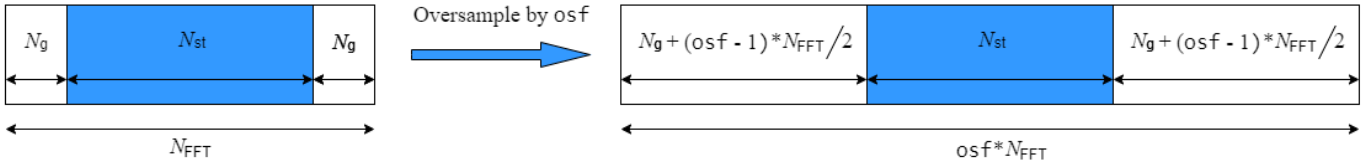
As described in IEEE Std 802.11-2012, Section 20.1.4, high throughput mixed (HT-mixed) format packets contain a preamble compatible with IEEE Std 802.11-2012, Section 18 and Section 19 receivers. Non-HT (Section 18 and Section19) STAs can decode the non-HT fields (L-STF, L-LTF, and L-SIG). The remaining preamble fields (HT-SIG, HT-STF, and HT-LTF) are for HT transmission, so the Section 18 and Section 19 STAs cannot decode them. The HT portion of the packet is described in IEEE Std 802.11-2012, Section 20.3.9.4. Support for the HT-mixed format is mandatory.

Algorithms

FFT-Based Oversampling

An oversampled signal is a signal sampled at a frequency that is higher than the Nyquist rate. WLAN signals maximize occupied bandwidth by using small guardbands, which can pose problems for anti-imaging and anti-aliasing filters. Oversampling increases guardband width relative to the total signal bandwidth, thereby increasing the number of samples in the signal.

This function performs oversampling by using a larger IFFT and zero pad when generating an OFDM waveform. This diagram shows the oversampling process for an OFDM waveform with N_{FFT} subcarriers comprising N_g guardband subcarriers on either side of N_{st} occupied bandwidth subcarriers.



Version History

Introduced in R2015b

References

[1] IEEE Std 802.11™-2012 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanHTConfig` | `wlanLSTF` | `wlanHTSIG` | `wlanHTLTF`

wlanInterpretScramblerState

Recover bandwidth signaling from initial scrambler state

Syntax

```
[bandwidth,dyn] = wlanInterpretScramblerState(scramInit)
[bandwidth,dyn] = wlanInterpretScramblerState(scramInit,idx1)
```

Description

`[bandwidth,dyn] = wlanInterpretScramblerState(scramInit)` recovers bandwidth signaling from initial scrambler state `scramInit`. The function returns `bandwidth`, the channel bandwidth, and `dyn`, an indication of whether bandwidth operation is dynamic or static.

`[bandwidth,dyn] = wlanInterpretScramblerState(scramInit,idx1)` specifies `idx1`, the *dot11CurrentChannelCenterFrequencyIndex1* parameter, as defined in Table 17-9 of [1].

Examples

Recover Bandwidth Signaling from Initial Scrambler State

Configure and generate a non-HT Data signal with a channel bandwidth of 160 MHz and dynamic bandwidth operation.

```
bandwidth = 'CBW160';
cfg = wlanNonHTConfig('ChannelBandwidth',bandwidth,'PSDULength',1, ...
    'SignalChannelBandwidth',true,'BandwidthOperation','Dynamic');
bits = randi([0 1],8*cfg.PSDULength,1,'int8');
[range,~] = scramblerRange(cfg);
scramInit = randi(range);
y = wlanNonHTData(bits,cfg,scramInit);
```

Transmit the waveform through an AWGN channel with an SNR of 50.

```
snr = 50;
noiseVarEst = 10^(-snr/10);
rx = awgn(y,snr);
```

Recover the frequency-domain signal by OFDM demodulating the non-HT Data signal, specifying an OFDM symbol sampling offset.

```
field = 'NonHT-Data';
symOffset = 0.5;
sym = wlanNonHTOFDMDemodulate(rx,field,bandwidth,'OFDMSymbolOffset',symOffset);
```

Extract the data subcarriers.

```
info = wlanNonHTOFDMInfo(field,bandwidth);
sym = sym(info.DataIndices,:);
```

Recover the first 20 MHz subchannel of the PSDU, enhancing the demapping of the OFDM subcarriers by specifying channel state information. Confirm that the received and transmitted PSDUs match.

```
csi = ones(48,1);
[psdu,scramInit] = wlanNonHTDataBitRecover(sym(1:48,:),noiseVarEst,csi,cfg);
isequal(bits,psdu)
```

```
ans = logical
   1
```

Recover and display bandwidth signaling by interpreting the scrambler state.

```
[bandwidth,dyn] = wlanInterpretScramblerState(scramInit)
```

```
bandwidth =
'CBW160'

dyn = logical
   1
```

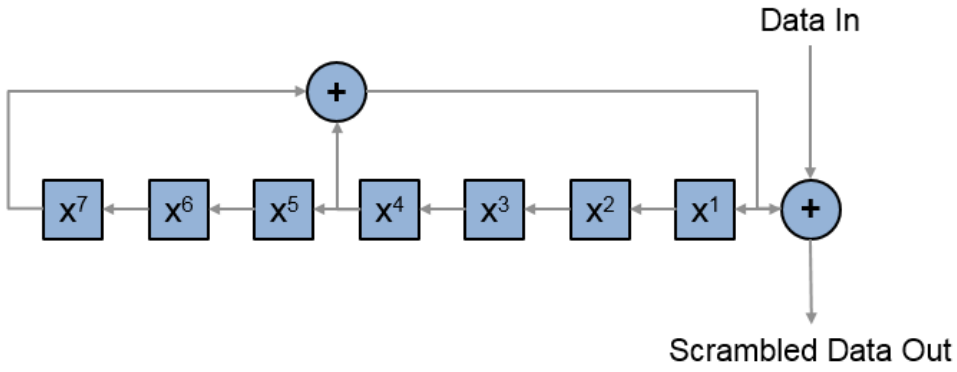
Input Arguments

`scramInit` — Initial scrambler state

integer in the interval [1, 127] | binary-valued column vector

Initial scrambler state, specified as an integer in the interval [1, 127], or the corresponding binary-valued column vector of length 7.

Section 17.3.5.5 of [1] specifies the scrambling and descrambling process applied to the transmitted data. The header and data fields that follow the scrambler initialization field (including data padding bits) are scrambled by XORing each bit with a length-127 periodic sequence generated by the polynomial $S(x) = x^7 + x^4 + 1$. The octets of the PSDU are placed into a bit stream, and, within each octet, bit 0 (LSB) is first and bit 7 (MSB) is last. This figure demonstrates the sequence generation and XOR operation.



Conversion from integer to bits uses left-MSB orientation. For example, initializing the scrambler with decimal 1, the bits map to these elements.

Element	X^7	X^6	X^5	X^4	X^3	X^2	X^1
Bit Value	0	0	0	0	0	0	1

To generate the bit stream equivalent to a decimal, use the `int2bit` function. For example, for decimal `1`:

```
int2bit(1,7)'
ans =

     0     0     0     0     0     0     1
```

Example: `[1; 0; 1; 1; 1; 0; 1]` conveys the scrambler initialization state of 93 as a binary-valued column vector.

Data Types: `double`

`idx1` — *dot11CurrentChannelCenterFrequencyIndex1* parameter

`0` (default) | scalar in the interval [0, 200]

dot11CurrentChannelCenterFrequencyIndex1 parameter, as defined in Table 17-9 of [1], specified as a scalar in the interval [0, 200].

Data Types: `double`

Output Arguments

`bandwidth` — Recovered channel bandwidth

`'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'` | `'CBW80+80'`

Recovered channel bandwidth, returned as one of these values.

- `'CBW20'` — Channel bandwidth of 20 MHz
- `'CBW40'` — Channel bandwidth of 40 MHz
- `'CBW80'` — Channel bandwidth of 80 MHz
- `'CBW160'` — Channel bandwidth of 160 MHz
- `'CBW80+80'` — Channel bandwidth of 160 MHz comprising two non-adjacent 80 MHz channels

Data Types: `char`

dyn — Indication of whether bandwidth operation is dynamic or static
`0` | `1`

Indication of whether bandwidth operation is dynamic or static, returned as a logical `1` or `0`. A value of `0` indicates static bandwidth operation. A value of `1` indicates dynamic bandwidth operation.

Data Types: `logical`

Version History

Introduced in R2020b

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

Extended Capabilities

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

See Also

Functions
`wlanNonHTDataBitRecover` | `wlanScramble`

wlanLLTF

Generate L-LTF waveform

Syntax

```
y = wlanLLTF(cfg)
y = wlanLLTF(cfg,OversamplingFactor=osf)
```

Description

`y = wlanLLTF(cfg)` generates an "L-LTF" on page 3-300[10] time-domain waveform with transmission parameters `cfg`.

`y = wlanLLTF(cfg,OversamplingFactor=osf)` generates an oversampled L-LTF waveform with the specified oversampling factor. For more information about oversampling, see "FFT-Based Oversampling" on page 3-301.

Examples

Generate L-LTF Waveform

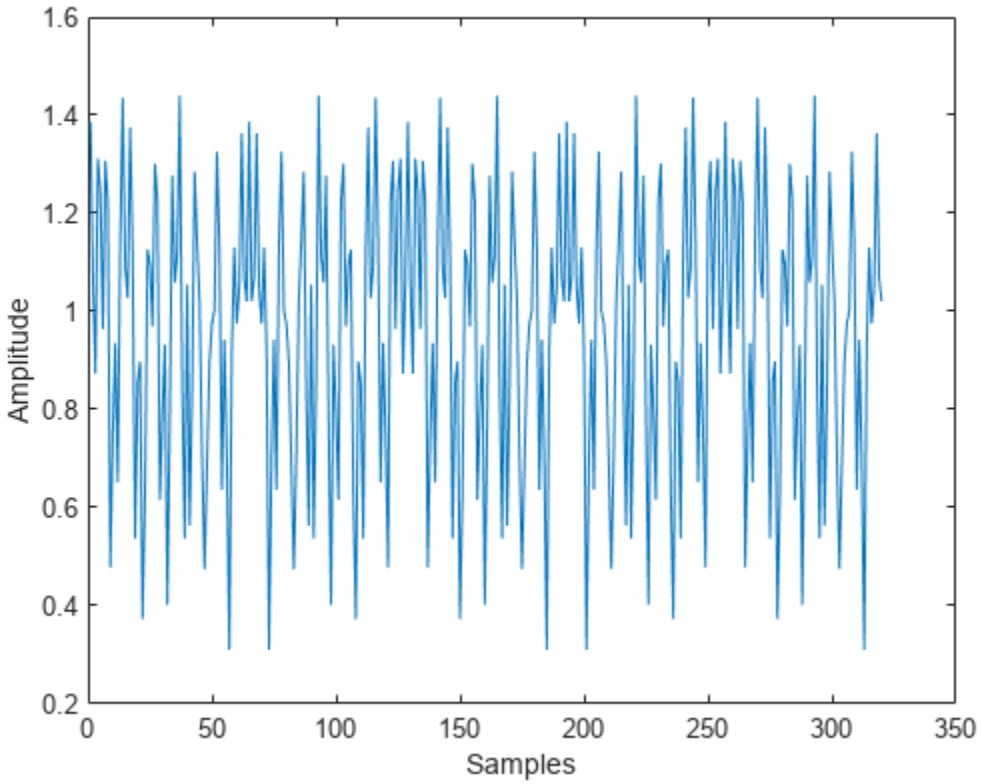
Generate the L-LTF for a 40 MHz single antenna VHT packet.

```
cfgVHT = wlanVHTConfig('ChannelBandwidth', 'CBW40');
y = wlanLLTF(cfgVHT);
size(y)
```

```
ans = 1×2

   320     1
```

```
plot(abs(y))
xlabel('Samples')
ylabel('Amplitude')
```

10 IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.



The output L-LTF waveform contains 320 time-domain samples for a 40 MHz channel bandwidth.

Input Arguments

`cfg` — Transmission parameters

`wlanVHTConfig` object | `wlanHTConfig` object | `wlanNonHTConfig` object

Transmission parameters, specified as a `wlanVHTConfig`, `wlanHTConfig`, or `wlanNonHTConfig` object.

`osf` — Oversampling factor

1 (default) | scalar greater than or equal to 1

Oversampling factor, specified as a scalar greater than or equal to 1. The oversampled cyclic prefix length must be an integer number of samples.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`y` — L-LTF time-domain waveform

matrix

"L-LTF" on page 3-300 time-domain waveform, returned as an N_S-by-N_T matrix. N_S is the number of time-domain samples, and N_T is the number of transmit antennas.

N_S is proportional to the channel bandwidth. The time-domain waveform consists of two symbols.

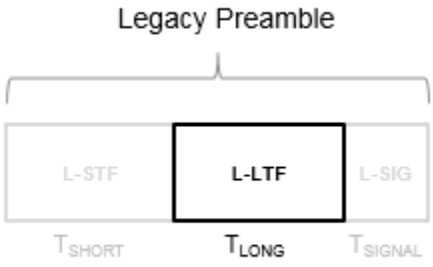
ChannelBandwidth	N_S
'CBW5', 'CBW10', 'CBW20'	160
'CBW40'	320
'CBW80'	640
'CBW160'	1280
'CBW320'	2560

Data Types: double
Complex Number Support: Yes

More About

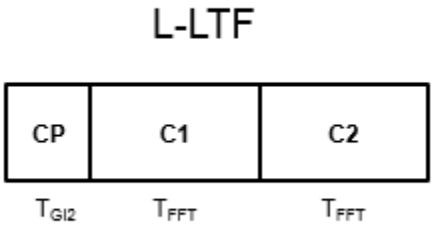
L-LTF

The L-LTF is the second field in the 802.11 OFDM PLCP legacy preamble. The L-LTF is a component of HE, VHT, HT, and non-HT PPDUs.



Channel estimation, fine frequency offset estimation, and fine symbol timing offset estimation rely on the L-LTF.

The L-LTF is composed of a cyclic prefix (CP) followed by two identical long training symbols (C1 and C2). The CP consists of the second half of the long training symbol.



The L-LTF duration varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period ($T_{FFT} = 1 / \Delta_F$)	Cyclic Prefix or Training Symbol Guard Interval (GI2) Duration ($T_{GI2} = T_{FFT} / 2$)	L-LTF Duration ($T_{LONG} = T_{GI2} + 2 \times T_{FFT}$)
20, 40, 80, 160, and 320	312.5	3.2 µs	1.6 µs	8 µs
10	156.25	6.4 µs	3.2 µs	16 µs
5	78.125	12.8 µs	6.4 µs	32 µs

Algorithms

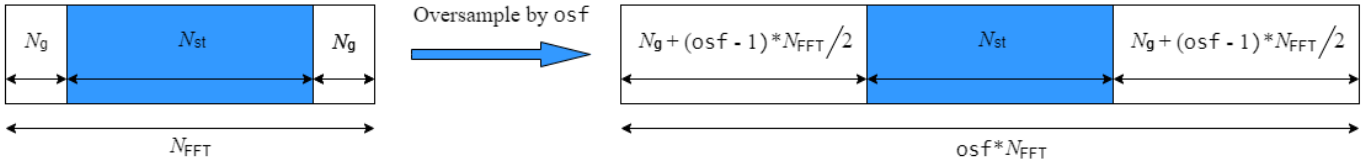
L-LTF Processing

The “L-LTF” on page 3-300 is two OFDM symbols long and follows the L-STF of the preamble in the packet structure for the VHT, HT, and non-HT formats. For algorithm details, refer to IEEE Std 802.11ac-2013 [1], Section 22.3.8.2.3 and IEEE Std 802.11-2012 [2], Section 20.3.9.3.4.

FFT-Based Oversampling

An oversampled signal is a signal sampled at a frequency that is higher than the Nyquist rate. WLAN signals maximize occupied bandwidth by using small guardbands, which can pose problems for anti-imaging and anti-aliasing filters. Oversampling increases guardband width relative to the total signal bandwidth, thereby increasing the number of samples in the signal.

This function performs oversampling by using a larger IFFT and zero pad when generating an OFDM waveform. This diagram shows the oversampling process for an OFDM waveform with N_{FFT} subcarriers comprising N_g guardband subcarriers on either side of N_{st} occupied bandwidth subcarriers.



Version History

Introduced in R2015b

References

[1] IEEE Std 802.11ac™-2013 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications — Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz.

[2] IEEE Std 802.11™-2012 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanVHTConfig` | `wlanHTConfig` | `wlanNonHTConfig` | `wlanLSTF` | `wlanLSIG` | `wlanLLTFDemodulate` | `wlanLLTFChannelEstimate`

wlanLLTFChannelEstimate

Channel estimation using L-LTF

Syntax

```
chEst = wlanLLTFChannelEstimate(demodSig,cfg)
chEst = wlanLLTFChannelEstimate(demodSig,cbw)
chEst = wlanLLTFChannelEstimate( ___ ,span)
```

Description

`chEst = wlanLLTFChannelEstimate(demodSig,cfg)` returns the channel estimate between the transmitter and all receive antennas using the demodulated "L-LTF" on page 3-310[11], `demodSig`, given the parameters specified in configuration object `cfg`.

`chEst = wlanLLTFChannelEstimate(demodSig,cbw)` returns the channel estimate given channel bandwidth `cbw`. The channel bandwidth can be used instead of the configuration object.

`chEst = wlanLLTFChannelEstimate( ___ ,span)` returns the channel estimate and performs frequency smoothing over the specified filter span. For more information, see "Frequency Smoothing" on page 3-311.

This syntax supports input options from prior syntaxes.

Examples

Estimate SISO Channel Using L-LTF

Create VHT format configuration object. Generate a time-domain waveform for an 802.11ac VHT packet.

```
vht = wlanVHTConfig;
txWaveform = wlanWaveformGenerator([1;0;0;1],vht);
```

Multiply the transmitted VHT signal by -0.1 + 0.5i and pass it through an AWGN channel with a 30 dB signal-to-noise ratio.

```
rxWaveform = awgn(txWaveform*(-0.1+0.5i),30);
```

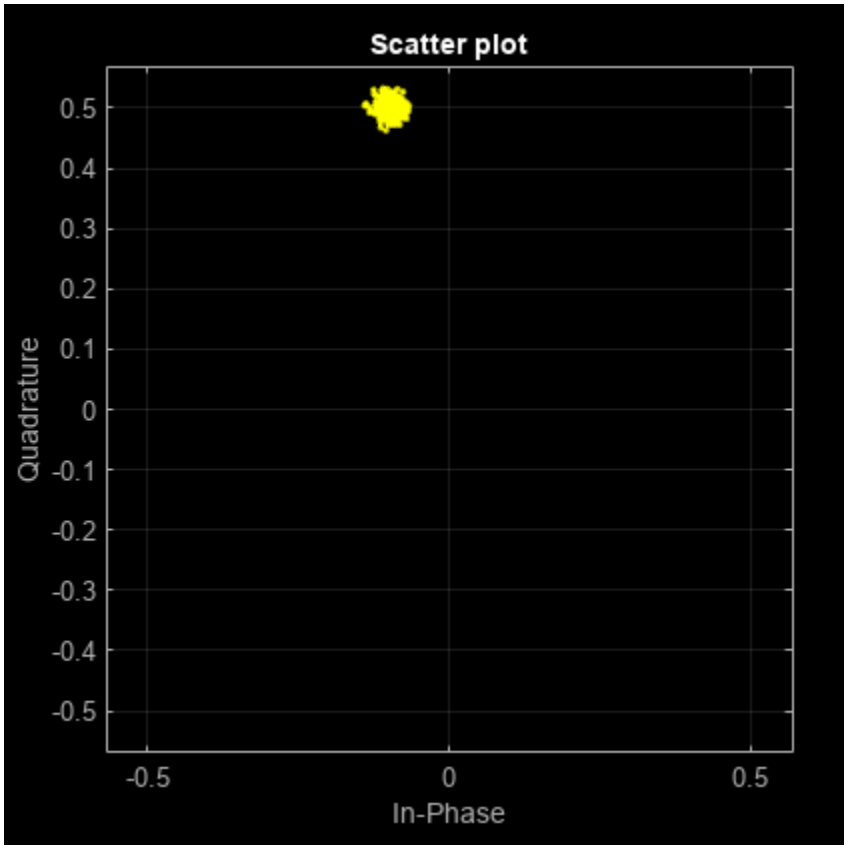
Extract the L-LTF field indices and demodulate the L-LTF. Perform channel estimation without frequency smoothing.

```
idxLLTF = wlanFieldIndices(vht,'L-LTF');
demodSig = wlanLLTFDemodulate(rxWaveform(idxLLTF(1):idxLLTF(2),:),vht);

est = wlanLLTFChannelEstimate(demodSig,vht);
```

Plot the channel estimate.

11 IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.

```
scatterplot(est)
grid
```



The channel estimate matches the complex channel multiplier.

L-LTF Channel Estimation After TGn Channel

Generate a time domain waveform for an 802.11n HT packet, pass it through a TGn fading channel and perform L-LTF channel estimation. Trailing zeros are added to the waveform to allow for TGn channel delay.

Create the HT packet configuration and transmit waveform.

```
cfgHT = wlanHTConfig;
txWaveform = wlanWaveformGenerator([1;0;0;1],cfgHT);
```

Configure a TGn channel with 20 MHz bandwidth.

```
tgnChannel = wlanTGnChannel;
tgnChannel.SampleRate = 20e6;
```

Pass the waveform through the TGn channel, adding trailing zeros to allow for channel delay.

```
rxWaveform = tgnChannel([txWaveform; zeros(15,1)]);
```

Skip the first four samples to synchronize the received waveform for channel delay.

```
rxWaveform = rxWaveform(5:end,:);
```

Extract the L-LTF and perform channel estimation.

```
idnLLTF = wlanFieldIndices(cfgHT,'L-LTF');
sym = wlanLLTFDemodulate(rxWaveform(idnLLTF(1):idnLLTF(2),:),cfgHT);
est = wlanLLTFChannelEstimate(sym,cfgHT);
```

Estimate 80 MHz SISO Channel Using L-LTF

Create a VHT format configuration object. Using these objects, generate a time-domain waveform for an 802.11ac VHT packet.

```
vht = wlanVHTConfig('ChannelBandwidth','CBW80');
txWaveform = wlanWaveformGenerator([1;0;0;1],vht);
```

Multiply the transmitted VHT signal by -0.4 + 0.3i and pass it through an AWGN channel.

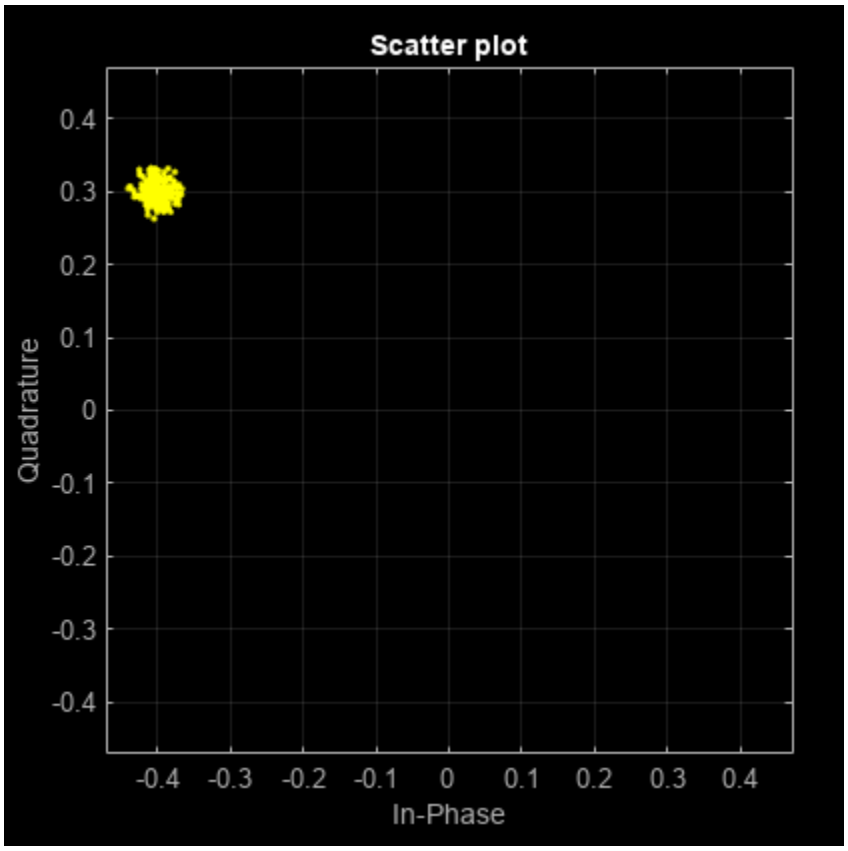
```
rxWaveform = awgn(txWaveform*(-0.4+0.3i),30);
```

Specify the channel bandwidth for demodulation and channel estimation. Extract the L-LTF field indices, demodulate the L-LTF, and perform channel estimation without frequency smoothing.

```
chanBW = 'CBW80';
idxLLTF = wlanFieldIndices(vht,'L-LTF');
demodSig = wlanLLTFDemodulate(rxWaveform(idxLLTF(1):idxLLTF(2),:),chanBW);
est = wlanLLTFChannelEstimate(demodSig,chanBW);
```

Plot the channel estimate.

```
scatterplot(est)
grid
```



The channel estimate matches the complex channel multiplier.

Estimate SISO Channel Using L-LTF and Smoothing Filter

Create a VHT format configuration object. Generate a time-domain waveform for an 802.11ac VHT packet.

```
vht = wlanVHTConfig;
txWaveform = wlanWaveformGenerator([1;0;0;1],vht);
```

Multiply the transmitted VHT signal by 0.2 - 0.6i and pass it through an AWGN channel having a 10 dB SNR.

```
rxWaveform = awgn(txWaveform*complex(0.2,-0.6),10);
```

Extract the L-LTF from the received waveform. Demodulate the L-LTF.

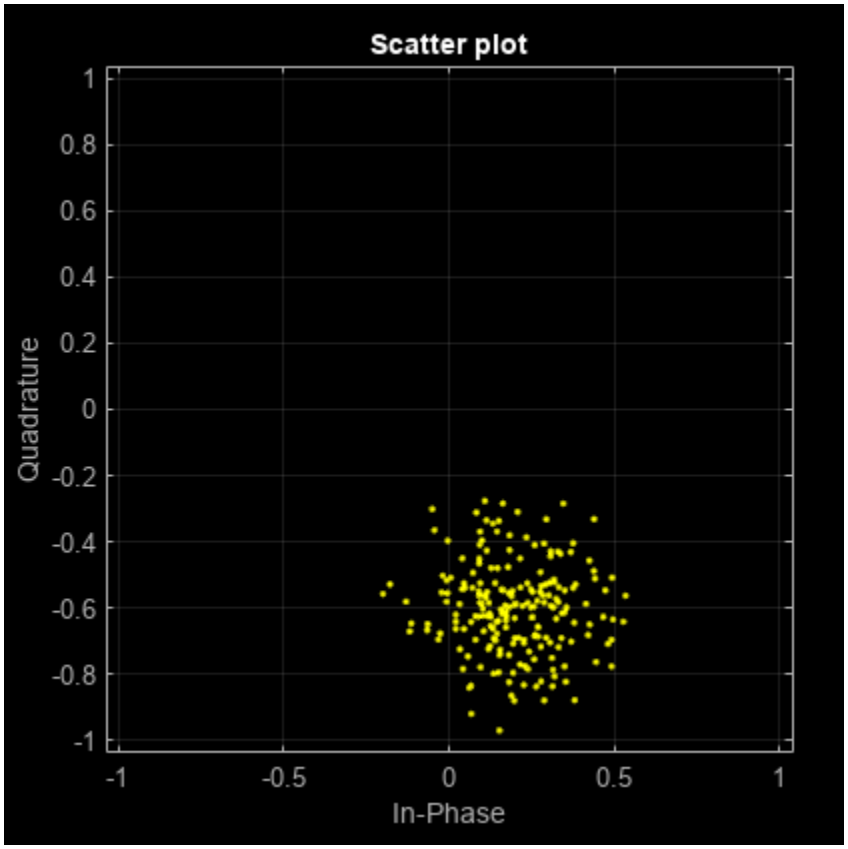
```
idxLLTF = wlanFieldIndices(vht, 'L-LTF');
lltfDemodSig = wlanLLTFDemodulate(rxWaveform(idxLLTF(1):idxLLTF(2),:),vht);
```

Use the demodulated L-LTF signal to generate the channel estimate.

```
est = wlanLLTFChannelEstimate(lltfDemodSig,vht);
```

Plot the channel estimate.

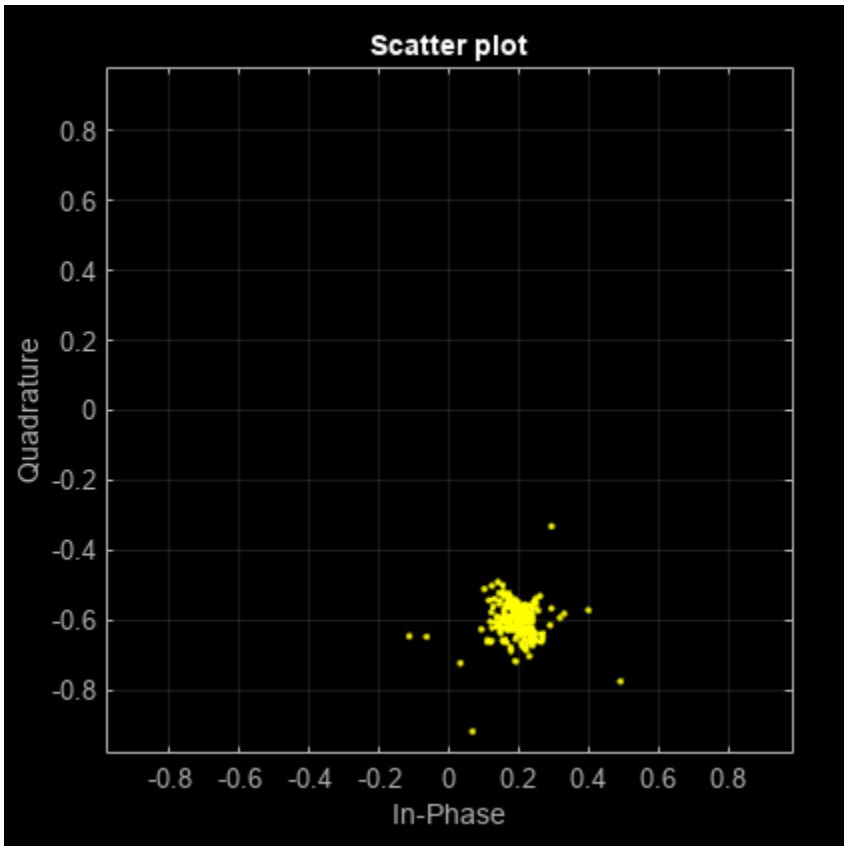
```
scatterplot(est)
grid
```



The channel estimate is noisy, which may lead to inaccurate data recovery.

Estimate the channel again with the filter span set to 11.

```
est = wlanLLTFChannelEstimate(lltfDemodSig,vht,11);
scatterplot(est)
grid
```



The filtering provides a better channel estimate.

Estimate Channel with L-LTF and Recover VHT-SIG-A

Create a VHT format configuration object. Generate L-LTF and VHT-SIG-A fields.

```
vht = wlanVHTConfig;
txLLTF = wlanLLTF(vht);
txSig = wlanVHTSIGA(vht);
```

Create a TGac channel for an 80 MHz bandwidth and a Model-A delay profile. Pass the transmitted L-LTF and VHT-SIG-A signals through the channel.

```
tgacChan = wlanTGacChannel('SampleRate',80e6,'ChannelBandwidth','CBW80', ...
    'DelayProfile','Model-A');

rxLLTFNoNoise = tgacChan(txLLTF);
rxSigNoNoise = tgacChan(txSig);
```

Create an AWGN noise channel with an SNR = 15 dB. Add the AWGN noise to L-LTF and VHT-SIG-A signals.

```
chNoise = comm.AWGNChannel('NoiseMethod','Signal to noise ratio (SNR)', ...
    'SNR',15);
```

```
rxLLTF = chNoise(rxLLTFNoNoise);
rxSig = chNoise(rxSigNoNoise);
```

Create an AWGN channel having a noise variance corresponding to a 9 dB noise figure receiver. Pass the faded signals through the AWGN channel.

```
nVar = 10^((-228.6 + 10*log10(290) + 10*log10(80e6) + 9)/10);
awgnChan = comm.AWGNChannel('NoiseMethod','Variance','Variance',nVar);

rxLLTF = awgnChan(rxLLTF);
rxSig = awgnChan(rxSig);
```

Demodulate the received L-LTF.

```
demodLLTF = wlanLLTFDemodulate(rxLLTF,vht);
```

Estimate the channel using the demodulated L-LTF.

```
chEst = wlanLLTFChannelEstimate(demodLLTF,vht);
```

Recover the VHT-SIG-A signal and verify that there was no CRC failure.

```
[recBits,crcFail] = wlanVHTSIGARecover(rxSig,chEst,nVar,'CBW80');
crcFail
```

```
crcFail = logical
   0
```

Input Arguments

demodSig — Demodulated L-LTF OFDM symbols

3-D array

Demodulated L-LTF OFDM symbols, specified as an N_{ST}-by-N_{SYM}-by-N_R array. N_{ST} is the number of occupied subcarriers. N_{SYM} is the number of demodulated L-LTF symbols (one or two). N_R is the number of receive antennas. Each column of the 3-D array is a demodulated L-LTF OFDM symbol. If you specify two L-LTF symbols, `wlanLLTFChannelEstimate` averages the channel estimate over both symbols.

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

cfg — Format configuration

`wlanVHTConfig` object | `wlanHTConfig` object | `wlanNonHTConfig` object

Format configuration, specified as one of these objects:

- `wlanVHTConfig` for VHT format
- `wlanHTConfig` for HT format
- `wlanNonHTConfig` for non-HT format

The `wlanLLTFChannelEstimate` function uses the `ChannelBandwidth` property of `cfg`.

cbw — Channel bandwidth

`'CBW5'` | `'CBW10'` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'` | `'CBW320'`

Channel bandwidth, specified as one of these values.

- `'CBW5'` – Channel bandwidth of 5 MHz
- `'CBW10'` – Channel bandwidth of 10 MHz
- `'CBW20'` – Channel bandwidth of 20 MHz
- `'CBW40'` – Channel bandwidth of 40 MHz
- `'CBW80'` – Channel bandwidth of 80 MHz
- `'CBW160'` – Channel bandwidth of 160 MHz
- `'CBW320'` – Channel bandwidth of 320 MHz

Data Types: `char` | `string`

`span` — Filter span
positive odd integer

Filter span of the frequency smoothing filter, specified as a positive odd integer and expressed as a number of subcarriers. Frequency smoothing is applied only when `span` is specified and is greater than one. See “Frequency Smoothing” on page 3-311.

Data Types: `single` | `double`

Output Arguments

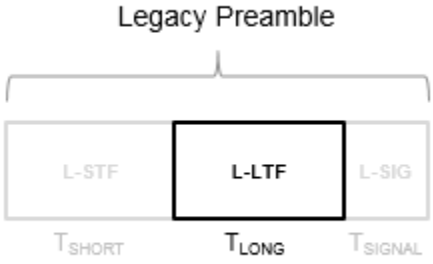
`chEst` — Channel estimate
3-D array

Channel estimate containing data and pilot subcarriers, returned as an N_{ST}-by-1-by-N_R array. N_{ST} is the number of occupied subcarriers. The value of 1 corresponds to the single transmitted stream in the L-LTF. N_R is the number of receive antennas.

More About

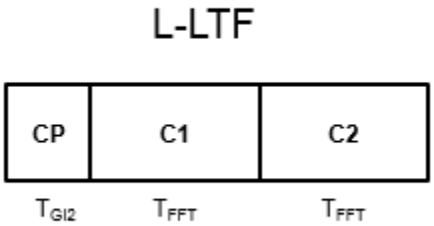
L-LTF

The L-LTF is the second field in the 802.11 OFDM PLCP legacy preamble. The L-LTF is a component of HE, VHT, HT, and non-HT PPDUs.



Channel estimation, fine frequency offset estimation, and fine symbol timing offset estimation rely on the L-LTF.

The L-LTF is composed of a cyclic prefix (CP) followed by two identical long training symbols (C1 and C2). The CP consists of the second half of the long training symbol.



The L-LTF duration varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period ($T_{FFT} = 1 / \Delta_F$)	Cyclic Prefix or Training Symbol Guard Interval (GI2) Duration ($T_{GI2} = T_{FFT} / 2$)	L-LTF Duration ($T_{LONG} = T_{GI2} + 2 \times T_{FFT}$)
20, 40, 80, 160, and 320	312.5	3.2 µs	1.6 µs	8 µs
10	156.25	6.4 µs	3.2 µs	16 µs
5	78.125	12.8 µs	6.4 µs	32 µs

Frequency Smoothing

Frequency smoothing can improve channel estimation by averaging out noise.

Frequency smoothing is recommended only for cases in which a single transmit antenna is used. Frequency smoothing consists of applying a moving-average filter that spans multiple adjacent subcarriers. Channel conditions dictate whether frequency smoothing is beneficial.

- If adjacent subcarriers are highly correlated, frequency smoothing results in significant noise reduction.
- In a highly frequency-selective channel, smoothing can degrade the quality of the channel estimate.

Version History

Introduced in R2015b

Single precision support

You can specify the function's numeric inputs as single precision values.

References

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

[2] IEEE Std 802.11™-2016 (Revision of IEEE Std 802.11-2012). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements.

Extended Capabilities

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

See Also

`wlanNonHTConfig` | `wlanHTConfig` | `wlanVHTConfig` | `wlanLLTFDemodulate` | `wlanHTLTFChannelEstimate` | `wlanVHTLTFChannelEstimate`

wlanLLTFDemodulate

Demodulate L-LTF waveform

Syntax

```
y = wlanLLTFDemodulate(x,cbw)
y = wlanLLTFDemodulate(x,cfg)
y = wlanLLTFDemodulate( ___ ,symOffset)
```

Description

`y = wlanLLTFDemodulate(x,cbw)` returns the demodulated "L-LTF" on page 3-316[12] waveform given time-domain input signal `x` and channel bandwidth `cbw`.

`y = wlanLLTFDemodulate(x,cfg)` returns the demodulated L-LTF given the format configuration object, `cfg`.

`y = wlanLLTFDemodulate( ___ ,symOffset)` specifies the OFDM symbol offset, `symOffset`, using any of the arguments from the previous syntaxes.

Examples

Demodulate L-LTF for Non-HT Format Transmission

Demodulate the L-LTF used in a non-HT OFDM transmission, after passing the L-LTF through an AWGN channel.

Create a non-HT configuration object and use it to generate an L-LTF signal.

```
cfg = wlanNonHTConfig;
txSig = wlanLLTF(cfg);
```

Pass the L-LTF signal through an AWGN channel. Demodulate the received signal.

```
rxSig = awgn(txSig,15,'measured');
y = wlanLLTFDemodulate(rxSig,'CBW20');
```

Demodulate L-LTF for VHT Format Transmission

Demodulate the L-LTF used in a VHT transmission, after passing the L-LTF through an AWGN channel.

Create a VHT configuration object and use it to generate an L-LTF signal.

```
cfg = wlanVHTConfig;
txSig = wlanLLTF(cfg);
```

12 IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.

Pass the L-LTF signal through an AWGN channel.

```
rxSig = awgn(txSig,5);
```

Demodulate the received L-LTF using the information from the `wlanVHTConfig` object.

```
y = wlanLLTFDemodulate(rxSig,cfg);
```

Demodulate L-LTF with OFDM Symbol Offset

Demodulate the L-LTF for the HT-mixed transmission format, given a custom OFDM symbol offset.

Set the channel bandwidth to 40 MHz and the OFDM symbol offset to 1. That way, the FFT takes place after the guard interval.

```
cbw = 'CBW40';
ofdmSymOffset = 1;
```

Create an HT configuration object and use it to generate an L-LTF signal.

```
cfg = wlanHTConfig('ChannelBandwidth',cbw);
txSig = wlanLLTF(cfg);
```

Pass the L-LTF signal through an AWGN channel.

```
rxSig = awgn(txSig,10);
```

Demodulate the received L-LTF using a custom OFDM symbol offset.

```
y = wlanLLTFDemodulate(rxSig,'CBW40',ofdmSymOffset);
```

Input Arguments

x — Time-domain input signal

vector | matrix

Time-domain input signal corresponding to the L-LTF of the “PPDU” on page 3-317, specified as an N_S-by-N_R vector or matrix. N_S is the number of samples and N_R is the number of receive antennas.

N_S is proportional to the channel bandwidth. The time-domain waveform consists of two symbols.

`ChannelBandwidth`	N_S
`'CBW5'`, `'CBW10'`, `'CBW20'`	160
`'CBW40'`	320
`'CBW80'`	640
`'CBW160'`	1280
`'CBW320'`	2560

Data Types: `single` | `double`

`cbw` — Channel bandwidth
`'CBW5'` | `'CBW10'` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'` | `'CBW320'`

Channel bandwidth, specified as one of these values.

- `'CBW5'` – Channel bandwidth of 5 MHz
- `'CBW10'` – Channel bandwidth of 10 MHz
- `'CBW20'` – Channel bandwidth of 20 MHz
- `'CBW40'` – Channel bandwidth of 40 MHz
- `'CBW80'` – Channel bandwidth of 80 MHz
- `'CBW160'` – Channel bandwidth of 160 MHz
- `'CBW320'` – Channel bandwidth of 320 MHz

Data Types: `char` | `string`

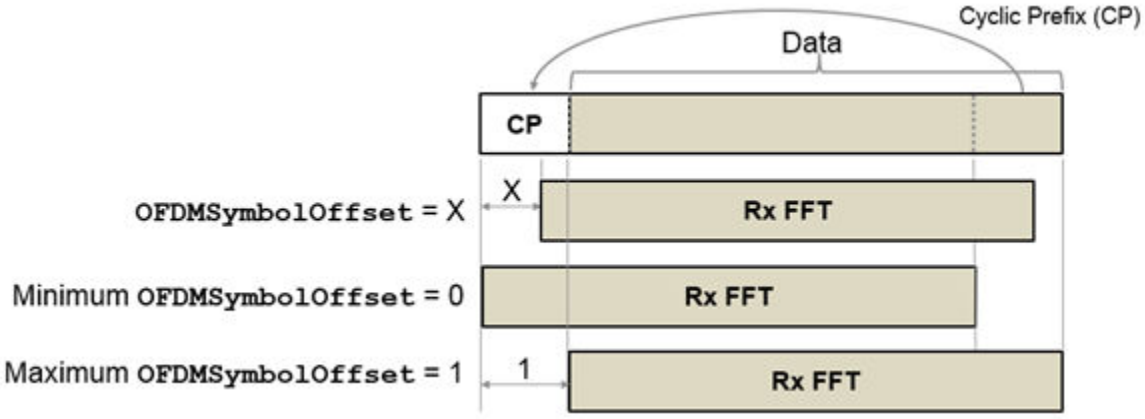
`cfg` — Format information
`wlanNonHTConfig` | `wlanHTConfig` | `wlanVHTConfig`

Format information, specified as a WLAN configuration object. To create these objects, see `wlanNonHTConfig`, `wlanHTConfig`, or `wlanVHTConfig`.

`symOffset` — OFDM symbol sampling offset
`0.75` (default) | scalar in the interval [0, 1]

OFDM symbol sampling offset, as a fraction of the cyclic prefix length, specified as a scalar in the interval [0, 1].

The value that you specify indicates the start location for OFDM demodulation relative to the beginning of the cyclic prefix.



Example: `0.45`

Data Types: `single` | `double`

Output Arguments

`y` — Demodulated L-LTF signal
3-D OFDM symbol array

Demodulated L-LTF signal, returned as an N_{ST}-by-N_{SYM}-by-N_R array. N_{ST} is the number of occupied subcarriers, N_{SYM} is the number of OFDM symbols, and N_R is the number of receive antennas. For the L-LTF, N_{SYM} is always 2.

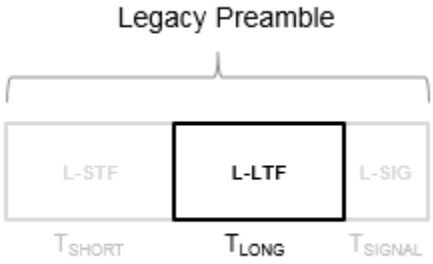
N_{ST} varies with channel bandwidth.

`ChannelBandwidth`	Number of Occupied Subcarriers (N_{ST})
`'CBW20'`, `'CBW10'`, `'CBW5'`	52
`'CBW40'`	104
`'CBW80'`	208
`'CBW160'`	416
`'CBW320'`	832

More About

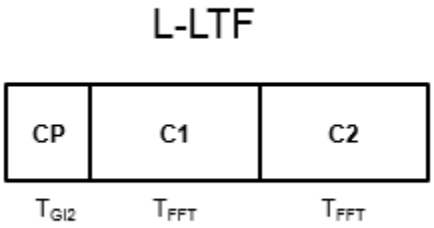
L-LTF

The L-LTF is the second field in the 802.11 OFDM PLCP legacy preamble. The L-LTF is a component of HE, VHT, HT, and non-HT PPDUs.



Channel estimation, fine frequency offset estimation, and fine symbol timing offset estimation rely on the L-LTF.

The L-LTF is composed of a cyclic prefix (CP) followed by two identical long training symbols (C1 and C2). The CP consists of the second half of the long training symbol.



The L-LTF duration varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period ($T_{FFT} = 1 / \Delta_F$)	Cyclic Prefix or Training Symbol Guard Interval (GI2) Duration ($T_{GI2} = T_{FFT} / 2$)	L-LTF Duration ($T_{LONG} = T_{GI2} + 2 \times T_{FFT}$)
20, 40, 80, 160, and 320	312.5	3.2 µs	1.6 µs	8 µs
10	156.25	6.4 µs	3.2 µs	16 µs
5	78.125	12.8 µs	6.4 µs	32 µs

PPDU

The PLCP protocol data unit (PPDU) is the complete “PLCP” on page 3-317 frame, including PLCP headers, MAC headers, the MAC data field, and the MAC and PLCP trailers [2].

PLCP

The physical layer convergence procedure (PLCP) is the upper component of the physical layer in 802.11 networks. Each physical layer has its own PLCP, which provides auxiliary framing to the MAC [2].

Version History

Introduced in R2015b

Single precision support

You can specify the function's numeric inputs as single precision values.

References

[1] IEEE Std 802.11™-2016 (Revision of IEEE Std 802.11-2012). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements.

[2] Gast, Matthew S. *802.11n: A Survival Guide*. Sebastopol, CA: O’Reilly Media Inc., 2012, p. 120.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanLLTF` | `wlanLLTFChannelEstimate`

wlanLSIG

Generate L-SIG waveform

Syntax

```
[y,bits] = wlanLSIG(cfgFormat)
[y,bits] = wlanLSIG(cfgFormat,OversamplingFactor=osf)
```

Description

`[y,bits] = wlanLSIG(cfgFormat)` generates an "L-SIG" on page 3-320[13] time-domain waveform using the specified transmission parameters.

`[y,bits] = wlanLSIG(cfgFormat,OversamplingFactor=osf)` generates an oversampled HT-LTF waveform with the specified oversampling factor. For more information about oversampling, see "FFT-Based Oversampling" on page 3-323.

Examples

Extract Rate Information from L-SIG

Create a non-HT configuration object. The default modulation and coding scheme (MCS) is `0`.

```
cfg = wlanNonHTConfig;
```

Generate the L-SIG waveform and information bits. Extract the rate from the returned bits.

```
[~,bits] = wlanLSIG(cfg);
```

Display the first four bits, which contain the rate information. As defined in Table 18-6 of IEEE Std 802.11™-2012, a value of `[1 1 0 1]` corresponds to a rate of 6 Mbps for 20 MHz channel spacing.

```
disp(bits(1:4)')
```

```
   1   1   0   1
```

Change the `MCS` to 7 then generate the corresponding L-SIG waveform and information bits. Extract the rate from the returned bits and analyze. The rate information is contained in the first four bits.

```
cfg.MCS = 7;
[y,bits] = wlanLSIG(cfg);
```

Display the first four bits. As defined in IEEE Std 802.11-2012, Table 18-6, a value of `[0 0 1 1]` corresponds to a rate of 54 Mbps for 20 MHz channel spacing.

```
disp(bits(1:4)')
```

```
   0   0   1   1
```

13 IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.

Generate Oversampled L-SIG Waveform for 80 MHz VHT Packet

Configure an 80 MHz VHT transmission.

```
cfgVHT = wlanVHTConfig(ChannelBandwidth="CBW80");
```

Specify an oversampling rate and generate the L-SIG waveform.

```
osf = 2;
[y,bits] = wlanLSIG(cfgVHT,OversamplingFactor=osf);
size(y)
```

```
ans = 1×2

   640     1
```

Input Arguments

`cfgFormat` — Transmission parameters

`wlanVHTConfig` object | `wlanHTConfig` object | `wlanNonHTConfig` object

Transmission parameters, specified as a `wlanVHTConfig`, `wlanHTConfig`, or `wlanNonHTConfig` object.

`osf` — Oversampling factor

1 (default) | scalar greater than or equal to 1

Oversampling factor, specified as a scalar greater than or equal to 1. The oversampled cyclic prefix length must be an integer number of samples.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`y` — L-SIG time-domain waveform

matrix

"L-SIG" on page 3-320 time-domain waveform, returned as an N_S-by-N_T matrix. N_S is the number of time-domain samples, and N_T is the number of transmit antennas.

N_S is proportional to the channel bandwidth.

`ChannelBandwidth`	N_S
`'CBW5'`, `'CBW10'`, `'CBW20'`	80
`'CBW40'`	160
`'CBW80'`	320
`'CBW160'`	640

Data Types: `double`
Complex Number Support: Yes

`bits` — Signaling bits
column vector

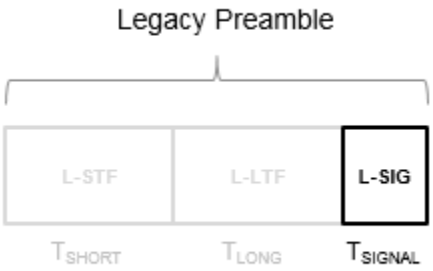
Signaling bits from the legacy signal field, returned as a 24-by-1 bit column vector. See “L-SIG” on page 3-320 for the bit field description.

Data Types: `int8`

More About

L-SIG

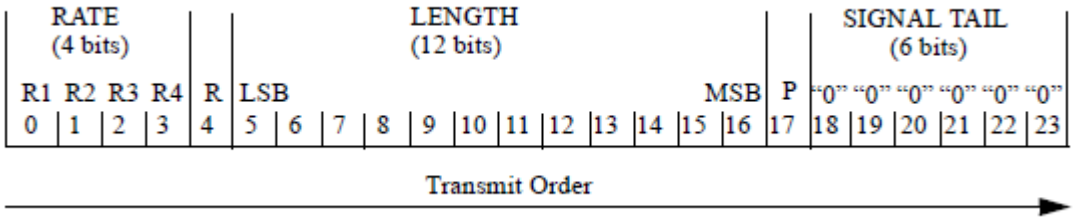
The L-SIG is the third field of the 802.11 OFDM PLCP legacy preamble. This field is a component of EHT, HE, VHT, HT, and non-HT PPDUs. It consists of 24 bits that contain rate, length, and parity information. The L-SIG field is transmitted using BPSK modulation with rate 1/2 binary convolutional coding (BCC).



The L-SIG is one OFDM symbol with a duration that varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period ($T_{FFT} = 1 / \Delta_F$)	Guard Interval (GI) Duration ($T_{GI} = T_{FFT} / 4$)	L-SIG Duration ($T_{SIGNAL} = T_{GI} + T_{FFT}$)
20, 40, 80, and 160	312.5	3.2 µs	0.8 µs	4 µs
10	156.25	6.4 µs	1.6 µs	8 µs
5	78.125	12.8 µs	3.2 µs	16 µs

The L-SIG contains packet information for the received configuration.



- Bits 0 through 3 specify the data rate (modulation and coding rate) for the non-HT format.

Rate (Bits 0–3)	Modulation	Coding Rate (*R*)	Data Rate (Mb/s)		
			20 MHz Channel Bandwidth	10 MHz Channel Bandwidth	5 MHz Channel Bandwidth
1101	BPSK	1/2	6	3	1.5
1111	BPSK	3/4	9	4.5	2.25
0101	QPSK	1/2	12	6	3
0111	QPSK	3/4	18	9	4.5
1001	16-QAM	1/2	24	12	6
1011	16-QAM	3/4	36	18	9
0001	64-QAM	2/3	48	24	12
0011	64-QAM	3/4	54	27	13.5

  For HT and VHT formats, the L-SIG rate bits are set to `'1 1 0 1'`. Data rate information for HT and VHT formats is signaled in format-specific signaling fields.
- Bit 4 is reserved for future use.
- Bits 5 through 16:
  - For non-HT, specify the data length (amount of data transmitted in octets) as described in table 17-1 and section 10.27.4 IEEE Std 802.11-2020.
  - For HT-mixed, specify the transmission time as described in sections 19.3.9.3.5 and 10.27.4 of IEEE Std 802.11-2020.
  - For VHT, specify the transmission time as described in section 21.3.8.2.4 of IEEE Std 802.11-2020.
- Bit 17 has the even parity of bits 0 through 16.
- Bits 18 through 23 contain all zeros for the signal tail bits.

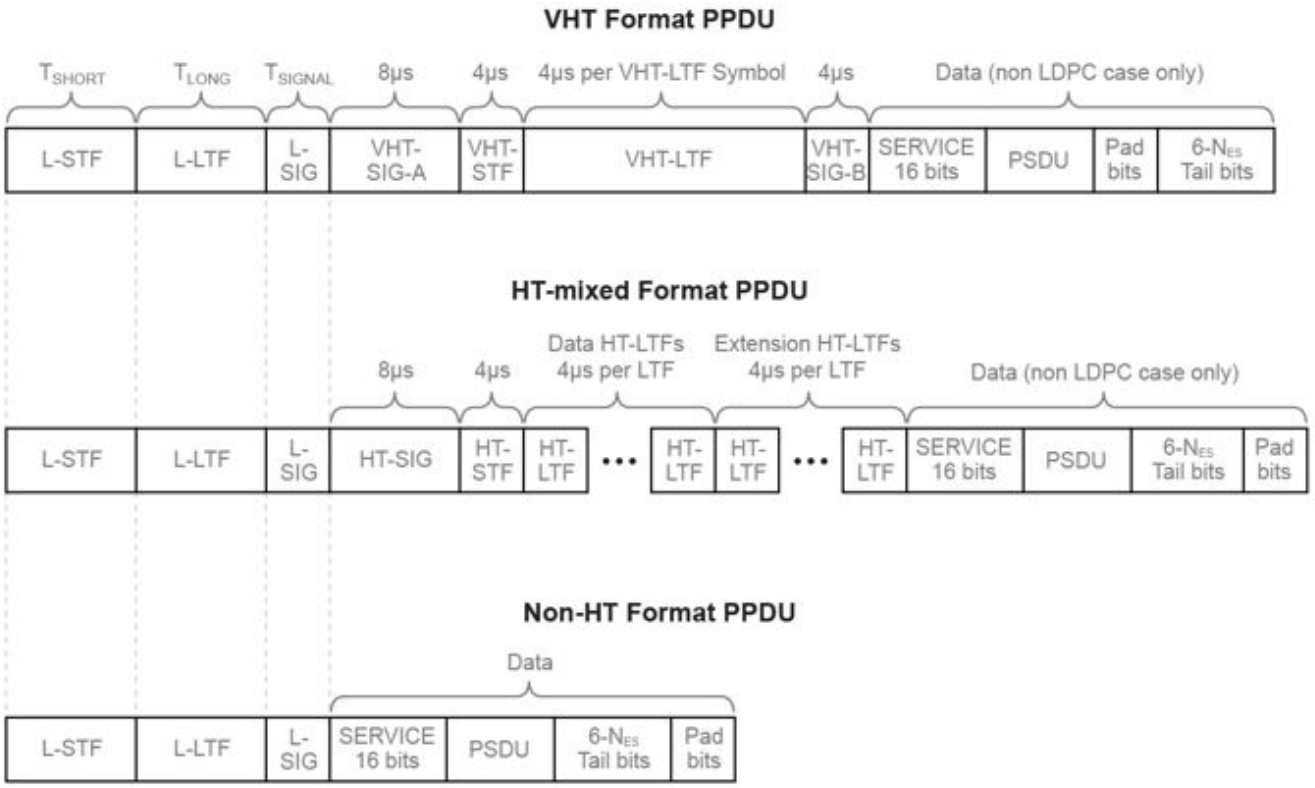
Note Signaling fields added for HT (`wlanHTSIG`) and VHT (`wlanVHTSIGA`, `wlanVHTSIGB`) formats provide data rate and configuration information for those formats.

- For the HT-mixed format, section 19.3.9.4.3 of IEEE Std 802.11-2020 describes HT-SIG bit settings.
- For the VHT format, sections 21.3.8.3.3 and 21.3.8.3.6 of IEEE Std 802.11-2020 describe bit settings for the VHT-SIG-A and VHT-SIG-B fields, respectively.

Algorithms

L-SIG Processing

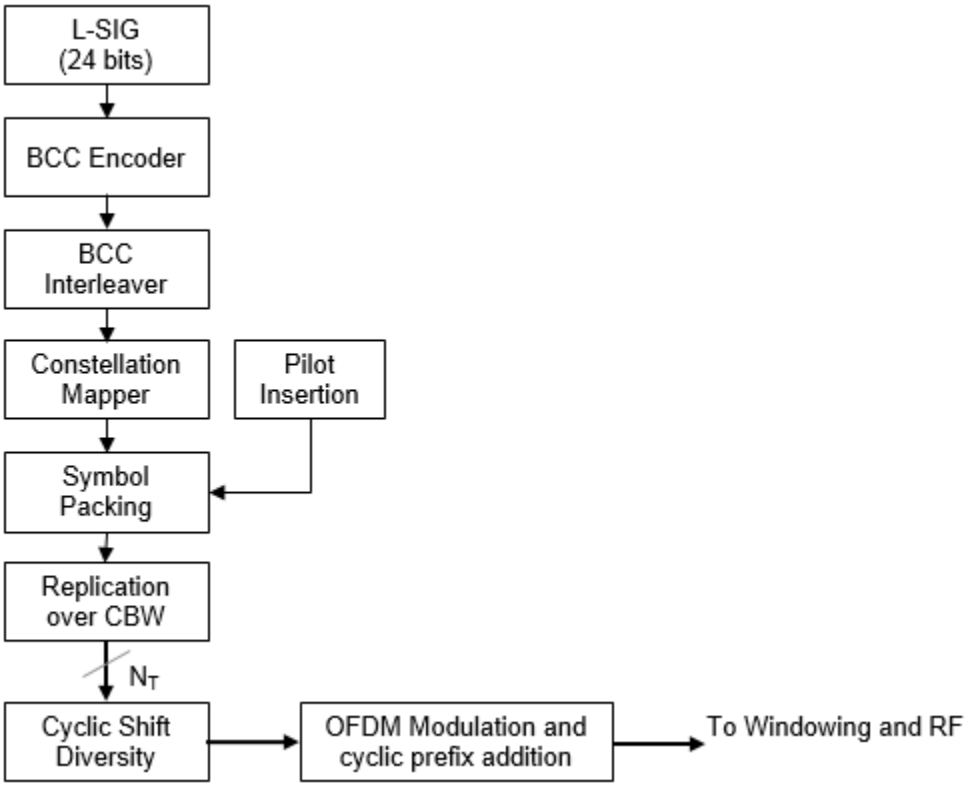
The “L-SIG” on page 3-320 follows the L-STF and L-LTF of the preamble in the packet structure.



For “L-SIG” on page 3-320 transmission processing algorithm details, see:

- VHT format – refer to IEEE Std 802.11ac-2013 [1], Section 22.3.8.2.4
- HT format – refer to IEEE Std 802.11-2012 [2], Sections 20.3.9.3.5
- non-HT format – refer to IEEE Std 802.11-2012 [2], Sections 18.3.4

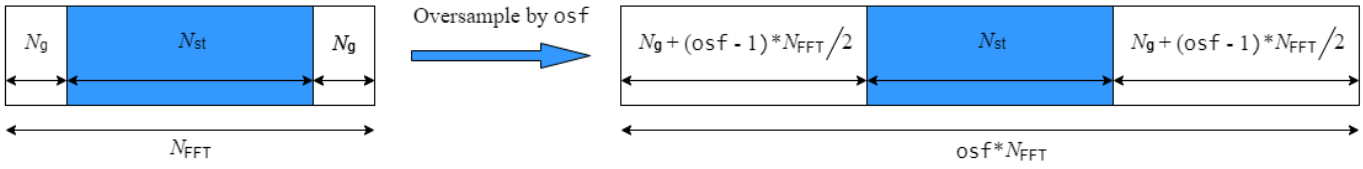
The `wlanLSIG` function performs transmitter processing on the “L-SIG” on page 3-320 field and outputs the time-domain waveform.



FFT-Based Oversampling

An oversampled signal is a signal sampled at a frequency that is higher than the Nyquist rate. WLAN signals maximize occupied bandwidth by using small guardbands, which can pose problems for anti-imaging and anti-aliasing filters. Oversampling increases guardband width relative to the total signal bandwidth, thereby increasing the number of samples in the signal.

This function performs oversampling by using a larger IFFT and zero pad when generating an OFDM waveform. This diagram shows the oversampling process for an OFDM waveform with N_{FFT} subcarriers comprising N_g guardband subcarriers on either side of N_{st} occupied bandwidth subcarriers.



Version History

Introduced in R2015b

References

[1] IEEE Std 802.11ac™-2013 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications — Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz.

[2] IEEE Std 802.11™-2016 (Revision of IEEE Std 802.11-2012). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanVHTConfig` | `wlanHTConfig` | `wlanNonHTConfig` | `wlanLLTF` | `wlanLSIGRecover`

wlanLSIGBitRecover

Recover information bits in L-SIG field

Syntax

```
[bits,failCheck,info] = wlanLSIGBitRecover(lsig,noiseVarEst)
[bits,failCheck,info] = wlanLSIGBitRecover(lsig,noiseVarEst,csi)
```

Description

`[bits,failCheck,info] = wlanLSIGBitRecover(lsig,noiseVarEst)` recovers information bits, `bits`, for legacy signal (L-SIG) field `lsig` and channel noise variance estimate `noiseVarEst`. The function also returns `failCheck`, the result of the parity check on `bits` and `info`, a structure containing modulation and coding scheme (MCS) value and physical layer convergence procedure service data unit (PSDU) length.

`[bits,failCheck,info] = wlanLSIGBitRecover(lsig,noiseVarEst,csi)` recovers the L-SIG information bits for channel state information `csi`.

Examples

Recover Information Bits in L-SIG Field

Recover the information bits in the L-SIG field of a WLAN HE single-user (HE-SU) waveform.

Create a WLAN HE-SU-format configuration object with default settings and use it to generate an HE-SU waveform.

```
cfgHE = wlanHESUConfig;
cbw = cfgHE.ChannelBandwidth;
waveform = wlanWaveformGenerator(1,cfgHE);
```

Obtain the WLAN field indices, which contain the modulated L-SIG and RL-SIG fields.

```
ind = wlanFieldIndices(cfgHE);
rxLSIG = waveform(ind.LSIG(1):ind.RLSIG(2),:);
```

Perform orthogonal frequency-division multiplexing (OFDM) demodulation to extract the L-SIG field.

```
lsigDemod = wlanHEDemodulate(rxLSIG,'L-SIG',cbw);
```

Average the L-SIG and RL-SIG symbols, return the pre-HE OFDM information, and extract the demodulated L-SIG symbols.

```
lsigDemodAverage = mean(lsigDemod,2);
preHEInfo = wlanHEOFDMInfo('L-SIG',cbw);
lsig = lsigDemodAverage(preHEInfo.DataIndices,:);
```

Recover the L-SIG information bits and other information, assuming no channel noise. Display the parity check result.

```
noiseVarEst = 0;
[bits,failCheck,info] = wlanLSIGBitRecover(lsig,noiseVarEst);
disp(failCheck);
```

```
   0
```

Recover L-SIG Information Bits with Channel State Information

Recover the information bits in the L-SIG field of a WLAN HE multiuser (HE-MU) waveform with specified channel state information.

Create a WLAN HE-MU-format configuration object with an allocation index of 192 and use it to generate an HE-MU waveform.

```
cfgHE = wlanHEMUConfig(192);
cbw = cfgHE.ChannelBandwidth;
waveform = wlanWaveformGenerator(1,cfgHE);
```

Obtain the WLAN field indices, which contain the modulated L-SIG and RL-SIG fields.

```
ind = wlanFieldIndices(cfgHE);
rxLSIG = waveform(ind.LSIG(1):ind.RLSIG(2),:);
```

Perform OFDM demodulation to extract the L-SIG field.

```
lsigDemod = wlanHEDemodulate(rxLSIG,'L-SIG',cbw);
```

Average the L-SIG and RL-SIG symbols, return the pre-HE OFDM information, and extract the demodulated L-SIG symbols.

```
lsigDemodAverage = mean(lsigDemod,2);
preHEInfo = wlanHEOFDMInfo('L-SIG',cbw);
lsig = lsigDemodAverage(preHEInfo.DataIndices,:);
```

Specify the channel state information and assume no channel noise.

```
csi = ones(52,1);
noiseVarEst = 0;
```

Recover the L-SIG information bits and other information. Display the parity check result.

```
[bits,failCheck,info] = wlanLSIGBitRecover(lsig,noiseVarEst,csi);
disp(failCheck);
```

```
   0
```

Input Arguments

`lsig` — Demodulated L-SIG symbols

complex-valued column vector

Demodulated L-SIG symbols, specified as a complex-valued column vector. The size of `lsig` depends on the WLAN format. For a high-efficiency (HE) format, specify `lsig` as a 52-by-1 column vector. For

a very-high-throughput (VHT), high-throughput (HT), or non-high-throughput (non-HT) format, specify `lsig` as a 48-by-1 column vector.

Data Types: `double`
Complex Number Support: Yes

`noiseVarEst` — Channel noise variance estimate
nonnegative scalar

Channel noise variance estimate, specified as a nonnegative scalar.

Data Types: `double`

`csi` — Channel state information
real-valued column vector

Channel state information, specified as a real-valued column vector. The size of `csi` depends on the WLAN format. For a high-efficiency (HE) format, specify `csi` as a 52-by-1 column vector. For a very-high-throughput (VHT), high-throughput (HT), or non-high-throughput (non-HT) formats, specify `csi` as a 48-by-1 column vector.

To use the channel state information for enhanced demapping of the orthogonal frequency-division multiplexing (OFDM) symbols, specify `csi`.

Data Types: `double`

Output Arguments

`bits` — Information bits recovered from L-SIG field
24-by-1 binary column vector

Information bits recovered from L-SIG field, returned as a 24-by-1 binary column.

Data Types: `int8`

`failCheck` — Parity check result
`1` (`true`) | `0` (`false`)

Parity check result, returned as a logical value of `1` (`true`) or `0` (`false`). The `wlanLSIGBitRecover` function returns `failCheck` as `1` if the recovered bits fail the parity check.

Data Types: `logical`

`info` — MCS value and PSDU length
structure

MCS value and PSDU length, returned as a structure containing these fields.

`MCS` — MCS value
integer in the interval [0, 7]

MCS value, returned as an integer in the interval [0, 7]. Each value of `MCS` corresponds to a rate, as shown in this table.

MCS	Rate (Mb/s)
0	6
1	9
2	12
3	18
4	24
5	36
6	48
7	54

For more information, see Section 17.3.4 of [1].

Data Types: `double`

`length` — Length of PSDU
nonnegative integer

Length of PSDU, returned as a nonnegative integer. The value of `length` is the number of octets in the PSDU that the physical (PHY) layer attempts to send.

Data Types: `double`

Data Types: `struct`

Version History

Introduced in R2019a

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

[2] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

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

See Also

Objects
`wlanHERecoveryConfig`

Functions
`wlanFieldIndices` | `wlanHESIGABitRecover` | `wlanHEDataBitRecover` | `wlanLSIGRecover` | `wlanLSIG`

wlanLSIGRecover

Recover L-SIG information bits

Syntax

```
recBits = wlanLSIGRecover(rxSig,chEst,noiseVarEst,cbw)
recBits = wlanLSIGRecover(rxSig,chEst,noiseVarEst,cbw,Name,Value)
[recBits,failCheck] = wlanLSIGRecover( ___ )
[recBits,failCheck,eqSym] = wlanLSIGRecover( ___ )
[recBits,failCheck,eqSym,cpe] = wlanLSIGRecover( ___ )
```

Description

`recBits = wlanLSIGRecover(rxSig,chEst,noiseVarEst,cbw)` returns the recovered "L-SIG" on page 3-336[14] information bits, `recBits`, given the time-domain L-SIG waveform, `rxSig`. Specify the channel estimate, `chEst`, the noise variance estimate, `noiseVarEst`, and the channel bandwidth, `cbw`.

`recBits = wlanLSIGRecover(rxSig,chEst,noiseVarEst,cbw,Name,Value)` returns information bits and specifies algorithm parameters by using one or more name-value pair arguments.

`[recBits,failCheck] = wlanLSIGRecover( ___ )` returns the status of a validity check, `failCheck`, using the arguments from previous syntaxes.

`[recBits,failCheck,eqSym] = wlanLSIGRecover( ___ )` returns the equalized symbols, `eqSym`.

`[recBits,failCheck,eqSym,cpe] = wlanLSIGRecover( ___ )` returns the common phase error, `cpe`.

Examples

Recover L-SIG Information from 2x2 MIMO Channel

Recover L-SIG information transmitted in a noisy 2x2 MIMO channel, and calculate the number of bit errors present in the received information bits.

Set the channel bandwidth and sample rate.

```
chanBW = 'CBW40';
fs = 40e6;
```

Create a VHT configuration object corresponding to a 40 MHz 2x2 MIMO channel.

```
vht = wlanVHTConfig( ...
    'ChannelBandwidth',chanBW, ...
```

14 IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.

```
    'NumTransmitAntennas',2, ...
    'NumSpaceTimeStreams',2);
```

Generate the L-LTF and L-SIG field signals.

```
txLLTF = wlanLLTF(vht);
[txLSIG,txLSIGData] = wlanLSIG(vht);
```

Create a 2x2 TGac channel and an AWGN channel with an SNR=10 dB.

```
tgacChan = wlanTGacChannel('SampleRate',fs,'ChannelBandwidth',chanBW, ...
    'NumTransmitAntennas',2,'NumReceiveAntennas',2);

chNoise = comm.AWGNChannel('NoiseMethod','Signal to noise ratio (SNR)',...
    'SNR',10);
```

Pass the signals through the noisy 2x2 multipath fading channel.

```
rxLLTF = chNoise(tgacChan(txLLTF));
rxLSIG = chNoise(tgacChan(txLSIG));
```

Add additional white noise corresponding to a receiver with a 9 dB noise figure. The noise variance is equal to *k*T*B*F*, where *k* is Boltzmann's constant, *T* is the ambient temperature, *B* is the channel bandwidth (sample rate), and *F* is the receiver noise figure.

```
nVar = 10^((-228.6+10*log10(290) + 10*log10(fs) + 9 )/10);
rxNoise = comm.AWGNChannel('NoiseMethod','Variance','Variance',nVar);

rxLLTF = rxNoise(rxLLTF);
rxLSIG = rxNoise(rxLSIG);
```

Perform channel estimation based on the L-LTF.

```
demodLLTF = wlanLLTFDemodulate(rxLLTF,chanBW,1);
chanEst = wlanLLTFChannelEstimate(demodLLTF,chanBW);
```

Recover the L-SIG information bits.

```
rxLSIGData = wlanLSIGRecover(rxLSIG,chanEst,0.1,chanBW);
```

Verify that there are no bit errors in the recovered L-SIG data.

```
numErrors = biterr(txLSIGData,rxLSIGData)
```

```
numErrors = 0
```

Recover L-SIG Field with Zero-Forcing Equalizer

Recover L-SIG information using the zero-forcing equalizer algorithm. Calculate the number of bit errors in the received data.

Create an HT configuration object.

```
cfgHT = wlanHTConfig;
```

Generate the L-SIG field and pass it through an AWGN channel.

```
[txLSIG,txLSIGData] = wlanLSIG(cfgHT);
rxSIG = awgn(txLSIG,20);
```

Recover the L-SIG field using the zero-forcing algorithm. The channel estimate is a vector of ones because fading was not introduced.

```
chEst = ones(52,1);
noiseVarEst = 0.1;
rxLSIGData = wlanLSIGRecover(rxSIG,chEst,noiseVarEst,'CBW20','EqualizationMethod','ZF');
```

Verify that there are no bit errors in the recovered L-SIG data.

```
numErrors = biterr(txLSIGData,rxLSIGData)
```

```
numErrors = 0
```

Recover L-SIG Field from Phase and Frequency Offset

Recover the L-SIG field rom a channel that introduces a fixed phase and frequency offset.

Create a VHT configuration object corresponding to a 160 MHz SISO channel. Generate the transmitted L-SIG field.

```
cfgVHT = wlanVHTConfig('ChannelBandwidth','CBW160');
txLSIG = wlanLSIG(cfgVHT);
```

To introduce a 45 degree phase offset and a 100 Hz frequency offset, create a phase and frequency offset System object.

```
pfOffset = comm.PhaseFrequencyOffset('SampleRate',160e6,'PhaseOffset',45, ...
    'FrequencyOffset',100);
```

Introduce phase and frequency offsets to the transmitted L-SIG field, then pass it through an AWGN channel.

```
rxSIG = awgn(pfOffset(txLSIG),20);
```

Recover the L-SIG information bits, the failure check status, and the equalized symbols, disabling pilot phase tracking.

```
chEst = ones(416,1);
noiseVarEst = 0.01;
[recBits,failCheck,eqSym] = wlanLSIGRecover(rxSIG,chEst,noiseVarEst,'CBW160','PilotPhaseTracking
```

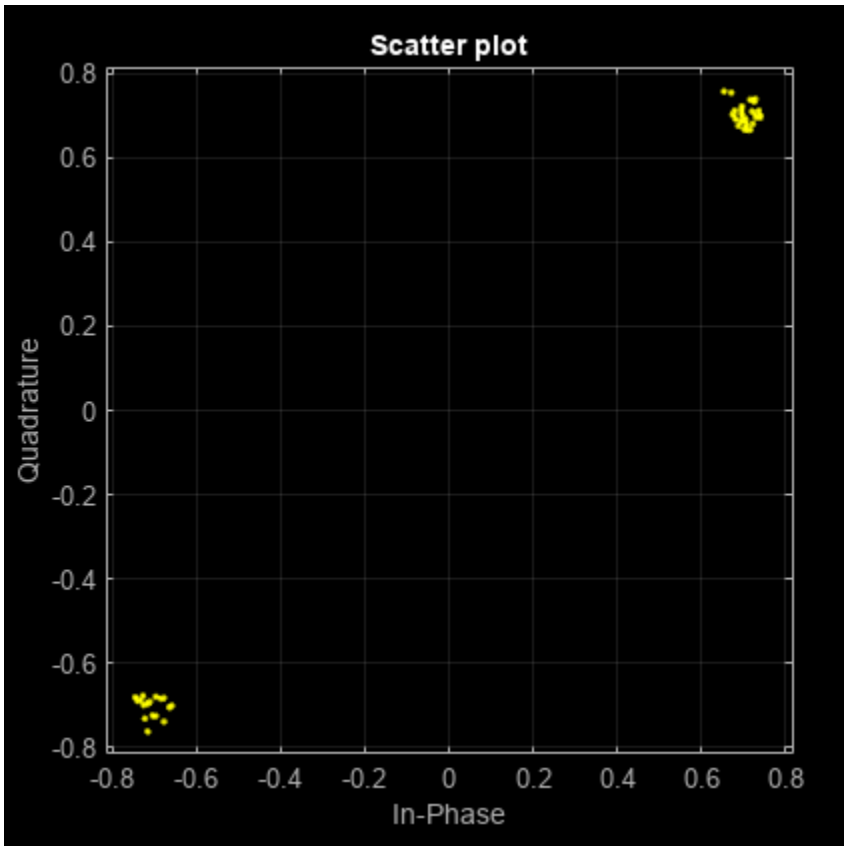
Verify that the L-SIG passed the failure checks.

```
disp(failCheck)
```

```
   0
```

Visualize the phase offset by plotting the equalized symbols.

```
scatterplot(eqSym)
grid
```



Input Arguments

`rxSig` — Received L-SIG field

vector | matrix

Received L-SIG field, specified as an N_S-by-N_R matrix. N_S is the number of samples, and N_R is the number of receive antennas.

N_S is proportional to the channel bandwidth.

`ChannelBandwidth`	N_S
`'CBW5'`, `'CBW10'`, `'CBW20'`	80
`'CBW40'`	160
`'CBW80'`	320
`'CBW160'`	640

Data Types: `double`

`chEst` — Channel estimate

vector | 3-D array

Channel estimate, specified as an N_{ST}-by-1-by-N_R array. N_{ST} is the number of occupied subcarriers, and N_R is the number of receive antennas.

Channel Bandwidth	N_{ST}
'CBW5', 'CBW10', 'CBW20'	52
'CBW40'	104
'CBW80'	208
'CBW160'	416

Data Types: double

noiseVarEst — Noise variance estimate
nonnegative scalar

Noise variance estimate, specified as a nonnegative scalar.

Data Types: double

cbw — Channel bandwidth
'CBW5' | 'CBW10' | 'CBW20' | 'CBW40' | 'CBW80' | 'CBW160'

Channel bandwidth in MHz, specified as 'CBW5', 'CBW10', 'CBW20', 'CBW40', 'CBW80', or 'CBW160'.

Example: 'CBW80' corresponds to a channel bandwidth of 80 MHz

Data Types: char | string

Name-Value Pair Arguments

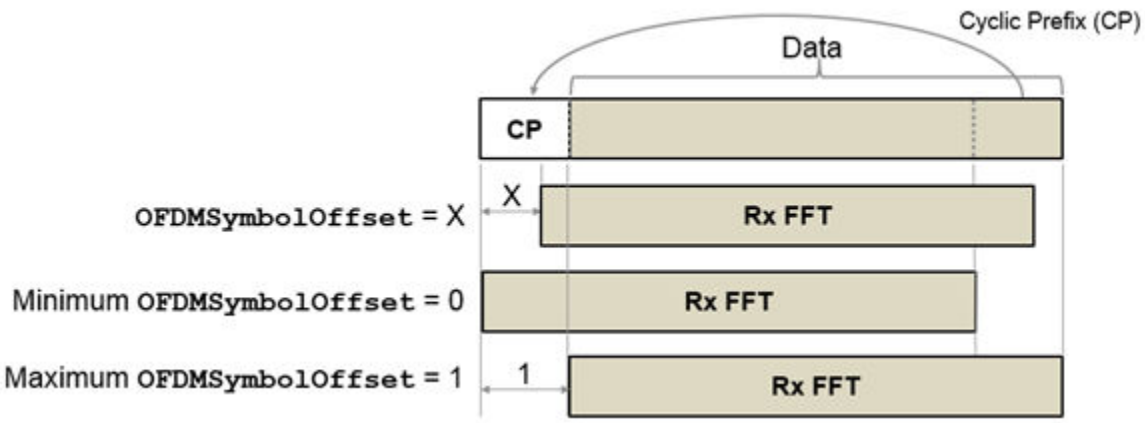
Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

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

Example: 'PilotPhaseTracking','None' disables pilot phase tracking.

OFDMSymbolOffset — OFDM symbol sampling offset
0.75 (default) | scalar in the interval [0, 1]

OFDM symbol sampling offset represented as a fraction of the cyclic prefix (CP) length, specified as the comma-separated pair consisting of 'OFDMSymbolOffset' and a scalar in the interval [0, 1]. The value you specify indicates the start location for OFDM demodulation relative to the beginning of the CP. The value 0 represents the start of the CP, and the value 1 represents the end of the CP.



Data Types: `double`

`EqualizationMethod` — Equalization method
`'MMSE'` (default) | `'ZF'`

Equalization method, specified as one of these values.

- `'MMSE'` — The receiver uses a minimum mean-square error equalizer.
- `'ZF'` — The receiver uses a zero-forcing equalizer.

When the received signal has multiple receive antennas, the function exploits receiver diversity during equalization. When the number of transmitted space-time streams is one and you specify this argument as `'ZF'`, the function performs maximal-ratio combining.

Data Types: `char` | `string`

`PilotPhaseTracking` — Pilot phase tracking
`'PreEQ'` (default) | `'None'`

Pilot phase tracking, specified as the comma-separated pair consisting of `'PilotPhaseTracking'` and one of these values.

- `'PreEQ'` — Enable pilot phase tracking, which the function performs before any equalization operation.
- `'None'` — Disable pilot phase tracking.

Data Types: `char` | `string`

Output Arguments

`recBits` — Recovered L-SIG information
binary vector

Recovered L-SIG information bits, returned as a 24-element column vector containing binary data. The 24 elements correspond to the length of the L-SIG field.

Data Types: `int8`

`failCheck` — Failure check status
`true` | `false`

Failure check status, returned as a logical scalar. If L-SIG fails the parity check, or if its first four bits do not correspond to one of the eight allowable data rates, `failCheck` is `true`.

Data Types: `logical`

`eqSym` — Equalized symbols
vector

Equalized symbols, returned as 48-by-1 vector. There are 48 data subcarriers in the L-SIG field.

Data Types: `double`

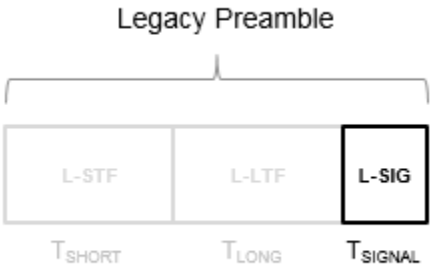
`cpe` — Common phase error
column vector

Common phase error in radians, returned as a scalar.

More About

L-SIG

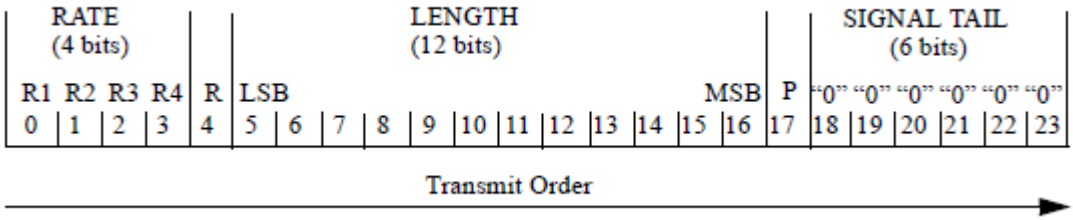
The L-SIG is the third field of the 802.11 OFDM PLCP legacy preamble. This field is a component of EHT, HE, VHT, HT, and non-HT PPDUs. It consists of 24 bits that contain rate, length, and parity information. The L-SIG field is transmitted using BPSK modulation with rate 1/2 binary convolutional coding (BCC).



The L-SIG is one OFDM symbol with a duration that varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period ($T_{FFT} = 1 / \Delta_F$)	Guard Interval (GI) Duration ($T_{GI} = T_{FFT} / 4$)	L-SIG Duration ($T_{SIGNAL} = T_{GI} + T_{FFT}$)
20, 40, 80, and 160	312.5	3.2 µs	0.8 µs	4 µs
10	156.25	6.4 µs	1.6 µs	8 µs
5	78.125	12.8 µs	3.2 µs	16 µs

The L-SIG contains packet information for the received configuration.



- Bits 0 through 3 specify the data rate (modulation and coding rate) for the non-HT format.

Rate (Bits 0–3)	Modulation	Coding Rate (*R*)	Data Rate (Mb/s)		
			20 MHz Channel Bandwidth	10 MHz Channel Bandwidth	5 MHz Channel Bandwidth
1101	BPSK	1/2	6	3	1.5
1111	BPSK	3/4	9	4.5	2.25
0101	QPSK	1/2	12	6	3
0111	QPSK	3/4	18	9	4.5
1001	16-QAM	1/2	24	12	6
1011	16-QAM	3/4	36	18	9
0001	64-QAM	2/3	48	24	12
0011	64-QAM	3/4	54	27	13.5

  For HT and VHT formats, the L-SIG rate bits are set to `'1 1 0 1'`. Data rate information for HT and VHT formats is signaled in format-specific signaling fields.
- Bit 4 is reserved for future use.
- Bits 5 through 16:
  - For non-HT, specify the data length (amount of data transmitted in octets) as described in table 17-1 and section 10.27.4 IEEE Std 802.11-2020.
  - For HT-mixed, specify the transmission time as described in sections 19.3.9.3.5 and 10.27.4 of IEEE Std 802.11-2020.
  - For VHT, specify the transmission time as described in section 21.3.8.2.4 of IEEE Std 802.11-2020.
- Bit 17 has the even parity of bits 0 through 16.
- Bits 18 through 23 contain all zeros for the signal tail bits.

Note Signaling fields added for HT (`wlanHTSIG`) and VHT (`wlanVHTSIGA`, `wlanVHTSIGB`) formats provide data rate and configuration information for those formats.

- For the HT-mixed format, section 19.3.9.4.3 of IEEE Std 802.11-2020 describes HT-SIG bit settings.
- For the VHT format, sections 21.3.8.3.3 and 21.3.8.3.6 of IEEE Std 802.11-2020 describe bit settings for the VHT-SIG-A and VHT-SIG-B fields, respectively.

Version History

Introduced in R2015b

References

[1] IEEE Std 802.11™-2016 (Revision of IEEE Std 802.11-2012). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanLSIG` | `wlanLLTF` | `wlanLLTFDemodulate` | `wlanLLTFChannelEstimate`

wlanLSTF

Generate L-STF waveform

Syntax

```
y = wlanLSTF(cfg)
y = wlanLSTF(cfg,OversamplingFactor=osf)
```

Description

`y = wlanLSTF(cfg)` generates an "L-STF" on page 3-341[15] time-domain waveform using the specified transmission parameters.

`y = wlanLSTF(cfg,OversamplingFactor=osf)` generates an L-STF waveform for the specified oversampling factor. For more information about oversampling, see "FFT-Based Oversampling" on page 3-342.

Examples

Generate L-STF Waveform

Generate the L-STF waveform for a 40 MHz single antenna VHT packet.

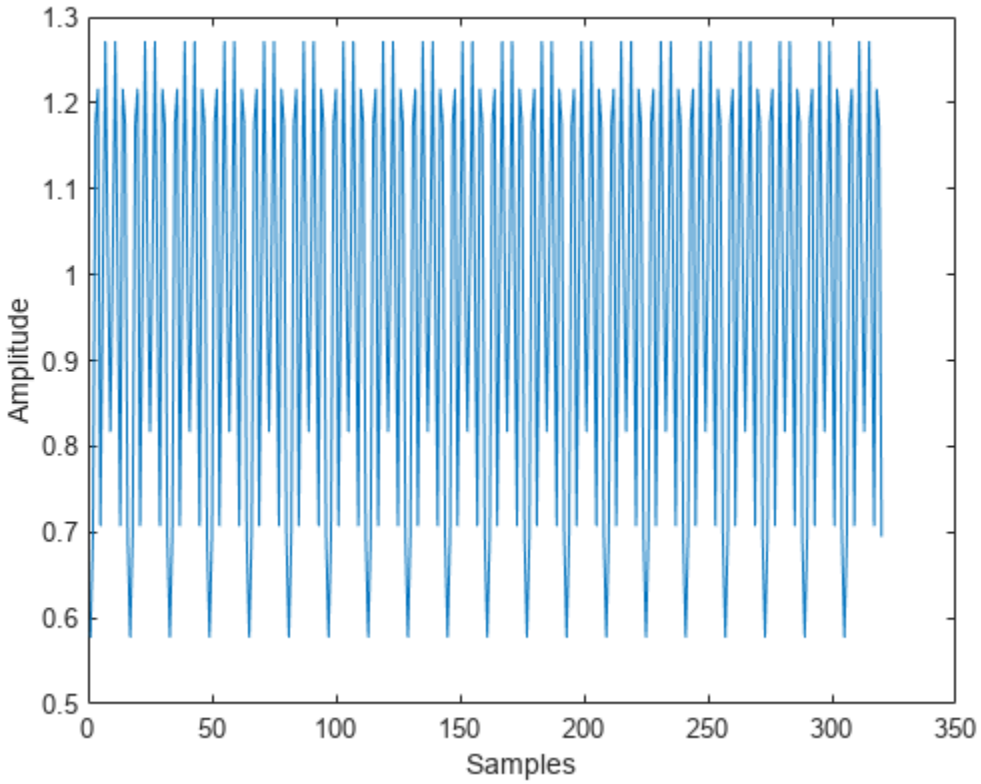
Create a VHT configuration object. Use this object to generate the L-STF waveform.

```
cfgVHT = wlanVHTConfig('ChannelBandwidth','CBW40');
y = wlanLSTF(cfgVHT);
size(y)
```

```
ans = 1×2

   320     1
```

```
plot(abs(y))
xlabel('Samples')
ylabel('Amplitude')
```

15 IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.



The output L-STF waveform contains 320 samples for a 40 MHz channel bandwidth.

Input Arguments

`cfg` — Transmission parameters

`wlanVHTConfig` object | `wlanHTConfig` object | `wlanNonHTConfig` object

Transmission parameters, specified as a `wlanVHTConfig`, `wlanHTConfig`, or `wlanNonHTConfig` object.

Example: `wlanVHTConfig`

`osf` — Oversampling factor

1 (default) | scalar greater than or equal to 1

Oversampling factor, specified as a scalar greater than or equal to 1. The oversampled cyclic prefix length must be an integer number of samples. The resultant inverse fast Fourier transform (IFFT) length must be even.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`y` — L-STF time-domain waveform

matrix

("L-STF" on page 3-341) time-domain waveform, returned as an N_S-by-N_T matrix. N_S is the number of time-domain samples, and N_T is the number of transmit antennas.

N_S is proportional to the channel bandwidth. The time-domain waveform consists of two symbols.

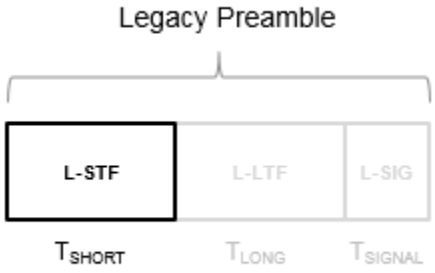
ChannelBandwidth	N_S
'CBW5', 'CBW10', 'CBW20'	160
'CBW40'	320
'CBW80'	640
'CBW160'	1280
'CBW320'	2560

Data Types: double
Complex Number Support: Yes

More About

L-STF

The legacy short training field (L-STF) is the first field of the 802.11 OFDM PLCP legacy preamble. The L-STF is a component of VHT, HT, and non-HT PPDUs.



The L-STF duration varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period ($T_{FFT} = 1 / \Delta_F$)	L-STF Duration ($T_{SHORT} = 10 \times T_{FFT} / 4$)
20, 40, 80, 160, and 320	312.5	3.2 µs	8 µs
10	156.25	6.4 µs	16 µs
5	78.125	12.8 µs	32 µs

Because the sequence has good correlation properties, it is used for start-of-packet detection, for coarse frequency correction, and for setting the AGC. The sequence uses 12 of the 52 subcarriers that are available per 20 MHz channel bandwidth segment. For 5 MHz, 10 MHz, and 20 MHz bandwidths, the number of channel bandwidths segments is 1.

Algorithms

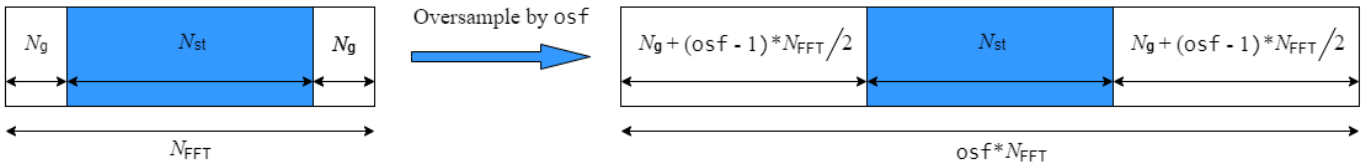
L-STF Processing

The “L-STF” on page 3-341 is two OFDM symbols long and is the first field in the packet structure for the VHT, HT, and non-HT OFDM formats. For algorithm details, see IEEE Std 802.11ac-2013 [1], Section 22.3.8.2.2.

FFT-Based Oversampling

An oversampled signal is a signal sampled at a frequency that is higher than the Nyquist rate. WLAN signals maximize occupied bandwidth by using small guardbands, which can pose problems for anti-imaging and anti-aliasing filters. Oversampling increases guardband width relative to the total signal bandwidth, thereby increasing the number of samples in the signal.

This function performs oversampling by using a larger IFFT and zero pad when generating an OFDM waveform. This diagram shows the oversampling process for an OFDM waveform with N_{FFT} subcarriers comprising N_g guardband subcarriers on either side of N_{st} occupied bandwidth subcarriers.



Version History

Introduced in R2015b

References

[1] IEEE Std 802.11ac™-2013 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications — Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanVHTConfig` | `wlanHTConfig` | `wlanNonHTConfig` | `wlanLLTF`

wlanMACFrame

Generate WLAN MAC frame (MPDU or A-MPDU)

Syntax

```
[frame,frameLength] = wlanMACFrame(cfgMAC)
[frame,frameLength] = wlanMACFrame(payload,cfgMAC)
[frame,frameLength] = wlanMACFrame(payload,cfgMAC,cfgPHY)
[frame,frameLength] = wlanMACFrame(cfgMAC,cfgPHY)
[frame,frameLength] = wlanMACFrame( ___ ,OutputFormat=format)
```

Description

`[frame,frameLength] = wlanMACFrame(cfgMAC)` generates `frame`, a WLAN medium access control (MAC) frame for the specified MAC frame configuration object. The function generates a MAC protocol data unit (MPDU) or an aggregate MPDU (A-MPDU), depending on the property values of the `cfgMAC` input. The function also returns `frameLength`, the total length of the generated MAC frame.

`[frame,frameLength] = wlanMACFrame(payload,cfgMAC)` generates a WLAN MAC frame containing MAC service data units (MSDUs) `payload`. For the MAC frame to contain the MSDUs, you must set the `FrameType` property of the `cfgMAC` input to `'Data'` or `'QoS Data'`. Otherwise, the function ignores `payload`.

`[frame,frameLength] = wlanMACFrame(payload,cfgMAC,cfgPHY)` generates a WLAN MAC frame for the physical layer (PHY) format and transmission parameters specified in `cfgPHY`. To generate A-MPDUs in high-efficiency (HE), very high throughput (VHT), or high-throughput (HT) format, use this syntax.

`[frame,frameLength] = wlanMACFrame(cfgMAC,cfgPHY)` generates a WLAN MAC trigger frame for the specified MAC and PHY configurations. To create a MAC trigger frame, the `cfgPHY` input must be a `wlanNonHTConfig` object, and you must set the `FrameType` property of the `cfgMAC` input to `'Trigger'`.

`[frame,frameLength] = wlanMACFrame( ___ ,OutputFormat=format)` specifies the data format of the generated frame in addition to any input argument combination from previous syntaxes.

Examples

Generate RTS MAC Frame

Create a `wlanMACFrameConfig` object for a request-to-send (RTS) MAC frame.

```
cfgMAC = wlanMACFrameConfig(FrameType="RTS");
```

Generate the frame by calling the `wlanMACFrame` function and display the result.

```
[frame,frameLength] = wlanMACFrame(cfgMAC);
disp(frame')
```

```
B000FFFFFF013579A952
4000FFFFFF02468B7AB8
```

Generate QoS Data Frame with Specified Payload

Generate a quality of service (QoS) Data MAC frame with a specified payload.

```
macConfig = wlanMACFrameConfig(FrameType="QoS Data");
payload = "00576000103afffe80";
[frame,frameLength] = wlanMACFrame(payload,macConfig);
disp(frame')
```

```
8000FFFFFF013579013579002005 6013FF88241
8200FFFFFF02468B02468B000007000AFE0EA33
```

Generate HT A-MPDU

Create a MAC frame configuration object.

```
cfgMAC = wlanMACFrameConfig(FrameType="QoS Data", ...
    FrameFormat="HT-Mixed", ...
    MPDUAggregation=true);
```

Create an HT PHY configuration object.

```
cfgPHY = wlanHTConfig(MCS=4);
```

Calculate the MSDU lengths required to generate a 5000-octet A-MPDU frame, displaying the result.

```
msduLengths = wlanMSDULengths(5000,cfgMAC,cfgPHY);
disp(msduLengths)
```

```
        2302        2302         294
```

Create MSDUs with random data using the MSDU length vector.

```
numMSDUs = numel(msduLengths);
payload = cell(1,numMSDUs);
 for i = 1:numMSDUs
     payload{i} = randi([0 255],1,msduLengths(i));
 end
```

Generate the 5000-octet A-MPDU.

```
[frame,frameLength] = wlanMACFrame(payload,cfgMAC,cfgPHY);
disp(frameLength)
```

```
        5000
```

Generate Beacon MAC Frame with SSID

Create a `wlanMACManagementConfig` configuration object, specifying the service set identifier (SSID).

```
cfgMgmt = wlanMACManagementConfig(SSID="demo SSID");
```

Create a `wlanMACFrameConfig` configuration object, specifying the management frame-body configuration object as `mgmtConfig` and a beacon MAC frame.

```
cfgMAC = wlanMACFrameConfig(FrameType="Beacon", ...
    ManagementConfig=cfgMgmt);
```

Generate the beacon MAC frame with the specified SSID.

```
[macFrame,frameLength] = wlanMACFrame(cfgMAC);
```

Display the frame length.

```
frameLength
```

```
frameLength = 56
```

Create Basic MAC Trigger Frame

Create a basic MAC trigger frame to carry information for two users.

Create a MAC trigger frame-body configuration object, specifying a channel bandwidth of 40 MHz.

```
cfgTrigger = wlanMACTriggerConfig(ChannelBandwidth="CBW40");
```

Create configuration objects for the User Info fields of the trigger frame.

```
cfgUser1 = wlanMACTriggerUserConfig(AID12=1, ...
    RUSize=242,RUIndex=1);
cfgUser2 = wlanMACTriggerUserConfig(AID12=2, ...
    RUSize=242,RUIndex=2);
```

Add the User Info fields to the trigger frame.

```
cfgTrigger = addUserInfo(cfgTrigger,cfgUser1);
cfgTrigger = addUserInfo(cfgTrigger,cfgUser2);
```

Configure the trigger frame by creating a MAC frame-body configuration object, specifying the frame type and the trigger frame-body configuration.

```
cfgMAC = wlanMACFrameConfig(FrameType="Trigger", ...
    TriggerConfig=cfgTrigger);
```

Specify a non-HT PHY configuration by creating a default non-HT configuration object.

```
cfgPHY = wlanNonHTConfig;
```

Create the MAC trigger frame and display its length.

```
[frame,frameLength] = wlanMACFrame(cfgMAC,cfgPHY);
disp(frameLength)
```

```
40
```

Generate RTS MAC Frame in Bit Format

Create a `wlanMACFrameConfig` object for an RTS MAC frame.

```
cfgMAC = wlanMACFrameConfig(FrameType="RTS");
```

Generate the RTS MAC frame in bit format.

```
[frame,frameLength] = wlanMACFrame(cfgMAC,OutputFormat="bits");
```

Input Arguments

`cfgMAC` — MAC frame configuration

`wlanMACFrameConfig` object

MAC frame configuration, specified as a `wlanMACFrameConfig` object. This object defines the type of MAC frame and its applicable properties.

`payload` — One or more MSDUs

numeric vector | character vector | string | cell array

One or more MSDUs, specified as a numeric vector, character vector, string, or cell array. The value you specify depends on whether the frame is aggregated.

- To generate an MPDU, specify this argument as one of these values:
  - A numeric vector of octets in decimal format, where each element is an integer in the interval [0, 255]
  - A character vector of octets in hexadecimal format
  - A string scalar of octets in hexadecimal format

  The value you specify represents one MSDU.
- To generate an A-MPDU, specify this argument as one of these values:
  - A cell array of numeric vectors
  - A cell array of character vectors
  - A string array

  Each element of the specified array represents one MSDU.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `char` | `string` | `cell`

`cfgPHY` — PHY format configuration

`wlanHTConfig` object (default) | `wlanVHTConfig` object | `wlanHESUConfig` object

PHY format configuration, specified as a configuration object of type `wlanHESUConfig`, `wlanVHTConfig`, or `wlanHTConfig`. The value you specify must be compatible with the frame format specified in the `cfgMAC` input.

- If the `FrameFormat` property of `cfgMAC` is `"HE-SU"` or `"HE-EXT-SU"`, specify this argument as a `wlanHESUConfig` object.
- If the `FrameFormat` property of `cfgMAC` is `"VHT"`, specify this argument as a `wlanVHTConfig` object.
- If the `FrameFormat` property of `cfgMAC` is `"HT-Mixed"`, specify this argument as a `wlanHTConfig` object.

Specify this argument to:

- Ensure that the frame does not exceed the transmission time limit.
- Add end-of-frame (EOF) padding to frames in very-high-throughput (VHT) or high-efficiency (HE) format.
- Maintain minimum start spacing between MPDUs in an A-MPDU.

`format` — MAC frame format
`"octets"` (default) | `"bits"`

MAC frame format, specified as `"octets"` or `"bits"`.

Data Types: `char` | `string`

Output Arguments

`frame` — MAC frame
character array | binary-valued column vector

MAC frame (MPDU or A-MPDU), returned as one of these values.

- A character array, where each row is an octet in hexadecimal format, when you specify the `format` input as `"octets"`
- A binary-valued column vector when you specify the `format` input as `"bits"`

Data Types: `int8` | `char`

`frameLength` — Length of generated MAC frame
nonnegative integer

Length of generated MAC frame, in octets, returned as a nonnegative integer. For VHT- and HE-format A-MPDUs, this output is the A-MPDU pre-EOF padding (APEP) length, which is less than or equal to the length of the `frame` output. For all other formats, this output is the physical layer convergence procedure (PLCP) service data unit (PSDU) length.

Data Types: `double`

Version History
Introduced in R2018b

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for

Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

[2] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`wlanHTConfig` | `wlanVHTConfig` | `wlanHESUConfig` | `wlanMACFrameConfig` | `wlanMACManagementConfig`

Functions

`wlanMSDULengths` | `wlanWaveformGenerator` | `wlanAMPDUDeaggregate` | `wlanMPDUDecode`

Topics

“Generate and Parse WLAN MAC Frames”

wlanMPDUDecode

Decode MPDU

Syntax

```
[cfgMAC,payload,status] = wlanMPDUDecode(mpdu,phyFormat)
[cfgMAC,payload,status] = wlanMPDUDecode(mpdu,cfgPHY)
[cfgMAC,payload,status] = wlanMPDUDecode( ___ ,Name=Value)
```

Description

`[cfgMAC,payload,status] = wlanMPDUDecode(mpdu,phyFormat)` recovers `payload`, one or more MAC service data units (MSDUs), by decoding MAC protocol data unit (MPDU) `mpdu`. The function decodes the MPDU by using parameters appropriate for the specified PHY format.

The function also returns `status`, the result of the MPDU decoding, and `cfgMAC`, a `wlanMACFrameConfig` configuration object.

`[cfgMAC,payload,status] = wlanMPDUDecode(mpdu,cfgPHY)` decodes the MPDU by using PHY transmission parameters `cfgPHY`.

`[cfgMAC,payload,status] = wlanMPDUDecode( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition any input argument combination from previous syntaxes.

Examples

Decode HE SU MPDU

Create a WLAN MAC frame configuration object for an MPDU in high-efficiency single-user (HE SU) format, and then generate the MPDU.

```
phyFormat = 'HE-SU';
cfgMAC =  wlanMACFrameConfig('FrameFormat',phyFormat);
payload = randi([0 255],1,40);
mpdu = wlanMACFrame(payload,cfgMAC,'OutputFormat','bits');
```

Return the MSDUs by decoding the MPDU for the specified PHY format configuration.

```
[rxCfgMAC,payload,status] = wlanMPDUDecode(mpdu,phyFormat);
```

Confirm successful decoding by displaying the status.

```
disp(status)
```

```
    Success
```

Decode Non-HT MPDU

Create a WLAN MAC frame configuration object for a QoS Data frame, and then generate the MPDU.

```
cfgMAC = wlanMACFrameConfig('FrameType','QoS Data');
payload = randi([0 255],1,40);
mpdu = wlanMACFrame(payload,cfgMAC,'OutputFormat','bits');
```

Create a non-high-throughput-format (non-HT-format) configuration object with default settings.

```
cfgPHY = wlanNonHTConfig;
```

Return the MSDUs by decoding the MPDU for the specified PHY format configuration.

```
[cfgMAC,payload,status] = wlanMPDUDecode(mpdu,cfgPHY);
```

Confirm successful decoding by displaying the status.

```
disp(status)
```

```
    Success
```

Decode MPDUs Extracted from A-MPDU

Deaggregate a VHT A-MPDU and decode the extracted MPDUs.

Create a WLAN MAC frame configuration object for a VHT A-MPDU.

```
txCfgMAC = wlanMACFrameConfig('FrameType','QoS Data', ...
    'FrameFormat','VHT');
```

Create a VHT-format configuration object with default settings.

```
cfgPHY = wlanVHTConfig;
```

Generate a random payload of eight MSDUs.

```
txPayload =  repmat({randi([0 255],1,40)},1,8);
```

Generate the A-MPDU containing eight MPDUs for the specified MAC and PHY configurations.

```
ampdu = wlanMACFrame(txPayload,txCfgMAC,cfgPHY);
```

Extract the list of MPDUs by deaggregating the A-MPDU. Display the status of the deaggregation and the delimiter CRC.

```
[mpduList,failCRC,status] = wlanAMPDUDeaggregate(ampdu,cfgPHY, ...
    'DataFormat','octets');
disp(status)
```

```
    Success
```

```
disp(failCRC)
```

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

Decode all of the MPDUs in the extracted. Confirm successful decoding by displaying the status.

```
if strcmp(status,'Success')
    for i = 1:numel(mpduList)
        if ~failCRC(i)
            [cfgMAC,payload,status(i)] = ...
                wlanMPDUDecode(mpduList{i},cfgPHY, ...
                'DataFormat','octets');
        end
    end
end
disp(status)
```

```
  Columns 1 through 7

    Success    Success    Success    Success    Success    Success    Success

  Column 8

    Success
```

Input Arguments

`mpdu` — MPDU to be decoded

binary-valued vector | vector of integers in the interval [0, 255] | string scalar | character array

MPDU to be decoded, specified as one of these values.

- A binary-valued vector representing the MPDU in bit form
- A vector of integers in the interval [0, 255] representing octets in decimal format
- A string scalar representing the MPDU as octets in hexadecimal format
- A character vector representing the MPDU as octets in hexadecimal format
- A character array, where each row represents an octet in hexadecimal format

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `char` | `string`

`phyFormat` — PHY format

`'HE-SU'` | `'HE-EXT-SU'` | `'HE-MU'` | `'HE-TB'` | `'VHT'` | `'HT'` | `'Non-HT'`

PHY format, specified as one of these values.

- `'HE-SU'` — High-efficiency single-user (HE SU) format
- `'HE-EXT-SU'` — HE extended-range SU (HE ER SU) format
- `'HE-MU'` — HE multi-user (HE MU) format
- `'HE-TB'` — HE trigger-based (HE TB) format
- `'VHT'` — Very-high-throughput (VHT) format
- `'HT'` — High-throughput (HT) format
- `'Non-HT'` — Non-HT format

Data Types: `char` | `string`

`cfgPHY` — PHY format and transmission parameters
`wlanHESUConfig` object | `wlanHEMUConfig` object | `wlanHETBConfig` object | `wlanHERecoveryConfig` object | `wlanVHTConfig` object | `wlanHTConfig` object | `wlanNonHTConfig` object

PHY format and transmission parameters, specified as one of these objects.

- `wlanHESUConfig` — HE SU or HE ER SU format
- `wlanHEMUConfig` — HE MU format
- `wlanHETBConfig` — HE TB format
- `wlanHERecoveryConfig` — Recovered HE transmission in HE SU, HE ER SU, or HE MU format
- `wlanVHTConfig` — VHT format
- `wlanHTConfig` — HT format
- `wlanNonHTConfig` — Non-high-throughput (non-HT) format

Name-Value Arguments

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

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

Example: `DataFormat='octets'`

`DataFormat` — Format of input MPDU
`'bits'` (default) | `'octets'`

Format of input MPDU, specified as one of these values.

- `'bits'` — Specify the `mpdu` input in bit format
- `'octets'` — Specify the `mpdu` input in octet format

Data Types: `char` | `string`

`SuppressWarnings` — Suppress warning messages
`false` or `0` (default) | `true` or `1`

Suppress warning messages, specified as one of these values.

- `false` or `0` — Allow warning messages.
- `true` or `1` — Suppress warning messages.

Data Types: `logical`

`IsMeshFrame` — Mesh frame indication
`false` or `0` (default) | `true` or `1`

Mesh frame indication, specified as a numeric or logical `1` (`true`) or `0` (`false`). To indicate that the frame originates from a mesh station in a mesh BSS, set this property to `1` (`true`). The function uses this argument only when decoding a frame of type QoS Data or QoS Null.

Data Types: `logical`

Output Arguments

`cfgMAC` — MAC frame configuration

`wlanMACFrameConfig` object

MAC frame configuration, returned as a `wlanMACFrameConfig` object.

`payload` — One or more MSDUs

cell array of character vectors

One or more MSDUs, returned as a cell array of character arrays. The function returns a character array for each MSDU. In these character arrays, each row is the hexadecimal representation of an octet. For each MAC frame that contains no data, the function returns `payload` as an empty cell array.

Data Types: `cell`

`status` — Status of MPDU decoding

integer in the interval [–31, 0]

Status of MPDU decoding, returned as an integer in the interval [–31, 0]. Each value of `status` corresponds to a member of the `wlanMACDecodeStatus` enumeration class, which indicates the status of MAC frame decoding according to this table.

Enumeration Value	Member of Enumeration Class	Decoding Status
`0`	`Success`	MAC frame successfully decoded
`-1`	`FCSFailed`	Frame check sequence (FCS) failed
`-2`	`InvalidProtocolVersion`	Invalid protocol version
`-3`	`UnsupportedFrameType`	Unsupported frame type
`-4`	`UnsupportedFrameSubtype`	Unsupported frame subtype
`-5`	`NotEnoughData`	Insufficient data to decode frame
`-6`	`UnsupportedBAVariant`	Unsupported variant of Block Ack frame
`-7`	`UnknownBitmapSize`	Unknown bitmap size
`-8`	`UnknownAddressExtMode`	Unknown address extension mode
`-9`	`MalformedAMSDULength`	Malformed aggregate MAC service data unit (A-MSDU) with invalid length
`-10`	`MalformedSSID`	Malformed service set identifier (SSID) information element (IE)
`-11`	`MalformedSupportedRatesIE`	Malformed supported rates IE
`-12`	`MalformedIELength`	Malformed IE length field

-13	MissingMandatoryIEs	Mandatory IEs missing
-14	NoMPDUFound	No MPDU found in A-MPDU
-15	CorruptedAMPDU	All the delimiters in received A-MPDU failed cyclic redundancy check (CRC)
-16	InvalidDelimiterLength	Invalid length field in MPDU delimiter
-17	MaxAMSDULenthExceeded	A-MSDU exceeds maximum length limit
-18	MaxMPDULengthExceeded	MPDU exceeds maximum length limit
-19	MaxMMPDULengthExceeded	MAC management frame exceeds maximum length limit
-20	MaxMSDULengthExceeded	MSDU exceeds maximum length limit
-21	UnexpectedProtectedFrame	Invalid value of protected bit for this frame type
-22	UnsupportedTriggerType	Unsupported trigger frame type
-23	UnknownHELTFTypeAndGI	Unknown guard interval (GI) and high-efficiency long training field (HE-LTF) type
-24	UnknownAPTxPower	Unknown value for AP Tx Power subfield of Common Info field
-25	UnknownAID12Value	Unknown value for AID12 subfield of User Info field
-26	UnknownRUAllocation	Unknown value for B7-B1 in RU Allocation subfield of User Info field
-27	UnknownULMCS	Unknown value for UL MCS subfield of User Info field
-28	UnknownTargetRSSI	Unknown value for UL Target RSSI subfield of User Info field
-29	UnsupportedBARType	Unsupported value for BAR Type subfield of BAR Control field
-30	MissingUserInfo	Received trigger frame contains invalid User Info field
-31	InvalidLSIGLength	Invalid value for UL Length subfield of Common Info field, corresponding to length of legacy signal (L-SIG) field.

An enumeration value other than `0` means that MPDU decoding failed. If the decoding fails, the `cfgMAC` output displays no properties, and the function returns the `payload` output as an empty cell array.

Data Types: `int16`

Version History

Introduced in R2019a

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`wlanMACFrame` | `wlanAMPDUDeaggregate`

Objects

`wlanMACFrameConfig` | `wlanMACManagementConfig`

Topics

“Generate and Parse WLAN MAC Frames”

wlanMSDULengths

Calculate MSDU lengths

Syntax

```
msduLengths = wlanMSDULengths(frameLength,macConfig)
msduLengths = wlanMSDULengths(frameLength,macConfig,phyConfig)
```

Description

`msduLengths = wlanMSDULengths(frameLength,macConfig)` returns `msduLengths`, a vector of medium access control (MAC) service data unit (MSDU) lengths for the specified MAC frame length and configuration. The function calculates the MSDU lengths by removing the overhead of MAC headers, frame check sequence (FCS), and subframe overheads (if applicable).

`msduLengths = wlanMSDULengths(frameLength,macConfig,phyConfig)` returns MSDU lengths for the specified physical layer (PHY) format configuration object. Use this syntax to return MSDU lengths for aggregate MAC protocol data units (A-MPDUs).

Examples

Generate QoS Data Frame

Generate a QoS Data frame of length 2000 octets.

Create a WLAN MAC frame configuration object.

```
macConfig = wlanMACFrameConfig('FrameType','QoS Data');
```

Calculate the MSDU lengths required to generate a 2000-octet QoS Data frame and display the result.

```
frameLength = 2000;
msduLengths = wlanMSDULengths(frameLength,macConfig);
disp(msduLengths)
```

```
        1970
```

Create a random payload of the obtained MSDU length.

```
msdu = randi([0 255],1,msduLengths);
```

Generate the 2000-octet QoS data frame.

```
[frame,frameLength] = wlanMACFrame(msdu,macConfig);
disp(frameLength)
```

```
        2000
```

Generate HT A-MPDU

Create a MAC frame configuration object.

```
cfgMAC = wlanMACFrameConfig(FrameType="QoS Data", ...
    FrameFormat="HT-Mixed", ...
    MPDUAggregation=true);
```

Create an HT PHY configuration object.

```
cfgPHY = wlanHTConfig(MCS=4);
```

Calculate the MSDU lengths required to generate a 5000-octet A-MPDU frame, displaying the result.

```
msduLengths = wlanMSDULengths(5000,cfgMAC,cfgPHY);
disp(msduLengths)
```

```
        2302        2302         294
```

Create MSDUs with random data using the MSDU length vector.

```
numMSDUs = numel(msduLengths);
payload = cell(1,numMSDUs);
 for i = 1:numMSDUs
     payload{i} = randi([0 255],1,msduLengths(i));
 end
```

Generate the 5000-octet A-MPDU.

```
[frame,frameLength] = wlanMACFrame(payload,cfgMAC,cfgPHY);
disp(frameLength)
```

```
        5000
```

Input Arguments

`frameLength` — Total length of MAC frame

integer in the interval [28, 6,500,631]

Total length of the MAC frame, specified as an integer in the interval [28, 6,500,631].

Note The maximum value of this input depends on the MAC and PHY configuration in accordance with Table 9-25 of [2].

For MAC frames in very-high-throughput (VHT) or high-efficiency (HE) format, this input is the A-MPDU pre-end-of-frame padding (APEP) length. For all other formats, this input is the physical layer convergence procedure (PLCP) service data unit PSDU length.

Note APEP length is always a multiple of four octets due to final subframe padding. If you do not specify this argument as a multiple of four octets for VHT- or HE-format frames, the function rounds the value to the nearest multiple of four.

Data Types: `double`

`macConfig` — MAC frame configuration
`wlanMACFrameConfig` object (default)

MAC frame configuration, specified as a `wlanMACFrameConfig` object. This object defines the type of MAC frame and its applicable properties.

`phyConfig` — PHY format configuration
`wlanHTConfig` object (default) | `wlanVHTConfig` object | `wlanHESUConfig` object

PHY format configuration, specified as a configuration object of type `wlanHESUConfig`, `wlanVHTConfig`, or `wlanHTConfig`. The value you specify must be compatible with the frame format specified in the `macConfig` input.

- If the `FrameFormat` property of `macConfig` is `'HE-SU'` or `'HE-EXT-SU'`, then you must specify this argument as a `wlanHESUConfig` object.
- If the `FrameFormat` property of `macConfig` is `'VHT'`, then you must specify this argument as a `wlanVHTConfig` object.
- If the `FrameFormat` property of `macConfig` is `'HT-Mixed'`, then you must specify this argument as a `wlanHTConfig` object.

Specify this argument to:

- Ensure that the frame does not exceed the transmission time limit.
- Add end-of-frame (EOF) padding to VHT- or HE-format frames.
- Maintain minimum start spacing between MPDUs in an A-MPDU.

Output Arguments

`msduLengths` — MSDU lengths
vector of integers

MSDU lengths, returned as a vector of integers. The number of elements in this vector corresponds to the number of MSDUs. Each element is the length of its corresponding MSDU.

Data Types: `double`

Version History

Introduced in R2018b

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

[2] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanMACFrameConfig` | `wlanMACFrame`

wlanNonHTData

Generate non-HT-Data field waveform

Syntax

```
y = wlanNonHTData(psdu,cfg)
y = wlanNonHTData(psdu,cfg,scramInit)
y = wlanNonHTData( ___ ,OversamplingFactor=osf)
```

Description

`y = wlanNonHTData(psdu,cfg)` generates a non-HT Data field time-domain waveform for "PSDU" on page 3-363 bits `psdu` and non-HT transmission parameters `cfg`.

`y = wlanNonHTData(psdu,cfg,scramInit)` specifies the scrambler initialization state.

`y = wlanNonHTData( ___ ,OversamplingFactor=osf)` generates an oversampled non-HT Data field waveform for the specified oversampling factor. For more information about oversampling, see "FFT-Based Oversampling" on page 3-365.

Examples

Generate Non-HT-Data Waveform

Generate the waveform for a 20MHz non-HT-Data field for 36 Mbps.

Create a non-HT configuration object and assign `MCS` to 5.

```
cfg = wlanNonHTConfig('MCS',5);
```

Assign random data to the PSDU and generate the data field waveform.

```
psdu = randi([0 1],cfg.PSDULength*8,1);
y = wlanNonHTData(psdu,cfg);
size(y)
```

```
ans = 1×2

        4480           1
```

Generate Data Field Signal of Non-HT Transmission

Configure transmission parameters by creating a `wlanNonHTConfig` object, specifying a channel bandwidth of 80 MHz and static bandwidth operation.

```
cfg = wlanNonHTConfig('ChannelBandwidth','CBW80','SignalChannelBandwidth',true, ...
     'BandwidthOperation','Static');
```

Generate a random PSDU of the appropriate length.

```
psdu = randi([0 1],8*cfg.PSDULength,1,'int8');
```

Generate the initial pseudorandom scrambler sequence.

```
[range,numBits] = scramblerRange(cfg);
scramInit = randi(range);
```

Generate the non-HT Data field signal.

```
y = wlanNonHTData(psdu,cfg,scramInit);
```

Input Arguments

psdu — PSDU bits

binary-valued column vector

Physical layer convergence procedure (PLCP) service data unit (“PSDU” on page 3-363) bits, specified as a binary-valued column vector of length $8 \times L$, where L is the PSDU length in bytes. To specify L, set the `PSDULength` property of the `cfg` input.

Data Types: `double`

cfg — Non-HT transmission parameters

`wlanNonHTConfig` object

Non-HT transmission parameters, specified as a `wlanNonHTConfig` object.

scramInit — Initial scrambler state or initial pseudorandom scrambler sequence

93 (default) | integer in the interval [1, 127] | binary-valued column vector

Initial scrambler state or initial pseudorandom scrambler sequence for each generated packet.

When you disable bandwidth signaling by setting the `SignalchannelBandwidth` property of the `cfg` input to `0` (`false`), this input represents the initial scrambler state. In this case, this input must be an integer in the interval [1, 127], or as the corresponding binary-valued column vector of length seven. The default value, `93`, is the example state in section I.1.5.2 of [1].

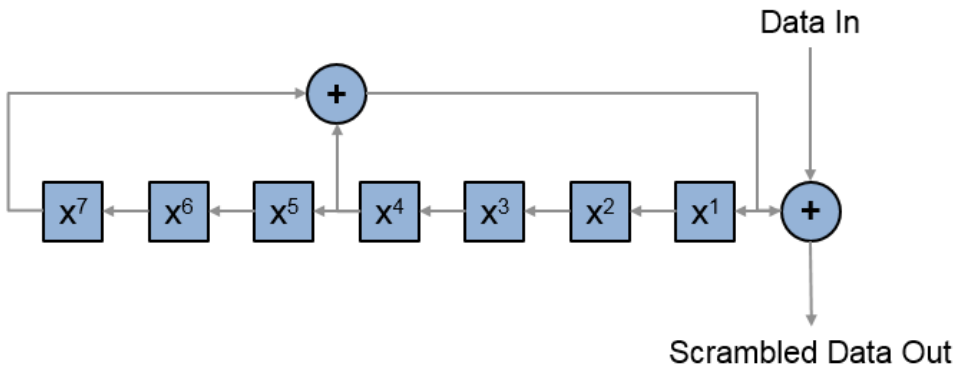
When you enable bandwidth signaling by setting the `SignalchannelBandwidth` property of the `cfg` input to `1` (`true`), this input represents the pseudorandom scrambler sequence described in Table 17-7 of [1]. In this case, this input must be an integer in the interval [*min*, *max*], or the corresponding binary-valued column vector of length N_B. The values of *min*, *max*, and N_B depend on the values of the `BandwidthOperation` and `ChannelBandwidth` properties of the `cfg` input according to this table.

Value of `cfg.BandwidthOperation`	Value of `cfg.ChannelBandwidth`	Value of *min*	Value of *max*	Value of N_B
`'Absent'`	`'CBW20'`	1	31	5
`'Absent'`	`'CBW5'`, `'CBW10'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`	0	31	5

Value of `cfg.BandwidthOperation`	Value of `cfg.ChannelBandwidth`	Value of *min*	Value of *max*	Value of N_B
`'Static'` or `'Dynamic'`	`'CBW20'`	1	15	4
`'Static'` or `'Dynamic'`	`'CBW5'`, `'CBW10'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`	0	15	4

If you do not specify this input, the function uses the N_B most significant bits of the default value, `93`.

Section 17.3.5.5 of [1] specifies the scrambling and descrambling process applied to the transmitted data. The header and data fields that follow the scrambler initialization field (including data padding bits) are scrambled by XORing each bit with a length-127 periodic sequence generated by the polynomial $S(x) = x^7 + x^4 + 1$. The octets of the PSDU are placed into a bit stream, and, within each octet, bit 0 (LSB) is first and bit 7 (MSB) is last. This figure demonstrates the sequence generation and XOR operation.



Conversion from integer to bits uses left-MSB orientation. For example, initializing the scrambler with decimal `1`, the bits map to these elements.

Element	X^7	X^6	X^5	X^4	X^3	X^2	X^1
Bit Value	0	0	0	0	0	0	1

To generate the bit stream equivalent to a decimal, use the `int2bit` function. For example, for decimal `1`:

```
int2bit(1,7)'
ans =
```

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

Example: `[1; 0; 1; 1; 1; 0; 1]` conveys the scrambler initialization state of 93 as a binary vector.

Data Types: `double` | `int8`

`osf` — Oversampling factor

1 (default) | scalar greater than or equal to 1

Oversampling factor, specified as a scalar greater than or equal to 1. The oversampled cyclic prefix length must be an integer number of samples.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

y — Non-HT Data field time-domain waveform

complex-valued matrix

Non-HT Data field time-domain waveform, returned as a complex-valued matrix of size N_S-by-N_T.

- N_S is the number of time domain samples
- N_T is the number of transmit antennas.

Data Types: `double`

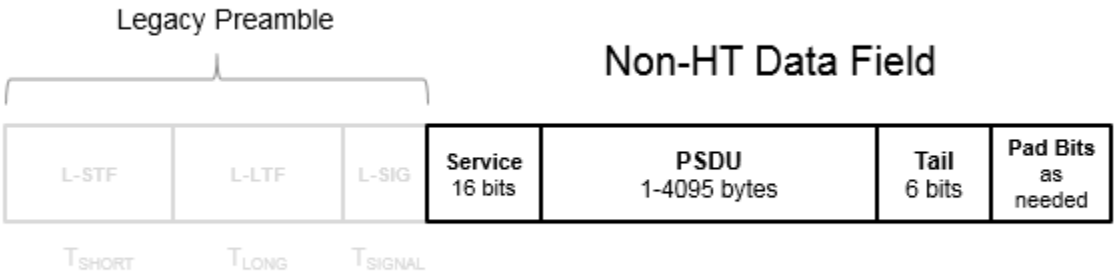
More About

PSDU

Physical layer (PHY) service data unit (PSDU). This field is composed of a variable number of octets. The minimum is 0 (zero) and the maximum is 2500. For more information, see IEEE Std 802.11™-2012, Section 15.3.5.7.

Non-HT Data field

The non-high throughput Data (non-HT Data) field is used to transmit MAC frames and is composed of a service field, a PSDU, tail bits, and pad bits.

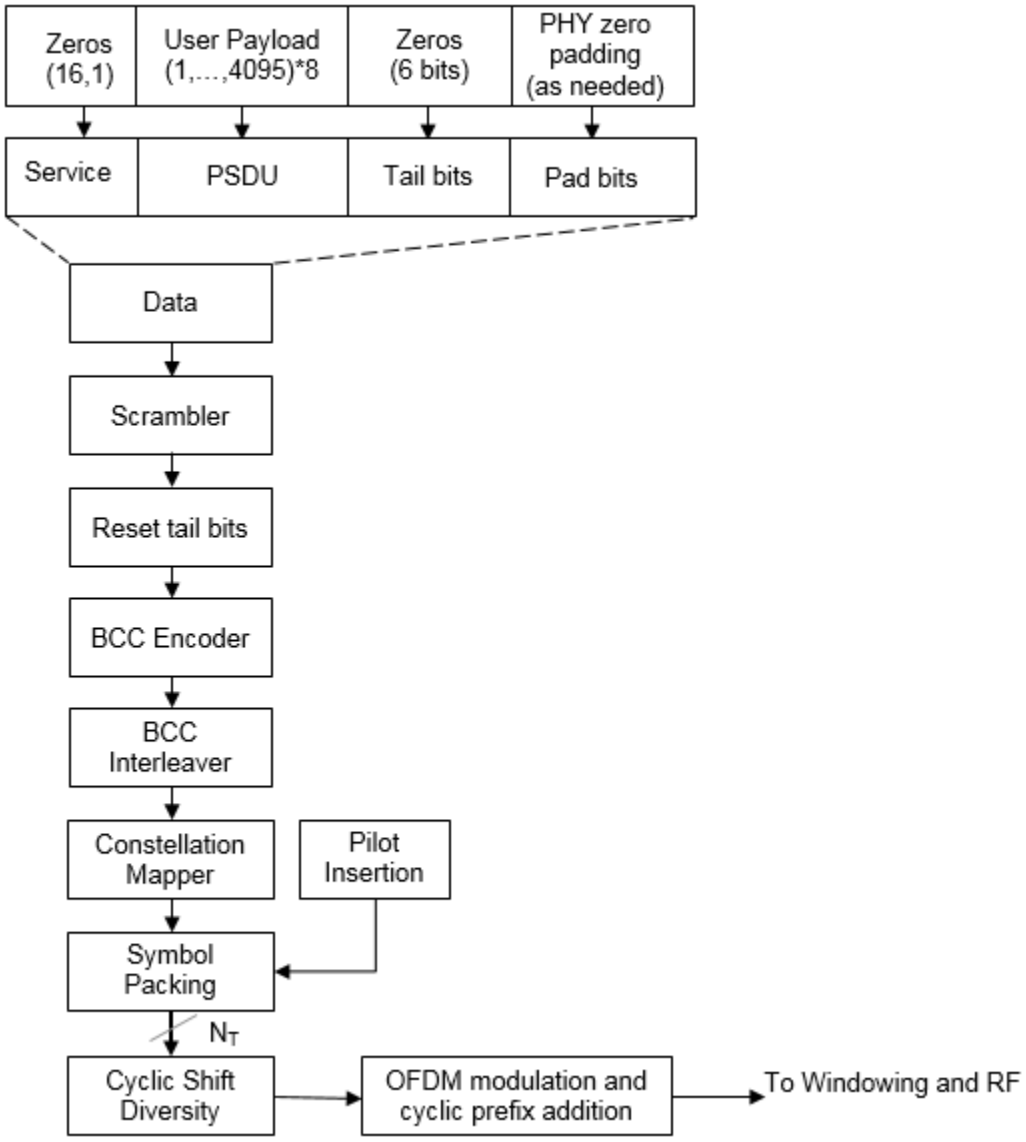


- **Service field** — Contains 16 zeros to initialize the data scrambler.
- **PSDU** — Variable-length field containing the PLCP service data unit (PSDU).
- **Tail** — Tail bits required to terminate a convolutional code. The field uses six zeros for the single encoding stream.
- **Pad Bits** — Variable-length field required to ensure that the non-HT data field contains an integer number of symbols.

Algorithms

Non-HT Data Field Processing

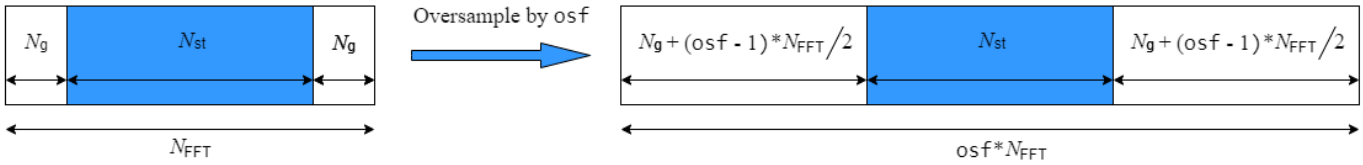
The non-HT Data field follows the L-SIG in the packet. For algorithm details, refer to section 17.3.5 of [1]. The non-HT Data includes the user payload in the PSDU plus 16 service bits, six tail bits, and additional padding bits as required to fill out the last OFDM symbol. The function performs transmitter processing on the non-HT Data field and generates the time-domain waveform.



FFT-Based Oversampling

An oversampled signal is a signal sampled at a frequency that is higher than the Nyquist rate. WLAN signals maximize occupied bandwidth by using small guardbands, which can pose problems for anti-imaging and anti-aliasing filters. Oversampling increases guardband width relative to the total signal bandwidth, thereby increasing the number of samples in the signal.

This function performs oversampling by using a larger IFFT and zero pad when generating an OFDM waveform. This diagram shows the oversampling process for an OFDM waveform with N_{FFT} subcarriers comprising N_{g} guardband subcarriers on either side of N_{st} occupied bandwidth subcarriers.



Version History

Introduced in R2015b

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanNonHTConfig` | `wlanNonHTDataRecover` | `wlanLSIG`

wlanNonHTDataBitRecover

Recover PSDU from non-HT Data field

Syntax

```
psdu = wlanNonHTDataBitRecover(sym,noiseVarEst,cfg)
psdu = wlanNonHTDataBitRecover(sym,noiseVarEst,csi,cfg)
[psdu,scramInit] = wlanNonHTDataBitRecover( ___ ,'OFDMSymbolOffset',symOffset)
```

Description

`psdu = wlanNonHTDataBitRecover(sym,noiseVarEst,cfg)` recovers `psdu`, a column vector of physical layer service data unit (PSDU) bits, from `sym`, the demodulated and equalized orthogonal frequency-division multiplexing (OFDM) symbols comprising the non-HT Data field of a non-high-throughput (non-HT) waveform. The function recovers the PSDU by using noise variance estimate `noiseVarEst` and non-HT transmission parameters `cfg`.

`psdu = wlanNonHTDataBitRecover(sym,noiseVarEst,csi,cfg)` enhances the demappinng of OFDM subcarriers by using channel state information `csi`.

`[psdu,scramInit] = wlanNonHTDataBitRecover( ___ ,'OFDMSymbolOffset',symOffset)` recovers initial scrambler state `scramInit` for any combination of input arguments from previous syntaxes.

Examples

Recover PSDU from Non-HT Data Signal

Configure and generate a non-HT time-domain waveform.

```
cfg = wlanNonHTConfig('MCS',4);
bits = randi([0 1],8*cfg.PSDULength,1,'int8');
waveform = wlanWaveformGenerator(bits,cfg);
```

Transmit the waveform through an additive white Gaussian noise (AWGN) channel with a signal-to-noise ratio (SNR) of 30.

```
snr = 30;
rxWaveform = awgn(waveform,snr);
```

Extract the non-HT Data field from the received waveform.

```
field = 'NonHT-Data';
ind = wlanFieldIndices(cfg,field);
rx = rxWaveform(ind(1):ind(2),:);
```

Recover the frequency-domain signal by OFDM demodulating the time-domain data signal.

```
sym = wlanNonHTOFDMDemodulate(rx,field,cfg);
```

Extract the data subcarriers from the demodulated signal.

```
info = wlanNonHTOFDMInfo(field,cfg);
sym = sym(info.DataIndices,:,:);
```

Recover the PSDU and confirm that it matches the transmitted PSDU.

```
noiseVarEst = 10^(-snr/10);
psdu = wlanNonHTDataBitRecover(sym,noiseVarEst,cfg);
isequal(bits,psdu)
```

```
ans = logical
   1
```

Recover Bandwidth Signaling from Initial Scrambler State

Configure and generate a non-HT Data signal with a channel bandwidth of 160 MHz and dynamic bandwidth operation.

```
bandwidth = 'CBW160';
cfg = wlanNonHTConfig('ChannelBandwidth',bandwidth,'PSDULength',1, ...
    'SignalChannelBandwidth',true,'BandwidthOperation','Dynamic');
bits = randi([0 1],8*cfg.PSDULength,1,'int8');
[range,~] = scramblerRange(cfg);
scramInit = randi(range);
y = wlanNonHTData(bits,cfg,scramInit);
```

Transmit the waveform through an AWGN channel with an SNR of 50.

```
snr = 50;
noiseVarEst = 10^(-snr/10);
rx = awgn(y,snr);
```

Recover the frequency-domain signal by OFDM demodulating the non-HT Data signal, specifying an OFDM symbol sampling offset.

```
field = 'NonHT-Data';
symOffset = 0.5;
sym = wlanNonHTOFDMDemodulate(rx,field,bandwidth,'OFDMSymbolOffset',symOffset);
```

Extract the data subcarriers.

```
info = wlanNonHTOFDMInfo(field,bandwidth);
sym = sym(info.DataIndices,:);
```

Recover the first 20 MHz subchannel of the PSDU, enhancing the demapping of the OFDM subcarriers by specifying channel state information. Confirm that the received and transmitted PSDUs match.

```
csi = ones(48,1);
[psdu,scramInit] = wlanNonHTDataBitRecover(sym(1:48,:),noiseVarEst,csi,cfg);
isequal(bits,psdu)
```

```
ans = logical
   1
```

Recover and display bandwidth signaling by interpreting the scrambler state.

```
[bandwidth,dyn] = wlanInterpretScramblerState(scramInit)
```

```
bandwidth =
'CBW160'
```

```
dyn = logical
   1
```

Input Arguments

`sym` — Demodulated and equalized OFDM symbols

complex-valued matrix

Demodulated and equalized OFDM symbols comprising the non-HT Data field, specified as a complex-valued matrix of size 48-by-N_{sym}, where N_{sym} is the number of OFDM symbols.

Data Types: `double`
Complex Number Support: Yes

`noiseVarEst` — Noise variance estimate

nonnegative scalar

Noise variance estimate, specified as a nonnegative scalar.

Data Types: `double`

`cfg` — Non-HT transmission parameters

`wlanNonHTConfig` object

Non-HT transmission parameters, specified as a `wlanNonHTConfig` object.

`csi` — Channel state information

real-valued column vector

Channel state information, specified as a real-valued column vector of length 48.

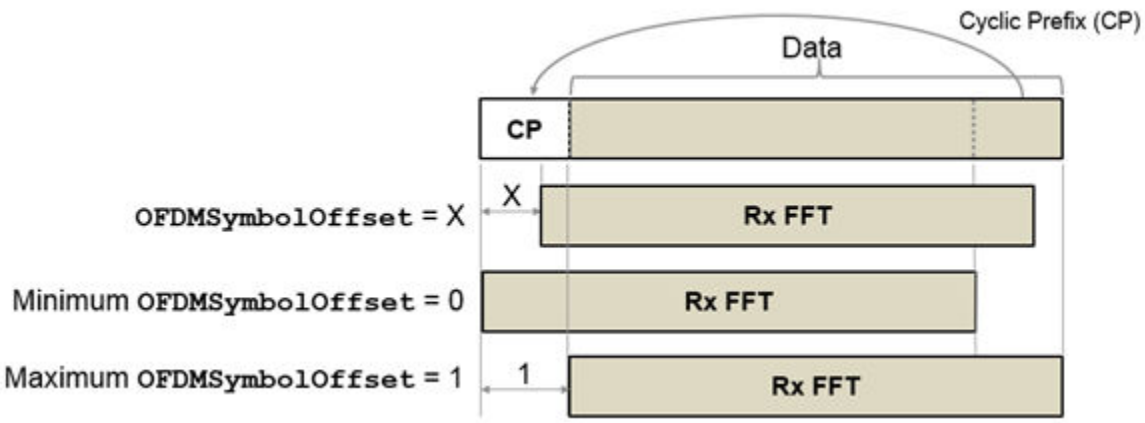
Data Types: `double`

`symOffset` — OFDM symbol sampling offset

`0.75` (default) | scalar in the interval [0, 1]

OFDM symbol sampling offset, as a fraction of the cyclic prefix length, specified as a scalar in the interval [0, 1].

The value that you specify indicates the start location for OFDM demodulation relative to the beginning of the cyclic prefix.



Example: `0.45`

Data Types: `double`

Output Arguments

`psdu` — Recovered PSDU bits

binary-valued column vector

Recovered PSDU bits, returned as a binary-valued column vector of length $8 \times L$, where L is the PSDU length in bytes. To specify L, set the `PSDULength` property of the `cfg` input.

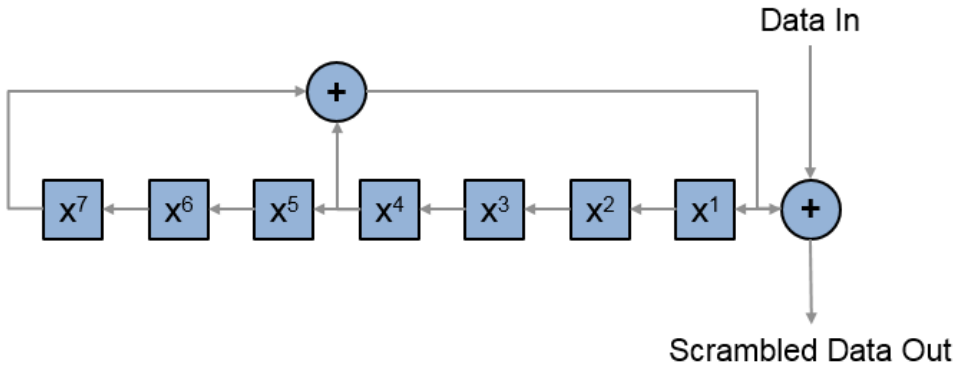
Data Types: `int8`

`scramInit` — Initial scrambler state

integer in the interval [1, 127] | binary-valued column vector

Initial scrambler state, returned as an integer in the interval [1, 127], or the corresponding binary-valued column vector of length 7.

Section 17.3.5.5 [1]of specifies the scrambling and descrambling process applied to the transmitted data. The header and data fields that follow the scrambler initialization field (including data padding bits) are scrambled by XORing each bit with a length-127 periodic sequence generated by the polynomial $S(x) = x^7 + x^4 + 1$. The octets of the PSDU are placed into a bit stream, and, within each octet, bit 0 (LSB) is first and bit 7 (MSB) is last. This figure demonstrates the sequence generation and XOR operation.



Conversion from integer to bits uses left-MSB orientation. For example, initializing the scrambler with decimal `1`, the bits map to these elements.

Element	X^7	X^6	X^5	X^4	X^3	X^2	X^1
Bit Value	0	0	0	0	0	0	1

To generate the bit stream equivalent to a decimal, use the `int2bit` function. For example, for decimal `1`:

```
int2bit(1,7)'
ans =

     0     0     0     0     0     0     1
```

Example: `[1; 0; 1; 1; 1; 0; 1]` conveys the scrambler initialization state of 93 as a binary-valued column vector.

Data Types: `double`

Version History

Introduced in R2020b

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`wlanInterpretScramblerState` | `wlanNonHTData` | `wlanNonHTOFDMDemodulate` | `wlanNonHTOFDMInfo`

Objects

`wlanNonHTConfig`

wlanNonHTDataRecover

Recover non-HT Data

Syntax

```
recData = wlanNonHTDataRecover(rxSig,chEst,noiseVarEst,cfg)
recData = wlanNonHTDataRecover(rxSig,chEst,noiseVarEst,cfg,Name,Value)
[recData,eqSym] = wlanNonHTDataRecover( ___ )
[recData,eqSym,cpe,scramInit] = wlanNonHTDataRecover( ___ )
[recData,eqSym,cpe,scramInit,ae] = wlanNonHTDataRecover( ___ )
```

Description

`recData = wlanNonHTDataRecover(rxSig,chEst,noiseVarEst,cfg)` recovers "Non-HT Data field" on page 3-379[16] bits from received time-domain waveform `rxSig`, channel estimate data `chEst`, noise variance estimate `noiseVarEst`, and non-high-throughput (non-HT) transmission parameters `cfg`.

Note This function only supports data recovery for OFDM modulation.

`recData = wlanNonHTDataRecover(rxSig,chEst,noiseVarEst,cfg,Name,Value)` specifies recovery algorithm parameters by using one or more name-value pair arguments in addition to the inputs from the previous syntax.

`[recData,eqSym] = wlanNonHTDataRecover( ___ )` returns the recovered equalized symbols.

`[recData,eqSym,cpe,scramInit] = wlanNonHTDataRecover( ___ )` returns recovered common phase error `cpe` and initial scrambler state `scramInit`.

`[recData,eqSym,cpe,scramInit,ae] = wlanNonHTDataRecover( ___ )` returns `ae`, the average amplitude error with respect to the estimated received pilots.

Examples

Recover Non-HT Data Bits

Create a non-HT configuration object having a PSDU length of 2048 bytes. Generate the corresponding data sequence.

```
cfg = wlanNonHTConfig('PSDULength',2048);
txBits = randi([0 1],8*cfg.PSDULength,1);
txSig = wlanNonHTData(txBits,cfg);
```

Pass the signal through an AWGN channel with a signal-to-noise ratio of 15 dB.

```
rxSig = awgn(txSig,15);
```

16 IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.

Recover the data and determine the number of bit errors.

```
rxBits = wlanNonHTDataRecover(rxSig,ones(52,1),0.05,cfg);
[numerr,ber] = biterr(rxBits,txBits)
```

```
numerr = 0
```

```
ber = 0
```

Recover Non-HT Data Bits Using Zero-Forcing Algorithm

Create a non-HT configuration object, specifying a PSDU length of 1024 bytes. Generate the corresponding non-HT data sequence.

```
cfg = wlanNonHTConfig('PSDULength',1024);
txBits = randi([0 1],8*cfg.PSDULength,1);
txSig = wlanNonHTData(txBits,cfg);
```

Pass the signal through an AWGN channel with a signal-to-noise ratio of 10 dB.

```
rxSig = awgn(txSig,10);
```

Recover the data by using a zero-forcing algorithm and determine the number of bit errors.

```
chanEst = ones(52,1);
noiseVarEst = 0.1;
rxBits = wlanNonHTDataRecover(rxSig,chanEst,noiseVarEst,cfg,'EqualizationMethod','ZF');
[numerr,ber] = biterr(rxBits,txBits)
```

```
numerr = 0
```

```
ber = 0
```

Recover Non-HT Data in Fading Channel

Configure a non-HT data object.

```
cfg = wlanNonHTConfig;
```

Generate and transmit a non-HT PSDU.

```
txPSDU = randi([0 1],8*cfg.PSDULength,1);
txSig = wlanNonHTData(txPSDU,cfg);
```

Generate an L-LTF for channel estimation.

```
txLLTF = wlanLLTF(cfg);
```

Create an 802.11g channel with a 3 Hz maximum Doppler shift and a 100 ns RMS path delay. Disable the reset before filtering option so that the L-LTF and data fields use the same channel realization.

```
ch802 = comm.RayleighChannel('SampleRate',20e6,'MaximumDopplerShift',3,'PathDelays',100e-9);
```

Pass the L-LTF and data signals through an 802.11g channel with AWGN.

```
rxLLTF = awgn(ch802(txLLTF),10);
rxSig = awgn(ch802(txSig),10);
```

Demodulate the L-LTF and use it to estimate the fading channel.

```
dLLTF = wlanLLTFDemodulate(rxLLTF,cfg);
chEst = wlanLLTFChannelEstimate(dLLTF,cfg);
```

Recover the non-HT data using the L-LTF channel estimate and determine the number of bit errors in the transmitted packet.

```
rxPSDU = wlanNonHTDataRecover(rxSig,chEst,0.1,cfg);

[numErr,ber] = biterr(txPSDU,rxPSDU)
```

```
numErr = 0

ber = 0
```

Recover Non-HT Data Field and Calculate Amplitude Error

Investigate how applying an amplitude droop affects the amplitude error of the non-HT Data field generated from a non-HT signal.

Configure a non-HT transmission with default parameters, then generate the corresponding non-HT Data field.

```
cfgNonHT = wlanNonHTConfig;
psduLength = 8*cfgNonHT.PSDULength;
bits = randi([0 1],psduLength,1);
tx = wlanNonHTData(bits,cfgNonHT);
```

Modify the signal by applying an amplitude droop of 10 dB, starting at the halfway point.

```
signalLength = size(tx,1);
droopGain = 10;
droopGainLinear = 10^(droopGain/20);
txDroop = [ones(signalLength/2,1); droopGainLinear*ones(signalLength/2,1)].*tx;
```

Specify a channel estimate.

```
chEst = ones(52,1);
```

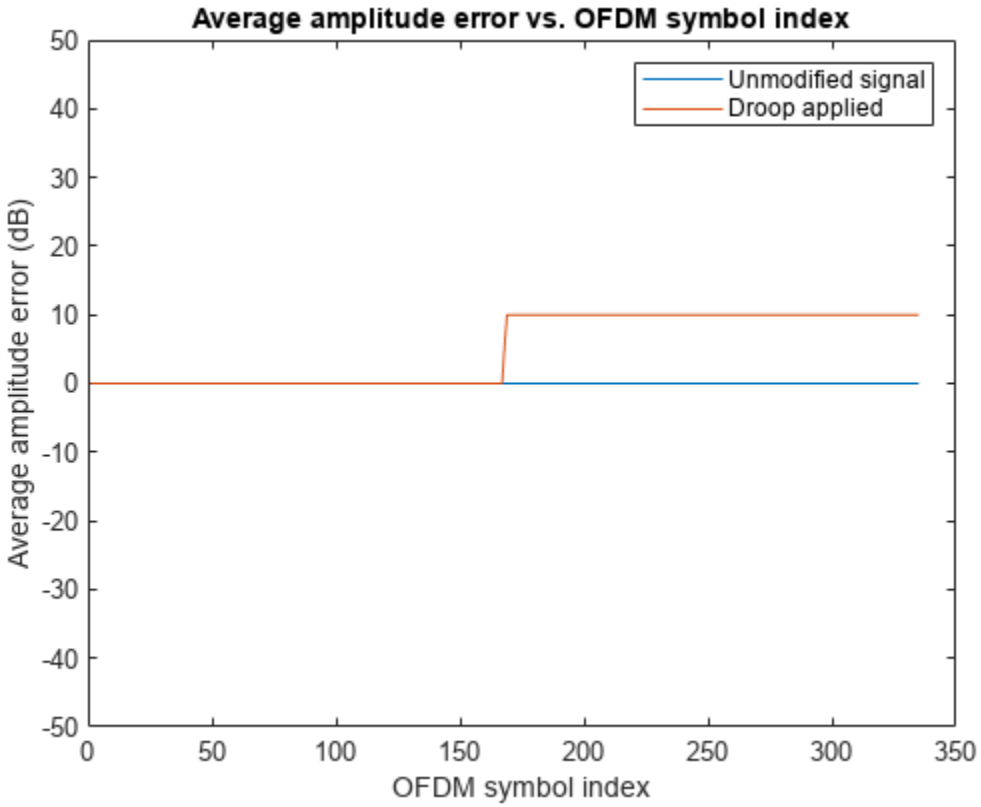
Recover the bits from the ideal and impaired non-HT Data fields and confirm that the recovered bits match the transmitted bits.

```
[databits_1, eqSym_1, cpe_1, scram_1, ae_1] = wlanNonHTDataRecover(tx, chEst, 0, cfgNonHT, Equali
[databits_2, eqSym_2, cpe_2, scram_2, ae_2] = wlanNonHTDataRecover(txDroop, chEst, 0, cfgNonHT, E
isequal(databits_1, databits_2, bits)
```

```
ans = logical
   1
```

Plot the absolute value of the measured amplitude errors for the ideal and impaired non-HT Data fields.

```
plot(abs(ae_1))
title('Average amplitude error vs. OFDM symbol index')
ylabel('Average amplitude error (dB)')
xlabel('OFDM symbol index')
ylim([-50 50])
hold on
plot(abs(ae_2))
legend('Unmodified signal', 'Droop applied')
```



Input Arguments

`rxSig` — Received non-HT Data field time-domain waveform

complex-valued matrix

Received non-HT Data field time-domain waveform, specified as a complex-valued matrix of size N_S-by-N_R.

- N_S is the number of time-domain samples in the non-HT Data field. If you specify this input as a matrix with more than N_S rows, the function does not use the redundant samples after the first N_S.
- N_R is the number of receive antennas.

Data Types: `double`

`chEst` — Channel estimate data

complex-valued array

Channel estimate data, specified as a complex-valued array of size N_{ST}-by-1-by-N_R.

- N_{ST} is the number of occupied subcarriers.
- N_R is the number of receive antennas.

The singleton dimension corresponds to the single transmitted stream in the legacy long training field (L-LTF), which includes the combined cyclic shifts if the transmitter uses multiple antennas.

Data Types: `double`

`noiseVarEst` — Noise variance estimate

nonnegative scalar

Noise variance estimate, specified as a nonnegative scalar.

Example: `0.7071`

Data Types: `double`

`cfg` — Non-HT transmission parameters

`wlanNonHTConfig` object

Non-HT transmission parameters, specified as a `wlanNonHTConfig` object.

Name-Value Pair Arguments

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

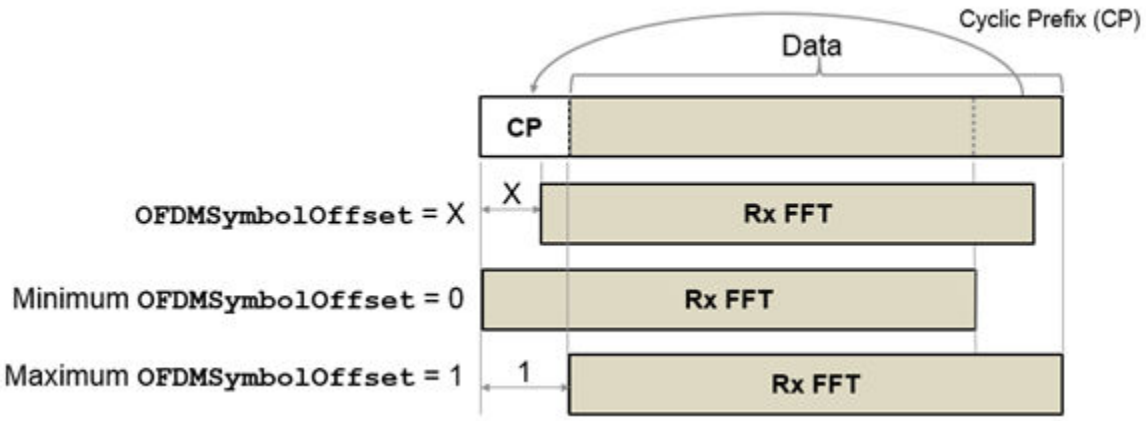
Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'PilotPhaseTracking','None'` disables pilot phase tracking.

`OFDMSymbolOffset` — OFDM symbol sampling offset

`0.75` (default) | scalar in the interval [0, 1]

OFDM symbol sampling offset represented as a fraction of the cyclic prefix (CP) length, specified as the comma-separated pair consisting of `'OFDMSymbolOffset'` and a scalar in the interval [0, 1]. The value you specify indicates the start location for OFDM demodulation relative to the beginning of the CP. The value `0` represents the start of the CP, and the value `1` represents the end of the CP.



Data Types: double

EqualizationMethod — Equalization method
'MMSE' (default) | 'ZF'

Equalization method, specified as one of these values.

- 'MMSE' — The receiver uses a minimum mean-square error equalizer.
- 'ZF' — The receiver uses a zero-forcing equalizer.

When the received signal has multiple receive antennas, the function exploits receiver diversity during equalization. When the number of transmitted space-time streams is one and you specify this argument as 'ZF', the function performs maximal-ratio combining.

Data Types: char | string

PilotPhaseTracking — Pilot phase tracking
'PreEQ' (default) | 'None'

Pilot phase tracking, specified as the comma-separated pair consisting of 'PilotPhaseTracking' and one of these values.

- 'PreEQ' — Enable pilot phase tracking, which the function performs before any equalization operation.
- 'None' — Disable pilot phase tracking.

Data Types: char | string

PilotAmplitudeTracking — Pilot amplitude tracking
'None' (default) | 'PreEQ'

Pilot amplitude tracking, specified as the comma-separated pair consisting of 'PilotAmplitudeTracking' and one of these values.

- 'None' — Disable pilot amplitude tracking.
- 'PreEQ' — Enable pilot amplitude tracking, which the function performs before any equalization operation.

Note Due to the limitations of the algorithm used, disable pilot amplitude tracking when filtering a waveform through a MIMO fading channel.

Data Types: char | string

Output Arguments

recData — Recovered PSDU bits
binary-valued column vector

Recovered physical layer convergence procedure (PLCP) service data unit (“PSDU” on page 3-363) bits, returned as a binary-valued column vector of length $8 \times L$, where L is the value of the PSDULength property of the cfg input.

Data Types: int8

`eqSym` — Equalized symbols

complex-valued matrix

Equalized symbols, returned as a complex-valued matrix of size N_{SD}-by-N_{Sym}.

- N_{SD} is the number of data subcarriers.
- N_{Sym} is the number of OFDM symbols in the non-HT Data field.

Data Types: `double`

`cpe` — Common phase error

real-valued column vector

Common phase error, in radians, returned as a real-valued column vector of length N_{Sym}, the number of OFDM symbols in the non-HT Data field. This output is averaged over the receive antennas.

Data Types: `double`

`scramInit` — Recovered initial scrambler state

integer in the interval [0, 127]

Recovered initial scrambler state, returned as an integer in the interval [0, 127]. For more information, see section 17.3.5.5 of [1].

Data Types: `int8`

`ae` — Average amplitude error

real-valued array

Average amplitude error, in dB, returned as a real-valued array of size N_{Sym}-by- N_R.

- N_{Sym} is the number of OFDM symbols in the HT-Data field.
- N_R is the number of receive antennas.

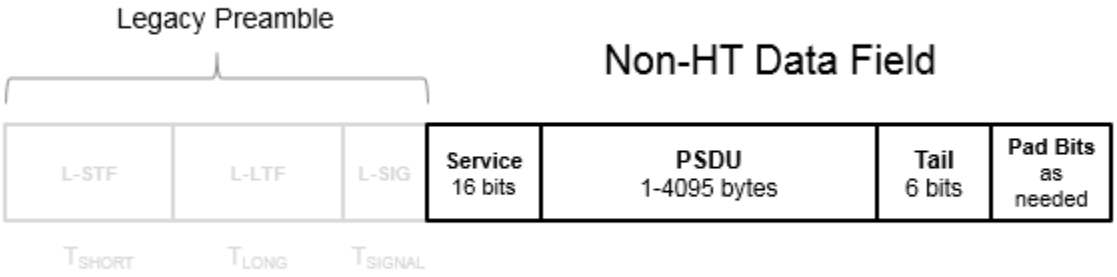
Each element of this matrix contains the amplitude error for all subcarriers with respect to the estimated received pilots for the corresponding OFDM symbol and receive antenna.

Data Types: `double`

More About

Non-HT Data field

The non-high throughput Data (non-HT Data) field is used to transmit MAC frames and is composed of a service field, a PSDU, tail bits, and pad bits.



- **Service field** — Contains 16 zeros to initialize the data scrambler.
- **PSDU** — Variable-length field containing the PLCP service data unit (PSDU).
- **Tail** — Tail bits required to terminate a convolutional code. The field uses six zeros for the single encoding stream.
- **Pad Bits** — Variable-length field required to ensure that the non-HT data field contains an integer number of symbols.

Version History

Introduced in R2015b

Pilot amplitude tracking

You can perform pilot amplitude tracking by setting the `PilotAmplitudeTracking` input argument to `'PreEQ'`. The average amplitude error is returned in the output argument `ae`.

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanNonHTConfig` | `wlanNonHTData`

wlanNonHTOFDMDemodulate

Demodulate fields of non-HT OFDM waveform

Syntax

```
sym = wlanNonHTOFDMDemodulate(rx,field,cfg)
sym = wlanNonHTOFDMDemodulate(rx,field,bandwidth)
sym = wlanNonHTOFDMDemodulate( ___ ,'OFDMSymbolOffset',symOffset)
```

Description

`sym = wlanNonHTOFDMDemodulate(rx,field,cfg)` recovers a frequency-domain signal by orthogonal frequency-division multiplexing (OFDM) demodulating non-high-throughput (non-HT) time-domain signal `rx`. The function demodulates `rx` by using non-HT transmission parameters `cfg` and signal field value `field`.

`sym = wlanNonHTOFDMDemodulate(rx,field,bandwidth)` specifies the channel bandwidth of the transmission.

`sym = wlanNonHTOFDMDemodulate( ___ ,'OFDMSymbolOffset',symOffset)` specifies the OFDM symbol sampling offset as a fraction of the cyclic prefix length in addition to any combination of input arguments from the previous syntaxes..

Examples

Recover PSDU from Non-HT Data Signal

Configure and generate a non-HT time-domain waveform.

```
cfg = wlanNonHTConfig('MCS',4);
bits = randi([0 1],8*cfg.PSDULength,1,'int8');
waveform = wlanWaveformGenerator(bits,cfg);
```

Transmit the waveform through an additive white Gaussian noise (AWGN) channel with a signal-to-noise ratio (SNR) of 30.

```
snr = 30;
rxWaveform = awgn(waveform,snr);
```

Extract the non-HT Data field from the received waveform.

```
field = 'NonHT-Data';
ind = wlanFieldIndices(cfg,field);
rx = rxWaveform(ind(1):ind(2),:);
```

Recover the frequency-domain signal by OFDM demodulating the time-domain data signal.

```
sym = wlanNonHTOFDMDemodulate(rx,field,cfg);
```

Extract the data subcarriers from the demodulated signal.

```
info = wlanNonHTOFDMInfo(field,cfg);
sym = sym(info.DataIndices,:,:);
```

Recover the PSDU and confirm that it matches the transmitted PSDU.

```
noiseVarEst = 10^(-snr/10);
psdu = wlanNonHTDataBitRecover(sym,noiseVarEst,cfg);
isequal(bits,psdu)
```

```
ans = logical
   1
```

Recover Bandwidth Signaling from Initial Scrambler State

Configure and generate a non-HT Data signal with a channel bandwidth of 160 MHz and dynamic bandwidth operation.

```
bandwidth = 'CBW160';
cfg = wlanNonHTConfig('ChannelBandwidth',bandwidth,'PSDULength',1, ...
    'SignalChannelBandwidth',true,'BandwidthOperation','Dynamic');
bits = randi([0 1],8*cfg.PSDULength,1,'int8');
[range,~] = scramblerRange(cfg);
scramInit = randi(range);
y = wlanNonHTData(bits,cfg,scramInit);
```

Transmit the waveform through an AWGN channel with an SNR of 50.

```
snr = 50;
noiseVarEst = 10^(-snr/10);
rx = awgn(y,snr);
```

Recover the frequency-domain signal by OFDM demodulating the non-HT Data signal, specifying an OFDM symbol sampling offset.

```
field = 'NonHT-Data';
symOffset = 0.5;
sym = wlanNonHTOFDMDemodulate(rx,field,bandwidth,'OFDMSymbolOffset',symOffset);
```

Extract the data subcarriers.

```
info = wlanNonHTOFDMInfo(field,bandwidth);
sym = sym(info.DataIndices,:);
```

Recover the first 20 MHz subchannel of the PSDU, enhancing the demapping of the OFDM subcarriers by specifying channel state information. Confirm that the received and transmitted PSDUs match.

```
csi = ones(48,1);
[psdu,scramInit] = wlanNonHTDataBitRecover(sym(1:48,:),noiseVarEst,csi,cfg);
isequal(bits,psdu)
```

```
ans = logical
   1
```

Recover and display bandwidth signaling by interpreting the scrambler state.

```
[bandwidth,dyn] = wlanInterpretScramblerState(scramInit)
```

```
bandwidth =
'CBW160'
```

```
dyn = logical
   1
```

Input Arguments

`rx` — Received time-domain signal

complex-valued matrix

Received time-domain signal, specified as a complex-valued matrix of size N_s-by-N_r.

- N_s is the number of time-domain samples. If N_s is not an integer multiple of the OFDM symbol length, L_s, for the specified field,then the function ignores the remaining `mod(`N_s`,`L_s`)` symbols.
- N_r is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

`field` — Field to be demodulated

`'L-LTF'` | `'L-SIG'` | `'NonHT-Data'`

Field to be demodulated, specified as one of these values.

- `'L-LTF'` – Demodulate legacy long training field (L-LTF).
- `'L-SIG'` – Demodulate the legacy signaling (L-SIG) field.
- `'NonHT-Data'` – Demodulate the non-HT Data field.

Data Types: `char` | `string`

`cfg` — Non-HT transmission parameters

`wlanNonHTConfig` object

Non-HT transmission parameters, specified as a `wlanNonHTConfig` object.

`bandwidth` — Channel bandwidth

`'CBW5'` | `'CBW10'` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth, specified as one of these values.

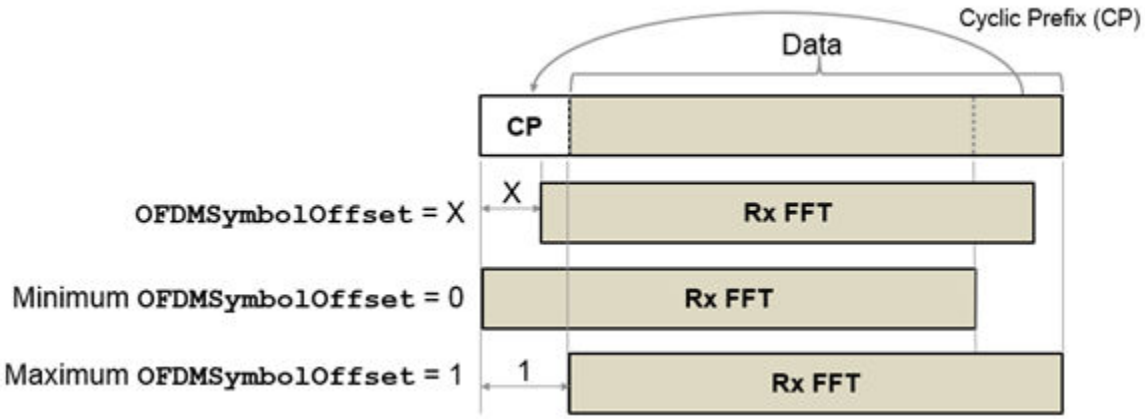
- `'CBW5'` – Channel bandwidth of 5 MHz
- `'CBW10'` – Channel bandwidth of 10 MHz
- `'CBW20'` – Channel bandwidth of 20 MHz
- `'CBW40'` – Channel bandwidth of 40 MHz
- `'CBW80'` – Channel bandwidth of 80 MHz
- `'CBW160'` – Channel bandwidth of 160 MHz

Data Types: `char` | `string`

`symOffset` — OFDM symbol sampling offset
`0.75` (default) | scalar in the interval [0, 1]

OFDM symbol sampling offset, as a fraction of the cyclic prefix length, specified as a scalar in the interval [0, 1].

The value that you specify indicates the start location for OFDM demodulation relative to the beginning of the cyclic prefix.



Example: `0.45`

Data Types: `double`

Output Arguments

`sym` — Demodulated frequency-domain signal
complex-valued array

Demodulated frequency-domain signal, returned as a complex-valued array of size N_{sc}-by-N_{sym}-by-N_r.

- N_{sc} is the number of active occupied subcarriers in the demodulated field.
- N_{sym} is the number of OFDM symbols.
- N_r is the number of receive antennas.

Data Types: `double`

Version History

Introduced in R2020b

Extended Capabilities

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

See Also

Functions
`wlanNonHTDataBitRecover` | `wlanNonHTOFDMInfo`

wlanNonHTOFDMInfo

OFDM information for non-HT transmission

Syntax

```
info = wlanNonHTOFDMInfo(field)
info = wlanNonHTOFDMInfo(field,bandwidth)
info = wlanNonHTOFDMInfo(field,cfg)
info = wlanNonHTOFDMInfo( ___ ,OversamplingFactor=osf)
```

Description

`info = wlanNonHTOFDMInfo(field)` returns orthogonal frequency-division multiplexing (OFDM) information for the specified field of a non-high-throughput (non-HT) transmission.

`info = wlanNonHTOFDMInfo(field,bandwidth)` specifies the channel bandwidth of the non-HT transmission.

`info = wlanNonHTOFDMInfo(field,cfg)` specifies non-HT transmission parameters.

`info = wlanNonHTOFDMInfo( ___ ,OversamplingFactor=osf)` returns OFDM information for the specified oversampling factor. For more information about oversampling, see “FFT-Based Oversampling” on page 3-389.

Examples

Get OFDM Information for Non-HT-Data Field

Get and display the OFDM information for the non-HT-Data field.

```
info = wlanNonHTOFDMInfo('NonHT-Data');
disp(info);
```

```
                FFTLength: 64
               SampleRate: 20000000
                 CPLength: 16
            NumSubchannels: 1
                  NumTones: 52
    ActiveFrequencyIndices: [52x1 double]
          ActiveFFTIndices: [52x1 double]
               DataIndices: [48x1 double]
              PilotIndices: [4x1 double]
```

Demodulate Non-HT L-LTF and Get OFDM Information

OFDM demodulate the L-LTF in a non-HT transmission, then extract the data and pilot subcarriers.

Generate a WLAN waveform for a non-HT transmission.

```
cfg = wlanNonHTConfig;
bits = [1; 0; 0; 1];
waveform = wlanWaveformGenerator(bits,cfg);
```

Obtain the field indices and extract the L-LTF.

```
ind = wlanFieldIndices(cfg);
rx = waveform(ind.LLTF(1):ind.LLTF(2),:);
```

Perform OFDM demodulation on the L-LTF.

```
sym = wlanLLTFDemodulate(rx,cfg);
```

Get the OFDM information, then extract the data and pilot subcarriers.

```
bandwidth = cfg.ChannelBandwidth;
info = wlanNonHTOFDMInfo('L-LTF',bandwidth);
data = sym(info.DataIndices,:,:);
pilots = sym(info.PilotIndices,:,:);
```

Recover PSDU from Non-HT Data Signal

Configure and generate a non-HT time-domain waveform.

```
cfg = wlanNonHTConfig('MCS',4);
bits = randi([0 1],8*cfg.PSDULength,1,'int8');
waveform = wlanWaveformGenerator(bits,cfg);
```

Transmit the waveform through an additive white Gaussian noise (AWGN) channel with a signal-to-noise ratio (SNR) of 30.

```
snr = 30;
rxWaveform = awgn(waveform,snr);
```

Extract the non-HT Data field from the received waveform.

```
field = 'NonHT-Data';
ind = wlanFieldIndices(cfg,field);
rx = rxWaveform(ind(1):ind(2),:);
```

Recover the frequency-domain signal by OFDM demodulating the time-domain data signal.

```
sym = wlanNonHTOFDMDemodulate(rx,field,cfg);
```

Extract the data subcarriers from the demodulated signal.

```
info = wlanNonHTOFDMInfo(field,cfg);
sym = sym(info.DataIndices,:,:);
```

Recover the PSDU and confirm that it matches the transmitted PSDU.

```
noiseVarEst = 10^(-snr/10);
psdu = wlanNonHTDataBitRecover(sym,noiseVarEst,cfg);
isequal(bits,psdu)
```

```
ans = logical
   1
```

Input Arguments

`field` — Field for which the function returns OFDM information

`'L-LTF'` | `'L-SIG'` | `'NonHT-Data'`

Field for which the function returns OFDM information, specified as one of these values.

- `'L-LTF'` – Return OFDM information for the legacy long training field (L-LTF).
- `'L-SIG'` – Return OFDM information for the legacy signal (L-SIG) field.
- `'NonHT-Data'` – Return OFDM information for the non-HT Data field.

Data Types: `char` | `string`

`bandwidth` — Channel bandwidth of transmission

`'CBW20'` (default) | `'CBW5'` | `'CBW10'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth of PPDU transmission, specified as one of these values.

- `'CBW5'` — Channel bandwidth of 5 MHz
- `'CBW10'` — Channel bandwidth of 10 MHz
- `'CBW20'` — Channel bandwidth of 20 MHz
- `'CBW40'` — Channel bandwidth of 40 MHz for non-HT duplicate
- `'CBW80'` — Channel bandwidth of 80 MHz for non-HT duplicate
- `'CBW160'` — Channel bandwidth of 160 MHz for non-HT duplicate

Data Types: `char` | `string`

`cfg` — Non-HT transmission parameters

`wlanNonHTConfig` object

Non-HT transmission parameters, specified as a `wlanNonHTConfig` object. Because this function supports only OFDM modulation, you must set the `Modulation` property of this input to `'OFDM'`.

`osf` — Oversampling factor

1 (default) | scalar greater than or equal to 1

Oversampling factor, specified as a scalar greater than or equal to 1. The oversampled cyclic prefix length must be an integer number of samples.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`info` — OFDM information

structure

OFDM information, returned as a structure containing these fields.

Name	Values	Description	Data Types
FFTLength	Positive integer	Length of the fast Fourier transform (FFT)	double
SampleRate	Positive scalar	Sample rate of the waveform	double
CPLength	Positive integer	Cyclic prefix length, in samples	double
NumTones	Nonnegative integer	Number of active subcarriers	double
NumSubchannels	Positive integer	Number of 20-MHz subchannels	double
ActiveFrequencyIndices	Column vector of integers in the interval [-FFTLength/2, (FFTLength/2 – 1)]	Indices of active subcarriers. Each element of this field is the index of an active subcarrier, such that the direct current (DC) or null subcarrier is at the center of the frequency band.	double
ActiveFFTIndices	Column vector of integers in the interval [1, FFTLength]	Indices of active subcarriers within the FFT	double
DataIndices	Column vector of integers in the interval [1, NumTones]	Indices of data within the active subcarriers	double
PilotIndices	Column vector of integers in the interval [1, NumTones]	Indices of pilots within the active subcarriers	double

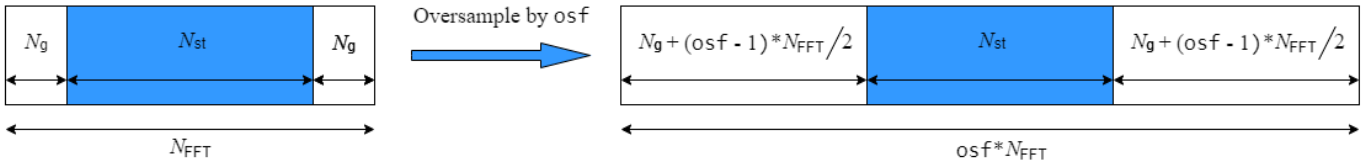
Data Types: struct

Algorithms

FFT-Based Oversampling

An oversampled signal is a signal sampled at a frequency that is higher than the Nyquist rate. WLAN signals maximize occupied bandwidth by using small guardbands, which can pose problems for anti-imaging and anti-aliasing filters. Oversampling increases guardband width relative to the total signal bandwidth, thereby increasing the number of samples in the signal.

This function performs oversampling by using a larger IFFT and zero pad when generating an OFDM waveform. This diagram shows the oversampling process for an OFDM waveform with N_{FFT} subcarriers comprising N_{g} guardband subcarriers on either side of N_{st} occupied bandwidth subcarriers.



Version History

Introduced in R2019a

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`wlanLLTFDemodulate` | `wlanNonHTOFDMDemodulate`

Objects

`wlanNonHTConfig`

wlanPacketDetect

Estimate timing offset of OFDM packet

Syntax

```
startOffset = wlanPacketDetect(rxSig,cbw)
startOffset = wlanPacketDetect(rxSig,cbw,offset)
startOffset = wlanPacketDetect(rxSig,cbw,offset,threshold)
[startOffset,M] = wlanPacketDetect( ___ )
```

Description

`startOffset = wlanPacketDetect(rxSig,cbw)` estimates the timing offset between the start of received signal `rxSig` and the start of the detected preamble for channel bandwidth `cbw`. For more information, see “Packet Detection Processing” on page 3-396.

Note This function supports packet detection of OFDM modulated signals only.

`startOffset = wlanPacketDetect(rxSig,cbw,offset)` specifies the sample at which the function begins autocorrelation processing relative to the start of the received signal.

`startOffset = wlanPacketDetect(rxSig,cbw,offset,threshold)` specifies the threshold that the decision statistic must meet or exceed to detect a packet.

`[startOffset,M] = wlanPacketDetect( ___ )` also returns the decision statistics of the packet detection algorithm for any of the input argument combinations in previous syntaxes.

Examples

Detect 802.11n Packet

Detect a received 802.11n™ packet at a signal-to-noise ratio (SNR) of 20 dB.

Create an HT configuration object and TGn channel object. Generate a transmit waveform.

```
cfgHT = wlanHTConfig;
tgn = wlanTGnChannel('LargeScaleFadingEffect','None');
txWaveform = wlanWaveformGenerator([1;0;0;1],cfgHT);
```

Pass the waveform through a TGn channel with an SNR of 20 dB. Detect the start of the packet.

```
snr = 20;
fadedSig = tgn(txWaveform);
rxWaveform = awgn(fadedSig,snr,0);
startOffset = wlanPacketDetect(rxWaveform,cfgHT.ChannelBandwidth)
```

```
startOffset = 1
```

Detect Delayed 802.11ac Packet

Detect a received 802.11ac™ packet that has been delayed. Specify an offset of 25 to begin the autocorrelation process.

Create an VHT configuration object and generate the transmit waveform.

```
cfgVHT = wlanVHTConfig;

txWaveform = wlanWaveformGenerator([1;0;0;1],cfgVHT,...
    'WindowTransitionTime',0);
```

Delay the signal by appending zeros at the start. Specify an offset of 25 for the beginning of autocorrelation processing. Detect the start of the packet.

```
rxWaveform = [zeros(100,1);txWaveform];
offset = 25;
startOffset = wlanPacketDetect(rxWaveform,cfgVHT.ChannelBandwidth,offset)
```

```
startOffset = 48
```

Calculate the detected packet offset by adding the returned `startOffset` and the input `offset`. This coarse approximation of the packet-start offset is useful for determining where to begin autocorrelation for the first packet and for subsequent packets when a multipacket waveform is transmitted.

```
pktOffset = offset + startOffset
```

```
pktOffset = 73
```

Detect Delayed 802.11a Packet

Detect a received 802.11a™ packet that has been delayed. No channel impairments are added. Set the input offset to 5 and use a threshold setting very close to 1.

Create an non-HT configuration object. Generate the transmit waveform.

```
cfgNonHT = wlanNonHTConfig;

txWaveform = wlanWaveformGenerator([1;0;0;1],cfgNonHT,...
    'WindowTransitionTime',0);
```

Delay the signal by appending zeros at the start. Set an initial offset of 5 and a threshold very close to 1. Detect the delayed packet.

```
rxWaveform = [zeros(20,1);txWaveform];

offset = 5;
threshold = 1-10*eps;
startOffset = wlanPacketDetect(rxWaveform,...
    cfgNonHT.ChannelBandwidth,offset,threshold)
```

```
startOffset = 15
```

Calculate the detected packet offset by adding the returned `startOffset` and the input `offset`.

```
totalOffset = offset + startOffset
```

```
totalOffset = 20
```

Generate WLAN Packet Decision Statistics

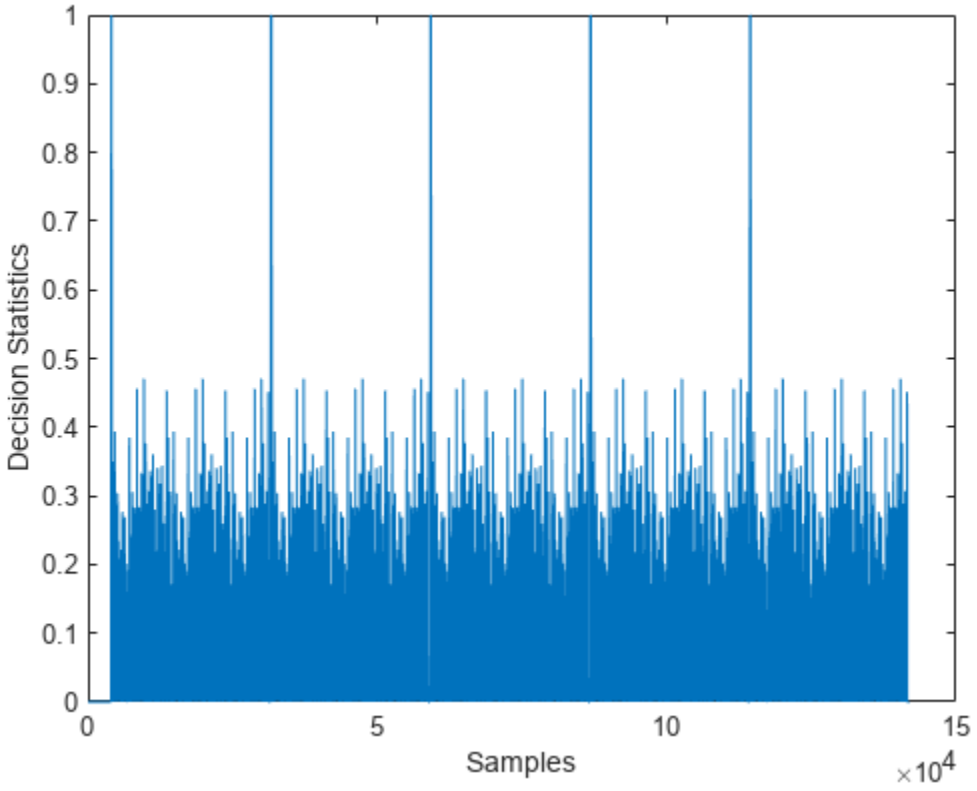
Return the decision statistics of a WLAN waveform that consists of five 802.11a packets.

Create a non-HT configuration object and a five-packet waveform. Delay the waveform by 4000 samples.

```
cfgNonHT = wlanNonHTConfig;
txWaveform = wlanWaveformGenerator([1;0;0;1],cfgNonHT, ...
    'NumPackets',5,'IdleTime',20e-6);
rxWaveform = [zeros(4000,1);txWaveform];
```

Generate and plot packet decision statistics for the waveform. The decision statistics show five peaks, which correspond to the first sample of each packet detected.

```
offset = 0;
threshold = 1;
[startOffset,M] = wlanPacketDetect(rxWaveform,cfgNonHT.ChannelBandwidth,...
    offset,threshold);
plot(M)
xlabel('Samples')
ylabel('Decision Statistics')
```



Input Arguments

`rxSig` — Received time-domain signal

complex-valued matrix

Received time-domain signal, specified as a complex-valued matrix of size N_S-by-N_R matrix. N_S is the number of time-domain samples in the received signal. N_R is the number of receive antennas.

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

`cbw` — Channel bandwidth

`'CBW5'` | `'CBW10'` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'` | `'CBW320'`

Channel bandwidth, specified as one of these values.

- `'CBW5'` – Channel bandwidth of 5 MHz
- `'CBW10'` – Channel bandwidth of 10 MHz
- `'CBW20'` – Channel bandwidth of 20 MHz
- `'CBW40'` – Channel bandwidth of 40 MHz
- `'CBW80'` – Channel bandwidth of 80 MHz
- `'CBW160'` – Channel bandwidth of 160 MHz

- `'CBW320'` – Channel bandwidth of 320 MHz

Data Types: `char` | `string`

`offset` — Starting sample for autocorrelation process

`0` (default) | nonnegative integer

Starting sample for the autocorrelation process, in samples after the start of the received signal, specified as a nonnegative integer. To detect `startOffset` for successive packets in multipacket waveforms, specify this input.

Note Since the packet detection searches forward in time, the function cannot detect the first packet if the value of `offset` indicates a sample after the first “L-STF” on page 3-396.

Data Types: `double`

`threshold` — Decision statistic threshold

`0.5` (default) | scalar in the interval (0, 1]

Decision statistic threshold that must be met or exceeded for the function to detect a packet, specified as a scalar in the interval (0, 1].

Data Types: `double`

Output Arguments

`startOffset` — Timing offset

nonnegative integer | [ ]

Timing offset, in samples, between the start of the received signal and the start of the detected preamble, returned as a nonnegative integer. This value, shifted by `offset`, indicates the detected start of a packet from the first sample of `rxSig`.

- The function returns this output as `[]` if it does not detect a packet, or if the `threshold` input is `1`.
- The function returns this output as `0` if it detects the packet at the first sample of the waveform.

Data Types: `double`

`M` — Decision statistics

real-valued row vector

Decision statistics based on autocorrelation of the input signal, returned as a real-valued row vector of length *N*. The value of *N* depends on the starting location of the autocorrelation process and the number of samples before which the function detects a packet. When `threshold` is 1, the function returns this output as the decision statistics of the full waveform and the `startOffset` output as `[]`.

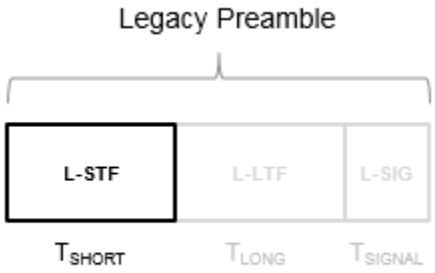
For more information, see “Packet Detection Processing” on page 3-396.

Data Types: `double`

More About

L-STF

The legacy short training field (L-STF) is the first field of the 802.11 OFDM PLCP legacy preamble. The L-STF is a component of VHT, HT, and non-HT PPDUs.



The L-STF duration varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period ($T_{FFT} = 1 / \Delta_F$)	L-STF Duration ($T_{SHORT} = 10 \times T_{FFT} / 4$)
20, 40, 80, 160, and 320	312.5	3.2 µs	8 µs
10	156.25	6.4 µs	16 µs
5	78.125	12.8 µs	32 µs

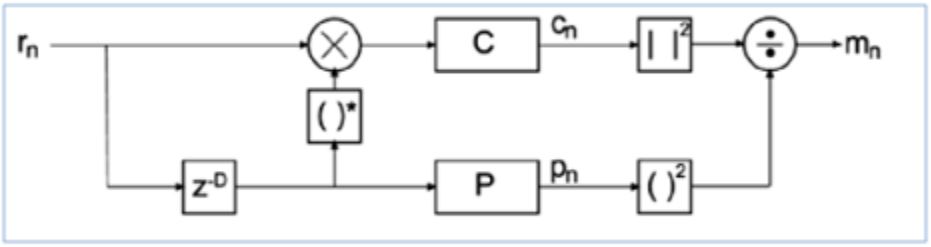
Because the sequence has good correlation properties, it is used for start-of-packet detection, for coarse frequency correction, and for setting the AGC. The sequence uses 12 of the 52 subcarriers that are available per 20 MHz channel bandwidth segment. For 5 MHz, 10 MHz, and 20 MHz bandwidths, the number of channel bandwidths segments is 1.

Algorithms

Packet Detection Processing

The packet detection algorithm is implemented as a double sliding window as described in OFDM Wireless LANs [1], Chapter 2. The autocorrelation of “L-STF” on page 3-396 short training symbols is used to return an estimated packet-start offset. In a robust system, the next stage will refine this estimate with symbol timing detection using the L-LTF.

As shown in the figure, the received signal, r_n, is delayed then correlated in two sliding windows independently. The packet detection processing output provides decision statistics (m_n) of the received waveform.



- Window *C* autocorrelates between the received signal and the delayed version, c_n.

  $$c_n = \sum_{l=1}^{N_R} \sum_{K=0}^{D-1} r_{n+k,l} r^*_{n+k+D,l}$$

- Window *P* calculates the energy received in the autocorrelation window, p_n.

  $$p_n = \sum_{l=1}^{N_R} \sum_{k=0}^{D-1} \left| r_{n+k+D,l} \right|^2$$

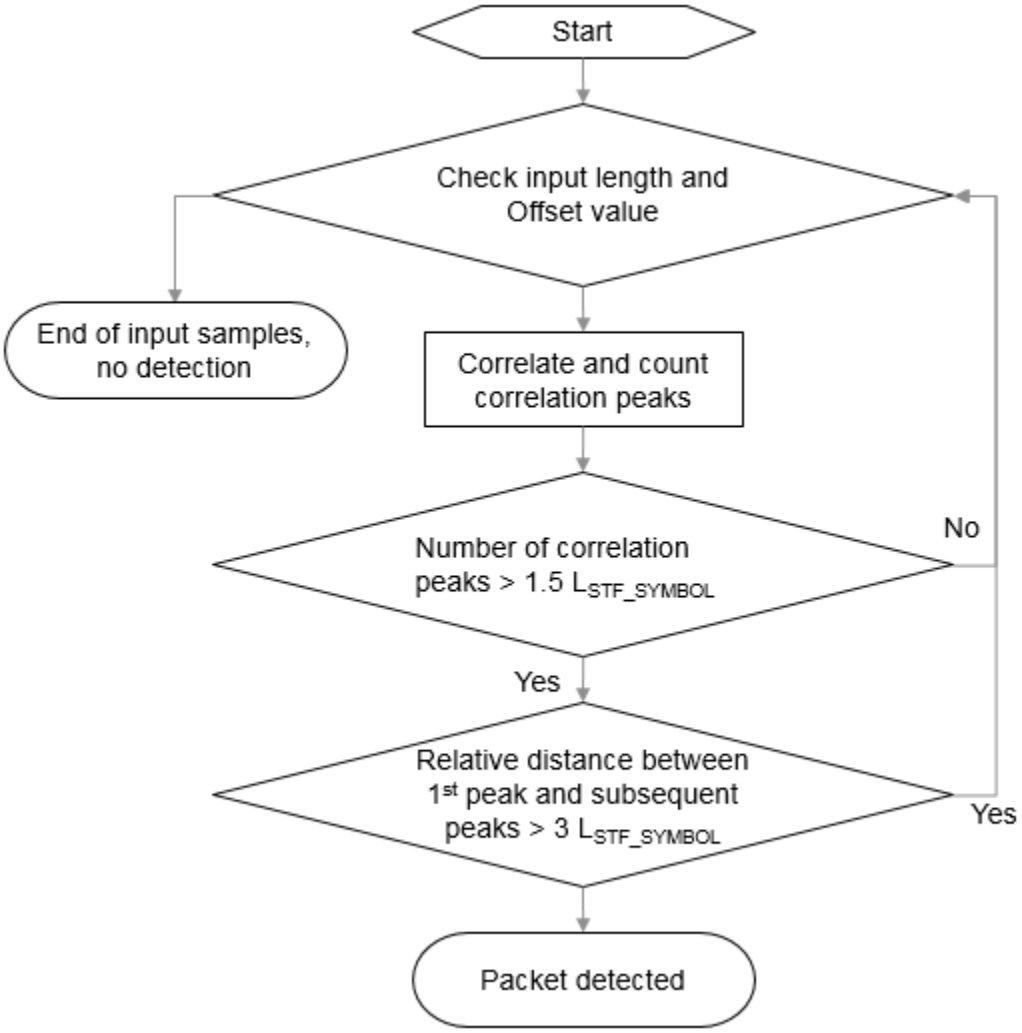
- The decision statistics, m_n, normalize the autocorrelation by p_n so that the decision statistic is not dependent on the absolute received power level.

  $$m_n = \frac{|c_n|^2}{(p_n)^2}$$

  The decision statistics provide visual information resulting from the autocorrelation process that is useful when selecting the appropriate threshold value for the input waveform. The recommended default value of 0.5 for `threshold` favors false detections over missed detections considering a range of SNRs and various antenna configurations.

In the sliding window calculations, *D* is the period of the “L-STF” on page 3-396 short training symbols and N_R is the number of receive antennas.

Packet detection processing follows this flow chart:



L_{STF_SYMBOL} is the length of an "L-STF" on page 3-396 symbol.

Note This function supports packet detection of OFDM modulated signals only.

Version History

Introduced in R2016b

References

[1] Terry, J., and J. Heiskala. *OFDM Wireless LANs: A Theoretical and Practical Guide*. Indianapolis, IN: Sams, 2002.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanCoarseCFOEstimate` | `wlanFieldIndices`

wlanPreHEChannelEstimate

Channel estimation using pre-HE fields

Syntax

```
chEst = wlanPreHEChannelEstimate(demodSym,chEstLLTF,cbw)
chEst = wlanPreHEChannelEstimate( ___ ,span)
```

Description

`chEst = wlanPreHEChannelEstimate(demodSym,chEstLLTF,cbw)` returns the channel estimate at the legacy signal field (L-SIG) using the demodulated L-SIG field symbol, `demodSym`, and the channel estimate at the legacy long training field (L-LTF) , `chEstLLTF`. The channel bandwidth is specified in `cbw`. For more information about these fields, see “L-SIG” on page 3-402 and “L-LTF” on page 3-404.

`chEst = wlanPreHEChannelEstimate( ___ ,span)` also specifies the span of a moving-average filter in addition to the input arguments in the previous syntax. The function uses this filter to perform frequency smoothing.

Examples

Channel Estimation Using Pre-HE Fields

Calculate and plot the channel estimate of an HE SU format channel by using pre-HE fields.

Create an HE SU format configuration object and extract its channel bandwidth. Generate a time-domain waveform for an 802.11ax packet.

```
cfg = wlanHESUConfig;
cbw = cfg.ChannelBandwidth;
txSig = wlanWaveformGenerator([1;0;0;1],cfg);
```

Multiply the transmitted signal by 0.1 + 0.4i and pass it through an AWGN channel with a signal-to-noise ratio of 30 dB.

```
rxSig = awgn((0.1 + 0.4i)*txSig,30);
```

Extract the field indices of the HE SU configuration object, demodulate the L-LTF, and perform L-LTF channel estimation.

```
ind = wlanFieldIndices(cfg);
demodSigLLTF = wlanHEDemodulate(rxSig(ind.LLTF(1):ind.LLTF(2),:),"L-LTF",cfg);
chEstLLTF = wlanLLTFChannelEstimate(demodSigLLTF,cbw);
```

Demodulate the L-SIG and RL-SIG fields.

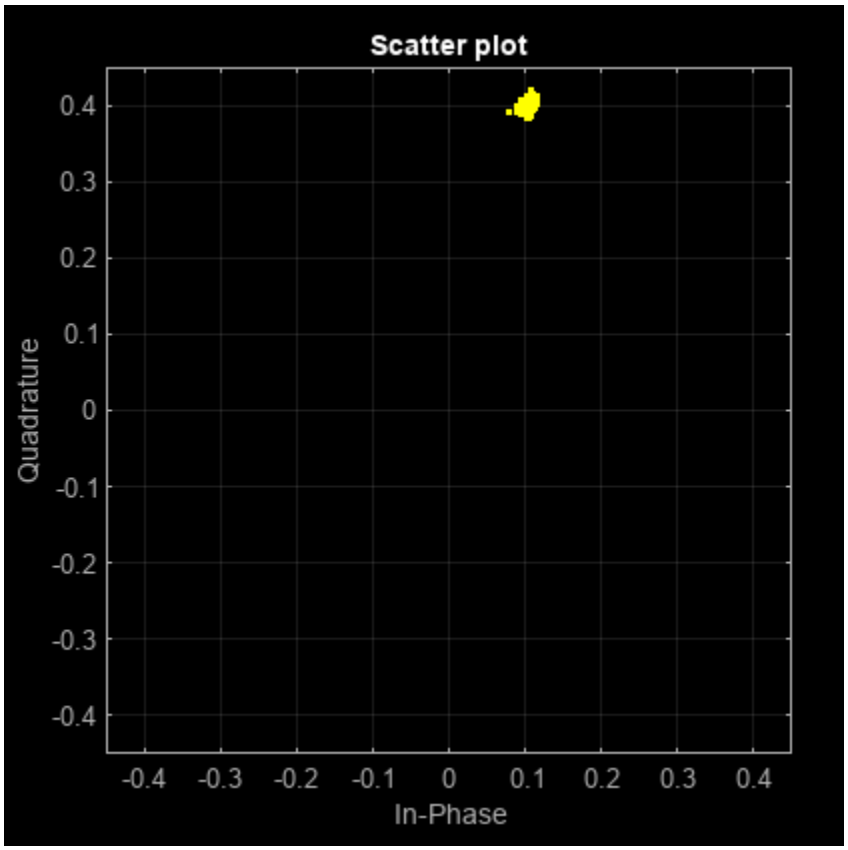
```
demodSigLSIG = wlanHEDemodulate(rxSig(ind.LSIG(1):ind.RLSIG(2),:),"L-SIG",cfg);
```

Using a frequency smoothing span of 3, perform the full channel estimation.

```
est = wlanPreHEChannelEstimate(demodSigLSIG,chEstLLTF,cbw,3);
```

Plot the channel estimate.

```
scatterplot(est)
grid
```



Input Arguments

`demodSym` — Demodulated L-SIG field symbol

3-D array

Demodulated L-SIG field symbol, specified as an N_{ST}-by-N_{SYM}-by-N_R array. N_{ST} is the number of occupied subcarriers. N_{SYM} is the number of OFDM symbols in the L-SIG and repeated legacy signal (RL-SIG) fields. N_R is the number of receive antennas.

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

`chEstLLTF` — L-LTF channel estimate

3-D array

Channel estimate at the legacy long training field, specified as an N_{ST}-by-1-by-N_R array. N_{ST} is the number of occupied subcarriers and N_R is the number of receive antennas.

Data Types: `single` | `double`

Complex Number Support: Yes

cbw — Channel bandwidth
`"CBW20"` | `"CBW40"` | `"CBW80"` | `"CBW160"` | `"CBW320"`

Channel bandwidth, specified as one of these values.

- `"CBW20"` – Channel bandwidth of 20 MHz.
- `"CBW40"` – Channel bandwidth of 40 MHz.
- `"CBW80"` – Channel bandwidth of 80 MHz.
- `"CBW160"` – Channel bandwidth of 160 MHz.
- `"CBW320"` – Channel bandwidth of 320 MHz.

Data Types: `char` | `string`

span — Filter span
positive odd integer

Span of the frequency smoothing filter, specified as a positive odd integer and expressed as a number of subcarriers. The function applies frequency smoothing only when `span` is greater than one. For more information on when to specify this input, see “Frequency Smoothing” on page 3-405.

Data Types: `double`

Output Arguments

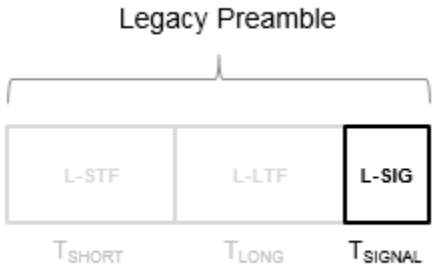
chEst — Channel estimate
3-D array

Channel estimate at all data and pilot subcarriers, returned as an N_{ST}-by-1-by-N_R array. N_{ST} is the number of occupied subcarriers and N_R is the number of receive antennas. The output includes the channel estimates for the extra four subcarriers per each 20 MHz subchannel in the L-SIG field.

More About

L-SIG

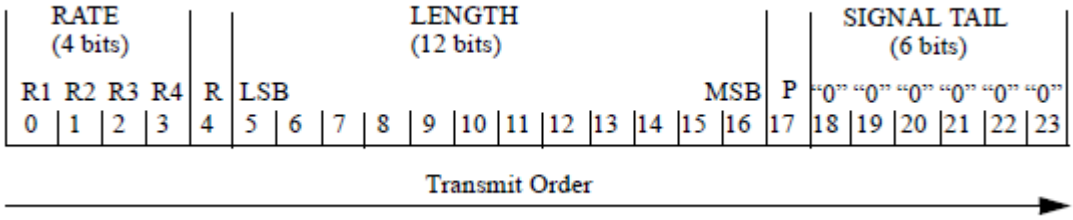
The L-SIG is the third field of the 802.11 OFDM PLCP legacy preamble. This field is a component of EHT, HE, VHT, HT, and non-HT PPDUs. It consists of 24 bits that contain rate, length, and parity information. The L-SIG field is transmitted using BPSK modulation with rate 1/2 binary convolutional coding (BCC).



The L-SIG is one OFDM symbol with a duration that varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period ($T_{FFT} = 1 / \Delta_F$)	Guard Interval (GI) Duration ($T_{GI} = T_{FFT} / 4$)	L-SIG Duration ($T_{SIGNAL} = T_{GI} + T_{FFT}$)
20, 40, 80, and 160	312.5	3.2 µs	0.8 µs	4 µs
10	156.25	6.4 µs	1.6 µs	8 µs
5	78.125	12.8 µs	3.2 µs	16 µs

The L-SIG contains packet information for the received configuration.



- Bits 0 through 3 specify the data rate (modulation and coding rate) for the non-HT format.

Rate (Bits 0–3)	Modulation	Coding Rate (R)	Data Rate (Mb/s)		
			20 MHz Channel Bandwidth	10 MHz Channel Bandwidth	5 MHz Channel Bandwidth
1101	BPSK	1/2	6	3	1.5
1111	BPSK	3/4	9	4.5	2.25
0101	QPSK	1/2	12	6	3
0111	QPSK	3/4	18	9	4.5
1001	16-QAM	1/2	24	12	6
1011	16-QAM	3/4	36	18	9
0001	64-QAM	2/3	48	24	12
0011	64-QAM	3/4	54	27	13.5

  For HT and VHT formats, the L-SIG rate bits are set to `'1 1 0 1'`. Data rate information for HT and VHT formats is signaled in format-specific signaling fields.
- Bit 4 is reserved for future use.
- Bits 5 through 16:
  - For non-HT, specify the data length (amount of data transmitted in octets) as described in table 17-1 and section 10.27.4 IEEE Std 802.11-2020.
  - For HT-mixed, specify the transmission time as described in sections 19.3.9.3.5 and 10.27.4 of IEEE Std 802.11-2020.

- For VHT, specify the transmission time as described in section 21.3.8.2.4 of IEEE Std 802.11-2020.

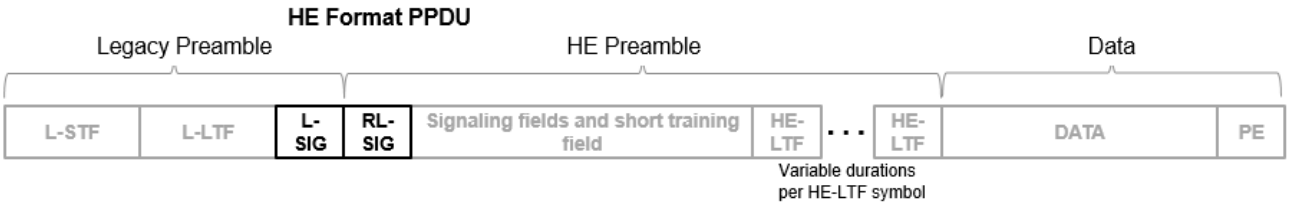
- Bit 17 has the even parity of bits 0 through 16.
- Bits 18 through 23 contain all zeros for the signal tail bits.

Note Signaling fields added for HT (`wlanHTSIG`) and VHT (`wlanVHTSIGA`, `wlanVHTSIGB`) formats provide data rate and configuration information for those formats.

- For the HT-mixed format, section 19.3.9.4.3 of IEEE Std 802.11-2020 describes HT-SIG bit settings.
- For the VHT format, sections 21.3.8.3.3 and 21.3.8.3.6 of IEEE Std 802.11-2020 describe bit settings for the VHT-SIG-A and VHT-SIG-B fields, respectively.

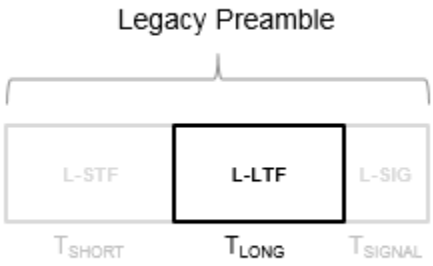
RL-SIG

The RL-SIG is a repeat of the L-SIG, used to distinguish an HE or EHT PPDU from a non-HT, HT, and VHT PPDU.



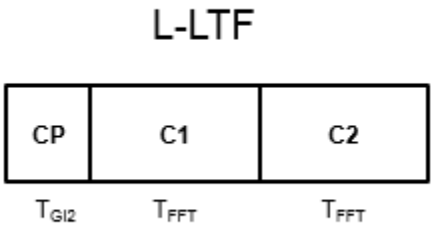
L-LTF

The L-LTF is the second field in the 802.11 OFDM PLCP legacy preamble. The L-LTF is a component of HE, VHT, HT, and non-HT PPDUs.



Channel estimation, fine frequency offset estimation, and fine symbol timing offset estimation rely on the L-LTF.

The L-LTF is composed of a cyclic prefix (CP) followed by two identical long training symbols (C1 and C2). The CP consists of the second half of the long training symbol.



The L-LTF duration varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period ($T_{FFT} = 1 / \Delta_F$)	Cyclic Prefix or Training Symbol Guard Interval (GI2) Duration ($T_{GI2} = T_{FFT} / 2$)	L-LTF Duration ($T_{LONG} = T_{GI2} + 2 \times T_{FFT}$)
20, 40, 80, 160, and 320	312.5	3.2 µs	1.6 µs	8 µs
10	156.25	6.4 µs	3.2 µs	16 µs
5	78.125	12.8 µs	6.4 µs	32 µs

Frequency Smoothing

Frequency smoothing can improve channel estimation by averaging out noise.

Frequency smoothing is recommended only for cases in which a single transmit antenna is used. Frequency smoothing consists of applying a moving-average filter that spans multiple adjacent subcarriers. Channel conditions dictate whether frequency smoothing is beneficial.

- If adjacent subcarriers are highly correlated, frequency smoothing results in significant noise reduction.
- In a highly frequency-selective channel, smoothing can degrade the quality of the channel estimate.

Version History

Introduced in R2022b

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

[2] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements

for High Efficiency WLAN." IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

[3] IEEE P802.11be/D1.5. "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 8: Enhancements for Extremely High Throughput (EHT)." Draft Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`wlanHEMUConfig` | `wlanHERecoveryConfig` | `wlanHESUConfig` | `wlanHETBConfig`

Functions

`wlanHELTFChannelEstimate` | `wlanLLTFChannelEstimate` | `wlanHTLTFChannelEstimate` | `wlanVHTLTFChannelEstimate`

wlanPSDULength

Calculate PSDU length in octets

Syntax

```
PSDULength = wlanPSDULength(cfgPHY,unit,value)
```

Description

`PSDULength = wlanPSDULength(cfgPHY,unit,value)` returns `PSDULength`, the physical layer conformance procedure (PLCP) service data unit (PSDU) length, in octets from the given `value` and the physical layer configuration `cfgPHY`. The `value` can be in terms of PLCP protocol data unit (PPDU) transmission time or number of data symbols, specified by the `unit` input argument.

Examples

Generate Non-HT Waveform with Specified Transmission Time

Create a `wlanNonHTConfig` object, `'cfgPHY'`, and specify the transmission time, `'txTime'`, in microseconds.

```
cfgPHY = wlanNonHTConfig;
txTime = 300;
```

Calculate the PSDU length in octets.

```
psduLength = wlanPSDULength(cfgPHY,'TxTime',txTime)
```

```
psduLength = 207
```

Set the number of bytes carried in the user payload for the configuration object, `'cfgPHY'` to this PSDU length. Create a random PSDU, `'psdu'`, using the calculated PSDU length.

```
cfgPHY.PSDULength = psduLength;
data = randi([0 1],psduLength*8,1);
```

Generate a non-HT waveform using `'cfgPHY'` and `'data'`.

```
waveform = wlanWaveformGenerator(data,cfgPHY);
```

Generate HT Waveform with Specified Number of Data Symbols

Create a `wlanHTConfig` object, `'cfgPHY'`, and specify the number of data symbols, `'numDataSymbols'`.

```
cfgPHY = wlanNonHTConfig;
numDataSymbols = 200;
```

Calculate the PSDU length in octets.

```
psduLength = wlanPSDULength(cfgPHY,'NumDataSymbols',numDataSymbols)
```

```
psduLength = 597
```

Set the number of bytes carried in the user payload for the configuration object, `'cfgPHY'`, to this PSDU length. Create a random PSDU, `'psdu'`, with the calculated PSDU length.

```
cfgPHY.PSDULength = psduLength;
data = randi([0 1],psduLength*8,1);
```

Generate a non-HT waveform using `'cfgPHY'` and `'data'`.

```
waveform = wlanWaveformGenerator(data,cfgPHY);
```

Input Arguments

`cfgPHY` — PHY format configuration

`wlanHESUConfig` object | `wlanVHTConfig` object | `wlanHTConfig` object | `wlanNonHTConfig` object

PHY format configuration, specified as a `wlanHESUConfig`, `wlanVHTConfig`, `wlanHTConfig`, or `wlanNonHTConfig` object. This object defines a PHY format configuration and its applicable properties.

`unit` — Units of argument `value`

`'TxTime'` | `'NumDataSymbols'`

Units of argument `value`, specified as `'TxTime'` or `'NumDataSymbols'`. This value indicates the units of `value` from which the PSDU length is calculated.

Data Types: `char` | `string`

`value` — Value from which PSDU length is calculated

numeric scalar

Value from which PSDU length is calculated, specified as a numeric scalar. Input argument `unit` specifies the unit of `value`. This table describes how the function interprets `value` based on `unit`.

`unit` Value	`value` Description
`'TxTime'`	Scalar number specifying PPDU transmission time in microseconds
`'NumDataSymbols'`	Scalar number specifying the number of symbols in the `'Data'` field of PPDU

Data Types: `double`

Output Arguments

`PSDULength` — Length of PSDU

numeric scalar

Length of PSDU, in octets, returned as a numeric scalar. This value returns the maximum PSDU length that fits into the specified value of `'TxTime'` or `'NumDataSymbols'`.

Data Types: `double`

Version History

Introduced in R2019b

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`wlanMSDULengths` | `wlanAPEPLength`

wlanReferenceSymbols

Find reference symbols of constellation diagram

Syntax

```
refSym = wlanReferenceSymbols(mod)
refSym = wlanReferenceSymbols(mod,phase)

refSym = wlanReferenceSymbols(cfg)
refSym = wlanReferenceSymbols(cfg,userNumber)
```

Description

`refSym = wlanReferenceSymbols(mod)` returns the reference symbols of the constellation diagram for the specified modulation scheme.

`refSym = wlanReferenceSymbols(mod,phase)` returns reference symbols with counterclockwise rotation for the specified modulation scheme.

`refSym = wlanReferenceSymbols(cfg)` returns the reference symbols used in the data field of a single-user (SU) transmission or a recovered high-efficiency (HE) transmission parameterized by the configuration object `cfg`.

`refSym = wlanReferenceSymbols(cfg,userNumber)` returns the reference symbols used in the data field for the user specified by `userNumber` in a multiuser (MU) transmission specified by MU configuration object `cfg`.

Examples

Plot Noisy QPSK Constellation with Reference Symbols

Fine reference symbols for a quadrature phase-shift keying (QPSK) modulation scheme and plot the resulting constellation.

Generate noisy QPSK symbols.

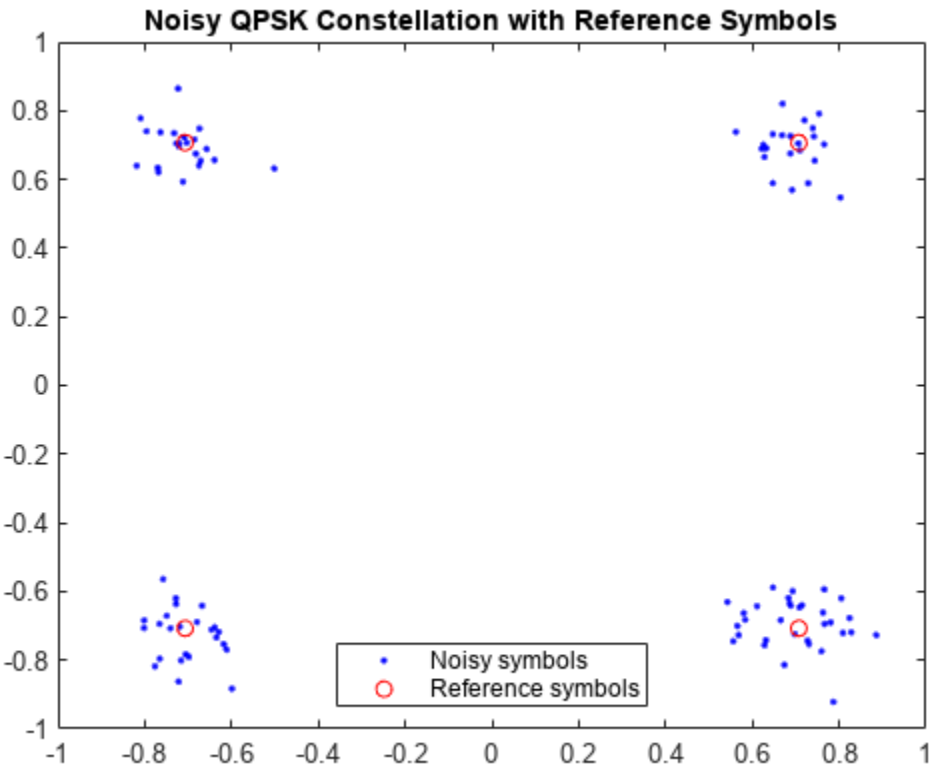
```
sym =  awgn(qammod(randi([0 3],100,1),4)/sqrt(2),20);
```

Find the reference symbols.

```
refSym =  wlanReferenceSymbols('QPSK');
```

Plot the constellation diagram.

```
figure;
plot(sym,'b.');
hold on;
plot(refSym,'ro');
title('Noisy QPSK Constellation with Reference Symbols');
legend('Noisy symbols','Reference symbols','Location','South');
```



Find Reference Symbols with Counterclockwise Rotation

Find reference symbols for a chosen modulation scheme and counterclockwise rotation.

Reference Symbols for $\frac{\pi}{2}$-BPSK

Specify a $\frac{\pi}{2}$-BPSK modulation scheme and a counterclockwise rotation of $\frac{\pi}{6}$.

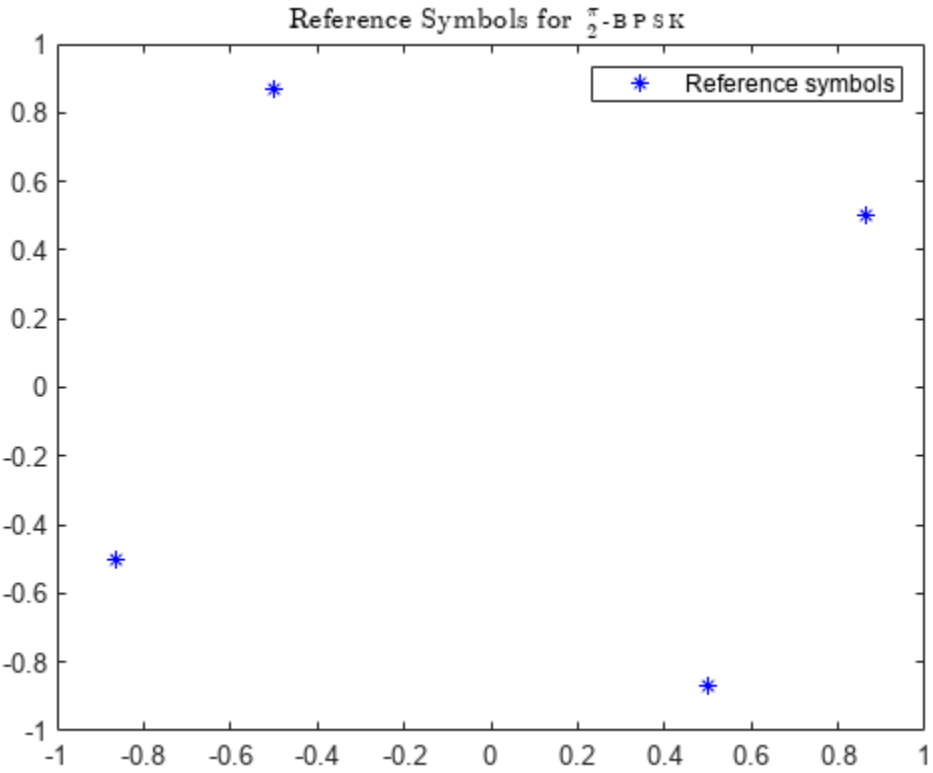
```
mod = 'pi/2-BPSK';
phase =  pi/6;
```

Find the reference symbols for the chosen modulation and rotation.

```
refSym = wlanReferenceSymbols(mod,phase);
```

Display the reference symbols on a constellation diagram.

```
figure;
plot(refSym,'b*');
hold on;
title('Reference Symbols for $\frac{\pi}{2}$-BPSK','Interpreter','latex');
legend('Reference symbols');
```



Reference Symbols for $\frac{\pi}{2}$-16-QAM

Specify a $\frac{\pi}{2}$-16-QAM modulation scheme and a counterclockwise rotation of $\frac{\pi}{3}$.

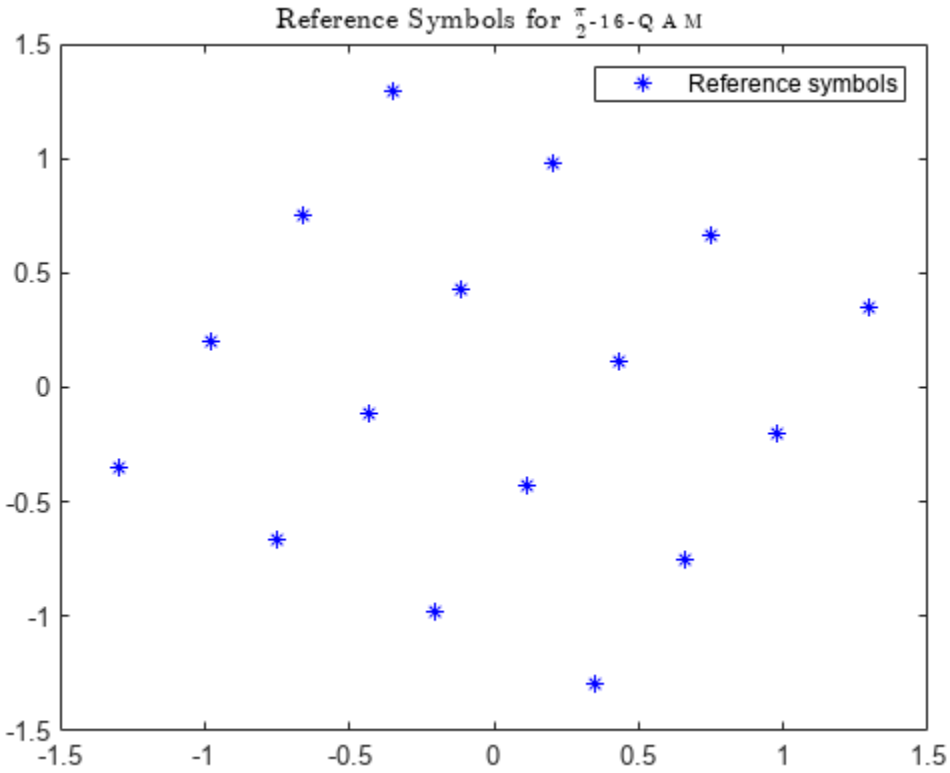
```
mod = 'pi/2-16QAM';
phase =  pi/3;
```

Find the reference symbols for the chosen modulation and rotation.

```
refSym = wlanReferenceSymbols(mod,phase);
```

Display the reference symbols on a constellation diagram.

```
figure;
plot(refSym,'b*');
hold on;
title('Reference Symbols for $\frac{\pi}{2}$-16-QAM','Interpreter','latex');
legend('Reference symbols');
```



Plot Reference Constellation for User in VHT Configuration

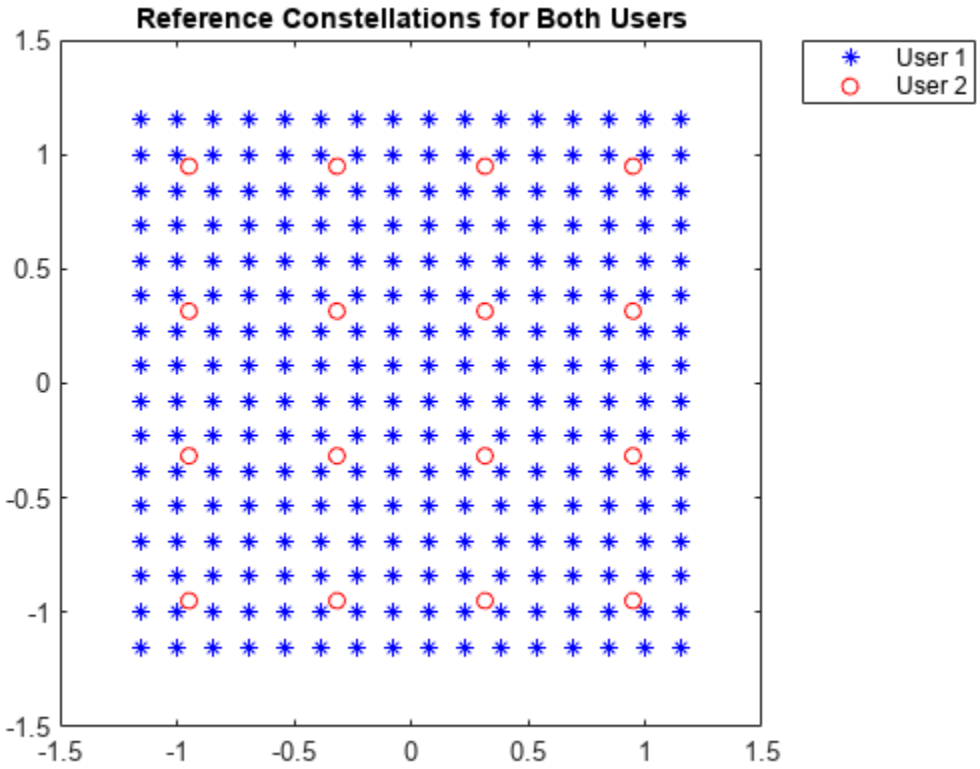
Plot the reference constellation for a user in a very-high-throughput (VHT) multiuser configuration.

Create a VHT-format configuration object, setting channel bandwidth, number of users, group identifier, number of transmit antennas, number of space-time streams, and modulation and coding schemes.

```
cfg = wlanVHTConfig('ChannelBandwidth','CBW20','NumUsers',2,'GroupID',2, ...
    'NumTransmitAntennas',2,'NumSpaceTimeStreams',[1 1],'MCS',[8 4]);
```

Find the reference symbols for both users and plot the constellations.

```
refSym1 = wlanReferenceSymbols(cfg,1);
refSym2 = wlanReferenceSymbols(cfg,2);
figure;
plot(refSym1,'b*'); hold on
plot(refSym2,'ro');
title('Reference Constellations for Both Users')
legend('User 1','User 2','Location','bestoutside');
```



Input Arguments

mod — Modulation scheme

`'BPSK'` | `'pi/2-BPSK'` | `'QPSK'` | `'pi/2-QPSK'` | `'16QAM'` | `'pi/2-16QAM'` | `'64QAM'` | `'pi/2-64QAM'` | `'256QAM'` | `'1024QAM'` | `'4096QAM'`

Modulation scheme, specified as one of these values.

- `'BPSK'` — Binary phase-shift keying (BPSK)
- `'pi/2-BPSK'` — π/2-BPSK
- `'QPSK'` — Quadrature phase-shift keying (QPSK)
- `'pi/2-QPSK'` — π/2-QPSK
- `'16QAM'` — 16-point quadrature amplitude modulation (16-QAM)
- `'pi/2-16QAM'` — π/2-16-QAM
- `'64QAM'` — 64-QAM
- `'pi/2-64QAM'` — π/2-64-QAM
- `'256QAM'` — 256-QAM
- `'1024QAM'` — 1024-QAM
- `'4096QAM'` — 4096-QAM

Data Types: `char` | `string`

`phase` — Counterclockwise rotation

real-valued scalar (default) | real-valued row vector

Counterclockwise rotation, in radians, specified as a real-valued scalar or real-valued row vector. To return reference symbols for different phases, specify `phase` as a row vector in which each element represents a chosen phase.

Data Types: `double`

`cfg` — PHY format configuration

`wlanHESUConfig` object | `wlanHEMUConfig` object | `wlanHETBConfig` object | `wlanHERecoveryConfig` object | `wlanDMGConfig` object | `wlanS1GConfig` object | `wlanVHTConfig` object | `wlanHTConfig` object | `wlanNonHTConfig` object

Physical layer (PHY) format configuration, specified as one of these objects: `wlanHESUConfig`, `wlanHEMUConfig`, `wlanHETBConfig`, `wlanHERecoveryConfig`, `wlanDMGConfig`, `wlanS1GConfig`, `wlanVHTConfig`, `wlanHTConfig`, or `wlanNonHTConfig`.

`userNumber` — Number assigned to user of interest

positive integer

Number assigned to user of interest, specified as a positive integer in the interval [1, N_u], where N_u is the number users in the transmission.

This argument is required when you specify the `cfg` input as an object of type `wlanHEMUConfig`, `wlanS1GConfig`, or `wlanVHTConfig`.

If `cfg` is a `wlanHEMUConfig` object, N_u is equal to the number of elements in the value of its `User` property. If `cfg` is a `wlanS1GConfig` or `wlanVHTConfig` object, N_u is equal to the value of its `NumUsers`.

Dependencies

This argument applies only when the `cfg` input is an object of type `wlanHEMUConfig`, `wlanS1GConfig`, or `wlanVHTConfig`.

Data Types: `double`

Output Arguments

`refSym` — Reference symbols of constellation diagram

complex-valued column vector

Reference symbols of constellation diagram, returned as a complex-valued column vector.

Data Types: `double`
Complex Number Support: Yes

Version History

Introduced in R2019a

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`wlanClosestReferenceSymbol`

Objects

`comm.EVM`

wlanS1GDemodulate

Demodulate fields of S1G waveform

Syntax

```
sym = wlanS1GDemodulate(rx,field,cfg)
sym = wlanS1GDemodulate( ___ ,'OFDMSymbolOffset',symOffset)
```

Description

`sym = wlanS1GDemodulate(rx,field,cfg)` recovers a demodulated frequency-domain signal by orthogonal frequency-division multiplexing (OFDM) demodulating received time-domain signal `rx`. The function demodulates `rx` by using S1G transmission parameters `cfg` and signal field value `field`.

`sym = wlanS1GDemodulate( ___ ,'OFDMSymbolOffset',symOffset)` specifies the OFDM symbol sampling offset as a fraction of the cyclic prefix length.

Examples

Demodulate S1G-SIG Field and Get OFDM Information

Perform OFDM demodulation on the S1G-SIG field and extract the data and pilot subcarriers.

Generate a WLAN waveform for an S1G transmission.

```
cfg = wlanS1GConfig;
bits = [1; 0; 0; 1];
waveform = wlanWaveformGenerator(bits,cfg);
```

Obtain the field indices and extract the S1G-SIG field.

```
ind = wlanFieldIndices(cfg);
rx = waveform(ind.S1GSIG(1):ind.S1GSIG(2),:);
```

Perform OFDM demodulation on the S1G-SIG field.

```
sym = wlanS1GDemodulate(rx,'S1G-SIG',cfg);
```

Get the OFDM information, then extract the data and pilot subcarriers.

```
info = wlanS1GOFDMInfo('S1G-SIG',cfg);
data = sym(info.DataIndices,:,:);
pilots =  sym(info.PilotIndices,:,:);
```

Demodulate S1G-Data field for OFDM Symbol Offset

Perform OFDM demodulation on the S1G-Data field for an OFDM symbol offset, specified as a fraction of the cyclic prefix length.

Generate a WLAN waveform for an S1G transmission with the specified modulation and coding scheme (MCS).

```
cfg = wlanS1GConfig('MCS',7);
bits = [0; 0; 0; 1];
waveform = wlanWaveformGenerator(bits,cfg);
```

Obtain the field indices and extract the S1G-Data field.

```
ind = wlanFieldIndices(cfg);
rx = waveform(ind.S1GData(1):ind.S1GData(2),:);
```

Perform OFDM demodulation on the S1G-Data field, specifying an OFDM symbol offset of `0`.

```
field = 'S1G-Data';
sym = wlanS1GDemodulate(rx,field,cfg,'OFDMSymbolOffset',0);
```

Input Arguments

`rx` — Received time-domain signal

complex-valued matrix

Received time-domain signal, specified as a complex-valued matrix of size N_s-by-N_r.

- N_s is the number of time-domain samples. If N_s is not an integer multiple of the OFDM symbol length, L_s, for the specified field,then the function ignores the remaining $\mathtt{mod}(N_s,L_s)$ symbols.
- N_r is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

`field` — Field to be demodulated

`'S1G-LTF1'` | `'S1G-SIG'` | `'S1G-LTF2N'` | `'S1G-SIG-A'` | `'S1G-SIG-B'` | `'S1G-DLTF'` | `'S1G-Data'`

Field to be demodulated, specified as one of these values.

- `'S1G-LTF1'` – Demodulate the first S1G long training field (S1G-LTF1).
- `'S1G-SIG'` – Demodulate the S1G signaling (S1G-SIG) field.
- `'S1G-LTF2N'` – Demodulate the subsequent S1G long training fields (S1G-LTF2N).
- `'S1G-SIG-A'` – Demodulate the S1G signal A (S1G-SIG-A) field.
- `'S1G-SIG-B'` – Demodulate the S1G signal B (S1G-SIG-B) field.
- `'S1G-Data'` – Demodulate the S1G-Data field.

Data Types: `char` | `string`

`cfg` — PHY format configuration

`wlanS1GConfig` object

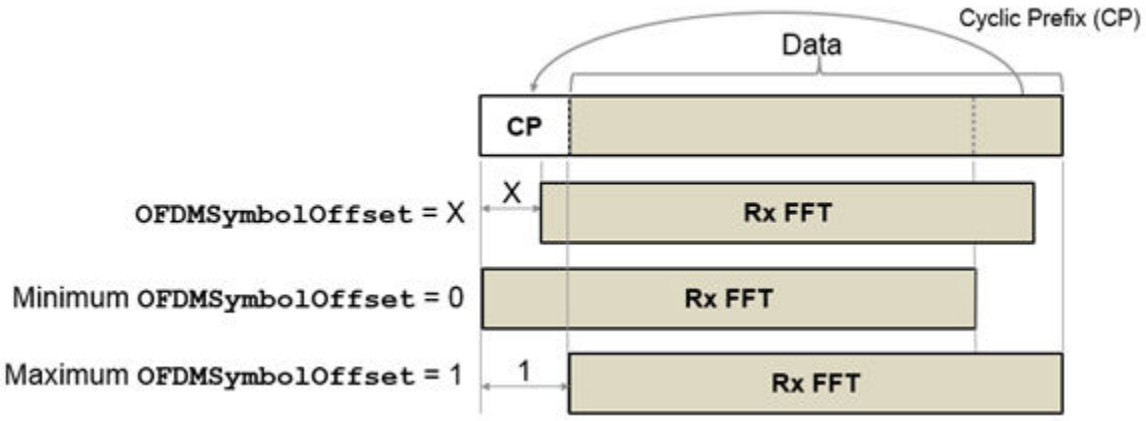
Physical layer (PHY) format configuration, specified as a `wlanS1GConfig` object.

`symOffset` — OFDM symbol sampling offset

`0.75` (default) | scalar in the interval [0, 1]

OFDM symbol sampling offset, as a fraction of the cyclic prefix length, specified as a scalar in the interval [0, 1].

The value that you specify indicates the start location for OFDM demodulation relative to the beginning of the cyclic prefix.



Example: `0.45`

Data Types: `double`

Output Arguments

`sym` — Demodulated frequency-domain signal

complex-valued array

Demodulated frequency-domain signal, returned as a complex-valued array of size N_{sc}-by-N_{sym}-by-N_r.

- N_{sc} is the number of active occupied subcarriers in the demodulated field.
- N_{sym} is the number of OFDM symbols.
- N_r is the number of receive antennas.

Data Types: `double`

Version History

Introduced in R2019a

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`wlanHEDemodulate` | `wlanDMGOFDMDemodulate` | `wlanS1GOFDMInfo`

Objects
`wlanS1GConfig`

wlanS1GOFDMInfo

OFDM Information for S1G transmission

Syntax

```
info = wlanS1GOFDMInfo(field,cfg)
info = wlanS1GOFDMInfo(field,cfg,OversamplingFactor=osf)
```

Description

`info = wlanS1GOFDMInfo(field,cfg)` returns `info`, a structure containing orthogonal frequency-division multiplexing (OFDM) information for the input field of a sub-1-Ghz (S1G) transmission parameterized by configuration object `cfg`.

`info = wlanS1GOFDMInfo(field,cfg,OversamplingFactor=osf)` returns OFDM information for the specified oversampling factor. For more information about oversampling, see “FFT-Based Oversampling” on page 3-423.

Examples

Demodulate S1G-SIG Field and Get OFDM Information

Perform OFDM demodulation on the S1G-SIG field and extract the data and pilot subcarriers.

Generate a WLAN waveform for an S1G transmission.

```
cfg = wlanS1GConfig;
bits = [1; 0; 0; 1];
waveform = wlanWaveformGenerator(bits,cfg);
```

Obtain the field indices and extract the S1G-SIG field.

```
ind = wlanFieldIndices(cfg);
rx = waveform(ind.S1GSIG(1):ind.S1GSIG(2),:);
```

Perform OFDM demodulation on the S1G-SIG field.

```
sym = wlanS1GDemodulate(rx,'S1G-SIG',cfg);
```

Get the OFDM information, then extract the data and pilot subcarriers.

```
info = wlanS1GOFDMInfo('S1G-SIG',cfg);
data = sym(info.DataIndices,:,:);
pilots =  sym(info.PilotIndices,:,:);
```

Get OFDM Information for S1G-Data Field

Get OFDM information for the S1G-Data field in a transmission with specified channel bandwidth and oversampling factor.

Create a WLAN S1G format configuration object, specifying the channel bandwidth.

```
cfg = wlanS1GConfig('ChannelBandwidth','CBW1');
```

Specify an oversampling factor.

```
osf = 1.5;
```

Return and display the OFDM information for the S1G-Data field.

```
info = wlanS1GOFDMInfo('S1G-Data',cfg,OversamplingFactor=osf);
disp(info)
```

```
                FFTLength: 48
               SampleRate: 1500000
                 CPLength: 12
            NumSubchannels: 1
                 NumTones: 26
    ActiveFrequencyIndices: [26x1 double]
          ActiveFFTIndices: [26x1 double]
              DataIndices: [24x1 double]
             PilotIndices: [2x1 double]
```

Input Arguments

`field` — Field for which to return OFDM information

`'S1G-LTF1'` | `'S1G-SIG'` | `'S1G-LTF2N'` | `'S1G-SIG-A'` | `'S1G-SIG-B'` | `'S1G-DLTF'` | `'S1G-Data'`

Field for which to return OFDM information, specified as one of these values.

- `'S1G-LTF1'` – Return OFDM information for the first S1G long training field (S1G-LTF1).
- `'S1G-SIG'` – Return OFDM information for the S1G signaling (S1G-SIG) field.
- `'S1G-LTF2N'` – Return OFDM information for the subsequent S1G long training fields (S1G-LTF2N).
- `'S1G-SIG-A'` – Return OFDM information for the S1G signal A (S1G-SIG-A) field.
- `'S1G-SIG-B'` – Return OFDM information for the S1G signal B (S1G-SIG-B) field.
- `'S1G-DLTF'` – Return OFDM information for the S1G beamformed LTF (D-LTF).
- `'S1G-Data'` – Return OFDM information for the S1G-Data field.

Data Types: `char` | `string`

`cfg` — PHY format configuration

`wlanS1GConfig` object

Physical layer (PHY) format configuration, specified as a `wlanS1GConfig` object.

`osf` — Oversampling factor

1 (default) | scalar greater than or equal to 1

Oversampling factor, specified as a scalar greater than or equal to 1. The oversampled cyclic prefix length must be an integer number of samples.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`info` — OFDM information

structure

OFDM information, returned as a structure containing these fields.

Name	Values	Description	Data Types
`FFTLength`	Positive integer	Length of the fast Fourier transform (FFT)	`double`
`SampleRate`	Positive scalar	Sample rate of the waveform	`double`
`CPLength`	Positive integer	Cyclic prefix length, in samples	`double`
`NumTones`	Nonnegative integer	Number of active subcarriers	`double`
`NumSubchannels`	Positive integer	Number of 20-MHz subchannels	`double`
`ActiveFrequencyIndices`	Column vector of integers in the interval [-`FFTLength`/2, (`FFTLength`/2 – 1)]	Indices of active subcarriers. Each element of this field is the index of an active subcarrier, such that the direct current (DC) or null subcarrier is at the center of the frequency band.	`double`
`ActiveFFTIndices`	Column vector of integers in the interval [1, `FFTLength`]	Indices of active subcarriers within the FFT	`double`
`DataIndices`	Column vector of integers in the interval [1, `NumTones`]	Indices of data within the active subcarriers	`double`
`PilotIndices`	Column vector of integers in the interval [1, `NumTones`]	Indices of pilots within the active subcarriers	`double`

Data Types: `struct`

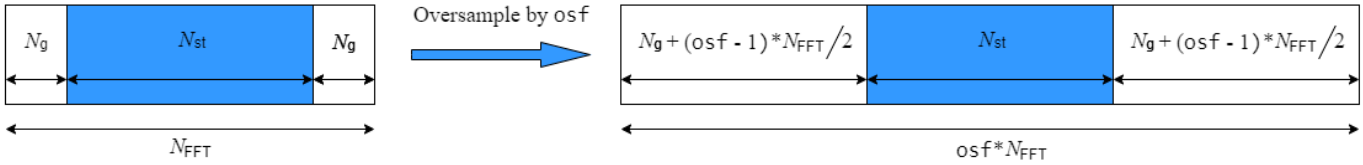
Algorithms

FFT-Based Oversampling

An oversampled signal is a signal sampled at a frequency that is higher than the Nyquist rate. WLAN signals maximize occupied bandwidth by using small guardbands, which can pose problems for anti-imaging and anti-aliasing filters. Oversampling increases guardband width relative to the total signal bandwidth, thereby increasing the number of samples in the signal.

This function performs oversampling by using a larger IFFT and zero pad when generating an OFDM waveform. This diagram shows the oversampling process for an OFDM waveform with N_{FFT}

subcarriers comprising N_g guardband subcarriers on either side of N_{st} occupied bandwidth subcarriers.



Version History

Introduced in R2019a

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`wlanS1GDemodulate`

Objects

`wlanS1GConfig`

wlanSampleRate

Nominal sample rate

Syntax

```
fs = wlanSampleRate(cfgFormat)
fs = wlanSampleRate(cbw)
fs = wlanSampleRate( ___ ,OversamplingFactor=osf)
```

Description

`fs = wlanSampleRate(cfgFormat)` returns the nominal sample rate for the specified WLAN transmission parameters.

`fs = wlanSampleRate(cbw)` returns the nominal sample rate for the specified channel bandwidth.

`fs = wlanSampleRate( ___ ,OversamplingFactor=osf)` returns the nominal sample for the specified oversampling factor. For more information about oversampling, see “FFT-Based Oversampling” on page 3-427.

Examples

Sample Rate for WUR Transmission

Get the sample rate for a WLAN wake-up radio (WUR) transmission.

```
cfgWUR = wlanWURConfig;
fs = wlanSampleRate(cfgWUR)
```

```
fs = 20000000
```

Sample Rate for 160 MHz WLAN Transmission

Get the sample rate for WLAN transmissions with specified bandwidths.

Specify a channel bandwidth of 160 MHz

```
cbw = 'CBW160';
```

Return the sample rate.

```
fs = wlanSampleRate(cbw)
```

```
fs = 160000000
```

Sample Rate for Oversampled HE TB Transmission

Get the sample rate for an oversampled high-efficiency trigger-based (HE TB) WLAN transmission.

```
cfgHETB = wlanHETBConfig;
osf = 2.5;
fs = wlanSampleRate(cfgHETB,Oversamplingfactor=osf)
```

```
fs = 50000000
```

Input Arguments

`cfgFormat` — WLAN transmission parameters

`wlanHESUConfig` object | `wlanHEMUConfig` object | `wlanHETBConfig` object | `wlanHERecoveryConfig` object | `wlanWURConfig` object | `wlanDMGConfig` object | `wlanS1GConfig` object | `wlanVHTConfig` object | `wlanHTConfig` object | `wlanNonHTConfig` object

WLAN transmission parameters, specified as one of these configuration objects: `wlanHESUConfig`, `wlanHEMUConfig`, `wlanHETBConfig`, `wlanHERecoveryConfig`, `wlanWURConfig`, `wlanDMGConfig`, `wlanS1GConfig`, `wlanVHTConfig`, `wlanHTConfig`, or `wlanNonHTConfig`.

`cbw` — Channel bandwidth

`'CBW1'` | `'CBW2'` | `'CBW4'` | `'CBW5'` | `'CBW8'` | `'CBW10'` | `'CBW16'` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'` | `'CBW320'`

Channel bandwidth, specified as one of these values.

- `'CBW1'` — Channel bandwidth of 1 MHz
- `'CBW2'` — Channel bandwidth of 2 MHz
- `'CBW4'` — Channel bandwidth of 4 MHz
- `'CBW5'` — Channel bandwidth of 5 MHz
- `'CBW8'` — Channel bandwidth of 8 MHz
- `'CBW10'` — Channel bandwidth of 10 MHz
- `'CBW16'` — Channel bandwidth of 16 MHz
- `'CBW20'` — Channel bandwidth of 20 MHz
- `'CBW40'` — Channel bandwidth of 40 MHz
- `'CBW80'` — Channel bandwidth of 80 MHz
- `'CBW160'` — Channel bandwidth of 160 MHz
- `'CBW320'` — Channel bandwidth of 320 MHz

Data Types: `char` | `string`

`osf` — Oversampling factor

1 (default) | scalar greater than or equal to 1

Oversampling factor, specified as a scalar greater than or equal to 1.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`fs` — Sample rate

scalar

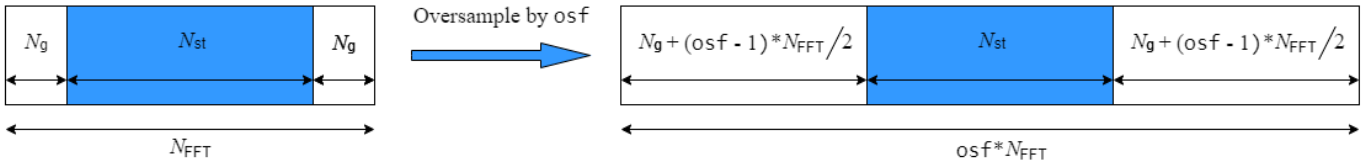
Sample rate, in samples per second, returned as a scalar.

Algorithms

FFT-Based Oversampling

An oversampled signal is a signal sampled at a frequency that is higher than the Nyquist rate. WLAN signals maximize occupied bandwidth by using small guardbands, which can pose problems for anti-imaging and anti-aliasing filters. Oversampling increases guardband width relative to the total signal bandwidth, thereby increasing the number of samples in the signal.

This function performs oversampling by using a larger IFFT and zero pad when generating an OFDM waveform. This diagram shows the oversampling process for an OFDM waveform with N_{FFT} subcarriers comprising N_g guardband subcarriers on either side of N_{st} occupied bandwidth subcarriers.



Version History

Introduced in R2017b

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`wlanHESUConfig` | `wlanHEMUConfig` | `wlanHETBConfig` | `wlanHERecoveryConfig` | `wlanWURConfig` | `wlanDMGConfig` | `wlanS1GConfig` | `wlanVHTConfig` | `wlanHTConfig` | `wlanNonHTConfig`

wlanScramble

Scramble and descramble binary input sequence

Syntax

```
y = wlanScramble(bits,scramInit)
```

Description

`y = wlanScramble(bits,scramInit)` scrambles or descrambles the binary input `bits` for the specified initial scramble state, using a 127-length frame-synchronous scrambler. The frame-synchronous scrambler uses the generator polynomial defined in sections 17.3.5.5 and 20.3.9 of [1]. The transmitter and receiver use the same scrambler to scramble bits at the transmitter and descramble bits at the receiver, respectively.

Examples

Scramble and Descramble bits

Create the scrambler initialization and the input sequence of random bits.

```
scramInit = 93;
bits = randi([0,1],1000,1);
```

Scramble and descramble the bits by using the scrambler initialization.

```
scrambledData = wlanScramble(bits,scramInit);
descrambledData = wlanScramble(scrambledData,scramInit);
```

Verify that the descrambled data matches the original data.

```
isequal(bits,descrambledData)
```

```
ans = logical
   1
```

Input Arguments

`bits` — Input sequence

binary-valued column vector | binary-valued matrix

Input sequence to be scrambled, specified as a binary-valued column vector or matrix.

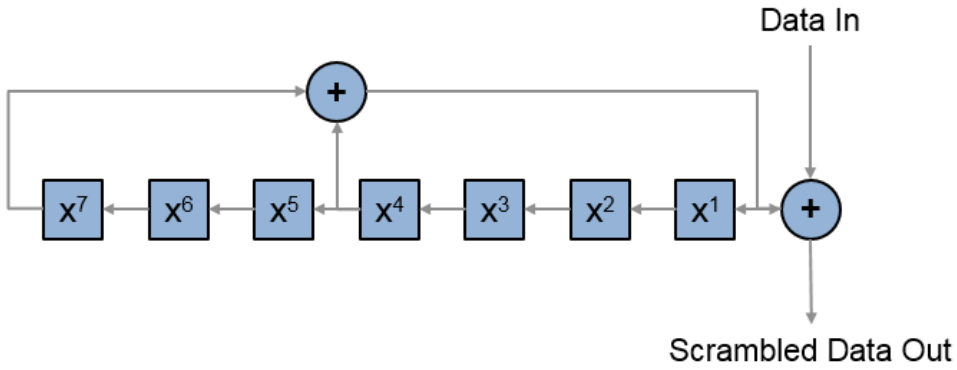
Data Types: `double` | `int8`

`scramInit` — Initial scrambler state

integer in the interval [1, 127] | binary-valued column vector

Initial scrambler state, specified as an integer in the interval [1, 127], or the corresponding binary-valued column vector of length 7.

Section 17.3.5.5 of [1] specifies the scrambling and descrambling process applied to the transmitted data. The header and data fields that follow the scrambler initialization field (including data padding bits) are scrambled by XORing each bit with a length-127 periodic sequence generated by the polynomial $S(x) = x^7+x^4+1$. The octets of the PSDU are placed into a bit stream, and, within each octet, bit 0 (LSB) is first and bit 7 (MSB) is last. This figure demonstrates the sequence generation and XOR operation.



Conversion from integer to bits uses left-MSB orientation. For example, initializing the scrambler with decimal `1`, the bits map to these elements.

Element	X^7	X^6	X^5	X^4	X^3	X^2	X^1
Bit Value	0	0	0	0	0	0	1

To generate the bit stream equivalent to a decimal, use the `int2bit` function. For example, for decimal `1`:

```
int2bit(1,7)'
ans =

     0     0     0     0     0     0     1
```

Example: `[1; 0; 1; 1; 1; 0; 1]` conveys the scrambler initialization state of 93 as a binary-valued columnm vector.

Data Types: `double`

Output Arguments

y — Scrambled or descrambled output
column vector | matrix

Scrambled or descrambled output, returned as a binary column vector or matrix with the same size and type as `bits`.

Version History

Introduced in R2017b

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

Extended Capabilities

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

See Also

`comm.Scrambler` | `comm.Descrambler` | `wlanInterpretScramblerState` | `wlanWaveformGenerator`

wlanSegmentDeparseBits

Segment-deparse data bits

Syntax

```
y = wlanSegmentDeparseBits(bits,cbw,numES,numCBPS,numBPSCS)
```

Description

`y = wlanSegmentDeparseBits(bits,cbw,numES,numCBPS,numBPSCS)` performs the inverse operation of the segment parsing defined in IEEE 802.11ac-2013 Section 22.3.10.7 when `cbw` is `'CBW16'` or `'CBW160'`.

Note Segment deparsing of the bits applies only when the channel bandwidth is either 16 MHz or 160 MHz, and is bypassed for the remaining channel bandwidths (as stated in the aforementioned section of IEEE802.11ac-2013). Therefore, when `cbw` is any accepted value other than `'CBW16'` or `'CBW160'`, `wlanSegmentParseBits` returns the input unchanged.

Examples

Segment-Deparse Coded Bits in Two OFDM Symbols

Segment-deparse the coded bits for a VHT configuration (with a channel bandwidth of 160 MHz and three spatial streams) into two OFDM symbols.

Define the input parameters. Set the channel bandwidth to 160 MHz, the number of coded bits per OFDM symbol to 2808, the number of spatial streams to 3, the number of encoded streams to 1, the number of coded bits per subcarrier per spatial stream to 2, and the number of OFDM symbols to 2. Calculate the number of coded bits per OFDM symbol per spatial stream by dividing the number of coded bits per OFDM symbol by the number of spatial streams.

```
chanBW = 'CBW160';
numCBPS = 2808;
numSS = 3;
numES = 1;
numBPSCS = 2;
numSym = 2;
numCBPSS = numCBPS/numSS;
```

Create the input sequence of bits.

```
bits = randi([0 1],numCBPSS*numSym,numSS);
```

Perform segment parsing on the bits.

```
parsedBits = wlanSegmentParseBits(bits,chanBW,numES,numCBPS,numBPSCS);
size(parsedBits)
```

```
ans = 1×3
```

```
   936     3     2
```

Perform segment deparsing on the parsed bits.

```
deparsedBits = wlanSegmentDeparseBits(parsedBits,chanBW,numES,numCBPS,numBPSCS);
size(deparsedBits)
```

```
ans = 1×2

        1872           3
```

Verify that the deparsed data matches the original data.

```
isequal(bits,deparsedBits)
```

```
ans = logical
   1
```

Input Arguments

`bits` — Input sequence

matrix | 3-D array

Input sequence of deinterleaved bits, specified as an ($N_{CBPSSI} \times N_{SYM}$)-by-$N_{SS}$-by-$N_{SEG}$ array, where:

- N_{CBPSSI} is the number of coded bits per OFDM symbol per spatial stream per interleaver block.
- N_{SYM} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.
- N_{SEG} is the number of segments. When `cbw` is `'CBW16'` or `'CBW160'`, N_{SEG} must be 2. Otherwise it must be 1.

Data Types: `double` | `int8`

`cbw` — Channel bandwidth

`'CBW1'` | `'CBW2'` | `'CBW4'` | `'CBW8'` | `'CBW16` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth in MHz, specified as `'CBW1'`,`'CBW2'`, `'CBW4'`,`'CBW8'`, `'CBW16'`, `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`.

Example: `'CBW160'`

Data Types: `char` | `string`

`numES` — Number of encoded streams

1 to 9 | 12

Number of encoded streams, specified as an integer from 1 to 9, or 12.

Data Types: `double`

`numCBPS` — Number of coded bits per OFDM symbol

positive integer

Number of coded bits per OFDM symbol, specified as a positive integer. When `cbw` is `'CBW16'` or `'CBW160'`, `numCBPS` must be an integer equal to $468 \times N_{BPSCS} \times N_{SS}$, where:

- N_{BPSCS} is the number of coded bits per subcarrier per spatial stream.
- N_{SS} is the number of spatial streams. It accounts for the number of columns (second dimension) of the input `bits`.

Data Types: `double`

`numBPSCS` — Number of coded bits per subcarrier per spatial stream
1 | 2 | 4 | 6 | 8

Number of coded bits per subcarrier per spatial stream, specified as log2(M), where M is the modulation order. Therefore, `numBPSCS` must equal:

- 1 for a BPSK modulation
- 2 for a QPSK modulation
- 4 for a 16QAM modulation
- 6 for a 64QAM modulation
- 8 for a 256QAM modulation

Data Types: `double`

Output Arguments

`y` — Merged segments of data
matrix

Merged segments of data, specified as an ($N_{CBPSS} \times N_{SYM}$)-by-N_{SS} matrix, where:

- N_{CBPSS} is the number of coded bits per OFDM symbol per spatial stream.
- N_{SYM} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.

Version History

Introduced in R2017b

Extended Capabilities

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

See Also

`wlanSegmentParseBits`

wlanSegmentDeparseSymbols

Segment-deparse data subcarriers

Syntax

```
y = wlanSegmentDeparseSymbols(sym,cbw)
```

Description

`y = wlanSegmentDeparseSymbols(sym,cbw)` performs segment deparsing on the input `sym` as per IEEE 802.11ac-2013, Section 22.3.10.9.3, when `cbw` is `'CBW16'` or `'CBW160'`.

Note Segment deparsing of the data subcarriers applies only when the channel bandwidth is either 16 MHz or 160 MHz, and is bypassed for the remaining channel bandwidths (as stated in the aforementioned section of IEEE802.11ac-2013). Therefore, when `cbw` is any accepted value other than `'CBW16'` or `'CBW160'`, `wlanSegmentDeparseSymbols` returns the input unchanged.

Examples

Segment-Deparse Symbols

Segment-deparse the symbols in four OFDM symbols for a VHT configuration with a channel bandwidth of 16 MHz and 3 spatial streams.

Define the input parameters. Since the channel bandwidth is 16 MHz, set the number of data subcarriers to 468 and the number of frequency segments to two.

```
chanBW = 'CBW16';
numSD = 468;
numSym = 4;
numSS = 3;
numSeg = 2;
```

Create the input sequence of symbols.

```
data = randi([0 1],numSD/numSeg,numSym,numSS,numSeg);
```

Segment-deparse the symbols into data subcarriers. The first dimension of the parsed output accounts for the total number of data subcarriers.

```
deparsedData = wlanSegmentDeparseSymbols(data,chanBW);
size(deparsedData)
```

```
ans = 1×3

   468     4     3
```

Get Symbol Order for VHT Configuration

Get the symbol order after stream deparsing a sequence for a VHT configuration with a channel bandwidth of 160 MHz and one spatial stream.

Define the input parameters. Since the channel bandwidth is 160 MHz, set the number of data subcarriers to 468 and the number of frequency segments to two.

```
chanBW = "CBW160";
numSD = 468;
numSym = 1;
numSS = 1;
numSeg = 2;
```

Create the input sequence of symbols.

```
sequence = (1:numSD*numSym*numSS).';
inp = reshape(sequence,numSD/numSeg,numSym,numSS,numSeg);
```

Segment-deparse the symbols. The output is a column vector with the sequence order of the symbols.

```
deparsedData = wlanSegmentDeparseSymbols(inp,chanBW);
deparsedData(1:10)
```

```
ans = 10×1

     1
     2
     3
     4
     5
     6
     7
     8
     9
    10
```

Input Arguments

`sym` — Input sequence

4-D array

Input sequence of frequency segments to deparse, specified as an (N_{SD}/N_{SEG})-by-$N_{SYM}$by-N_{SS}-by-N_{SEG} array, where:

- N_{SD} is the number of data subcarriers.
- N_{SEG} is the number of segments. When `cbw` is `'CBW16'` or `'CBW160'`, N_{SEG} is 2. Otherwise it is 1.
- N_{SYM} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.

Data Types: `double`
Complex Number Support: Yes

cbw — Channel bandwidth
`'CBW1'` | `'CBW2'` | `'CBW4'` | `'CBW8'` | `'CBW16` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth in MHz, specified as `'CBW1'`,`'CBW2'`, `'CBW4'`,`'CBW8'`, `'CBW16'`, `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`.

Example: `'CBW160'`

Data Types: `char` | `string`

Output Arguments

y — Deparsed frequency segments
3-D array

Deparsed frequency segments, specified as an N_{SD}-by-N_{SYM}-by-N_{SS} array, where:

- N_{SD} is the number of data subcarriers.
- N_{SYM} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.

Version History

Introduced in R2017b

Extended Capabilities

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

See Also

`wlanSegmentParseSymbols`

wlanSegmentParseBits

Segment-parse data bits

Syntax

```
y = wlanSegmentParseBits(bits,cbw,numES,numCBPS,numBPSCS)
```

Description

`y = wlanSegmentParseBits(bits,cbw,numES,numCBPS,numBPSCS)` performs segment parsing on the input `bits` as per IEEE 802.11ac-2013, Section 22.3.10.7, when `cbw` is `'CBW16'` or `'CBW160'`.

Note Segment parsing of the bits applies only when the channel bandwidth is either 16 MHz or 160 MHz, and is bypassed for the remaining channel bandwidths (as stated in the aforementioned section of IEEE802.11ac-2013). Therefore, when `cbw` is any accepted value other than `'CBW16'` or `'CBW160'`, `wlanSegmentParseBits` returns the input unchanged.

Examples

Segment-Parse Bits in Two OFDM Symbols

Segment-parse coded bits for a VHT configuration (with a channel bandwidth of 160 MHz and three spatial streams) into two OFDM symbols.

Define the input parameters. Set the channel bandwidth to 160 MHz, the number of coded bits per OFDM symbol to 2808, the number of spatial streams to 3, the number of encoded streams to 1, the number of coded bits per subcarrier per spatial stream to 2, and the number of OFDM symbols to 2. Calculate the number of coded bits per OFDM symbol per spatial stream by dividing the number of coded bits per OFDM symbol by the number of spatial streams.

```
chanBW = 'CBW160';
numCBPS = 2808;
numSS = 3;
numES = 1;
numBPSCS = 2;
numSym = 2;
numCBPSS = numCBPS/numSS;
```

Create the input sequence of bits.

```
bits = randi([0 1],numCBPSS*numSym,numSS,'int8');
```

Perform segment parsing on the bits.

```
parsedBits = wlanSegmentParseBits(bits,chanBW,numES,numCBPS,numBPSCS);
```

The parsed sequence is a three-dimensional array of bits.

```
size(parsedBits)

ans = 1×3

   936     3     2


parsedBits(1:5,:,:)

ans = 5x3x2 int8 array
ans(:,:,1) =

   1   0   1
   0   1   1
   1   0   1
   0   0   0
   1   0   1


ans(:,:,2) =

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

Get Bit Order of OFDM Symbol

Get the bit order after the segment parsing of an OFDM symbol of an S1G configuration with a channel bandwidth of 16 MHz, and two spatial streams.

Define the input parameters. Set the channel bandwidth to 16 MHz, the number of coded bits per OFDM symbol to 1872, the number of spatial streams to 2, the number of encoded streams to 1, the number of coded bits per subcarrier per spatial stream to 2 and the number of OFDM symbols to 2. Calculate the number of coded bits per OFDM symbol per spatial stream by dividing the number of coded bits per OFDM symbol by the number of spatial streams.

```
chanBW = 'CBW16';
numCBPS = 1872;
numSS = 2;
numES = 1;
numBPSCS = 2;
numSym = 1;
numCBPSS = numCBPS/numSS;
```

Create the input sequence.

```
sequence = (1:numCBPS*numSym).';
inp = reshape(sequence,numCBPSS*numSym,numSS);
```

Perform segment parsing on the sequence.

```
parsedSequence = wlanSegmentParseBits(inp,chanBW,numES,numCBPS,numBPSCS);
```

The parsed sequence is a three-dimensional array containing the corresponding bit order.

```
size(parsedSequence)
```

```
ans = 1×3

   468     2     2
```

Input Arguments

`bits` — Input sequence

matrix

Input sequence of stream-parsed bits, specified as an (N_{CBPSS}×N_{SYM})-by-N_{SS} matrix, where:

- N_{CBPSS} is the number of coded bits per OFDM symbol per spatial stream.
- N_{SYM} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.

Data Types: `double` | `int8`

`cbw` — Channel bandwidth

`'CBW1'` | `'CBW2'` | `'CBW4'` | `'CBW8'` | `'CBW16'` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth in MHz, specified as `'CBW1'`,`'CBW2'`, `'CBW4'`,`'CBW8'`, `'CBW16'`, `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`.

Example: `'CBW160'`

Data Types: `char` | `string`

`numES` — Number of encoded streams

1 to 9 | 12

Number of encoded streams, specified as an integer from 1 to 9, or 12.

Data Types: `double`

`numCBPS` — Number of coded bits per OFDM symbol

positive integer

Number of coded bits per OFDM symbol, specified as a positive integer. When `cbw` is `'CBW16'` or `'CBW160'`, `numCBPS` must be an integer equal to 468×N_{BPSCS}×N_{SS}, where:

- N_{BPSCS} is the number of coded bits per subcarrier per spatial stream.
- N_{SS} is the number of spatial streams. It accounts for the number of columns (second dimension) of the input `bits`.

Data Types: `double`

`numBPSCS` — Number of coded bits per subcarrier per spatial stream

1 | 2 | 4 | 6 | 8

Number of coded bits per subcarrier per spatial stream, specified as log2(M), where M is the modulation order. Therefore, `numBPSCS` must equal:

- 1 for a BPSK modulation
- 2 for a QPSK modulation
- 4 for a 16QAM modulation
- 6 for a 64QAM modulation
- 8 for a 256QAM modulation

Data Types: `double`

Output Arguments

y — Segment-parsed bits

matrix | 3-D array

Segment-parsed bits, specified as an (N_{CBPSSI}×N_{SYM})-by-N_{SS}-by-N_{SEG} array, where:

- N_{CBPSSI} is the number of coded bits per OFDM symbol per spatial stream per interleaver block.
- N_{SYM} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.
- N_{SEG} is the number of segments. When `cbw` is `'CBW16'` or `'CBW160'`, N_{SEG} is 2. Otherwise it is 1.

Version History

Introduced in R2017b

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanSegmentDeparseBits`

wlanSegmentParseSymbols

Segment-parse data subcarriers

Syntax

```
y = wlanSegmentParseSymbols(sym,cbw)
```

Description

`y = wlanSegmentParseSymbols(sym,cbw)` performs the inverse operation of the segment deparsing on the input `sym` defined in IEEE 802.11ac-2013, Section 22.3.10.9.3, when `cbw` is `'CBW16'` or `'CBW160'`.

Note Segment parsing of the data subcarriers applies only when the channel bandwidth is either 16 MHz or 160 MHz, and is bypassed for the remaining channel bandwidths (as stated in the aforementioned section of IEEE802.11ac-2013). Therefore, when `cbw` is any accepted value other than `'CBW16'` or `'CBW160'`, `wlanSegmentParseSymbols` returns the input unchanged.

Examples

Segment-Parse Symbols

Segment-deparse and segment-parse the symbols in four OFDM symbols for a VHT configuration with a channel bandwidth of 160 MHz and two spatial streams.

Define the input parameters. Since the channel bandwidth is 160 MHz, set the number of data subcarriers to 468 and the number of frequency segments to two.

```
chanBW = 'CBW160';
numSD = 468;
numSym = 4;
numSS = 2;
numSeg = 2;
```

Create the input sequence of symbols.

```
data = randi([0 1],numSD/numSeg,numSym,numSS,numSeg);
```

Segment-deparse the symbols into data subcarriers. The first dimension of the parsed output accounts for the total number of data subcarriers.

```
deparsedData = wlanSegmentDeparseSymbols(data,chanBW);
size(deparsedData)
```

```
ans = 1×3

   468     4     2
```

Segment-parse the symbols into data subcarriers. The size of the output is equal to the size of the original sequence.

```
segments = wlanSegmentParseSymbols(deparsedData,chanBW);
size(segments)
```

```
ans = 1×4

   234     4     2     2
```

Input Arguments

`sym` — Input sequence

3-D array

Input sequence of equalized data to be segmented, specified as an N_{SD}-by-N_{SYM}-by-N_{SS} array, where:

- N_{SD} is the number of data subcarriers.
- N_{SYM} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.

Data Types: `double`
Complex Number Support: Yes

`cbw` — Channel bandwidth

`'CBW1'` | `'CBW2'` | `'CBW4'` | `'CBW8'` | `'CBW16'` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth in MHz, specified as `'CBW1'`,`'CBW2'`, `'CBW4'`,`'CBW8'`, `'CBW16'`, `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`.

Example: `'CBW160'`

Data Types: `char` | `string`

Output Arguments

`y` — Frequency segments

4-D array

Frequency segments, specified as an (N_{SD}/N_{SEG})-by-$N_{SYM}$by-N_{SS}-by-N_{SEG} array, where:

- N_{SD} is the number of data subcarriers.
- N_{SEG} is the number of segments. When `cbw` is `'CBW16'` or `'CBW160'`, N_{SEG} is 2. Otherwise it is 1.
- N_{SYM} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.

Version History

Introduced in R2017b

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanSegmentDeparseSymbols`

wlanSpectralFlatness

Test spectral flatness

Syntax

```
pass = wlanSpectralFlatness(chanEst,format,cbw)
[pass,deviation,testSC] = wlanSpectralFlatness(chanEst,format,cbw)
```

Description

`pass = wlanSpectralFlatness(chanEst,format,cbw)` tests spectral flatness by using channel estimates `channelEst` for WLAN packet format `format` and channel bandwidth `cbw`. The function determines the test result by comparing the spectral flatness measurements with the standard-specified range.

`[pass,deviation,testSC] = wlanSpectralFlatness(chanEst,format,cbw)` also returns `deviation`, the power deviation of each test subcarrier from the average power level across lower test subcarriers. The function splits the test subcarrier indices into upper and lower sets based on their absolute frequency and returns these indices as `testSC`.

Examples

Test Spectral Flatness for VHT Transmission

Configure and generate a very high throughput (VHT) waveform.

```
cfgVHT = wlanVHTConfig;
bits = [1; 0; 0; 1];
waveform = wlanWaveformGenerator(bits,cfgVHT);
```

Demodulate the VHT long training field (VHT-LTF).

```
field = "VHT-LTF";
ind = wlanFieldIndices(cfgVHT,field);
sym = wlanVHTLTFDemodulate(waveform(ind(1):ind(2),:),cfgVHT);
```

Perform channel estimation on the demodulated signal.

```
chanEst = wlanVHTLTFChannelEstimate(sym,cfgVHT);
```

Test the spectral flatness of the transmission, specifying the channel bandwidth and packet format, and display the result.

```
format = "VHT";
cbw = cfgVHT.ChannelBandwidth;
[pass,deviation,testSC] = wlanSpectralFlatness(chanEst,format,cbw);
disp(pass)
```

```
   1
```

Input Arguments

`chanEst` — Channel estimates for data and pilot subcarriers

complex-valued 3-D array

Channel estimates for the data and pilot subcarriers, specified as a complex-valued array of size N_{st}-by-N_{sts}-by-N_r.

- N_{st} is the number of occupied subcarriers.
- N_{sts} is the number of space-time streams.
- N_r is the number of receive antennas.

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

`format` — WLAN packet format

`"EHT"` | `"HE"` | `"VHT"` | `"HT"` | `"Non-HT"` | `"S1G"`

WLAN packet format, specified as a string scalar or character vector corresponding to these IEEE packet formats and 802.11 standards.

Value	Format	Standard
`"EHT"`	Extremely high throughput (EHT)	IEEE P802.11be/D1.2 [1]
`"HE"`	High-efficiency single-user (HE SU), HE extended-range single-user (HE ER SU), HE multi-user (HE MU), or HE trigger-based (HE TB)	IEEE Std 802.11ax-2021 [2]
`"VHT"`	Very high throughput (VHT)	IEEE Std 802.11-2020 [3]
`"HT"`	High throughput (HT)	
`"Non-HT"`	Non-high-throughput (non-HT)	
`"S1G"`	Sub 1 GHz (S1G)	

Data Types: `char` | `string`

`cbw` — Channel bandwidth

`"CBW1"` | `"CBW2"` | `"CBW4"` | `"CBW5"` | `"CBW8"` | `"CBW10"` | `"CBW16"` | `"CBW20"` | `"CBW40"` | `"CBW80"` | `"CBW160"` | `"CBW320"`

Channel bandwidth, specified as one of these values.

- `"CBW1"` — Channel bandwidth of 1 MHz
- `"CBW2"` — Channel bandwidth of 2 MHz
- `"CBW4"` — Channel bandwidth of 4 MHz
- `"CBW5"` — Channel bandwidth of 5 MHz
- `"CBW8"` — Channel bandwidth of 8 MHz
- `"CBW10"` — Channel bandwidth of 10 MHz

- `"CBW16"` — Channel bandwidth of 16 MHz
- `"CBW20"` — Channel bandwidth of 20 MHz
- `"CBW40"` — Channel bandwidth of 40 MHz
- `"CBW80"` — Channel bandwidth of 80 MHz
- `"CBW160"` — Channel bandwidth of 160 MHz
- `"CBW320"` — Channel bandwidth of 320 MHz

Valid values of this input depend on the value of the `format` input according to this table.

Value of `format`	Valid Values of `cbw`
`"EHT"`	`"CBW20"`, `"CBW40"`, `"CBW80"`, `"CBW160"`, `"CBW320"`
`"HE"`	`"CBW20"`, `"CBW40"`, `"CBW80"`, `"CBW160"`
`"VHT"`	`"CBW20"`, `"CBW40"`, `"CBW80"`, `"CBW160"`
`"HT"`	`"CBW20"`, `"CBW40"`
`"Non-HT"`	`"CBW5"`, `"CBW10"`, `"CBW20"`, `"CBW40"`, `"CBW80"`, `"CBW160"`
`"S1G"`	`"CBW1"`, `"CBW2"`, `"CBW4"`, `"CBW8"`, `"CBW16"`

Data Types: `char` | `string`

Output Arguments

`pass` — Spectral flatness test result

`1` | `0`

Spectral flatness test result, returned as `1` or `0`. If the measured spectral flatness falls within the standard-specified range, the received signal passes the test and the function returns this output as `1`. Otherwise, the function returns this output as `0`.

Data Types: `logical`

`deviation` — Power deviation

1-by-2 cell array

Power deviation of each test subcarrier from the average power level across lower test subcarriers, returned as a 1-by-2 cell array of real-valued matrices.

The first entry contains the power deviations per antenna across the lower test subcarriers in a matrix of size N_{lsc}-by-N_{r}, where N_{lsc} is the number of lower test subcarriers and N_{r} is the number of receive antennas.

The second entry contains the power deviations per antenna across the upper test subcarriers in a matrix of size N_{usc}-by-N_{r}, where N_{usc} is the number of upper test subcarriers.

Data Types: `cell`

`testSC` — Test subcarrier indices

1-by-2 cell array

Test subcarrier indices, returned as a 1-by-2 cell array of real-valued column vectors.

The first element contains the lower test subcarrier indices in a vector of length equal to the number of lower test subcarriers.

The second element contains the upper test subcarrier indices in a vector of length equal to the number of upper test subcarriers.

Data Types: `cell`

Version History

Introduced in R2022a

References

[1] IEEE P802.11be/D1.2. “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 8: Enhancements for Extremely High Throughput (EHT).” Draft Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

[2] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

[3] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`wlanReferenceSymbols` | `wlanClosestReferenceSymbol`

wlanStreamDeparse

Stream-deparse spatial streams

Syntax

```
y = wlanStreamDeparse(bits,numES,numCBPS,numBPSCS)
```

Description

`y = wlanStreamDeparse(bits,numES,numCBPS,numBPSCS)` stream-deparses spatial streams `bits` for the specified number of encoded streams, `numES`, coded bits per OFDM symbol, `numCBPS`, and coded bits per subcarrier per spatial stream, `numBPSCS`. The function implements the inverse of the stream-parsing operation defined in sections 19.3.11.8.2 and 21.3.10.6 of [1] and section 27.3.12.6 of [2].

Examples

Stream-Deparse Input Bits

Stream-deparse five OFDM symbols with two spatial streams into one encoded stream.

Specify the number of coded bits per OFDM symbol, coded bits per subcarrier per spatial stream, encoded streams, spatial streams, and OFDM symbols.

```
numCBPS = 432;
numBPSCS = 2;
numES = 1;
numSS = 2;
numSym = 5;
```

Create a parsed input of hard bits.

```
bits = randi([0 1],numCBPS/numSS*numSym,numSS);
```

Stream-deparse the bits.

```
y = wlanStreamDeparse(bits,numES,numCBPS,numBPSCS);
```

Input Arguments

`bits` — Spatial streams

real-valued matrix

Spatial streams, specified as a real-valued matrix of size ($N_{CBPSS} \times N_{Sym}$)-by-N_{SS}.

- N_{CBPSS} is the number of coded bits per OFDM symbol per spatial stream.
- N_{Sym} is the number of OFDM symbols.

- N_{SS} is the number of spatial streams.

Data Types: `single` | `double`

`numES` — Number of encoded streams
positive integer

Number of encoded streams, specified as a positive integer.

Data Types: `double`

`numCBPS` — Number of coded bits per OFDM symbol
positive integer

Number of coded bits per OFDM symbol, specified as a positive integer. This value is typically $N_{BPSCS} \times N_{SS} \times N_{SD}$, where:

- N_{BPSCS} is the number of coded bits per subcarrier per spatial stream (that is, the `numBPSCS` input).
- N_{SS} is the number of spatial streams.
- N_{SD} is the number of complex data numbers per frequency segment.

Data Types: `double`

`numBPSCS` — Number of coded bits per subcarrier per spatial stream
`1` | positive even integer

Number of coded bits per subcarrier per spatial stream, specified as `1` or a positive even integer.

Data Types: `double`

Output Arguments

`y` — Stream-deparsed data
real-valued matrix

Stream-deparsed data, returned as a matrix of size ($N_{CBPS} \times N_{Sym} \div N_{ES}$)-by-$N_{ES}$.

- N_{CBPS} is the number of coded bits per OFDM symbol (that is, the `numCBPS` input).
- N_{Sym} is the number of OFDM symbols.
- N_{ES} is the number of encoded streams.

The function returns this output with the same data type as the `bits` input.

Data Types: `single` | `double`

Version History

Introduced in R2017b

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for

Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

[2] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanStreamParse`

wlanStreamParse

Stream-parse encoded streams

Syntax

```
y = wlanStreamParse(bits,numSS,numCBPS,numBPSCS)
```

Description

`y = wlanStreamParse(bits,numSS,numCBPS,numBPSCS)` stream-parses encoded streams `bits` into `numSS` spatial streams for the specified number of coded bits per OFDM symbol, `numCBPS`, and coded bits per subcarrier per spatial stream, `numBPSCS`. The function implements the operation defined in sections 19.3.11.8.2 and 21.3.10.6 of [1] and section 27.3.12.6 of [2].

Examples

Stream-Parse Input Bits

Stream-parse three OFDM symbols with two encoded streams into five spatial streams.

Specify the number of coded bits per OFDM symbol, coded bits per subcarrier per spatial stream, encoded streams, spatial streams, and OFDM symbols.

```
numCBPS = 3240;
numBPSCS = 6;
numES = 2;
numSS = 5;
numSym = 3;
```

Create a random encoded stream of bits.

```
bits = randi([0 1],numCBPS*numSym/numES,numES,"int8");
```

Stream-parse the encoded stream.

```
y = wlanStreamParse(bits,numSS,numCBPS,numBPSCS);
```

Verify the size of the stream-parsed bits.

```
size(y)
```

```
ans = 1×2

        1944           5
```

Get Bit Order After Stream Parsing

Get the bit order of an OFDM symbol after stream-parsing one encoded stream into three spatial streams.

Specify the number of coded bits per OFDM symbol, coded bits per subcarrier per spatial stream, encoded streams, spatial streams, and OFDM symbols.

```
numCBPS = 156;
numBPSCS = 1;
numES = 1;
numSS = 3;
numSym = 1;
```

Create an input sequence of ordered symbols with appropriate dimensions.

```
sequence = (1:numCBPS*numSym).';
bits = reshape(sequence,numCBPS*numSym/numES,numES);
```

Stream-parse the symbols.

```
y = wlanStreamParse(bits,numSS,numCBPS,numBPSCS);
```

Input Arguments

`bits` — Encoded streams

real-valued matrix

Encoded streams, specified as a real-valued matrix of size ($N_{CBPS} \times N_{Sym} \div N_{ES}$)-by-$N_{ES}$.

- N_{CBPS} is the number of coded bits per OFDM symbol (that is, the `numCBPS` input).
- N_{Sym} is the number of OFDM symbols.
- N_{ES} is the number of encoded streams.

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

`numSS` — Number of spatial streams

positive integer

Number of spatial streams, specified as a positive integer.

Data Types: `double`

`numCBPS` — Number of coded bits per OFDM symbol

positive integer

Number of coded bits per OFDM symbol, specified as a positive integer. This value is typically $N_{BPSCS} \times N_{SS} \times N_{SD}$, where:

- N_{BPSCS} is the number of coded bits per subcarrier per spatial stream (that is, the `numBPSCS` input).
- N_{SS} is the number of spatial streams.
- N_{SD} is the number of complex data numbers per frequency segment.

Data Types: `double`

numBPSCS — Number of coded bits per subcarrier per spatial stream
1 | positive even integer

Number of coded bits per subcarrier per spatial stream, specified as 1 or a positive even integer.

Data Types: `double`

Output Arguments

y — Stream-parsed data
real-valued matrix

Stream-parsed data, returned as a real-valued matrix of size ($N_{CBPSS} \times N_{Sym}$)-by-N_{SS}.

- N_{CBPSS} is the number of coded bits per OFDM symbol per spatial stream.
- N_{Sym} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.

The function returns this output with the same data type as the `bits` input.

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

Version History

Introduced in R2017b

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

[2] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

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

See Also

`wlanStreamDeparse`

wlanSymbolTimingEstimate

Fine symbol timing estimate using L-LTF

Syntax

```
startOffset = wlanSymbolTimingEstimate(rxSig,cbw)
startOffset = wlanSymbolTimingEstimate(rxSig,cbw,threshold)
[startOffset,M] = wlanSymbolTimingEstimate( ___ )
```

Description

`startOffset = wlanSymbolTimingEstimate(rxSig,cbw)` estimates the timing offset between the start of received waveform `rxSig` to the start of the "L-STF" on page 3-459 [17] for channel bandwidth `cbw`.

`startOffset = wlanSymbolTimingEstimate(rxSig,cbw,threshold)` specifies the threshold that the decision metric must meet or exceed to obtain a symbol timing estimate.

`[startOffset,M] = wlanSymbolTimingEstimate( ___ )` also returns the decision metric of the symbol timing algorithm using any combination of input arguments in previous syntaxes.

Examples

Detect HT Packet and Estimate Symbol Timing

Detect a received 802.11n™ packet and estimate its symbol timing at 20 dB SNR.

Create an HT format configuration object and TGn channel configuration object.

```
cfgHT = wlanHTConfig;
tgn = wlanTGnChannel;
```

Generate a transmit waveform and add a delay at the start of the waveform.

```
txWaveform = wlanWaveformGenerator([1;0;0;1],cfgHT);
txWaveform = [zeros(100,1);txWaveform];
```

Pass the waveform through the TGn channel model and add noise.

```
SNR = 20; % In decibels
fadedSig = tgn(txWaveform);
rxWaveform = awgn(fadedSig,SNR,0);
```

Detect the packet. Extract the non-HT fields. Estimate the fine packet offset using the coarse detection for the first symbol of the waveform and the non-HT preamble field indices.

```
startOffset = wlanPacketDetect(rxWaveform,cfgHT.ChannelBandwidth);
ind = wlanFieldIndices(cfgHT);
nonHTFields = rxWaveform(startOffset+(ind.LSTF(1):ind.LSIG(2)),:);
```

17 IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.

```
startOffset = wlanSymbolTimingEstimate(nonHTFields, ...
    cfgHT.ChannelBandwidth)
```

```
startOffset = 6
```

Detect HT Packet and Set Threshold When Estimating Symbol Timing

Impair an HT waveform by passing it through a TGn channel configured to model a large delay spread. Detect the waveform and estimate the symbol timing. Adjust the decision metric threshold and estimate the symbol timing again.

Create an HT format configuration object and TGn channel configuration object. Specify the Model-E delay profile, which introduces a large delay spread.

```
cfgHT = wlanHTConfig;

tgn = wlanTGnChannel;
tgn.DelayProfile = 'Model-E';
```

Generate a transmit waveform and add a delay at the start of the waveform.

```
txWaveform = wlanWaveformGenerator([1;0;0;1],cfgHT);
txWaveform = [zeros(100,1);txWaveform];
```

Pass the waveform through the TGn channel model and add noise.

```
SNR = 50; % In decibels
fadedSig = tgn(txWaveform);
rxWaveform = awgn(fadedSig,SNR,0);
```

Detect the packet. Extract the non-HT fields. Estimate the fine packet offset using the coarse detection for the first symbol of the waveform and the non-HT preamble field indices. Adjust the decision metric threshold and estimate the fine packet offset again.

```
startOffset = wlanPacketDetect(rxWaveform,cfgHT.ChannelBandwidth);
ind = wlanFieldIndices(cfgHT);
nonHTFields = rxWaveform(startOffset+(ind.LSTF(1):ind.LSIG(2)),:);

startOffset = wlanSymbolTimingEstimate(nonHTFields, ...
    cfgHT.ChannelBandwidth)
```

```
startOffset = 5
```

```
threshold = 0.1
```

```
threshold = 0.1000
```

```
startOffset = wlanSymbolTimingEstimate(nonHTFields, ...
    cfgHT.ChannelBandwidth,threshold)
```

```
startOffset = 9
```

Detecting the correct timing offset is more challenging for a channel model with large delay spread. For large delay spread channels, you can try lowering the threshold setting to see if performance improves in an end-to-end simulation.

Estimate Symbol Timing of TGn-Impaired HT Waveform

Detect a received 802.11n™ packet and estimate its symbol timing at 15 dB SNR.

Create an HT format configuration object. Specify two transmit antennas and two space-time streams.

```
cfgHT = wlanHTConfig;
nAnt = 2;
cfgHT.NumTransmitAntennas = nAnt;
cfgHT.NumSpaceTimeStreams = nAnt;
```

Show the logic behind the MCS selection for BPSK modulation.

```
if cfgHT.NumSpaceTimeStreams == 1
    cfgHT.MCS = 0;
elseif cfgHT.NumSpaceTimeStreams == 2
    cfgHT.MCS = 8;
elseif cfgHT.NumSpaceTimeStreams == 3
    cfgHT.MCS = 16;
elseif cfgHT.NumSpaceTimeStreams == 4
    cfgHT.MCS = 24;
end
```

Generate a transmit waveform and add a delay at the start of the waveform.

```
txWaveform = wlanWaveformGenerator([1;0;0;1],cfgHT);
txWaveform = [zeros(100,cfgHT.NumTransmitAntennas);txWaveform];
```

Create a TGn channel configuration object for two transmit antennas and two receive antennas. Specify the Model-B delay profile. Pass the waveform through the TGn channel model and add noise.

```
tgn = wlanTGnChannel;
tgn.NumTransmitAntennas = nAnt;
tgn.NumReceiveAntennas = nAnt;
tgn.DelayProfile = 'Model-B';

SNR = 15; % In decibels
fadedSig = tgn(txWaveform);
rxWaveform = awgn(fadedSig,SNR,0);
```

Detect the packet. Extract the non-HT fields. Estimate the fine packet offset using the coarse detection for the first symbol of the waveform and the non-HT preamble field indices.

```
startOffset = wlanPacketDetect(rxWaveform,cfgHT.ChannelBandwidth);
ind = wlanFieldIndices(cfgHT);
nonHTFields = rxWaveform(startOffset+(ind.LSTF(1):ind.LSIG(2)),:);

startOffset = wlanSymbolTimingEstimate(nonHTFields, ...
    cfgHT.ChannelBandwidth)
```

```
startOffset = 8
```

Estimate VHT Packet Symbol Timing

Return the symbol timing and decision metric of an 802.11ac™ packet without channel impairments.

Create a VHT format configuration object. Specify two transmit antennas and two space-time streams.

```
cfgVHT = wlanVHTConfig;
cfgVHT.NumTransmitAntennas = 2;
cfgVHT.NumSpaceTimeStreams = 2;
```

Generate a VHT format transmit waveform. Add a 50-sample delay at the start of the waveform.

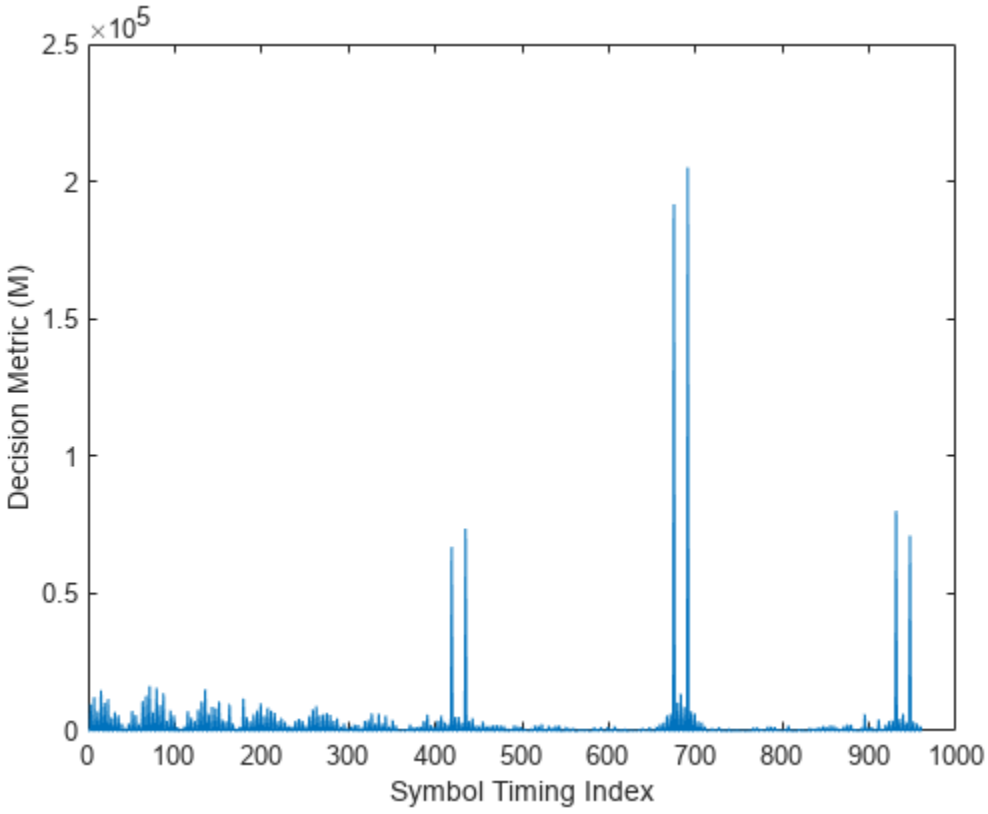
```
txWaveform = wlanWaveformGenerator([1;0;0;1],cfgVHT);
txWaveform = [zeros(50,cfgVHT.NumTransmitAntennas); txWaveform];
```

Extract the non-HT preamble fields. Obtain the timing offset estimate and decision metric.

```
ind = wlanFieldIndices(cfgVHT);
nonhtfields = txWaveform(ind.LSTF(1):ind.LSIG(2),:);
[startOffset,M] = wlanSymbolTimingEstimate(nonhtfields, ...
    cfgVHT.ChannelBandwidth);
```

Plot the returned decision metric for the non-HT preamble of the VHT format transmission waveform.

```
figure
plot(M)
xlabel('Symbol Timing Index')
ylabel('Decision Metric (M)')
```



Input Arguments

`rxSig` — Received signal

complex-valued matrix

Received signal containing an L-LTF, specified as a complex-valued matrix of size N_S-by-N_R. N_S is the number of time-domain samples in the L-LTF and N_R is the number of receive antennas.

Data Types: `single` | `double`

`cbw` — Channel bandwidth

`'CBW5'` | `'CBW10'` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'` | `'CBW320'`

Channel bandwidth, specified as one of these values.

- `'CBW5'` – Channel bandwidth of 5 MHz
- `'CBW10'` – Channel bandwidth of 10 MHz
- `'CBW20'` – Channel bandwidth of 20 MHz
- `'CBW40'` – Channel bandwidth of 40 MHz
- `'CBW80'` – Channel bandwidth of 80 MHz
- `'CBW160'` – Channel bandwidth of 160 MHz
- `'CBW320'` – Channel bandwidth of 320 MHz

Data Types: `char` | `string`

`threshold` — Decision threshold

`1` (default) | scalar in the interval [0, 1]

Decision threshold, specified as a scalar in the interval [0, 1].

To maximize the packet reception performance, you can try different valued of this input. For channels with small delay spread with respect to the cyclic prefix length, MathWorks® recommends the default value. For a wireless channel with large delay spread with respect to the cyclic prefix length, such as TGn channel with `'Model E'` delay profile, MathWorks suggests a value of 0.5.

By lowering the threshold setting, you add a nonnegative corrector to the symbol timing estimate as compared to the estimate using the default threshold setting. The range of the timing corrector is [0, CSD ns/sampling duration]. For more information, see “Cyclic Shift Delay (CSD)” on page 3-460.

Data Types: `double`

Output Arguments

`startOffset` — Timing offset

integer | `[]`

Timing offset, in samples, between the start of `rxSig` and the start of the L-STF, returned as an integer in the interval [–L, N_S– 2L]. L is the length of the L-LTF and N_S is the number of samples. Using the `cbw` input to determine the range of symbol timing, the function estimates the offset to the start of L-STF by cross-correlating the received signal with a locally generated “L-LTF” on page 3-459 of the first antenna.

- The function returns this output as `[]` when N_S is less than L.
- The function returns this output as a negative integer when the input waveform does not contain a complete “L-STF” on page 3-459.

Data Types: `double`

`M` — Cross-correlation
real-valued row vector

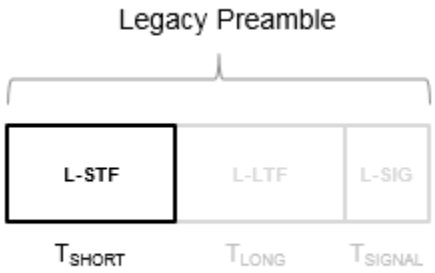
Cross-correlation between the received signal and the locally generated L-LTF of the first transmit antenna, returned as a real-valued row vector of length $N_S - L + 1$.

Data Types: `double`

More About

L-STF

The legacy short training field (L-STF) is the first field of the 802.11 OFDM PLCP legacy preamble. The L-STF is a component of VHT, HT, and non-HT PPDUs.



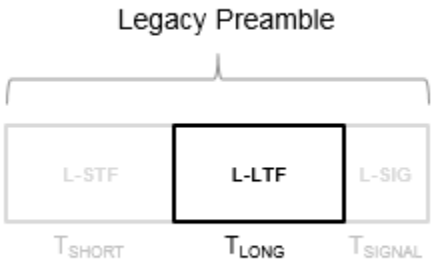
The L-STF duration varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period ($T_{FFT} = 1 / \Delta_F$)	L-STF Duration ($T_{SHORT} = 10 \times T_{FFT} / 4$)
20, 40, 80, 160, and 320	312.5	3.2 µs	8 µs
10	156.25	6.4 µs	16 µs
5	78.125	12.8 µs	32 µs

Because the sequence has good correlation properties, it is used for start-of-packet detection, for coarse frequency correction, and for setting the AGC. The sequence uses 12 of the 52 subcarriers that are available per 20 MHz channel bandwidth segment. For 5 MHz, 10 MHz, and 20 MHz bandwidths, the number of channel bandwidths segments is 1.

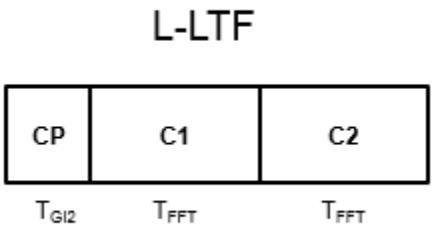
L-LTF

The L-LTF is the second field in the 802.11 OFDM PLCP legacy preamble. The L-LTF is a component of HE, VHT, HT, and non-HT PPDUs.



Channel estimation, fine frequency offset estimation, and fine symbol timing offset estimation rely on the L-LTF.

The L-LTF is composed of a cyclic prefix (CP) followed by two identical long training symbols (C1 and C2). The CP consists of the second half of the long training symbol.



The L-LTF duration varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period ($T_{FFT} = 1 / \Delta_F$)	Cyclic Prefix or Training Symbol Guard Interval (GI2) Duration ($T_{GI2} = T_{FFT} / 2$)	L-LTF Duration ($T_{LONG} = T_{GI2} + 2 \times T_{FFT}$)
20, 40, 80, 160, and 320	312.5	3.2 µs	1.6 µs	8 µs
10	156.25	6.4 µs	3.2 µs	16 µs
5	78.125	12.8 µs	6.4 µs	32 µs

Cyclic Shift Delay (CSD)

A CSD is added to the L-LTF for each transmit antenna, which causes multiple strong peaks in the correlation function *M*. The multiple peaks affect the accuracy of fine symbol timing estimation. For more information, see section 21.3.8.2.1 and Table 21-10 of [1].

Version History

Introduced in R2017a

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanCoarseCFOEstimate` | `comm.PhaseFrequencyOffset` | `wlanLLTF`

wlanVHTData

Generate VHT-Data field

Syntax

```
y = wlanVHTData(psdu,cfg)
y = wlanVHTData(psdu,cfg,scramInit)
y = wlanVHTData( ___ ,OversamplingFactor=osf)
```

Description

`y = wlanVHTData(psdu,cfg)` generates a "VHT-Data field" on page 3-465[18] time-domain waveform from user data bits `psdu` for transmission parameters `cfg`. See "VHT-Data Field Processing" on page 3-466 for waveform generation details.

`y = wlanVHTData(psdu,cfg,scramInit)` uses `scramInit` for the scrambler initialization state.

`y = wlanVHTData( ___ ,OversamplingFactor=osf)` generates an oversampled VHT-Data waveform with the specified oversampling factor. For more information about oversampling, see "FFT-Based Oversampling" on page 3-468.

Examples

Generate VHT-Data Waveform

Generate the waveform for a MIMO 20 MHz VHT-Data field.

Create a VHT configuration object. Assign a 20 MHz channel bandwidth, two transmit antennas, two space-time streams, and set MCS to four.

```
cfgVHT = wlanVHTConfig('ChannelBandwidth','CBW20','NumTransmitAntennas',2,'NumSpaceTimeStreams',2
```

Generate the user payload data and the VHT-Data field waveform.

```
psdu = randi([0 1],cfgVHT.PSDULength*8,1);
y = wlanVHTData(psdu,cfgVHT);
size(y)
```

```
ans = 1×2

        2160           2
```

The 20 MHz waveform is an array with two columns, corresponding to two transmit antennas. There are 2160 complex samples in each column.

```
y(1:10,:)
```

```
ans = 10×2 complex
```

18 IEEE Std 802.11ac-2013 Adapted and reprinted with permission from IEEE. Copyright IEEE 2013. All rights reserved.

```
-0.0598 + 0.1098i  -0.1904 + 0.1409i
 0.6971 - 0.3068i  -0.0858 - 0.2701i
-0.1284 + 0.9268i  -0.8318 + 0.3314i
-0.1180 + 0.0731i   0.1313 + 0.4956i
 0.3591 + 0.5485i   0.9749 + 0.2859i
-0.9751 + 1.3334i   0.0559 + 0.4248i
 0.0881 - 0.8230i  -0.1878 - 0.2959i
-0.2952 - 0.4433i  -0.1005 - 0.4035i
-0.5562 - 0.3940i  -0.1292 - 0.5976i
 1.0999 + 0.3292i  -0.2036 - 0.0200i
```

Input Arguments

`psdu` — PHY service data unit

vector

PHY service data unit (“PSDU” on page 3-466), specified as an N_b-by-1 vector. N_b is the number of bits and equals `PSDULength` × 8.

Data Types: `double`

`cfg` — Transmission parameters

`wlanVHTConfig` object

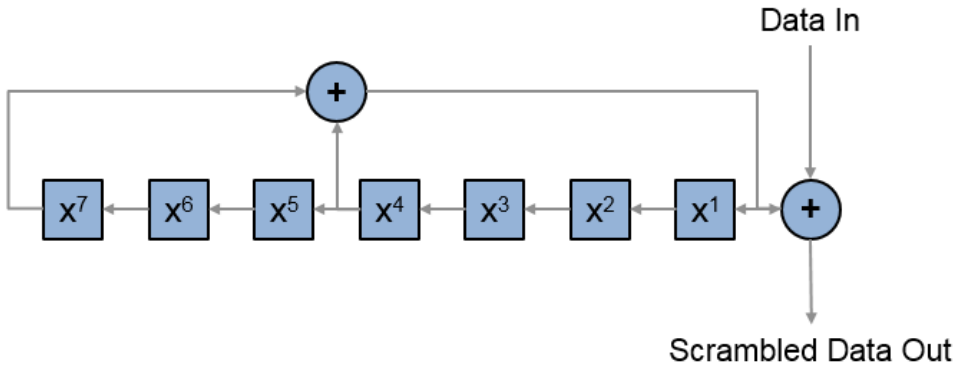
Transmission parameters, specified as a `wlanVHTConfig` object.

`scramInit` — Scrambler initialization state

93 (default) | integer in the interval [1, 127] | integer row vector | binary vector | binary matrix

Initial scrambler state of the data scrambler for each packet generated, specified as an integer, a binary vector, a 1-by-N_U integer row vector, or a 7-by-N_U binary matrix. N_U is the number of users, from 1 to 4. If specified as an integer or binary vector, the setting applies to all users. If specified as a row vector or binary matrix, the setting for each user is specified in the corresponding column, as an integer in the interval [1, 127] or the corresponding binary vector.

The scrambler initialization used on the transmission data follows the process described in IEEE Std 802.11-2012, Section 18.3.5.5 and IEEE Std 802.11ad-2012, Section 21.3.9. The header and data fields that follow the scrambler initialization field (including data padding bits) are scrambled by XORing each bit with a length-127 periodic sequence generated by the polynomial $S(x) = x^7+x^4+1$. The octets of the PSDU are placed into a bit stream and, within each octet, bit 0 (LSB) is first and bit 7 (MSB) is last. This figure shows the generation of the sequence and the XOR operation.



Conversion from integer to bits uses left-MSB orientation. For example, initializing the scrambler with decimal `1`, the bits map to these elements.

Element	X^7	X^6	X^5	X^4	X^3	X^2	X^1
Bit Value	0	0	0	0	0	0	1

To generate the bit stream equivalent to a decimal, use the `int2bit` function. For example, for decimal `1`:

```
int2bit(1,7)'
ans =

     0     0     0     0     0     0     1
```

Example: `[1;0;1;1;1;0;1]` conveys the scrambler initialization state of 93 as a binary vector.

Data Types: `double` | `int8`

`osf` — Oversampling factor
1 (default) | scalar greater than or equal to 1

Oversampling factor, specified as a scalar greater than or equal to 1. The oversampled cyclic prefix length must be an integer number of samples.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

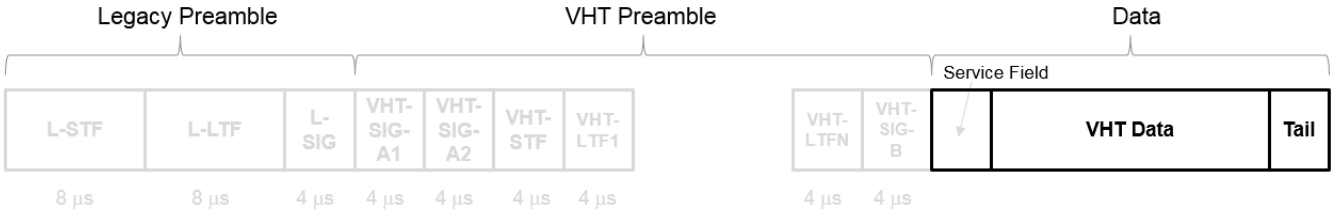
y — VHT-Data field time-domain waveform
matrix

"VHT-Data field" on page 3-465 time-domain waveform, returned as an N_S-by-N_T matrix. N_S is the number of time-domain samples and N_T is the number of transmit antennas. See "VHT-Data Field Processing" on page 3-466 for waveform generation details.

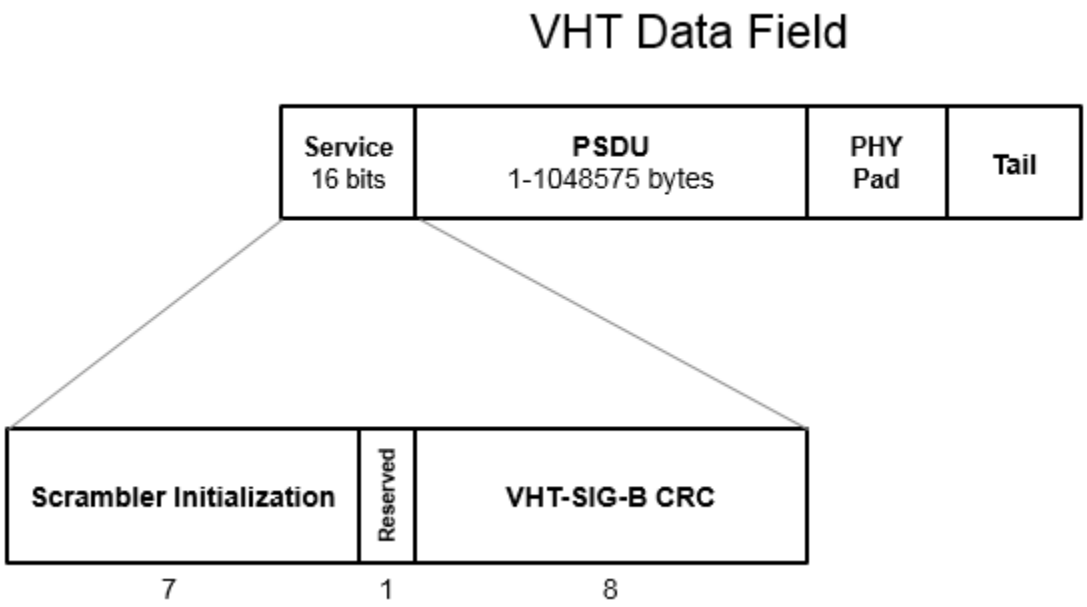
More About

VHT-Data field

The VHT-Data field carries one or more frames from the medium access control (MAC) layer. This field follows the VHT-SIG-B field in a VHT PPDUs.



For a detailed description of the VHT-Data field, see section 21.3.10 of IEEE Std 802.11-2016. The VHT Data field consists of four subfields.



- **Service field** — Contains a seven-bit scrambler initialization state, one bit reserved for future considerations, and eight bits for the VHT-SIG-B cyclic redundancy check (CRC) field
- **PSDU** — Variable-length field containing a PLCP service data unit
- **PHY Pad** — Variable number of bits passed to the transmitter to create a complete OFDM symbol
- **Tail** — Bits required to terminate a convolutional code (not required when the transmission uses LDPC channel coding)

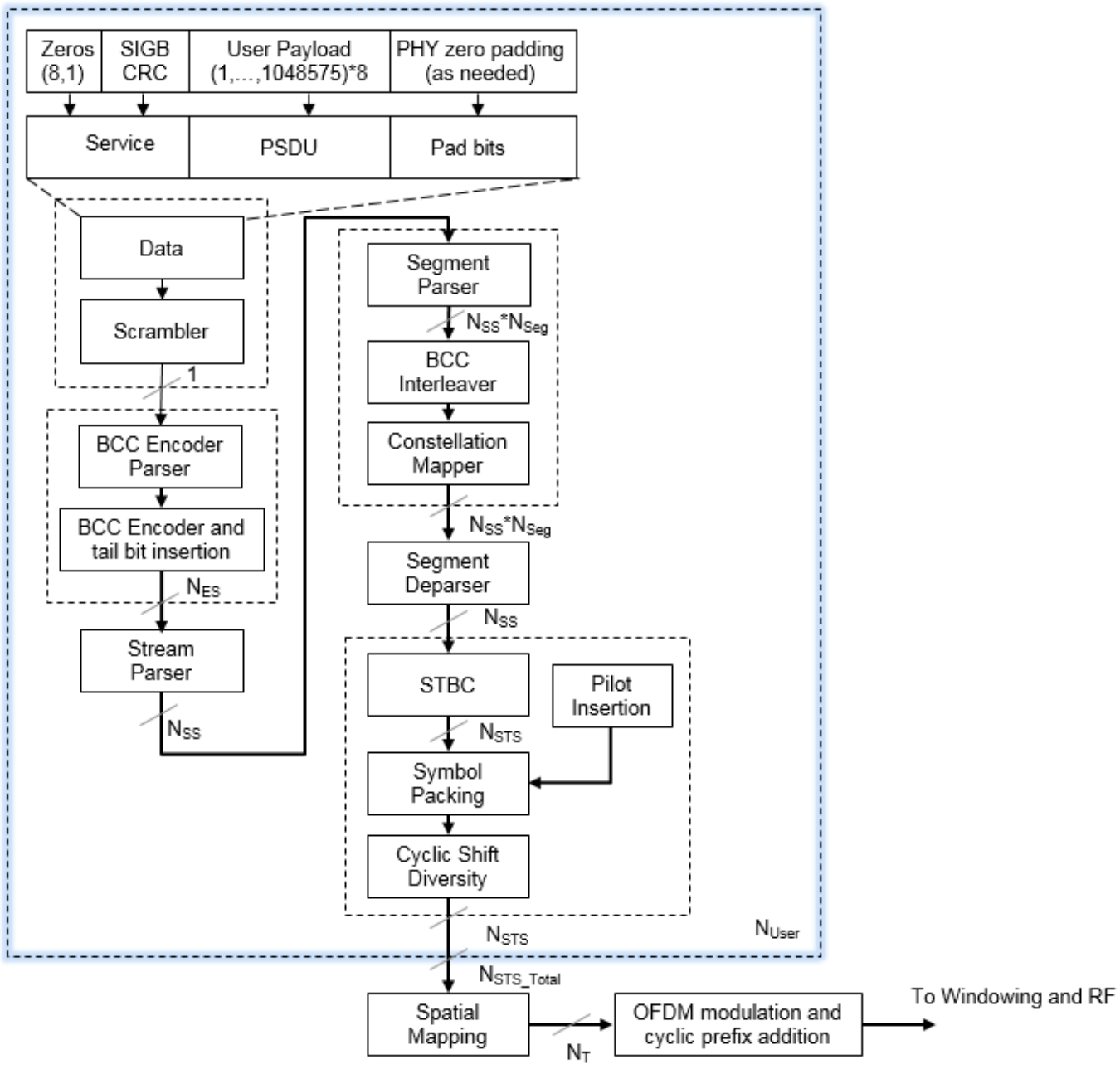
PSDU

Physical layer (PHY) Service Data Unit (PSDU). A PSDU can consist of one medium access control (MAC) protocol data unit (MPDU) or several MPDUs in an aggregate MPDU (A-MPDU). In a single user scenario, the VHT-Data field contains one PSDU. In a multi-user scenario, the VHT-Data field carries up to four PSDUs for up to four users.

Algorithms

VHT-Data Field Processing

The “VHT-Data field” on page 3-465 encodes the service, “PSDU” on page 3-466, pad bits, and tail bits. The `wlanVHTData` function performs transmitter processing on the “VHT-Data field” on page 3-465 and outputs the time-domain waveform for N_T transmit antennas.



N_{ES} is the number of BCC encoders.
N_{SS} is the number of spatial streams.
N_{STS} is the number of space-time streams.
N_T is the number of transmit antennas.

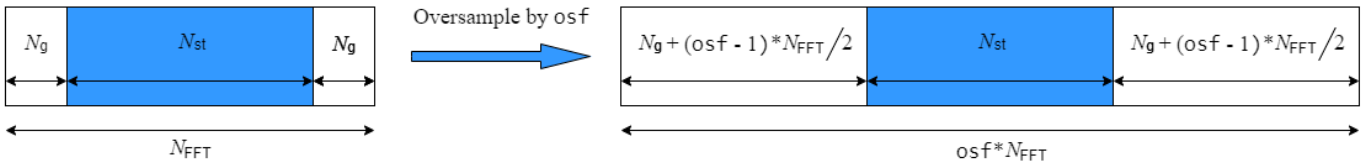
BCC channel coding is shown.

For algorithm details, refer to IEEE Std 802.11ac-2013 [1], Section 22.3.4.9 and 22.3.4.10, respectively, single user and multi-user.

FFT-Based Oversampling

An oversampled signal is a signal sampled at a frequency that is higher than the Nyquist rate. WLAN signals maximize occupied bandwidth by using small guardbands, which can pose problems for anti-imaging and anti-aliasing filters. Oversampling increases guardband width relative to the total signal bandwidth, thereby increasing the number of samples in the signal.

This function performs oversampling by using a larger IFFT and zero pad when generating an OFDM waveform. This diagram shows the oversampling process for an OFDM waveform with N_{FFT} subcarriers comprising N_g guardband subcarriers on either side of N_{st} occupied bandwidth subcarriers.



Version History

Introduced in R2015b

References

[1] IEEE Std 802.11ac™-2013 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications — Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanHTConfig` | `wlanVHTDataRecover` | `wlanWaveformGenerator`

wlanVHTDataRecover

Recover bits from VHT-Data field

Syntax

```
dataBits = wlanVHTDataRecover(rxDataSig,chEst,noiseVarEst,cfgVHT)
dataBits = wlanVHTDataRecover(rxDataSig,chEst,noiseVarEst,cfgVHT,userIdx)
dataBits = wlanVHTDataRecover(rxDataSig,chEst,noiseVarEst,cfgVHT,userIdx,
numSTS)
dataBits = wlanVHTDataRecover( ___ ,Name,Value)

[dataBits,crcBits] = wlanVHTDataRecover( ___ )
[dataBits,crcBits,eqSym] = wlanVHTDataRecover( ___ )
[dataBits,crcBits,eqSym,cpe] = wlanVHTDataRecover( ___ )
[dataBits,crcBits,eqSym,cpe,ae] = wlanVHTDataRecover( ___ )
```

Description

`dataBits = wlanVHTDataRecover(rxDataSig,chEst,noiseVarEst,cfgVHT)` recovers `dataBits`, a column vector of bits, from `rxDataSig`, the received VHT-Data field of a very-high-throughput (VHT) single-user transmission. The function recovers `dataBits` by using `chEst`, a channel estimate for the occupied subcarriers, `noiseVarEst`, an estimate of noise variance, and `cfgVHT`, a configuration object that contains VHT transmission parameters.

For more information about the VHT-Data field, see “VHT-Data Field” on page 3-480.

`dataBits = wlanVHTDataRecover(rxDataSig,chEst,noiseVarEst,cfgVHT,userIdx)` recovers `dataBits` for one user, specified by user index `userIdx`, in a VHT multi-user transmission.

`dataBits = wlanVHTDataRecover(rxDataSig,chEst,noiseVarEst,cfgVHT,userIdx, numSTS)` recovers `dataBits` for one user in a VHT multi-user transmission for `numSTS`, the number of space-time streams in the transmission.

`dataBits = wlanVHTDataRecover( ___ ,Name,Value)` specifies algorithm options by using one or more name-value pair arguments, in addition to any input argument combination from previous syntaxes. For example, `'LDPCDecodingMethod','layered-bp'` specifies the layered belief propagation low-density parity-check (LDPC) decoding algorithm.

`[dataBits,crcBits] = wlanVHTDataRecover( ___ )` returns the VHT-SIG-B checksum bits, `crcBits`, using any input argument combination from previous syntaxes.

`[dataBits,crcBits,eqSym] = wlanVHTDataRecover( ___ )` returns `eqSym`, the equalized OFDM symbols that comprise the data subcarriers of the VHT-Data field, using any input argument combination from the previous syntaxes.

`[dataBits,crcBits,eqSym,cpe] = wlanVHTDataRecover( ___ )` returns `cpe`, the common phase error between the received and expected OFDM symbols, using any input argument combination from the previous syntaxes.

`[dataBits,crcBits,eqSym,cpe,ae] = wlanVHTDataRecover( ___ )` returns `ae`, the average amplitude error with respect to the estimated received pilots.

Examples

Recover Bits from VHT Signal Transmitted Through 2x2 Fading Channel

Recover bits from the VHT-Data field of a VHT waveform transmitted though a 2x2 fading channel by using channel estimation on the VHT long training field (VHT-LTF).

Configure a VHT transmission with a channel bandwidth of 160 MHz, two transmit antennas, and two transmission paths.

```
cfgVHT = wlanVHTConfig('ChannelBandwidth','CBW160','NumTransmitAntennas',2,'NumSpaceTimeStreams'
```

Generate VHT-LTF and VHT-Data fields signals.

```
psduLength = 8*cfgVHT.PSDULength;
bits = randi([0 1],psduLength,1);
txLTF  = wlanVHTLTF(cfgVHT);
txDataSig = wlanVHTData(bits,cfgVHT);
```

Pass the transmitted waveform through a 2x2 quasi-static fading channel with additive white Gaussian noise (AWGN).

```
snr = 10;
H = complex(randn(2,2),randn(2,2))/sqrt(2);
rxLTF  = awgn(txLTF*H,snr);
rxDataSig = awgn(txDataSig*H,snr);
```

Calculate the received signal power and estimate the noise variance.

```
powerDB = 10*log10(var(rxDataSig));
noiseVarEst = mean(10.^(0.1*(powerDB-snr)));
```

Perform channel estimation based on the VHT-LTF.

```
sym = wlanVHTLTFDemodulate(rxLTF,cfgVHT,1);
chEst = wlanVHTLTFChannelEstimate(sym,cfgVHT);
```

Recover the bits from the received VHT-Data field and confirm that the received bits match the transmitted bits.

```
dataBits = wlanVHTDataRecover(rxDataSig,chEst,noiseVarEst,cfgVHT);
numErr = biterr(bits,dataBits)
```

```
numErr = 0
```

Recover Bits from VHT Signal Transmitted through MU-MIMO Channel

Recover bits from the VHT-Data field of a VHT multi-user transmission recovered from a fading MU-MIMO channel by using channel estimation on the VHT-LTF.

This example can return high bit error rates because the transmission does not include precoding to mitigate the interference between space-time streams. However, the example shows a typical VHT signal recovery workflow and appropriate syntax use for the functions involved.

Configure a VHT transmission with a channel bandwidth of 160 MHz, two users, and four transmit antennas. Assign one space-time stream to the first user and three space-time streams to the second user.

```
cbw = 'CBW160';
numSTS = [1 3];
cfgVHT = wlanVHTConfig('ChannelBandwidth',cbw,'NumUsers',2, ...
    'NumTransmitAntennas',4,'NumSpaceTimeStreams',numSTS);
```

Generate a payload of bits for each user. This payload must be in the a 1-by-*N* cell array, where *N* is the number of users.

```
psduLength = 8*cfgVHT.PSDULength;
numUsers = cfgVHT.NumUsers;
bits = cell(1,2);
for nu = 1:numUsers
    bits{nu} = randi([0 1],psduLength(nu),1);
end
```

Generate VHT-LTF and VHT-Data field signals.

```
txLTF  = wlanVHTLTF(cfgVHT);
txDataSym = wlanVHTData(bits,cfgVHT);
```

Pass the VHT-Data field signal for the first user through a 4x1 channel because this signal consists of a single space-time stream. Pass the VHT-Data field for the second user data through a 4x3 channel because this signal consists of three space-time streams. Apply AWGN to each signal, assuming an SNR of 15 dB.

```
snr = 15;
H{1} = complex(randn(4,1),randn(4,1))/sqrt(2);
H{2} = complex(randn(4,3),randn(4,3))/sqrt(2);
number = zeros(2,1);
ratio = zeros(2,1);
for userIdx = 1:numUsers
    rxDataSym = awgn(txDataSym*H{userIdx},snr,'measured');
```

Apply the same channel processing to the VHT-LTF for each user.

```
    rxLTF = awgn(txLTF*H{userIdx},snr,'measured');
```

Calculate the received signal power for each user and estimate the noise variance.

```
    powerDB = 10*log10(var(rxDataSym));
    noiseVarEst = mean(10.^(0.1*(powerDB-snr)));
```

Estimate the channel characteristics by using the VHT-LTF.

```
    demod = wlanVHTLTFDemodulate(rxLTF,cbw,numSTS);
    chEst = wlanVHTLTFChannelEstimate(demod,cbw,numSTS);
```

Recover the bits from the received VHT-Data field for each user and determine the bit error rate by comparing the recovered bits with the original payload bits.

```
    dataBits = wlanVHTDataRecover(rxDataSym,chEst,noiseVarEst,cfgVHT,userIdx);
    [number(userIdx),ratio(userIdx)] = biterr(bits{userIdx},dataBits);
    disp(number(userIdx))
    disp(ratio(userIdx))
end
```

```
        4269

    0.5082

        2444

    0.0968
```

Recover Bits from VHT-Data Field Using Zero-Forcing Equalization

Recover bits from the VHT-Data field signal of a VHT transmission recovered from a SISO AWGN channel by using a zero-forcing equalization algorithm.

Configure a VHT transmission and generate the VHT-Data field for a random payload of bits.

```
cfgVHT = wlanVHTConfig('APEPLength',512);
psduLength = 8*cfgVHT.PSDULength;
bits = randi([0 1],psduLength,1);
txDataSig = wlanVHTData(bits,cfgVHT);
```

Pass the transmission through an AWGN channel.

```
snr = 10;
rxDataSig = awgn(txDataSig,snr);
```

Recover the payload bits using a perfect channel estimate of all ones and zero-forcing equalization.

```
chEst = ones(242,1);
noiseVarEst = 10^(-snr/10);
[dataBits,crcBits,eqSym,cpe] = wlanVHTDataRecover(rxDataSig,chEst,noiseVarEst,cfgVHT,'Equalizati
```

Verify that the recovered signal contains no bit errors.

```
number = biterr(bits,dataBits)
```

```
number = 0
```

Display the VHT-Data field CRC checksum bits.

```
disp(crcBits')
```

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

Calculate and display the maximum common phase error.

```
max(abs(cpe))
```

```
ans = 0.2828
```

Recover VHT-Data Field and Calculate Amplitude Error

Investigate how applying an amplitude droop affects the amplitude error of the VHT-Data field generated from a VHT signal.

Configure a VHT transmission with default parameters, then generate the corresponding VHT-Data field.

```
cfgVHT = wlanVHTConfig;
psduLength = 8*cfgVHT.PSDULength;
bits = randi([0 1],psduLength,1);
tx = wlanVHTData(bits,cfgVHT);
```

Modify the signal by applying an amplitude droop of 10 dB, starting at the halfway point.

```
signalLength = size(tx,1);
droopGain = 10;
droopGainLinear = 10^(droopGain/20);
txDroop = [ones(signalLength/2,1); droopGainLinear*ones(signalLength/2,1)].*tx;
```

Specify a channel estimate.

```
chEst = ones(242,1);
```

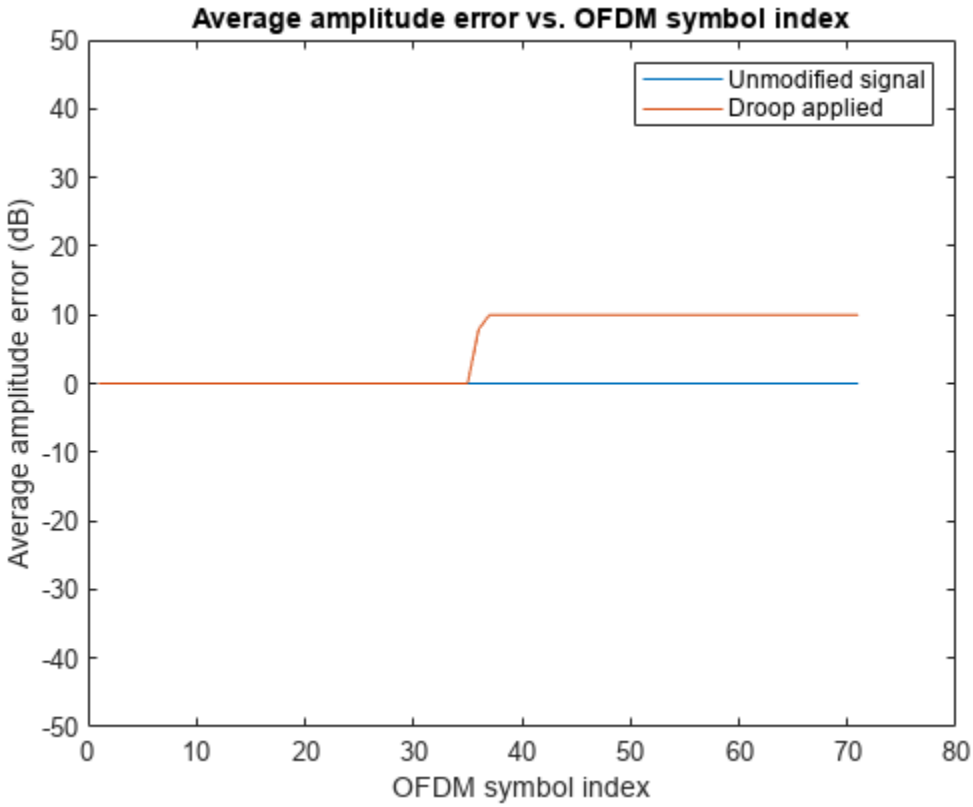
Recover the bits from the ideal and impaired VHT-Data fields and confirm that the recovered bits match the transmitted bits.

```
[databits_1, crcBits_1, eqSym_1, cpe_1, ae_1] = wlanVHTDataRecover(tx, chEst, 0, cfgVHT, Equaliza
[databits_2, crcBits_2, eqSym_2, cpe_2, ae_2] = wlanVHTDataRecover(txDroop, chEst, 0, cfgVHT, Equ
isequal(databits_1, databits_2, bits)
```

```
ans = logical
   1
```

Plot the absolute value of the measured amplitude errors for the ideal and impaired VHT-Data fields.

```
plot(abs(ae_1))
title('Average amplitude error vs. OFDM symbol index')
ylabel('Average amplitude error (dB)')
xlabel('OFDM symbol index')
ylim([-50 50])
hold on
plot(abs(ae_2))
legend('Unmodified signal', 'Droop applied')
```



Input Arguments

`rxDataSig` — Received VHT-Data field

complex-valued array

Received VHT-Data field, specified as a complex-valued array of size N_S-by-N_R.

- N_S is an integer greater than or equal to the number of time-domain samples.
- N_R is the number of receive antennas.

Note The function processes one PPDU data field per entry. If you specify N_S as a value greater than the field length, the function does not process additional samples at the end of `rxDataSig`. To process a concatenated stream of PPDU data fields, you must call the function multiple times.

Data Types: `double`
Complex Number Support: Yes

`chEst` — Channel estimate for occupied subcarriers

complex-valued array

Channel estimate for occupied subcarriers, specified as a complex-valued array of size N_{ST}-by-N_{STS}-by-N_R.

- N_{ST} is the number of occupied subcarriers, which depends on the `ChannelBandwidth` property of the `cfgVHT` input in accordance with this table.

Value of `ChannelBandwidth` Property	Value of N_{ST}
`'CBW20'`	56
`'CBW40'`	114
`'CBW80'`	242
`'CBW160'`	484

- N_{STS} is the number of space-time streams, which must match the `NumSpacetimeStreams` property of the `cfgVHT` input. For multi-user transmissions, N_{STS} is the total number of space-time streams for all users.
- N_R is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

`noiseVarEst` — Noise variance estimate
nonnegative scalar

Noise variance estimate, specified as a nonnegative scalar.

Data Types: `double`

`cfgVHT` — VHT transmission configuration
`wlanVHTConfig` object

VHT transmission configuration, specified as a `wlanVHTConfig` object.

`userIdx` — User index
integer in the interval [1, N_{Users}]

User index, specified as an integer in the interval [1, N_{Users}], where N_{Users} is the total number of users in the transmission.

`numSTS` — Number of space-time streams
integer in the interval [1, 4] | row vector of integers in the interval [1, 4]

Number of space-time streams.

- For a single-user transmission, specify this input as an integer in the interval [1, 4]
- for a multi-user transmission, specify this input as a row vector of integers in the interval [1, 4] of length N_{Users}, where N_{Users} is the total number of users in the transmission.

Example: `[1 3 2]` indicates the number of space-time streams in a three-user transmission. In this case, the transmission allocates one, three, and two space-time streams to the first, second, and third users, respectively.

Note The sum of the elements of this property must not exceed eight.

Data Types: `double`

Name-Value Pair Arguments

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

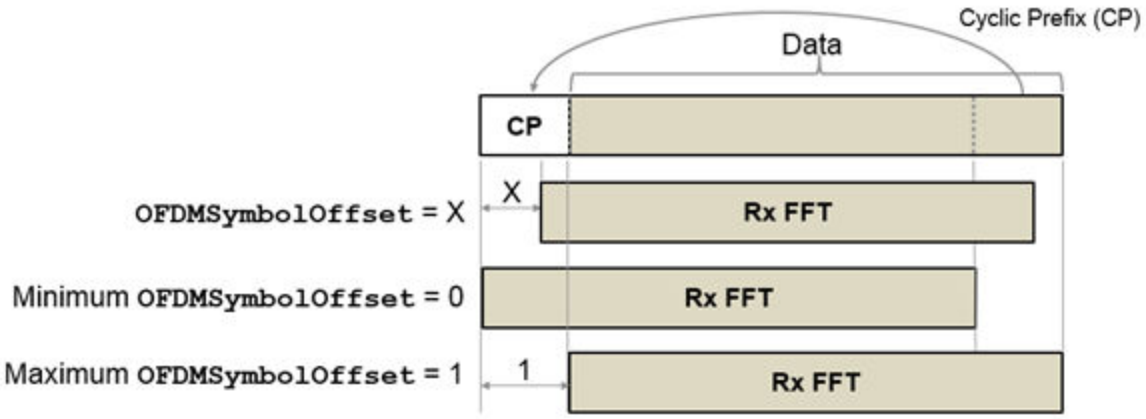
Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'PilotPhaseTracking','None'` disables pilot phase tracking.

`OFDMSymbolOffset` — OFDM symbol sampling offset

`0.75` (default) | scalar in the interval [0, 1]

OFDM symbol sampling offset represented as a fraction of the cyclic prefix (CP) length, specified as the comma-separated pair consisting of `'OFDMSymbolOffset'` and a scalar in the interval [0, 1]. The value you specify indicates the start location for OFDM demodulation relative to the beginning of the CP. The value `0` represents the start of the CP, and the value `1` represents the end of the CP.



Data Types: `double`

`EqualizationMethod` — Equalization method

`'MMSE'` (default) | `'ZF'`

Equalization method, specified as the comma-separated pair consisting of `'EqualizationMethod'` and one of these values.

- `'MMSE'` — The receiver uses a minimum mean-square error equalizer.
- `'ZF'` — The receiver uses a zero-forcing equalizer.

When the received signal has multiple receive antennas, the function exploits receiver diversity during equalization. When the number of transmitted space-time streams is one and you specify this argument as `'ZF'`, the function performs maximal-ratio combining.

Note Specify this argument as `'ZF'` when either of these conditions applies.

- The `NumSpaceTimeStreams` property of the `cfgVHT` input is `1`.
- The `NumSpaceTimeStreams` and `STBC` properties of the `cfgVHT` input are `2` and `1` (`true`), respectively.

Data Types: `char` | `string`

`PilotPhaseTracking` — Pilot phase tracking
`'PreEQ'` (default) | `'None'`

Pilot phase tracking, specified as the comma-separated pair consisting of `'PilotPhaseTracking'` and one of these values.

- `'PreEQ'` — Enable pilot phase tracking, which the function performs before any equalization operation.
- `'None'` — Disable pilot phase tracking.

Data Types: `char` | `string`

`PilotAmplitudeTracking` — Pilot amplitude tracking
`'None'` (default) | `'PreEQ'`

Pilot amplitude tracking, specified as the comma-separated pair consisting of `'PilotAmplitudeTracking'` and one of these values.

- `'None'` — Disable pilot amplitude tracking.
- `'PreEQ'` — Enable pilot amplitude tracking, which the function performs before any equalization operation.

Note Due to the limitations of the algorithm used, disable pilot amplitude tracking when filtering a waveform through a MIMO fading channel.

Data Types: `char` | `string`

`LDPCDecodingMethod` — LDPC decoding algorithm
`'bp'` (default) | `'layered-bp'` | `'norm-min-sum'` | `'offset-min-sum'`

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

- `'bp'` — Use the belief propagation (BP) decoding algorithm. For more information, see “Belief Propagation Decoding” on page 3-481.
- `'layered-bp'` — Use the layered BP decoding algorithm, suitable for quasi-cyclic parity check matrices (PCMs). For more information, see “Layered Belief Propagation Decoding” on page 3-482.
- `'norm-min-sum'` — Use the layered BP decoding algorithm with the normalized min-sum approximation. For more information, see “Normalized Min-Sum Decoding” on page 3-482.
- `'offset-min-sum'` — Use the layered BP decoding algorithm with the offset min-sum approximation. For more information, see “Offset Min-Sum Decoding” on page 3-482.

Note When you specify this input as `'norm-min-sum'` or `'offset-min-sum'`, the function sets input log-likelihood ratio (LLR) values that are greater than `1e10` or less than `-1e10` to `1e10` and `-1e10`, respectively. The function then uses these values when executing the LDPC decoding algorithm.

Dependencies

To enable this argument, set the `ChannelCoding` property of the `cfgVHT` input to `'LDPC'` for the user corresponding to the `userIdx` input.

Data Types: `char` | `string`

`MinSumScalingFactor` — Scaling factor for normalized min-sum LDPC decoding

`0.75` (default) | scalar in the interval (0, 1]

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

Dependencies

To enable this argument, specify the `'LDPCDecodingMethod'` name-value pair argument as `'norm-min-sum'`.

Data Types: `double`

`MinSumOffset` — Offset for offset min-sum LDPC decoding

`0.5` (default) | nonnegative scalar

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

Dependencies

To enable this argument, specify the `'LDPCDecodingMethod'` name-value pair argument as `'offset-min-sum'`.

Data Types: `double`

`MaximumLDPCIterationCount` — Maximum number of LDPC decoding iterations

`12` (default) | positive integer

Maximum number of LDPC decoding iterations, specified as the comma-separated pair consisting of `'MaximumLDPCIterationCount'` and a positive integer.

Dependencies

To enable this argument, set the `ChannelCoding` property of the `cfgVHT` input to `'LDPC'` for the user corresponding to the `userIdx` input.

Data Types: `double`

`EarlyTermination` — Enable early termination of LDPC decoding

`false` or `0` (default) | `true` or `1`

Enable early termination of LDPC decoding, specified as the comma-separated pair consisting of `'EarlyTermination'` and `1` (`true`) or `0` (`false`).

- When you set this value to `0` (`false`), LDPC decoding completes the number of iterations specified by `'MaximumLDPCIterationCount'` regardless of parity check status.
- When you set this value to `1` (`true`), LDPC decoding terminates when all parity checks are satisfied.

Dependencies

To enable this argument, set the `ChannelCoding` property of the `cfgVHT` input to `'LDPC'` for the user corresponding to the `userIdx` input.

Data Types: `logical`

Output Arguments

`dataBits` — Bits recovered from VHT-Data field

binary-valued column vector

Bits recovered from the VHT-Data field, returned as a column vector of length $8 \times L_{\mathrm{PSDU}}$, where L_{PSDU} is the length of the PSDU in bytes.

Data Types: `int8`

`crcBits` — VHT-SIG-B checksum bits

binary-valued column vector

VHT-SIG-B checksum bits, returned as a binary-valued column vector of length 8.

Data Types: `int8`

`eqSym` — Equalized OFDM symbols

complex-valued array

Equalized OFDM symbols comprising the VHT-Data field, returned as a complex-valued array of size N_{SD}-by-N_{Sym}-by-N_{SS}.

- N_{SD} is the number of data subcarriers.
- N_{Sym} is the number of OFDM symbols in the VHT-Data field.
- N_{SS} is the number of spatial streams. When the `STBC` property of the `cfgVHT` input is `0` (`false`), N_{SS} is equal to N_{STS}, the number of space-time streams in the transmission. When the `STBC` property of the `cfgVHT` input is `0` (`false`), N_{SS} is equal to $N_{\mathrm{STS}}/2$.

Data Types: `double`
Complex Number Support: Yes

`cpe` — Common phase error

real-valued column vector

Common phase error between the received and expected OFDM symbols, in radians, returned as a real-valued column vector. The length of this output is N_{Sym}, the number of OFDM symbols in the VHT-Data field. This output is averaged over the receive antennas.

Data Types: `double`

`ae` — Average amplitude error

real-valued array

Average amplitude error, in dB, returned as a real-valued array of size N_{Sym}-by- N_{R}.

- N_{Sym} is the number of OFDM symbols in the VHT-Data field.
- N_{R} is the number of receive antennas.

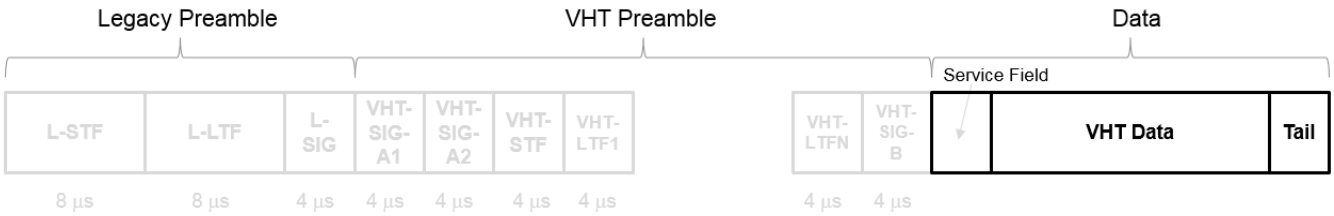
Each element of this matrix contains the amplitude error for all subcarriers with respect to the estimated received pilots for the corresponding OFDM symbol and receive antenna.

Data Types: `double`

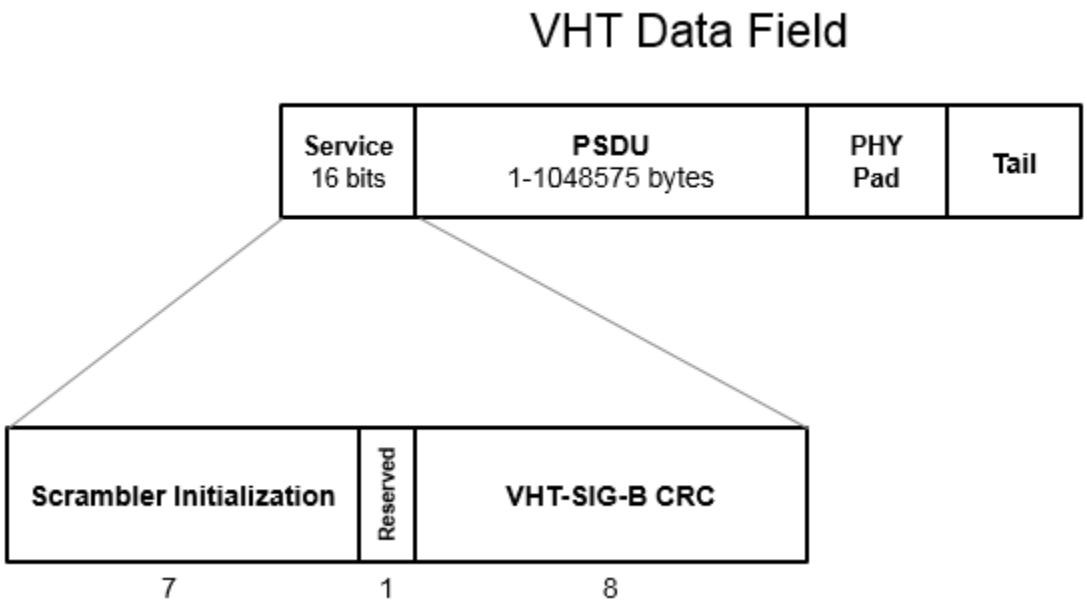
More About

VHT-Data Field

The VHT-Data field carries one or more frames from the medium access control (MAC) layer. This field follows the VHT-SIG-B field in a VHT PPDUs.



For a detailed description of the VHT-Data field, see section 21.3.10 of IEEE Std 802.11-2016. The VHT Data field consists of four subfields.



- **Service field** — Contains a seven-bit scrambler initialization state, one bit reserved for future considerations, and eight bits for the VHT-SIG-B cyclic redundancy check (CRC) field
- **PSDU** — Variable-length field containing a PLCP service data unit
- **PHY Pad** — Variable number of bits passed to the transmitter to create a complete OFDM symbol
- **Tail** — Bits required to terminate a convolutional code (not required when the transmission uses LDPC channel coding)

Algorithms

This function supports these four LDPC decoding algorithms.

Belief Propagation Decoding

The function implements the BP algorithm based on the decoding algorithm presented in [2]. For transmitted LDPC-encoded codeword $c = (c_0, c_1, \ldots, c_{n-1})$, the input to the LDPC decoder is the LLR given by

$$L(c_i) = \log\left(\frac{\Pr(c_i = 0 \,|\, \text{channel output for } c_i)}{\Pr(c_i = 0 \,|\, \text{channel output for } c_i)}\right).$$

In each iteration, the function updates the key components of the algorithm based on these equations:

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

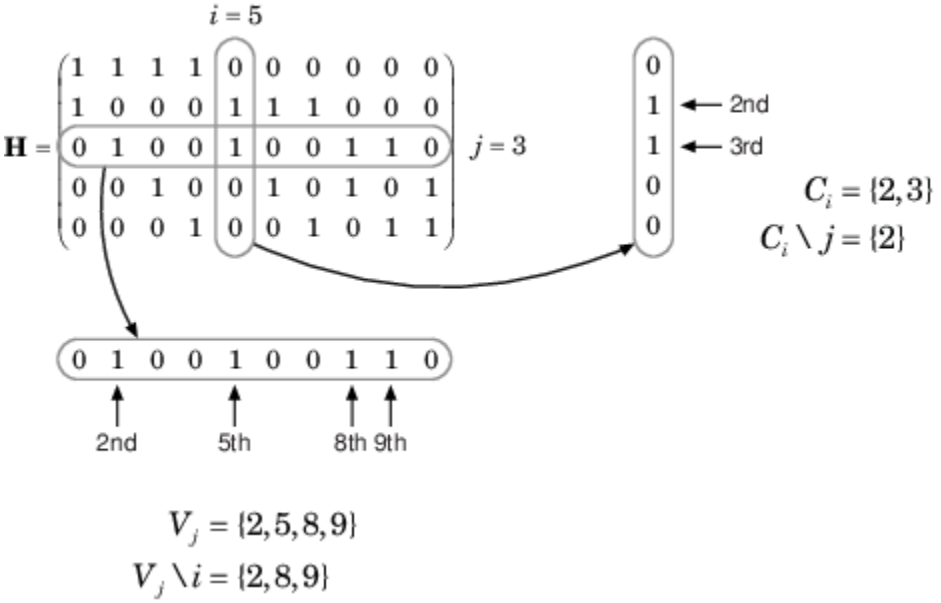
$L(q_{ij}) = L(c_i) + \sum_{j' \in C_i \backslash \{j\}} L(r_{j'i})$, initialized as $L(q_{ij}) = L(c_i)$ before the first iteration, and

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

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

Index sets $C_i \backslash \{j\}$ and $V_j \backslash \{i\}$ are based on the PCM such that the sets C_i and V_j correspond to all nonzero elements in column i and row j of the PCM, respectively.

This figure demonstrates how to compute these index sets for PCM **H** for the case $i = 5$ and $j = 3$.



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

When you specify the `'EarlyTermination'` name-value pair argument as `0` (`false`), the decoding terminates after the number of iterations specified by the `'MaximumLDPCIterationCount'` name-value pair argument. When you specify the `'EarlyTermination'` name-value pair argument as `1` (`true`), the decoding terminates when all parity checks are satisfied ($\mathbf{Hc}^T = 0$) or after the number of iterations specified by the `'MaximumLDPCIterationCount'` name-value pair argument.

Layered Belief Propagation Decoding

The function implements the layered BP algorithm based on the decoding algorithm presented in Section II.A of [3]. The decoding loop iterates over subsets of rows (layers) of the PCM.

For each row, m, in a layer and each bit index, j, the implementation updates the key components of the algorithm based on these equations.

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

(2) $\Psi(x) = \log(|\tanh(x/2)|)$

(3) $A_{mj} = \sum_{n \in N(m)\backslash\{j\}} \Psi(L(q_{mn}))$

(4) $s_{mj} = \prod_{n \in N(m)\backslash\{j\}} \mathrm{sgn}(L(q_{mn}))$

(5) $R_{mj} = -s_{mj}\Psi(A_{mj})$

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

For each layer, the decoding equation (6) works on the combined input obtained from the current LLR inputs, $L(q_{mj})$, and the previous layer updates, R_{mj}.

Because the layered BP algorithm updates only a subset of the nodes in a layer, this algorithm is faster than the BP algorithm. To achieve the same error rate as attained with BP decoding, use half the number of decoding iterations when using the layered BP algorithm.

Normalized Min-Sum Decoding

The function implements the normalized min-sum decoding algorithm by following the layered BP algorithm with equation (3) replaced by

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

where α is the scaling factor specified by the `'MinSumScalingFactor'` name-value pair argument. This equation is an adaptation of equation (4) presented in [4].

Offset Min-Sum Decoding

The function implements the offset min-sum decoding algorithm by following the layered BP algorithm with equation (3) replaced by

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

where β is the offset specified by the `'MinSumOffset'` name-value pair argument. This equation is an adaptation of equation (5) presented in [4].

Version History

Introduced in R2015b

Pilot amplitude tracking

You can perform pilot amplitude tracking by setting the `PilotAmplitudeTracking` input argument to `'PreEQ'`. The average amplitude error is returned in the output argument `ae`.

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

[2] Gallager, Robert G. *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.

[3] Hocevar, D.E. "A Reduced Complexity Decoder Architecture via Layered Decoding of LDPC Codes." In *IEEE Workshop on Signal Processing Systems, 2004. SIPS 2004.*, 107-12. Austin, Texas, USA: IEEE, 2004. https://doi.org/10.1109/SIPS.2004.1363033.

[4] Jinghu Chen, R.M. Tanner, C. Jones, and Yan Li. "Improved Min-Sum Decoding Algorithms for Irregular LDPC Codes." In *Proceedings. International Symposium on Information Theory, 2005. ISIT 2005.*, 449-53, 2005. https://doi.org/10.1109/ISIT.2005.1523374.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanVHTConfig` | `wlanVHTData` | `wlanVHTLTFDemodulate` | `wlanVHTLTFChannelEstimate` | `wlanVHTSIGARecover` | `wlanVHTSIGBRecover`

wlanVHTLTF

Generate VHT-LTF waveform

Syntax

```
y = wlanVHTLTF(cfg)
y = wlanVHTLTF(cfg,OversamplingFactor=osf)
```

Description

`y = wlanVHTLTF(cfg)` generates a "VHT-LTF" on page 3-485 [19] time-domain waveform for the specified transmission parameters. See "VHT-LTF Processing" on page 3-486 for waveform generation details.

`y = wlanVHTLTF(cfg,OversamplingFactor=osf)` generates an oversampled VHT-LTF waveform with the specified oversampling factor. For more information about oversampling, see "FFT-Based Oversampling" on page 3-486.

Examples

Generate VHT-LTF Waveform

Create a VHT configuration object with an 80 MHz channel bandwidth.

```
cfgVHT = wlanVHTConfig;
cfgVHT.ChannelBandwidth = 'CBW80';
```

Generate a VHT-LTF waveform.

```
vltfOut = wlanVHTLTF(cfgVHT);
size(vltfOut)
```

```
ans = 1×2

   320     1
```

The 80 MHz waveform is a single OFDM symbol with 320 complex output samples.

Input Arguments

`cfg` — Transmission parameters

`wlanVHTConfig` object

Transmission parameters, specified as a `wlanVHTConfig` object.

19 IEEE Std 802.11ac-2013 Adapted and reprinted with permission from IEEE. Copyright IEEE 2013. All rights reserved.

`osf` — Oversampling factor

1 (default) | scalar greater than or equal to 1

Oversampling factor, specified as a scalar greater than or equal to 1. The oversampled cyclic prefix length must be an integer number of samples.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

y — VHT-LTF time-domain waveform

matrix

"VHT-LTF" on page 3-485 time-domain waveform, returned as an ($N_S \times N_{VHTLTF}$)-by-N_T matrix. N_S is the number of time-domain samples per N_{VHTLTF}, where N_{VHTLTF} is the number of OFDM symbols in the VHT-LTF. N_T is the number of transmit antennas.

N_S is proportional to the channel bandwidth.

`ChannelBandwidth`	N_S
`'CBW20'`	80
`'CBW40'`	160
`'CBW80'`	320
`'CBW160'`	640

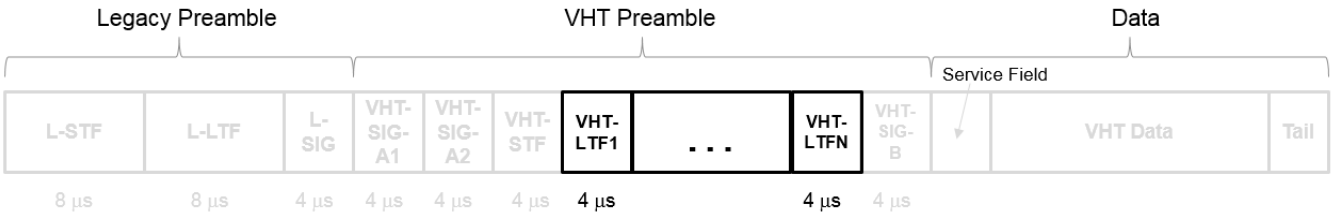
See "VHT-LTF Processing" on page 3-486 for waveform generation details.

Data Types: `double`
Complex Number Support: Yes

More About

VHT-LTF

The very high throughput long training field (VHT-LTF) is located between the VHT-STF and VHT-SIG-B portion of the VHT packet.



It is used for MIMO channel estimation and pilot subcarrier tracking. The VHT-LTF includes one VHT long training symbol for each spatial stream indicated by the selected MCS. Each symbol is 4 µs long. A maximum of eight symbols are permitted in the VHT-LTF.

For a detailed description of the VHT-LTF, see section 21.3.8.3.5 of IEEE Std 802.11-2016.

Algorithms

VHT-LTF Processing

The “VHT-LTF” on page 3-485 is used for MIMO channel estimation and pilot subcarrier tracking. The number of OFDM symbols in the “VHT-LTF” on page 3-485 (N_{VHTLTF}) is derived from the total number of space-time streams (N_{STS_Total}). $N_{STS_Total} = \Sigma N_{STS}(u)$ for user u, $u = 0,\ldots, N_{Users}–1$ and $N_{STS}(u)$ is the number of space-time streams per user.

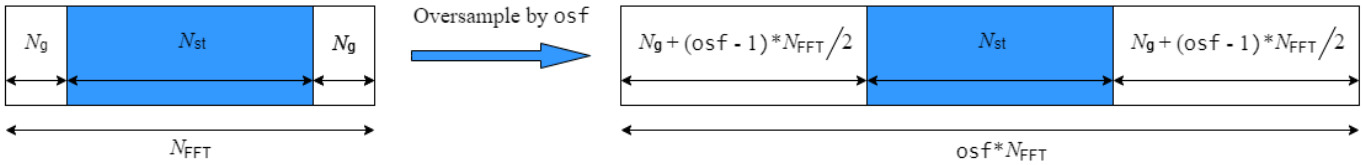
N_{STS_Total}	N_{VHTLTF}
1	1
2	2
3	4
4	4
5	6
6	6
7	8
8	8

For algorithm details refer to IEEE Std 802.11ac-2013 [1], Section 22.3.4.7.

FFT-Based Oversampling

An oversampled signal is a signal sampled at a frequency that is higher than the Nyquist rate. WLAN signals maximize occupied bandwidth by using small guardbands, which can pose problems for anti-imaging and anti-aliasing filters. Oversampling increases guardband width relative to the total signal bandwidth, thereby increasing the number of samples in the signal.

This function performs oversampling by using a larger IFFT and zero pad when generating an OFDM waveform. This diagram shows the oversampling process for an OFDM waveform with N_{FFT} subcarriers comprising N_{g} guardband subcarriers on either side of N_{st} occupied bandwidth subcarriers.



Version History

Introduced in R2015b

References

[1] IEEE Std 802.11ac™-2013 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer

(PHY) Specifications — Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanVHTConfig` | `wlanLLTF` | `wlanVHTSTF` | `wlanVHTData` | `wlanVHTLTFDemodulate` | `wlanVHTLTFChannelEstimate`

wlanVHTLTFChannelEstimate

Channel estimation using VHT-LTF

Syntax

```
chEst = wlanVHTLTFChannelEstimate(demodSig,cfg)
chEst = wlanVHTLTFChannelEstimate(demodSig,cbw,numSTS)
chEst = wlanVHTLTFChannelEstimate( ___ ,span)
```

Description

`chEst = wlanVHTLTFChannelEstimate(demodSig,cfg)` returns the channel estimate, using the demodulated "VHT-LTF" on page 3-493[20] signal, `demodSig`, given the parameters specified in `wlanVHTConfig` object `cfg`.

`chEst = wlanVHTLTFChannelEstimate(demodSig,cbw,numSTS)` returns the channel estimate for the specified channel bandwidth, `cbw`, and the number of space-time streams, `numSTS`.

`chEst = wlanVHTLTFChannelEstimate( ___ ,span)` specifies the span of a moving-average filter used to perform frequency smoothing.

Examples

Estimate SISO Channel Using VHT-LTF

Display the channel estimate of the data and pilot subcarriers for a VHT format channel using its long training field.

Create a VHT format configuration object. Generate a VHT-LTF based on `cfg`.

```
cfg = wlanVHTConfig;
txSig = wlanVHTLTF(cfg);
```

Multiply the transmitted VHT-LTF signal by 0.3 - 0.15i and pass it through an AWGN channel having a 30 dB signal-to-noise ratio. Demodulate the received signal.

```
rxSig = awgn(txSig*(0.3-0.15i),30);
demodSig = wlanVHTLTFDemodulate(rxSig,cfg);
```

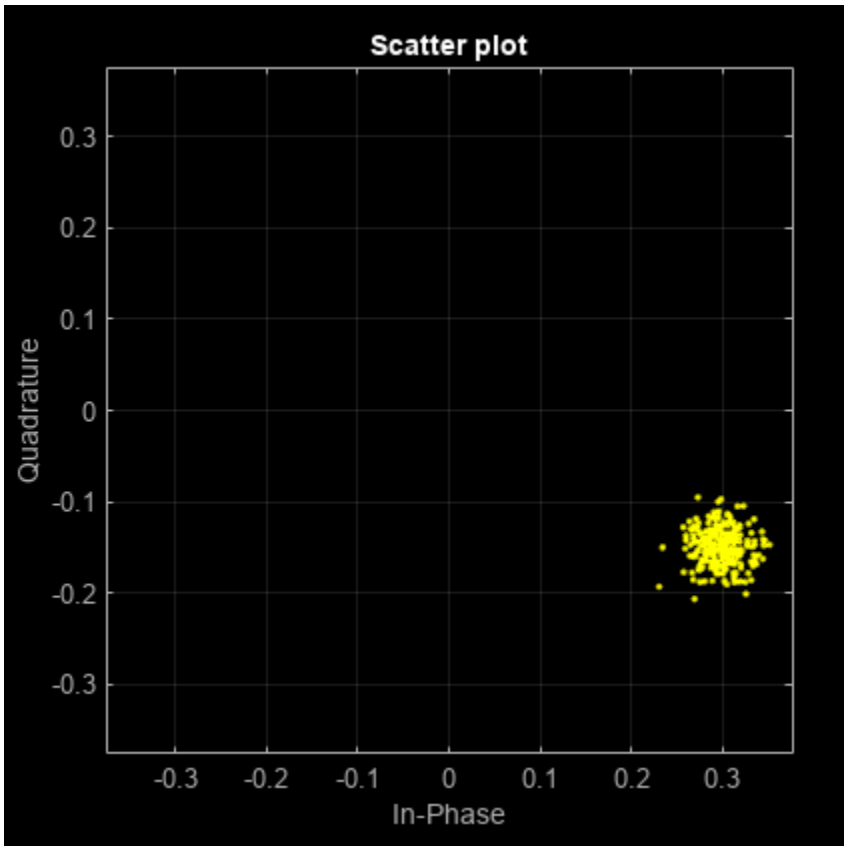
Estimate the channel response using the demodulated VHT-LTF signal.

```
est = wlanVHTLTFChannelEstimate(demodSig,cfg);
```

Plot the channel estimate.

```
scatterplot(est)
grid
```

20 IEEE Std 802.11ac-2013 Adapted and reprinted with permission from IEEE. Copyright IEEE 2013. All rights reserved.



The channel estimate matches the complex channel multiplier.

Estimate MIMO Channel Using VHT-LTF

Estimate and display the channel coefficients of a 4x2 MIMO channel using the VHT-LTF.

Create a VHT format configuration object for a channel having four spatial streams and four transmit antennas. Transmit a complete VHT waveform.

```
cfg = wlanVHTConfig('NumTransmitAntennas',4, ...
    'NumSpaceTimeStreams',4,'MCS',5);
txWaveform = wlanWaveformGenerator([1;0;0;1;1;0],cfg);
```

Set the sampling rate, and then pass the transmitted waveform through a 4x2 TGac channel.

```
fs = 80e6;
tgacChan = wlanTGacChannel('SampleRate',fs, ...
    'NumTransmitAntennas',4,'NumReceiveAntennas',2);
rxWaveform = tgacChan(txWaveform);
```

Determine the VHT-LTF field indices and demodulate the VHT-LTF from the received waveform.

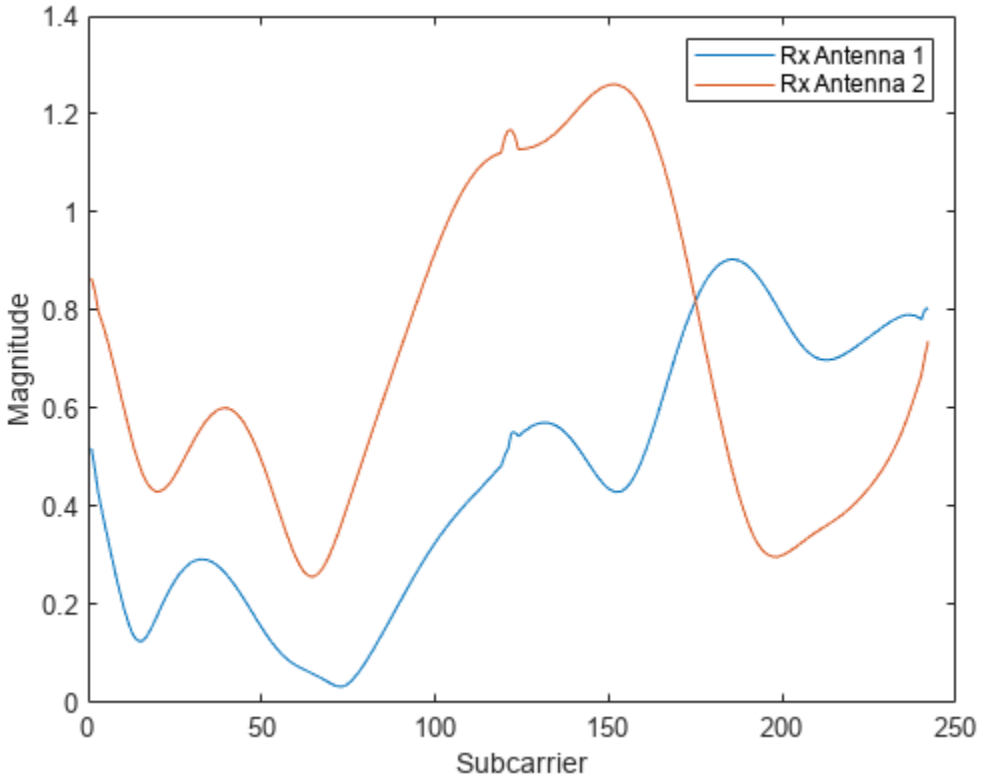
```
indVHTLTF = wlanFieldIndices(cfg,'VHT-LTF');
ltfDemodSig = wlanVHTLTFDemodulate(rxWaveform(indVHTLTF(1):indVHTLTF(2),:), cfg);
```

Generate the channel estimate by using the demodulated VHT-LTF signal. Specify a smoothing filter span of five subcarriers.

```
est = wlanVHTLTFChannelEstimate(ltfDemodSig,cfg,5);
```

Plot the magnitude response of the first space-time stream for both receive antennas. Due to the random nature of the fading channel, your results may vary.

```
plot(abs(est(:,1,1)))
hold on
plot(abs(est(:,1,2)))
xlabel('Subcarrier')
ylabel('Magnitude')
legend('Rx Antenna 1','Rx Antenna 2')
```



Recover Bits from VHT Signal Transmitted through MU-MIMO Channel

Recover bits from the VHT-Data field of a VHT multi-user transmission recovered from a fading MU-MIMO channel by using channel estimation on the VHT-LTF.

This example can return high bit error rates because the transmission does not include precoding to mitigate the interference between space-time streams. However, the example shows a typical VHT signal recovery workflow and appropriate syntax use for the functions involved.

Configure a VHT transmission with a channel bandwidth of 160 MHz, two users, and four transmit antennas. Assign one space-time stream to the first user and three space-time streams to the second user.

```
cbw = 'CBW160';
numSTS = [1 3];
cfgVHT = wlanVHTConfig('ChannelBandwidth',cbw,'NumUsers',2, ...
    'NumTransmitAntennas',4,'NumSpaceTimeStreams',numSTS);
```

Generate a payload of bits for each user. This payload must be in the a 1-by-*N* cell array, where *N* is the number of users.

```
psduLength = 8*cfgVHT.PSDULength;
numUsers = cfgVHT.NumUsers;
bits = cell(1,2);
for nu = 1:numUsers
    bits{nu} = randi([0 1],psduLength(nu),1);
end
```

Generate VHT-LTF and VHT-Data field signals.

```
txLTF  = wlanVHTLTF(cfgVHT);
txDataSym = wlanVHTData(bits,cfgVHT);
```

Pass the VHT-Data field signal for the first user through a 4x1 channel because this signal consists of a single space-time stream. Pass the VHT-Data field for the second user data through a 4x3 channel because this signal consists of three space-time streams. Apply AWGN to each signal, assuming an SNR of 15 dB.

```
snr = 15;
H{1} = complex(randn(4,1),randn(4,1))/sqrt(2);
H{2} = complex(randn(4,3),randn(4,3))/sqrt(2);
number = zeros(2,1);
ratio = zeros(2,1);
for userIdx = 1:numUsers
    rxDataSym = awgn(txDataSym*H{userIdx},snr,'measured');
```

Apply the same channel processing to the VHT-LTF for each user.

```
    rxLTF = awgn(txLTF*H{userIdx},snr,'measured');
```

Calculate the received signal power for each user and estimate the noise variance.

```
    powerDB = 10*log10(var(rxDataSym));
    noiseVarEst = mean(10.^(0.1*(powerDB-snr)));
```

Estimate the channel characteristics by using the VHT-LTF.

```
    demod = wlanVHTLTFDemodulate(rxLTF,cbw,numSTS);
    chEst = wlanVHTLTFChannelEstimate(demod,cbw,numSTS);
```

Recover the bits from the received VHT-Data field for each user and determine the bit error rate by comparing the recovered bits with the original payload bits.

```
    dataBits = wlanVHTDataRecover(rxDataSym,chEst,noiseVarEst,cfgVHT,userIdx);
    [number(userIdx),ratio(userIdx)] = biterr(bits{userIdx},dataBits);
    disp(number(userIdx))
    disp(ratio(userIdx))
end
```

```
        4269

    0.5082

        2444

    0.0968
```

Input Arguments

`demodSig` — Demodulated VHT-LTF signal

3-D array

Demodulated VHT-LTF signal, specified as an N_{ST}-by-N_{SYM}-by-N_R array. N_{ST} is the number of occupied subcarriers, N_{SYM} is the number of VHT-LTF OFDM symbols, and N_R is the number of receive antennas.

Data Types: `double`

`cfg` — Format configuration

`wlanVHTConfig`

Format configuration, specified as a `wlanVHTConfig` object.

`cbw` — Channel bandwidth

`'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth, specified as `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`. If the transmission has multiple users, the same channel bandwidth is applied to all users.

Data Types: `char` | `string`

`numSTS` — Number of space-time streams

1-by-N_{Users} vector of integers from 1 to 4

Number of space-time streams in the transmission, specified as a scalar or vector.

- For a single user, the number of space-time streams is a scalar integer from 1 to 8.
- For multiple users, the number of space-time streams is a 1-by-N_{Users} vector of integers from 1 to 4, where the vector length, N_{Users}, is an integer from 1 to 4.

Example: `[1 3 2]` indicates that one space-time stream is assigned to user 1, three space-time streams are assigned to user 2, and two space-time streams are assigned to user 3.

Note The sum of the space-time stream vector elements must not exceed eight.

Data Types: `double`

`span` — Filter span

positive odd integer

Filter span of the frequency smoothing filter, specified as a positive odd integer and expressed as a number of subcarriers. Frequency smoothing is applied only when `span` is specified and greater than one. See “Frequency Smoothing” on page 3-493.

Data Types: `double`

Output Arguments

`chEst` — Channel estimate

3-D array

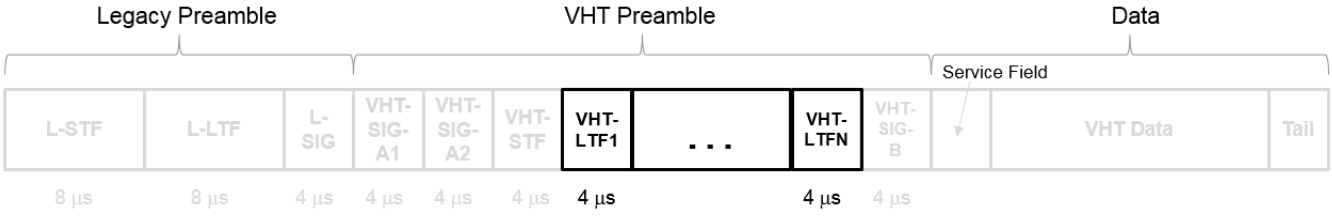
Channel estimate between all combinations of space-time streams and receive antennas, returned as an N_{ST}-by-$N_{STS,total}$-by-N_R array. N_{ST} is the number of occupied subcarriers. $N_{STS,total}$ is the total number of space-time streams for all users. For the single-user case, $N_{STS,total} = N_{STS}$. N_R is the number of receive antennas. The channel estimate includes coefficients for both the data and pilot subcarriers.

Data Types: `double`

More About

VHT-LTF

The very high throughput long training field (VHT-LTF) is located between the VHT-STF and VHT-SIG-B portion of the VHT packet.



It is used for MIMO channel estimation and pilot subcarrier tracking. The VHT-LTF includes one VHT long training symbol for each spatial stream indicated by the selected MCS. Each symbol is 4 µs long. A maximum of eight symbols are permitted in the VHT-LTF.

For a detailed description of the VHT-LTF, see section 21.3.8.3.5 of IEEE Std 802.11-2016.

Frequency Smoothing

Frequency smoothing can improve channel estimation by averaging out noise.

Frequency smoothing is recommended only for cases in which a single transmit antenna is used. Frequency smoothing consists of applying a moving-average filter that spans multiple adjacent subcarriers. Channel conditions dictate whether frequency smoothing is beneficial.

- If adjacent subcarriers are highly correlated, frequency smoothing results in significant noise reduction.
- In a highly frequency-selective channel, smoothing can degrade the quality of the channel estimate.

Version History

Introduced in R2015b

References

[1] IEEE Std 802.11ac™-2013 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications — Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz.

[2] IEEE Std 802.11™-2012 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

[3] Perahia, E., and R. Stacey. *Next Generation Wireless LANs: 802.11n and 802.11ac*. 2nd Edition, United Kingdom: Cambridge University Press, 2013.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanVHTConfig` | `wlanVHTLTFDemodulate` | `wlanVHTDataRecover`

wlanVHTLTFDemodulate

Demodulate VHT-LTF waveform

Syntax

```
sym = wlanVHTLTFDemodulate(rx,cfg)
sym = wlanVHTLTFDemodulate(rx,cbw,numSTS)
sym = wlanVHTLTFDemodulate( ___ ,symOffset)
```

Description

`sym = wlanVHTLTFDemodulate(rx,cfg)` returns a demodulated "VHT-LTF" on page 3-500[21] waveform `y` by demodulating time-domain input signal `rx` for very high throughput (VHT) transmission parameters `cfg`.

`sym = wlanVHTLTFDemodulate(rx,cbw,numSTS)` specifies channel bandwidth `cbw` and number of space-time streams `numSTS`.

`sym = wlanVHTLTFDemodulate( ___ ,symOffset)` specifies the OFDM symbol offset as a fraction of the cyclic prefix length.

Examples

Demodulate Received VHT-LTF Signal

Create a VHT format configuration object.

```
vht = wlanVHTConfig;
```

Generate a VHT-LTF signal.

```
txVHTLTF = wlanVHTLTF(vht);
```

Add white noise to the signal.

```
rxVHTLTF = awgn(txVHTLTF,1);
```

Demodulate the received signal.

```
y = wlanVHTLTFDemodulate(rxVHTLTF,vht);
```

Demodulate VHT-LTF and Estimate Channel Coefficients

Specify a VHT format configuration object and generate a VHT-LTF.

```
vht = wlanVHTConfig;
txltf = wlanVHTLTF(vht);
```

21 IEEE Std 802.11ac-2013 Adapted and reprinted with permission from IEEE. Copyright IEEE 2013. All rights reserved.

Multiply the transmitted VHT-LTF by 0.1 + 0.1i . Pass the signal through an AWGN channel.

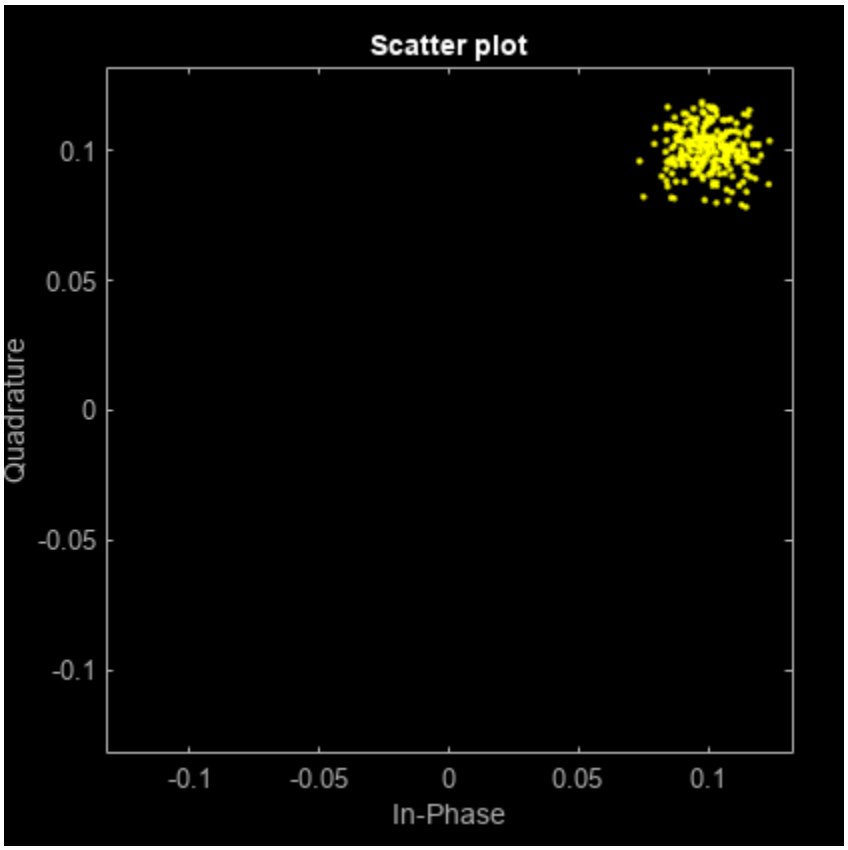
```
rxltfNoNoise = txltf * complex(0.1,0.1);
rxltf = awgn(rxltfNoNoise,20,'measured');
```

Demodulated the received VHT-LTF with a symbol offset of 0.5.

```
dltf = wlanVHTLTFDemodulate(rxltf,vht,0.5);
```

Estimate the channel using the demodulated VHT-LTF. Plot the result.

```
chEst = wlanVHTLTFChannelEstimate(dltf,vht);
scatterplot(chEst)
```



The estimate is very close to the previously introduced 0.1+0.1i multiplier.

Extract VHT-LTF and Recover VHT Data

Generate a VHT waveform. Extract and demodulate the VHT long training field (VHT-LTF) to estimate the channel coefficients. Recover the data field by using the channel estimate and use this field to determine the number of bit errors.

Configure a VHT-format configuration object with two paths.

```
vht = wlanVHTConfig('NumTransmitAntennas',2,'NumSpaceTimeStreams',2);
```

Generate a random PSDU and create the corresponding VHT waveform.

```
txPSDU = randi([0 1],8*vht.PSDULength,1);
txSig = wlanWaveformGenerator(txPSDU,vht);
```

Pass the signal through a TGac 2x2 MIMO channel.

```
tgacChan = wlanTGacChannel('NumTransmitAntennas',2,'NumReceiveAntennas',2, ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
rxSigNoNoise = tgacChan(txSig);
```

Add AWGN to the received signal. Set the noise variance for the case in which the receiver has a 9-dB noise figure.

```
nVar = 10^((-228.6+10*log10(290)+10*log10(80e6)+9)/10);
awgnChan = comm.AWGNChannel('NoiseMethod','Variance','Variance',nVar);
rxSig = awgnChan(rxSigNoNoise);
```

Determine the indices for the VHT-LTF and extract the field from the received signal.

```
indVHT = wlanFieldIndices(vht,'VHT-LTF');
rxLTF = rxSig(indVHT(1):indVHT(2),:);
```

Demodulate the VHT-LTF and estimate the channel coefficients.

```
dLTF = wlanVHTLTFDemodulate(rxLTF,vht);
chEst = wlanVHTLTFChannelEstimate(dLTF,vht);
```

Extract the VHT-Data field and recover the information bits.

```
indData = wlanFieldIndices(vht,'VHT-Data');
rxData = rxSig(indData(1):indData(2),:);
rxPSDU = wlanVHTDataRecover(rxData,chEst,nVar,vht);
```

Determine the number of bit errors.

```
numErrs = biterr(txPSDU,rxPSDU)
```

```
numErrs = 0
```

Recover Bits from VHT Signal Transmitted through MU-MIMO Channel

Recover bits from the VHT-Data field of a VHT multi-user transmission recovered from a fading MU-MIMO channel by using channel estimation on the VHT-LTF.

This example can return high bit error rates because the transmission does not include precoding to mitigate the interference between space-time streams. However, the example shows a typical VHT signal recovery workflow and appropriate syntax use for the functions involved.

Configure a VHT transmission with a channel bandwidth of 160 MHz, two users, and four transmit antennas. Assign one space-time stream to the first user and three space-time streams to the second user.

```
cbw = 'CBW160';
numSTS = [1 3];
```

```
cfgVHT = wlanVHTConfig('ChannelBandwidth',cbw,'NumUsers',2, ...
    'NumTransmitAntennas',4,'NumSpaceTimeStreams',numSTS);
```

Generate a payload of bits for each user. This payload must be in the a 1-by-*N* cell array, where *N* is the number of users.

```
psduLength = 8*cfgVHT.PSDULength;
numUsers = cfgVHT.NumUsers;
bits = cell(1,2);
for nu = 1:numUsers
    bits{nu} = randi([0 1],psduLength(nu),1);
end
```

Generate VHT-LTF and VHT-Data field signals.

```
txLTF  = wlanVHTLTF(cfgVHT);
txDataSym = wlanVHTData(bits,cfgVHT);
```

Pass the VHT-Data field signal for the first user through a 4x1 channel because this signal consists of a single space-time stream. Pass the VHT-Data field for the second user data through a 4x3 channel because this signal consists of three space-time streams. Apply AWGN to each signal, assuming an SNR of 15 dB.

```
snr = 15;
H{1} = complex(randn(4,1),randn(4,1))/sqrt(2);
H{2} = complex(randn(4,3),randn(4,3))/sqrt(2);
number = zeros(2,1);
ratio = zeros(2,1);
for userIdx = 1:numUsers
    rxDataSym = awgn(txDataSym*H{userIdx},snr,'measured');
```

Apply the same channel processing to the VHT-LTF for each user.

```
    rxLTF = awgn(txLTF*H{userIdx},snr,'measured');
```

Calculate the received signal power for each user and estimate the noise variance.

```
    powerDB = 10*log10(var(rxDataSym));
    noiseVarEst = mean(10.^(0.1*(powerDB-snr)));
```

Estimate the channel characteristics by using the VHT-LTF.

```
    demod = wlanVHTLTFDemodulate(rxLTF,cbw,numSTS);
    chEst = wlanVHTLTFChannelEstimate(demod,cbw,numSTS);
```

Recover the bits from the received VHT-Data field for each user and determine the bit error rate by comparing the recovered bits with the original payload bits.

```
    dataBits = wlanVHTDataRecover(rxDataSym,chEst,noiseVarEst,cfgVHT,userIdx);
    [number(userIdx),ratio(userIdx)] = biterr(bits{userIdx},dataBits);
    disp(number(userIdx))
    disp(ratio(userIdx))
end
```

```
        4269

    0.5082

        2444
```

```
0.0968
```

Input Arguments

`rx` — Received time-domain signal

complex-valued matrix

Received time-domain signal, specified as a complex-valued matrix of size N_s-by-N_r.

- N_s is the number of time-domain samples. If N_s is not an integer multiple of the OFDM symbol length, L_s, for the specified field,then the function ignores the remaining `mod(`N_s`,`L_s`)` symbols.
- N_r is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

`cfg` — VHT format configuration

`wlanVHTConfig` object

VHT format configuration, specified as a `wlanVHTConfig` object.

`cbw` — Channel bandwidth

`'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth, specified as `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`. If the transmission has multiple users, the same channel bandwidth is applied to all users.

Data Types: `char` | `string`

`numSTS` — Number of space-time streams

integer from 1 to 8 | 1-by-N_{Users} vector of integers from 1 to 4

Number of space-time streams in the transmission, specified as a scalar or vector.

- For a single user, the number of space-time streams is a scalar integer from 1 to 8.
- For multiple users, the number of space-time streams is a 1-by-N_{Users} vector of integers from 1 to 4, where the vector length, N_{Users}, is an integer from 1 to 4.

Example: `[1 3 2]` indicates that one space-time stream is assigned to user 1, three space-time streams are assigned to user 2, and two space-time streams are assigned to user 3.

Note The sum of the space-time stream vector elements must not exceed eight.

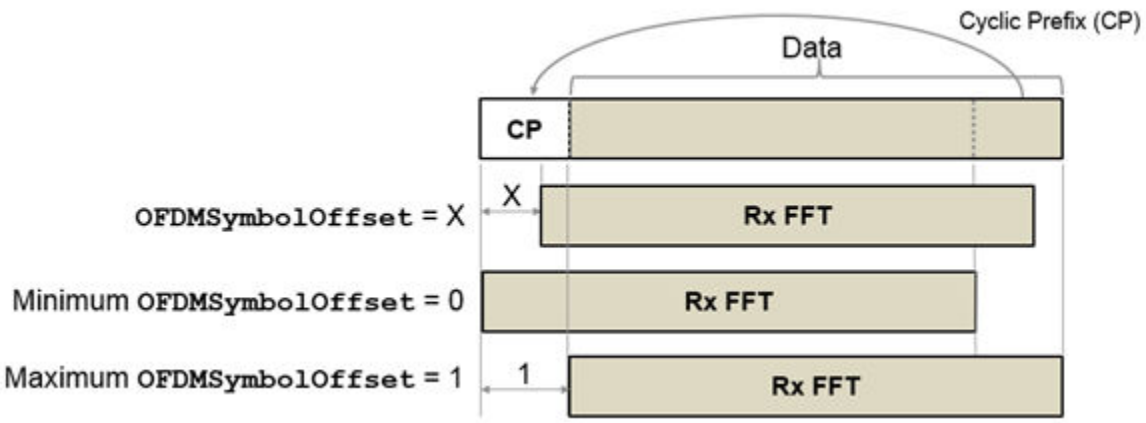
Data Types: `double`

`symOffset` — OFDM symbol sampling offset

`0.75` (default) | scalar in the interval [0, 1]

OFDM symbol sampling offset, as a fraction of the cyclic prefix length, specified as a scalar in the interval [0, 1].

The value that you specify indicates the start location for OFDM demodulation relative to the beginning of the cyclic prefix.



Example: `0.45`

Data Types: `double`

Output Arguments

`sym` — Demodulated frequency-domain signal

complex-valued array

Demodulated frequency-domain signal, returned as a complex-valued array of size N_{sc}-by-N_{sym}-by-N_r.

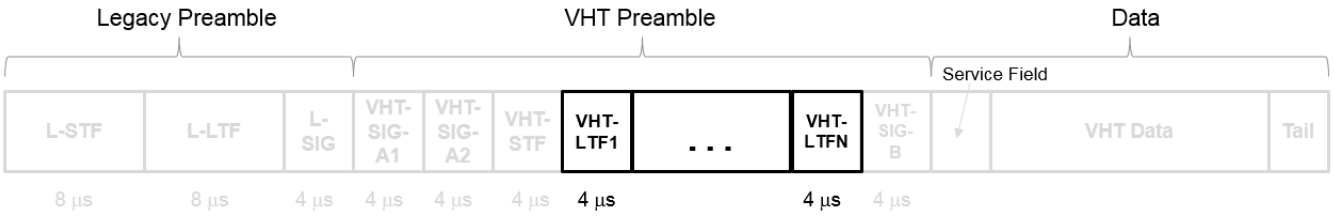
- N_{sc} is the number of active occupied subcarriers in the demodulated field.
- N_{sym} is the number of OFDM symbols.
- N_r is the number of receive antennas.

Data Types: `double`

More About

VHT-LTF

The very high throughput long training field (VHT-LTF) is located between the VHT-STF and VHT-SIG-B portion of the VHT packet.



It is used for MIMO channel estimation and pilot subcarrier tracking. The VHT-LTF includes one VHT long training symbol for each spatial stream indicated by the selected MCS. Each symbol is 4 µs long. A maximum of eight symbols are permitted in the VHT-LTF.

For a detailed description of the VHT-LTF, see section 21.3.8.3.5 of IEEE Std 802.11-2016.

Version History

Introduced in R2015b

References

[1] IEEE Std 802.11ac™-2013 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications — Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz.

[2] IEEE Std 802.11™-2012 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanVHTLTF` | `wlanVHTConfig` | `wlanVHTLTFChannelEstimate`

wlanVHTOFDMInfo

OFDM information for VHT transmission

Syntax

```
info = wlanVHTOFDMInfo(field,cfg)
info = wlanVHTOFDMInfo(field,cbw,gi)
info = wlanVHTOFDMInfo(field,cbw)
info = wlanVHTOFDMInfo( ___ ,OversamplingFactor=osf)
```

Description

`info = wlanVHTOFDMInfo(field,cfg)` returns `info`, a structure containing orthogonal frequency-division multiplexing (OFDM) information for the input field of a very-high-throughput (VHT) transmission parameterized by `cfg`.

`info = wlanVHTOFDMInfo(field,cbw,gi)` returns OFDM information for channel bandwidth `cbw` and guard interval `gi`. To return OFDM information for the VHT-Data field when the format configuration is unknown, use this syntax.

`info = wlanVHTOFDMInfo(field,cbw)` returns OFDM information for the specified field and channel bandwidth `cbw`. To return OFDM information for any field other than VHT-Data when the format configuration is unknown, use this syntax.

`info = wlanVHTOFDMInfo( ___ ,OversamplingFactor=osf)` returns OFDM information for the specified oversampling factor. For more information about oversampling, see “FFT-Based Oversampling” on page 3-505.

Examples

Demodulate VHT-LTF and Get OFDM Information

Perform OFDM demodulation on the VHT-LTF, then extract the data and pilot subcarriers.

Generate a WLAN waveform for a VHT transmission.

```
cfg = wlanVHTConfig;
bits = [1; 0; 0; 1];
waveform = wlanWaveformGenerator(bits,cfg);
```

Obtain the field indices and extract the VHT-LTF.

```
ind = wlanFieldIndices(cfg);
rx = waveform(ind.VHTLTF(1):ind.VHTLTF(2),:);
```

Perform OFDM demodulation on the VHT-LTF.

```
sym = wlanVHTLTFDemodulate(rx,cfg);
```

Get the OFDM information, then extract the data and pilot subcarriers.

```
info = wlanVHTOFDMInfo('VHT-LTF',cfg);
data = sym(info.DataIndices,:,:);
pilots = sym(info.PilotIndices,:,:);
```

Get OFDM Information for the VHT L-LTF

Get OFDM information for the VHT-LTF in a transmission with specified channel bandwidth.

Specify a channel bandwidth of 40 MHz.

```
cbw = 'CBW40';
```

Return and display the OFDM information for the L-LTF.

```
info = wlanVHTOFDMInfo('L-LTF',cbw);
disp(info);
```

```
                FFTLength: 128
               SampleRate: 40000000
                 CPLength: [64 0]
           NumSubchannels: 2
                 NumTones: 104
    ActiveFrequencyIndices: [104x1 double]
          ActiveFFTIndices: [104x1 double]
              DataIndices: [96x1 double]
             PilotIndices: [8x1 double]
```

Get OFDM Information for VHT-Data Field

Get OFDM information for the VHT-Data field in a transmission with specified channel bandwidth and guard interval.

Specify a channel bandwidth of 80 MHz and a short guard interval.

```
cbw = 'CBW80';
gi = 'Short';
```

Return and display the OFDM information for the VHT-Data field.

```
info = wlanVHTOFDMInfo('VHT-Data',cbw,gi);
disp(info);
```

```
                FFTLength: 256
               SampleRate: 80000000
                 CPLength: 32
           NumSubchannels: 4
                 NumTones: 242
    ActiveFrequencyIndices: [242x1 double]
          ActiveFFTIndices: [242x1 double]
              DataIndices: [234x1 double]
             PilotIndices: [8x1 double]
```

Input Arguments

`field` — Field for which to return OFDM information

`'L-LTF'` | `'L-SIG'` | `'VHT-SIG-A'` | `'VHT-SIG-B'` | `'VHT-LTF'` | `'VHT-Data'`

Field for which to return OFDM information, specified as one of these values.

- `'L-LTF'` – Return OFDM information for the legacy long training field (L-LTF).
- `'L-SIG'` – Return OFDM information for the legacy signal (L-SIG) field.
- `'VHT-SIG-A'` – Return OFDM information for the VHT signal A (VHT-SIG-A) field.
- `'VHT-SIG-B'` – Return OFDM information for the VHT signal B (VHT-SIG-B) field.
- `'VHT-LTF'` – Return OFDM information for the VHT long training field (VHT-LTF).
- `'VHT-Data'` – Return OFDM information for the VHT-Data field.

Data Types: `char` | `string`

`cfg` — Transmission parameters

`wlanVHTConfig` object

Transmission parameters, specified as a `wlanVHTConfig` object.

`cbw` — Channel bandwidth

`'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth, specified as one of these values.

- `'CBW20'` – Channel bandwidth of 20 MHz
- `'CBW40'` – Channel bandwidth of 40 MHz
- `'CBW80'` – Channel bandwidth of 80 MHz
- `'CBW160'` – Channel bandwidth of 160 MHz

Data Types: `char` | `string`

`gi` — Guard interval duration

`'Short'` | `'Long'`

Guard interval duration, specified as `'Short'` or `'Long'`.

Data Types: `char` | `string`

`osf` — Oversampling factor

1 (default) | scalar greater than or equal to 1

Oversampling factor, specified as a scalar greater than or equal to 1. The oversampled cyclic prefix length must be an integer number of samples.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`info` — OFDM information

structure

OFDM information, returned as a structure containing these fields.

Name	Values	Description	Data Types
FFTLength	Positive integer	Length of the fast Fourier transform (FFT)	double
SampleRate	Positive scalar	Sample rate of the waveform	double
CPLength	Positive integer	Cyclic prefix length, in samples	double
NumTones	Nonnegative integer	Number of active subcarriers	double
NumSubchannels	Positive integer	Number of 20-MHz subchannels	double
ActiveFrequencyIndices	Column vector of integers in the interval [-FFTLength/2, (FFTLength/2 – 1)]	Indices of active subcarriers. Each element of this field is the index of an active subcarrier, such that the direct current (DC) or null subcarrier is at the center of the frequency band.	double
ActiveFFTIndices	Column vector of integers in the interval [1, FFTLength]	Indices of active subcarriers within the FFT	double
DataIndices	Column vector of integers in the interval [1, NumTones]	Indices of data within the active subcarriers	double
PilotIndices	Column vector of integers in the interval [1, NumTones]	Indices of pilots within the active subcarriers	double

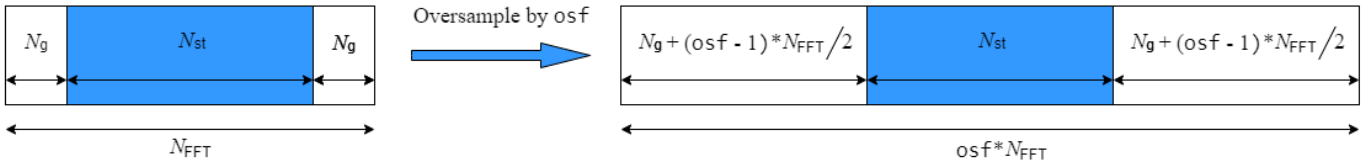
Data Types: struct

Algorithms

FFT-Based Oversampling

An oversampled signal is a signal sampled at a frequency that is higher than the Nyquist rate. WLAN signals maximize occupied bandwidth by using small guardbands, which can pose problems for anti-imaging and anti-aliasing filters. Oversampling increases guardband width relative to the total signal bandwidth, thereby increasing the number of samples in the signal.

This function performs oversampling by using a larger IFFT and zero pad when generating an OFDM waveform. This diagram shows the oversampling process for an OFDM waveform with N_{FFT} subcarriers comprising N_{g} guardband subcarriers on either side of N_{st} occupied bandwidth subcarriers.



Version History

Introduced in R2019a

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`wlanVHTLTFDemodulate` | `wlanLLTFDemodulate`

Objects

`wlanVHTConfig`

wlanVHTSIGA

Generate VHT-SIG-A waveform

Syntax

```
y= wlanVHTSIGA(cfg)
[y,bits] = wlanVHTSIGA(cfg)
[ ___ ] = wlanVHTSIGA(cfg,OversamplingFactor=osf)
```

Description

`y= wlanVHTSIGA(cfg)` generates a "VHT-SIG-A" on page 3-509[22] time-domain waveform for the specified transmission parameters. See "VHT-SIG-A Processing" on page 3-510 for waveform generation details.

`[y,bits] = wlanVHTSIGA(cfg)` also outputs "VHT-SIG-A" on page 3-509 information bits.

`[ ___ ] = wlanVHTSIGA(cfg,OversamplingFactor=osf)` generates an oversampled waveform with the specified oversampling factor. For more information about oversampling, see "FFT-Based Oversampling" on page 3-511.

Examples

Generate VHT-SIG-A Waveform

Generate the VHT-SIG-A waveform for an 80 MHz transmission packet.

Create a VHT configuration object, assign an 80 MHz channel bandwidth, and generate the waveform.

```
cfgVHT = wlanVHTConfig;
cfgVHT.ChannelBandwidth = 'CBW80';
y = wlanVHTSIGA(cfgVHT);
size(y)
```

```
ans = 1×2

   640     1
```

The 80 MHz waveform has two OFDM symbols and is a total of 640 samples long. Each symbol contains 320 samples.

Extract VHT-SIG-A Bandwidth Information

Generate the VHT-SIG-A waveform for a 40 MHz transmission packet.

22 IEEE Std 802.11ac-2013 Adapted and reprinted with permission from IEEE. Copyright IEEE 2013. All rights reserved.

Create a VHT configuration object, and assign a 40 MHz channel bandwidth.

```
cfgVHT = wlanVHTConfig;
cfgVHT.ChannelBandwidth = 'CBW40';
```

Generate the VHT-SIG-A waveform and information bits.

```
[y,bits] = wlanVHTSIGA(cfgVHT);
```

Extract the bandwidth from the returned bits and analyze. The bandwidth information is contained in the first two bits.

```
bwBits = bits(1:2);
bi2de(bwBits)
```

```
ans = 2x1 int8 column vector

   1
   0
```

As defined in IEEE Std 802.11ac-2013, Table 22-12, a value of `'1'` corresponds to 40 MHz bandwidth.

Input Arguments

`cfg` — Transmission parameters

`wlanVHTConfig` object

Transmission parameters, specified as a `wlanVHTConfig` object.

`osf` — Oversampling factor

1 (default) | scalar greater than or equal to 1

Oversampling factor, specified as a scalar greater than or equal to 1. The oversampled cyclic prefix length must be an integer number of samples.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`y` — VHT-SIG-A time-domain waveform

matrix

"VHT-SIG-A" on page 3-509 time-domain waveform, returned as an N_S-by-N_T matrix. N_S is the number of time-domain samples, and N_T is the number of transmit antennas.

N_S is proportional to the channel bandwidth. The time-domain waveform consists of two symbols.

`ChannelBandwidth`	N_S
`'CBW20'`	160
`'CBW40'`	320
`'CBW80'`	640

ChannelBandwidth	N_S
'CBW160'	1280

See “VHT-SIG-A Processing” on page 3-510 for waveform generation details.

Data Types: double
Complex Number Support: Yes

bits — Signaling bits used for the VHT-SIG-A field

48-bit column vector

Signaling bits used for the “VHT-SIG-A” on page 3-509, returned as a 48-bit column vector.

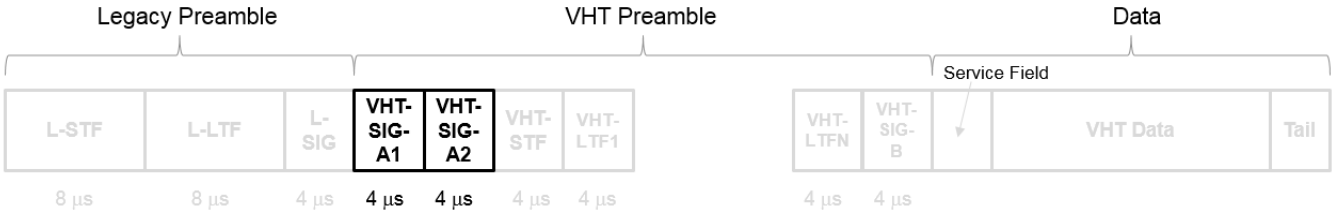
Data Types: int8

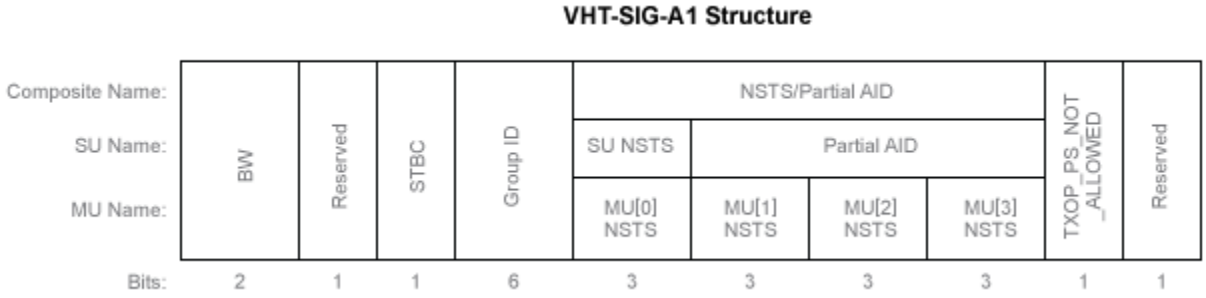
More About

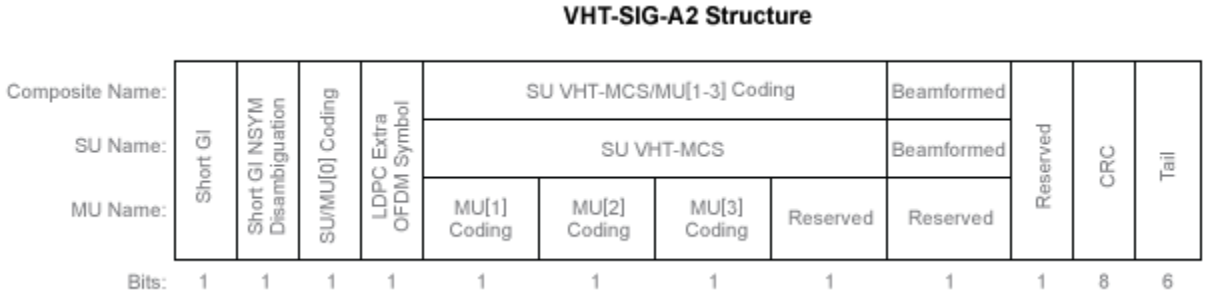
VHT-SIG-A

The very high throughput signal A (VHT-SIG-A) field contains information required to interpret VHT format packets. Similar to the non-HT signal (L-SIG) field for the non-HT OFDM format, this field stores the actual rate value, channel coding, guard interval, MIMO scheme, and other configuration details for the VHT format packet. Unlike the HT-SIG field, this field does not store the packet length information. Packet length information is derived from L-SIG and is captured in the VHT-SIG-B field for the VHT format.

For a detailed description of the VHT-SIG-A field, see section 21.3.8.3.3 of IEEE Std 802.11-2016. The VHT-SIG-A field consists of two symbols: VHT-SIG-A1 and VHT-SIG-A2. These symbols are located between the L-SIG and the VHT-STF portion of the VHT format PPDU.







The VHT-SIG-A field includes these components. The bit field structures for VHT-SIG-A1 and VHT-SIG-A2 vary for single user or multiuser transmissions.

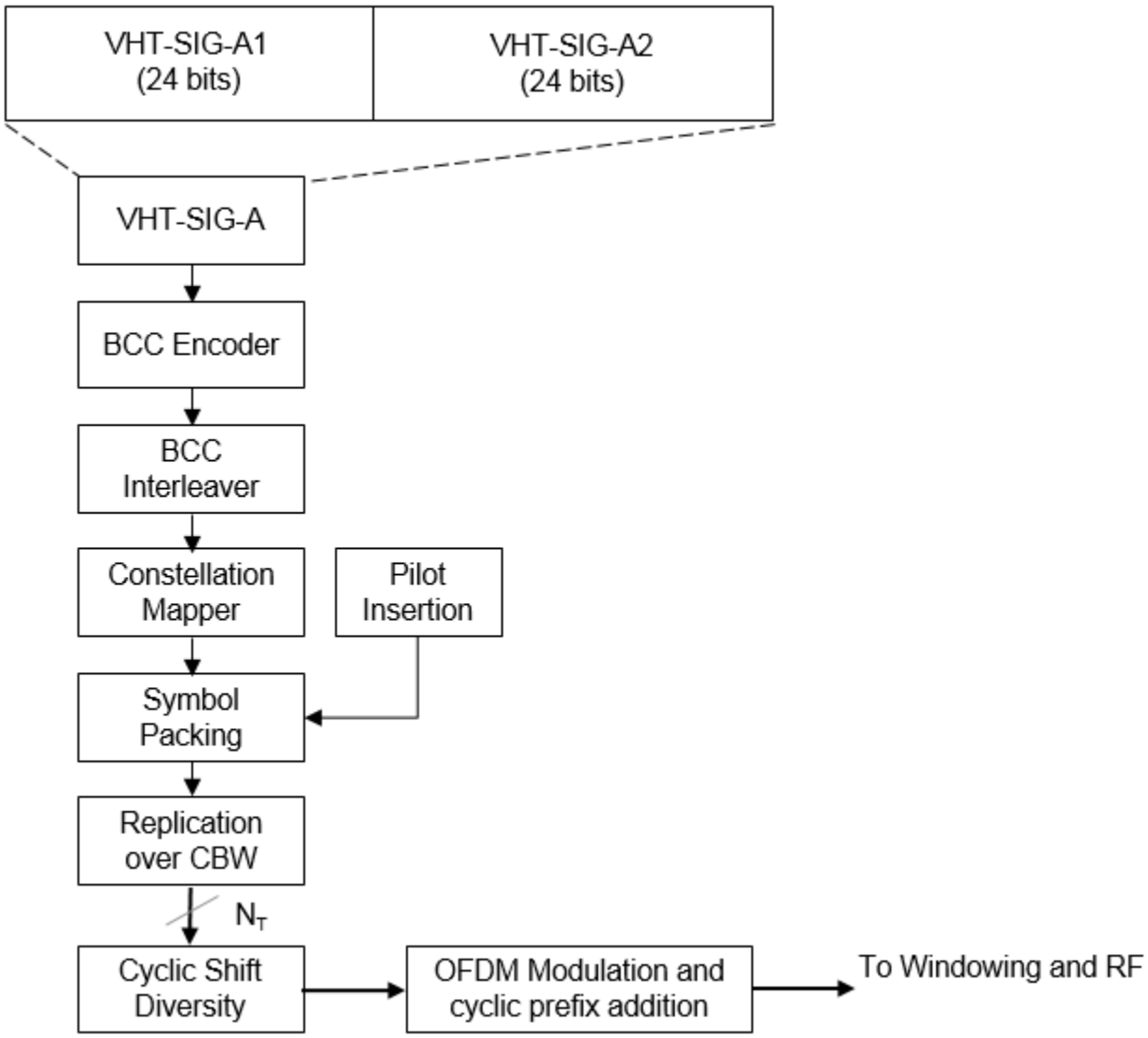
- **BW** — A two-bit field that indicates 0 for 20 MHz, 1 for 40 MHz, 2 for 80 MHz, or 3 for 160 MHz.
- **STBC** — A bit that indicates the presence of space-time block coding.
- **Group ID** — A six-bit field that indicates the group and user position assigned to a STA.
- $\mathbf{N_{STS}}$ — A three-bit field for a single user or 4 three-bit fields for a multiuser scenario, that indicates the number of space-time streams per user.
- **Partial AID** — An identifier that combines the association ID and the BSSID.
- **TXOP_PS_NOT_ALLOWED** — An indicator bit that shows if client devices are allowed to enter dose state. This bit is set to false when the VHT-SIG-A structure is populated, indicating that the client device is allowed to enter dose state.
- **Short GI** — A bit that indicates use of the 400 ns guard interval.
- **Short GI NSYM Disambiguation** — A bit that indicates if an extra symbol is required when the short GI is used.
- **SU/MU[0] Coding** — A bit field that indicates if convolutional or LDPC coding is used for a single user or for user MU[0] in a multiuser scenario.
- **LDPC Extra OFDM Symbol** — A bit that indicates if an extra OFDM symbol is required to transmit the data field.
- **MCS** — A four-bit field.
  - For a single user scenario, it indicates the modulation and coding scheme used.
  - For a multiuser scenario, it indicates use of convolutional or LDPC coding and the MCS setting is conveyed in the VHT-SIG-B field.
- **Beamformed** — An indicator bit set to 1 when a beamforming matrix is applied to the transmission.
- **CRC** — An eight-bit field used to detect errors in the VHT-SIG-A transmission.
- **Tail** — A six-bit field used to terminate the convolutional code.

Algorithms

VHT-SIG-A Processing

The “VHT-SIG-A” on page 3-509 field includes information required to process VHT format packets.

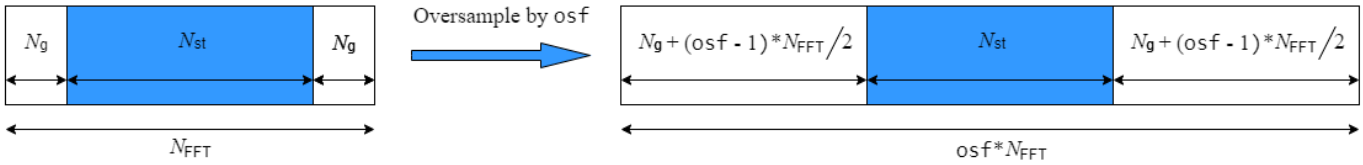
For algorithm details, refer to IEEE Std 802.11ac-2013 [1], Section 22.3.4.5. The `wlanVHTSIGA` function performs transmitter processing on the "VHT-SIG-A" on page 3-509 field and outputs the time-domain waveform.



FFT-Based Oversampling

An oversampled signal is a signal sampled at a frequency that is higher than the Nyquist rate. WLAN signals maximize occupied bandwidth by using small guardbands, which can pose problems for anti-imaging and anti-aliasing filters. Oversampling increases guardband width relative to the total signal bandwidth, thereby increasing the number of samples in the signal.

This function performs oversampling by using a larger IFFT and zero pad when generating an OFDM waveform. This diagram shows the oversampling process for an OFDM waveform with N_{FFT} subcarriers comprising N_{g} guardband subcarriers on either side of N_{st} occupied bandwidth subcarriers.



Version History

Introduced in R2015b

References

[1] IEEE Std 802.11ac™-2013 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications — Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanVHTConfig` | `wlanLSIG` | `wlanVHTSTF` | `wlanVHTSIGARecover`

wlanVHTSIGARecover

Recover VHT-SIG-A information bits

Syntax

```
recBits = wlanVHTSIGARecover(rxSig,chEst,noiseVarEst,cbw)
recBits = wlanVHTSIGARecover(rxSig,chEst,noiseVarEst,cbw,Name,Value)
[recBits,failCRC] = wlanVHTSIGARecover( ___ )
[recBits,failCRC,eqSym] = wlanVHTSIGARecover( ___ )
[recBits,failCRC,eqSym,cpe] = wlanVHTSIGARecover( ___ )
```

Description

`recBits = wlanVHTSIGARecover(rxSig,chEst,noiseVarEst,cbw)` returns the recovered information bits from the "VHT-SIG-A" on page 3-519[23] field. Inputs include the received "VHT-SIG-A" on page 3-519 field, the channel estimate, the noise variance estimate, and the channel bandwidth.

`recBits = wlanVHTSIGARecover(rxSig,chEst,noiseVarEst,cbw,Name,Value)` specifies algorithm parameters by using one or more name-value pair arguments.

`[recBits,failCRC] = wlanVHTSIGARecover( ___ )` returns the failure status of the CRC check, `failCRC`, using the arguments from previous syntaxes.

`[recBits,failCRC,eqSym] = wlanVHTSIGARecover( ___ )` returns the equalized symbols, `eqSym`.

`[recBits,failCRC,eqSym,cpe] = wlanVHTSIGARecover( ___ )` returns the common phase error, `cpe`.

Examples

Recover VHT-SIG-A Information Bits

Recover the information bits in the VHT-SIG-A field by performing channel estimation on the L-LTF over a 1x2 quasi-static fading channel

Create a `wlanVHTConfig` object having a channel bandwidth of 80 MHz. Generate L-LTF and VHT-SIG-A field signals using this object.

```
cfg = wlanVHTConfig('ChannelBandwidth','CBW80');
txLLTF = wlanLLTF(cfg);
[txVHTSIGA, txBits] = wlanVHTSIGA(cfg);
chanBW = cfg.ChannelBandwidth;
noiseVarEst = 0.1;
```

Pass the L-LTF and VHT-SIG-A signals through a 1x2 quasi-static fading channel with AWGN.

23 IEEE Std 802.11ac-2013 Adapted and reprinted with permission from IEEE. Copyright IEEE 2013. All rights reserved.

```
H = 1/sqrt(2)*complex(randn(1,2),randn(1,2));
rxLLTF    = awgn(txLLTF*H,10);
rxVHTSIGA = awgn(txVHTSIGA*H,10);
```

Perform channel estimation based on the L-LTF.

```
demodLLTF = wlanLLTFDemodulate(rxLLTF,chanBW,1);
chanEst = wlanLLTFChannelEstimate(demodLLTF,chanBW);
```

Recover the VHT-SIG-A. Verify that the CRC check was successful.

```
[rxBits,failCRC] = wlanVHTSIGARecover(rxVHTSIGA,chanEst,noiseVarEst,'CBW80');
failCRC
```

```
failCRC = logical
   0
```

The CRC failure check returns a `0`, indicating that the CRC passed.

Compare the transmitted bits to the received bits. Confirm that the reported CRC result is correct because the output matches the input.

```
isequal(txBits,rxBits)
```

```
ans = logical
   1
```

Recover VHT-SIG-A Using Zero-Forcing Equalizer

Recover the VHT-SIG-A in an AWGN channel. Configure the VHT signal to have a 160 MHz channel bandwidth, one space-time stream, and one receive antenna.

Create a `wlanVHTConfig` object, specifying a channel bandwidth of 160 MHz.

```
cfg = wlanVHTConfig('ChannelBandwidth','CBW160');
```

Generate L-LTF and VHT-SIG-A field signals.

```
txLLTF = wlanLLTF(cfg);
[txSig,txBits] = wlanVHTSIGA(cfg);
chanBW = cfg.ChannelBandwidth;
noiseVarEst = 0.1;
```

Pass the transmitted VHT-SIG-A through an AWGN channel.

```
awgnChan = comm.AWGNChannel('NoiseMethod','Variance','Variance',noiseVarEst);
rxLLTF = awgnChan(txLLTF);
rxSig = awgnChan(txSig);
```

Perform channel estimation based on the L-LTF.

```
demodLLTF = wlanLLTFDemodulate(rxLLTF,chanBW,1);
chEst = wlanLLTFChannelEstimate(demodLLTF,chanBW);
```

Recover the VHT-SIG-A, specifying a zero-forcing equalization method, and check the CRC result

```
[recBits,failCRC] = wlanVHTSIGARecover(rxSig,chEst,noiseVarEst,'CBW160','EqualizationMethod','ZF
disp(failCRC)
```

```
   0
```

Verify that the received signal contains no bit errors.

```
biterr(txBits,recBits)
```

```
ans = 0
```

Recover VHT-SIG-A in 2x2 MIMO Channel

Recover VHT-SIG-A in a 2x2 MIMO channel with AWGN. Confirm that the CRC check passes.

Configure a 2x2 MIMO VHT channel.

```
chanBW = 'CBW20';
cfgVHT = wlanVHTConfig('ChannelBandwidth', chanBW, 'NumTransmitAntennas', 2, 'NumSpaceTimeStreams
```

Generate L-LTF and VHT-SIG-A waveforms.

```
txLLTF  = wlanLLTF(cfgVHT);
txVHTSIGA = wlanVHTSIGA(cfgVHT);
```

Pass the L-LTF and VHT-SIG-A waveforms through a 2×2 MIMO channel with white noise.

```
mimoChan = comm.MIMOChannel('SampleRate', 20e6);
rxLLTF = awgn(mimoChan(txLLTF), 15);
rxVHTSIGA = awgn(mimoChan(txVHTSIGA),15);
```

Demodulate the L-LTF signal. To generate a channel estimate, use the demodulated L-LTF.

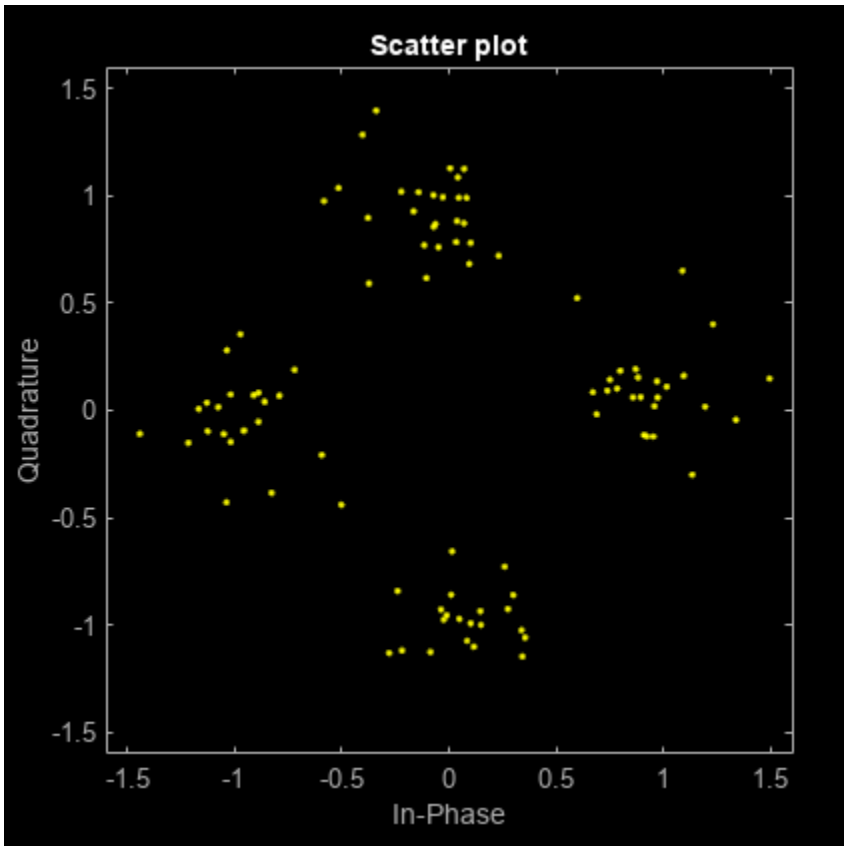
```
demodLLTF = wlanLLTFDemodulate(rxLLTF, chanBW, 1);
chanEst = wlanLLTFChannelEstimate(demodLLTF, chanBW);
```

Recover the information bits in VHT-SIG-A.

```
[recVHTSIGABits, failCRC, eqSym] = wlanVHTSIGARecover(rxVHTSIGA, chanEst, 0, chanBW);
```

Visualize the scatter plot of the equalized symbols, eqSym.

```
scatterplot(eqSym(:))
```



Input Arguments

`rxSig` — Received VHT-SIG-A

matrix

Received VHT-SIG-A field, specified as an N_S-by-N_R matrix. N_S is the number of samples and increases with channel bandwidth.

Channel Bandwidth	N_S
`'CBW20'`	160
`'CBW40'`	320
`'CBW80'`	640
`'CBW160'`	1280

N_R is the number of receive antennas.

Data Types: `double`

`chEst` — Channel estimate

3-D array

Channel estimate, specified as an N_{ST}-by-1-by-N_R array. N_{ST} is the number of occupied subcarriers and increases with channel bandwidth.

Channel Bandwidth	N_{ST}
'CBW20'	52
'CBW40'	104
'CBW80'	208
'CBW160'	416

N_R is the number of receive antennas.

The channel estimate is based on the “L-LTF” on page 3-519.

Data Types: double

noiseVarEst — Noise variance estimate
nonnegative scalar

Noise variance estimate, specified as a nonnegative scalar.

Data Types: double

cbw — Channel bandwidth
'CBW20' | 'CBW40' | 'CBW80' | 'CBW160'

Channel bandwidth in MHz, specified as 'CBW20', 'CBW40', 'CBW80', or 'CBW160'.

Data Types: char | string

Name-Value Pair Arguments

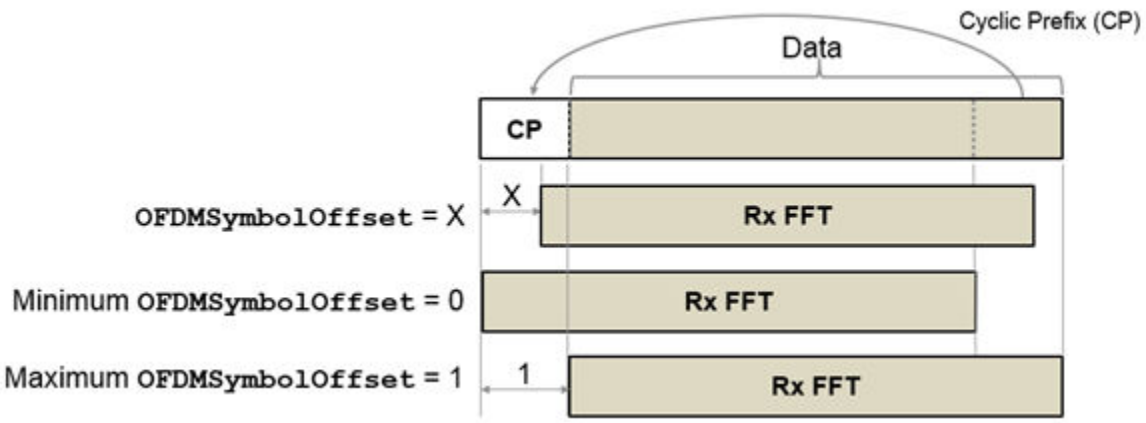
Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

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

Example: 'PilotPhaseTracking','None' disables pilot phase tracking.

OFDMSymbolOffset — OFDM symbol sampling offset
0.75 (default) | scalar in the interval [0, 1]

OFDM symbol sampling offset represented as a fraction of the cyclic prefix (CP) length, specified as the comma-separated pair consisting of 'OFDMSymbolOffset' and a scalar in the interval [0, 1]. The value you specify indicates the start location for OFDM demodulation relative to the beginning of the CP. The value 0 represents the start of the CP, and the value 1 represents the end of the CP.



Data Types: `double`

`EqualizationMethod` — Equalization method
`'MMSE'` (default) | `'ZF'`

Equalization method, specified as one of these values.

- `'MMSE'` — The receiver uses a minimum mean-square error equalizer.
- `'ZF'` — The receiver uses a zero-forcing equalizer.

When the received signal has multiple receive antennas, the function exploits receiver diversity during equalization. When the number of transmitted space-time streams is one and you specify this argument as `'ZF'`, the function performs maximal-ratio combining.

Data Types: `char` | `string`

`PilotPhaseTracking` — Pilot phase tracking
`'PreEQ'` (default) | `'None'`

Pilot phase tracking, specified as the comma-separated pair consisting of `'PilotPhaseTracking'` and one of these values.

- `'PreEQ'` — Enable pilot phase tracking, which the function performs before any equalization operation.
- `'None'` — Disable pilot phase tracking.

Data Types: `char` | `string`

Output Arguments

`recBits` — Recovered VHT-SIG-A information bits
column vector

Recovered VHT-SIG-A information bits, returned as a 48-by-1 column vector. See “VHT-SIG-A” on page 3-519 for more information.

`failCRC` — CRC failure check
`true` | `false`

CRC failure check, returned as `true` if the CRC check fails or `false` if the CRC check passes.

`eqSym` — Equalized symbols
matrix

Equalized symbols at the data carrying subcarriers, returned as 48-by-2 matrix. Each 20 MHz channel bandwidth segment has two symbols and 48 data carrying subcarriers. These segments are combined into a single 48-by-2 matrix that comprises the “VHT-SIG-A” on page 3-519 field.

`cpe` — Common phase error
column vector

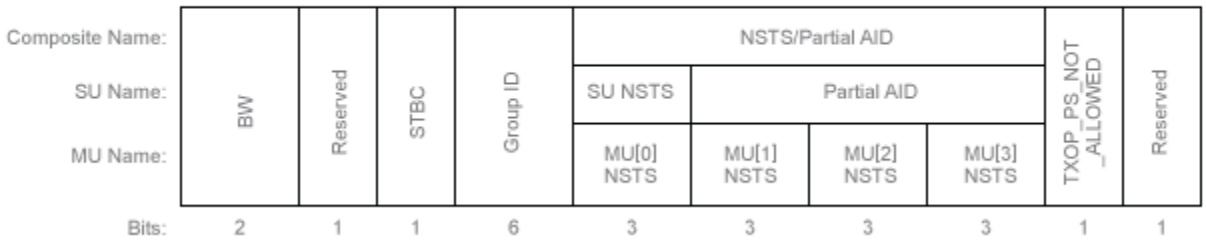
Common phase error in radians, returned as a 2-by-1 column vector.

More About

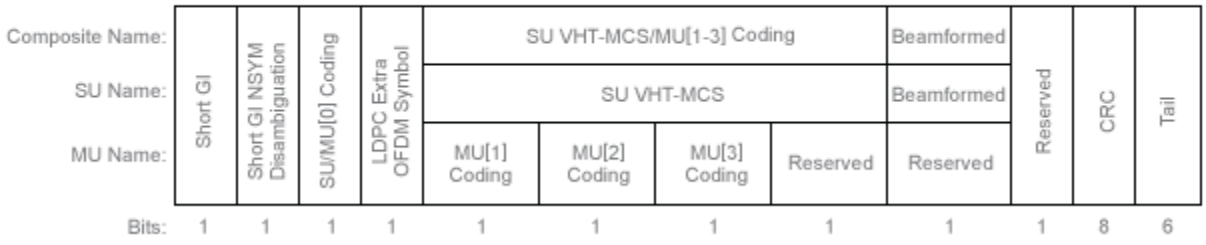
VHT-SIG-A

The very high throughput signal A (VHT-SIG-A) field consists of two symbols: VHT-SIG-A1 and VHT-SIG-A2. The VHT-SIG-A field carries information required to interpret VHT PPDU information.

VHT-SIG-A1 Structure



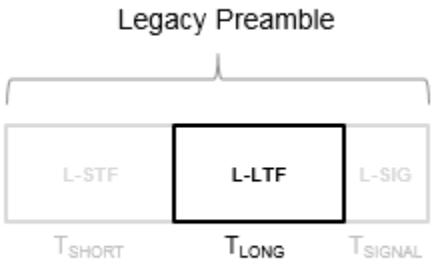
VHT-SIG-A2 Structure



For VHT-SIG-A field bit details, refer to IEEE Std 802.11ac-2013 [1], Table 22-12.

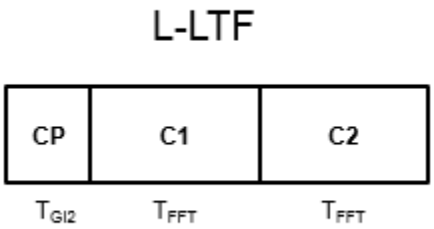
L-LTF

The L-LTF is the second field in the 802.11 OFDM PLCP legacy preamble. The L-LTF is a component of HE, VHT, HT, and non-HT PPDUs.



Channel estimation, fine frequency offset estimation, and fine symbol timing offset estimation rely on the L-LTF.

The L-LTF is composed of a cyclic prefix (CP) followed by two identical long training symbols (C1 and C2). The CP consists of the second half of the long training symbol.



The L-LTF duration varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period ($T_{FFT} = 1 / \Delta_F$)	Cyclic Prefix or Training Symbol Guard Interval (GI2) Duration ($T_{GI2} = T_{FFT} / 2$)	L-LTF Duration ($T_{LONG} = T_{GI2} + 2 \times T_{FFT}$)
20, 40, 80, 160, and 320	312.5	3.2 µs	1.6 µs	8 µs
10	156.25	6.4 µs	3.2 µs	16 µs
5	78.125	12.8 µs	6.4 µs	32 µs

PPDU

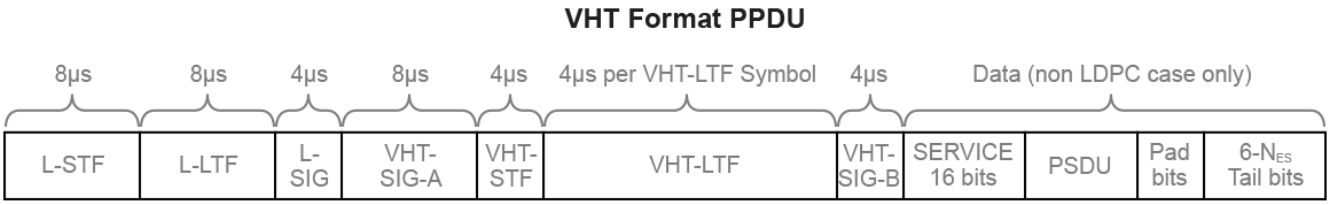
PLCP protocol data unit

The PPDU is the complete PLCP frame, including PLCP headers, MAC headers, the MAC data field, and the MAC and PLCP trailers.

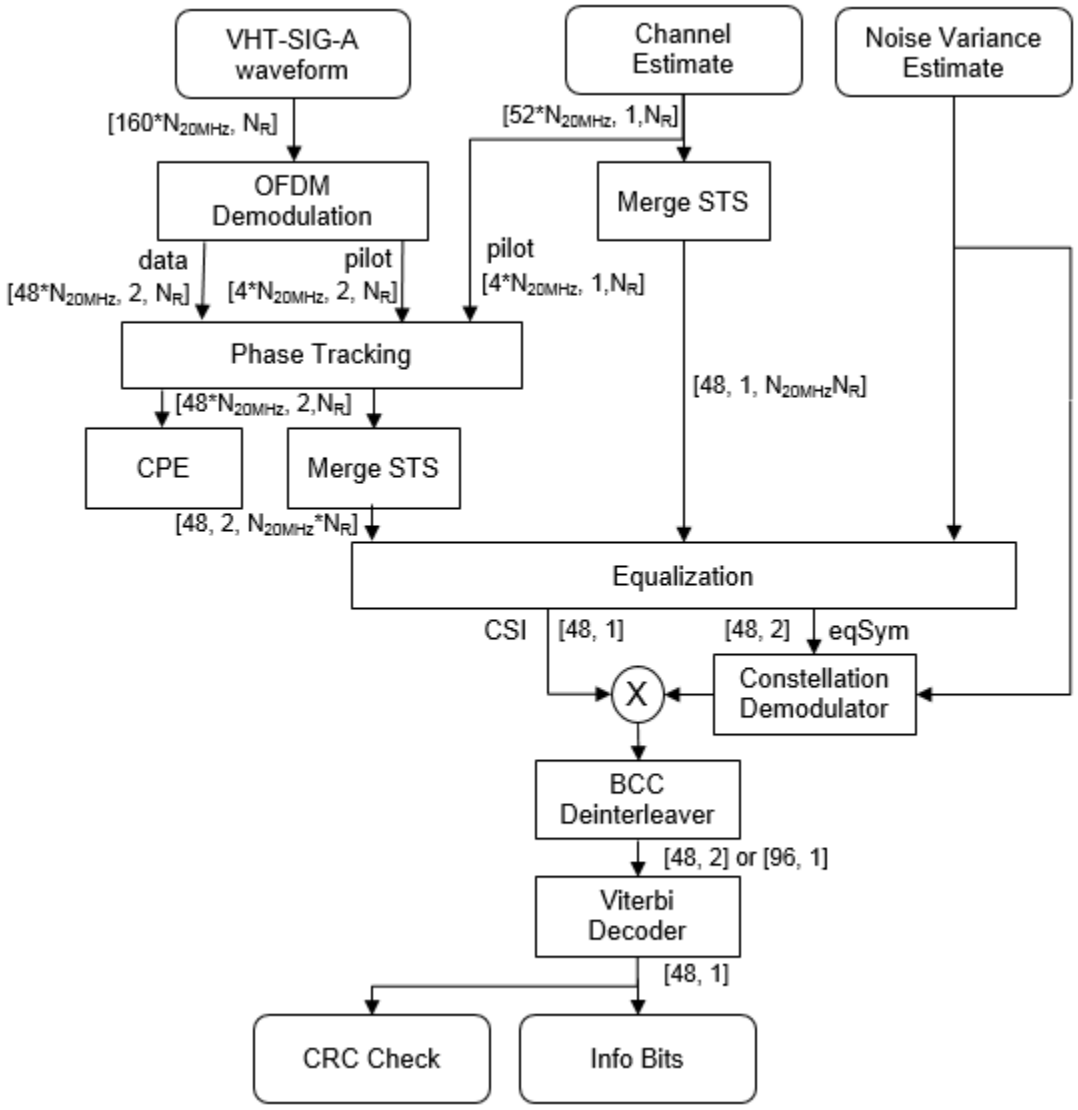
Algorithms

VHT-SIG-A Recovery

The “VHT-SIG-A” on page 3-519 field consists of two symbols and resides between the L-SIG field and the VHT-STF portion of the packet structure for the VHT format “PPDU” on page 3-520.



For single-user packets, you can recover the length information from the L-SIG and VHT-SIG-A field information. Therefore, it is not strictly required for the receiver to decode the “VHT-SIG-A” on page 3-519 field.



For “VHT-SIG-A” on page 3-519 details, refer to IEEE Std 802.11ac-2013 [1], Section 22.3.4.5, and Perahia [2], Section 7.3.2.1.

Version History

Introduced in R2015b

References

[1] IEEE Std 802.11ac™-2013 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications — Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz.

[2] Perahia, E., and R. Stacey. *Next Generation Wireless LANs: 802.11n and 802.11ac* . 2nd Edition, United Kingdom: Cambridge University Press, 2013.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanVHTSIGA` | `wlanLLTF` | `wlanLLTFDemodulate` | `wlanLLTFChannelEstimate` | `wlanVHTDataRecover` | `wlanVHTSIGBRecover`

wlanVHTSIGB

Generate VHT-SIG-B waveform

Syntax

```
y= wlanVHTSIGB(cfg)
[y,bits] = wlanVHTSIGB(cfg)
[ ___  = wlanVHTSIGB(cfg,OversamplingFactor=osf)
```

Description

`y= wlanVHTSIGB(cfg)` generates a "VHT-SIG-B" on page 3-526[24] time-domain waveform for the specified transmission parameters. See "VHT-SIG-B Processing" on page 3-527 for waveform generation details.

`[y,bits] = wlanVHTSIGB(cfg)` also outputs "VHT-SIG-B" on page 3-526 information bits.

`[ ___  = wlanVHTSIGB(cfg,OversamplingFactor=osf)` generates an oversampled waveform with the specified oversampling factor. For more information about oversampling, see "FFT-Based Oversampling" on page 3-527.

Examples

Generate VHT-SIG-B Waveform

Generate the VHT-SIG-B waveform for an 80 MHz transmission packet.

Create a VHT configuration object, assign an 80 MHz channel bandwidth, and generate the waveform.

```
cfgVHT = wlanVHTConfig('ChannelBandwidth','CBW80');
vhtsigb = wlanVHTSIGB(cfgVHT);
size(vhtsigb)
```

```
ans = 1×2

   320     1
```

The 80 MHz waveform has one OFDM symbol and is a total of 320 samples long.

Input Arguments

`cfg` — Transmission parameters

`wlanVHTConfig` object

Transmission parameters, specified as a `wlanVHTConfig` object.

24 IEEE Std 802.11ac-2013 Adapted and reprinted with permission from IEEE. Copyright IEEE 2013. All rights reserved.

`osf` — Oversampling factor

1 (default) | scalar greater than or equal to 1

Oversampling factor, specified as a scalar greater than or equal to 1. The oversampled cyclic prefix length must be an integer number of samples.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

y — VHT-SIG-B time-domain waveform

matrix

"VHT-SIG-B" on page 3-526 time-domain waveform, returned as an N_S-by-N_T matrix. N_S is the number of time-domain samples and N_T is the number of transmit antennas.

N_S is proportional to the channel bandwidth.

ChannelBandwidth	N_S
'CBW20'	80
'CBW40'	160
'CBW80'	320
'CBW160'	640

See "VHT-SIG-B Processing" on page 3-527. for waveform generation details.

Data Types: `double`
Complex Number Support: Yes

`bits` — Signaling bits used for the VHT-SIG-B field

N_{bits} column vector

Signaling bits used for "VHT-SIG-B" on page 3-526 field, returned as an N_{bits} column vector. N_{bits} is the number of bits.

The number of output bits changes with the channel bandwidth.

ChannelBandwidth	N_b
'CBW20'	26
'CBW40'	27
'CBW80'	29
'CBW160'	29

See "VHT-SIG-B Processing" on page 3-527. for waveform generation details.

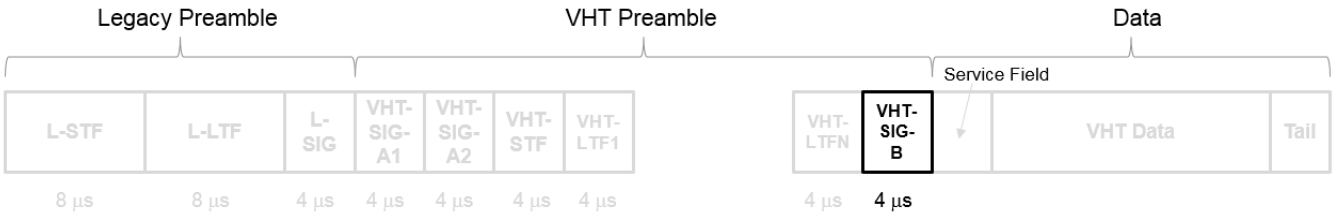
Data Types: `int8`

More About

VHT-SIG-B

The very high throughput signal B field (VHT-SIG-B) is used for multiuser scenario to set up the data rate and to fine-tune MIMO reception. It is modulated using MCS 0 and is transmitted in a single OFDM symbol.

The VHT-SIG-B field consists of a single OFDM symbol located between the VHT-LTF and the data portion of the VHT format PPDU.



The very high throughput signal B (VHT-SIG-B) field contains the actual rate and A-MPDU length value per user. For a detailed description of the VHT-SIG-B field, see section 21.3.8.3.6 of IEEE Std 802.11-2016. The number of bits in the VHT-SIG-B field varies with the channel bandwidth and the assignment depends on whether single user or multiuser scenario in allocated. For single user configurations, the same information is available in the L-SIG field but the VHT-SIG-B field is included for continuity purposes.

Field	VHT MU PPDU Allocation (bits)			VHT SU PPDU Allocation (bits)			Description
	20 MHz	40 MHz	80 MHz, 160 MHz	20 MHz	40 MHz	80 MHz, 160 MHz	
VHT-SIG-B	B0-15 (16)	B0-16 (17)	B0-18 (19)	B0-16 (17)	B0-18 (19)	B0-20 (21)	A variable-length field that indicates the size of the data payload in four-byte units. The length of the field depends on the channel bandwidth.

Field	VHT MU PPDU Allocation (bits)			VHT SU PPDU Allocation (bits)			Description
	20 MHz	40 MHz	80 MHz, 160 MHz	20 MHz	40 MHz	80 MHz, 160 MHz	
VHT-MCS	B16-19 (4)	B17-20 (4)	B19-22 (4)	N/A	N/A	N/A	A four-bit field that is included for multiuser scenarios only.
Reserved	N/A	N/A	N/A	B17–19 (3)	B19-20 (2)	B21-22 (2)	All ones
Tail	B20-25 (6)	B21-26 (6)	B23-28 (6)	B20-25 (6)	B21-26 (6)	B23-28 (6)	Six zero-bits used to terminate the convolutional code.
Total # bits	26	27	29	26	27	29	
Bit field repetition	1	2	4 *For 160 MHz, the 80 MHz channel is repeated twice.*	1	2	4 *For 160 MHz, the 80 MHz channel is repeated twice.*	

For a null data packet (NDP), the VHT-SIG-B bits are set according to Table 21-15 of IEEE Std 802.11-2016.

Algorithms

VHT-SIG-B Processing

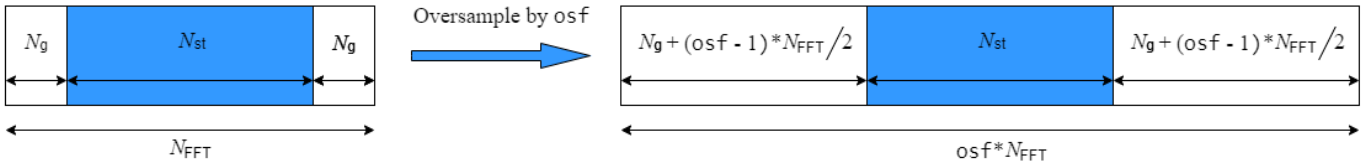
The “VHT-SIG-B” on page 3-526 field is used to set up the data rate and to fine-tune MIMO reception. For single user packets, since the length information can be recovered from the L-SIG and VHT-SIG-A field information, it is not strictly required for the receiver to decode the “VHT-SIG-B” on page 3-526 field.

For algorithm details, refer to IEEE Std 802.11ac-2013 [1], Section 22.3.4.8.

FFT-Based Oversampling

An oversampled signal is a signal sampled at a frequency that is higher than the Nyquist rate. WLAN signals maximize occupied bandwidth by using small guardbands, which can pose problems for anti-imaging and anti-aliasing filters. Oversampling increases guardband width relative to the total signal bandwidth, thereby increasing the number of samples in the signal.

This function performs oversampling by using a larger IFFT and zero pad when generating an OFDM waveform. This diagram shows the oversampling process for an OFDM waveform with N_{FFT} subcarriers comprising N_g guardband subcarriers on either side of N_{st} occupied bandwidth subcarriers.



Version History

Introduced in R2015b

References

[1] IEEE Std 802.11ac™-2013 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications — Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanVHTConfig` | `wlanVHTLTF` | `wlanVHTData` | `wlanVHTSIGBRecover`

wlanVHTSIGBRecover

Recover VHT-SIG-B information bits

Syntax

```
recBits = wlanVHTSIGBRecover(rxSig,chEst,noiseVarEst,cbw)
recBits = wlanVHTSIGBRecover(rxSig,chEst,noiseVarEst,cbw,userNumber,numSTS)
recBits = wlanVHTSIGBRecover( ___ ,Name,Value)

[recBits,eqSym] = wlanVHTSIGBRecover( ___ )
[recBits,eqSym,cpe] = wlanVHTSIGBRecover( ___ )
```

Description

`recBits = wlanVHTSIGBRecover(rxSig,chEst,noiseVarEst,cbw)` returns the recovered information bits from the "VHT-SIG-B" on page 3-536[25] field for a single-user transmission. Inputs include the received "VHT-SIG-B" on page 3-536 field, the channel estimate, the noise variance estimate, and the channel bandwidth.

`recBits = wlanVHTSIGBRecover(rxSig,chEst,noiseVarEst,cbw,userNumber,numSTS)` returns the recovered information bits of a multiuser transmission for the user of interest, `userNumber`, and the number of space-time streams, `numSTS`.

`recBits = wlanVHTSIGBRecover( ___ ,Name,Value)` specifies algorithm parameters by using one or more name-value pair arguments.

`[recBits,eqSym] = wlanVHTSIGBRecover( ___ )` returns the equalized symbols, `eqSym`, using the arguments from previous syntaxes.

`[recBits,eqSym,cpe] = wlanVHTSIGBRecover( ___ )` returns the common phase error, `cpe`.

Examples

Recover VHT-SIG-B Information Bits

Recover VHT-SIG-B bits in a perfect channel having 80 MHz channel bandwidth, one space-time stream, and one receive antenna.

Create a `wlanVHTConfig` object having a channel bandwidth of 80 MHz. Using the object, create a VHT-SIG-B waveform.

```
cfg = wlanVHTConfig('ChannelBandwidth','CBW80');
[txSig,txBits] = wlanVHTSIGB(cfg);
```

For a channel bandwidth of 80 MHz, there are 242 occupied subcarriers. The channel estimate array dimensions for this example must be [Nst,Nsts,Nr] = [242,1,1]. The example assumes a perfect channel and one receive antenna. Therefore, specify the channel estimate as a column vector of ones and the noise variance estimate as zero.

25 IEEE Std 802.11ac-2013 Adapted and reprinted with permission from IEEE. Copyright IEEE 2013. All rights reserved.

```
chEst = ones(242,1);
noiseVarEst = 0;
```

Recover the VHT-SIG-B. Verify that the received information bits are identical to the transmitted bits.

```
rxBits = wlanVHTSIGBRecover(txSig,chEst,noiseVarEst,'CBW80');
isequal(txBits,rxBits)
```

```
ans = logical
   1
```

Recover VHT-SIG-B Using Zero-Forcing Equalizer

Recover the VHT-SIG-B field using a zero-forcing equalizer in an AWGN channel with a channel bandwidth of 160 MHz, one space-time stream, and one receive antenna.

Create a `wlanVHTConfig` object, specifying a channel bandwidth of 160 MHz. Using the object, create a VHT-SIG-B waveform.

```
cfg = wlanVHTConfig('ChannelBandwidth','CBW160');
[txSig,txBits] = wlanVHTSIGB(cfg);
```

Pass the transmitted VHT-SIG-B through an AWGN channel.

```
noiseVarEst = 0.1;
awgnChan = comm.AWGNChannel('NoiseMethod','Variance','Variance',noiseVarEst);
rxSig = awgnChan(txSig);
```

Recover the VHT-SIG-B field, specifying zero-forcing equalization. Verify that the received information has no bit errors.

```
chEst = ones(484,1);
recBits = wlanVHTSIGBRecover(rxSig,chEst,noiseVarEst,'CBW160','EqualizationMethod','ZF');
numErr = biterr(txBits,recBits)
```

```
numErr = 0
```

Recover VHT-SIG-B in 2x2 MIMO Channel

Recover VHT-SIG-B in a 2x2 MIMO channel for an SNR=10 dB and a receiver that has a 9 dB noise figure. Confirm that the information bits are recovered correctly.

Set the channel bandwidth and the corresponding sample rate.

```
cbw = 'CBW20';
fs = 20e6;
```

Create a VHT configuration object with 20 MHz bandwidth and two transmission paths. Generate the L-LTF and VHT-SIG-B waveforms.

```
vht = wlanVHTConfig('ChannelBandwidth',cbw,'NumTransmitAntennas',2, ...
    'NumSpaceTimeStreams',2);
```

```
txVHTLTF = wlanVHTLTF(vht);
[txVHTSIGB,txVHTSIGBBits] = wlanVHTSIGB(vht);
```

Pass the VHT-LTF and VHT-SIG-B waveforms through a 2x2 TGac channel.

```
tgacChan = wlanTGacChannel('NumTransmitAntennas',2, ...
    'NumReceiveAntennas',2, 'ChannelBandwidth',cbw,'SampleRate',fs);
rxVHTLTF = tgacChan(txVHTLTF);
rxVHTSIGB = tgacChan(txVHTSIGB);
```

Add white noise for an SNR = 10dB.

```
chNoise = comm.AWGNChannel('NoiseMethod','Signal to noise ratio (SNR)',...
    'SNR',10);

rxVHTLTF = chNoise(rxVHTLTF);
rxVHTSIGB = chNoise(rxVHTSIGB);
```

Add additional white noise corresponding to a receiver with a 9 dB noise figure. The noise variance is equal to *k*T*B*F*, where *k* is Boltzmann's constant, *T* is the ambient temperature, *B* is the channel bandwidth (sample rate), and *F* is the receiver noise figure.

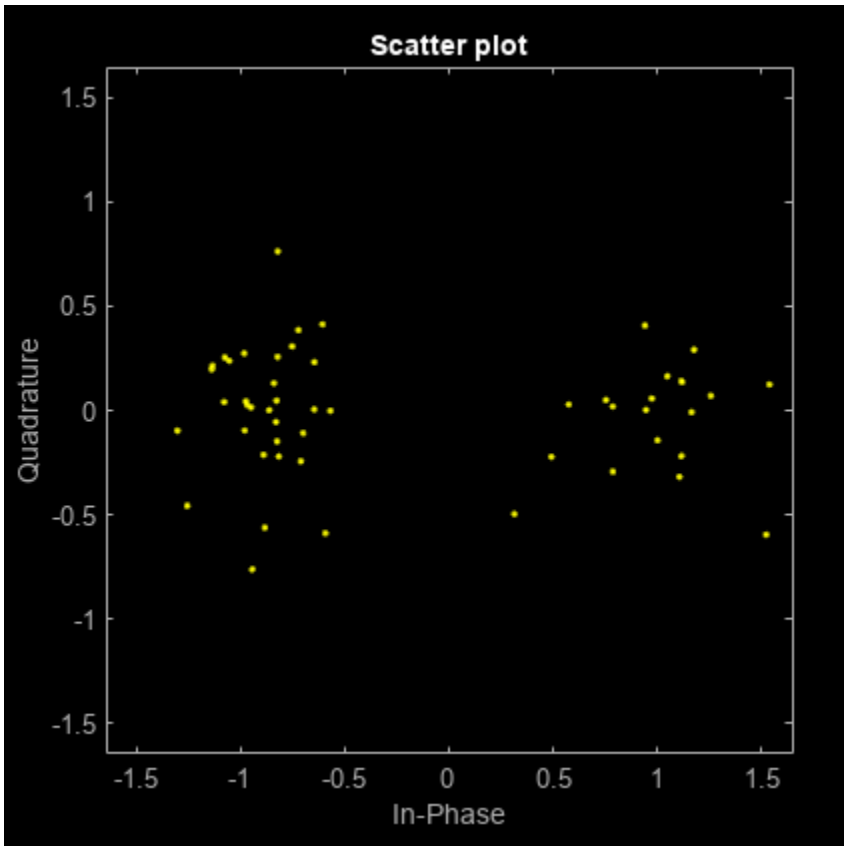
```
nVar = 10^((-228.6+10*log10(290)+10*log10(fs)+9)/10);
rxNoise = comm.AWGNChannel('NoiseMethod','Variance','Variance',nVar);

rxVHTLTF = rxNoise(rxVHTLTF);
rxVHTSIGB = rxNoise(rxVHTSIGB);
```

Demodulate the VHT-LTF signal and use it to generate a channel estimate.

```
demodVHTLTF = wlanVHTLTFDemodulate(rxVHTLTF,vht);
chEst = wlanVHTLTFChannelEstimate(demodVHTLTF,vht);
```

Recover the VHT-SIG-B information bits. Display the scatter plot of the equalized symbols.

```
[recVHTSIGBBits,eqSym,cpe] = wlanVHTSIGBRecover(rxVHTSIGB,chEst,nVar,cbw);
scatterplot(eqSym)
```



Display the common phase error.

```
cpe
```

```
cpe = 0.0485
```

Determine the number of errors between the transmitted and received VHT-SIG-B information bits.

```
numErr = biterr(txVHTSIGBBits,recVHTSIGBBits)
```

```
numErr = 0
```

Input Arguments

`rxSig` — Received VHT-SIG-B

matrix

Received VHT-SIG-B field, specified as an N_S-by-N_R matrix. N_S is the number of samples and increases with channel bandwidth.

Channel Bandwidth	N_S
`'CBW20'`	80
`'CBW40'`	160
`'CBW80'`	320

Channel Bandwidth	N_S
'CBW160'	640

N_R is the number of receive antennas.

Data Types: double

chEst — Channel estimate
3-D array

Channel estimate, specified as an N_{ST}-by-N_{STS}-by-N_R array. N_{ST} is the number of occupied subcarriers. N_{STS} is the number of space-time streams. For multiuser transmissions, N_{STS} is the total number of space-time streams for all users . N_R is the number of receive antennas.

N_{ST} increases with channel bandwidth.

ChannelBandwidth	Number of Occupied Subcarriers (N_{ST})	Number of Data Subcarriers (N_{SD})	Number of Pilot Subcarriers (N_{SP})
'CBW20'	56	52	4
'CBW40'	114	108	6
'CBW80'	242	234	8
'CBW160'	484	468	16

The channel estimate is based on the “VHT-LTF” on page 3-537.

noiseVarEst — Noise variance estimate
nonnegative scalar

Noise variance estimate, specified as a nonnegative scalar.

Data Types: double

cbw — Channel bandwidth
'CBW20' | 'CBW40' | 'CBW80' | 'CBW160'

Channel bandwidth, specified as 'CBW20', 'CBW40', 'CBW80', or 'CBW160'.

Data Types: char | string

userNumber — Number of the user
integer from 1 to N_{Users}

Number of the user in a multiuser transmission, specified as an integer having a value from 1 to N_{Users}. N_{Users} is the total number of users.

Data Types: double

numSTS — Number of space-time streams
1-by-N_{Users} vector of integers from 1 to 4

Number of space-time streams in a multiuser transmission, specified as a vector. The number of space-time streams is a 1-by-N_{Users} vector of integers from 1 to 4, where N_{Users} is an integer from 1 to 4.

Example: [1 3 2] is the number of space-time streams for each user.

Note The sum of the space-time stream vector elements must not exceed eight.

Data Types: `double`

Name-Value Pair Arguments

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

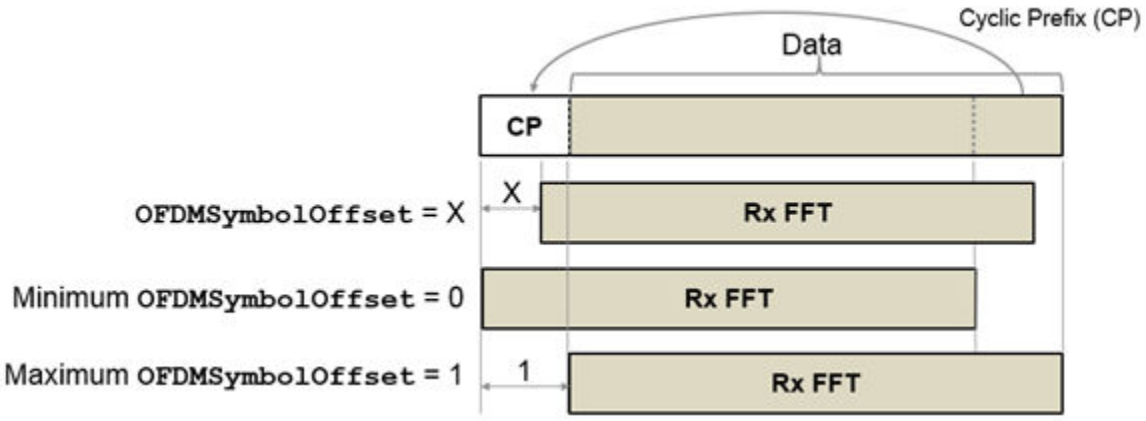
Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'PilotPhaseTracking','None'` disables pilot phase tracking.

`OFDMSymbolOffset` — OFDM symbol sampling offset

`0.75` (default) | scalar in the interval [0, 1]

OFDM symbol sampling offset represented as a fraction of the cyclic prefix (CP) length, specified as the comma-separated pair consisting of `'OFDMSymbolOffset'` and a scalar in the interval [0, 1]. The value you specify indicates the start location for OFDM demodulation relative to the beginning of the CP. The value `0` represents the start of the CP, and the value `1` represents the end of the CP.



Data Types: `double`

`EqualizationMethod` — Equalization method

`'MMSE'` (default) | `'ZF'`

Equalization method, specified as one of these values.

- `'MMSE'` — The receiver uses a minimum mean-square error equalizer.
- `'ZF'` — The receiver uses a zero-forcing equalizer.

When the received signal has multiple receive antennas, the function exploits receiver diversity during equalization. When the number of transmitted space-time streams is one and you specify this argument as `'ZF'`, the function performs maximal-ratio combining.

Data Types: `char` | `string`

`PilotPhaseTracking` — Pilot phase tracking

`'PreEQ'` (default) | `'None'`

Pilot phase tracking, specified as the comma-separated pair consisting of `'PilotPhaseTracking'` and one of these values.

- `'PreEQ'` — Enable pilot phase tracking, which the function performs before any equalization operation.
- `'None'` — Disable pilot phase tracking.

Data Types: `char` | `string`

Output Arguments

`recBits` — Recovered VHT-SIG information

vector

Recovered VHT-SIG-B information bits, returned as an N_b-by-1 column vector. N_b is the number of recovered VHT-SIG-B information bits and increases with the channel bandwidth. The output is for a single user as determined by `userNumber`.

The number of output bits is proportional to the channel bandwidth.

`ChannelBandwidth`	N_b
`'CBW20'`	26
`'CBW40'`	27
`'CBW80'`	29
`'CBW160'`	29

See “VHT-SIG-B” on page 3-536 for information about the meaning of each bit in the field.

`eqSym` — Equalized symbols

matrix

Equalized symbols, returned as an N_{SD}-by-1 column vector. N_{SD} is the number of data subcarriers.

N_{SD} increases with the channel bandwidth.

`ChannelBandwidth`	N_{SD}
`'CBW20'`	52
`'CBW40'`	108
`'CBW80'`	234
`'CBW160'`	468

`cpe` — Common phase error

column vector

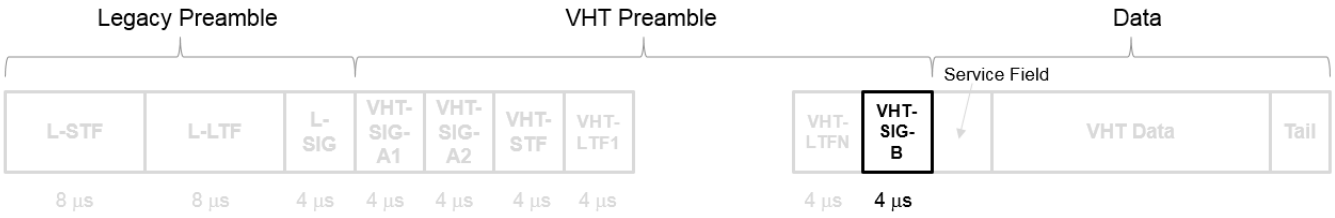
Common phase error in radians, returned as a scalar.

More About

VHT-SIG-B

The very high throughput signal B field (VHT-SIG-B) is used for multiuser scenario to set up the data rate and to fine-tune MIMO reception. It is modulated using MCS 0 and is transmitted in a single OFDM symbol.

The VHT-SIG-B field consists of a single OFDM symbol located between the VHT-LTF and the data portion of the VHT format PPDU.



The very high throughput signal B (VHT-SIG-B) field contains the actual rate and A-MPDU length value per user. For a detailed description of the VHT-SIG-B field, see section 21.3.8.3.6 of IEEE Std 802.11-2016. The number of bits in the VHT-SIG-B field varies with the channel bandwidth and the assignment depends on whether single user or multiuser scenario in allocated. For single user configurations, the same information is available in the L-SIG field but the VHT-SIG-B field is included for continuity purposes.

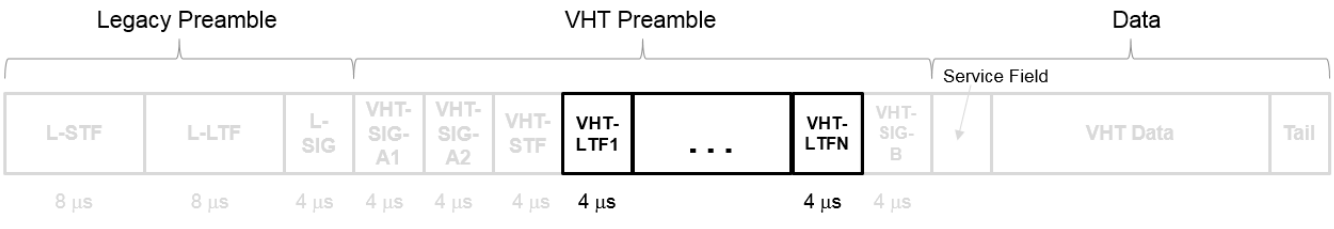
Field	VHT MU PPDU Allocation (bits)			VHT SU PPDU Allocation (bits)			Description
	20 MHz	40 MHz	80 MHz, 160 MHz	20 MHz	40 MHz	80 MHz, 160 MHz	
VHT-SIG-B	B0-15 (16)	B0-16 (17)	B0-18 (19)	B0-16 (17)	B0-18 (19)	B0-20 (21)	A variable-length field that indicates the size of the data payload in four-byte units. The length of the field depends on the channel bandwidth.

Field	VHT MU PPDU Allocation (bits)			VHT SU PPDU Allocation (bits)			Description
	20 MHz	40 MHz	80 MHz, 160 MHz	20 MHz	40 MHz	80 MHz, 160 MHz	
VHT-MCS	B16-19 (4)	B17-20 (4)	B19-22 (4)	N/A	N/A	N/A	A four-bit field that is included for multiuser scenarios only.
Reserved	N/A	N/A	N/A	B17–19 (3)	B19-20 (2)	B21-22 (2)	All ones
Tail	B20-25 (6)	B21-26 (6)	B23-28 (6)	B20-25 (6)	B21-26 (6)	B23-28 (6)	Six zero-bits used to terminate the convolutional code.
Total # bits	26	27	29	26	27	29	
Bit field repetition	1	2	4 *For 160 MHz, the 80 MHz channel is repeated twice.*	1	2	4 *For 160 MHz, the 80 MHz channel is repeated twice.*	

For a null data packet (NDP), the VHT-SIG-B bits are set according to Table 21-15 of IEEE Std 802.11-2016.

VHT-LTF

The very high throughput long training field (VHT-LTF) is located between the VHT-STF and VHT-SIG-B portion of the VHT packet.



It is used for MIMO channel estimation and pilot subcarrier tracking. The VHT-LTF includes one VHT long training symbol for each spatial stream indicated by the selected MCS. Each symbol is 4 µs long. A maximum of eight symbols are permitted in the VHT-LTF.

For a detailed description of the VHT-LTF, see section 21.3.8.3.5 of IEEE Std 802.11-2016.

PPDU

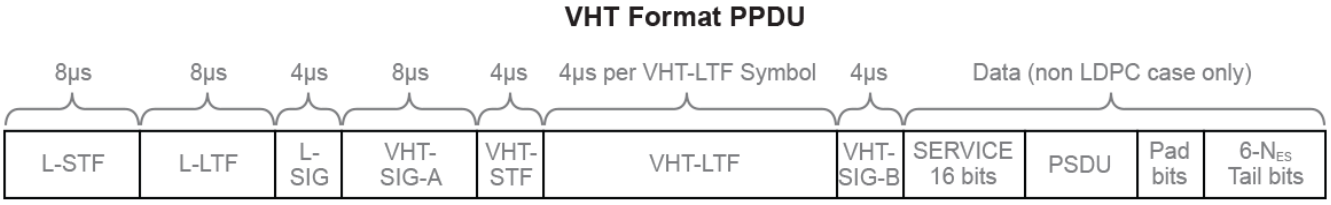
PLCP protocol data unit

The PPDU is the complete PLCP frame, including PLCP headers, MAC headers, the MAC data field, and the MAC and PLCP trailers.

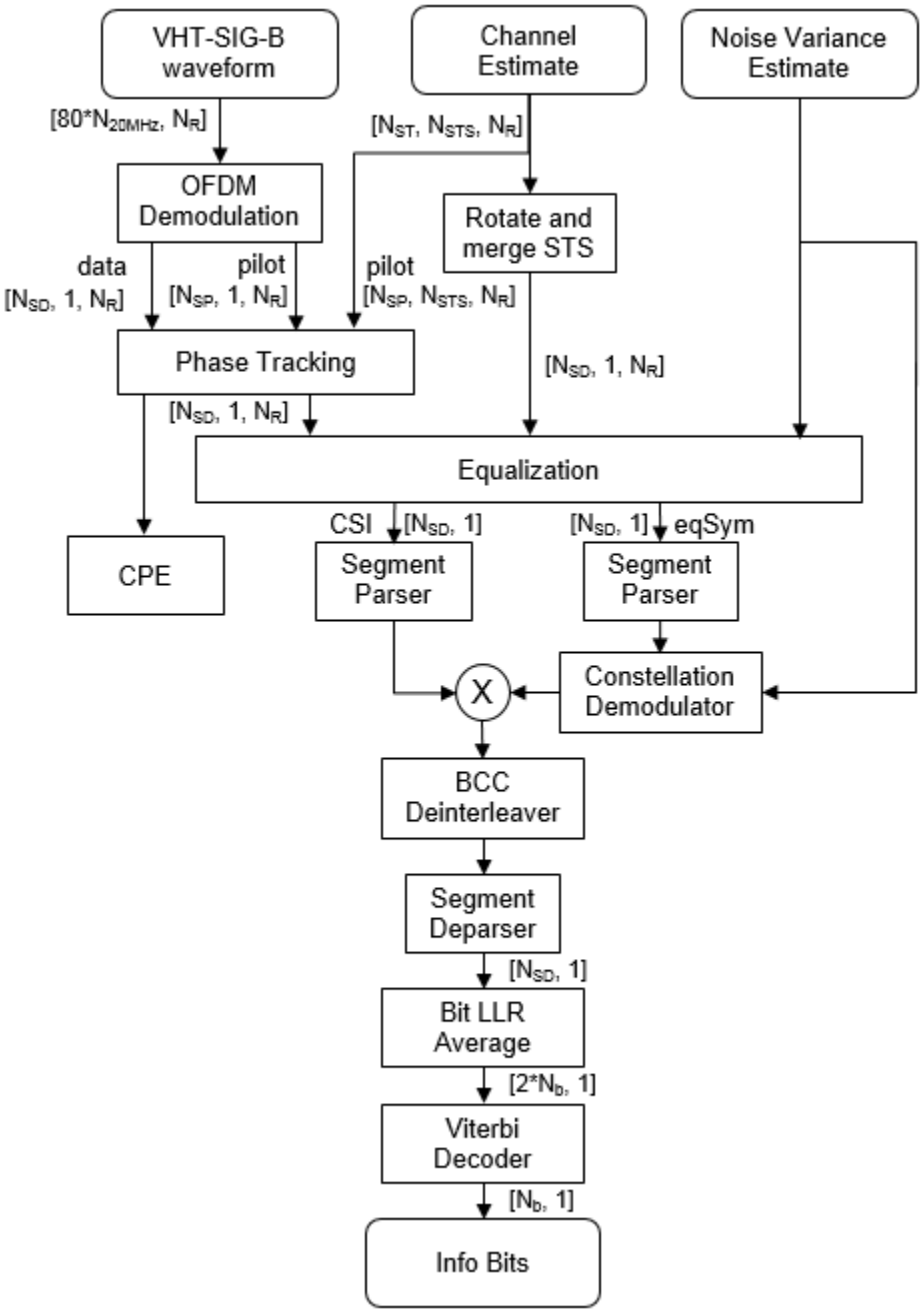
Algorithms

VHT-SIG-B Recovery

The “VHT-SIG-B” on page 3-536 field consists of one symbol and resides between the VHT-LTF field and the data portion of the packet structure for the VHT format PPDUs.



For single-user packets, you can recover the length information from the L-SIG and VHT-SIG-A field information. Therefore, it is not strictly required for the receiver to decode the “VHT-SIG-B” on page 3-536 field. For multiuser transmissions, recovering the VHT-SIG-B field provides packet length and MCS information for each user.



For “VHT-SIG-B” on page 3-536 details, refer to IEEE Std 802.11ac™-2013 [1], Section 22.3.4.8, and Perahia [2], Section 7.3.2.4.

Version History

Introduced in R2015b

References

[1] IEEE Std 802.11ac™-2013 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications — Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz.

[2] Perahia, E., and R. Stacey. *Next Generation Wireless LANs: 802.11n and 802.11ac* . 2nd Edition, United Kingdom: Cambridge University Press, 2013.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanVHTSIGB` | `wlanVHTConfig` | `wlanVHTLTFDemodulate` | `wlanVHTLTFChannelEstimate` | `wlanVHTDataRecover` | `wlanVHTSIGARecover`

wlanVHTSTF

Generate VHT-STF waveform

Syntax

```
y = wlanVHTSTF(cfg)
y = wlanVHTSTF(cfg,OversamplingFactor=osf)
```

Description

`y = wlanVHTSTF(cfg)` generates a "VHT-STF" on page 3-543[26] time-domain waveform for the specified transmission parameters. See "VHT-STF Processing" on page 3-543 for waveform generation details.

`y = wlanVHTSTF(cfg,OversamplingFactor=osf)` generates an oversampled VHT-STF waveform with the specified oversampling factor. For more information about oversampling, see "FFT-Based Oversampling" on page 3-543.

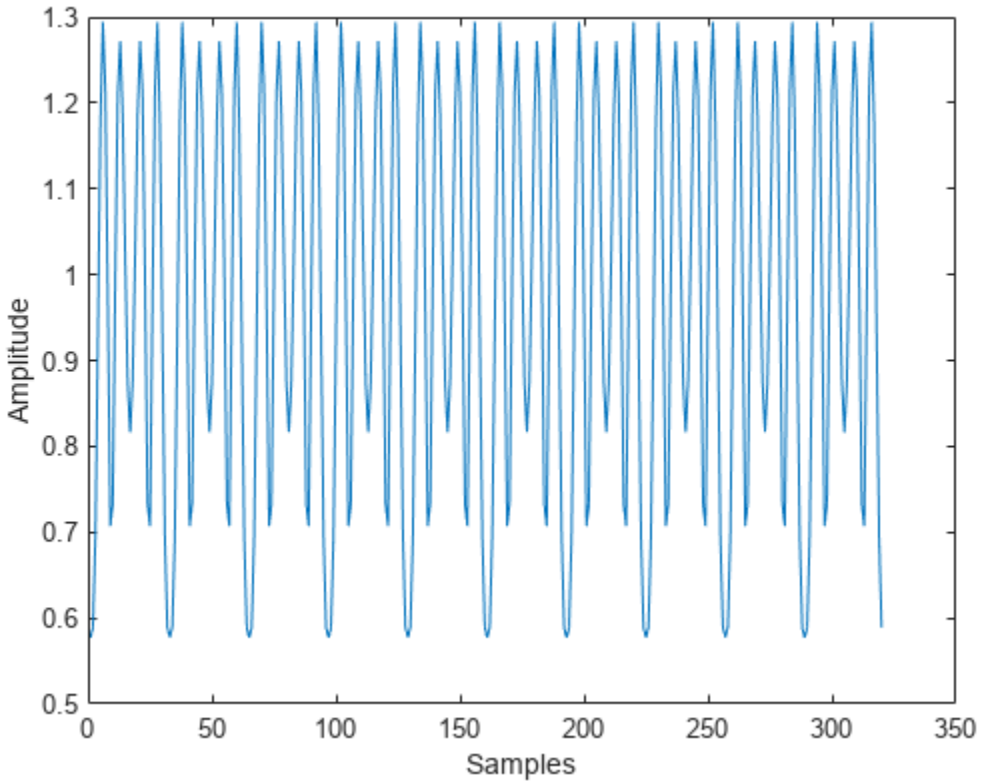
Examples

Generate VHT-STF Waveform

Create a VHT configuration object with an 80 MHz channel bandwidth. Generate and plot the VHT-STF waveform.

```
cfgVHT = wlanVHTConfig;
cfgVHT.ChannelBandwidth = 'CBW80';

vstfOut = wlanVHTSTF(cfgVHT);
size(vstfOut);
plot(abs(vstfOut))
xlabel('Samples')
ylabel('Amplitude')
```

26 IEEE Std 802.11ac-2013 Adapted and reprinted with permission from IEEE. Copyright IEEE 2013. All rights reserved.



The 80 MHz waveform is a single OFDM symbol with 320 complex time-domain output samples. The waveform contains the repeating short training field pattern.

Input Arguments

`cfg` — Format configuration

`wlanVHTConfig` object

Format configuration, specified as a `wlanVHTConfig` object.

`osf` — Oversampling factor

1 (default) | scalar greater than or equal to 1

Oversampling factor, specified as a scalar greater than or equal to 1. The oversampled cyclic prefix length must be an integer number of samples. The resultant inverse fast Fourier transform (IFFT) length must be even.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`y` — VHT-STF time-domain waveform

matrix

“VHT-STF” on page 3-543 time-domain waveform, returned as an N_S-by-N_T matrix. N_S is the number of time-domain samples, and N_T is the number of transmit antennas.

N_S is proportional to the channel bandwidth.

`ChannelBandwidth`	N_S
`'CBW20'`	80
`'CBW40'`	160
`'CBW80'`	320
`'CBW160'`	640

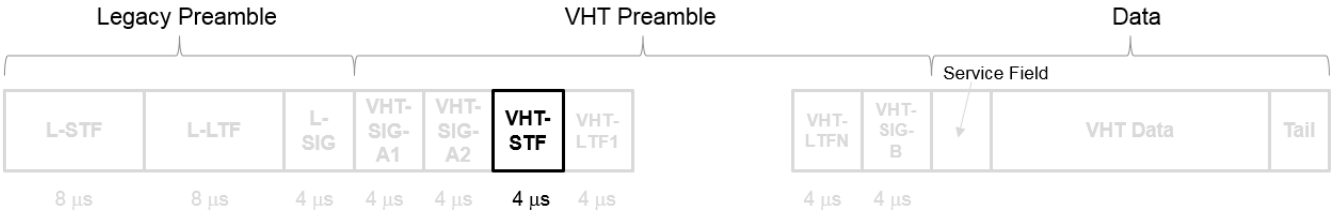
See “VHT-STF Processing” on page 3-543 for waveform generation details.

Data Types: `double`
Complex Number Support: Yes

More About

VHT-STF

The very high throughput short training field (VHT-STF) is a single OFDM symbol (4 µs in length) that is used to improve automatic gain control estimation in a MIMO transmission. It is located between the VHT-SIG-A and VHT-LTF portions of the VHT packet.



The frequency domain sequence used to construct the VHT-STF for a 20 MHz transmission is identical to the L-STF sequence. Duplicate L-STF sequences are frequency shifted and phase rotated to support VHT transmissions for the 40 MHz, 80 MHz, and 160 MHz channel bandwidths. As such, the L-STF and HT-STF are subsets of the VHT-STF.

For a detailed description of the VHT-STF, see section 21.3.8.3.4 of IEEE Std 802.11-2016.

Algorithms

VHT-STF Processing

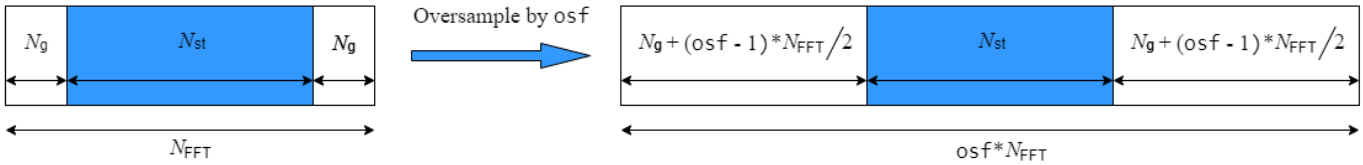
The “VHT-STF” on page 3-543 is one OFDM symbol long and is processed for improved gain control in MIMO configurations. For algorithm details, refer to IEEE Std 802.11ac-2013 [1], Section 22.3.4.6.

FFT-Based Oversampling

An oversampled signal is a signal sampled at a frequency that is higher than the Nyquist rate. WLAN signals maximize occupied bandwidth by using small guardbands, which can pose problems for anti-

imaging and anti-aliasing filters. Oversampling increases guardband width relative to the total signal bandwidth, thereby increasing the number of samples in the signal.

This function performs oversampling by using a larger IFFT and zero pad when generating an OFDM waveform. This diagram shows the oversampling process for an OFDM waveform with N_{FFT} subcarriers comprising N_g guardband subcarriers on either side of N_{st} occupied bandwidth subcarriers.



Version History

Introduced in R2015b

References

[1] IEEE Std 802.11ac™-2013 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications — Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanVHTConfig` | `wlanLSTF` | `wlanVHTLTF` | `wlanVHTSIGA`

wlanWaveformGenerator

Generate WLAN waveform

Syntax

```
waveform = wlanWaveformGenerator(bits,cfg)
waveform = wlanWaveformGenerator(bits,cfg,Name,Value)
```

Description

`waveform = wlanWaveformGenerator(bits,cfg)` generates a waveform for `bits`, the specified information bits, and `cfg`, the physical layer (PHY) format configuration. For more information, see “IEEE 802.11 PPDU Format” on page 3-557.

`waveform = wlanWaveformGenerator(bits,cfg,Name,Value)` specifies additional options using one or more name-value pair arguments.

Examples

Generate EHT MU Waveform

Create a configuration object for a non-OFDMA EHT MU packet. Set the channel bandwidth to 160 MHz.

```
cfgEHTMU = wlanEHTMUConfig("CBW160");
```

Obtain the PSDU length, in bytes, from the configuration object by using the `psduLength` object function.

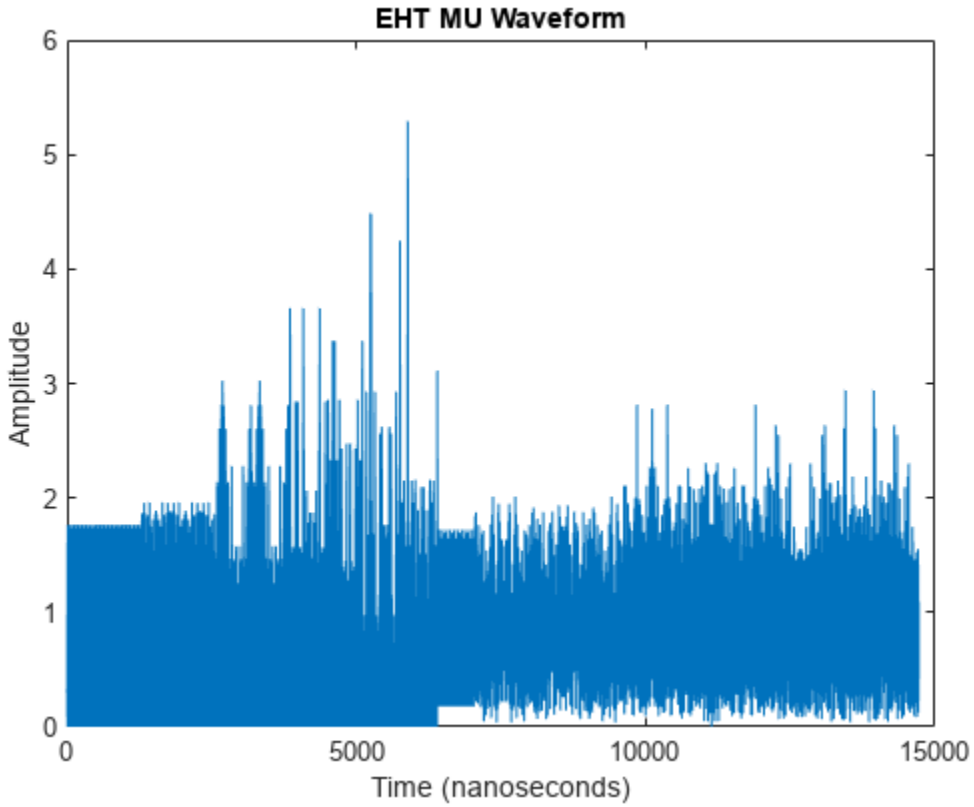
```
length = psduLength(cfgEHTMU);
```

Generate a PSDU of the relevant length.

```
psdu = randi([0 1],8*length,1);
```

Generate and plot the waveform.

```
waveform = wlanWaveformGenerator(psdu,cfgEHTMU);
figure;
plot(abs(waveform));
title('EHT MU Waveform');
xlabel('Time (nanoseconds)');
ylabel('Amplitude');
```



Generate HE TB Waveform

Configure and generate a WLAN waveform containing an HE TB uplink packet.

Create a configuration object for a WLAN HE TB uplink transmission.

```
cfgHETB = wlanHETBConfig;
```

Obtain the PSDU length, in bytes, from the configuration object by using the `getPSDULength` object function.

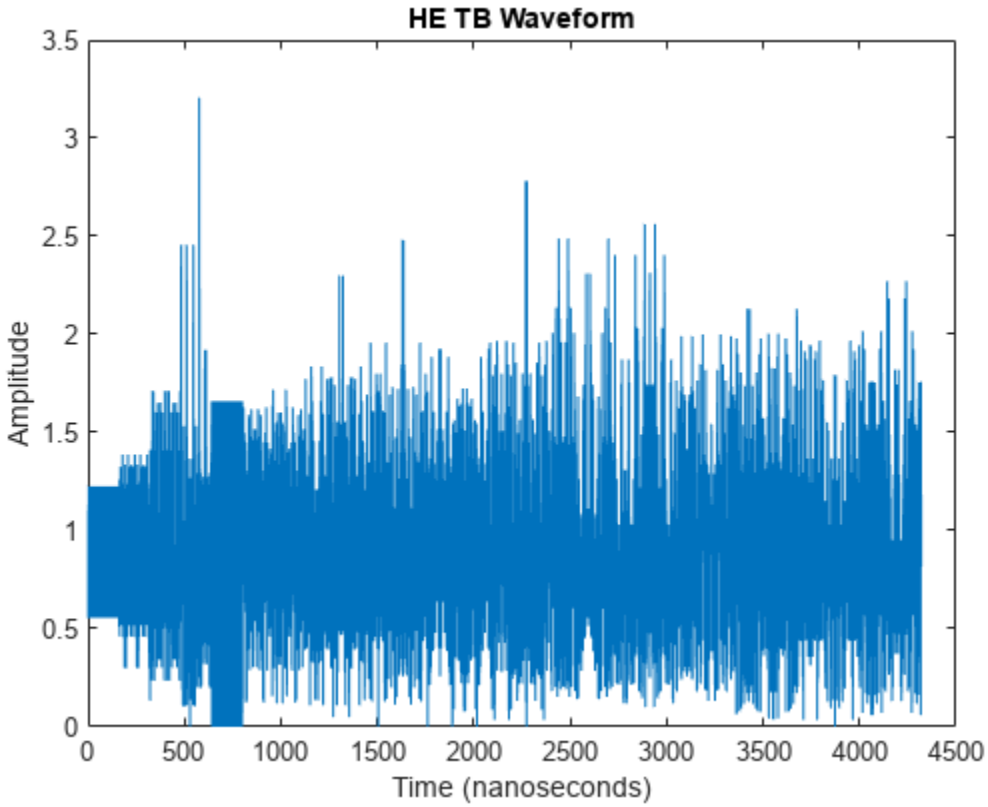
```
psduLength = getPSDULength(cfgHETB);
```

Generate a PSDU of the relevant length.

```
psdu = randi([0 1],8*psduLength,1);
```

Generate and plot the waveform.

```
waveform = wlanWaveformGenerator(psdu,cfgHETB);
figure;
plot(abs(waveform));
title('HE TB Waveform');
xlabel('Time (nanoseconds)');
ylabel('Amplitude');
```



Generate VHT Waveform

Generate a time-domain signal for an 802.11ac VHT transmission with one packet.

Create a VHT configuration object. Assign two transmit antennas and two spatial streams, and disable space-time block coding (STBC). Set the modulation and coding scheme to `1`, which assigns QPSK modulation and a 1/2 rate coding scheme per the 802.11 standard. Set the number of bytes in the A-MPDU pre-EOF padding, `APEPLength`, to `1024`.

```
cfg = wlanVHTConfig('NumTransmitAntennas',2,'NumSpaceTimeStreams',2,'STBC',0,'MCS',1,'APEPLength
```

Generate the transmit waveform.

```
bits = [1;0;0;1];
txWaveform = wlanWaveformGenerator(bits,cfg);
```

Demonstrate SIGB Compression in HE MU Waveforms

HE MU-MIMO Configuration With SIGB Compression

Generate a full bandwidth HE MU-MIMO configuration at 20 MHz bandwidth with SIGB compression. All three users are on a single content channel, which includes only the user field bits.

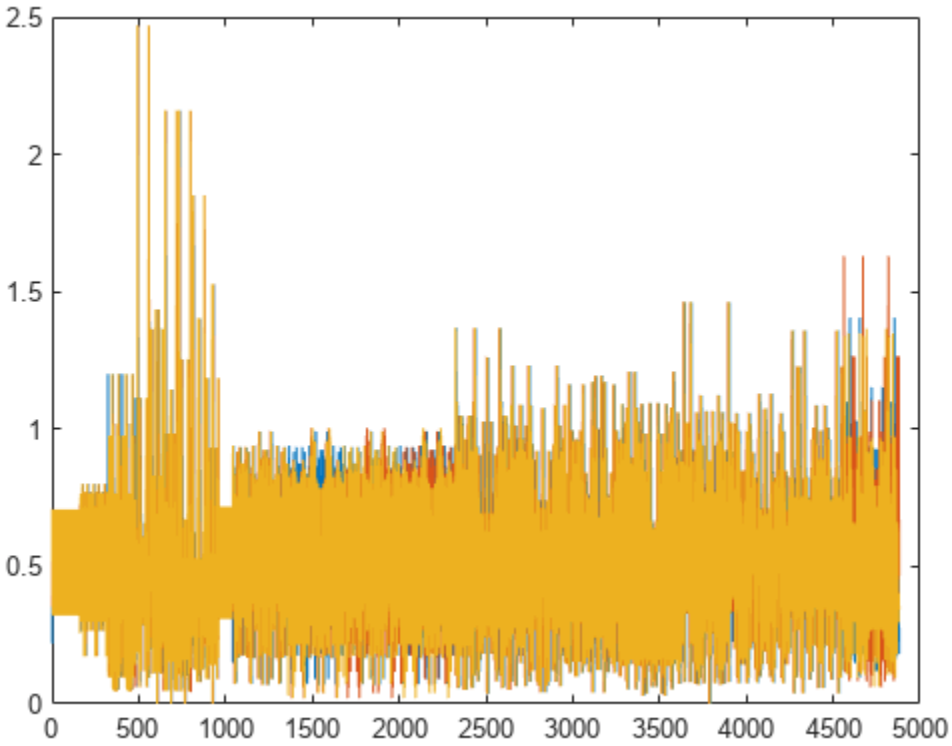
```
cfgHE = wlanHEMUConfig(194);
cfgHE.NumTransmitAntennas = 3;
```

Create PSDU data for all users.

```
psdu = cell(1,numel(cfgHE.User));
psduLength = getPSDULength(cfgHE);
for j = 1:numel(cfgHE.User)
    psdu = randi([0 1],psduLength(j)*8,1,'int8');
end
```

Generate and plot the waveform.

```
y = wlanWaveformGenerator(psdu,cfgHE);
plot(abs(y))
```



Generate a full bandwidth HE MU-MIMO waveform at 80 MHz bandwidth with SIGB compression. HE-SIG-B content channel 1 has four users. HE-SIG-B content channel 2 has three users.

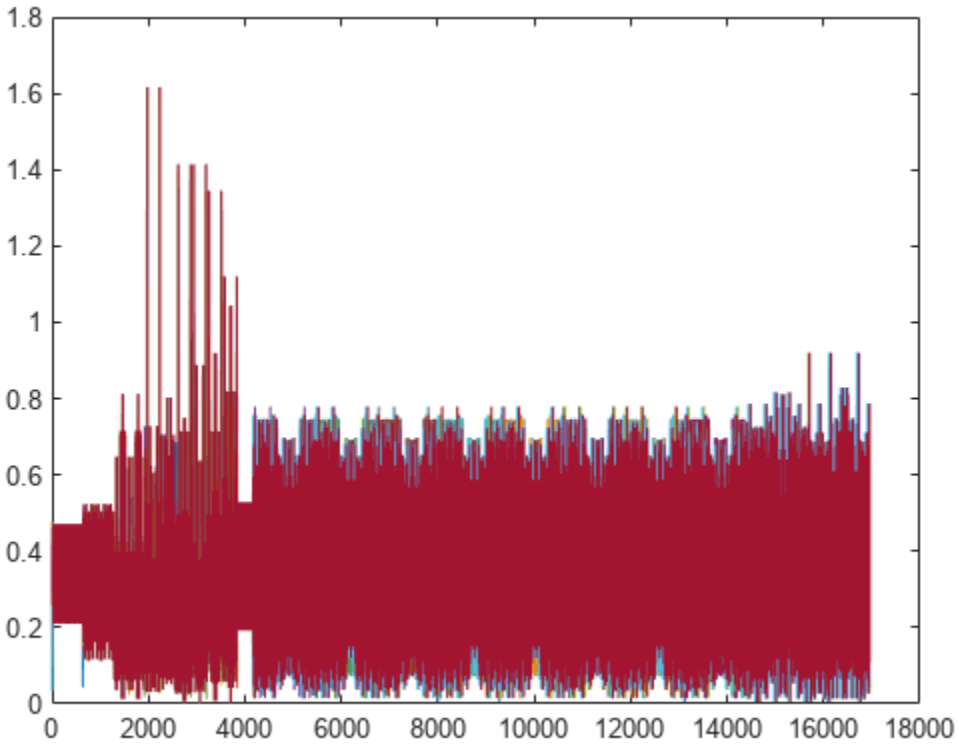
```
cfgHE = wlanHEMUConfig(214);
cfgHE.NumTransmitAntennas = 7;
```

Create PSDU data for all users.

```
psdu = cell(1,numel(cfgHE.User));
psduLength = getPSDULength(cfgHE);
for j = 1:numel(cfgHE.User)
    psdu = randi([0 1],psduLength(j)*8,1,'int8');
end
```

Generate and plot the waveform.

```
y = wlanWaveformGenerator(psdu,cfgHE);
plot(abs(y));
```



HE MU-MIMO Configuration Without SIGB Compression

Generate a full bandwidth HE MU-MIMO configuration at 20 MHz bandwidth without SIGB compression. All three users are on a single content channel, which includes both common and user field bits.

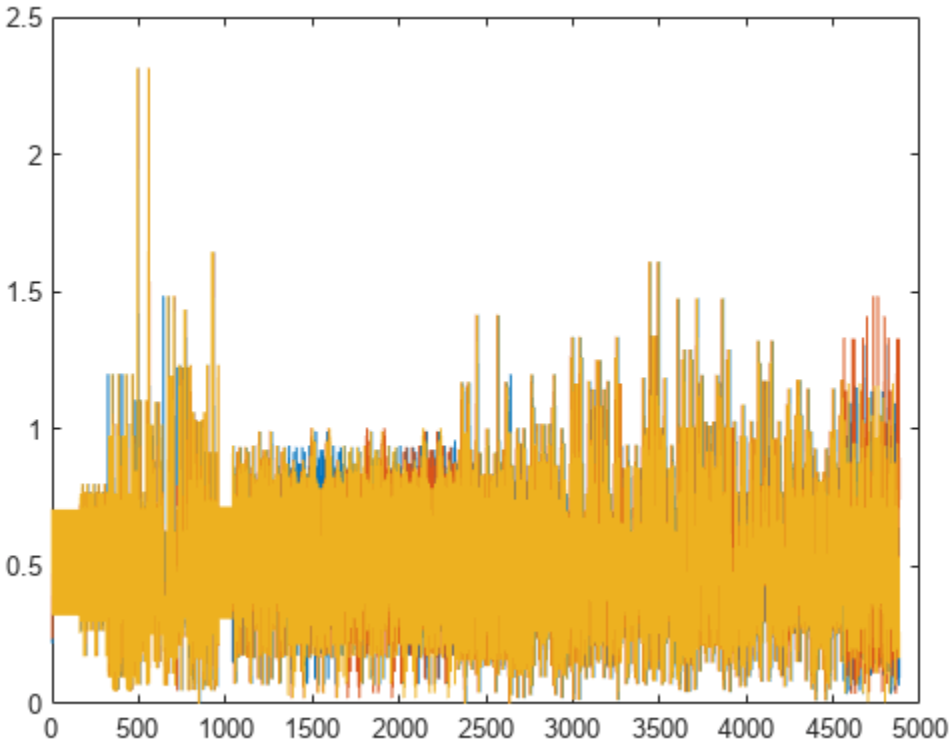
```
cfgHE = wlanHEMUConfig(194);
cfgHE.SIGBCompression = false;
cfgHE.NumTransmitAntennas = 3;
```

Create PSDU data for all users.

```
psdu = cell(1,numel(cfgHE.User));
psduLength = getPSDULength(cfgHE);
for j = 1:numel(cfgHE.User)
    psdu = randi([0 1],psduLength(j)*8,1,'int8');
end
```

Generate and plot the waveform.

```
y = wlanWaveformGenerator(psdu,cfgHE);
plot(abs(y))
```



Generate an 80 MHz HE MU waveform for six users without SIGB compression. HE-SIG-B content channel 1 has four users. HE-SIG-B content channel 2 has two users.

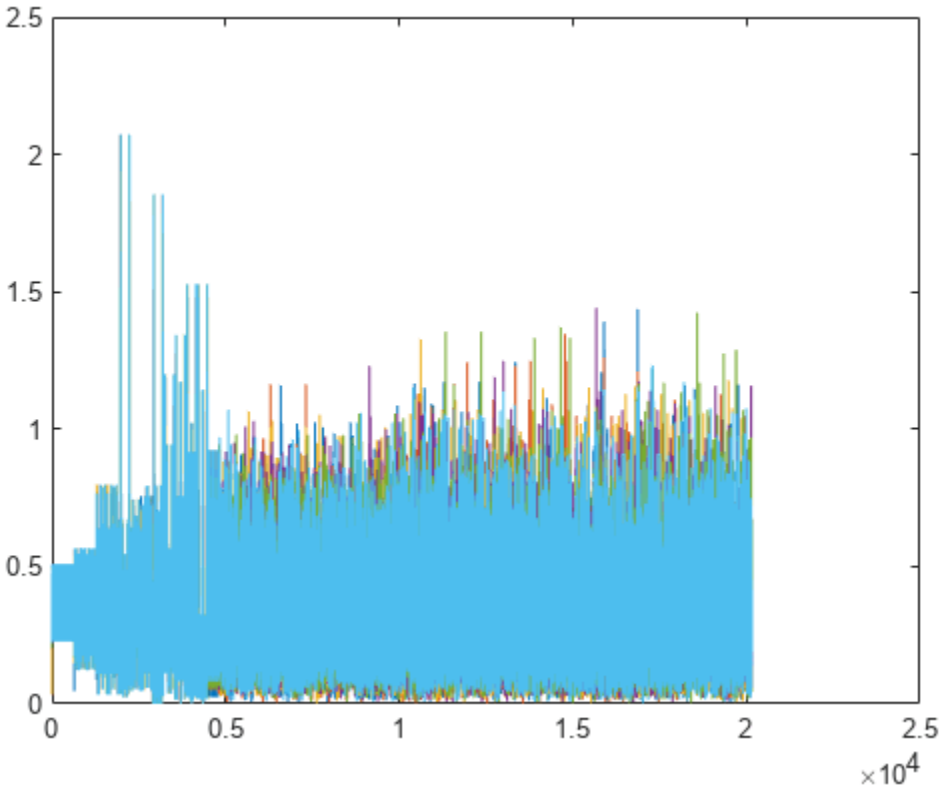
```
cfgHE = wlanHEMUConfig([202 114 192 193]);
cfgHE.NumTransmitAntennas = 6;
for i = 1:numel(cfgHE.RU)
    cfgHE.RU{i}.SpatialMapping = 'Fourier';
end
```

Create PSDU data for all users.

```
psdu = cell(1,numel(cfgHE.User));
psduLength = getPSDULength(cfgHE);
for j = 1:numel(cfgHE.User)
    psdu = randi([0 1],psduLength(j)*8,1,'int8');
end
```

Generate and plot the waveform.

```
y = wlanWaveformGenerator(psdu,cfgHE);
plot(abs(y));
```



Generate a full bandwidth HE MU-MIMO waveform at 80 MHz bandwidth without SIGB compression. HE-SIG-B content channel 1 has seven users. HE-SIG-B content channel 2 has zero users.

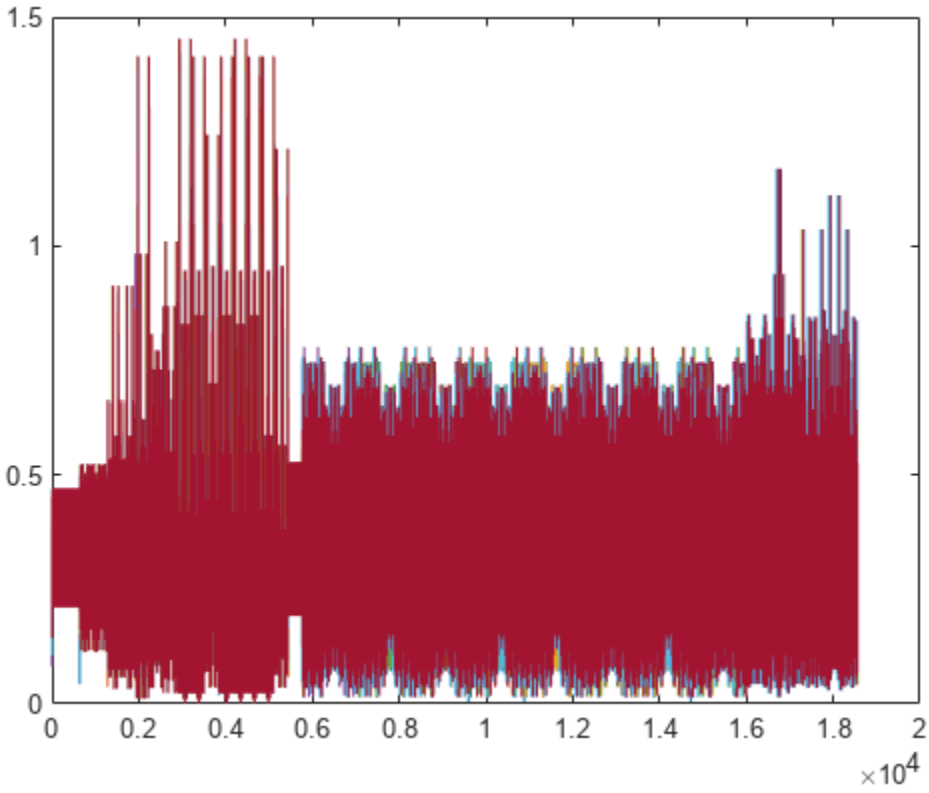
```
cfgHE = wlanHEMUConfig([214 115 115 115]);
cfgHE.NumTransmitAntennas = 7;
```

Create PSDU data for all users.

```
psdu = cell(1,numel(cfgHE.User));
psduLength = getPSDULength(cfgHE);
for j = 1:numel(cfgHE.User)
    psdu = randi([0 1],psduLength(j)*8,1,'int8');
end
```

Generate and plot the waveform.

```
y = wlanWaveformGenerator(psdu,cfgHE);
plot(abs(y))
```



Generate VHT Waveform with Random Scrambler State

Generate a time-domain signal for an 802.11ac VHT transmission with five packets and a 30-microsecond idle period between packets. Use a random scrambler initial state for each packet.

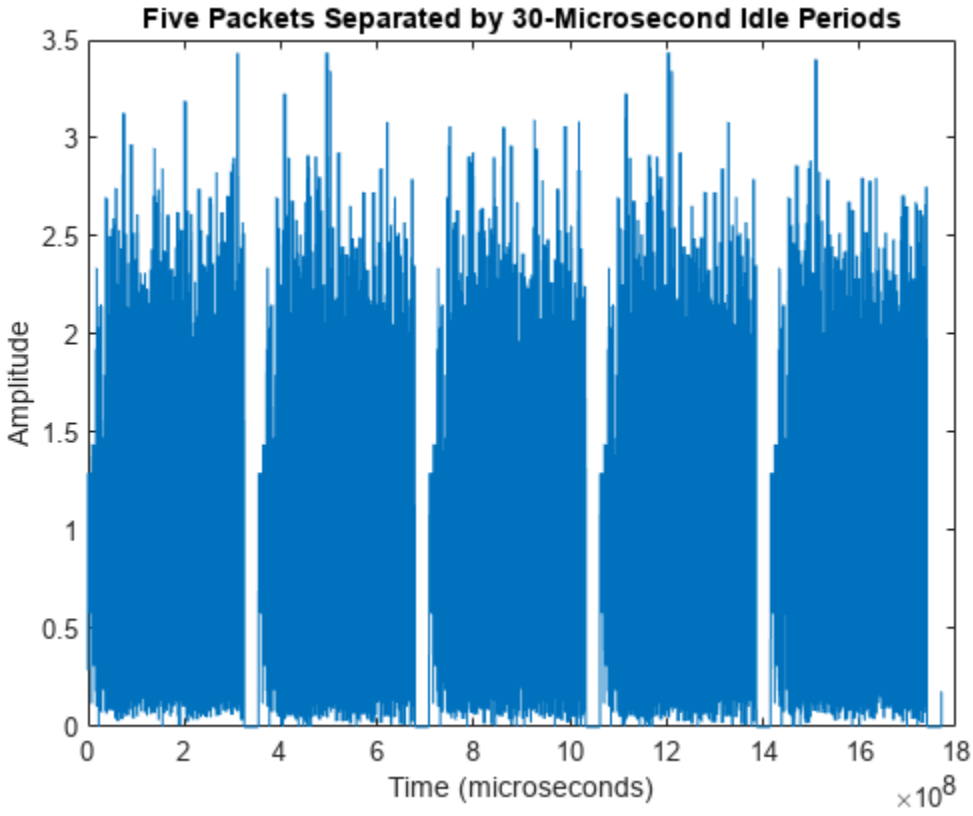
Create a VHT configuration object and confirm the channel bandwidth for scaling the *x*-axis of the plot.

```
cfg = wlanVHTConfig;
disp(cfg.ChannelBandwidth)
```

```
CBW80
```

Generate and plot the waveform. Display the time in microseconds on the *x*-axis.

```
numPkts = 5;
bits = [1;0;0;1];
scramInit = randi([1 127],numPkts,1);
txWaveform = wlanWaveformGenerator(bits,cfg,'NumPackets',numPkts,'IdleTime',30e-6,'ScramblerInit
time = [0:length(txWaveform)-1]/80e-6;
plot(time,abs(txWaveform));
title('Five Packets Separated by 30-Microsecond Idle Periods');
xlabel ('Time (microseconds)');
ylabel('Amplitude');
```



Input Arguments

`bits` — Information bits

`0` | `1` | binary-valued vector | cell array | vector cell array

Information bits for a single user, including any MAC padding representing multiple concatenated PSDUs, specified as one of these values.

- `0` or `1`.
- A binary-valued vector.
- A one-by-one cell containing a binary-valued scalar or vector– The specified bits apply to all users.
- A vector cell array of binary-valued scalars or vectors – Each element applies to each user correspondingly. The length of this cell array must be equal to the number of users. For each user, if the number of bits required across all packets of the generation exceeds the length of the vector provided, the function loops the applied bit vector. Looping on the bits allows you to define a short pattern, for example, `[1; 0; 0; 1]`, that is repeated as the input to the PSDU coding across packets and users. In each packet generation, for the *k*th user, the *k*th element of the `PSDULength` property of the `cfg` input indicates the number of data bytes taken from its stream. To compute the number of bits, multiply `PSDULength` by 8.

Internally, the function loops this input to generate the specified number of packets. The `PSDULength` property of the `cfg` input specifies the number of data bits taken from the bit stream

for each transmission packet generated. The `'NumPackets'` input specifies the number of packets to generate.

Example: `[1 1 0 1 0 1 1]`

Data Types: `double` | `int8`

`cfg` — Packet format configuration

`wlanHEMUConfig` object | `wlanHESUConfig` object | `wlanHETBConfig` object | `wlanEHTMUConfig` object | `wlanWURConfig` object | `wlanDMGConfig` object | `wlanS1GConfig` object | `wlanVHTConfig` object | `wlanHTConfig` object | `wlanNonHTConfig` object

Packet format configuration, specified as one of these objects: `wlanHEMUConfig`, `wlanHESUConfig`, `wlanHETBConfig`, `wlanEHTMUConfig`, `wlanWURConfig`, `wlanDMGConfig`, `wlanS1GConfig`, `wlanVHTConfig`, `wlanHTConfig`, or `wlanNonHTConfig`. The type of object you specify determines the IEEE 802.11 format of the generated waveform.

The properties of the packet format configuration object determine the data rate and PSDU length of generated PPDUs.

Name-Value Pair Arguments

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

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

Example: `'NumPackets',21,'ScramblerInitialization',[52,17]`

`NumPackets` — Number of packets

`1` (default) | positive integer

Number of packets to generate in a single function call, specified as a positive integer.

Data Types: `double`

`IdleTime` — Idle time added after each packet

`0` (default) | nonnegative scalar

Idle time, in seconds, added after each packet, specified as a nonnegative scalar. Except for the default value, this input must be greater than or equal to:

- `1e-6` for DMG format
- `2e-6` for all other formats

Example: `2e-5`

Data Types: `double`

`OversamplingFactor` — Oversampling factor

`1` (default) | scalar greater than or equal to 1

Oversampling factor, specified as a scalar greater than or equal to 1. The oversampled cyclic prefix length must be an integer number of samples. For more information about oversampling, see “FFT-Based Oversampling” on page 3-563.

Dependencies

This argument applies only for EHT, HE, WUR, VHT, HT, S1G, and non-HT OFDM formats.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`ScramblerInitialization` — Initial scrambler state or initial pseudorandom scrambler sequence

93 (default) | integer in the interval [1, 2047] | matrix of integers in the interval [1, 2047]

Initial scrambler state or initial pseudorandom scrambler sequence for each generated packet and each user, specified as one of these values.

- An integer in the interval [1, 127] — This input represents the initial scrambler state for all packets and users in HE, S1G, VHT, and HT waveforms, and non-HT OFDM waveforms with bandwidth signaling disabled. For multi-user and multipacket waveforms, the function uses the value you specify for all packets and users. The default value, `93`, is the example state in Section I.1.5.2 of [1]. For more information, see “Scrambler Initialization” on page 3-563.
- An integer in the interval [1, 2047]. This input represents the initial scrambler state for all packets and users in EHT waveforms. For multi-user and multipacket waveforms, the function uses the value you specify for all packets and users.
- An integer in the interval [*min*, *max*] — This input represents the initial pseudorandom scrambler sequence of a non-HT transmission with bandwidth signaling enabled, described in Table 17-7 of [1]. If you do not specify this input, the function uses the N_B most significant bits of the default value, `93`. The values of *min*, *max*, and N_B depend on the values of the `BandwidthOperation` and `ChannelBandwidth` properties of the `cfg` input according to this table.

Value of `cfg.BandwidthOperation`	Value of `cfg.ChannelBandwidth`	Value of *min*	Value of *max*	Value of N_B
`'Absent'`	`'CBW20'`	1	31	5
`'Absent'`	`'CBW5'`, `'CBW10'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`	0	31	5
`'Static'` or `'Dynamic'`	`'CBW20'`	1	15	4
`'Static'` or `'Dynamic'`	`'CBW5'`, `'CBW10'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`	0	15	4

- A matrix of integers in the interval [1, 127], of size N_P-by-N_{Users} — Each element represents an initial state of the scrambler for each packet and for each user in VHT, S1G, and HE multi-user (MU) waveforms comprising multiple packets. Each column specifies the initial states for a single user. You can specify up to eight columns for HE MU waveforms, or up to four columns for VHT and S1G. If you specify a single column, the function uses the same initial states for all users. Each row represents the initial state of each packet to generate. A matrix with multiple rows enables you to use a different initial state per packet, where the first row contains the initial state

of the first packet. If the number of packets to generate exceeds the number of rows of the matrix provided, the function loops the rows internally.

  - N_P is the number of packets.
  - N_{Users} is the number of users.

- A matrix of integers in the interval [1, 2047], of size N_P-by-N_{Users} — Each element represents an initial state of the scrambler for each packet and for each user in EHT multi-user (MU) waveforms comprising multiple packets. Each column specifies the initial states for a single user. You can specify up to 144 columns for EHT MU waveforms. If you specify a single column, the function uses the same initial states for all users. Each row represents the initial state of each packet to generate. A matrix with multiple rows enables you to use a different initial state per packet, where the first row contains the initial state of the first packet. If the number of packets to generate exceeds the number of rows of the matrix provided, the function loops the rows internally.

  - N_P is the number of packets.
  - N_{Users} is the number of users.

For DMG transmissions, specifying this argument overrides the value of the `ScramblerInitialization` property of the `wlanDMGConfig` configuration object.

Example: `[3 56 120]`

Dependencies

This argument is not valid for WUR and DSSS non-HT formats.

Data Types: `double` | `int8`

`WindowTransitionTime` — Duration of window transition

nonnegative scalar

Duration, in seconds, of the window transition applied to each OFDM symbol, specified as a nonnegative scalar. The function does not apply windowing if you specify this input as `0`. This table shows the default and maximum values permitted for each format, the type of guard interval, and the channel bandwidth.

Format	Bandwidth	Permitted `WindowTransitionTime` (seconds)					
		Default Value	Maximum Value	Maximum Permitted Value Based on Guard Interval Duration			
				3.2 µs	1.6 µs	0.8 µs (Long)	0.4 µs (Short)
EHT MU	20, 40, 80, 160, or 320 MHz	`1.0e-07`	Not applicable	`6.4e-06`	`3.2e-06`	`1.6e-06`	Not applicable
HE SU , HE MU, and HE TB	20, 40, 80, or 160 MHz	`1.0e-07`	Not applicable	`6.4e-06`	`3.2e-06`	`1.6e-06`	Not applicable
VHT	20, 40, 80, or 160 MHz	`1.0e-07`	Not applicable	Not applicable	Not applicable	`1.6e-06`	`8.0e-07`

Format	Bandwidth	Permitted `WindowTransitionTime` (seconds)					
		Default Value	Maximum Value	Maximum Permitted Value Based on Guard Interval Duration			
				3.2 µs	1.6 µs	0.8 µs (Long)	0.4 µs (Short)
HT-mixed	20 or 40 MHz	`1.0e-07`	Not applicable	Not applicable	Not applicable	`1.6e-06`	`8.0e-07`
non-HT	20, 40, 80, or 160 MHz	`1.0e-07`	Not applicable	Not applicable	Not applicable	`1.6e-06`	Not applicable
	10 MHz	`1.0e-07`	Not applicable	Not applicable	Not applicable	`3.2e-06`	Not applicable
	5 MHz	`1.0e-07`	Not applicable	Not applicable	Not applicable	`6.4e-06`	Not applicable
WUR	20, 40, 80 MHz	`1.0e-07`	Not applicable	Not applicable	Not applicable	Not applicable	Not applicable
DMG	2640 MHz	`6.0606e-09` (= `16/2640e6`)	`9.6969e-08` (= `256/2640e6`)	Not applicable	Not applicable	Not applicable	Not applicable
S1G	1, 2, 4, 8, or 16 MHz	`1.0e-07`	Not applicable	Not applicable	Not applicable	`1.6e-05`	`8.0e-06`

Data Types: `double`

Output Arguments

`waveform` — Packetized waveform

matrix

Packetized waveform, returned as an N_S-by-N_T matrix. N_S is the number of time-domain samples, and N_T is the number of transmit antennas. `waveform` contains one or more packets of the same PPDU format. Each packet can contain different information bits. Enable waveform packet windowing by setting the `WindowTransitionTime` input to a positive value. Windowing is enabled by default.

For more information, see “Waveform Sampling Rate” on page 3-558, “OFDM Symbol Windowing” on page 3-559, and “Waveform Looping” on page 3-561.

Data Types: `double`
Complex Number Support: Yes

More About

IEEE 802.11 PPDU Format

Supported IEEE 802.11 PPDU formats defined for transmission include EHT, HE, WUR, VHT, HT, non-HT, S1G, and DMG. For all formats, the PPDU field structure includes preamble and data portions.

For a detailed description of the packet structures for the various formats supported, see “WLAN PPDU Structure”.

Waveform Sampling Rate

At the output of this function, the generated waveform has a sampling rate equal to the channel bandwidth.

For all EHT, HE, VHT, HT, and non-HT format OFDM modulation, the channel bandwidth is configured via the `ChannelBandwidth` property of the format configuration object.

For the DMG format modulation schemes, the channel bandwidth is always 2640 MHz and the channel spacing is always 2160 MHz, as specified in sections 20.3.4 and E.1 of [1], respectively.

For the non-HT format DSSS modulation scheme, the chipping rate is always 11 MHz, as specified in section 16.1.1 of [1].

This table indicates the waveform sampling rates associated with standard channel spacing for each configuration format prior to filtering.

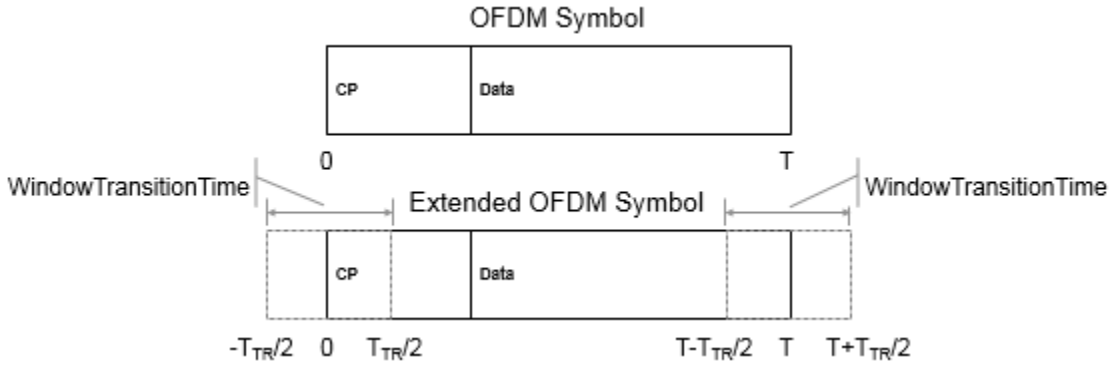
Configuration Object	Modulation Type	`ChannelBandwidth` Property Value	Channel Spacing (MHz)	Sampling Rate (MHz) (F_S, F_C)
`wlanEHTMUConfig`	OFDMA	`'CBW20'`	20	$F_S = 20$
		`'CBW40'`	40	$F_S = 40$
		`'CBW80'`	80	$F_S = 80$
		`'CBW160'`	160	$F_S = 160$
		`'CBW320'`	320	$F_S = 320$
`wlanHEMUConfig`, `wlanHESUConfig`, and `wlanHETBConfig`	OFDMA	`'CBW20'`	20	$F_S = 20$
		`'CBW40'`	40	$F_S = 40$
		`'CBW80'`	80	$F_S = 80$
		`'CBW160'`	160	$F_S = 160$
`wlanVHTConfig`	OFDM	`'CBW20'`	20	$F_S = 20$
		`'CBW40'`	40	$F_S = 40$
		`'CBW80'`	80	$F_S = 80$
		`'CBW160'`	160	$F_S = 160$
`wlanHTConfig`	OFDM	`'CBW20'`	20	$F_S = 20$
		`'CBW40'`	40	$F_S = 40$
`wlanNonHTConfig`	DSSS/CCK	Not applicable	11	$F_C = 11$
	OFDM	`'CBW5'`	5	$F_S = 5$
		`'CBW10'`	10	$F_S = 10$
		`'CBW20'`	20	$F_S = 20$
		`'CBW40'`	40	$F_S = 40$
		`'CBW80'`	80	$F_S = 80$

Configuration Object	Modulation Type	ChannelBandwidth Property Value	Channel Spacing (MHz)	Sampling Rate (MHz) (F_S, F_C)
		'CBW160'	160	$F_S = 160$
wlanWURConfig	OFDM	'CBW20'	20	$F_S = 20$
		'CBW40'	40	$F_S = 40$
		'CBW80'	80	$F_S = 80$
wlanDMGConfig	Control PHY	For DMG, the channel bandwidth is fixed at 2640 MHz.	2160	$F_C = ⅔ F_S = 1760$
	SC			
	OFDM			$F_S = 2640$
wlanS1GConfig	OFDM	'CBW1'	1	$F_S = 1$
		'CBW2'	2	$F_S = 2$
		'CBW4'	4	$F_S = 4$
		'CBW8'	8	$F_S = 8$
		'CBW16'	16	$F_S = 16$
F_S is the OFDM sampling rate. F_C is the chip rate for single-carrier, control PHY, DSSS, and CCK modulations.				

OFDM Symbol Windowing

OFDM naturally lends itself to processing with Fourier transforms. A negative side effect of using an IFFT to process OFDM symbols is the resulting symbol-edge discontinuities. These discontinuities cause out-of-band emissions in the transition region between consecutive OFDM symbols. To smooth the discontinuity between symbols and reduce the intersymbol out-of-band emissions, you can use the `wlanWaveformGenerator` function to apply OFDM symbol windowing. To apply windowing, set the `WindowTransitionTime` input to a positive value.

When windowing is applied, the function adds transition regions to the leading and trailing edge of the OFDM symbol. Windowing extends the length of the OFDM symbol by `WindowTransitionTime` (T_{TR}).

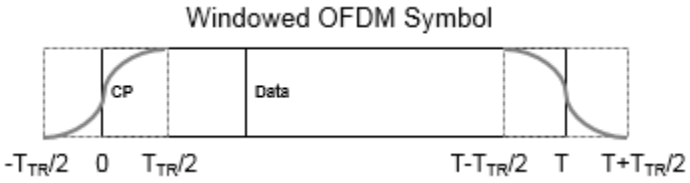


The extended waveform is windowed by pointwise multiplication in the time domain, using this windowing function specified in section 17.3.2.5 of [1]:

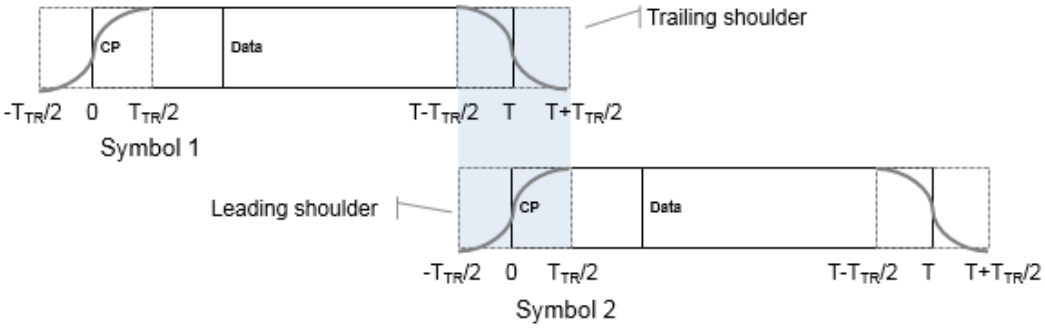
$$w_T(t) = \begin{cases} \sin^2\left[\frac{\pi}{2}\left(\frac{1}{2} + \frac{t}{T_{\mathrm{TR}}}\right)\right] & \text{if } t \in \left[-\frac{T_{\mathrm{TR}}}{2}, \frac{T_{\mathrm{TR}}}{2}\right], \\ 1 & \text{if } t \in \left[\frac{T_{\mathrm{TR}}}{2}, T - \frac{T_{\mathrm{TR}}}{2}\right], \\ \sin^2\left[\frac{\pi}{2}\left(\frac{1}{2} + \frac{t}{T_{\mathrm{TR}}}\right)\right] & \text{if } t \in \left[T - \frac{T_{\mathrm{TR}}}{2}, T + \frac{T_{\mathrm{TR}}}{2}\right]. \end{cases}$$

The windowing function applies over the leading and trailing portion of the OFDM symbol:

- $-T_{\mathrm{TR}}/2$ to $T_{\mathrm{TR}}/2$
- $-T - T_{\mathrm{TR}}/2$ to $T + T_{\mathrm{TR}}/2$



After windowing is applied to each symbol, pointwise addition is used to combine the overlapped regions between consecutive OFDM symbols. Specifically, the trailing shoulder samples at the end of OFDM symbol 1 ($T - T_{\mathrm{TR}}/2$ to $T + T_{\mathrm{TR}}/2$) are added to the leading shoulder samples at the beginning of OFDM symbol 2 ($-T_{\mathrm{TR}}/2$ to $T_{\mathrm{TR}}/2$).



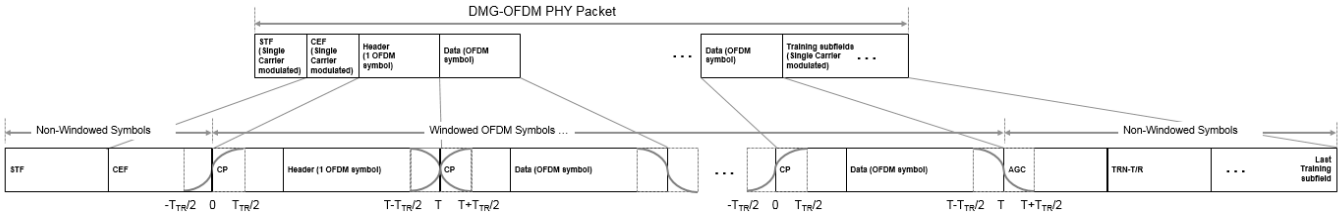
Smoothing the overlap between consecutive OFDM symbols in this manner reduces the out-of-band emissions. The function applies OFDM symbol windowing between:

- Each OFDM symbol within a packet
- Consecutive packets within the waveform, considering the idle time `IdleTime` between packets specified by the `'IdleTime'` input
- The last and the first packet of the generated waveform

Windowing DMG Format Packets

For the DMG format, windowing applies only to packets transmitted using the OFDM PHY and is applied only to the OFDM modulated symbols. For OFDM PHY, only the header and data symbols are OFDM modulated. The preamble (STF and CEF) and the training fields are single carrier modulated and are not windowed. Similar to the out-of-band emissions experienced by consecutive OFDM

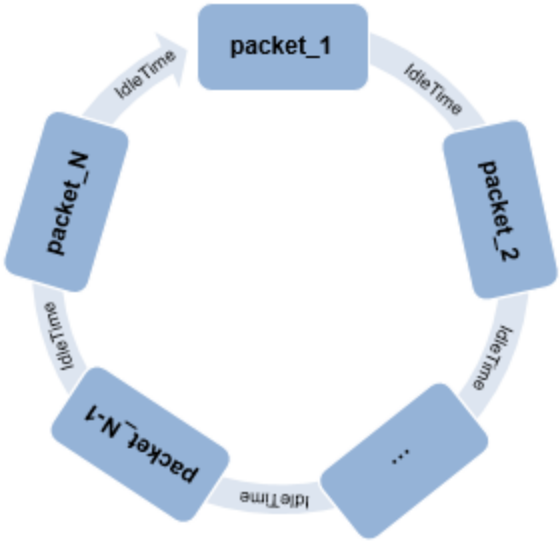
symbols, as shown here, the CEF and the first training subfield are subject to a nominal amount of out-of-band emissions from the adjacent windowed OFDM symbol.



For more information on how the function handles windowing for the consecutive packet idle time and for the last waveform packet, see “Waveform Looping” on page 3-561.

Waveform Looping

To produce a continuous input stream, you can have your code loop on a waveform from the last packet back to the first packet.



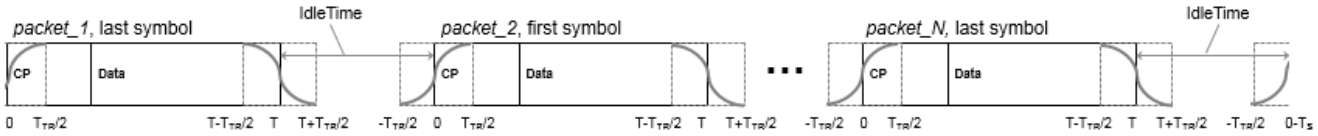
Applying windowing to the last and first OFDM symbols of the generated waveform smooths the transition between the last and first packet of the waveform. When the `'WindowTransitionTime'` input is positive, the `wlanWaveformGenerator` function applies OFDM symbol windowing.

When looping a waveform, the last symbol of *packet_N* is followed by the first OFDM symbol of *packet_1*. If the waveform has only one packet, the waveform loops from the last OFDM symbol of the packet to the first OFDM symbol of the same packet.

When windowing is applied to the last OFDM symbol of a packet and the first OFDM of the next packet, the idle time between the packets factors into the windowing applied. Specify the idle time by using the `'IdleTime'` input to the `wlanWaveformGenerator` function.

- If `'IdleTime'` is `0`, the function applies windowing as it would be for consecutive OFDM symbols within a packet.

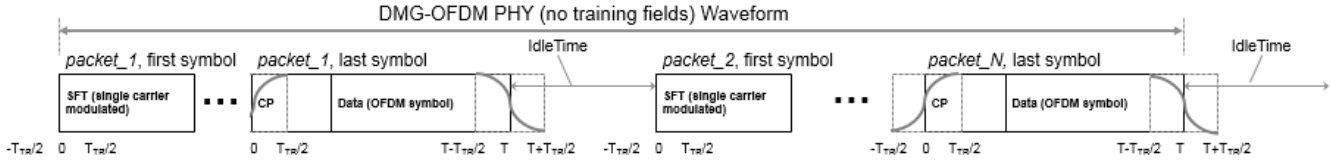
- Otherwise, the extended windowed portion of the first OFDM symbol in *packet_1* (from $-T_{TR}/2$ to $0-T_S$), is included at the end of the waveform. This extended windowed portion is applied for looping when computing the windowing between the last OFDM symbol of *packet_N* and the first OFDM symbol of *packet_1*. T_S is the sample time.



Looping DMG Waveforms

DMG waveforms have these three looping scenarios.

- The looping behavior for a waveform composed of DMG OFDM-PHY packets with no training subfields is similar to the general case outlined in “Waveform Looping” on page 3-561, but the first symbol of the waveform (and each packet) is not windowed.



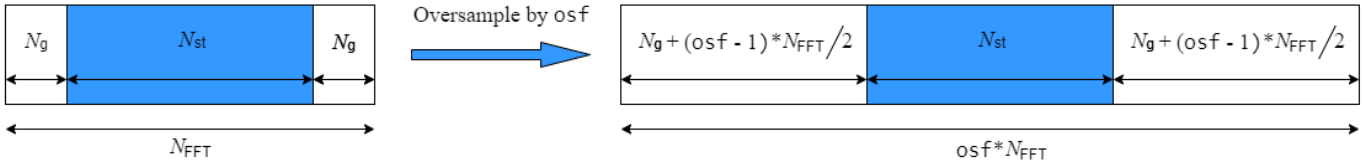
  - If `'IdleTime'` is `0` for the waveform, the windowed portion (from T to $T + T_{TR}/2$) of the last data symbol is added to the start of the STF field.
  - Otherwise, the idle time is appended at the end of the windowed portion (after $T + T_{TR}/2$) of the last OFDM symbol.
- When a waveform composed of DMG OFDM PHY packets includes training subfields, no windowing is applied to the single-carrier modulated symbols the end of the waveform. The last sample of the last training subfield is followed by the first STF sample of the first packet in the waveform.
  - If `'IdleTime'` is `0` for the waveform, there is no overlap.
  - Otherwise, the value of `'IdleTime'` specifies the delay between the last sample of *packet_N* and the first sample of in *packet_1*.
- When a waveform is composed of DMG-SC or DMG-Control PHY packets, the end of the waveform is single carrier modulated, so no windowing is applied to the last waveform symbol. The last sample of the last training subfield is followed by the first STF sample of the first packet in the waveform.
  - If `'IdleTime'` is `0` for the waveform, there is no overlap.
  - Otherwise, the value of `'IdleTime'` specifies the delay between the last sample of *packet_N* and the first sample of in *packet_1*.

Note The same looping behavior applies for a waveform composed of DMG OFDM-PHY packets with training subfields, DMG-SC PHY packets, or DMG-Control PHY packets.

FFT-Based Oversampling

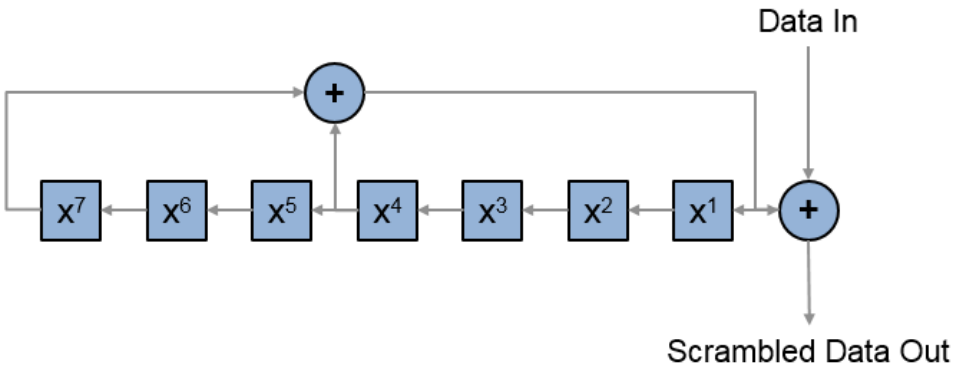
An oversampled signal is a signal sampled at a frequency that is higher than the Nyquist rate. WLAN signals maximize occupied bandwidth by using small guardbands, which can pose problems for anti-imaging and anti-aliasing filters. Oversampling increases guardband width relative to the total signal bandwidth, thereby increasing the number of samples in the signal.

This function performs oversampling by using a larger IFFT and zero pad when generating an OFDM waveform. This diagram shows the oversampling process for an OFDM waveform with N_{FFT} subcarriers comprising N_g guardband subcarriers on either side of N_{st} occupied bandwidth subcarriers.



Scrambler Initialization

The scrambler initialization used on the transmission data follows the process described in IEEE Std 802.11-2012, Section 18.3.5.5 and IEEE Std 802.11ad-2012, Section 21.3.9. The header and data fields that follow the scrambler initialization field (including data padding bits) are scrambled by XORing each bit with a length-127 periodic sequence generated by the polynomial $S(x) = x^7+x^4+1$. The octets of the PSDU are placed into a bit stream and, within each octet, bit 0 (LSB) is first and bit 7 (MSB) is last. This figure shows the generation of the sequence and the XOR operation.



Conversion from integer to bits uses left-MSB orientation. For example, initializing the scrambler with decimal 1, the bits map to these elements.

Element	X^7	X^6	X^5	X^4	X^3	X^2	X^1
Bit Value	0	0	0	0	0	0	1

To generate the bit stream equivalent to a decimal, use the `int2bit` function. For example, for decimal `1`:

```
int2bit(1,7)'
ans =

     0     0     0     0     0     0     1
```

Version History

Introduced in R2015b

EHT MU waveform generation

You can specify the input `cfg` as an object of type `wlanEHTMUConfig`.

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

[2] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`wlanEHTMUConfig` | `wlanHESUConfig` | `wlanHEMUConfig` | `wlanHETBConfig` | `wlanWURConfig` | `wlanDMGConfig` | `wlanVHTConfig` | `wlanHTConfig` | `wlanNonHTConfig` | `wlanS1GConfig`

Functions

`wlanFieldIndices` | `getActiveSubchannelIndex` | `getPSDULength` | `psduLength` | `ruInfo` | `packetFormat` | `phyType`

Apps

WLAN Waveform Generator

Topics
“Packet Size and Duration Dependencies”
“WLAN PPDU Structure”
“HE MU Transmission”

write

Write protocol packet data to PCAP or PCAPNG file

Syntax

```
write(pcapObj,packet,timestamp)
write(pcapngObj,packet,timestamp,interfaceID)
write(obj,packet,timestamp)
write( ___ ,Name,Value)
```

Description

`write(pcapObj,packet,timestamp)` writes the protocol packet data to the PCAP file specified in the PCAP file writer object, `pcapObj`. Input `packet` specifies the protocol packet and input `timestamp` specifies the packet arrival time.

`write(pcapngObj,packet,timestamp,interfaceID)` writes protocol packet data to a PCAPNG file specified in the PCAPNG file writer object, `pcapngObj`. Input `packet`, `timestamp`, and `interfaceID` specifies the protocol packet, packet arrival time, and interface identifier, respectively.

`write(obj,packet,timestamp)` writes the WLAN MAC packet data to the PCAP or PCAPNG file specified in the WLAN PCAP file writer object, `obj`. Input `packet` specifies the WLAN MAC packet and input `timestamp` specifies the packet arrival time.

`write( ___ ,Name,Value)` specifies options using one or more name-value pair arguments in addition to the input argument combinations from any of the previous syntaxes. For example, `'PacketFormat','bits'` sets the format of the protocol packets to bits.

Examples

Write WLAN Packet Data to PCAP File

Create a default PCAP file writer object. Specify the link type for WLAN packet.

```
pcapObj = pcapWriter;
wlanLinkType = 105;
```

Write a global header to the PCAP file.

```
writeGlobalHeader(pcapObj,wlanLinkType);
```

Generate a WLAN packet in bits.

```
macConfig = wlanMACFrameConfig;
[mpdu,frameLength] = wlanMACFrame(macConfig,'OutputFormat','bits');
```

Write the WLAN packet to the PCAP file, specifying the packet format as bits.

```
timestamp = 124800; % Number of microseconds
write(pcapObj,mpdu,timestamp,'PacketFormat','bits');
```

Write WLAN Packet Data to PCAPNG File

Create a PCAPNG file writer object, specifying the name of the PCAPNG file.

```
pcapngObj = pcapngWriter('FileName','sample');
```

Write the interface block for WLAN.

```
interfaceName = 'WLAN';
wlanLinkType = 105;
interfaceId = writeInterfaceDescriptionBlock(pcapngObj,wlanLinkType, ...
    interfaceName);
```

Generate a WLAN packet in bits.

```
macConfig = wlanMACFrameConfig;
[mpdu,frameLength] = wlanMACFrame(macConfig,'OutputFormat','bits');
```

Write the WLAN packet to the PCAPNG file, specifying the packet comment and packet format.

```
timestamp = 0; % Number of microseconds
packetComment = 'This is the first packet';
write(pcapngObj,mpdu,timestamp,interfaceId,'PacketComment', ...
    packetComment,'PacketFormat','bits');
```

Write WLAN MAC Packet Data to PCAP File Using WLAN PCAP Writer

Create a default WLAN PCAP file writer object.

```
obj = wlanPCAPWriter;
```

Generate a WLAN MAC packet of type QoS Data. Specify the frame format as HT-Mixed.

```
macConfig = wlanMACFrameConfig('FrameType','QoS Data','FrameFormat','HT-Mixed');
payload ='00576000103afffe80';
mpdu = wlanMACFrame(payload,macConfig);
```

Write the WLAN MAC packet data to the PCAP file.

```
timestamp = 0; % Number of microseconds
write(obj,mpdu,timestamp);
```

Input Arguments

Note The `pcapWriter`, `pcapngWriter`, and `wlanPCAPWriter` objects do not overwrite the existing PCAP or PCAPNG files. During each call of these objects, specify a unique PCAP or PCAPNG file name.

`pcapObj` — PCAP file writer object

`pcapWriter` object

PCAP file writer object, specified as a `pcapWriter` object.

`packet` — Protocol packet
binary-valued vector | character vector | string scalar | numeric vector | *n*-by-2 character array

Protocol packet, specified as one of these values.

- Binary-valued vector – This value represents bits.
- Character vector – This value represents octets in hexadecimal format.
- String scalar – This value represents octets in hexadecimal format.
- Numeric vector with each element in the range [0, 255] – This value represents octets in decimal format.
- *n*-by-2 character array – In this value, each row represents an octet in hexadecimal format .

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

`timestamp` — Packet arrival time
nonnegative integer

Packet arrival time in POSIX® microseconds elapsed since 1/1/1970, specified as a nonnegative integer.

Data Types: `double`

`pcapngObj` — PCAPNG file writer object
`pcapngWriter` object

PCAPNG file writer object, specified as a `pcapngWriter` object.

`interfaceID` — Unique identifier for an interface
nonnegative scalar

Unique identifier for an interface, specified as a nonnegative scalar.

Data Types: `double`

`obj` — WLAN PCAP file writer object
`wlanPCAPWriter` object

WLAN PCAP file writer object, specified as a `wlanPCAPWriter` object.

Name-Value Pair Arguments

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

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

Example: `'PacketFormat','bits'` specifies the format of the protocol packet to bits.

`PacketFormat` — Format of the protocol packet
`'octets'` (default) | `'bits'`

Format of the protocol packet, specified as the comma-separated pair consisting of `PacketFormat` and `'octets'` or `'bits'`. If this value is specified as `'octets'`, `packet` is specified as one of these values.

- Binary-valued vector – This value represents bits.
- Character vector – This value represents octets in hexadecimal format.
- String scalar – This value represents octets in hexadecimal format.
- Numeric vector with each element in the range [0, 255] – This value represents octets in decimal format.
- *n*-by-2 character array – In this value, each row represents an octet in hexadecimal format .

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

`PacketComment` — Comment for protocol packet
`''` (default) | character vector | string scalar

Comment for the protocol packet, specified as the comma-separated pair consisting of `PacketComment` and a character vector or a string scalar.

Dependencies

To enable this name-value pair argument, specify the `pcapngObj` input argument.

Data Types: `char` | `string`

`Radiotap` — WLAN packet metadata
binary-valued vector | character vector | string scalar | numeric vector | *n*-by-2 character array

WLAN packet metadata , specified as the comma-separated pair consisting of `Radiotap` and one of these values.

- Binary-valued vector – This value represents bits.
- Character vector – This value represents octets in hexadecimal format.
- String scalar – This value represents octets in hexadecimal format.
- Numeric vector with each element in the range [0, 255] – This value represents octets in decimal format.
- *n*-by-2 character array – In this value, each row represents an octet in hexadecimal format .

Dependencies

To enable this name-value pair argument, specify the `obj` input argument.

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

`RadiotapFormat` — Format of radiotap
`'octets'` (default) | `'bits'`

Format of the radiotap, specified as the comma-separated pair consisting of `RadiotapFormat` and `'octets'` or `'bits'`. If this value is specified as `'octets'`, `Radiotap` is specified as one of these values.

- Binary-valued vector – This value represents bits.
- Character vector – This value represents octets in hexadecimal format.

- String scalar – This value represents octets in hexadecimal format.
- Numeric vector with each element in the range [0, 255] – This value represents octets in decimal format.
- *n*-by-2 character array – In this value, each row represents an octet in hexadecimal format .

Dependencies

To enable this name-value pair argument, specify the `obj` input argument.

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

Version History

Introduced in R2020b

References

[1] Tuexen, M. “PCAP Next Generation (Pcapng) Capture File Format.” 2020. https://www.ietf.org/.

[2] “Radiotap - Introduction.” Accessed May 20, 2020. https://www.radiotap.org/.

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

[4] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`writeCustomBlock` | `writeInterfaceDescriptionBlock` | `writeGlobalHeader`

Objects

`pcapWriter` | `pcapngWriter` | `wlanPCAPWriter`

psduLength

EHT PSDU length

Syntax

```
length = psduLength(cfg)
```

Description

`length = psduLength(cfg)` calculates the PSDU length for the extremely high throughput (EHT) configuration `cfg`.

Examples

Get PSDU Length for EHT Transmission

Parameterize an extremely high-throughput multi-user (EHT MU) transmission by creating an OFDMA `wlanEHTMUConfig` object. Set the allocation index to 5. This setting specifies seven RUs with one user each. Two of the RUs are 52-tone and the remaining five are 26-tone. The channel bandwidth is 20 MHz.

```
cfgEHTMU = wlanEHTMUConfig(5);
```

Get and display the PSDU length for the configured transmission.

```
length = psduLength(cfgEHTMU)
```

```
length = 1×7

   100   100   202   100   100   100   202
```

Input Arguments

`cfg` — EHT transmission parameters

`wlanEHTMUConfig` object

EHT transmission parameters, specified as a `wlanEHTMUConfig` object.

Output Arguments

`length` — PSDU length

positive integer | vector

PSDU length, in bytes, returned as a positive integer or row vector of size 1-by-*N*, where *N* is the number of users in the transmission. For an EHT transmission, *N* is an integer in the range [1, 144].

For more information about *N* in EHT transmissions, see the `ruInfo` function. The `NumUsers` field of the `info` output corresponds to *N*.

Version History

Introduced in R2022b

See Also

Objects
`wlanEHTMUConfig`

Functions
`getPSDULength` | `packetFormat` | `ruInfo` | `showAllocation` | `wlanWaveformGenerator`

writeCustomBlock

Write custom block to PCAPNG file

Syntax

```
writeCustomBlock(pcapngObj,customData)
```

Description

`writeCustomBlock(pcapngObj,customData)` writes the custom block data, `customData`, to a PCAPNG file specified in the PCAPNG file writer object.

Examples

Write WLAN User-Defined Custom Block to PCAPNG File

Create a default PCAPNG file writer object.

```
pcapngObj = pcapngWriter('FileName','writeWLANcustom');
```

Write the interface block for WLAN.

```
interfaceName = 'WLAN';
wlanLinkType = 105;
interfaceId = writeInterfaceDescriptionBlock(pcapngObj,wlanLinkType, ...
    interfaceName);
```

Write the custom block to specify user-defined data.

```
writeCustomBlock(pcapngObj,"This block writes user-defined data");
```

Input Arguments

Note The `pcapngWriter` object does not overwrite the existing PCAPNG file. During each call of this object, specify a unique PCAPNG file name.

`pcapngObj` — PCAPNG file writer object

`pcapngWriter` object

PCAPNG file writer object, specified as a `pcapngWriter` object.

`customData` — User-defined data

character vector | string scalar

User-defined data, specified as a character vector or a string scalar.

Data Types: `char` | `string`

Version History

Introduced in R2020b

References

[1] Tuexen, M. “PCAP Next Generation (Pcapng) Capture File Format.” 2020. https://www.ietf.org/.

[2] “Radiotap - Introduction.” Accessed May 20, 2020. https://www.radiotap.org/.

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

[4] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`write` | `writeInterfaceDescriptionBlock`

Objects

`pcapWriter` | `pcapngWriter`

writeGlobalHeader

Write global header to PCAP file

Syntax

```
writeGlobalHeader(pcapObj,linkType)
```

Description

`writeGlobalHeader(pcapObj,linkType)` writes a global header to the PCAP file specified in the PCAP file writer object, `pcapObj`. Input `linkType` specifies the unique identifier for a protocol.

Examples

Write WLAN Packet Global Header to PCAP File

Create a default PCAP file writer object. Specify the link type for WLAN packet.

```
pcapObj = pcapWriter;
wlanLinkType = 105;
```

Using the PCAP file writer object and the WLAN link type, write a global header to the PCAP file.

```
writeGlobalHeader(pcapObj,wlanLinkType);
```

Input Arguments

Note The `pcapWriter` object does not overwrite the existing PCAP file. During each call of this object, specify a unique PCAP file name.

`pcapObj` — PCAP file writer object

`pcapWriter` object

PCAP file writer object, specified as a `pcapWriter` object.

`linkType` — Unique identifier for a protocol

integer in the range [0, 65,535].

Unique identifier for a protocol, specified as an integer in the range [0, 65535].

Data Types: `double`

Version History

Introduced in R2020b

References

[1] “Radiotap - Introduction.” Accessed May 20, 2020. https://www.radiotap.org/.

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

[3] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`write`

Objects

`pcapWriter` | `pcapngWriter`

writeInterfaceDescriptionBlock

Write interface description block to PCAPNG file

Syntax

```
interfaceID = writeInterfaceDescriptionBlock(pcapngObj,linkType,interface)
```

Description

`interfaceID = writeInterfaceDescriptionBlock(pcapngObj,linkType,interface)` writes an interface description block to the PCAPNG file specified in the PCAPNG file writer object, `pcapngObj`. Input `linkType` specifies the unique identifier for the protocol and input `interface` specifies the interface on which the protocol packets are captured. This object function returns the unique identifier for the interface.

Examples

Write WLAN Interface Description Block to PCAPNG File

Create a default PCAPNG file writer object.

```
pcapngObj = pcapngWriter('FileName','writeWLANinterface');
```

Write the interface description block for WLAN.

```
interfaceName = 'WLAN';
wlanLinkType = 105;
interfaceId = writeInterfaceDescriptionBlock(pcapngObj,wlanLinkType, ...
    interfaceName);
```

Input Arguments

Note The `pcapngWriter` object does not overwrite the existing PCAPNG file. During each call of this object, specify a unique PCAPNG file name.

`pcapngObj` — PCAPNG file writer object

`pcapngWriter` object

PCAPNG file writer object, specified as a `pcapngWriter` object.

`linkType` — Unique identifier for protocol

integer in the range [0, 65,535].

Unique identifier for a protocol, specified as an integer in the range [0, 65,535].

Data Types: `double`

`interface` — Name of the interface on which protocol packets are captured
character vector | string scalar

Name of the interface on which protocol packets are captured, specified as a character vector or a string scalar in 8-bit unicode transformation format (UTF-8) format.

Data Types: `char` | `string`

Output Arguments

`interfaceID` — Unique identifier for an interface
nonnegative scalar

Unique identifier for an interface, specified as a nonnegative scalar.

Data Types: `double`

Version History

Introduced in R2020b

References

[1] Tuexen, M. “PCAP Next Generation (Pcapng) Capture File Format.” 2020. https://www.ietf.org/.

[2] “Radiotap - Introduction.” Accessed May 20, 2020. https://www.radiotap.org/.

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

[4] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org.

Extended Capabilities

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

See Also

Functions
`write` | `writeCustomBlock`

Objects
`pcapWriter` | `pcapngWriter`

Classes

networkTrafficFTP

FTP application traffic pattern generator

Description

The `networkTrafficFTP` object specifies the configuration parameters to generate a file transfer protocol (FTP) application traffic pattern based on the 3GPP TR 36.814 specification and the IEEE 802.11ax Evaluation Methodology.

You can use the FTP application traffic pattern in WLAN and 5G (requires 5G Toolbox™) system-level simulations to accurately model the real-world data traffic.

Creation

Syntax

```
cfgFTP = networkTrafficFTP
cfgFTP = networkTrafficFTP(Name,Value)
```

Description

`cfgFTP = networkTrafficFTP` creates a default FTP application traffic pattern object.

`cfgFTP = networkTrafficFTP(Name,Value)` sets properties on page 4-2 using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'GeneratePacket',true` generates an FTP packet.

Properties

`FixedFileSize` — Custom size of file to be transmitted

`[ ]` (default) | positive scalar

Custom size of the file to be transmitted, specified as a positive scalar. This property must be expressed in megabytes. If you do not specify this property, the object uses the truncated Lognormal distribution to calculate the file size.

Data Types: `double`

`LogNormalMu` — Truncated Lognormal distribution mu value

`14.45` (default) | positive scalar

Truncated Lognormal distribution mu value, specified as a positive scalar. The object uses this property to calculate the file size.

Dependencies

To enable this property, set the `FixedFileSize` property to `[ ]`.

Data Types: `double`

`LogNormalSigma` — Truncated Lognormal distribution sigma value

`0.35` (default) | positive scalar

Truncated Lognormal distribution sigma value, specified as a positive scalar. The object uses this value to calculate the file size.

Dependencies

To enable this property, set the `FixedFileSize` property to `[ ]`.

Data Types: `double`

`UpperLimit` — Truncated Lognormal distribution upper limit

`5` (default) | positive scalar

Truncated Lognormal distribution upper limit, specified as a positive scalar. The object uses this value to calculate the file size. The generated file size value must be less than or equal to the `UpperLimit` value. If the generated file size value is greater than `UpperLimit`, the object discards the file size and creates a new one.

Dependencies

To enable this property, set the `FixedFileSize` property to `[ ]`.

Data Types: `double`

`ReadingTime` — Time interval between two successive file transfers

[ ] (default) | positive scalar

Time interval between two consecutive file transfers, specified as a positive scalar. This property must be expressed in milliseconds. To specify a customized value for the reading time, specify this property. If you do not specify this property, the object uses the exponential distribution to calculate the reading time.

Data Types: `double`

`ExponentialMean` — Exponential distribution mean value

`180000` (default) | nonnegative scalar

Exponential distribution mean value, specified as a nonnegative scalar. This property must be expressed in milliseconds. The object uses this property to calculate the reading time.

Dependencies

To enable this property, set the `ReadingTime` property to `[ ]`.

Data Types: `double`

`PoissonMean` — Poisson distribution mean value

[ ] (default) | nonnegative scalar

Poisson distribution mean value, specified as a nonnegative scalar. This property must be expressed in milliseconds. The object uses this property to calculate the packet interarrival time.

Data Types: `double`

`PacketInterArrivalTime` — Time interval between two successively generated packets

`0` (default) | nonnegative scalar

Time interval between two successively generated packets, specified as a nonnegative scalar. This property must be expressed in milliseconds.

Dependencies

To enable this property, set the `PoissonMean` property to `[ ]`.

Data Types: `double`

`GeneratePacket` — Flag to indicate whether to generate FTP packet

`false` or `0` (default) | `true` or `1`

Flag to indicate whether the object generates an FTP packet, specified as a `logical` `1` (`true`) or `0` (`false`).

Data Types: `logical`

`ApplicationData` — Application data to be added in FTP packet

1500-by-1 vector of 1s (default) | column vector of integers in the range [0, 255]

Application data to be added in the FTP packet, specified as a column vector of integers in the range [0, 255]. If the size of the application data is greater than the packet size, the object truncates the application data. If the size of the application data is less than the packet size, the object appends zeros.

Dependencies

To enable this property, set the `GeneratePacket` property to `true` or `1`.

Data Types: `double`

Object Functions

Specific to This Object

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

Examples

Generate FTP Application Traffic Pattern

Create a default FTP application traffic pattern object.

```
cfgFTP = networkTrafficFTP;
```

Generate an FTP application traffic pattern.

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

Generate FTP Application Traffic Pattern Using Reading Time

Create an FTP application traffic pattern object, specifying the reading time.

```
cfgFTP = networkTrafficFTP('ReadingTime',5);
```

Generate an FTP application traffic pattern.

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

Generate FTP Application Traffic Pattern and Data Packet

Create an FTP application traffic pattern object to generate an FTP data packet.

```
cfgFTP = networkTrafficFTP('GeneratePacket',true);
```

Generate an FTP application traffic pattern and data packet.

```
[dt,packetSize,packet] = generate(cfgFTP);
```

Generate FTP Application Traffic Pattern to Visualize Packet Sizes and Intervals

Create a default FTP application traffic pattern object.

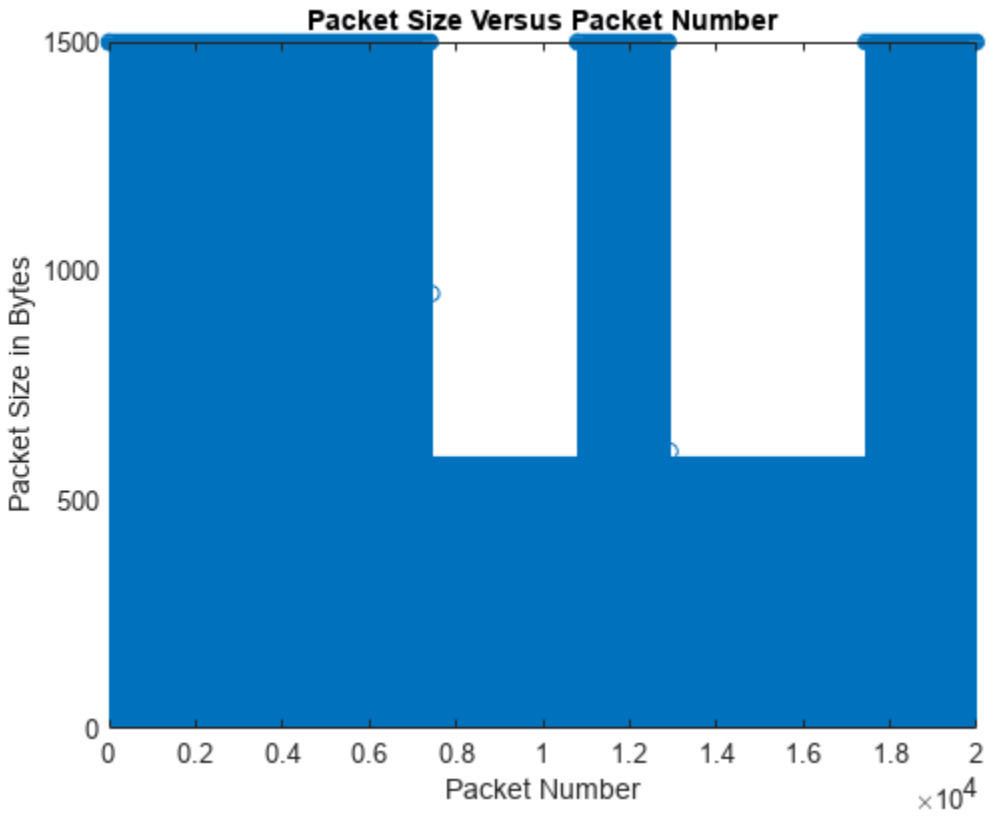
```
cfgFTP = networkTrafficFTP;
```

Generate an FTP application traffic pattern with 20,000 FTP packets.

```
for packetCount = 1:20000
    [dt(packetCount),packetSize(packetCount)] = generate(cfgFTP);
end
```

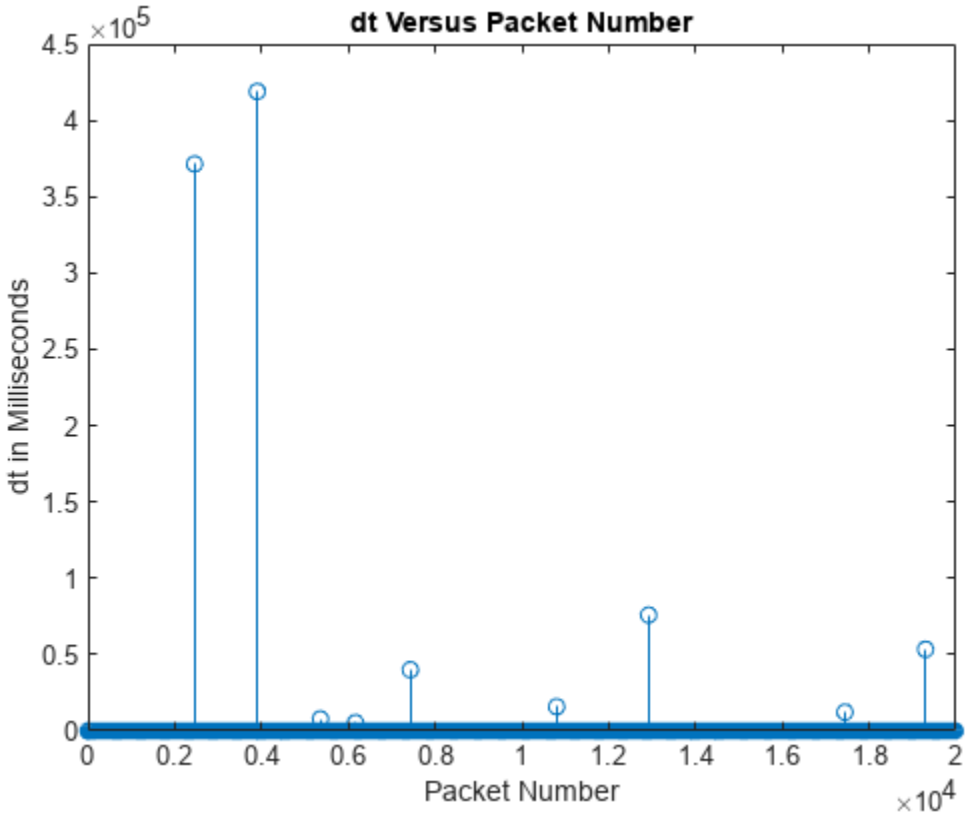
Visualize the FTP packet sizes.

```
stem(packetSize);
title('Packet Size Versus Packet Number');
xlabel('Packet Number');
ylabel('Packet Size in Bytes');
```



Visualize the FTP packet intervals.

```
figure;
stem(dt);
title('dt Versus Packet Number');
xlabel('Packet Number');
ylabel('dt in Milliseconds');
```



Version History

Introduced in R2020b

References

[1] 3GPP TR 36.814. "Evolved Universal Terrestrial Radio Access (E-UTRA). Further advancements for E-UTRA physical layer aspects". Release 15. *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. https://www.3gpp.org.

[2] IEEE 802.11-14/0571r12. "11ax Evaluation Methodology." IEEE P802.11. Wireless LANs. https://www.ieee.org.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`networkTrafficOnOff` | `networkTrafficVoIP` | `networkTrafficVideoConference`

networkTrafficOnOff

On-Off application traffic pattern generator

Description

The `networkTrafficOnOff` object specifies the configuration parameters to generate an On-Off application traffic pattern.

You can use the On-Off application traffic pattern in WLAN, Bluetooth (requires Bluetooth® Toolbox), and 5G (requires 5G Toolbox) system-level simulations to accurately model the real-world data traffic.

Creation

Syntax

```
cfgOnOff = networkTrafficOnOff
cfgOnOff = networkTrafficOnOff(Name,Value)
```

Description

`cfgOnOff = networkTrafficOnOff` creates a default On-Off application traffic pattern object.

`cfgOnOff = networkTrafficOnOff(Name,Value)` sets properties on page 4-8 using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'GeneratePacket',true` generates an application packet.

Properties

`OnTime` — On state duration
[ ] (default) | positive scalar

On state duration, specified as a positive scalar. Units are in seconds. To specify a customized value for the On time, specify this property. If you do not specify this property, the object uses the exponential distribution to calculate the On time.

Data Types: `double`

`OffTime` — Off state duration
[ ] (default) | nonnegative scalar

Off state duration, specified as a nonnegative scalar. Units are in seconds. To specify a customized value for the Off time, specify this property. If you do not specify this property, the object uses the exponential distribution to calculate the Off time.

Data Types: `double`

`OnExponentialMean` — Exponential distribution mean value to calculate On state duration
1 (default) | positive scalar

Exponential distribution mean value to calculate the On state duration, specified as a positive scalar. Units are in seconds.

Dependencies

To enable this property, set the `OnTime` property to `[ ]`.

Data Types: `double`

`OffExponentialMean` — Exponential distribution mean value to calculate Off state duration

`2` (default) | nonnegative scalar

Exponential distribution mean value to calculate the Off state duration, specified as a nonnegative scalar. Units are in seconds.

Dependencies

To enable this property, set the `OffTime` property to `[ ]`.

Data Types: `double`

`DataRate` — Packet generation rate during On state

`5` (default) | positive scalar

Packet generation rate during the On state, specified as a positive scalar. Units are in Kbps. If the value of this property is low and the `PacketSize` is large, the object might not generate packets in the On state.

Data Types: `double`

`PacketSize` — Length of packet to be generated

`1500` (default) | positive scalar

Length of the packet to be generated, specified as a positive scalar. Units are in bytes. If the value of this property is greater than the `DataRate` property value, the object accumulates the data across multiple On times to generate a packet.

Data Types: `double`

`GeneratePacket` — Flag to indicate whether to generate application packet

`false` or `0` (default) | `true` or `1`

Flag to indicate whether the object generates an application packet, specified as a `logical` `1` (`true`) or `0` (`false`).

Data Types: `logical`

`ApplicationData` — Application data to be added in packet

1500-by-1 vector of 1s (default) | column vector of integers in the range [0, 255]

Application data to be added in the packet, specified as a column vector of integers in the range [0, 255]. If the size of the application data is greater than the `PacketSize` property value, the object truncates the application data. If the size of the application data is less than the `PacketSize` property value, the object appends zeros.

Dependencies

To enable this property, set the `GeneratePacket` property to `1` (`true`).

Data Types: `double`

Object Functions

Specific to This Object

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

Examples

Generate On-Off Application Traffic Pattern

Create a default On-Off application traffic pattern object.

```
cfgOnOff = networkTrafficOnOff;
```

Generate an On-Off application traffic pattern.

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

Generate On-Off Application Traffic Pattern Using On State Mean Value

Create an On-Off application traffic pattern object, specifying the exponentially distributed mean value of the On state.

```
cfgOnOff = networkTrafficOnOff('OnExponentialMean',5);
```

Generate an On-Off application traffic pattern.

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

Generate On-Off Application Traffic Pattern and Data Packet

Create an On-Off application traffic pattern object to generate an On-Off data packet.

```
cfgOnOff = networkTrafficOnOff('GeneratePacket',true);
```

Generate an On-Off application traffic pattern and data packet.

```
[dt,packetSize,packet] = generate(cfgOnOff);
```

Generate On-Off Application Traffic Pattern to Visualize Packet Sizes and Intervals

Create a default On-Off application traffic pattern object.

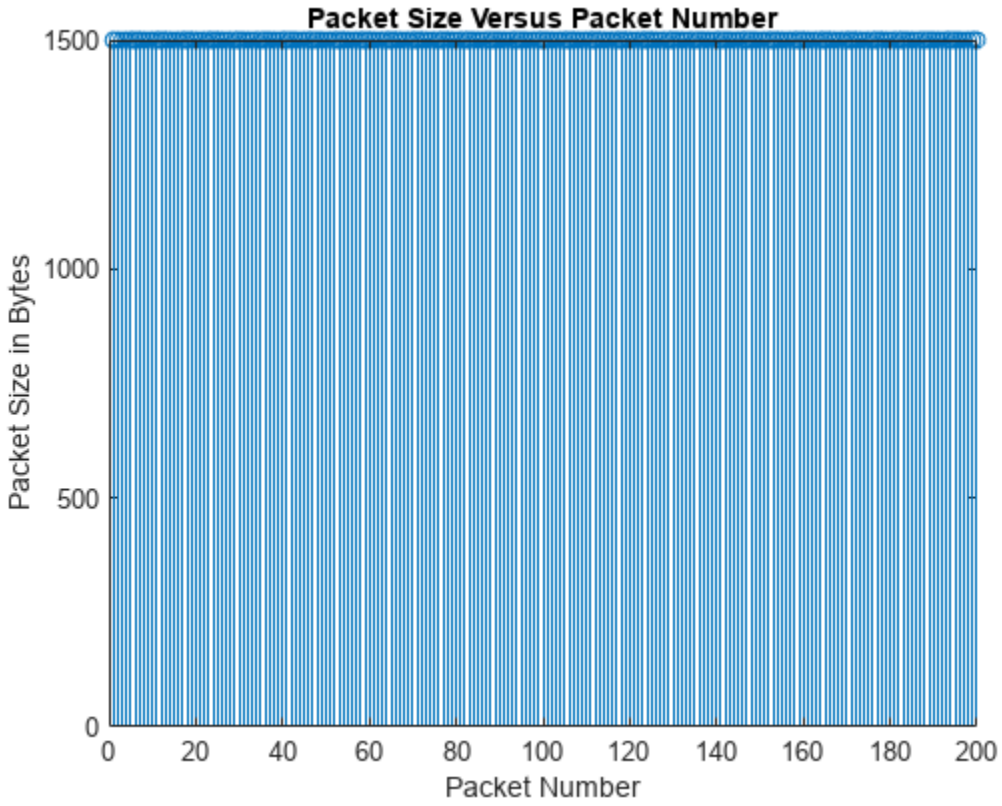
```
cfgOnOff = networkTrafficOnOff;
```

Generate an On-Off application traffic pattern with 200 On-Off packets.

```
for packetCount = 1:200
    [dt(packetCount),packetSize(packetCount)] = generate(cfgOnOff);
end
```

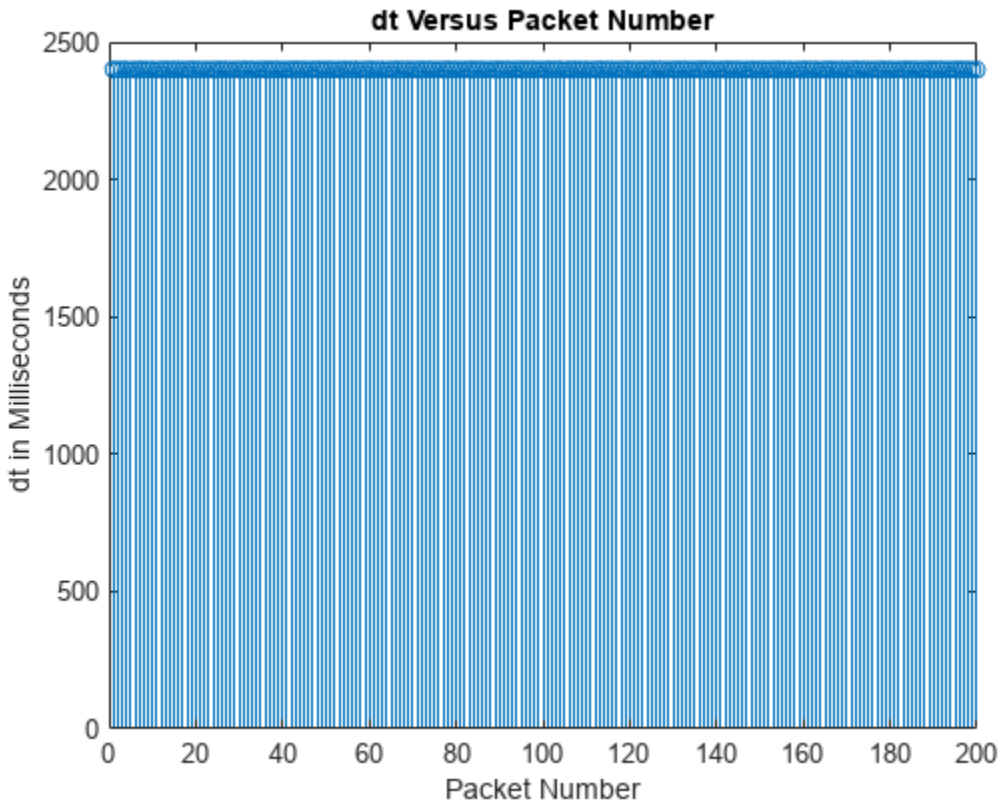
Visualize the On-Off packet sizes.

```
stem(packetSize);
title('Packet Size Versus Packet Number');
xlabel('Packet Number');
ylabel('Packet Size in Bytes');
```



Visualize the On-Off packet intervals.

```
figure;
stem(dt);
title('dt Versus Packet Number');
xlabel('Packet Number');
ylabel('dt in Milliseconds');
```



Version History

Introduced in R2020b

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`networkTrafficFTP` | `networkTrafficVoIP` | `networkTrafficVideoConference`

networkTrafficVideoConference

Video conference application traffic pattern generator

Description

The `networkTrafficVideoConference` object specifies the configuration parameters to generate a video conference application traffic pattern based on the IEEE 802.11ax Evaluation Methodology.

You can use the video conference application traffic pattern in WLAN and 5G (requires 5G Toolbox) system-level simulations to accurately model real-world data traffic.

Creation

Syntax

```
cfgVideo = networkTrafficVideoConference
cfgVideo = networkTrafficVideoConference(Name,Value)
```

Description

`cfgVideo = networkTrafficVideoConference` creates a default video application traffic pattern object.

`cfgVideo = networkTrafficVideoConference(Name,Value)` sets “Properties” on page 4-13 using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'HasJitter',false` specifies that the video application pattern does not model jitter.

Properties

`FrameInterval` — Time interval between two consecutive video frames

`40` (default) | positive integer

Time interval between two consecutive video frames, specified as a positive integer. This value is expressed in milliseconds.

Data Types: `double`

`FrameSizeMethod` — Option to set source for video frame size

`'WeibullDistribution'` (default) | `'FixedSize'`

Option to set source for video frame size, specified as one of these values.

- `'WeibullDistribution'` — Use the video frame size value calculated by the Weibull probability distribution function.
- `'FixedSize'` — Use the video frame size value specified by the FixedFrameSize property.

Data Types: `char` | `string`

`FixedFrameSize` — Size of video frame

`5000` (default) | integer in the range [1, 15,000]

Size of the video frame, specified as an integer in the range [1, 15,000]. This value is expressed in bytes. The video frame can be segmented into multiple network packets based on this value.

Dependencies

To enable this property, set the FrameSizeMethod property to `'FixedSize'`.

Data Types: `double`

`WeibullScale` — Scale parameter for Weibull distribution

`6950` (default) | scalar in the range (0, 54,210]

Scale parameter for the Weibull distribution to calculate the video frame size, specified as a scalar in the range (0, 54,210].

Dependencies

To enable this property, set the `FrameSizeMethod` property to `'WeibullDistribution'`.

Data Types: `double`

`WeibullShape` — Shape parameter for Weibull distribution

`0.8099` (default) | scalar in the range (0, 1]

Shape parameter for the Weibull distribution to calculate the video frame size, specified as a scalar in the range (0, 1].

Dependencies

To enable this property, set the `FrameSizeMethod` property to `'WeibullDistribution'`.

Data Types: `double`

`HasJitter` — Flag to indicate whether to model network jitter

`true` or `1` (default) | `false` or `0`

Flag to indicate whether to model network jitter, specified as a `logical` `1` (`true`) or `0` (`false`). The object applies jitter between the segmented packets. If you set this property to `1` (`true`), the object models jitter using the Gamma probability distribution function.

Data Types: `logical`

`GammaShape` — Shape parameter for Gamma distribution

`0.2463` (default) | scalar in the range (0, 5]

Shape parameter for the Gamma distribution to calculate network jitter, specified as a scalar in the range (0, 5].

Dependencies

To enable this property, set the HasJitter property to `1` (`true`).

Data Types: `double`

`GammaScale` — Scale parameter for Gamma distribution

`6.227` (default) | scalar in the range (0, 10]

Scale parameter for the Gamma distribution to calculate network jitter, specified as a scalar in the range (0, 10].

Dependencies

To enable this property, set the `HasJitter` property to `1` (`true`).

Data Types: `double`

`ProtocolOverhead` — Protocol overheads in network traffic
`28` (default) | integer in the range [0, 60]

Protocol overheads in network traffic, specified as an integer in the range [0, 60]. To add layer 3, layer 4, and application protocol overheads in network traffic, enable this property. This value is expressed in bytes.

Data Types: `double`

`GeneratePacket` — Flag to indicate whether to generate video packet with payload
`false` or `0` (default) | `true` or `1`

Flag to indicate whether to generate a video packet with payload, specified as a `logical` `1` (`true`) or `0` (`false`). To generate a video packet with payload, set this property to `1` (`true`). If you set this property to `0` (`false`), the `generate` object function generates no application data packet.

Data Types: `logical`

`ApplicationData` — Application data to be added in video packet
1500-by-1 vector of 1s (default) | column vector of integers in the range [0, 255]

Application data to be added in the video packet, specified as a column vector of integers in the range [0, 255].

- If the size of the application data is greater than the packet size, the object truncates the application data.
- If the size of the application data is less than the packet size, the object appends zeros.

Dependencies

To enable this property, set the GeneratePacket property to `1` (`true`).

Data Types: `double`

Object Functions

Specific to This Object

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

Examples

Generate Video Conference Traffic Using Default Values

Create a default video application traffic pattern object.

```
cfgVideo = networkTrafficVideoConference
```

```
cfgVideo =
  networkTrafficVideoConference with properties:

       FrameInterval: 40
     FrameSizeMethod: 'WeibullDistribution'
        WeibullScale: 6950
        WeibullShape: 0.8099
           HasJitter: 0
    ProtocolOverhead: 28
      GeneratePacket: 0
```

Generate a video application traffic pattern.

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

Generate Video Conference Traffic with Frame Interval and Data Packet

Create a video application traffic pattern object to generate a video data packet. Specify the frame interval in milliseconds.

```
cfgVideo = networkTrafficVideoConference('GeneratePacket',true);
cfgVideo.FrameInterval = 60;
```

Generate a video application traffic pattern and data packet.

```
[dt,packetSize,packet] = generate(cfgVideo);
```

Generate Multiple Video Data Packets Using Elapsed Time

Create a video application traffic pattern object to generate a video frame of size 400 bytes.

```
cfgVideo = networkTrafficVideoConference('FrameSizeMethod','FixedSize','FixedFrameSize',400);
```

Specify the elapsed time in milliseconds.

```
elapsedTime = 10;
```

After every elapsed time value, invoke the video application traffic pattern object to generate five video data packets.

```
for i = 1:5
    while true
        [dt,packetSize] = generate(cfgVideo,elapsedTime);
        if packetSize
            fprintf('Video data packet %d generated\n',i);
            break;
        end
    end
end
```

```
Video data packet 1 generated
Video data packet 2 generated
```

```
Video data packet 3 generated
Video data packet 4 generated
Video data packet 5 generated
```

Generate Video Conference Traffic Pattern to Visualize Packet Sizes and Intervals

Create a default video application traffic pattern object.

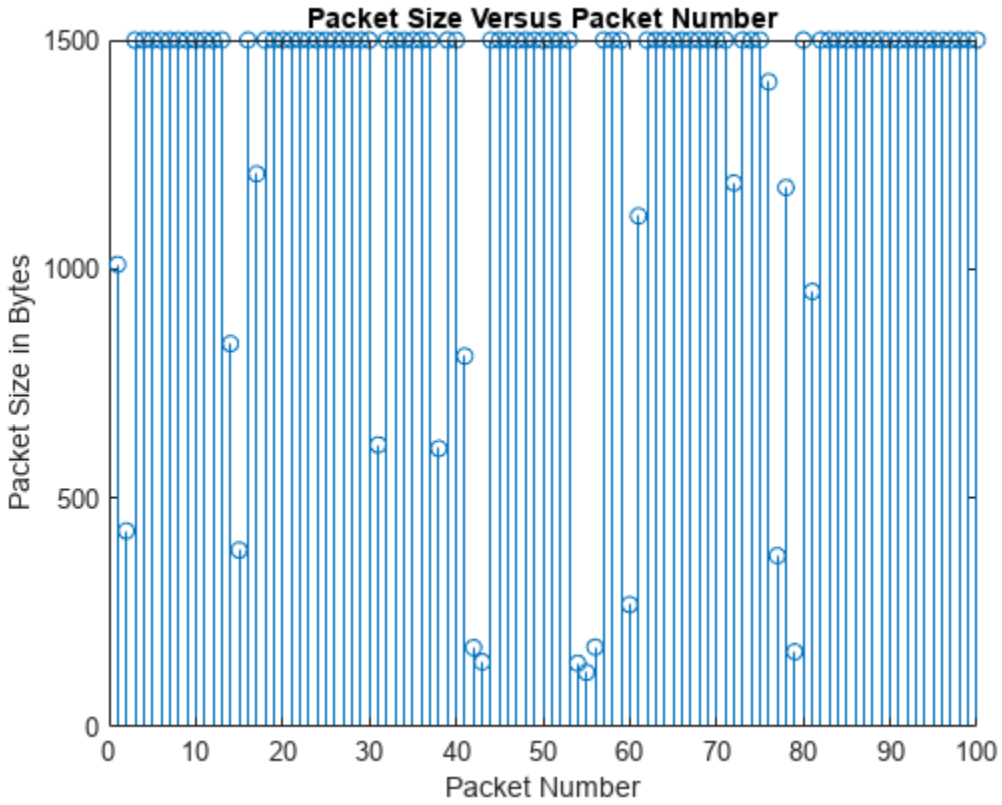
```
cfgVideo = networkTrafficVideoConference;
```

Generate a video application traffic pattern with 100 video packets.

```
for packetCount = 1:100
    [dt(packetCount),packetSize(packetCount)] = generate(cfgVideo);
end
```

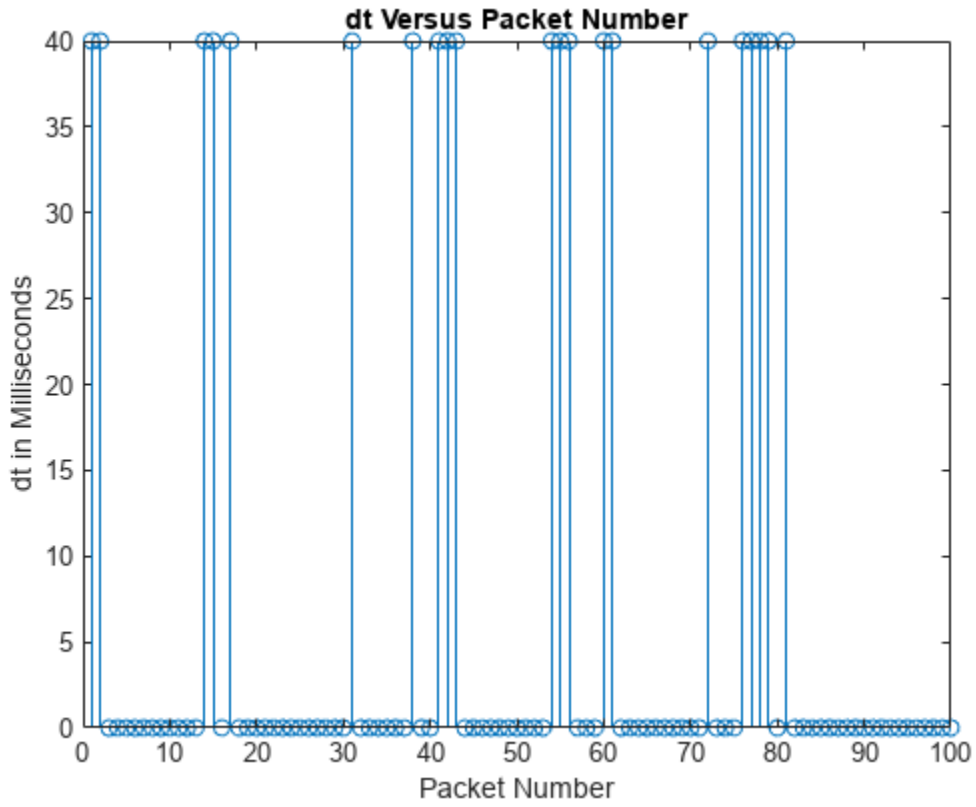
Visualize the video packet sizes.

```
stem(packetSize);
title('Packet Size Versus Packet Number');
xlabel('Packet Number');
ylabel('Packet Size in Bytes');
```



Visualize the video packet intervals.

```
figure;
stem(dt);
title('dt Versus Packet Number');
xlabel('Packet Number');
ylabel('dt in Milliseconds');
```



Version History

Introduced in R2021a

References

[1] IEEE 802.11-14/0571r12. "11ax Evaluation Methodology." IEEE P802.11. Wireless LANs. https://www.ieee.org.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects
`networkTrafficFTP` | `networkTrafficVoIP` | `networkTrafficOnOff`

networkTrafficVoIP

VoIP application traffic pattern generator

Description

The `networkTrafficVoIP` object specifies the configuration parameters to generate a voice over Internet protocol (VoIP) application traffic pattern based on the IEEE 802.11ax Evaluation Methodology.

You can use the VoIP application traffic pattern in WLAN and 5G (requires 5G Toolbox) system-level simulations to accurately model the real-world data traffic.

Creation

Syntax

```
cfgVoIP = networkTrafficVoIP
cfgVoIP = networkTrafficVoIP(Name,Value)
```

Description

`cfgVoIP = networkTrafficVoIP` creates a default VoIP application traffic pattern object.

`cfgVoIP = networkTrafficVoIP(Name,Value)` sets properties on page 4-20 using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'HasJitter',false` specifies that the VoIP application pattern does not model jitter.

Properties

`ExponentialMean` — Mean value for exponential distribution

`1250` (default) | nonnegative integer

Mean value for the exponential distribution, specified as a nonnegative integer. The object uses this property to calculate the exponentially distributed active or silent state duration in the VoIP traffic.

Data Types: `double`

`HasJitter` — Flag to indicate whether to model jitter

`true` or `1` (default) | `false` or `0`

Flag to indicate whether the object models jitter, specified as a `logical` `1` (`true`) or `0` (`false`).

Data Types: `logical`

`LaplaceScale` — Scale parameter for Laplace distribution

`5.11` (default) | scalar in the range [1, 100]

Scale parameter for the Laplace distribution, specified as a scalar in the range [1, 100]. The object uses this property and the `LaplaceMu` property to calculate the packet arrival delay jitter in milliseconds.

Dependencies

To enable this property, set the `HasJitter` property to `1` (`true`).

Data Types: `double`

`LaplaceMu` — Location parameter for Laplace distribution
`0` (default) | scalar in the range [0, 100]

Location parameter for the Laplace distribution, specified as a scalar in the range [0, 100]. The object uses this property and the `LaplaceScale` property to calculate packet arrival delay jitter in milliseconds.

Dependencies

To enable this property, set the `HasJitter` property to `1` (`true`).

Data Types: `double`

`GeneratePacket` — Flag to indicate whether to generate a VoIP packet
`false` or `0` (default) | `true` or `1`

Flag to indicate whether the object generates a VoIP packet, specified as a `logical` `1` (`true`) or `0` (`false`).

Data Types: `logical`

`ApplicationData` — Application data to be added in VoIP packet
36-element column vector of 1s (default) | column vector of integers in the range [0, 255]

Application data to be added in the VoIP packet, specified as a column vector of integers in the range [0, 255]. If the size of the application data is greater than the packet size, the object truncates the application data. If the size of the application data is less than the packet size, the object appends zeros.

Dependencies

To enable this property, set the `GeneratePacket` property to `1` (`true`).

Data Types: `double`

Object Functions

Specific to This Object

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

Examples

Generate VoIP Application Traffic Pattern

Create a default VoIP application traffic pattern object.

```
cfgVoIP = networkTrafficVoIP;
```

Generate a VoIP application traffic pattern.

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

Generate VoIP Application Traffic Pattern Using Mean Value of Exponential Distribution

Create a VoIP application traffic pattern object, specifying the mean value of the exponential distribution.

```
cfgVoIP = networkTrafficVoIP('ExponentialMean',5);
```

Generate a VoIP application traffic pattern.

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

Generate VoIP Application Traffic Pattern and Data Packet

Create a VoIP application traffic pattern object to generate a VoIP data packet.

```
cfgVoIP = networkTrafficVoIP('GeneratePacket',true);
```

Generate a VoIP application traffic pattern and data packet.

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

Generate VoIP Application Traffic Pattern to Visualize Packet Sizes and Intervals

Create a default VoIP application traffic pattern object.

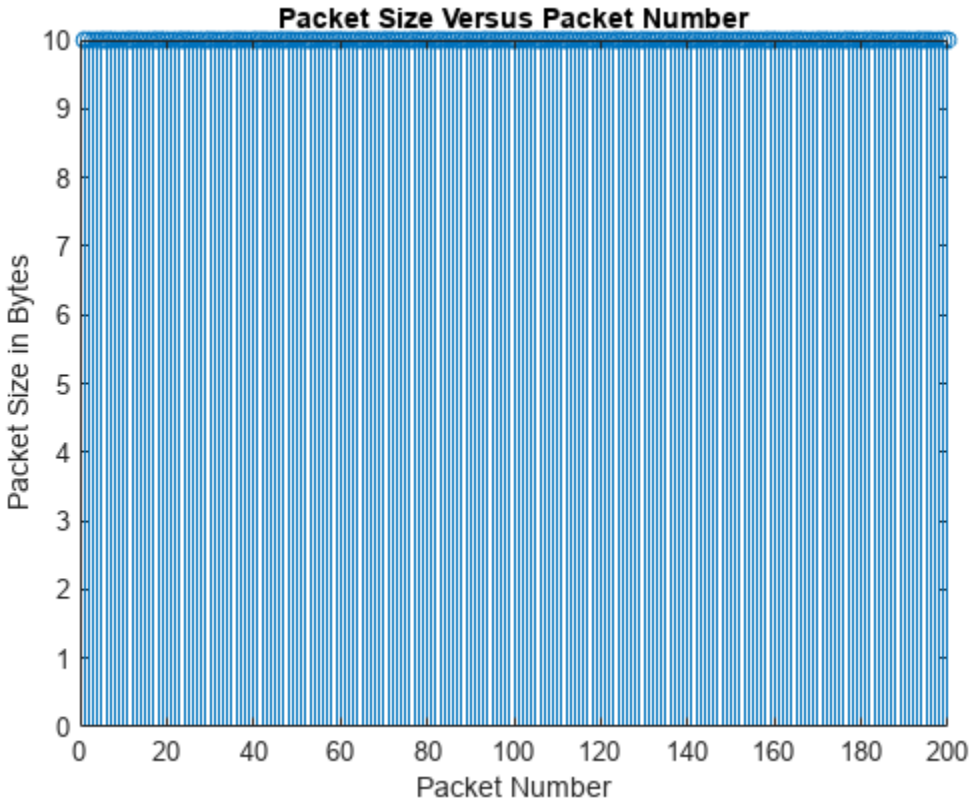
```
cfgVoIP = networkTrafficVoIP;
```

Generate a VoIP application traffic pattern with 200 VoIP packets.

```
for packetCount = 1:200
    [dt(packetCount),packetSize(packetCount)] = generate(cfgVoIP);
end
```

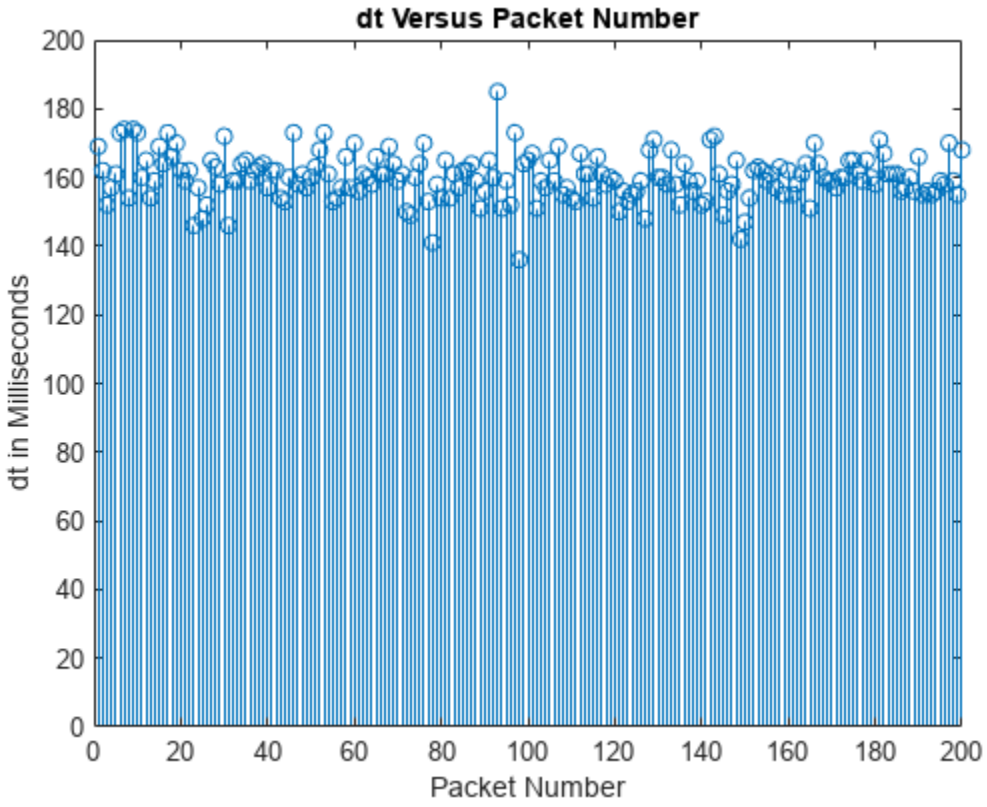
Visualize the VoIP packet sizes.

```
stem(packetSize);
title('Packet Size Versus Packet Number');
xlabel('Packet Number');
ylabel('Packet Size in Bytes');
```



Visualize the VoIP packet intervals.

```
figure;
stem(dt);
title('dt Versus Packet Number');
xlabel('Packet Number');
ylabel('dt in Milliseconds');
```



Version History

Introduced in R2020b

References

[1] IEEE 802.11-14/0571r12. "11ax Evaluation Methodology." IEEE P802.11. Wireless LANs. https://www.ieee.org.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`networkTrafficFTP` | `networkTrafficVideoConference` | `networkTrafficOnOff`

pcapWriter

PCAP file writer of protocol packets

Description

The `pcapWriter` object writes generated and recovered protocol packets to a packet capture (PCAP) file (`.pcap`).

You can write these packet types to a PCAP file:

- Generated and recovered WLAN protocol packets
- Generated and recovered 5G NR protocol packets (requires 5G Toolbox)
- Generated and recovered Bluetooth low energy (LE) link layer (LL) packets (requires Bluetooth Toolbox)

Creation

Syntax

```
pcapObj = pcapWriter
pcapObj = pcapWriter(Name,Value)
```

Description

`pcapObj = pcapWriter` creates a default PCAP file writer object.

`pcapObj = pcapWriter(Name,Value)` sets properties on page 4-25 using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'ByteOrder','big-endian'` specifies the byte order as big-endian.

Properties

Note The `pcapWriter` object does not overwrite the existing PCAP file. During each call of this object, specify a unique PCAP file name.

`FileName` — Name of the PCAP file

`'capture'` (default) | character row vector | string scalar

Name of the PCAP file, specified as a character row vector or a string scalar.

Data Types: `char` | `string`

`ByteOrder` — Byte order

`'little-endian'` (default) | `'big-endian'`

Byte order, specified as `'little-endian'` or `'big-endian'`.

Data Types: `char` | `string`

Object Functions

Specific to This Object

write	Write protocol packet data to PCAP or PCAPNG file
writeGlobalHeader	Write global header to PCAP file

Examples

Write WLAN Packet to PCAP File

Create a PCAP file writer object, specifying the name of the PCAP file. Specify the link type for WLAN packet.

```
pcapObj = pcapWriter('FileName','writeWLANpacketdata2');
wlanLinkType = 105;
```

Write a global header to the PCAP file.

```
writeGlobalHeader(pcapObj,wlanLinkType);
```

Specify a WLAN packet.

```
mpdu = 'B4000000FFFFFFFFFFFF00123456789BA79A5B28';
```

Write the WLAN packet to the PCAP file.

```
timestamp = 124800; % Number of microseconds
write(pcapObj,mpdu,timestamp);
```

Version History

Introduced in R2020b

References

[1] “Radiotap - Introduction.” Accessed May 20, 2020. https://www.radiotap.org/.

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

[3] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects
`pcapngWriter` | `wlanPCAPWriter`

pcapngWriter

PCAPNG file writer of protocol packets

Description

The `pcapngWriter` object writes generated and recovered protocol packets to a packet capture next generation (PCAPNG) file (`.pcapng`).

You can write these packet types to a PCAPNG file:

- Generated and recovered WLAN protocol packets
- Generated and recovered 5G NR protocol packets (requires 5G Toolbox)
- Generated and recovered Bluetooth low energy (LE) link layer (LL) packets (requires Bluetooth Toolbox)

Creation

Syntax

```
pcapngObj = pcapngWriter
pcapngObj = pcapngWriter(Name,Value)
```

Description

`pcapngObj = pcapngWriter` creates a default PCAPNG file writer object.

`pcapngObj = pcapngWriter(Name,Value)` sets properties on page 4-28 using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'ByteOrder','big-endian'` specifies the byte order as big-endian.

Properties

Note The `pcapngWriter` object does not overwrite the existing PCAPNG file. During each call of this object, specify a unique PCAPNG file name.

`FileName` — Name of the PCAPNG file

`'capture'` (default) | character row vector | string scalar

Name of the PCAPNG file, specified as a character row vector or a string scalar.

Data Types: `char` | `string`

`ByteOrder` — Byte order

`'little-endian'` (default) | `'big-endian'`

Byte order, specified as `'little-endian'` or `'big-endian'`.

Data Types: `char` | `string`

`FileComment` — Comment for PCAPNG file
`''` (default) | character vector | string scalar

Comment for the PCAPNG file, specified as a character vector or a string scalar.

Data Types: `char` | `string`

Object Functions

Specific to This Object

write	Write protocol packet data to PCAP or PCAPNG file
writeCustomBlock	Write custom block to PCAPNG file
writeInterfaceDescriptionBlock	Write interface description block to PCAPNG file

Examples

Write WLAN Packet to PCAPNG File

Create a PCAPNG file writer object, specifying the name of the PCAPNG file.

```
pcapngObj = pcapngWriter('FileName','writeWLANpacket');
```

Write the interface block for WLAN.

```
interfaceName = 'WLAN';
wlanLinkType = 105;
interfaceId = writeInterfaceDescriptionBlock(pcapngObj,wlanLinkType, ...
    interfaceName);
```

Specify a WLAN packet.

```
mpdu = 'B4000000FFFFFFFFFFFF00123456789BA79A5B28';
```

Specify the packet comment. Write the WLAN packet to the PCAPNG file, specifying the packet comment.

```
timestamp = 0; % Number of microseconds
packetComment = 'This is the first packet';
write(pcapngObj,mpdu,timestamp,interfaceId,'PacketComment', ...
    packetComment);
```

Write Packets from WLAN and BLE Interface to PCAPNG File

Create a PCAPNG file writer object, specifying the name of the PCAPNG file.

```
pcapngObj = pcapngWriter('FileName','writeWLANBLEpackets');
```

Write the interface block for WLAN.

```
interfaceName = 'WLAN';
wlanLinkType = 105;
```

```
interfaceId = writeInterfaceDescriptionBlock(pcapngObj,wlanLinkType, ...
    interfaceName);
```

Specify a WLAN packet.

```
mpdu = 'B4000000FFFFFFFFFFFF00123456789BA79A5B28';
```

Specify the packet comment. Write the WLAN packet to the PCAPNG file, specifying the packet comment.

```
timestamp = 0; % Number of microseconds
packetComment = 'This is a WLAN packet';
write(pcapngObj,mpdu,timestamp,interfaceId,'PacketComment', ...
    packetComment);
```

Write the interface block for Bluetooth low energy (BLE).

```
interfaceName = 'BLE';
bleLinkType = 251;
interfaceId = writeInterfaceDescriptionBlock(pcapngObj,bleLinkType, ...
    interfaceName);
```

Specify a BLE packet.

```
llpacket = '42BC13E206120E00050014010A001F00400017001700000007D47C0';
```

Specify the packet comment. Write the BLE packet to the PCAPNG file, specifying the packet comment.

```
timestamp = 0; % Number of microseconds
packetComment = 'This is a BLE packet';
write(pcapngObj,llpacket,timestamp,interfaceId,'PacketComment', ...
    packetComment);
```

Version History

Introduced in R2020b

References

[1] Tuexen, M. “PCAP Next Generation (Pcapng) Capture File Format.” 2020. https://www.ietf.org/.

[2] “Radiotap - Introduction.” Accessed May 20, 2020. https://www.radiotap.org/.

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

[4] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects
`pcapWriter` | `wlanPCAPWriter`

wlanDMGConfig

Configure DMG transmission

Description

The `wlanDMGConfig` object is a configuration object for the WLAN directional multi-gigabit (DMG) packet format.

Creation

Syntax

```
cfgDMG = wlanDMGConfig
cfgDMG = wlanDMGConfig(Name,Value)
```

Description

`cfgDMG = wlanDMGConfig` creates `cfgDMG`, a configuration object that initializes parameters for an IEEE 802.11 DMG “PPDU” on page 4-38.

`cfgDMG = wlanDMGConfig(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotation marks. For example, `wlanDMGConfig('MCS','13','TrainingLength',4)` specifies a DMG format with these properties:

- A modulation and coding scheme corresponding to orthogonal frequency division multiplexing (OFDM) physical layer (PHY) modulation and a coding rate of ½
- A PPDU with four training fields

At runtime, the calling function validates object settings for properties relevant to the operation of the function.

Properties

`MCS` — MCS used for transmission

`0` (default) | integer in the interval [0, 24] | `'9.1'` | `'12.1'` | `'12.2'` | `'12.3'` | `'12.4'` | `'12.5'` | `'12.6'`

Modulation and coding scheme (MCS) used for transmission, specified as an integer in the interval [0, 24], or as one of the extended MCS values: `'9.1'`, `'12.1'`, `'12.2'`, `'12.3'`, `'12.4'`,`'12.5'`, or `'12.6'`. Specify an extended (non-integer) MCS value as a character vector or string scalar. Specify an integer MCS value as a character vector, string scalar, or integer. The value of this property indicates the MCS used in transmitting the current packet in accordance with these tables:

- Control PHY:

MCS	Modulation	Coding Rate	Comment
0	Differential binary phase-shift keying (DBPSK)	1/2	The coding rate may be lower due to codeword shortening.

- Single-carrier modulation:

MCS	Modulation	Coding Rate	N_{CBPS}	Repetition
1	π/2-BPSK	1/2	1	2
2		1/2		1
3		5/8		
4		3/4		
5		13/16		
6	π/2 quadrature phase-shift keying (π/2-QPSK)	1/2	2	
7		5/8		
8		3/4		
9		13/16		
9.1		7/8		
10	π/2-16-point quadrature amplitude modulation (π/2-16-QAM)	1/2	4	
11		5/8		
12		3/4		
12.1		13/16		
12.2		7/8		
12.3	π/2-64QAM	5/8	6	
12.4		3/4		
12.5		13/16		
12.6		7/8		
N_{CBPS} is the number of coded bits per symbol.				

- OFDM modulation:

MCS	Modulation	Coding Rate	N_{BPSC}	N_{CBPS}	N_{DBPS}
13	Staggered QPSK (SQPSK)	1/2	1	336	168
14		5/8			210
15	QPSK	1/2	2	672	336
16		5/8			420
17		3/4			504
18	16-QAM	1/2	4	1344	672
19		5/8			840
20		3/4			1008
21		13/16			1092

MCS	Modulation	Coding Rate	N_{BPSC}	N_{CBPS}	N_{DBPS}
22	64-QAM	5/8	6	2016	1260
23		3/4			1512
24		13/16			1638

N_{BPSC} is the number of coded bits per single carrier.

N_{CBPS} is the number of coded bits per symbol.

N_{DBPS} is the number of data bits per symbol.

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

`TrainingLength` — Number of training fields

`0` (default) | multiple of four in the interval [0, 64]

Number of training fields, specified as a multiple of four in the interval [0, 64].

Data Types: `double`

`PacketType` — Packet training field type

`'TRN-R'` (default) | `'TRN-T'`

Packet training field type, specified as one of these values:

- `'TRN-R'` – Indicates that the packet includes or requests receive-training subfields
- `'TRN-T'` – Indicates that the packet includes transmit-training subfields

Dependencies

This property applies only when the value of the `TrainingLength` property is positive.

Data Types: `char` | `string`

`BeamTrackingRequest` — Beam tracking request indicator

`false` or `0` (default) | `true` or `1`

Beam tracking request indicator, specified as a numeric or logical `1` (`true`) or `0` (`false`). To indicate that beam tracking is requested, set this property to `1` (`true`).

Dependencies

This property applies only when the value of the `TrainingLength` property is positive.

Data Types: `logical`

`TonePairingType` — Tone pairing type

`'Static'` (default) | `'Dynamic'`

Tone pairing type, specified as `'Static'` or `'Dynamic'`.

Dependencies

This property applies only when the `MCS` property is in the interval [13, 17], specifying OFDM modulation with either SQPSK or QPSK.

Data Types: `char` | `string`

`DTPGroupPairIndex` — DTP group pair index

`(0:1:41)'` (default) | 42-by-1 vector of integers

Dynamic tone pairing (DTP) group pair index, specified as a 42-by-1 vector of integers in the interval [0, 41] with no duplicated elements. Specify one vector for each pair.

Dependencies

This property applies only when the `MCS` property is in the interval [13, 17] and when the `TonePairingType` property is `'Dynamic'`.

Data Types: `double`

`DTPIndicator` — DTP update indicator

`false` or `0` (default) | `true` or `1`

DTP update indicator, specified as a numeric or logical `1` (`true`) or `0` (`false`). To indicate that the DTP mapping has been updated, set this property to `1` (`true`).

Dependencies

This property applies only when the `MCS` property is in the interval [13, 17] and when the `TonePairingType` property is `'Dynamic'`.

Data Types: `logical`

`PSDULength` — PSDU length

`1000` (default) | integer in the interval [1, 2^{18}]

Physical layer convergence procedure (PLCP) service data unit (PSDU) length, in bytes, specified as an integer in the interval [1, 2^{18}].

Data Types: `double`

`ScramblerInitialization` — Initial scrambler state

2 (default) | integer in the interval [0, 127] | binary column vector

Initial scrambler state of the data scrambler for each packet generated, specified as an integer in the interval [0, 127] or a corresponding binary column vector. For example, you can set the initial scrambler state to 26 by specifying `'ScramblerInitialization',26` or `'ScramblerInitialization',[1;1;0;1;0]`. The value of the `MCS` property determines the values to which you can set this property.

- If you set `MCS` to `0`, specify this property as an integer in the interval [0, 15] or a 4-by-1 binary column vector.
- If you set `MCS` to `'9.1'`, `'12.1'`, `'12.2'`, `'12.3'`, `'12.4'`, `'12.5'`, or `'12.6'`, specify this property as an integer in the interval [0, 31] or a 5-by-1 binary column vector.
- If you set `MCS`, specify this property as an integer in the interval [0, 127] or a 7-by-1 column vector.

The default value of 2 is the example state given in Section L.5.2 of [1].

Data Types: `double` | `int8`

`AggregatedMPDU` — MPDU aggregation indicator

`false` or `0` (default) | `true` or `1`

MPDU aggregation indicator, specified as a numeric or logical `1` (`true`) or `0` (`false`). To indicate that the current packet contains aggregated MPDUs (A-MPDUs), set this property to `1` (`true`).

Dependencies

This property applies only when you set the `MCS` property to a value other than `0`.

Data Types: `logical`

`LastRSSI` — Received power level of the last packet

`0` (default) | integer in the interval [0, 15]

Received power level of the last packet, specified as an integer in the interval [0, 15]. This property corresponds to the *LAST_RSSI* parameter within the *TXVECTOR* or *RXVECTOR* field.

When transmitting a response frame immediately following a short interframe space (SIFS) period, a DMG STA sets the *LAST_RSSI* parameter of *TXVECTOR* as specified in Section 9.3.2.3.3 of [1].

- To represent power values greater than or equal to –42 dBm, set this property to `15` .
- To represent power values *p* between –68 dBm and –42 dBm, set this property to a value *v* in the interval [2, 14]. The values of *p* and *v* are related by *v* = `round`((*p* – (–71 dBm))/2).
- To represent power values less than or equal to –68 dBm, set this property to `1`.
- To indicate that the last packet was not received following a SIFS period, set this property to `0`.

The *LAST_RSSI* parameter of *RXVECTOR* indicates the received power level of the last packet with a valid PHY header according to Table 21-1 of [1].

- To represent power values greater than or equal to –42 dBm, set this property to `15` .
- To represent power values *p* between –68 dBm and –42 dBm, set this property to a value *v* in the interval [2, 14]. The values of *p* and *v* are related by *v* = `round`((*p* – (–71 dBm))/2).
- To represent power values less than or equal to –68 dBm, set this property to `1`.
- To indicate that the previous packet was not received during the SIFS period before the current transmission, set this property to `0`.

Dependencies

This property applies only when you set the `MCS` property to a value other than `0`.

Data Types: `double`

`Turnaround` — Turnaround indication

`false` or `0` (default) | `true` or `1`

Turnaround indication, specified as a numeric or logical `1` (`true`) or `0` (`false`). To indicate that the STA is required to listen for an incoming PPDU immediately following the transmission of the PPDU, set this property to `1` (`true`). For more information, see Section 9.3.2.3.3 of [1].

Data Types: `logical`

Object Functions

phyType	Return DMG PHY modulation type
transmitTime	Packet transmission time

Examples

Create DMG Configuration Object with Default Settings

```
cfgDMG = wlanDMGConfig
```

```
cfgDMG =
  wlanDMGConfig with properties:

                        MCS: '0'
             TrainingLength: 0
                 PSDULength: 1000
    ScramblerInitialization: 2
                 Turnaround: 0
```

Create DMG Configuration Object and Return DMG PHY Type

Create DMG configuration objects and change the default property settings by using dot notation. Use the `phyType` object function to access the DMG PHY modulation type.

Create a DMG configuration object and return the DMG PHY modulation type. By default, the configuration object creates properties to model the DMG control PHY.

```
dmg = wlanDMGConfig;
phyType(dmg)
```

```
ans =
'Control'
```

Model the SC PHY by modifying the defaults by using the dot notation to specify an MCS of 5.

```
dmg.MCS = 5;
phyType(dmg)
```

```
ans =
'SC'
```

Create DMG Configuration Object and Modify Default Settings

Create a DMG configuration object and use `Name,Value` pairs to override default settings.

```
dtpgrouppairs = (randperm(42)-1)';
cfgDMG = wlanDMGConfig('MCS',13,'TonePairingType','Dynamic', ...
    'DTPGroupPairIndex',dtpgrouppairs)
```

```
cfgDMG =
  wlanDMGConfig with properties:

                        MCS: 13
             TrainingLength: 0
           TonePairingType: 'Dynamic'
```

```
          DTPGroupPairIndex: [42x1 double]
               DTPIndicator: 0
                 PSDULength: 1000
    ScramblerInitialization: 2
             AggregatedMPDU: 0
                   LastRSSI: 0
                 Turnaround: 0
```

Create DMG Configuration Object with Extended MCS

Create DMG configuration objects and change the default MCS setting by using dot notation.

Create a DMG configuration object and return the DMG PHY modulation type. By default, the configuration object creates properties to model the DMG control PHY.

```
dmg = wlanDMGConfig;
phyType(dmg)
```

```
ans =
'Control'
```

Model the SC PHY by modifying the defaults by using the dot notation to specify an extended MCS of 9.1.

```
dmg.MCS = '9.1';
phyType(dmg)
```

```
ans =
'SC'
```

More About

PPDU

The physical layer (PHY) protocol data unit (PPDU) is the complete physical layer convergence procedure (PLCP) frame, including PLCP headers, MAC headers, the MAC data field, and the MAC and PLCP trailers.

Version History

Introduced in R2017b

References

[1] IEEE STD 802.11ad-2012 (Amendment to IEEE Std 802.11-2012, as amended by IEEE Std 802.11ae™-2012 and IEEE Std 802.11a™-2012). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 4: Enhancements for Very High Throughput Operation in Bands below 6 GHz.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations: To change the size of the `MCS` property during a simulation, you must set this property before using the object for the first time.

See Also

Objects

`wlanHEMUConfig` | `wlanHESUConfig` | `wlanHETBConfig` | `wlanHTConfig` | `wlanNonHTConfig` | `wlanS1GConfig` | `wlanVHTConfig`

Functions

`phyType` | `transmitTime` | `wlanDMGDataBitRecover` | `wlanDMGHeaderBitRecover` | `wlanDMGOFDMDemodulate` | `wlanDMGOFDMInfo` | `wlanWaveformGenerator`

Apps

WLAN Waveform Generator

Topics

“Packet Size and Duration Dependencies”

wlanEHTMUConfig

Configure EHT MU transmission

Description

The `wlanEHTMUConfig` object is a configuration object for the WLAN extremely high-throughput multi-user (EHT MU) packet format.

Creation

Syntax

```
cfgEHTMU = wlanEHTMUConfig(AllocationIndex)
cfgEHTMU = wlanEHTMUConfig(ChannelBandwidth)
cfgEHTMU = wlanEHTMUConfig(___,Name=Value)
```

Description

`cfgEHTMU = wlanEHTMUConfig(AllocationIndex)` creates `cfgEHTMU`, a configuration object that initializes transmit parameters for an IEEE 802.11 EHT MU PPDU for `AllocationIndex`, the input resource unit allocation. The PPDU type is OFDMA. For a detailed description of the EHT WLAN format, see [1].

`cfgEHTMU = wlanEHTMUConfig(ChannelBandwidth)` creates `cfgEHTMU`, a configuration object that initializes transmit parameters for an IEEE 802.11 EHT MU PPDU for `ChannelBandwidth`, the input channel bandwidth. The PPDU type is non-OFDMA.

`cfgEHTMU = wlanEHTMUConfig(___,Name=Value)` sets properties using one or more name-value pairs. For example, `wlanEHTMUConfig("CBW320",NumUsers=8)` specifies a non-OFDMA transmission with eight users and a bandwidth allocation of 320 MHz.

At runtime, the calling function validates object settings for properties relevant to the operation of the function.

Properties

`AllocationIndex` — Resource unit allocation index

integer | row vector of integers | matrix of integers

Resource unit (RU) allocation index for each 20 MHz subchannel, specified as an integer, row vector of integers, or matrix of integers. This property determines the number of RUs, the size of each RU, the number of users assigned to each RU, and which RUs are punctured. For an example of puncturing in OFDMA transmissions, see “802.11be Waveform Generation”.

The correspondence between allocation indices and resource unit allocations is given in the “EHT MU Transmission” tutorial. This property applies only when the PPDU type is OFDMA.

The format of `AllocationIndex` depends on the desired channel bandwidth.

- Specify `AllocationIndex` as an integer to set a channel bandwidth of 20 MHz. For example, specifying `AllocationIndex` as `0` gives nine RUs, each with one user and of size 26.
- Specify `AllocationIndex` as a 1-by-N vector of integers to set a channel bandwidth of 40, 80, 160, or 320 MHz. N is equal to the number of 20 MHz subchannels. For example, specifying `AllocationIndex` as `[0 0]` gives 18 RUs, each with one user and of size 26.
- Specify `AllocationIndex` as an M-by-N matrix of integers to set a channel bandwidth of 160 or 320 MHz. N is as defined above. M is equal to the number of 80 MHz segments. When you specify `AllocationIndex` as a matrix, all of its rows must be equal.

Note When you specify `AllocationIndex` as a 1-by-N vector in the 160 MHz or 320 MHz case, the value of the property is set to an M-by-N matrix. M is 2 for 160 MHz and 4 for 320 MHz. The new rows are generated by taking extra copies of the vector that you specify. This process is compliant with the standard [1]. The value of M is equal to the number of 80 MHz segments in the channel. An example is given in Annex Z.8 of the standard.

Note You can set this property only when you create the object. After creation, the object is read-only.

Data Types: `double`

`ChannelBandwidth` — Channel bandwidth

`"CBW20"` | `"CBW40"` | `"CBW80"` | `"CBW160"` | `"CBW320"`

Channel bandwidth of PPDU transmission, specified as one of these values:

- `"CBW20"` — Channel bandwidth of 20 MHz
- `"CBW40"` — Channel bandwidth of 40 MHz
- `"CBW80"` — Channel bandwidth of 80 MHz
- `"CBW160"` — Channel bandwidth of 160 MHz
- `"CBW320"` — Channel bandwidth of 320 MHz

Create a non-OFDMA `wlanEHTMUConfig` object by setting the value of this property. When you create an OFDMA `wlanEHTMUConfig` object, this property is configured based on the value to which you set the `AllocationIndex` property.

Note You can set this property only when you create the object. After creation, the object is read-only.

Data Types: `char` | `string`

`NumUsers` — Number of users in non-OFDMA configuration

1 (default) | integer in the range [1, 8]

Number of users in a non-OFDMA configuration, specified as an integer in the range [1, 8].

Note You can set this property only when you create the object. After creation, the object is read-only.

Data Types: `single` | `double`

`PuncturedChannelFieldValue` — Puncturing pattern value for non-OFDMA configuration

`0` (default) | integer in the range [0, 24]

Puncturing pattern value for a non-OFDMA configuration, specified as an integer in the range [0, 24]. The correspondence between puncturing patterns and integers in the range [0, 24] is defined in table 36-30 of [1]. For each channel bandwidth, the default, `0`, specifies that no puncturing takes place.

- For channel bandwidths of 20 and 40 MHz, `PuncturedChannelFieldValue` must be `0` because puncturing is not available at these bandwidths.
- For an 80 MHz channel bandwidth, `PuncturedChannelFieldValue` must be in the range [0, 4].
- For a 160 MHz channel bandwidth, `PuncturedChannelFieldValue` must be in the range [0, 12].

Note You can set this property only when you create the object. After creation, the object is read-only.

Data Types: `single` | `double`

`EHTDUPMode` — Enable EHT DUP mode

`false` or `0` (default) | `true` or `1`

Enable EHT DUP mode by setting this property to `true` or `1`. This property applies only to a non-OFDMA configuration with a channel bandwidth of 80, 160, or 320 MHz.

Note You can set this property only when you create the object. After creation, the object is read-only.

Data Types: `logical`

`PuncturingPattern` — Puncturing pattern in non-OFDMA configuration

logical vector

Puncturing pattern for the given value of the `PuncturedChannelFieldValue` property, specified as a logical vector. The correspondence between puncturing patterns and values of the `PuncturedChannelFieldValue` property is defined in table 36-30 of [1].

If the channel bandwidth is 80 or 160 MHz, the length of the vector is equal to the number of 20 MHz subchannels. If the channel bandwidth is 320 MHz, the length of the vector is equal to the number of 40 MHz subchannels. For each entry of the logical vector, a value of `true` or `1` means that the subchannel that corresponds to the entry is punctured.

Note This property is read-only and is determined by the value of the `PuncturedChannelFieldValue` property.

Data Types: `logical`

`RU` — Transmission properties of each RU and MRU in transmission
cell array of `wlanEHTRU` objects

Transmission properties of each RU and MRU in the transmission, specified as a cell array of `wlanEHTRU` objects. For an OFDMA `wlanEHTMUConfig` object, the `AllocationIndex` property determines the `RU` property. For a non-OFDMA `wlanEHTMUConfig` object, the `ChannelBandwidth` property determines the `RU` property.

Data Types: `cell`

`User` — Transmission properties of each user
cell array of `wlanEHTUser` objects

Transmission properties of each user, specified as a cell array of `wlanEHTUser` objects. For an OFDMA `wlanEHTMUConfig` object, the `AllocationIndex` property determines the `User` property. For a non-OFDMA `wlanEHTMUConfig` object, the `ChannelBandwidth` property determines the `User` property.

Data Types: `cell`

`NumTransmitAntennas` — Number of transmit antennas
`1` (default) | positive integer

Number of transmit antennas, specified as a positive integer.

Data Types: `double`

`PreEHTCyclicShifts` — Cyclic shift values of additional transmit antennas
`-75` (default) | integer in the range [–200, 0] | row vector

Cyclic shift values, in nanoseconds, of additional transmit antennas for the pre-EHT fields of the waveform. The first eight antennas use the cyclic shift values specified in table 21-10 of [2]. The remaining L antennas use the values that you specify in this property, where L = `NumTransmitAntennas` – 8. Specify this property as one of these values:

- An integer in the range [–200, 0] – the `wlanEHTMUConfig` object uses this cyclic shift value for each of the L additional antennas.
- A row vector of length L of integers in the range [–200, 0] – the `wlanEHTMUConfig` object uses the kth element as the cyclic shift value for the (k + 8)th transmit antenna.

> **Note** If you specify this property as a row vector of length greater than L, the `wlanEHTMUConfig` object uses only the first L elements. For example, if you set the `NumTransmitAntennas` property to `16`, the `wlanEHTMUConfig` object uses only the first L = 16 – 8 = 8 elements of this vector.

Dependencies

To enable this property, set the `NumTransmitAntennas` property to a value greater than `8`.

Data Types: `double`

`PreEHTPhaseRotation` — Pre-EHT phase rotation values
row vector

Phase rotation values for the pre-EHT portion of the waveform, specified as a row vector of length 16 with entries equal to 1 or -1. This property applies only when the channel bandwidth is 320 MHz. The

16 entries of this property correspond to 20 MHz subchannels in ascending order of frequency, as defined in equation 36-12 of [1].

Data Types: `double`

`GuardInterval` — Guard range (cyclic prefix) duration
`3.2` (default) | `1.6` | `0.8`

Guard range (cyclic prefix) duration for the data field within a packet, in microseconds, specified as `3.2`, `1.6`, or `0.8`.

Data Types: `double`

`EHTLTFType` — EHT-LTF compression mode
`4` (default) | `2` | `1`

EHT-LTF compression mode, specified as `1`, `2`, or `4`. These values correspond to the 1× EHT-LTF, 2× EHT-LTF, and 4× EHT-LTF compression modes. The EHT-LTF type is enumerated in table 36-18 of [1] as:

- 1× EHT-LTF — Duration of 3.2 μs with a guard range duration of 1.6 μs.
- 2× EHT-LTF — Duration of 6.4 μs with a guard range duration of 0.8 μs or 1.6 μs.
- 4× EHT-LTF — Duration of 12.8 μs with a guard range duration of 0.8 μs or 3.2 μs.

For more information on the EHT-LTF, see section 36.3.12.10 of [1].

Data Types: `double`

`NumExtraEHTLTFSymbols` — Number of extra EHT-LTF symbols
`0` (default) | integer in the range [0, 7]

Amount by which the number of EHT-LTF symbols exceeds the initial number of EHT-LTF symbols, specified as a positive integer. The initial number of EHT-LTF symbols is determined by the total number of spatial streams, per table 36-43 in [1]. The sum of `NumExtraEHTLTFSymbols` and the initial number of EHT-LTF symbols cannot exceed 8.

Data Types: `double`

`EHTSIGMCS` — MCS of the EHT-SIG field
`0` (default) | `1` | `3` | `15`

Modulation and coding scheme (MCS) of the extremely high throughput signal field (EHT-SIG), specified as `0`, `1`, `3`, or `15`. These respectively correspond to the EHT-SIG-MCS values 0, 1, 2, and 3 in table 36-88 of [1].

Data Types: `double`

`UplinkIndication` — Uplink indication
`false` or `0` (default) | `true` or `1`

Uplink indication, specified as a numeric or logical `1` (`true`) or `0` (`false`). To indicate that the PPDU is sent on a downlink transmission, set this property to `0` (`false`). To indicate that the PPDU is sent on an uplink transmission, set this property to `1` (`true`).

Data Types: `logical`

`BSSColor` — Basic service set color identifier

`0` (default) | integer in the range [0, 63]

Basic service set (BSS) color identifier, specified as an integer in the range [0, 63].

Data Types: `double`

`SpatialReuse` — Spatial reuse indication

`0` (default) | integer in the range [0, 15]

Spatial reuse indication, specified as an integer in the range [0, 15].

Data Types: `double`

`TXOPDuration` — Duration information for TXOP protection

`[]` (default) | integer in the range [0, 8448]

Duration information for transmit opportunity (TXOP) protection, specified as an integer in the range [0, 8448]. The `TXOPDuration` property represents a duration in microseconds. In the TXOP subfield of the U-SIG field, these durations are represented by integers in the range [0, 127]. Therefore, a duration in microseconds must be converted according to the procedure set out in table 36-1 of [1]. The default, `[]`, indicates that no information is specified. In this case, the TXOP subfield of the U-SIG field is set to 127.

Data Types: `double`

`Channelization` — Channelization for 320 MHz

`"320MHZ-1"` (default) | `"320MHZ-2"`

Channelization for a 320 MHz channel bandwidth, specified as `"320MHz-1"` or `"320MHz-2"`. A 320 MHz channel has three possible locations for the channel center frequencies. In accordance with section 36.3.23.2 of [1], when you specify `"320MHz-1"`, these locations are numbered 31, 95, and 159. When you specify `"320MHz-2"`, the locations are numbers 63, 127 and 191.

Data Types: `char` | `string`

Object Functions

numPostFECPaddingBits	Required number of post-FEC padding bits
packetFormat	WLAN packet format
psduLength	EHT PSDU length
ruInfo	Resource unit allocation information
transmitTime	Packet transmission time
showAllocation	Resource unit (RU) allocation

Examples

Create Multi-User EHT OFDMA Configuration Objects

Create a multi-user EHT configuration object with the allocation index set to 48. This setting specifies an OFDMA configuration with one 106+26 tone MRU and one 106 tone RU in a 20 MHz channel. Both resource units have one user.

```
allocationIndex = 48;
cfgSMRU = wlanEHTMUConfig(allocationIndex);
```

Display the properties of the MRU.

```
cfgSMRU.RU{1}
```

```
ans =
  wlanEHTRU with properties:

    PowerBoostFactor: 1
      SpatialMapping: 'Direct'

   Read-only properties:
                Size: [106 26]
               Index: [1 5]
         UserNumbers: 1
```

Now create a multi-user EHT configuration object with the allocation index set to the vector [151 30 30 30 64 64 29 29]. This setting specifies a 996+484-tone MRU and two 242-tone RUs in a 160 MHz channel. The MRU has eight users and the two RUs have one user each.

```
allocationIndex = [151 30 30 30 64 64 29 29];
cfgLMRU = wlanEHTMUConfig(allocationIndex);
```

For multi-user OFDMA transmissions with a channel bandwidth of 160 MHz or higher, the EHT-SIG content channels can carry different information per 80 MHz segment. For these bandwidths, the read-only property `AllocationIndex` is a matrix of size M-by-N, where M is the number of 80 MHz segments and N is the number of 20 MHz subchannels. In this example M is 2 and N is 8. Each row of the matrix represents an 80 MHz segment.

Display the `AllocationIndex` property.

```
disp(cfgLMRU.AllocationIndex);
```

```
   151    30    30    30    28    28    29    29
    30    30    30    30    64    64    29    29
```

Display the properties of the MRU.

```
cfgLMRU.RU{1}
```

```
ans =
  wlanEHTRU with properties:

    PowerBoostFactor: 1
      SpatialMapping: 'Direct'

   Read-only properties:
                Size: [996 484]
               Index: [1 4]
         UserNumbers: [1 2 3 4 5 6 7 8]
```

Now create a multi-user EHT configuration object with the allocation index set to the 2-by-8 matrix obtained by duplicating the previous allocation index. In this case, the EHT-SIG content channels carry the same information per 80 MHz segment.

```
allocationIndex = [allocationIndex;allocationIndex];
cfg2by8 = wlanEHTMUConfig(allocationIndex);
```

Display the `AllocationIndex` property.

```
disp(cfg2by8.AllocationIndex);
```

```
   151    30    30    30    64    64    29    29
   151    30    30    30    64    64    29    29
```

Verify that the two configuration objects have the same resource unit allocations.

```
isequal(ruInfo(cfgLMRU),ruInfo(cfg2by8))
```

```
ans = logical
   1
```

Create EHT Non-OFDMA Configuration Object with Puncturing

Create a non-OFDMA EHT MU configuration object. Set the channel bandwidth to 320 MHz and the number of users to 2. Specify a punctured channel field value of 20.

```
cfg = wlanEHTMUConfig("CBW320",NumUsers=2,PuncturedChannelFieldValue=20);
```

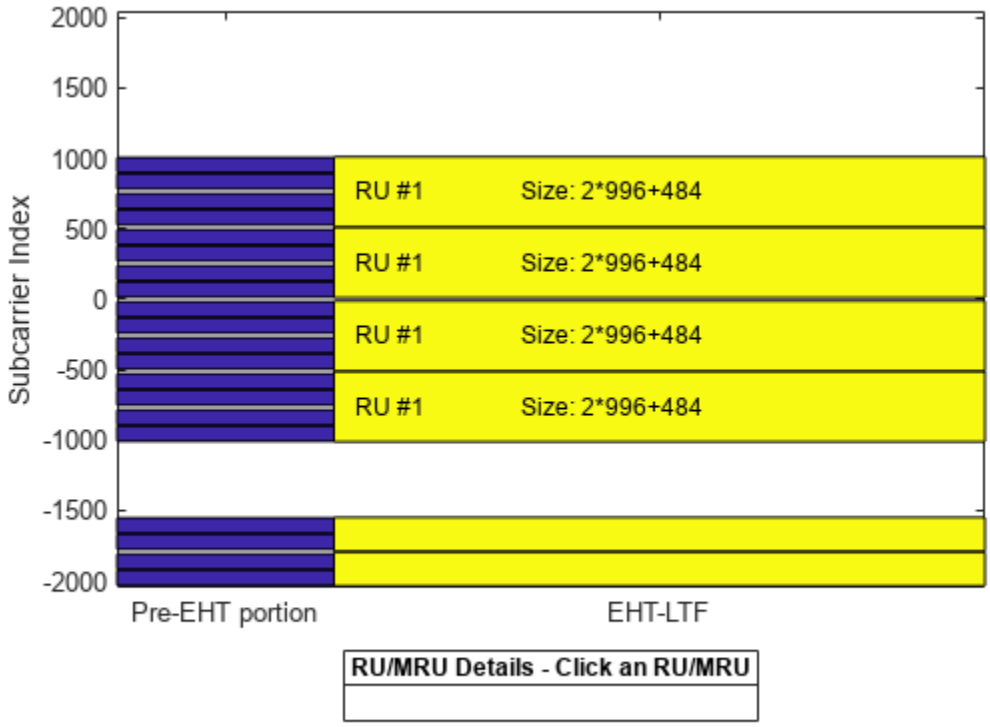
Display the puncturing pattern determined by the punctured channel field value.

```
cfg.PuncturingPattern
```

```
ans = 1x8 logical array

   0   1   0   0   0   0   1   1
```

Show the RU allocation of the configuration. The punctured 40 MHz subchannels do not appear in the figure.

```
showAllocation(cfg)
```



Version History

Introduced in R2022b

References

[1] IEEE P802.11be/D1.5. “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 8: Enhancements for Extremely High Throughput (EHT).” Draft Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

[2] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

Extended Capabilities

C/C++ Code Generation

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

You must specify the data type of the `PostFECPaddingBits` property of the `User` property as `int8`.

See Also

Objects
`wlanDMGConfig` | `wlanEHTRU` | `wlanEHTUser` | `wlanHEMUConfig` | `wlanHERecoveryConfig` | `wlanHESUConfig` | `wlanHETBConfig` | `wlanHTConfig` | `wlanNonHTConfig` | `wlanS1GConfig` | `wlanVHTConfig`

Functions
`numPostFECPaddingBits` | `packetFormat` | `psduLength` | `ruInfo` | `showAllocation` | `transmitTime` | `wlanWaveformGenerator`

wlanEHTRU

Configure RU for EHT MU transmission

Description

The `wlanEHTRU` object contains properties that configure a WLAN extremely high-throughput (EHT) resource unit (RU). The `RU` property of a `wlanEHTMUConfig` object is a cell array of `wlanEHTRU` objects. For an OFDMA `wlanEHTMUConfig` object, the `AllocationIndex` property determines the `RU` property. For a non-OFDMA `wlanEHTMUConfig` object, the `ChannelBandwidth` property determines the `RU` property.

Creation

Syntax

```
cfgEHT.RU = wlanEHTRU(Size,Index,UserNumbers)
cfgEHT.RU = wlanEHTRU( ___ ,Name=Value)
```

Description

`cfgEHT.RU = wlanEHTRU(Size,Index,UserNumbers)` creates an object that contains properties to configure an EHT-format RU. The `Size` input is the RU size, `Index` is the RU index, and `UserNumbers` specifies the indices of users transmitted on the RU.

`cfgEHT.RU = wlanEHTRU( ___ ,Name=Value)` sets properties using one or more name-value pairs.

Properties

`PowerBoostFactor` — Power boost factor
1 (default) | scalar in the range [0.5, 2]

Power boost factor, specified as a scalar in the range [0.5, 2].

Data Types: `double`

`SpatialMapping` — Spatial mapping scheme
`"Direct"` (default) | `"Hadamard"` | `"Fourier"` | `"Custom"`

Spatial mapping scheme, specified as `"Direct"`, `"Hadamard"`, `"Fourier"`, or `"Custom"`.

The default value, `"Direct"`, applies only when you set the `NumTransmitAntennas` property equal to the sum of the number of space-time streams for all users assigned to the RU.

Data Types: `char` | `string`

`SpatialMappingMatrix` — Spatial mapping matrix
1 (default) | complex-valued scalar | complex-valued matrix | complex-valued 3-D array

Spatial mapping matrix, specified as one of these values:

- A complex-valued scalar. This value applies to all the subcarriers.
- A complex-valued matrix of size $N_{STS_{Total}}$ -by-N_T, where:
  - $N_{STS_{Total}}$ is the sum of the number of the space-time streams for all users assigned to the RU;
  - N_T is the number of transmit antennas.

  In this case, the spatial mapping matrix applies to all the subcarriers.
- A complex-valued 3-D array of size `Size`-by-$N_{STS_{Total}}$-by-N_T. The `ChannelBandwidth` property determines the value of the `Size` property. In this case, each occupied subcarrier has its own spatial mapping matrix.

Use this property to rotate and scale the output vector of the constellation mapper. The spatial mapping matrix is used for beamforming and mixing space-time streams over the transmit antennas. The calling function normalizes the spatial mapping matrix for each subcarrier.

Example: `[0.6 0.2; 0.5 0.4; 0.5 0.9]` represents a spatial mapping matrix with three space-time streams and two transmit antennas.

Dependencies

This property applies only when you set the `SpatialMapping` property to `"Custom"`.

Data Types: `double`
Complex Number Support: Yes

`Beamforming` — Enable beamforming

`true` or `1` (default) | `false` or `0`

Enable beamforming by setting this property to `true` or `1`. The `SpatialMappingMatrix` property specifies the beamform steering matrix.

Dependencies

This property applies only when you set the `SpatialMapping` property to `"Custom"`.

Data Types: `logical`

`Size` — Resource unit size

`242` (default) | positive integer | vector of positive integers

Resource unit size, specified as a positive integer or a vector of positive integers. If the RU is an MRU, `Size` is a vector with entries given by the sizes of the RUs that comprise the MRU. Otherwise, `Size` is a scalar. The possible RU sizes are `26`, `52`, `106`, `242`, `484`, `968`, `996`, `1992`, and `3984`.

Note This property is read-only after you create the object.

Data Types: `double`

`Index` — Resource unit index

`1` (default) | integer in the range [1, 144] | vector of integers in the range [1, 144]

Resource unit index, specified as an integer in the range [1, 144] or a vector of such integers. If the RU is an MRU, `Index` is a vector with entries given by the indices of the RUs that comprise the MRU. Otherwise, `Index` is a scalar. Use this property to indicate the location of the RU within the channel.

Note This property is read-only after you create the object.

Example: An 80 MHz transmission has four possible 242-tone RUs, one in each 20 MHz subchannel. RU 242-1 (`Size` = 242, `Index` = 1) is the RU occupying the lowest absolute frequency within the 80MHz, and RU 242-4 (`Size` = 242, `Index` = 4) is the RU occupying the highest absolute frequency.

Data Types: `double`

`UserNumbers` — User index number
1 | integer | vector of integers

Indices of users transmitted on the RU, in one-based format, specified as an integer or a vector of integers. This property indexes the appropriate cell array element of the `User` property.

Data Types: `double`

Object Functions

Examples

Create Multi-User EHT OFDMA Configuration Objects

Create a multi-user EHT configuration object with the allocation index set to 48. This setting specifies an OFDMA configuration with one 106+26 tone MRU and one 106 tone RU in a 20 MHz channel. Both resource units have one user.

```
allocationIndex = 48;
cfgSMRU = wlanEHTMUConfig(allocationIndex);
```

Display the properties of the MRU.

```
cfgSMRU.RU{1}
```

```
ans =
  wlanEHTRU with properties:

    PowerBoostFactor: 1
      SpatialMapping: 'Direct'

   Read-only properties:
                Size: [106 26]
               Index: [1 5]
         UserNumbers: 1
```

Now create a multi-user EHT configuration object with the allocation index set to the vector [151 30 30 30 64 64 29 29]. This setting specifies a 996+484-tone MRU and two 242-tone RUs in a 160 MHz channel. The MRU has eight users and the two RUs have one user each.

```
allocationIndex = [151 30 30 30 64 64 29 29];
cfgLMRU = wlanEHTMUConfig(allocationIndex);
```

For multi-user OFDMA transmissions with a channel bandwidth of 160 MHz or higher, the EHT-SIG content channels can carry different information per 80 MHz segment. For these bandwidths, the

read-only property `AllocationIndex` is a matrix of size M-by-N, where M is the number of 80 MHz segments and N is the number of 20 MHz subchannels. In this example M is 2 and N is 8. Each row of the matrix represents an 80 MHz segment.

Display the `AllocationIndex` property.

```
disp(cfgLMRU.AllocationIndex);
```

```
   151    30    30    30    28    28    29    29
    30    30    30    30    64    64    29    29
```

Display the properties of the MRU.

```
cfgLMRU.RU{1}
```

```
ans =
  wlanEHTRU with properties:

    PowerBoostFactor: 1
      SpatialMapping: 'Direct'

   Read-only properties:
                Size: [996 484]
               Index: [1 4]
         UserNumbers: [1 2 3 4 5 6 7 8]
```

Now create a multi-user EHT configuration object with the allocation index set to the 2-by-8 matrix obtained by duplicating the previous allocation index. In this case, the EHT-SIG content channels carry the same information per 80 MHz segment.

```
allocationIndex = [allocationIndex;allocationIndex];
cfg2by8 = wlanEHTMUConfig(allocationIndex);
```

Display the `AllocationIndex` property.

```
disp(cfg2by8.AllocationIndex);
```

```
   151    30    30    30    64    64    29    29
   151    30    30    30    64    64    29    29
```

Verify that the two configuration objects have the same resource unit allocations.

```
isequal(ruInfo(cfgLMRU),ruInfo(cfg2by8))
```

```
ans = logical
   1
```

Version History

Introduced in R2022b

See Also

Objects
`wlanEHTMUConfig` | `wlanEHTUser`

wlanEHTUser

Configure users for EHT MU transmission

Description

The `wlanEHTUser` object contains properties of a user within a WLAN extremely high-throughput multi-user (EHT MU) resource unit (RU). The `User` property of a `wlanEHTMUConfig` object is a cell array of `wlanEHTUser` objects. For an OFDMA `wlanEHTMUConfig` object, the `AllocationIndex` property determines the `User` property. For a non-OFDMA `wlanEHTMUConfig` object, the `ChannelBandwidth` property determines the `User` property.

Creation

Syntax

```
cfgEHT.User = wlanEHTUser(RUNumber)
cfgEHT.User = wlanEHTUser( ___ ,Name=Value)
```

Description

`cfgEHT.User = wlanEHTUser(RUNumber)` creates an EHT user configuration object for `RUNumber`, the input RU number.

`cfgEHT.User = wlanEHTUser( ___ ,Name=Value)` sets properties using one or more name-value pairs.

Properties

`APEPLength` — APEP length

`100` (default) | integer in the range [0, 15523198]

Aggregated MPDU (A-MPDU) pre-end-of-frame (pre-EOF) padding (APEP) length, in bytes, specified as an integer in the range [0, 15523198].

The object uses this property to determine the number of OFDM symbols in the data field. For more information, see [1].

Data Types: `double`

`MCS` — MCS used for transmission

`0` (default) | integer in the range [0, 13] | 15

Modulation and coding scheme (MCS) used for transmission, specified as a nonnegative integer in the range [0, 13], or 15. This table shows the modulation type and coding rate for each valid value of `MCS`:

MCS	Modulation	Dual Carrier Modulation	Coding Rate
0	Binary phase-shift keying (BPSK)	Not applicable	1/2
1	Quadrature phase-shift keying (QPSK)	Not applicable	1/2
2		Not applicable	3/4
3	16-point quadrature amplitude modulation (16-QAM)	Not applicable	1/2
4			3/4
5	64-QAM	Not applicable	2/3
6			3/4
7			5/6
8	256-QAM		3/4
9			5/6
10	1024-QAM		3/4
11			5/6
12	4096-QAM		3/4
13			5/6
15	BPSK-DCM	`0` or `1`	1/2

A value of 14 for the `MCS` property indicates that EHT DUP mode is in use. To use EHT DUP mode, set the EHTDUPMode property to `true` or `1`.

Data Types: `double`

`NumSpaceTimeStreams` — Number of space-time streams

1 (default) | positive integer

Number of space-time streams in the transmission, specified as an integer in the range [1, 8]. The maximum number of space-time streams for any user within a MU-MIMO RU is four. The maximum value of the sum of the number of space-time streams over all users in an RU is eight. For information on these and other restrictions on the number of space-time streams, see table 36-42 of [1].

Data Types: `double`

`ChannelCoding` — Forward error correction coding type

`"LPDC"` (default) | `"BCC"`

Forward-error-correction (FEC) coding type for the EHT-Data field, specified as `"LDPC"` for low-density parity-check (LDPC) coding or `"BCC"` for binary convolutional coding (BCC).

You can set this property to `"BCC"` only when all of these conditions are satisfied:

- The `MCS` property is in the range [0, 9], or equal to 15.
- The size of any RU is less than or equal to 242. Obtain the RU sizes by using the `ruInfo` object function.
- The `NumSpaceTimeStreams` property is less than or equal to 4.

Data Types: `char` | `string`

`STAID` — STA identifier

`0` (default) | integer in the range [0, 2047]

Station (STA) identifier, specified as an integer in the range [0, 2047]. The value of this property specifies the station association identifier (AID) field as defined in section 35.12.1.1 of [1]. The 11 least significant bits (LSBs) of the AID field are used to address the STA. When you create a `wlanEHTUser` object, the default value of this property is 0 for the first `wlanEHTUser` and increases by 1 for each subsequent object. When you set this property to `2046`, the associated RU carries no data.

Data Types: `double`

`RUNumber` — RU number

`1` (default) | integer | vector of integers

RU number, specified as an integer or a vector of integers. This property indexes the appropriate cell array elements of the RU property.

Note This property is read-only after you create the object.

Data Types: `double`

`NominalPacketPadding` — Nominal packet padding

`0` (default) | `8` | `16` | `20`

Nominal packet padding, in microseconds, specified as `0`, `8`, `16`, or `20`. The `wlanEHTUser` object uses this property and the pre-forward-error-correction (pre-FEC) padding factor to calculate the duration, T_{PE}, of the packet extension field. For more information about the packet extension field, see section 36.3.14 of [1].

This table shows the possible values of T_{PE} for different values of this property and a, a parameter defined by equation (36-58) or (36-59) of [1].

Value of a	Value of T_{PE} in Microseconds			
	`NominalPacketPadding` Set to `0`	`NominalPacketPadding` Set to `8`	`NominalPacketPadding` Set to `16`	`NominalPacketPadding` Set to `20`
1	0	0	4	8
2	0	0	8	12
3	0	4	12	16
4	0	8	16	20

For an EHT MU PPDU, equation 36-91 of [1] defines the value of T_{PE} as the maximum of the T_{PE} values that you specify for each user.

Data Types: `double`

`PostFECPaddingSource` — Post-FEC padding bit source

`"mt19937arWithSeed"` (default) | `"GlobalStream"` | `"UserDefined"`

Post-FEC padding bit source used by the `wlanWaveformGenerator` function, specified as one of these values.

- `"mt19937arWithSeed"` — Generate normally distributed random bits by using the mt19937ar algorithm with seed specified in the `PostFECPaddingSeed` property.
- `"GlobalStream"` — Generate normally distributed random bits by using the current global random number stream.
- `"UserDefined"` — Use the bits specified in the `PostFECPaddingBits` property as the post-FEC padding bits.

Data Types: `char` | `string`

`PostFECPaddingSeed` — Post-FEC padding bit seed for mt19937ar algorithm
`73` (default) | nonnegative integer

Post-FEC padding bit seed for the mt19937ar algorithm, specified as a nonnegative integer.

Dependencies

To enable this property, set the `PostFECPaddingSource` property to `"mt19937arWithSeed"`.

Data Types: `double`

`PostFECPaddingBits` — Post-FEC padding bits
`0` (default) | binary-valued column vector

Post-FEC padding bits, specified as a binary-valued scalar or column vector.

To generate a waveform, the `wlanWaveformGenerator` function requires *n* bits, where *n* depends on the specified configuration. To calculate *n*, use the `numPostFECPaddingBits` object function with the specified configuration object as the input argument and specify this property as a vector of length *n*. Alternatively, specify this input as a binary-valued scalar or column vector of arbitrary length. If the length of this property is less than *n*, the waveform generator loops the vector to create a vector of length *n*. If the length of this property is greater than *n*, the function uses only the first *n* entries as the post-FEC padding bits.

Note For C/C++ code generation, you must specify the data type of this property as `int8`.

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

Object Functions

Examples

Create EHT Non-OFDMA Configuration Object Using DUP Mode

Create a non-OFDMA EHT MU configuration object. Set the channel bandwidth to 160 MHz.

```
cfg = wlanEHTMUConfig("CBW160");
```

Display the configuration properties of the user. The modulation and coding scheme is `0`.

```
cfg.User{1}
```

```
ans =
  wlanEHTUser with properties:

              APEPLength: 100
                     MCS: 0
     NumSpaceTimeStreams: 1
           ChannelCoding: 'LDPC'
                   STAID: 0
    NominalPacketPadding: 0
    PostFECPaddingSource: 'mt19937arWithSeed'
      PostFECPaddingSeed: 1

   Read-only properties:
                RUNumber: 1
```

Create another non-OFDMA EHT MU configuration object with the same channel bandwidth. Enable EHT DUP mode.

```
cfg = wlanEHTMUConfig("CBW160", EHTDUPMode=1);
```

Display the modulation and coding scheme used. The value 14 indicates that EHT DUP mode is being used.

```
cfg.User{1}.MCS
```

```
ans = 14
```

Version History

Introduced in R2022b

See Also

Objects

`wlanEHTMUConfig` | `wlanEHTRU`

wlanHEMUConfig

Configure HE MU transmission

Description

The `wlanHEMUConfig` object is a configuration object for the WLAN HE multi-user (HE MU) packet format.

Creation

Syntax

```
cfgHEMU = wlanHEMUConfig(AllocationIndex)
cfgHEMU = wlanHEMUConfig(AllocationIndex,Name,Value)
```

Description

`cfgHEMU = wlanHEMUConfig(AllocationIndex)` creates `cfgHEMU`, a configuration object that initializes transmit parameters for an IEEE 802.11 HE MU "PPDU" on page 4-77 for `AllocationIndex`, the input resource unit allocation. For a detailed description of the HE WLAN format, see [2].

`cfgHEMU = wlanHEMUConfig(AllocationIndex,Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotation marks. For example, `wlanHEMUConfig([200 114 114 200],'LowerCenter26ToneRU',true)` specifies an 80 MHz bandwidth allocation for three users on three RUs with lower center 26-tone RU allocation signaling.

At runtime, the calling function validates object settings for properties relevant to the operation of the function.

Properties

`AllocationIndex` — Resource unit allocation index

integer | vector of integers | character vector | cell array

Resource unit (RU) allocation index for each 20 MHz subchannel. This property defines the number of RUs, the size of each RU, and the number of users assigned to each RU. For more information, see "Allocation Index".

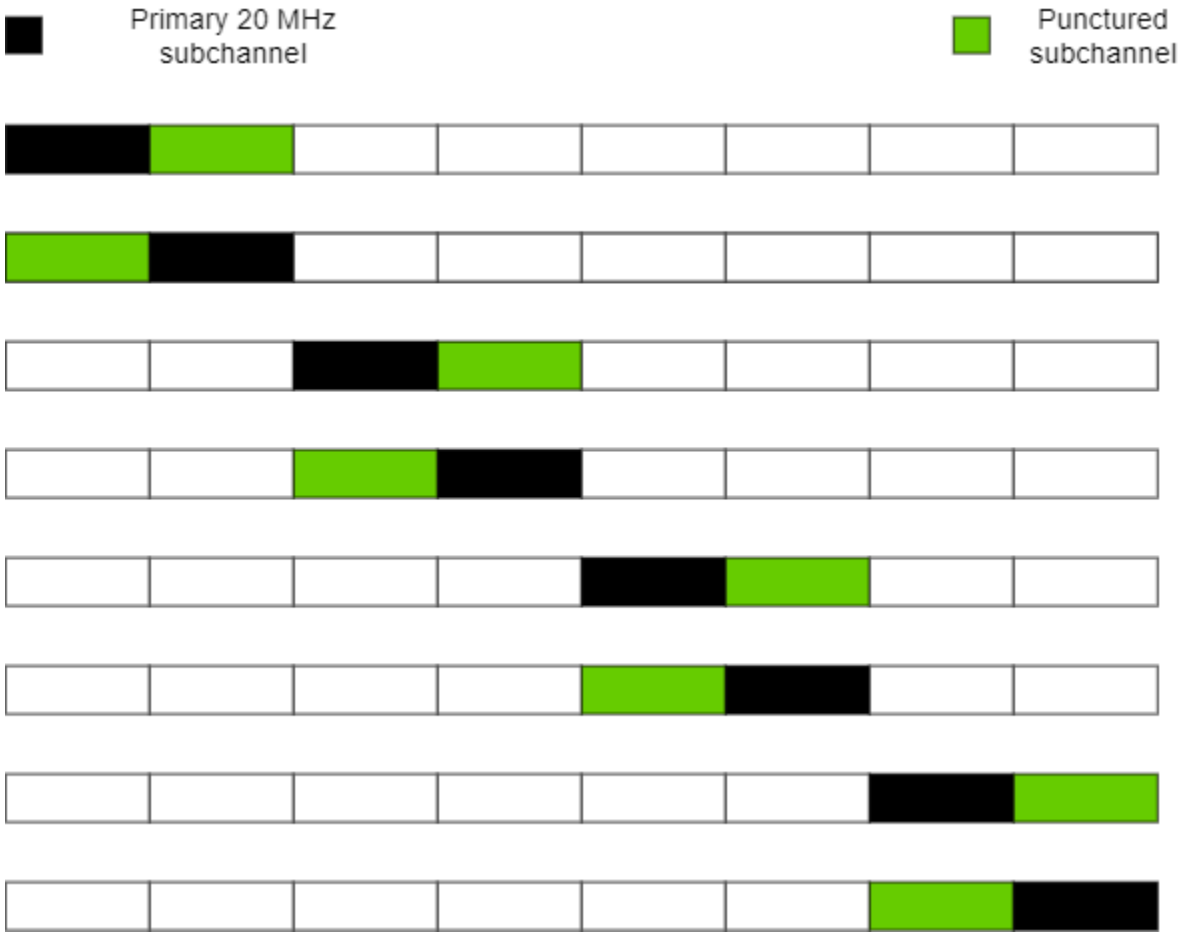
You can specify this value as an integer, a vector of integers, a string array, a character vector, or a cell array of character vectors. The format in which you specify these indices depends on how many you are specifying.

- Specify a single allocation index using one integer in either of these forms.
  - An integer in the interval [0, 223]
  - An 8-bit binary sequence specified as a string or character vector

- Specify multiple allocation indices using two, four, or eight integer values in any of these forms.
  - A vector of integers in the interval [0, 223]
  - An 8-bit binary sequence specified as a string array
  - An 8-bit binary sequence specified as a cell array of character vectors

  In an 80 MHz transmission, you can signal a punctured 20 MHz subchannel at any location.

  In a 160 MHz transmission, you can signal a punctured 20 or 40 MHz subchannel. You can only signal a punctured 20 MHz subchannel in the 40 MHz subchannel containing the primary 20 MHz subchannel, as shown in this figure.



  Similarly, you can only signal a punctured 40 MHz subchannel in the 80 MHz subchannel containing the primary 20 MHz subchannel, as shown in this figure.

To signal a punctured 20 MHz subchannel, set the corresponding element of this property to 113. To signal a punctured 40 MHz subchannel, set the two corresponding adjacent elements to 114.

To signal an empty HE-SIG-B user field in an HE-SIG-B content channel, set the corresponding element of this property to 114 or 115.

Note This property is read-only after the object is created.

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

`ChannelBandwidth` — Channel bandwidth of PPDU transmission

`'CBW20'` (default) | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth of PPDU transmission, specified as one of these values:

- `'CBW20'` – Channel bandwidth of 20 MHz
- `'CBW40'` – Channel bandwidth of 40 MHz
- `'CBW80'` – Channel bandwidth of 80 MHz
- `'CBW160'` – Channel bandwidth of 160 MHz

When you create a `wlanHEMUConfig` object, this property is configured based on the value to which you set the `AllocationIndex` property.

Note This property is read-only after the object is created.

Data Types: `char` | `string`

`LowerCenter26ToneRU` — Enable lower center 26-tone RU allocation signaling

`false` or `0` (default) | `true` or `1`

Enable lower center 26-tone RU allocation signaling, specified as a numeric or logical `1` (`true`) or `0` (`false`). To enable the lower frequency center 26-tone RU, set this property to `1` (`true`). This property can be set during object creation only.

Dependencies

This property applies only when the `AllocationIndex` property defines a channel bandwidth of 80 MHz or 160 MHz and does not specify a full bandwidth allocation.

Data Types: `logical`

`UpperCenter26ToneRU` — Enable upper center 26-tone RU allocation signaling

`false` or `0` (default) | `true` or `1`

Enable upper center 26-tone RU allocation signaling, specified as a numeric or logical `1` (`true`) or `0` (`false`). To enable the upper frequency center 26-tone RU, set this property to `1` (`true`). This property can be set during object creation only.

Dependencies

This property applies only when the `AllocationIndex` property defines a channel bandwidth of 80 MHz or 160 MHz and does not specify a full bandwidth allocation.

Data Types: `logical`

`RU` — Transmission properties of each RU in transmission

cell array of `wlanHEMURU` objects

Transmission properties of each RU in the transmission, specified as a cell array of `wlanHEMURU` objects. When you create a `wlanHEMUConfig` object, the `AllocationIndex` property determines this property.

Data Types: `cell`

`User` — Transmission properties of each user

cell array of `wlanHEMUUser` objects

Transmission properties of each user, specified as a cell array of `wlanHEMUUser` objects. When you create a `wlanHEMUConfig` object, the `AllocationIndex` property determines this property.

Data Types: `cell`

`PrimarySubchannel` — Index of primary 20 MHz subchannel

`1` (default) | integer in the interval [1, 8]

Index of primary 20 MHz subchannel in an 80 or 160 MHz transmission, specified as one of these options.

- When the `ChannelBandwidth` property is `'CBW80'`, set this property to an integer in the interval [1, 4].
- When the `ChannelBandwidth` property is `'CBW160'`, set this property to an integer in the interval [1, 8].

The location of the primary subchannel and the preamble puncturing pattern (defined by the `AllocationIndex` property) determine the bandwidth value signaled in the HE-SIG-A field of the transmission, as specified in Table 27-20 of [2].

Dependencies

This property applies only when the `AllocationIndex` property defines a channel bandwidth of 80 MHz or 160 MHz.

Data Types: `double`

`NumTransmitAntennas` — Number of transmit antennas

`1` (default) | positive integer

Number of transmit antennas, specified as a positive integer.

Data Types: `double`

`PreHECyclicShifts` — Cyclic shift values of additional transmit antennas

`-75` (default) | integer in the interval [–200, 0] | row vector

Cyclic shift values, in nanoseconds, of additional transmit antennas for the pre-HE fields of the waveform. The first eight antennas use the cyclic shift values specified in Table 21-10 of [1]. The remaining L antennas use the values you specify in this property, where L = `NumTransmitAntennas` – 8. Specify this property as one of these values:

- An integer in the interval [–200, 0] – the `wlanHEMUConfig` object uses this cyclic shift value for each of the L additional antennas.
- A row vector of length L of integers in the interval [–200, 0] – the `wlanHEMUConfig` object uses the kth element as the cyclic shift value for the (k + 8)th transmit antenna.

> **Note** If you specify this property as a row vector of length greater than L, the `wlanHEMUConfig` object uses only the first L elements. For example, if you set the `NumTransmitAntennas` property to `16`, the `wlanHEMUConfig` object uses only the first L = 16 – 8 = 8 elements of this vector.

Dependencies

To enable this property, set the `NumTransmitAntennas` property to a value greater than `8`.

Data Types: `double`

`STBC` — Enable space-time block coding

`false` or `0` (default) | `true` or `1`

Enable space-time block coding (STBC) of the PPDU data field for all users, specified as a numeric or logical `1` (`true`) or `0` (`false`). STBC transmits multiple copies of the data stream across assigned antennas.

- When you set this property to `0` (`false`), STBC is not applied to the data field. The number of space-time streams is equal to the number of spatial streams.
- When you set this property to `1` (`true`), STBC is applied to the data field. The number of space-time streams is twice the number of spatial streams.

Dependencies

This property applies only when all of these conditions are satisfied:

- The `NumSpaceTimeStreams` subproperty of each element of the `User` property is `2`.
- The `MCS` subproperty of each element of the `User` property is `0` (`false`).
- No RU specifies multi-user multiple input/multiple output (MU-MIMO).

Data Types: `logical`

`GuardInterval` — Guard interval (cyclic prefix) duration

`3.2` (default) | `1.6` | `0.8`

Guard interval (cyclic prefix) duration for the data field within a packet, in microseconds, specified as `3.2`, `1.6`, or `0.8`.

Data Types: `double`

`HELTFType` — HE-LTF compression mode

`4` (default) | `2`

HE-LTF compression mode, specified as `4` or `2`. This property indicates the type of HE-LTF, where a value of `4` or `2`corresponds to four times or two times HE-LTF duration compression mode, respectively. The HE-LTF type is enumerated in Table 27-1 of [2] as:

- 2xHE-LTF – Duration of 6.4 μs with a guard interval duration of 0.8 μs or 1.6 μs
- 4xHE-LTF – Duration of 12.8 μs with a guard interval duration of 0.8 μs or 3.2 μs

For more information on the HE-LTF, see section 27.3.11.10 of [2].

Data Types: `double`

`SIGBCompression` — HE-SIG-B compression indicator

`true` or `1` (default) | `false` or `0`

HE-SIG-B compression indicator, specified as a numeric or logical `1` (`true`) or `0` (`false`). To enable HE-SIG-B compression for a full-bandwidth 20 MHz MU-MIMO transmission, set this property to `1` (`true`).

Dependencies

This property applies only when you indicate a 20 MHz channel bandwidth by setting the `AllocationIndex` to a value in the interval [192,199].

Note This property is not used for bandwidths of 40, 80, or 160 MHz because, at these bandwidths, the `AllocationIndex` alone determines if HE-SIG-B compression is used:

- If you specify a single, full-band allocation index, the transmission uses HE-SIG-B compression. The allocation index is not signaled.

- If you specify 2, 4, or 8 full-band allocation indices, the transmission does not use HE-SIG-B compression. The allocation indices are signaled.

For more information, see the tutorial on “HE MU Transmission” or the example “Demonstrate SIGB Compression in HE MU Waveforms” on page 4-70.

Data Types: `logical`

`SIGBMCS` — MCS for HE-SIG-B field
`0` (default) | integer in the interval [0, 5]

Modulation and coding scheme (MCS) for the HE-SIG-B field, specified as an integer in the interval [0, 5].

Data Types: `double`

`SIGBDCM` — Enable DCM for HE-SIG-B field
`false` or `0` (default) | `true` or `1`

HE-SIG-B dual-carrier modulation (DCM) indicator, specified as a numeric or logical `1` (`true`) or `0` (`false`). A value of `1` (`true`) indicates that the HE-SIG-B field is modulated with DCM. A value of `0` (`false`) indicates that the HE-SIG-B field is not modulated with DCM.

Dependencies

This property applies only when the `MCS` subproperty of each element of the `User` property is `0`, `1`, `3`, or `4`.

Data Types: `logical`

`UplinkIndication` — Uplink indication
`false` or `0` (default) | `true` or `1`

Uplink indication, specified as a numeric or logical `1` (`true`) or `0` (`false`). To indicate that the PPDU is sent on a downlink transmission, set this property to `0` (`false`). To indicate that the PPDU is sent on an uplink transmission, set this property to `1` (`true`).

Data Types: `logical`

`BSSColor` — Basic service set color identifier
`0` (default) | integer in the interval [0, 63]

Basic service set (BSS) color identifier, specified as an integer in the interval [0, 63].

Data Types: `double`

`SpatialReuse` — Spatial reuse indication
`0` (default) | integer in the interval [0, 15]

Spatial reuse indication, specified as an integer in the interval [0, 15].

Data Types: `double`

`TXOPDuration` — Duration information for TXOP protection
`127` (default) | integer in the interval [0, 127]

Duration information for transmit opportunity (TXOP) protection, specified as an integer in the interval [0, 127]. Except for the first bit, which specifies TXOP length granularity, each bit of the

TXOP field of HE-SIG-A is equal to `TXOPDuration`. Therefore a duration in microseconds must be converted according to the procedure set out in Table 27-18 of [2].

Data Types: `double`

`HighDoppler` — High-Doppler mode indicator

`false` or `0` (default) | `true` or `1`

High-Doppler mode indicator, specified as a numeric or logical `0` (`false`) or `1` (`true`). To indicate high-Doppler mode in the HE-SIG-A field, set this property to `1` (`true`).

Dependencies

The `1` (`true`) value for this property is valid only when the `NumSpaceTimeStreams` subproperty of each element of the `RU` property is less than or equal to 4.

Data Types: `logical`

`MidamblePeriodicity` — Midamble periodicity of HE-data field

`10` (default) | `20`

Midamble periodicity of the HE-data field, in number of OFDM symbols, specified as `10` or `20`.

Dependencies

This property applies only when the `HighDoppler` property is `1` (`true`).

Data Types: `double`

Object Functions

getNumPostFECPaddingBits	Calculate required number of post-FEC padding bits
getPSDULength	Calculate HE or WUR PSDU length
packetFormat	WLAN packet format
ruInfo	Resource unit allocation information
showAllocation	Resource unit (RU) allocation
transmitTime	Packet transmission time

Examples

Create Multiuser HE Configuration Object

Create a 20 MHz multiuser HE configuration object with the allocation index set to 0. An allocation index of 0 specifies nine 26-tone RUs in a 20 MHz channel.

```
cfgMU = wlanHEMUConfig(0);
for i=1:numel(cfgMU.User)
    % Set the APEPLength of each user
    cfgMU.User{i}.APEPLength = 100;
end
```

Display the configuration object properties for the fourth user.

```
cfgMU.User{4}
```

```
ans =
  wlanHEMUUser with properties:
```

```
            APEPLength: 100
                   MCS: 0
     NumSpaceTimeStreams: 1
                   DCM: 0
          ChannelCoding: 'LDPC'
                 STAID: 0
    NominalPacketPadding: 0
    PostFECPaddingSource: 'mt19937ar with seed'
      PostFECPaddingSeed: 4

   Read-only properties:
              RUNumber: 4
```

Create HE MU Object Using Binary Allocation Indexing

Create an HE MU configuration object for a 40 MHz transmission with an allocation index of `11000000` for each 20 MHz subchannel. This configuration specifies two 242-tone RUs, each with one user.

```
cfgHEMU = wlanHEMUConfig(["11000000" "11000000"],'NumTransmitAntennas',2);
```

Configure the first RU and the first user.

```
cfgHEMU.RU{1}.SpatialMapping = 'Direct';
cfgHEMU.User{1}.APEPLength = 1e3;
cfgHEMU.User{1}.MCS = 2;
cfgHEMU.User{1}.NumSpaceTimeStreams = 2;
cfgHEMU.User{1}.ChannelCoding = 'LDPC';
cfgHEMU.User{1}.NominalPacketPadding = 16;
```

Configure the second RU and the second user.

```
cfgHEMU.RU{2}.SpatialMapping = 'Fourier';
cfgHEMU.User{2}.APEPLength = 500;
cfgHEMU.User{2}.MCS = 3;
cfgHEMU.User{2}.NumSpaceTimeStreams = 1;
cfgHEMU.User{2}.ChannelCoding = 'LDPC';
cfgHEMU.User{2}.NominalPacketPadding = 8;
```

Display the configuration object properties for both RUs and both users.

```
disp(cfgHEMU)
```

```
  wlanHEMUConfig with properties:

                    RU: {[1x1 wlanHEMURU]  [1x1 wlanHEMURU]}
                  User: {[1x1 wlanHEMUUser]  [1x1 wlanHEMUUser]}
    NumTransmitAntennas: 2
                  STBC: 0
         GuardInterval: 3.2000
             HELTFType: 4
               SIGBMCS: 0
               SIGBDCM: 0
       UplinkIndication: 0
```

```
              BSSColor: 0
           SpatialReuse: 0
           TXOPDuration: 127
            HighDoppler: 0

   Read-only properties:
       ChannelBandwidth: 'CBW40'
        AllocationIndex: [192 192]
```

```
cfgHEMU.RU{1:2}
```

```
ans =
  wlanHEMURU with properties:

    PowerBoostFactor: 1
      SpatialMapping: 'Direct'

   Read-only properties:
                Size: 242
               Index: 1
         UserNumbers: 1


ans =
  wlanHEMURU with properties:

    PowerBoostFactor: 1
      SpatialMapping: 'Fourier'

   Read-only properties:
                Size: 242
               Index: 2
         UserNumbers: 2
```

```
cfgHEMU.User{1:2}
```

```
ans =
  wlanHEMUUser with properties:

             APEPLength: 1000
                    MCS: 2
     NumSpaceTimeStreams: 2
                    DCM: 0
           ChannelCoding: 'LDPC'
                  STAID: 0
    NominalPacketPadding: 16
    PostFECPaddingSource: 'mt19937ar with seed'
      PostFECPaddingSeed: 1

   Read-only properties:
                RUNumber: 1


ans =
  wlanHEMUUser with properties:

             APEPLength: 500
                    MCS: 3
```

```
     NumSpaceTimeStreams: 1
                     DCM: 0
           ChannelCoding: 'LDPC'
                   STAID: 0
    NominalPacketPadding: 8
    PostFECPaddingSource: 'mt19937ar with seed'
      PostFECPaddingSeed: 2

   Read-only properties:
                RUNumber: 2
```

Demonstrate SIGB Compression in HE MU Waveforms

HE MU-MIMO Configuration With SIGB Compression

Generate a full bandwidth HE MU-MIMO configuration at 20 MHz bandwidth with SIGB compression. All three users are on a single content channel, which includes only the user field bits.

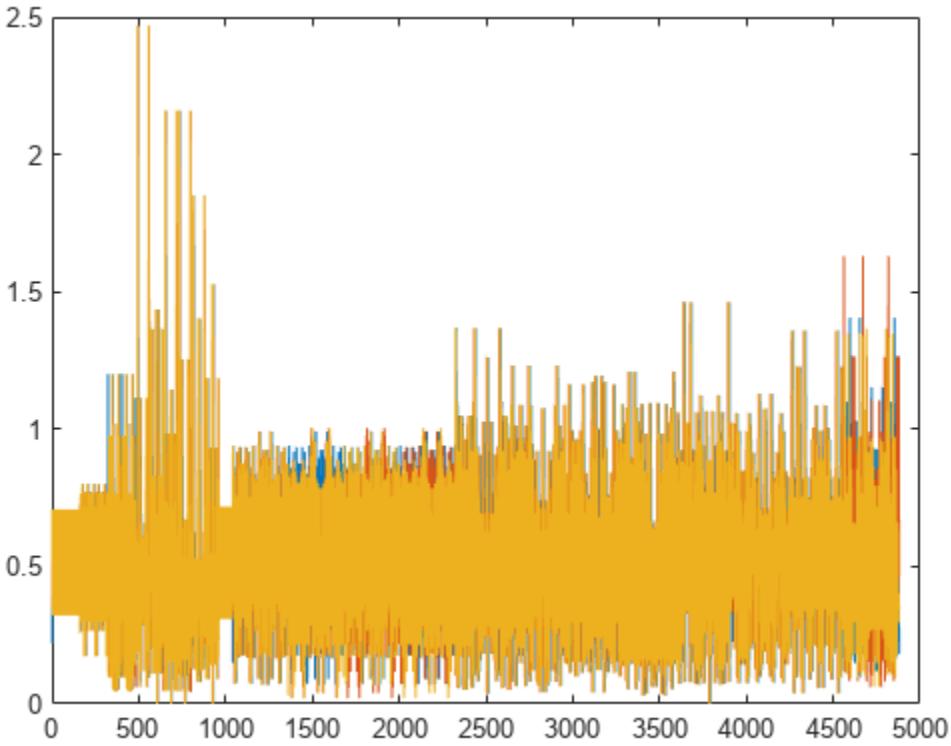
```
cfgHE = wlanHEMUConfig(194);
cfgHE.NumTransmitAntennas = 3;
```

Create PSDU data for all users.

```
psdu = cell(1,numel(cfgHE.User));
psduLength = getPSDULength(cfgHE);
for j = 1:numel(cfgHE.User)
    psdu = randi([0 1],psduLength(j)*8,1,'int8');
end
```

Generate and plot the waveform.

```
y = wlanWaveformGenerator(psdu,cfgHE);
plot(abs(y))
```



Generate a full bandwidth HE MU-MIMO waveform at 80 MHz bandwidth with SIGB compression. HE-SIG-B content channel 1 has four users. HE-SIG-B content channel 2 has three users.

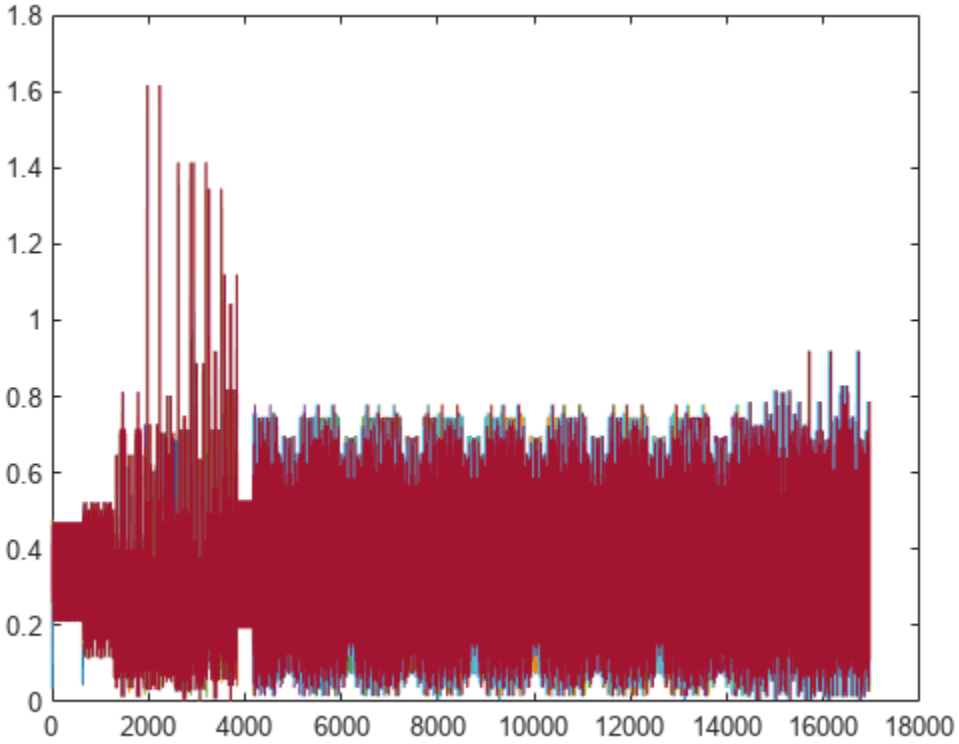
```
cfgHE = wlanHEMUConfig(214);
cfgHE.NumTransmitAntennas = 7;
```

Create PSDU data for all users.

```
psdu = cell(1,numel(cfgHE.User));
psduLength = getPSDULength(cfgHE);
for j = 1:numel(cfgHE.User)
    psdu = randi([0 1],psduLength(j)*8,1,'int8');
end
```

Generate and plot the waveform.

```
y = wlanWaveformGenerator(psdu,cfgHE);
plot(abs(y));
```



HE MU-MIMO Configuration Without SIGB Compression

Generate a full bandwidth HE MU-MIMO configuration at 20 MHz bandwidth without SIGB compression. All three users are on a single content channel, which includes both common and user field bits.

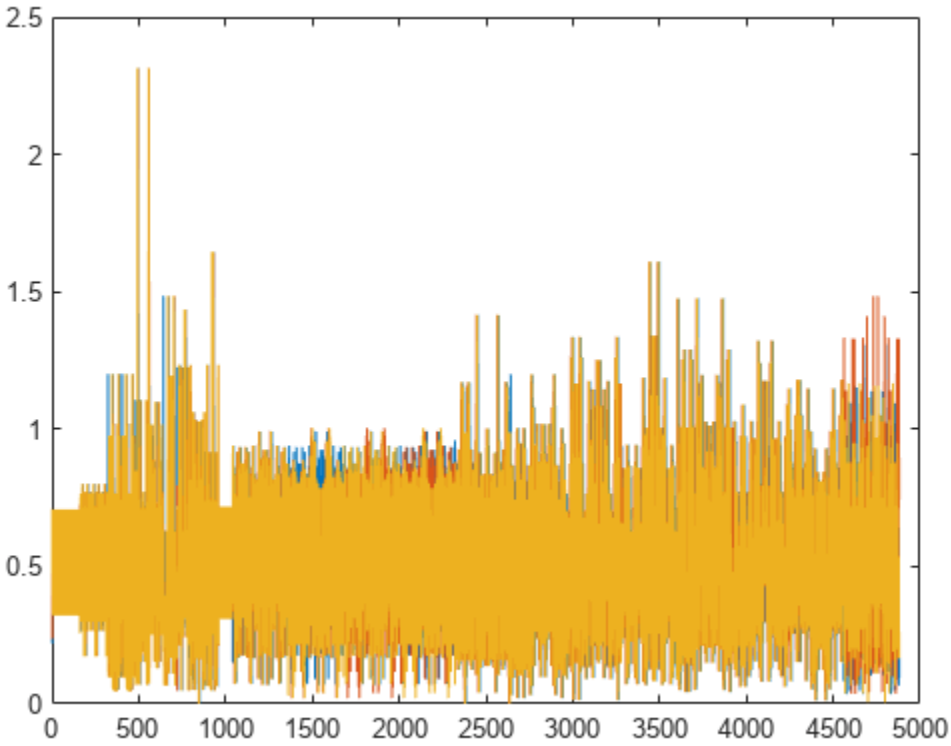
```
cfgHE = wlanHEMUConfig(194);
cfgHE.SIGBCompression = false;
cfgHE.NumTransmitAntennas = 3;
```

Create PSDU data for all users.

```
psdu = cell(1,numel(cfgHE.User));
psduLength = getPSDULength(cfgHE);
for j = 1:numel(cfgHE.User)
    psdu = randi([0 1],psduLength(j)*8,1,'int8');
end
```

Generate and plot the waveform.

```
y = wlanWaveformGenerator(psdu,cfgHE);
plot(abs(y))
```



Generate an 80 MHz HE MU waveform for six users without SIGB compression. HE-SIG-B content channel 1 has four users. HE-SIG-B content channel 2 has two users.

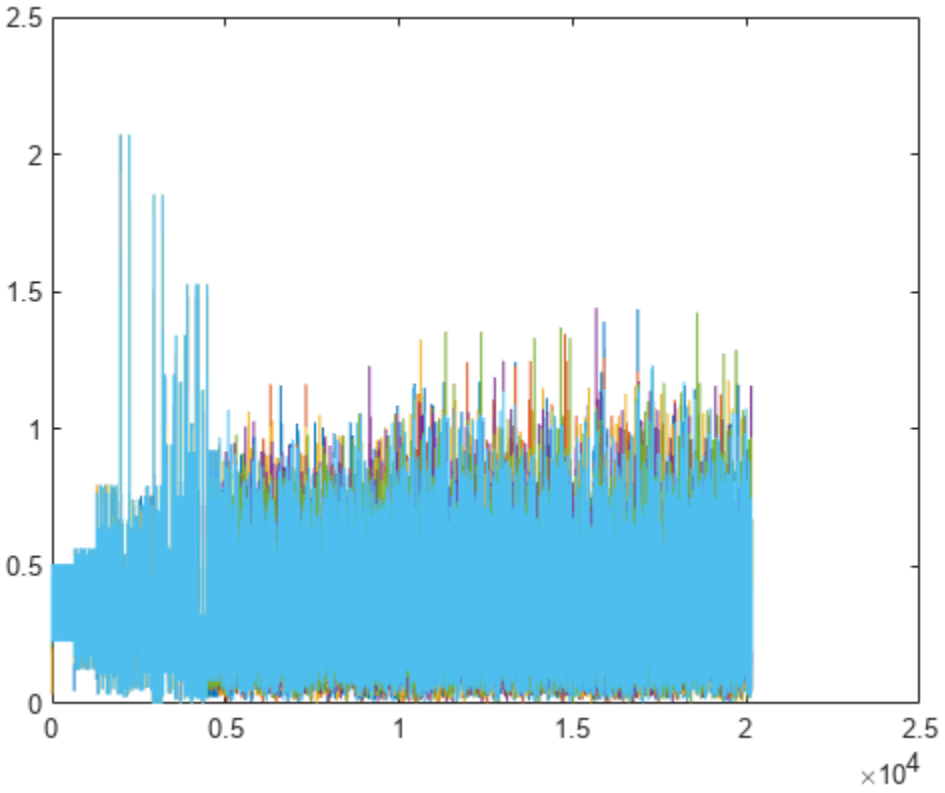
```
cfgHE = wlanHEMUConfig([202 114 192 193]);
cfgHE.NumTransmitAntennas = 6;
for i = 1:numel(cfgHE.RU)
    cfgHE.RU{i}.SpatialMapping = 'Fourier';
end
```

Create PSDU data for all users.

```
psdu = cell(1,numel(cfgHE.User));
psduLength = getPSDULength(cfgHE);
for j = 1:numel(cfgHE.User)
    psdu = randi([0 1],psduLength(j)*8,1,'int8');
end
```

Generate and plot the waveform.

```
y = wlanWaveformGenerator(psdu,cfgHE);
plot(abs(y));
```



Generate a full bandwidth HE MU-MIMO waveform at 80 MHz bandwidth without SIGB compression. HE-SIG-B content channel 1 has seven users. HE-SIG-B content channel 2 has zero users.

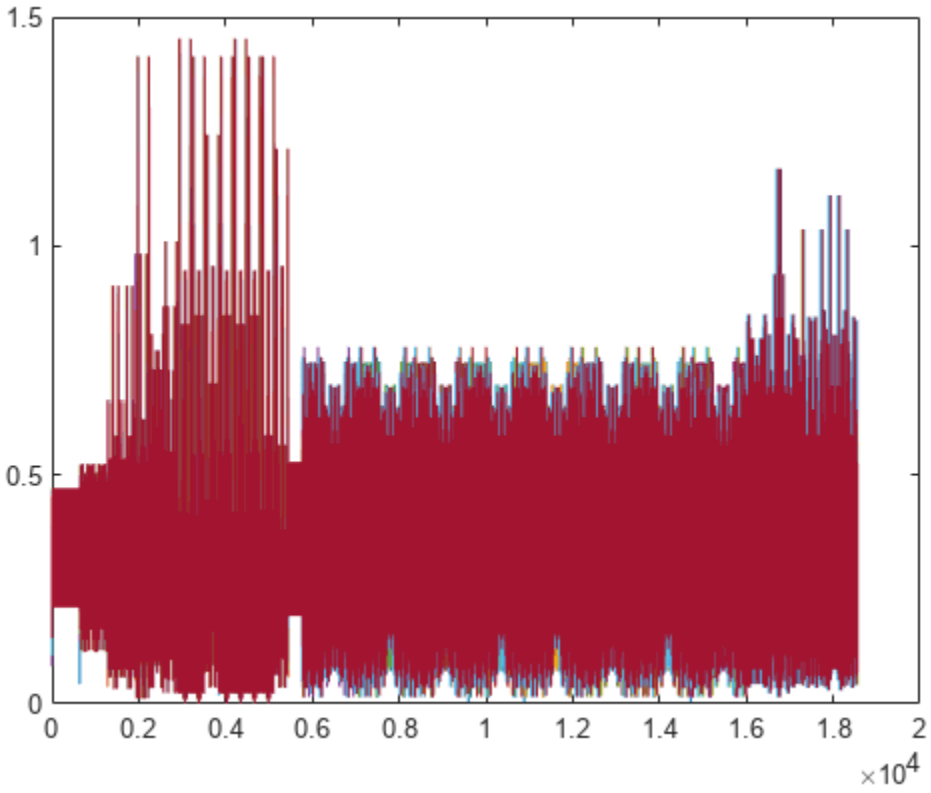
```
cfgHE = wlanHEMUConfig([214 115 115 115]);
cfgHE.NumTransmitAntennas = 7;
```

Create PSDU data for all users.

```
psdu = cell(1,numel(cfgHE.User));
psduLength = getPSDULength(cfgHE);
for j = 1:numel(cfgHE.User)
    psdu = randi([0 1],psduLength(j)*8,1,'int8');
end
```

Generate and plot the waveform.

```
y = wlanWaveformGenerator(psdu,cfgHE);
plot(abs(y))
```



Create Multiuser HE Object for Three Users in One RU with SIG-B Compression

Create an 80 MHz MU-MIMO configuration with three users in a single RU with SIG-B compression. Display the configuration object properties.

```
cfgMU = wlanHEMUConfig(210);
cfgMU.NumTransmitAntennas = 3;
cfgMU.User{1}.NumSpaceTimeStreams = 1;
cfgMU.User{2}.NumSpaceTimeStreams = 1;
cfgMU.User{3}.NumSpaceTimeStreams = 1;
disp(cfgMU)
```

```
  wlanHEMUConfig with properties:

                  RU: {[1x1 wlanHEMURU]}
                User: {1x3 cell}
    NumTransmitAntennas: 3
                  STBC: 0
         GuardInterval: 3.2000
             HELTFType: 4
               SIGBMCS: 0
               SIGBDCM: 0
       UplinkIndication: 0
              BSSColor: 0
          SpatialReuse: 0
          TXOPDuration: 127
```

```
           HighDoppler: 0

   Read-only properties:
       ChannelBandwidth: 'CBW80'
        AllocationIndex: 210
```

Create Multiuser HE Object Using Upper Center 26-Tone RU

Create a 160 MHz configuration using the upper center 26-tone RU. A total of four RUs are created. The RU tone assignments are 996, 484, 484, and 26. One user is allocated to each RU. The last RU created is the center 26-tone RU. Display the configuration properties of the object.

```
cfgMU = wlanHEMUConfig([208 115 115 115 200 114 114 200], ...
    'UpperCenter26ToneRU',true);
cfgMU.RU{:}
```

```
ans =
  wlanHEMURU with properties:

    PowerBoostFactor: 1
      SpatialMapping: 'Direct'

   Read-only properties:
                Size: 996
               Index: 1
         UserNumbers: 1


ans =
  wlanHEMURU with properties:

    PowerBoostFactor: 1
      SpatialMapping: 'Direct'

   Read-only properties:
                Size: 484
               Index: 3
         UserNumbers: 2


ans =
  wlanHEMURU with properties:

    PowerBoostFactor: 1
      SpatialMapping: 'Direct'

   Read-only properties:
                Size: 484
               Index: 4
         UserNumbers: 3


ans =
  wlanHEMURU with properties:

    PowerBoostFactor: 1
```

```
      SpatialMapping: 'Direct'

   Read-only properties:
                Size: 26
               Index: 56
         UserNumbers: 4
```

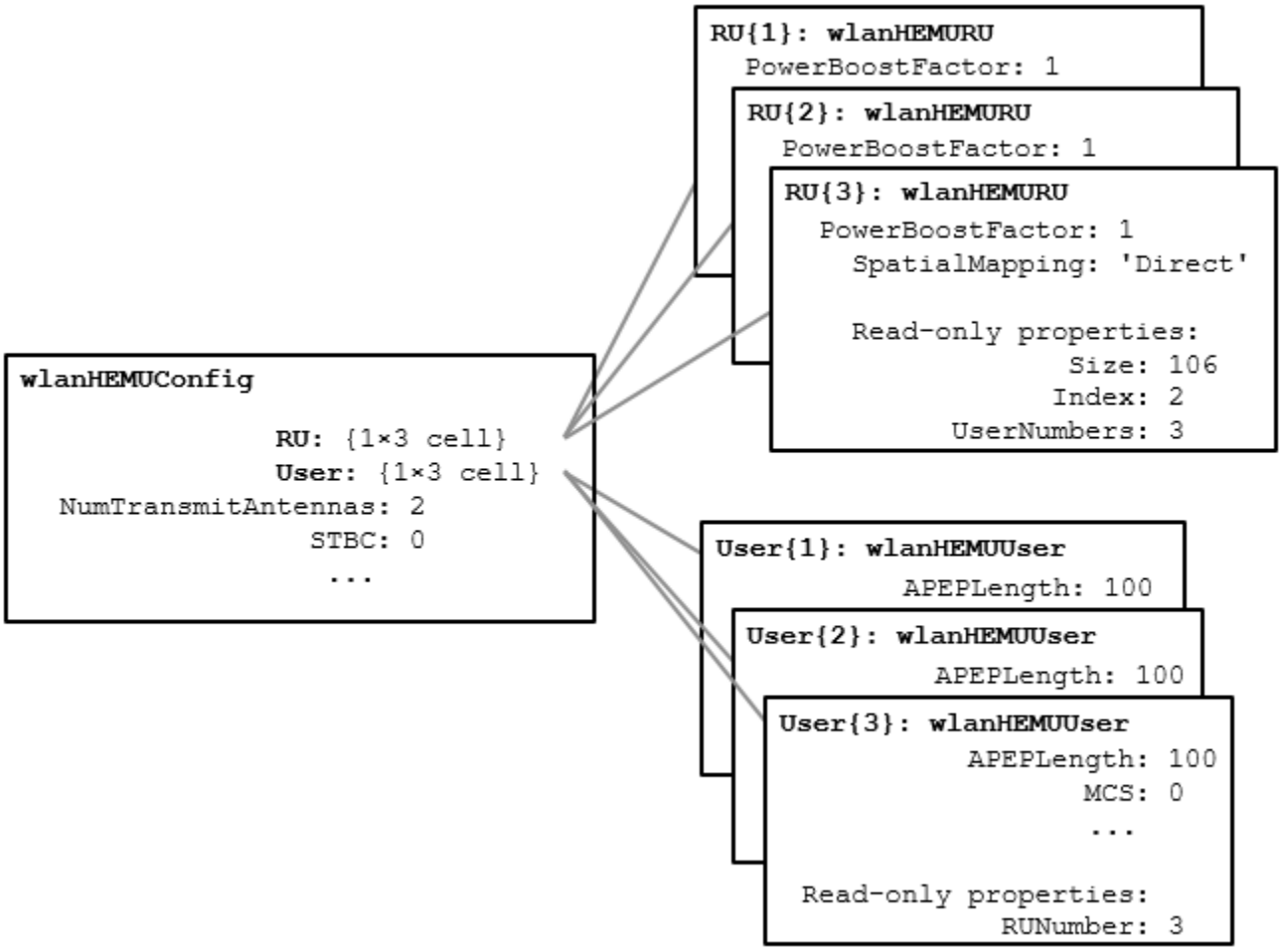
More About

PPDU

The physical layer (PHY) protocol data unit (PPDU) is the complete physical layer convergence procedure (PLCP) frame, including PLCP headers, MAC headers, the MAC data field, and the MAC and PLCP trailers.

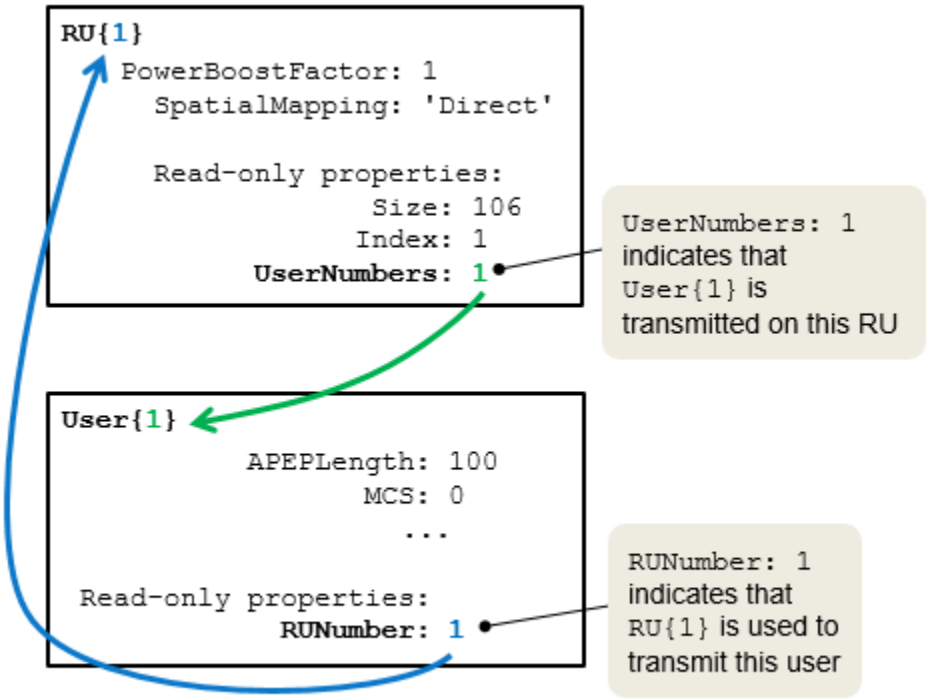
RU and User Cell Arrays

When you create a `wlanHEMUConfig` object, the properties are set based on the chosen `AllocationIndex` property and any name-value pairs you include. Upon creation of the object, `RU` and `User` cell arrays are configured. The `RU` cell array elements contain the configuration properties for each RU. The `User` cell array elements contain the configuration properties for each user.

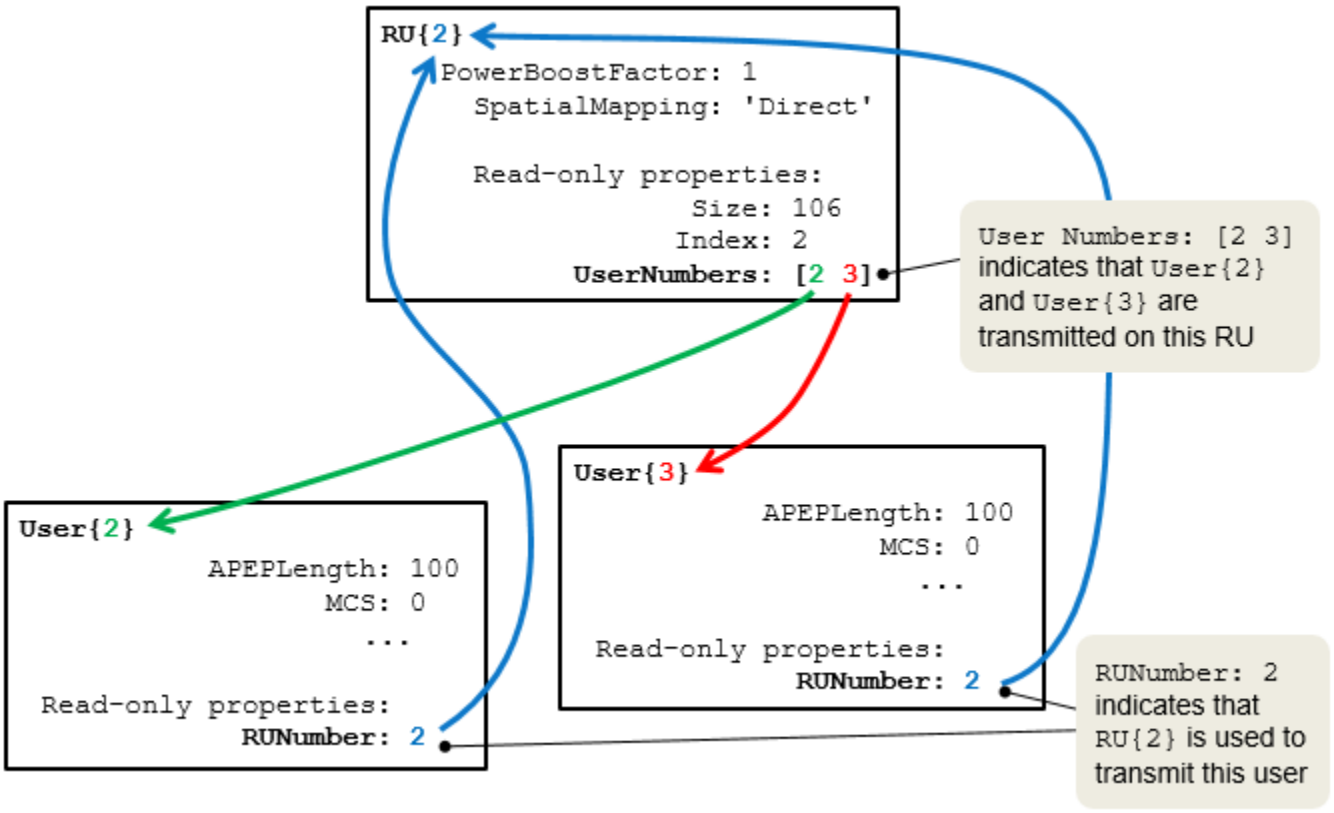


The `UserNumbers` subproperty of the `RU` property indicates which users are transmitted on each RU.

The `RUNumber` subproperty of the `User` property indicates which RUs are used to transmit data for each user.



As shown here, an RU can be assigned to multiple users.



After creating the object, you can modify some of the `RU` and `User` properties, but other `RU` and `User` properties are read-only. For more information about the elements of the `RU` cell array, see `wlanHEMURU`. For more information about the elements of the `User` cell array, see `wlanHEMUUser`.

Version History

Introduced in R2018b

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

[2] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

You must specify the data type of the `postFECPaddingBits` subproperty of the `user` property as `int8`.

See Also

Objects

`wlanDMGConfig` | `wlanHEMURU` | `wlanHEMUUser` | `wlanHERecoveryConfig` | `wlanHESUConfig` | `wlanHETBConfig` | `wlanHTConfig` | `wlanNonHTConfig` | `wlanS1GConfig` | `wlanVHTConfig`

Functions

`getPSDULength` | `packetFormat` | `ruInfo` | `showAllocation` | `transmitTime` | `wlanHELTFChannelEstimate` | `wlanWaveformGenerator`

Apps

WLAN Waveform Generator

Topics

“Allocation Index”

wlanHEMURU

Configure RU for HE MU transmission

Description

The `wlanHEMURU` object contains properties used to configure a WLAN high-efficiency (HE) resource unit (RU). When you create a `wlanHEMUConfig` object, the value to which you set its `AllocationIndex` property determines its `RU` property. The `RU` property is returned as a cell array of `wlanHEMURU` objects.

Creation

Syntax

```
cfgHEMU.RU = wlanHEMURU(Size,Index,UserNumbers)
cfgHEMU.RU = wlanHEMURU( ___ ,Name,Value)
```

Description

`cfgHEMU.RU = wlanHEMURU(Size,Index,UserNumbers)` creates an object that contains properties to configure an HE-format RU. The `Size` input is the RU size, `Index` is the RU index, and `UserNumbers` specifies the indices of users transmitted on the RU.

`cfgHEMU.RU = wlanHEMURU( ___ ,Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotation marks.

Properties

`PowerBoostFactor` — Power boost factor

1 (default) | scalar in the interval [0.5, 2]

Power boost factor, specified as a scalar in the interval [0.5, 2].

Data Types: `double`

`SpatialMapping` — Spatial mapping scheme

`'Direct'` (default) | `'Hadamard'` | `'Fourier'` | `'Custom'`

Spatial mapping scheme, specified as `'Direct'`, `'Hadamard'`, `'Fourier'`, or `'Custom'`.

Dependencies

The default value, `'Direct'`, applies when you set the `NumTransmitAntennas` property of the associated `wlanHEMUConfig` object to the sum of the number of space-time streams for all users assigned to the RU.

Data Types: `char` | `string`

`SpatialMappingMatrix` — Spatial mapping matrix

1 (default) | complex-valued scalar | complex-valued matrix | complex-valued 3-D array

Spatial mapping matrix, specified as one of these values:

- A complex-valued scalar. This value applies to all the subcarriers.
- A complex-valued matrix of size $N_{STS_{Total}}$ -by-N_T, where
  - $N_{STS_{Total}}$ is the sum of the number of space-time streams for all users assigned to the RU;
  - N_T is the number of transmit antennas.

  In this case, the spatial mapping matrix applies to all the subcarriers.
- A complex-valued 3-D array of size `Size`-by-$N_{STS_{Total}}$-by-N_T. The `ChannelBandwidth` property of the associated `wlanHEMUConfig` object determines the value of the `Size` property. In this case, each occupied subcarrier has its own spatial mapping matrix.

Use this property to rotate and scale the output vector of the constellation mapper. The spatial mapping matrix is used for beamforming and mixing space-time streams over the transmit antennas. The calling function normalizes the spatial mapping matrix for each subcarrier.

Example: `[0.5 0.3; 0.4 0.4; 0.5 0.8]` represents a spatial mapping matrix with three space-time streams and two transmit antennas.

Dependencies

This property applies only when you set the `SpatialMapping` property to `'Custom'`.

Data Types: `double`
Complex Number Support: Yes

`Beamforming` — Enable signaling of transmission with beamforming

`true` (default) | `false`

Enable signaling of a transmission with beamforming, specified as a logical value of `true` or `false`. To apply a beamforming steering matrix, set this property to `true`. The `SpatialMappingMatrix` property specifies the beamforming steering matrix.

Dependencies

This property applies only when you set the `SpatialMapping` property to `'Custom'`.

Data Types: `logical`

`Size` — Resource unit size

242 (default) | positive integer

Resource unit size, specified as `26`, `52`, `106`, `242`, `484`, `996`, or `1992`.

Note This property is read-only after the object is created.

Data Types: `double`

`Index` — Resource unit index

1 (default) | integer in the interval [1, 74]

Resource unit index, specified as an integer in the interval [1, 74]. Use this property to indicate the location of the RU within the channel.

Note This property is read-only after the object is created.

Example: In an 80-MHz transmission there are four possible 242-tone RUs, one in each 20-MHz subchannel. RU 242-1 (`Size` = 242, `Index` = 1) is the RU occupying the lowest absolute frequency within the 80MHz, and RU 242-4 (`Size` = 242, `Index` = 4) is the RU occupying the highest absolute frequency.

Data Types: `double`

`UserNumbers` — User index number

1 | integer | vector of integers

Indices of users transmitted on the RU, in one-based format, specified as an integer or a vector of integers. This property indexes the appropriate cell array elements of the User property within the associated `wlanHEMUConfig` object.

Data Types: `double`

Examples

Create HE MU Object Using Binary Allocation Indexing

Create an HE MU configuration object for a 40 MHz transmission with an allocation index of `11000000` for each 20 MHz subchannel. This configuration specifies two 242-tone RUs, each with one user.

```
cfgHEMU = wlanHEMUConfig(["11000000" "11000000"],'NumTransmitAntennas',2);
```

Configure the first RU and the first user.

```
cfgHEMU.RU{1}.SpatialMapping = 'Direct';
cfgHEMU.User{1}.APEPLength = 1e3;
cfgHEMU.User{1}.MCS = 2;
cfgHEMU.User{1}.NumSpaceTimeStreams = 2;
cfgHEMU.User{1}.ChannelCoding = 'LDPC';
cfgHEMU.User{1}.NominalPacketPadding = 16;
```

Configure the second RU and the second user.

```
cfgHEMU.RU{2}.SpatialMapping = 'Fourier';
cfgHEMU.User{2}.APEPLength = 500;
cfgHEMU.User{2}.MCS = 3;
cfgHEMU.User{2}.NumSpaceTimeStreams = 1;
cfgHEMU.User{2}.ChannelCoding = 'LDPC';
cfgHEMU.User{2}.NominalPacketPadding = 8;
```

Display the configuration object properties for both RUs and both users.

```
disp(cfgHEMU)
```

```
  wlanHEMUConfig with properties:
```

```
                    RU: {[1x1 wlanHEMURU]  [1x1 wlanHEMURU]}
                  User: {[1x1 wlanHEMUUser]  [1x1 wlanHEMUUser]}
    NumTransmitAntennas: 2
                  STBC: 0
         GuardInterval: 3.2000
             HELTFType: 4
               SIGBMCS: 0
               SIGBDCM: 0
       UplinkIndication: 0
                BSSColor: 0
           SpatialReuse: 0
           TXOPDuration: 127
            HighDoppler: 0

   Read-only properties:
       ChannelBandwidth: 'CBW40'
        AllocationIndex: [192 192]
```

```
cfgHEMU.RU{1:2}
```

```
ans =
  wlanHEMURU with properties:

    PowerBoostFactor: 1
      SpatialMapping: 'Direct'

   Read-only properties:
                Size: 242
               Index: 1
         UserNumbers: 1


ans =
  wlanHEMURU with properties:

    PowerBoostFactor: 1
      SpatialMapping: 'Fourier'

   Read-only properties:
                Size: 242
               Index: 2
         UserNumbers: 2
```

```
cfgHEMU.User{1:2}
```

```
ans =
  wlanHEMUUser with properties:

              APEPLength: 1000
                     MCS: 2
     NumSpaceTimeStreams: 2
                     DCM: 0
           ChannelCoding: 'LDPC'
                   STAID: 0
    NominalPacketPadding: 16
    PostFECPaddingSource: 'mt19937ar with seed'
      PostFECPaddingSeed: 1
```

```
   Read-only properties:
                RUNumber: 1


ans =
  wlanHEMUUser with properties:

              APEPLength: 500
                     MCS: 3
     NumSpaceTimeStreams: 1
                     DCM: 0
           ChannelCoding: 'LDPC'
                   STAID: 0
    NominalPacketPadding: 8
    PostFECPaddingSource: 'mt19937ar with seed'
      PostFECPaddingSeed: 2

   Read-only properties:
                RUNumber: 2
```

Version History

Introduced in R2018b

Extended Capabilities

C/C++ Code Generation

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

See Also

`wlanHEMUConfig` | `ruInfo`

wlanHEMUUser

Configure users for HE MU transmission

Description

The `wlanHEMUUser` object contains properties of a user within a WLAN high-efficiency (HE) resource unit (RU). When you create a `wlanHEMUConfig` object, the value to which you set its `AllocationIndex` property determines its `User` property. The `User` property is returned as a cell array of `wlanHEMUUser` objects.

Creation

Syntax

```
cfgHEMU.User = wlanHEMUUser(RUNumber)
cfgHEMU.User = wlanHEMUUser(RUNumber,Name,Value)
```

Description

`cfgHEMU.User = wlanHEMUUser(RUNumber)` creates an HE user configuration object for `RUNumber`, the input RU number.

`cfgHEMU.User = wlanHEMUUser(RUNumber,Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotation marks.

Properties

`APEPLength` — APEP length
`100` (default) | integer in the interval [1, 6451631]

Aggregated MPDU (A-MPDU) pre-end-of-frame (pre-EOF) padding (APEP) length, in bytes, specified as an integer in the interval [1, 6451631].

The object uses this property to determine the number of OFDM symbols in the data field. For more information, see [1].

Data Types: `double`

`MCS` — MCS used for transmission
`0` (default) | integer in the interval [0, 11]

Modulation and coding scheme (MCS) used for transmission, specified as a nonnegative integer in the interval [0, 11]. This table shows the modulation type and coding rate for each valid value of `MCS`:

MCS	Modulation	Dual Carrier Modulation	Coding Rate
0	Binary phase-shift keying (BPSK)	`0` or `1`	1/2
1	Quadrature phase-shift keying (QPSK)	`0` or `1`	1/2
2		Not applicable	3/4
3	16-point quadrature amplitude modulation (16-QAM)	`0` or `1`	1/2
4			3/4
5	64-QAM	Not applicable	2/3
6			3/4
7			5/6
8	256-QAM		3/4
9			5/6
10	1024-QAM		3/4
11			5/6

Data Types: `double`

`NumSpaceTimeStreams` — Number of space-time streams
`1` (default) | integer

Number of space-time streams in the transmission, specified as an integer in the interval [1, 8]. The maximum number of space-time streams for any user within a MU-MIMO RU is 4. The maximum value of the sum of the number of space-time streams over all users in an RU is 8. For information on these and other restrictions on the number of space-time streams, see Tables 18-1 and 27-30 of [1].

Data Types: `double`

`DCM` — DCM indicator
`false` or `0` (default) | `true` or `1`

Dual carrier modulation (DCM) indicator, specified as a logical value of `1` (`true`) or `0` (`false`). To indicate that DCM is used for the HE-Data field, set this property to `1` (`true`).

Dependencies

You can only set this property to `1` (`true`) when all of these conditions are satisfied:

- The `MCS` property is `0`, `1`, `3`, or `4`.
- The `STBC` property of the associated `wlanHEMUConfig` object is `0` (`false`).
- The `NumSpaceTimeStreams` property is less than or equal to 2.
- The `RU` property of the associated `wlanHEMUConfig` object defines a single-user RU.

Data Types: `logical`

`ChannelCoding` — FEC coding type
`'LDPC'` (default) | `'BCC'`

Forward-error-correction (FEC) coding type for the HE-Data field, specified as `'LDPC'` for low-density parity-check (LDPC) coding or `'BCC'` for binary convolutional coding (BCC).

Dependencies

You can only set this property to `'BCC'` when all of these conditions are satisfied:

- The `MCS` property is not `10` or `11`.
- The size of any RU is less than or equal to 242. Obtain the RU sizes by using the `ruInfo` object function with the associated `wlanHEMUConfig` object.
- The `NumSpaceTimeStreams` property is less than or equal to 4.

Data Types: `char` | `string`

`STAID` — STA identifier

`0` (default) | integer in the interval [0, 2047]

Station (STA) identifier, specified as an integer in the interval [0, 2047]. The value of this property specifies the station association identifier (AID) field as defined in section 26.11.1 of [1]. The 11 least significant bits (LSBs) of the AID field are used to address the STA. When you set this property to `2046`, the associated RU carries no data.

Data Types: `double`

`RUNumber` — RU number

`1` (default) | integer | vector of integers

RU number, specified as an integer or a vector of integers. This property indexes the appropriate cell array elements of the `RU` property within the associated `wlanHEMUConfig` object.

Note This property is read-only after the object is created.

Data Types: `double`

`NominalPacketPadding` — Nominal packet padding

`0` (default) | `8` | `16`

Nominal packet padding, in microseconds, specified as `0`, `8`, or `16`. The associated `wlanHEMUConfig` object uses this property and the pre-forward-error-correction (pre-FEC) padding factor to calculate the duration, T_{PE}, of the packet extension field. For more information about the packet extension field, see section 27.3.13 of [1].

This table shows the possible values of T_{PE} for different values of this property and a, a parameter defined by equation (27-83) or (27-84) of [1].

Value of a	Value of T_{PE} in Microseconds		
	`NominalPacketPadding` Set to `0`	`NominalPacketPadding` Set to `8`	`NominalPacketPadding` Set to `16`
1	0	0	4
2	0	0	8
3	0	4	12

Value of *a*	Value of T_{PE} in Microseconds		
	NominalPacketPadding Set to 0	NominalPacketPadding Set to 8	NominalPacketPadding Set to 16
4	0	8	16

Data Types: `double`

`PostFECPaddingSource` — Post-FEC padding bit source
`'mt19937ar with seed'` (default) | `'Global stream'` | `'User-defined'`

Post-FEC padding bit source used by the `wlanWaveformGenerator` function, specified as one of these values.

- `'mt19937ar with seed'` — Generate normally distributed random bits by using the mt19937ar algorithm with seed specified in the `PostFECPaddingSeed` property.
- `'Global stream'` — Generate normally distributed random bits by using the current global random number stream.
- `'User-defined'` — Use the bits specified in the `PostFECPaddingBits` property as the post-FEC padding bits.

Data Types: `char` | `string`

`PostFECPaddingSeed` — Post-FEC padding bit seed for mt19937ar algorithm
`73` (default) | nonnegative integer

Post-FEC padding bit seed for the mt19937ar algorithm, specified as a nonnegative integer. If this object is an element of the `User` property of the `wlanHEMUConfig` object, the default value of this property the user number, *i.e.*, the default value of `User{`k`}.PostFECPaddingSeed` is k for all integers k in the interval [1, N_{users}]. N_{users} is the number of users in the transmission.

Dependencies

To enable this property, set the `PostFECPaddingSource` property to `'mt19937ar with seed'`.

Data Types: `double`

`PostFECPaddingBits` — Post-FEC padding bits
`0` (default) | binary-valued column vector

Post-FEC padding bits, specified as a binary-valued scalar or column vector.

To generate a waveform, the `wlanWaveformGenerator` function requires n bits, where n depends on the specified configuration. To calculate n, use the `getNumPostFECPaddingBits` object function with the specified configuration object as the input argument and specify this property as a vector of length n. Alternatively, specify this input as a binary-valued scalar or column vector of arbitrary length. If the length of this property is less than n, the waveform generator loops the vector to create a vector of length n. If the length of this property is greater than n, the function uses only the first n entries as the post-FEC padding bits.

Note For C/C++ code generation, you must specify the data type of this property as `int8`.

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

Examples

Create Multiuser HE Configuration Object

Create a 20 MHz multiuser HE configuration object with the allocation index set to 0. An allocation index of 0 specifies nine 26-tone RUs in a 20 MHz channel.

```
cfgMU = wlanHEMUConfig(0);
for i=1:numel(cfgMU.User)
    % Set the APEPLength of each user
    cfgMU.User{i}.APEPLength = 100;
end
```

Display the configuration object properties for the fourth user.

```
cfgMU.User{4}
```

```
ans =
  wlanHEMUUser with properties:

              APEPLength: 100
                     MCS: 0
     NumSpaceTimeStreams: 1
                     DCM: 0
           ChannelCoding: 'LDPC'
                   STAID: 0
    NominalPacketPadding: 0
    PostFECPaddingSource: 'mt19937ar with seed'
      PostFECPaddingSeed: 4

   Read-only properties:
                RUNumber: 4
```

Create HE MU Object Using Binary Allocation Indexing

Create an HE MU configuration object for a 40 MHz transmission with an allocation index of `11000000` for each 20 MHz subchannel. This configuration specifies two 242-tone RUs, each with one user.

```
cfgHEMU = wlanHEMUConfig(["11000000" "11000000"],'NumTransmitAntennas',2);
```

Configure the first RU and the first user.

```
cfgHEMU.RU{1}.SpatialMapping = 'Direct';
cfgHEMU.User{1}.APEPLength = 1e3;
cfgHEMU.User{1}.MCS = 2;
cfgHEMU.User{1}.NumSpaceTimeStreams = 2;
cfgHEMU.User{1}.ChannelCoding = 'LDPC';
cfgHEMU.User{1}.NominalPacketPadding = 16;
```

Configure the second RU and the second user.

```
cfgHEMU.RU{2}.SpatialMapping = 'Fourier';
cfgHEMU.User{2}.APEPLength = 500;
```

```
cfgHEMU.User{2}.MCS = 3;
cfgHEMU.User{2}.NumSpaceTimeStreams = 1;
cfgHEMU.User{2}.ChannelCoding = 'LDPC';
cfgHEMU.User{2}.NominalPacketPadding = 8;
```

Display the configuration object properties for both RUs and both users.

```
disp(cfgHEMU)
```

```
  wlanHEMUConfig with properties:

                    RU: {[1x1 wlanHEMURU]  [1x1 wlanHEMURU]}
                  User: {[1x1 wlanHEMUUser]  [1x1 wlanHEMUUser]}
    NumTransmitAntennas: 2
                  STBC: 0
          GuardInterval: 3.2000
              HELTFType: 4
                SIGBMCS: 0
                SIGBDCM: 0
       UplinkIndication: 0
               BSSColor: 0
           SpatialReuse: 0
           TXOPDuration: 127
            HighDoppler: 0

   Read-only properties:
       ChannelBandwidth: 'CBW40'
        AllocationIndex: [192 192]
```

```
cfgHEMU.RU{1:2}
```

```
ans =
  wlanHEMURU with properties:

    PowerBoostFactor: 1
      SpatialMapping: 'Direct'

   Read-only properties:
                Size: 242
               Index: 1
         UserNumbers: 1


ans =
  wlanHEMURU with properties:

    PowerBoostFactor: 1
      SpatialMapping: 'Fourier'

   Read-only properties:
                Size: 242
               Index: 2
         UserNumbers: 2
```

```
cfgHEMU.User{1:2}
```

```
ans =
  wlanHEMUUser with properties:
```

```
              APEPLength: 1000
                     MCS: 2
     NumSpaceTimeStreams: 2
                     DCM: 0
           ChannelCoding: 'LDPC'
                   STAID: 0
    NominalPacketPadding: 16
    PostFECPaddingSource: 'mt19937ar with seed'
      PostFECPaddingSeed: 1

   Read-only properties:
                RUNumber: 1


ans =
  wlanHEMUUser with properties:

              APEPLength: 500
                     MCS: 3
     NumSpaceTimeStreams: 1
                     DCM: 0
           ChannelCoding: 'LDPC'
                   STAID: 0
    NominalPacketPadding: 8
    PostFECPaddingSource: 'mt19937ar with seed'
      PostFECPaddingSeed: 2

   Read-only properties:
                RUNumber: 2
```

Version History

Introduced in R2018b

References

[1] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

You must specify the data type of the `postFECPaddingBits` property as `int8`.

See Also

`wlanHEMUConfig` | `ruInfo`

wlanHERecoveryConfig

Store parameters recovered from HE transmission

Description

The `wlanHERecoveryConfig` is a high-efficiency (HE) recovery configuration object for HE single-user (HE SU), HE extended-range single-user (HE ER SU), and HE multi-user (HE MU) packet formats.

Creation

Syntax

```
cfg = wlanHERecoveryConfig
cfg = wlanHERecoveryConfig(Name,Value)
```

Description

`cfg = wlanHERecoveryConfig` creates an HE recovery configuration object, `cfg`, for HE SU, HE ER SU, and HE MU packet formats. The output `cfg` contains the parameters recovered from decoding the signaling fields of HE SU, HE ER SU, and HE MU transmissions as defined in [2].

On creation, the properties of a `wlanHERecoveryConfig` object are set to either `-1` or `'Unknown'` to indicate an unknown or undefined value or status. You can set and update the properties of this object by specifying the values as decoded signaling fields, as demonstrated in the “Recovery Procedure for an 802.11ax Packet” example. To update the properties relevant to the HE-SIG-A field, use the `interpretHESIGABits` object function. To update the properties relevant to the HE-SIG-B field, use the `interpretHESIGBCommonBits` and `interpretHESIGBUserBits` object functions.

`cfg = wlanHERecoveryConfig(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotation marks. For example, `wlanHERecoveryConfig('PacketFormat','HE-SU','GuardInterval',1.6)` creates an HE recovery configuration object for an HE SU packet with a guard interval of 1.6 microseconds.

Properties

PacketFormat — Recovered HE packet format

`'Unknown'` (default) | `'HE-SU'` | `'HE-EXT-SU'` | `'HE-MU'`

Recovered HE packet format, specified as `'Unknown'`, `'HE-SU'`, `'HE-EXT-SU'`, or `'HE-MU'`.

The length information in the L-SIG field and the four orthogonal frequency-division multiplexing (OFDM) symbols following the RL-SIG field determine the packet format. For more information, see “Recovery Procedure for an 802.11ax Packet”.

Data Types: `char` | `string`

ChannelBandwidth — Channel bandwidth of PPDU transmission
'Unknown' (default) | 'CBW20' | 'CBW40' | 'CBW80' | 'CBW160'

Channel bandwidth of "PPDU" on page 4-109 transmission, specified as one of these values:

- 'Unknown' – Unknown or undefined channel bandwidth
- 'CBW20' – Channel bandwidth of 20 MHz
- 'CBW40' – Channel bandwidth of 40 MHz
- 'CBW80' – Channel bandwidth of 80 MHz
- 'CBW160' – Channel bandwidth of 160 MHz

Data Types: char | string

LSIGLength — Length of L-SIG field
-1 (default) | integer in the interval [1, 4095]

Length of L-SIG field, specified as -1 to indicate an unknown or undefined packet length or as an integer in the interval [1, 4095]. You can set this property after decoding the L-SIG field by using the wlanLSIGBitRecover function.

Data Types: double

PreamblePuncturing — Preamble puncturing mode
'Unknown' (default) | 'None' | 'Mode-1' | 'Mode-2' | 'Mode-3' | 'Mode-4'

Preamble puncturing mode, specified as one of these values:

- 'Unknown' – Unknown or undefined preamble puncturing in the recovered waveform
- 'None' – No preamble puncturing in the recovered waveform
- 'Mode-1' – Preamble puncturing in the secondary 20-MHz subchannel. This value applies only when the ChannelBandwidth property is 'CBW80'.
- 'Mode-2' – Preamble puncturing in one of the 20-MHz subchannels of the secondary 40 MHz. This value applies only when the ChannelBandwidth property is 'CBW80'.
- 'Mode-3' – Preamble puncturing in the secondary 20-MHz subchannel. This value applies only when the ChannelBandwidth property is 'CBW160'.
- 'Mode-4' – Preamble puncturing in the primary 40-MHz subchannel. This value applies only when the ChannelBandwidth property is 'CBW160'.

Specifying PreamblePuncturing indicates a punctured 20-MHz or 40-MHz subchannel in the preamble. You can set this property by using the interpretHESIGABits object function after decoding the HE-SIG-A field.

Dependencies

This property applies only when the PacketFormat property is 'HE-MU'.

Data Types: char | string

SIGBCompression — HE-SIG-B compression indicator
-1 (default) | 1 (true) | 0 (false)

HE-SIG-B compression indicator, specified as `-1` to indicate an unknown or undefined state or as a logical value of `1` (`true`) or `0` (`false`). A value of `1` (`true`) indicates that the HE-SIG-B field is compressed. A value of `0` (`false`) indicates that the HE-SIG-B field is not compressed

You can set this property by using the `interpretHESIGABits` object functions after decoding the HE-SIG-A field.

Dependencies

This property applies only when the `PacketFormat` property is `'HE-MU'` and the `ChannelBandwidth` property is `'CBW20'`.

Data Types: `double` | `logical`

`SIGBMCS` — MCS of HE-SIG-B field

`-1` (default) | integer in the interval [–1, 5]

Modulation and coding scheme (MCS) of the HE-SIG-B field, specified as an integer in the interval [–1, 5]. A value of `-1` indicates an unknown or undefined MCS.

You can set this property by using the `interpretHESIGABits` object function after decoding the HE-SIG-A field.

Dependencies

This property applies only when the `PacketFormat` property is `'HE-MU'`.

Data Types: `double`

`SIGBDCM` — HE-SIG-B DCM indicator

`-1` (default) | `1` (`true`) | `0` (`false`)

HE-SIG-B dual-carrier modulation (DCM) indicator, specified as `-1` to indicate an unknown or undefined status or as a logical value of `1` (`true`) or `0` (`false`). A value of `1` (`true`) indicates that the HE-SIG-B field is modulated with DCM. A value of `0` (`false`) indicates that the HE-SIG-B field is not modulated with DCM.

You can set this property by using the `interpretHESIGABits` object function after decoding the HE-SIG-A field.

Dependencies

This property applies only when the `PacketFormat` property is `'HE-MU'` and when the `SIGBMCS` property is `0`, `1`, `3`, or `4`.

Data Types: `double` | `logical`

`NumSIGBSymbolsSignaled` — Number of HE-SIG-B symbols signaled in HE-SIG-A field

`-1` (default) | integer in the interval [1, 16]

Number of HE-SIG-B symbols signaled in the HE-SIG-A field, specified as `-1` to indicate an unknown or undefined number of symbols or as an integer in the interval [1, 16]. A value of `16` indicates that there are 16 or more HE-SIG-B symbols signaled.

You can set this property by using the `interpretHESIGABits` object function after decoding the HE-SIG-A field.

Dependencies

This property applies only when the `PacketFormat` property is `'HE-MU'` and when the `SIGBCompression` property is `0` (`false`).

Data Types: `double`

STBC — Space-time block coding indicator

`-1` (default) | `1` (`true`) | `0` (`false`)

Space-time block coding (STBC) indicator, specified as `-1` to indicate an unknown or undefined status or as a logical value of `1` (`true`) or `0` (`false`). A value of `1` (`true`) indicates that STBC is enabled in the PPDU data field transmission. A value of `0` (`false`) indicates that STBC is not enabled.

You can set this property by using the `interpretHESIGABits` object function after decoding the HE-SIG-A field.

Dependencies

This property can only be `1` (`true`) when the `NumSpaceTimeStreams` is `2` and when the `DCM` is `0`.

Data Types: `double` | `logical`

LDPCExtraSymbol — Extra OFDM symbol segment indicator

`-1` (default) | `1` (`true`) | `0` (`false`)

Extra orthogonal frequency-division multiplexing (OFDM) symbol segment indicator, specified as `-1` to indicate an unknown or undefined status or as a logical value of `1` (`true`) or `0` (`false`). A value of `1` (`true`) indicates the presence of an extra OFDM symbol segment for low-density parity-check (LDPC) coding. A value of `0` (`false`) indicates the absence of an extra OFDM symbol.

You can set this property by using the `interpretHESIGABits` object function after decoding the HE-SIG-A field.

Data Types: `double` | `logical`

PreFECPaddingFactor — Pre-FEC padding factor

`-1` (default) | integer

Pre-forward-error-correction (pre-FEC) padding factor, specified as `-1` to indicate an unknown or undefined padding factor or as a positive integer in the interval [1, 4].

You can set this property by using the `interpretHESIGABits` object function after decoding the HE-SIG-A field.

Data Types: `double`

PEDisambiguity — PE-disambiguity indicator

`-1` (default) | `1` (`true`) | `0` (`false`)

Packet extension (PE) disambiguity indicator, specified as `-1` to indicate an unknown or undefined PE disambiguity status or as a logical value of `1` (`true`) or `0` (`false`). For more information, see section 27.3.13 of [2].

You can set this property by using the `interpretHESIGABits` object function after decoding the HE-SIG-A field.

Data Types: `double` | `logical`

`GuardInterval` — Guard interval (cyclic prefix) duration
`-1` (default) | `0.8` | `1.6` | `3.2`

Guard interval (cyclic prefix) duration for the data field within a packet, in microseconds, specified as `-1` to indicate an unknown or undefined guard interval length, or as `0.8`, `1.6`, or `3.2`.

You can set this property by using the `interpretHESIGABits` object function after decoding the HE-SIG-A field.

Data Types: `double`

`HELTFType` — HE-LTF compression mode of recovered packet
`-1` (default) | `1` | `2` | `4`

HE long training field (HE-LTF) compression type of recovered packet, specified as one of these values:

- `-1` – Unknown or undefined HE-LTF compression mode
- `1` – A compression of HE-LTF duration
- `2` – A compression of twice the HE-LTF duration
- `4` – A compression of four times the HE-LTF duration

You can set this property by using the `interpretHESIGABits` object function after decoding the HE-SIG-A field.

Data Types: `double`

`NumHELTFSymbols` — Number of HE-LTF symbols
`-1` (default) | integer in the interval [1, 8]

Number of HE-LTF symbols, specified as `-1` or an integer in the interval [1, 8]. A value of `-1` indicates an unknown or undefined number of HE-LTF symbols.

You can set this property by using the `interpretHESIGABits` object function after decoding the HE-SIG-A field.

Data Types: `double`

`UplinkIndication` — Uplink transmission indicator
`-1` (default) | `1` (`true`) | `0` (`false`)

Uplink transmission indicator, specified as `-1` to indicate an unknown or undefined transmission direction or as a logical value of `1` (`true`) or `0` (`false`). A value of `1` (`true`) indicates that the PPDU is sent on an uplink transmission. A value of `0` (`false`) indicates that the PPDU is sent on a downlink transmission.

You can set this property by using the `interpretHESIGABits` object function after decoding the HE-SIG-A field.

Data Types: `double` | `logical`

`BSSColor` — BSS color identifier
`-1` (default) | integer in the interval [–1, 63]

Basic service set (BSS) color identifier, specified as an integer in the interval [–1, 63]. A value of `-1` indicates an unknown or undefined color. For more information, see section 26.11.4 of [2].

You can set this property by using the `interpretHESIGABits` object function after decoding the HE-SIG-A field.

Data Types: `double`

`SpatialReuse` — Spatial reuse indicator
–1 (default) | integer in the interval [–1, 15]

Spatial reuse indicator, specified as an integer in the interval [–1, 15]. A value of `-1` indicates an unknown or undefined status.

You can set this property by using the `interpretHESIGABits` object function after decoding the HE-SIG-A field.

Data Types: `double`

`TXOPDuration` — Duration information for TXOP protection
`-1` (default) | integer in the interval [–1, 127]

Duration for transmit opportunity (TXOP) protection, specified as an integer in the interval [–1, 127]. A value of `-1` indicates an unknown or undefined duration.

You can set this property by using the `interpretHESIGABits` object function after decoding the HE-SIG-A field.

Data Types: `double`

`HighDoppler` — High-Doppler mode indicator
`-1` (default) | `1` (`true`) | `0` (`false`)

High-Doppler mode indicator, specified as `-1` to indicate an unknown or undefined status or as a logical value of `1` (`true`) or `0` (`false`). A value of `1` (`true`) indicates high-Doppler mode in the HE-SIG-A field.

You can set this property by using `interpretHESIGABits` after decoding the HE-SIG-A field.

Data Types: `double` | `logical`

`MidamblePeriodicity` — Midamble periodicity of HE-Data field
`-1` (default) | `10` | `20`

Midamble periodicity of the HE-Data field, in OFDM symbols, specified as `-1` to indicate an unknown or undefined periodicity, or as `10` or `20`.

You can set this property by using the `interpretHESIGABits` object function after decoding the HE-SIG-A field.

Data Types: `double`

`AllocationIndex` — RU allocation indices for each 20-MHz subchannel
`-1` (default) | integer | vector of integers

Resource unit (RU) allocation indices for each 20-MHz subchannel, specified as an integer or a vector of integers in the interval [–1, 223]. A value of `-1` indicates an unknown or undefined allocation index. The recovered bits determine how many allocation indices are set, which determines the format of this property.

The allocation indices define bandwidth allocation by specifying the number of RUs, size of each RU, and number of users assigned to each RU. For more information, see “HE MU Transmission”.

For a full-bandwidth multiuser multiple-input/multiple output (MU-MIMO) waveform, you can set this property by using the `interpretHESIGABits` object function after decoding the HE-SIG-A field. For an OFDM waveform, you can set this property after decoding the HE-SIG-B field.

Dependencies

This property applies only when `PacketFormat` is `'HE-MU'`.

Data Types: `double`

`LowerCenter26ToneRU` — Enable lower center 26-tone RU allocation signaling

`-1` (default) | `1` (`true`) | `0` (`false`)

Indicate lower center 26-tone RU signaling, specified as `-1` to indicate an unknown status or as a logical value of `1` (`true`) or `0` (`false`). A value of `1` (`true`) indicates the presence of the lower frequency center 26-tone RU.

You can set this property after decoding the HE-SIG-B field.

Dependencies

This property applies only when the `PacketFormat` property is `'HE-MU'`, the `ChannelBandwidth` property is `'CBW80'` or `'CBW160'`, and a full bandwidth allocation is not used.

Data Types: `double` | `logical`

`UpperCenter26ToneRU` — Enable upper center 26-tone RU allocation signaling

`-1` (default) | `1` (`true`) | `0` (`false`)

Enable upper center 26-tone RU signaling, specified as `-1` to indicate an unknown status or as a logical value of `1` (`true`) or `0` (`false`). A value of `1` (`true`) indicates the presence of the upper frequency center 26-tone RU.

You can set this property after decoding the HE-SIG-B field.

Dependencies

This property applies only when the `PacketFormat` property is `'HE-MU'`, the `ChannelBandwidth` property is `'CBW160'`, and a full bandwidth allocation is not used.

Data Types: `double` | `logical`

`NumUsersPerContentChannel` — Number of users per SIGB content channel

`-1` (default) | positive integer

Number of users per SIGB content channel, specified as `-1` or a positive integer. A value of `-1` indicates an unknown or undefined number of users.

This property is applicable for both full-bandwidth MU-MIMO and OFDMA allocation. For a full-bandwidth MU-MIMO waveform, the distribution of users on the SIGB content channel is defined in section 27.3.11.8 of [2]. For an OFDMA waveform, the decoded HE-SIG-B common field determines the distribution of users.

For a full-bandwidth MU-MIMO waveform, you can set this property by using the `interpretHESIGABits` object function after decoding the HE-SIG-A field. For an OFDMA waveform, you can set this property after decoding the HE-SIG-B field.

Dependencies

This property applies only when the `PacketFormat` property is `'HE-MU'`.

Data Types: `double`

`RUTotalSpaceTimeStreams` — Total number of space-time streams in RU of interest

`-1` (default) | integer in the interval [1, 8]

Total number of space-time streams in RU of interest, specified as `-1` or as an integer in the interval [1, 8]. A value of `-1` indicates an unknown or undefined number of space-time streams.

You can set this property after decoding the HE-SIG-B field.

Dependencies

This property applies only when `PacketFormat` is `'HE-MU'`.

Data Types: `double`

`RUSize` — RU size for user of interest

`-1` (default) | `26` | `52` | `106` | `242` | `484` | `996` | `1992`

RU size for user of interest, specified as `-1`, `26`, `52`, `106`, `242`, `484`, `996`, or `1992`. A value of `-1` indicates an unknown or undefined RU size.

For an HE SU or HE ER SU packet, you can set this property by using the `interpretHESIGABits` object function after decoding the HE-SIG-A field. For an HE MU packet, you can set this property after decoding the HE-SIG-B field.

Data Types: `double`

`RUIndex` — RU index for user of interest

`-1` (default) | positive integer

RU index for user of interest, specified as `-1` or a positive integer. A value of `-1` indicates an unknown or undefined RU index. The RU index specifies the location of the RU within the channel. For example, an 80 MHz transmission contains four 242-tone RUs (one for each 20 MHz subchannel). RU number 242-1 (size 242, index 1) is the lowest absolute frequency within the 80 MHz channel. RU number 242-4 is the highest absolute frequency.

For an HE SU or HE ER SU packet, you can set this property by using the `interpretHESIGABits` object function after decoding the HE-SIG-A field. For an HE MU packet, you can set this property after decoding the HE-SIG-B field.

Data Types: `double`

`STAID` — STA identification number

–1 (default) | integer in the interval [–1, 2047]

Station (STA) identification number, specified as an integer in the interval [–1, 2047]. A value of `-1` indicates an unknown or undefined STA identification number.

The STA identification number is defined in section 26.11.1 of [2]. The 11 least significant bits (LSBs) of the association identifier (AID) field are used to address the STA. The associated RU carries no data when `STAID` is `2046`.

You can set this property after decoding the HE-SIG-B field.

Dependencies

This property applies only when the `PacketFormat` property is `'HE-MU'`.

Data Types: `double`

`MCS` — User-specific MCS

`-1` (default) | integer in the interval [–1, 11]

User-specific MCS, specified as an integer in the interval [–1, 11]. A value of `-1` indicates an unknown or undefined MCS. This table shows the modulation type and coding rate for each valid value of `MCS`:

`MCS`	Modulation	Coding Rate
`0`	Binary phase-shift keying (BPSK)	`1/2`
`1`	Quadrature phase-shift keying (QPSK)	`1/2`
`2`		`3/4`
`3`	16-point quadrature amplitude modulation (16-QAM)	`1/2`
`4`		`3/4`
`5`	64-QAM	`2/3`
`6`		`3/4`
`7`		`5/6`
`8`	256-QAM	`3/4`
`9`		`5/6`
`10`	1024-QAM	`3/4`
`11`		`5/6`

You can set this property after decoding the HE-SIG-B field.

Data Types: `double`

`DCM` — DCM indicator

`-1` (default) | `1` (`true`) | `0` (`false`)

DCM indicator, specified as `-1` to indicate an unknown or undefined status or as a logical value of `1` (`true`) or `0` (`false`). A value of `1` (`true`) indicates that DCM is used for the HE-Data field. A value of `0` (`false`) indicates that DCM is not used.

For an HE SU or HE ER SU packet, you can set this property by using the `interpretHESIGABits` object function after decoding the HE-SIG-A field. For an HE MU packet, you can set this property after decoding the HE-SIG-B field.

Dependencies

DCM can be used only when all of these conditions are satisfied:

- The

  `PacketFormat` property is `'HE-SU'`.
- The `NumSpaceTimeStreams` property is less than or equal to 2.
- `STBC` is `0` (`false`).
- The

  `MCS` property is `0`, `1`, `3`, or `4`.

Data Types: `double` | `logical`

`ChannelCoding` — FEC coding type
`'Unknown'` (default) | `'BCC'` | `'LDPC'`

Forward-error-correction (FEC) coding type for the HE-Data field, specified as one of these values:

- `'Unknown'` – Unknown or undefined channel coding type
- `'BCC'` – Binary convolutional coding (BCC)
- `'LDPC'`– LDPC coding

For an HE SU or HE ER SU packet, you can set this property by using the `interpretHESIGABits` object function after decoding the HE-SIG-A field. For an HE MU packet, you can set this property after decoding the HE-SIG-B field.

Data Types: `char` | `string`

`Beamforming` — Beamforming steering matrix indicator
`-1` (default) | `1` (`true`) | `0` (`false`)

Beamforming steering matrix indicator, specified as `-1` to indicate an unknown or undefined status or as a logical value of `1` (`true`) or `0` (`false`). A value of `1` (`true`) indicates that a beamforming steering matrix is applied to the received waveform.

For an HE SU waveform, you can set this property by using the `interpretHESIGABits` object function after decoding the HE-SIG-A field. For an HE MU waveform, you can set this property after decoding the HE-SIG-B field.

Data Types: `double` | `logical`

`PreHESpatialMapping` — Spatial mapping of pre-HE-STF portion
`-1` (default) | `1` (`true`) | `0` (`false`)

Spatial mapping of pre-HE-short-training-field (pre-HE-STF) portion of the PPDU, specified as `-1` to indicate an unknown or undefined status or as a logical value of `1` (`true`) or `0` (`false`). A value of `1` (`true`) indicates that the pre-HE-STF portion of the PPDU is spatially mapped in the same way as the first symbol of the HE-LTF on each tone.

For a full-bandwidth MU-MIMO waveform, you can set this property by using the `interpretHESIGABits` object function after decoding the HE-SIG-A field.

Dependencies

This property applies only when the `PacketFormat` property is `'HE-SU'`.

Data Types: `double` | `logical`

`NumSpaceTimeStreams` — Number of space-time streams for user of interest

`-1` (default) | integer in the interval [1, 8]

Number of space-time streams for user of interest, specified as `-1` or as an integer in the interval [1, 8]. A value of `-1` indicates an unknown or undefined number of space-time streams.

For an HE SU or HE ER SU packet, you can set this property by using the `interpretHESIGABits` object function after decoding the HE-SIG-A field. For an HE MU packet, you can set this property after decoding the HE-SIG-B field.

Data Types: `double`

`SpaceTimeStreamStartIndex` — Starting space-time stream index

`-1` (default) | integer

Starting space-time stream index, specified as an integer. A value of `-1` indicates an unknown or undefined index.

When multiple users are transmitting in the same RU in a MU-MIMO configuration, each user must transmit on different space-time streams. The `NumSpaceTimeStreams` and `SpaceTimeStreamStartIndex` properties determine the starting space-time stream for each user. You can set this property after decoding the HE-SIG-B field.

Dependencies

This property applies only when the `PacketFormat` property is `'HE-MU'`

Data Types: `double`

Object Functions

getPSDULength	Calculate HE or WUR PSDU length
getSIGBLength	Return information relevant to HE-SIG-B field length
interpretHESIGABits	Update transmission parameters with HE-SIG-A field bits
interpretHESIGBCommonBits	Update HE MU transmission parameters with HE-SIG-B common field bits
interpretHESIGBUserBits	Update HE MU transmission parameters with HE-SIG-B user field bits
packetFormat	WLAN packet format

Examples

Create HE-SU Recovery Configuration Object

Create a recovery configuration object with default property values.

```
cfg = wlanHERecoveryConfig;
```

Overwrite the default settings by specifying the channel bandwidth, packet format, and L-SIG length of the recovered signal. Display the resultant object.

```
cfg.ChannelBandwidth = 'CBW40';
cfg.PacketFormat = 'HE-SU';
cfg.LSIGLength = 100;
disp(cfg);
```

```
  wlanHERecoveryConfig with properties:

          PacketFormat: 'HE-SU'
      ChannelBandwidth: 'CBW40'
            LSIGLength: 100
                  STBC: -1
       LDPCExtraSymbol: -1
    PreFECPaddingFactor: -1
         PEDisambiguity: -1
          GuardInterval: -1
              HELTFType: -1
       NumHELTFSymbols: -1
      UplinkIndication: -1
               BSSColor: -1
           SpatialReuse: -1
           TXOPDuration: -1
            HighDoppler: -1
    MidamblePeriodicity: -1
                 RUSize: -1
                RUIndex: -1
                    MCS: -1
                    DCM: -1
          ChannelCoding: 'Unknown'
            Beamforming: -1
    PreHESpatialMapping: -1
    NumSpaceTimeStreams: -1
```

Create HE-MU Recovery Configuration Object

Create an HE recovery configuration object for the specified packet format, channel bandwidth, and L-SIG length.

```
cfg = wlanHERecoveryConfig('PacketFormat','HE-MU','ChannelBandwidth','CBW80','LSIGLength',100);
```

Display the recovery configuration object.

```
disp(cfg);
```

```
  wlanHERecoveryConfig with properties:

                  PacketFormat: 'HE-MU'
              ChannelBandwidth: 'CBW80'
                    LSIGLength: 100
             PreamblePuncturing: 'Unknown'
                SIGBCompression: -1
                       SIGBMCS: -1
                       SIGBDCM: -1
          NumSIGBSymbolsSignaled: -1
                          STBC: -1
                 LDPCExtraSymbol: -1
            PreFECPaddingFactor: -1
                 PEDisambiguity: -1
                  GuardInterval: -1
                      HELTFType: -1
               NumHELTFSymbols: -1
              UplinkIndication: -1
```

```
                    BSSColor: -1
                SpatialReuse: -1
                TXOPDuration: -1
                 HighDoppler: -1
          MidamblePeriodicity: -1
              AllocationIndex: -1
          LowerCenter26ToneRU: -1
    NumUsersPerContentChannel: -1
      RUTotalSpaceTimeStreams: -1
                       RUSize: -1
                      RUIndex: -1
                        STAID: -1
                          MCS: -1
                          DCM: -1
                ChannelCoding: 'Unknown'
                  Beamforming: -1
          NumSpaceTimeStreams: -1
SpaceTimeStreamStartingIndex: -1
```

Return HE-SIG-B Field Length Information

Create a WLAN HE-MU-format configuration object, specifying the allocation index.

```
cfgHEMU = wlanHEMUConfig(0);
```

Generate a WLAN waveform for the specified configuration and return the PPDU field indices.

```
waveform = wlanWaveformGenerator(1,cfgHEMU);
ind = wlanFieldIndices(cfgHEMU);
```

Decode the L-SIG field and obtain the OFDM information. This information is required to obtain the L-SIG length, which is used in the recovery configuration object.

```
lsig = waveform(ind.LSIG(1):ind.LSIG(2),:);
lsigDemod = wlanHEDemodulate(lsig,'L-SIG',cfgHEMU.ChannelBandwidth);
preHEInfo = wlanHEOFDMInfo('L-SIG',cfgHEMU.ChannelBandwidth);
```

Recover the L-SIG information bits and related information, making sure that the bits pass the parity check. For this example, we assume a noiseless channel. For more realistic results you can pass the waveform through an 802.11ax™ channel model by using the `wlanTGaxChannel` System object™ and work with the received waveform.

```
csi = ones(52,1);
[lsigBits,failCheck,lsigInfo] = wlanLSIGBitRecover(lsigDemod(preHEInfo.DataIndices,:,:),0,csi);
```

Decode the HE-SIG-A field and recover the HE-SIG-A information bits, ensuring that the bits pass the cyclic redundancy check (CRC).

```
siga = waveform(ind.HESIGA(1):ind.HESIGA(2),:);
sigaDemod = wlanHEDemodulate(siga,'HE-SIG-A',cfgHEMU.ChannelBandwidth);
preHEInfo = wlanHEOFDMInfo('HE-SIG-A',cfgHEMU.ChannelBandwidth);
[bits,failCRC] = wlanHESIGABitRecover(sigaDemod(preHEInfo.DataIndices,:,:),0,csi);
```

Create a WLAN recovery configuration object, specifying an HE-MU-format packet and the length of the L-SIG field.

```
cfg = wlanHERecoveryConfig('PacketFormat','HE-MU','LSIGLength',lsigInfo.Length);
```

Update the recovery configuration object with the recovered HE-SIG-A bits.

```
cfgUpdated = interpretHESIGABits(cfg,bits);
```

Return and display the HE-SIG-B information.

```
info = getSIGBLength(cfgUpdated);
disp(info);
```

```
    NumSIGBCommonFieldSamples: 80
                NumSIGBSymbols: 10
```

Recover HE-Data Field from HE SU Transmission

Recover bits from the HE-Data field of an HE SU transmission.

Configure an HE SU transmission by creating a configuration object with the specified modulation and coding scheme (MCS). Extract the channel bandwidth.

```
cfgHESU = wlanHESUConfig('MCS',0);
cbw = cfgHESU.ChannelBandwidth;       % Channel bandwidth of transmission
```

Create a sequence of data bits and generate an HE SU waveform.

```
bits = randi([0 1],8*getPSDULength(cfgHESU),1,'int8');
waveform = wlanWaveformGenerator(bits,cfgHESU);
```

Create a WLAN recovery configuration object, specifying the known channel bandwidth and packet format.

```
cfgRX = wlanHERecoveryConfig('ChannelBandwidth',cbw,'PacketFormat','HE-SU');
```

Recover the HE signaling fields by retrieving the field indices and performing the relevant demodulation operations.

```
ind = wlanFieldIndices(cfgRX);
heLSIGandRLSIG = waveform(ind.LSIG(1):ind.RLSIG(2),:);
symLSIG = wlanHEDemodulate(heLSIGandRLSIG,'L-SIG',cbw);
info = wlanHEOFDMInfo('L-SIG',cbw);
```

Merge the L-SIG and RL-SIG fields for diversity and obtain the data subcarriers.

```
symLSIG = mean(symLSIG,2);
lsig = symLSIG(info.DataIndices,:);
```

Decode the L-SIG field, assuming a noiseless channel, and use the length field to update the recovery object.

```
noiseVarEst = 0;
[~,~,lsigInfo] = wlanLSIGBitRecover(lsig,noiseVarEst);
cfgRX.LSIGLength = lsigInfo.Length;
```

Recover and demodulate the HE-SIG-A field, obtain the data subcarriers, and recover the HE-SIG-A bits.

```
heSIGA = waveform(ind.HESIGA(1):ind.HESIGA(2),:);
symSIGA = wlanHEDemodulate(heSIGA,'HE-SIG-A',cbw);
siga = symSIGA(info.DataIndices,:);
[sigaBits,failCRC] = wlanHESIGABitRecover(siga,0);
```

Update the recovery configuration object with the recovered HE-SIG-A bits and obtain the updated field indices.

```
cfgHE = interpretHESIGABits(cfgRX,sigaBits);
ind = wlanFieldIndices(cfgHE);
```

Retrieve and decode the HE-Data field.

```
heData = waveform(ind.HEData(1):ind.HEData(2),:);
symData = wlanHEDemodulate(heData,'HE-Data', ...
    cbw,cfgHE.GuardInterval,[cfgHE.RUSize cfgHE.RUIndex]);
infoData = wlanHEOFDMInfo('HE-Data',cbw,cfgHE.GuardInterval,[cfgHE.RUSize cfgHE.RUIndex]);
rxDataSym = symData(infoData.DataIndices,:,:);
dataBits = wlanHEDataBitRecover(rxDataSym,noiseVarEst,cfgHE);
```

Confirm that the recovered bits match the transmitted bits.

```
isequal(bits,dataBits)
```

```
ans = logical
   1
```

More About

PPDU

The physical layer (PHY) protocol data unit (PPDU) is the complete physical layer convergence procedure (PLCP) frame, including PLCP headers, MAC headers, the MAC data field, and the MAC and PLCP trailers.

Version History

Introduced in R2019a

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

[2] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`wlanHESUConfig` | `wlanHEMUConfig` | `wlanHETBConfig`

Functions

`wlanFieldIndices` | `wlanHEDataBitRecover` | `wlanHELTFChannelEstimate` | `wlanHESIGABitRecover` | `wlanHESIGBCommonBitRecover` | `wlanHESIGBUserBitRecover` | `wlanLSIGBitRecover` | `wlanSampleRate`

Topics

“HE MU Transmission”

wlanHESUConfig

Configure HE SU or HE ER SU transmission

Description

The `wlanHESUConfig` object is a configuration object for the WLAN HE single-user (HE SU) and HE extended-range single-user (HE ER SU) packet formats.

Creation

Syntax

```
cfgHESU = wlanHESUConfig
cfgHESU = wlanHESUConfig(Name,Value)
```

Description

`cfgHESU = wlanHESUConfig` creates a configuration object that initializes parameters for an IEEE 802.11 HE SU "PPDU" on page 4-121. For a detailed description of the HE WLAN formats, see [2].

`cfgHESU = wlanHESUConfig(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotation marks. For example, `wlanHESUConfig('GuardInterval',1.6)` specifies a 1.6 microsecond guard interval (cyclic prefix) duration.

At runtime, the calling function validates object settings for properties relevant to the operation of the function.

Properties

`ChannelBandwidth` — Channel bandwidth of PPDU transmission

`'CBW20'` (default) | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth of PPDU transmission, specified as one of these values:

- `'CBW20'` – Channel bandwidth of 20 MHz
- `'CBW40'` – Channel bandwidth of 40 MHz
- `'CBW80'` – Channel bandwidth of 80 MHz
- `'CBW160'` – Channel bandwidth of 160 MHz

Data Types: `char` | `string`

`ExtendedRange` — Enable HE ER SU format

`false` or `0` (default) | `true` or `1`

Enable HE ER SU format, specified as a numeric or logical `1` (`true`) or `0` (`false`). To create an HE ER SU format configuration object, set this property to `1` (`true`).

Dependencies

This property applies only when you set the `ChannelBandwidth` property to `'CBW20'`.

Data Types: `logical`

`Upper106ToneRU` — Enable higher frequency 106-tone RU

`false` or `0` (default) | `true` or `1`

Enable higher frequency 106-tone resource unit (RU), specified as a numeric or logical `1` (`true`) or `0` (`false`). To use only the higher frequency 106-tone RU within the primary 20 MHz channel bandwidth of an HE ER SU transmission, set this property to `1` (`true`).

Dependencies

This property applies only when you set the `ChannelBandwidth` property to `'CBW20'` and the `ExtendedRange` property to `1` (`true`).

Data Types: `logical`

`InactiveSubchannels` — Indicate inactive 20 MHz subchannels in HE sounding NDP

`false` or `0` (default) | logical vector

Indicate inactive 20 MHz subchannels in an HE sounding null data packet (NDP), specified as a numeric or logical `0` (`false`) or a logical vector with at least one element set to `0` (`false`). When specifying a vector, the elements correspond to subchannels in order of increasing absolute frequency. Each element indicates whether the corresponding 20 MHz subchannel is inactive. To indicate an inactive 20 MHz subchannel, set the corresponding element to `1` (`true`). If you set this property to `0` (`false`), the `wlanHESUConfig` object applies that value to all 20 MHz subchannels, indicating that all subchannels are active.

Example: `[0 0 0 1]` indicates an HE sounding NDP such that the subchannel with the highest absolute frequency value is inactive.

Dependencies

To enable this property, set the `ChannelBandwidth` property to either `'CBW80'` or `'CBW160'` and the `APEPLength` property to `0`.

Data Types: `logical`

`NumTransmitAntennas` — Number of transmit antennas

`1` (default) | positive integer

Number of transmit antennas, specified as a positive integer.

Data Types: `double`

`PreHECyclicShifts` — Cyclic shift values of additional transmit antennas

`-75` (default) | integer in the interval [–200, 0] | row vector

Cyclic shift values, in nanoseconds, of additional transmit antennas for the pre-HE fields of the waveform. The first eight antennas use the cyclic shift values specified in Table 21-10 of [1]. The remaining L antennas use the values you specify in this property, where L = `NumTransmitAntennas` – 8. Specify this property as one of these values:

- An integer in the interval [–200, 0] – the `wlanHESUConfig` object uses this cyclic shift value for each of the L additional antennas.

- A row vector of length L of integers in the interval [–200, 0] – the `wlanHESUConfig` object uses the kth element as the cyclic shift value for the (k + 8)th transmit antenna.

  Note If you specify this property as a row vector of length greater than L, the `wlanHESUConfig` object uses only the first L elements. For example, if you set the `NumTransmitAntennas` property to `16`, the `wlanHESUConfig` object uses only the first L = 16 – 8 = 8 elements of this vector.

Dependencies

To enable this property, set the `NumTransmitAntennas` property to a value greater than `8`.

Data Types: `double`

`NumSpaceTimeStreams` — Number of space-time streams

`1` (default) | integer in the interval [1, 8]

Number of space-time streams in the transmission, specified as an integer in the interval [1, 8].

Data Types: `double`

`SpatialMapping` — Spatial mapping scheme

`'Direct'` (default) | `'Hadamard'` | `'Fourier'` | `'Custom'`

Spatial mapping scheme, specified as `'Direct'`, `'Hadamard'`, `'Fourier'`, or `'Custom'`.

Dependencies

The default value, `'Direct'`, applies only when you set the `NumTransmitAntennas` and `NumSpaceTimeStreams` properties to the same value.

Data Types: `char` | `string`

`SpatialMappingMatrix` — Spatial mapping matrix

`1` (default) | complex-valued scalar | complex-valued matrix | complex-valued 3-D array

Spatial mapping matrix, specified as one of these values:

- A complex-valued scalar. This value applies to all the subcarriers.
- A complex-valued matrix of size N_{STS} -by-N_T, where:
  - N_{STS} is the number of space-time streams;
  - N_T is the number of transmit antennas.

  In this case, the spatial mapping matrix applies to all the subcarriers.
- A complex-valued 3-D array of size N_{ST}-by-N_{STS}-by-N_T, where N_{ST} is the number of occupied subcarriers. The `ChannelBandwidth` property determines the value of N_{ST}. In this case, each occupied subcarrier has its own spatial mapping matrix.

  This table shows the `ChannelBandwidth` setting and the corresponding N_{ST}:

`ChannelBandwidth`	N_{ST}
`'CBW20'`	242
`'CBW40'`	484

ChannelBandwidth	N_{ST}
'CBW80'	996
'CBW160'	1992

Use this property to rotate and scale the output vector of the constellation mapper. The spatial mapping matrix is used for beamforming and mixing space-time streams over the transmit antennas. The calling function normalizes the spatial mapping matrix for each subcarrier.

Example: `[0.5 0.3; 0.4 0.4; 0.5 0.8]` represents a spatial mapping matrix with three space-time streams and two transmit antennas.

Dependencies

This property applies only when you set the `SpatialMapping` property to `'Custom'`.

Data Types: `double`
Complex Number Support: Yes

`Beamforming` — Enable signaling of transmission with beamforming

`true` or `1` (default) | `false` or `0`

Enable signaling of a transmission with beamforming, specified as a numeric or logical `1` (`true`) or `0` (`false`). To apply a beamforming steering matrix, set this property to `1` (`true`). The `SpatialMappingMatrix` property specifies the beamforming steering matrix.

Dependencies

This property applies only when you set the `SpatialMapping` property to `'Custom'`.

Data Types: `logical`

`PreHESpatialMapping` — Enable spatial mapping of pre-HE-STF portion

`false` or `0` (default) | `true` or `1`

Enable spatial mapping of the pre-HE-short-training-field (pre-HE-STF) portion of the PPDU, specified as a numeric or logical `1` (`true`) or `0` (`false`). To spatially map the pre-HE-STF portion of the PPDU in the same way as the first symbol of the HE-LTF on each tone, set this property to `1` (`true`). To apply no spatial mapping to the pre-HE-STF portion of the PPDU, set this property to `0` (`false`).

Data Types: `logical`

`STBC` — Enable STBC

`false` or `0` (default) | `true` or `1`

Enable space-time block coding (STBC) of the PPDU data field, specified as a numeric or logical `1` (`true`) or `0` (`false`). STBC transmits multiple copies of the data stream across assigned antennas.

- When you set this property to `0` (`false`), STBC is not applied to the data field. The number of space-time streams is equal to the number of spatial streams.
- When you set this property to `1` (`true`), STBC is applied to the data field. The number of space-time streams is twice the number of spatial streams.

Dependencies

This property applies only when the `NumSpaceTimeStreams` property is `2` and the `DCM` property is `0` (`false`).

Data Types: `logical`

`MCS` — Modulation and coding scheme

`0` (default) | integer in the interval [0, 11]

Modulation and coding scheme (MCS) used in transmitting the current packet, specified as a nonnegative integer in the interval [0, 11]. This table shows the modulation type and coding rate for each valid value of `MCS`:

MCS	Modulation	Dual Carrier Modulation	Coding Rate
0	Binary phase-shift keying (BPSK)	`0` or `1`	1/2
1	Quadrature phase-shift keying (QPSK)	`0` or `1`	1/2
2		Not applicable	3/4
3	16-point quadrature amplitude modulation (16-QAM)	`0` or `1`	1/2
4			3/4
5	64-QAM	Not applicable	2/3
6			3/4
7			5/6
8	256-QAM		3/4
9			5/6
10	1024-QAM		3/4
11			5/6

Dependencies

- When you set the `ExtendedRange` to `1` (`true`), you can only set this property to `0`, `1`, or `2`.
- When you set the `Upper106ToneRU` to `1` (`true`), you can only set this property to `0`.

Data Types: `double`

`DCM` — DCM indicator

`false` or `0` (default) | `true` or `1`

Dual carrier modulation (DCM) indicator, specified as a numeric or logical `1` (`true`) or `0` (`false`). To indicate that DCM is used for the HE-Data field, set this property to `1` (`true`).

Dependencies

You can only set this property to `1` (`true`) when all of these conditions are satisfied:

- The `MCS` property is `0`, `1`, `3`, or `4`.
- The `STBC` property is `0` (`false`).
- The `NumSpaceTimeStreams` property is less than or equal to 2.

Data Types: `logical`

`ChannelCoding` — FEC coding type

`'LDPC'` (default) | `'BCC'`

Forward-error-correction (FEC) coding type for the HE-Data field, specified as `'LDPC'` for low-density parity-check (LDPC) coding or `'BCC'` for binary convolutional coding (BCC).

Dependencies

You can only set this property to `'BCC'` when all of these conditions are satisfied:

- The `MCS` property is not `10` or `11`.
- The size of any RU is less than or equal to 242. Obtain the RU sizes by using the `ruInfo` object function.
- The `NumSpaceTimeStreams` property is less than or equal to 4.

Data Types: `char` | `string`

`APEPLength` — APEP length

`100` (default) | integer in the interval [0, 6451631]

Aggregated MPDU (A-MPDU) pre-end-of-frame (pre-EOF) padding (APEP) length, in bytes, specified as an integer in the interval [0, 6451631]. Setting this property to `0` specifies transmission of an HE NDP.

The object uses this property to determine the number of OFDM symbols in the data field. For more information, see [2].

Data Types: `double`

`GuardInterval` — Guard interval (cyclic prefix) duration

`3.2` (default) | `1.6` | `0.8`

Guard interval (cyclic prefix) duration for the data field within a packet, in microseconds, specified as `3.2`, `1.6`, or `0.8`.

Data Types: `double`

`HELTFType` — HE-LTF compression mode of HE PPDU

`4` (default) | `2` | `1`

HE-LTF compression mode of HE PPDU, specified as `4`, `2`, or `1`. This property indicates the type of HE-LTF, where a value of `4`, `2`, or `1` corresponds to four times, two times, or one times HE-LTF duration compression mode, respectively. Table 27-1 of [2] enumerates the HE-LTF type as:

- 1 × HE-LTF – Duration of 3.2 μs with a guard interval duration of 0.8 μs or 1.6 μs
- 2 × HE-LTF – Duration of 6.4 μs with a guard interval duration of 0.8 μs or 1.6 μs
- 4 × HE-LTF – Duration of 12.8 μs with a guard interval duration of 0.8 μs or 3.2 μs

For more information on the HE-LTF, see section 27.3.11.10 of [2].

Data Types: `double`

`UplinkIndication` — Uplink transmission indicator

`false` or `0` (default) | `true` or `1`

Uplink transmission indicator, specified as a numeric or logical `1` (`true`) or `0` (`false`). To indicate that the PPDU is sent on a downlink transmission, set this property to `0` (`false`). To indicate that the PPDU is sent on an uplink transmission, set this property to `1` (`true`).

Data Types: `logical`

`BSSColor` — BSS color identifier
`0` (default) | integer in the interval [0, 63]

Basic service set (BSS) color identifier, specified as an integer in the interval [0, 63].

Data Types: `double`

`SpatialReuse` — Spatial reuse indicator
`0` (default) | integer in the interval [0, 15]

Spatial reuse indicator, specified as an integer in the interval [0, 15].

Data Types: `double`

`TXOPDuration` — Duration information for TXOP protection
`127` (default) | integer in the interval [0, 127]

Duration information for transmit opportunity (TXOP) protection, specified as an integer in the interval [0, 127]. Except for the first bit, which specifies TXOP length granularity, each bit of the TXOP subfield of the HE-SIG-A field is equal to `TXOPDuration`. Therefore a duration in microseconds must be converted according to the procedure set out in Table 27-18 of [2].

Data Types: `double`

`HighDoppler` — High-Doppler mode indicator
`false` or `0` (default) | `true` or `1`

High-Doppler mode indicator, specified as a numeric or logical `1` (`true`) or `0` (`false`). To indicate high-Doppler mode in the HE-SIG-A field, set this property to `1` (`true`).

Dependencies

The `1` (`true`) value of this property is valid only when the `NumSpaceTimeStreams` property is less than or equal to 4 for any RU.

Data Types: `logical`

`MidamblePeriodicity` — Midamble periodicity of HE-Data field
`10` (default) | `20`

Midamble periodicity of the HE-Data field, in number of OFDM symbols, specified as `10` or `20`.

Dependencies

This property applies only when the `HighDoppler` property is `1` (`true`).

Data Types: `double`

`NominalPacketPadding` — Nominal packet padding
`0` (default) | `8` | `16`

Nominal packet padding, in microseconds, specified as `0`, `8`, or `16`. The `wlanHESUConfig` object uses this property and *a*, the pre-forward-error-correction (pre-FEC) padding factor to calculate the

duration, T_{PE}, of the packet extension (PE) field. For more information about the packet extension field, see section 27.3.12 of [2].

This table shows the possible values of T_{PE} for different values of this property and a, which is defined by equation (27-83) or (27-84) of [2].

Value of a	Value of T_{PE} in Microseconds		
	`NominalPacketPadding` Set to `0`	`NominalPacketPadding` Set to `8`	`NominalPacketPadding` Set to `16`
1	0	0	4
2	0	0	8
3	0	4	12
4	0	8	16

Dependencies

To enable this property, set the `APEPLength` property to an integer in the interval [1, 6,500,531]. The duration of the PE field for an NDP, regardless of the nominal packet padding, is 4 microseconds.

Data Types: `double`

`PostFECPaddingSource` — Post-FEC padding bit source

`'mt19937ar with seed'` (default) | `'Global stream'` | `'User-defined'`

Post-FEC padding bit source used by the `wlanWaveformGenerator` function, specified as one of these values.

- `'mt19937ar with seed'` — Generate normally distributed random bits by using the mt19937ar algorithm with seed specified in the `PostFECPaddingSeed` property.
- `'Global stream'` — Generate normally distributed random bits by using the current global random number stream.
- `'User-defined'` — Use the bits specified in the `PostFECPaddingBits` property as the post-FEC padding bits.

Data Types: `char` | `string`

`PostFECPaddingSeed` — Post-FEC padding bit seed for mt19937ar algorithm

`73` (default) | nonnegative integer

Post-FEC padding bit seed for the mt19937ar algorithm, specified as a nonnegative integer.

Dependencies

To enable this property, set the `PostFECPaddingSource` property to `'mt19937ar with seed'`.

Data Types: `double`

`PostFECPaddingBits` — Post-FEC padding bits

`0` (default) | binary-valued column vector

Post-FEC padding bits, specified as a binary-valued scalar or column vector.

To generate a waveform, the `wlanWaveformGenerator` function requires n bits, where n depends on the specified configuration. To calculate n, use the `getNumPostFECPaddingBits` object function

with the specified configuration object as the input argument and specify this property as a vector of length *n*. Alternatively, specify this input as a binary-valued scalar or column vector of arbitrary length. If the length of this property is less than *n*, the waveform generator loops the vector to create a vector of length *n*. If the length of this property is greater than *n*, the function uses only the first *n* entries as the post-FEC padding bits.

Note For C/C++ code generation, you must specify the data type of this property as `int8`.

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

Object Functions

getNumPostFECPaddingBits	Calculate required number of post-FEC padding bits
getPSDULength	Calculate HE or WUR PSDU length
packetFormat	WLAN packet format
ruInfo	Resource unit allocation information
showAllocation	Resource unit (RU) allocation
transmitTime	Packet transmission time

Examples

Create HE SU Configuration Object

Create an HE SU configuration object for a 40-MHz transmission.

```
cfgHE = wlanHESUConfig;
cfgHE.ChannelBandwidth = 'CBW40'
```

```
cfgHE =
  wlanHESUConfig with properties:

         ChannelBandwidth: 'CBW40'
     NumTransmitAntennas: 1
     NumSpaceTimeStreams: 1
          SpatialMapping: 'Direct'
     PreHESpatialMapping: 0
                    STBC: 0
                     MCS: 0
                     DCM: 0
           ChannelCoding: 'LDPC'
              APEPLength: 100
           GuardInterval: 3.2000
               HELTFType: 4
        UplinkIndication: 0
                 BSSColor: 0
            SpatialReuse: 0
            TXOPDuration: 127
             HighDoppler: 0
    NominalPacketPadding: 0
    PostFECPaddingSource: 'mt19937ar with seed'
      PostFECPaddingSeed: 73
```

Create HE ER SU Configuration Object

Create an HE ER SE configuration object for a 20-MHz transmission.

```
cfgHE = wlanHESUConfig('ExtendedRange',true)
```

```
cfgHE =
  wlanHESUConfig with properties:

         ChannelBandwidth: 'CBW20'
            ExtendedRange: 1
          Upper106ToneRU: 0
      NumTransmitAntennas: 1
      NumSpaceTimeStreams: 1
           SpatialMapping: 'Direct'
      PreHESpatialMapping: 0
                     STBC: 0
                      MCS: 0
                      DCM: 0
            ChannelCoding: 'LDPC'
               APEPLength: 100
            GuardInterval: 3.2000
                HELTFType: 4
         UplinkIndication: 0
                  BSSColor: 0
             SpatialReuse: 0
             TXOPDuration: 127
              HighDoppler: 0
     NominalPacketPadding: 0
     PostFECPaddingSource: 'mt19937ar with seed'
       PostFECPaddingSeed: 73
```

Create HE SU Configuration Object for Packet Extension

Create an HE SU configuration object, specifying a channel bandwidth of 40 MHz and nominal packet padding value of eight microseconds.

```
cfgHESU = wlanHESUConfig('ChannelBandwidth','CBW40','NominalPacketPadding',8)
```

```
cfgHESU =
  wlanHESUConfig with properties:

         ChannelBandwidth: 'CBW40'
      NumTransmitAntennas: 1
      NumSpaceTimeStreams: 1
           SpatialMapping: 'Direct'
      PreHESpatialMapping: 0
                     STBC: 0
                      MCS: 0
                      DCM: 0
            ChannelCoding: 'LDPC'
               APEPLength: 100
            GuardInterval: 3.2000
```

```
              HELTFType: 4
        UplinkIndication: 0
                 BSSColor: 0
            SpatialReuse: 0
            TXOPDuration: 127
             HighDoppler: 0
    NominalPacketPadding: 8
    PostFECPaddingSource: 'mt19937ar with seed'
      PostFECPaddingSeed: 73
```

Update the configuration object to specify NDP transmission mode. Because the duration of the PE field for an NDP is always four microseconds, the `NominalPacketPadding` property does not apply.

```
cfgHESU.APEPLength = 0
```

```
cfgHESU =
  wlanHESUConfig with properties:

       ChannelBandwidth: 'CBW40'
    NumTransmitAntennas: 1
    NumSpaceTimeStreams: 1
         SpatialMapping: 'Direct'
    PreHESpatialMapping: 0
                   STBC: 0
                    MCS: 0
                    DCM: 0
          ChannelCoding: 'LDPC'
             APEPLength: 0
          GuardInterval: 3.2000
              HELTFType: 4
       UplinkIndication: 0
                BSSColor: 0
           SpatialReuse: 0
           TXOPDuration: 127
            HighDoppler: 0
```

More About

PPDU

The physical layer (PHY) protocol data unit (PPDU) is the complete physical layer convergence procedure (PLCP) frame, including PLCP headers, MAC headers, the MAC data field, and the MAC and PLCP trailers.

Version History

Introduced in R2018b

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for

Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

[2] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

You must specify the data type of the `postFECPaddingBits` property as `int8`.

See Also

Objects

`wlanDMGConfig` | `wlanHEMUConfig` | `wlanHETBConfig` | `wlanHERecoveryConfig` | `wlanHTConfig` | `wlanNonHTConfig` | `wlanS1GConfig` | `wlanVHTConfig`

Functions

`getPSDULength` | `ruInfo` | `packetFormat` | `showAllocation` | `transmitTime` | `wlanHEDemodulate` | `wlanHELTFChannelEstimate` | `wlanWaveformGenerator`

Apps

WLAN Waveform Generator

wlanHETBConfig

Configure HE TB transmission

Description

The `wlanHETBConfig` object is a configuration object for the WLAN HE trigger-based (HE TB) packet format.

Creation

Syntax

```
cfgHETB = wlanHETBConfig
cfgHETB = wlanHETBConfig(Name,Value)
```

Description

`cfgHETB = wlanHETBConfig` creates a configuration object that initializes parameters for an IEEE 802.11 HE TB uplink PPDU or HE TB feedback null data packet (NDP). For a detailed description of the HE WLAN formats, see [2].

`cfgHETB = wlanHETBConfig(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in single quotes. For example, `wlanHETBConfig('ChannelBandwidth','CBW80')` specifies a channel bandwidth of 80 MHz.

At runtime, the calling function validates object settings for properties relevant to the operation of the function.

Properties

HE TB PPDU and HE TB Feedback NDP Properties

Properties in this section apply to all HE TB transmission configurations.

FeedbackNDP — Generate HE TB feedback NDP
`false` or `0` (default) | `true` or `1`

Generate an HE TB feedback NDP, specified as one of these values.

- `0` (`false`) — Generate an HE TB PPDU.
- `1` (`true`) — Generate an HE TB feedback NDP.

The HE TB feedback NDP carries the NDP feedback report information described in “NDP Feedback Report Procedure” on page 4-137.

For more information about the HE TB feedback NDP, see section 27.3.17 of [2].

Note To generate a valid `wlanHETBConfig` object for an HE TB feedback NDP, use the `getNDPFeedbackConfiguration` object function.

Data Types: `logical`

`ChannelBandwidth` — Channel bandwidth of PPDU transmission

`'CBW20'` (default) | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth of PPDU transmission, specified as one of these values.

- `'CBW20'` – Channel bandwidth of 20 MHz
- `'CBW40'` – Channel bandwidth of 40 MHz
- `'CBW80'` – Channel bandwidth of 80 MHz
- `'CBW160'` – Channel bandwidth of 160 MHz

Data Types: `char` | `string`

`PreHEPowerScalingFactor` — Power scaling factor for pre-HE PPDU fields

1 (default) | scalar in the interval [1/√2, 1]

Power scaling factor of pre-HE PPDU fields, specified as a scalar in the interval [1/√2, 1].

Data Types: `double`

`NumTransmitAntennas` — Number of transmit antennas

1 (default) | positive integer

Number of transmit antennas, specified as a positive integer.

Data Types: `double`

`PreHECyclicShifts` — Cyclic shift values of additional transmit antennas

`-75` (default) | integer in the interval [-200, 0] | row vector

Cyclic shift values, in nanoseconds, of additional transmit antennas for the pre-HE fields of the waveform. The first eight antennas use the cyclic shift values specified in Table 21-10 of [1]. The remaining L antennas use the values you specify in this property, where L = `NumTransmitAntennas` – 8. Specify this property as one of these values:

- An integer in the interval [-200, 0] – the `wlanHETBConfig` object uses this cyclic shift value for each of the L additional antennas.
- A row vector of length L – the `wlanHETBConfig` object uses the kth entry as the cyclic shift value for the (k + 8)th transmit antenna.

Note If you specify this property as a row vector of length $N > L$, the `wlanHETBConfig` object uses only the first L entries. For example, if you set the `NumTransmitAntennas` property to `16`, the `wlanHETBConfig` object uses only the first L = 16 – 8 = 8 entries of this property.

Dependencies

To enable this property, set the `NumTransmitAntennas` property to a value greater than `8`.

Data Types: `double`

`NumSpaceTimeStreams` — Number of space-time streams

1 (default) | integer in the interval [1, 8]

Number of space-time streams in the transmission, specified as an integer in the interval [1, 8].

Data Types: `double`

`StartingSpaceTimeStream` — Starting space-time stream index

1 (default) | integer in the interval [1, 8]

Starting space-time stream index, in one-based form, specified as an integer in the interval [1, 8]. In a multi-user multiple-input multiple-output (MU-MIMO) configuration with multiple users on the same RU, each user must transmit on a distinct space-time stream. In this case, you must set this property and the `NumSpaceTimeStreams` property to ensure that each space-time stream transmits at most one user.

Data Types: `double`

`GuardInterval` — Guard interval (cyclic prefix) duration

`3.2` (default) | `1.6`

Guard interval (cyclic prefix) duration for the data field within a packet, in microseconds, specified as `3.2` or `1.6`.

Data Types: `double`

`HELTFType` — HE-LTF compression mode of HE PPDU

4 (default) | 2 | 1

HE-LTF compression mode of HE PPDU, specified as 4, 2, or 1. This property indicates the type of HE-LTF, where a value of 4, 2, or 1 corresponds to four, two, or one multiple of the HE-LTF duration compression mode, respectively. Table 27-1 of [2] enumerates the HE-LTF type options as:

- 1 × HE-LTF – Duration of 3.2 microseconds with a guard interval duration of 0.8 or 1.6 microseconds
- 2 × HE-LTF – Duration of 6.4 microseconds with a guard interval duration of 0.8 or 1.6 microseconds
- 4 × HE-LTF – Duration of 12.8 microseconds with a guard interval duration of 0.8 or 3.2 microseconds

For more information on the HE-LTF, see section 27.3.11.10 of [2].

Data Types: `double`

`NumHELTFSymbols` — Number of HE-LTF symbols in PPDU

1 (default) | 2 | 4 | 6 | 8

Number of HE-LTF symbols in the PPDU, specified as 1, 2, 4, 6, or 8.

Dependencies

- If you set the `TriggerMethod` property to `'TRS'`, then you must set this property to 1.
- If you set the `HighDoppler` property to 1 (`true`), then you must set this property to 1, 2, or 4.

Data Types: `double`

`BSSColor` — BSS color identifier
`0` (default) | integer in the interval [0, 63]

Basic service set (BSS) color identifier, specified as an integer in the interval [0, 63].

Data Types: `double`

`SpatialReuse1` — Value of Spatial Reuse 1 subfield
`15` (default) | integer in the interval [0, 15]

Value of Spatial Reuse 1 subfield in the HE-SIG-A field, specified as an integer in the interval [0, 15]. For more information, see Table 27-22 of [2].

Data Types: `double`

`SpatialReuse2` — Value of Spatial Reuse 2 subfield
`15` (default) | integer in the interval [0, 15]

Value of Spatial Reuse 2 subfield in the HE-SIG-A field, specified as an integer in the interval [0, 15]. For more information, see Table 27-22 of [2].

Data Types: `double`

`SpatialReuse3` — Value of Spatial Reuse 3 subfield
`15` (default) | integer in the interval [0, 15]

Value of Spatial Reuse 3 subfield in the HE-SIG-A field, specified as an integer in the interval [0, 15]. For more information, see Table 27-22 of [2].

Data Types: `double`

`SpatialReuse4` — Value of Spatial Reuse 4 subfield
`15` (default) | integer in the interval [0, 15]

Value of Spatial Reuse 4 subfield in the HE-SIG-A field, specified as an integer in the interval [0, 15]. For more information, see Table 27-22 of [2].

Data Types: `double`

`TXOPDuration` — Duration information for TXOP protection
`127` (default) | integer in the interval [0, 127]

Duration information for transmit opportunity (TXOP) protection, specified as an integer in the interval [0, 127]. Except for the first bit, which specifies TXOP length granularity, each bit of the TXOP subfield in the HE-SIG-A field is equal to the value of this property. Therefore, a duration in microseconds must be converted according to the procedure set out in Table 27-22 of [2].

Data Types: `double`

HE TB PPDU Properties

Properties in this section apply only when the `FeedbackNDP` property is `0` (`false`). Use these properties to configure an HE TB PPDU in response to a trigger frame or a frame that contains a triggered response scheduling (TRS) Control subfield.

`TriggerMethod` — Triggering frame type
`'TriggerFrame'` (default) | `'TRS'`

Triggering frame type, specified as one of these values.

- `'TriggerFrame'` – Generate an HE TB PPDU in response to a trigger frame. For more information about trigger frames, see section 9.3.1.22 of [2].
- `'TRS'` – Generate an HE TB PPDU in response to a frame that contains a TRS Control subfield. For more information, see section 9.2.4.6a.1 of [2].

Note To generate a valid `wlanHETBConfig` object for a transmission in response to a frame containing a TRS Control subfield, use the `getTRSConfiguration` object function.

Data Types: `char` | `string`

RUSize — RU size

`242` (default) | `26` | `52` | `106` | `484` | `996` | `1992`

Resource unit (RU) size specified as `26`, `52`, `106`, `242`, `484`, `996`, or `1992`.

Data Types: `double`

RUIndex — RU index for subcarrier allocation

`1` (default) | integer in the interval [1, 74]

RU index for subcarrier allocation, specified as an integer in the interval [1, 74]. The RU index specifies the location of the RU within the channel. For example, an 80 MHz transmission contains four 242-tone RUs (one for each 20 MHz subchannel). RU number 242-1 (size 242, index 1) is the lowest absolute frequency within the 80 MHz channel. Similarly, RU number 242-4 is the highest absolute frequency.

Data Types: `double`

SpatialMapping — Spatial mapping scheme

`'Direct'` (default) | `'Hadamard'` | `'Fourier'` | `'Custom'`

Spatial mapping scheme, specified as `'Direct'`, `'Hadamard'`, `'Fourier'`, or `'Custom'`.

Dependencies

The default value, `'Direct'`, applies only when you set the `NumTransmitAntennas` and `NumSpaceTimeStreams` properties to the same value.

Data Types: `char` | `string`

SpatialMappingMatrix — Spatial mapping matrix

`1` (default) | complex-valued scalar | complex-valued matrix | complex-valued 3-D array

Spatial mapping matrix, specified as one of these values.

- A complex-valued scalar – this value applies to all the subcarriers.
- A complex-valued matrix of size N_{STS} -by-N_T – N_{STS} is the number of space-time streams, and N_T is the number of transmit antennas. In this case, the spatial mapping matrix applies to all the subcarriers.
- A complex-valued 3-D array of size N_{ST}-by-N_{STS}-by-N_T – N_{ST} is the number of occupied subcarriers. The `ChannelBandwidth` property determines the value of N_{ST}. In this case, each occupied subcarrier has its own spatial mapping matrix.

This table shows the value of the `ChannelBandwidth` property and the corresponding value of N_{ST}.

Value of `ChannelBandwidth`	Value of N_{ST}
`'CBW20'`	242
`'CBW40'`	484
`'CBW80'`	996
`'CBW160'`	1992

Use this property to rotate and scale the output vector of the constellation mapper. The spatial mapping matrix is used for beamforming and mixing space-time streams over the transmit antennas. The calling function normalizes the spatial mapping matrix for each subcarrier.

Example: `[0.5 0.3; 0.4 0.4; 0.5 0.8]` represents a spatial mapping matrix with three space-time streams and two transmit antennas.

Dependencies

To enable this property, set the `SpatialMapping` property to `'Custom'`.

Data Types: `double`
Complex Number Support: Yes

STBC — Enable STBC

`false` or `0` (default) | `true` or `1`

Enable space-time block coding (STBC) of the HE-Data field, specified as `1` (`true`) or `0` (`false`). STBC transmits multiple copies of the data stream across assigned antennas.

- When you set this property to `0` (`false`), STBC is not applied to the HE-Data field. The number of space-time streams is equal to the number of spatial streams.
- When you set this property to `1` (`true`), STBC is applied to the HE-Data field. The number of space-time streams is twice the number of spatial streams.

Dependencies

To enable this property, set the `NumSpaceTimeStreams` property to `2` and the `DCM` property to `0` (`false`).

Data Types: `logical`

MCS — Modulation and coding scheme

`0` (default) | integer in the interval [0, 11]

Modulation and coding scheme (MCS) used in transmitting the current packet, specified as an integer in the interval [0, 11]. This table shows the modulation type and coding rate for each valid value of this property.

Value of `MCS`	Modulation Type	Dual Carrier Modulation	Coding Rate
0	Binary phase-shift keying (BPSK)	`0` or `1`	1/2

Value of MCS	Modulation Type	Dual Carrier Modulation	Coding Rate
1	Quadrature phase-shift keying (QPSK)		1/2
2		Not applicable	3/4
3	16-point quadrature amplitude modulation (16-QAM)	`0` or `1`	1/2
4			3/4
5	64-QAM	Not applicable	2/3
6			3/4
7			5/6
8	256-QAM		3/4
9			5/6
10	1024-QAM		3/4
11			5/6

Data Types: `double`

`DCM` — DCM indicator

`false` or `0` (default) | `true` or `1`

Dual carrier modulation (DCM) indicator, specified as `1` (`true`) or `0` (`false`). To use DCM for the HE-Data field, set this property to `1` (`true`). Otherwise, set this property to `0` (`false`).

Dependencies

You can set this property to `1` (`true`) only when all of these conditions are satisfied.

- The `NumSpaceTimeStreams` property is `1` or `2`.
- The `MCS` property is `0`, `1`, `3`, or `4`.
- The `STBC` property is `0` (`false`).

Data Types: `logical`

`ChannelCoding` — FEC coding type

`'LDPC'` (default) | `'BCC'`

Forward-error-correction (FEC) coding type for the HE-Data field, specified as `'LDPC'` for low-density parity-check (LDPC) coding or `'BCC'` for binary convolutional coding (BCC).

Dependencies

You can set this property to `'BCC'` only when all of these conditions are satisfied.

- The `RUSize` property is `26`, `52`, `106`, or `242`.
- The `NumSpaceTimeStreams` property is `1`, `2`, `3`, or `4`.
- The `MCS` property is not `10` or `11`.

If you set the `TriggerMethod` property to `'TRS'`, you can set this property to `'LDPC'` only when all of these conditions are satisfied.

- The `RUSize` property is `484`, `996`, or `1992`.
- The `PreFECPaddingFactor` property is `4`.
- The `LDPCExtraSymbol` property is `1` (`true`).

Data Types: `char` | `string`

`PreFECPaddingFactor` — Pre-FEC padding factor
`4` (default) | `1` | `2` | `3`

Pre-forward-error-correction (pre-FEC) padding factor, specified as `1`, `2`, `3`, or `4`.

Data Types: `double`

`LDPCExtraSymbol` — Extra OFDM symbol segment indicator
`0` (`false`) (default) | `1` (`true`)

Extra orthogonal frequency-division multiplexing (OFDM) symbol segment indicator, specified as `1` (`true`) or `0` (`false`). To indicate the presence of an extra OFDM symbol segment for LDPC coding, set this property to `1` (`true`). Otherwise, set this property to `0` (`false`).

Dependencies

To enable this property, set the `ChannelCoding` property to `'LDPC'`.

Data Types: `logical`

`PEDisambiguity` — PEDisambiguity subfield value
`0` (`false`) (default) | `1` (`true`)

PE Disambiguity subfield value, specified as `1` (`true`) or `0` (`false`). For more information, see section 27.3.12 of [2].

Data Types: `logical`

`LSIGLength` — Length of L-SIG field
`142` (default) | integer in the interval [1, 4093]

Length of L-SIG field, in OFDM symbols, specified as an integer in the interval [1, 4093]. The L-SIG length must satisfy `mod(LSIGLength,3) = 1`, where `mod(a,m)` returns the remainder after dividing `a` by `m`. For more information, see `mod`.

Dependencies

To enable this property, set the `TriggerMethod` property to `'TriggerFrame'`.

Data Types: `double`

`NumDataSymbols` — Number of OFDM symbols in HE-Data field
`10` (default) | positive integer

Number of OFDM symbols in the HE-Data field, specified as a positive integer.

Dependencies

To enable this property, set the `TriggerMethod` property to `'TRS'`.

Data Types: `double`

`DefaultPEDuration` — Packet extension duration

`0` (default) | `4` | `8` | `12` | `16`

Packet extension duration, in microseconds, specified as `0`, `4`, `8`, `12`, or `16`. For more information about the packet extension field, see section 27.3.12 of [2].

Dependencies

To enable this property, set the `TriggerMethod` property to `'TRS'`.

Data Types: `double`

`SingleStreamPilots` — HE-LTF single-stream pilots indicator

`true` or `1` (default) | `false` or `0`

HE-LTF single-stream pilots indicator, specified as `1` (`true`) or `0` (`false`). To indicate that the HE-LTF uses single-stream pilots, set this property to `1` (`true`). Otherwise, set this property to `0` (`false`).

Data Types: `logical`

`HighDoppler` — High-Doppler mode indicator

`false` or `0` (default) | `true` or `1`

High-Doppler mode indicator, specified as `1` (`true`) or `0` (`false`). To indicate high-Doppler mode in the HE-SIG-A field, set this property to `1` (`true`). Otherwise, set this property to `0` (`false`).

Dependencies

You can set this property to `1` (`true`) only when the `TriggerMethod` property is `'TriggerFrame'` and the `NumSpaceTimeStreams` property is `1`, `2`, `3`, or `4` for any RU.

Data Types: `logical`

`MidamblePeriodicity` — Midamble periodicity of HE-Data field

`10` (default) | `20`

Midamble periodicity of the HE-Data field, in number of OFDM symbols, specified as `10` or `20`.

Dependencies

To enable this property, set the `HighDoppler` property to `1` (`true`).

Data Types: `double`

`HESIGAReservedBits` — Reserved bits in HE-SIG-A field

`ones(9,1)` (default) | nine-element binary-valued

Reserved bits in the HE-SIG-A field, specified as a nine-element binary-valued column vector.

Data Types: `double`

`PostFECPaddingSource` — Post-FEC padding bit source

`'mt19937ar with seed'` (default) | `'Global stream'` | `'User-defined'`

Post-FEC padding bit source used by the `wlanWaveformGenerator` function, specified as one of these values.

- `'mt19937ar with seed'` — Generate normally distributed random bits by using the mt19937ar algorithm with seed specified in the `PostFECPaddingSeed` property.

- `'Global stream'` — Generate normally distributed random bits by using the current global random number stream.
- `'User-defined'` — Use the bits specified in the `PostFECPaddingBits` property as the post-FEC padding bits.

Data Types: `char` | `string`

`PostFECPaddingSeed` — Post-FEC padding bit seed for mt19937ar algorithm

`73` (default) | nonnegative integer

Post-FEC padding bit seed for the mt19937ar algorithm, specified as a nonnegative integer.

Dependencies

To enable this property, set the `PostFECPaddingSource` property to `'mt19937ar with seed'`.

Data Types: `double`

`PostFECPaddingBits` — Post-FEC padding bits

`0` (default) | binary-valued column vector

Post-FEC padding bits, specified as a binary-valued scalar or column vector.

To generate a waveform, the `wlanWaveformGenerator` function requires *n* bits, where *n* depends on the specified configuration. To calculate *n*, use the `getNumPostFECPaddingBits` object function with the specified configuration object as the input argument and specify this property as a vector of length *n*. Alternatively, specify this input as a binary-valued scalar or column vector of arbitrary length. If the length of this property is less than *n*, the waveform generator loops the vector to create a vector of length *n*. If the length of this property is greater than *n*, the function uses only the first *n* entries as the post-FEC padding bits.

Note For C/C++ code generation, you must specify the data type of this property as `int8`.

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

HE TB Feedback NDP Properties

Properties in this section apply only when the `FeedbackNDP` property is `1` (`true`).

`RUToneSetIndex` — RU tone set index for HE TB feedback NDP

`1` (default) | integer in the interval [1, 144]

RU tone set index for an HE TB feedback NDP, specified as one of these options.

- When the `ChannelBandwidth` property is `'CBW20'`, set this property to an integer in the interval [1, 18].
- When the `ChannelBandwidth` property is `'CBW40'`, set this property to an integer in the interval [1, 36].
- When the `ChannelBandwidth` property is `'CBW80'`, set this property to an integer in the interval [1, 72].
- When the `ChannelBandwidth` property is `'CBW160'`, set this property to an integer in the interval [1, 144].

This property defines the subcarrier allocation tone sets in the high-efficiency long training field (HE-LTF) on which the STA transmits the HE TB feedback NDP.

Data Types: `double`

`FeedbackStatus` — Feedback status
`true` or `1` (default) | `false` or `0`

Feedback status, specified as `1` (`true`) or `0` (`false`). The value of this property indicates the value of the bit used for tone modulation in each tone set specified by the `RUToneSetIndex` property. The feedback status and RU tone set index determine the HE-LTF subcarrier mapping in accordance with Table 27-30 of [2].

Data Types: `logical`

Object Functions

getNDPFeedbackConfiguration	Valid HE TB feedback NDP PHY configuration
getNumPostFECPaddingBits	Calculate required number of post-FEC padding bits
getPSDULength	Calculate HE or WUR PSDU length
getTRSConfiguration	Valid HE TB PHY configuration in response to triggering frame containing TRS Control subfield
packetFormat	WLAN packet format
ruInfo	Resource unit allocation information
showAllocation	Resource unit (RU) allocation
transmitTime	Packet transmission time

Examples

Generate HE TB Waveform

Configure and generate a WLAN waveform containing an HE TB uplink packet.

Create a configuration object for a WLAN HE TB uplink transmission.

```
cfgHETB = wlanHETBConfig;
```

Obtain the PSDU length, in bytes, from the configuration object by using the `getPSDULength` object function.

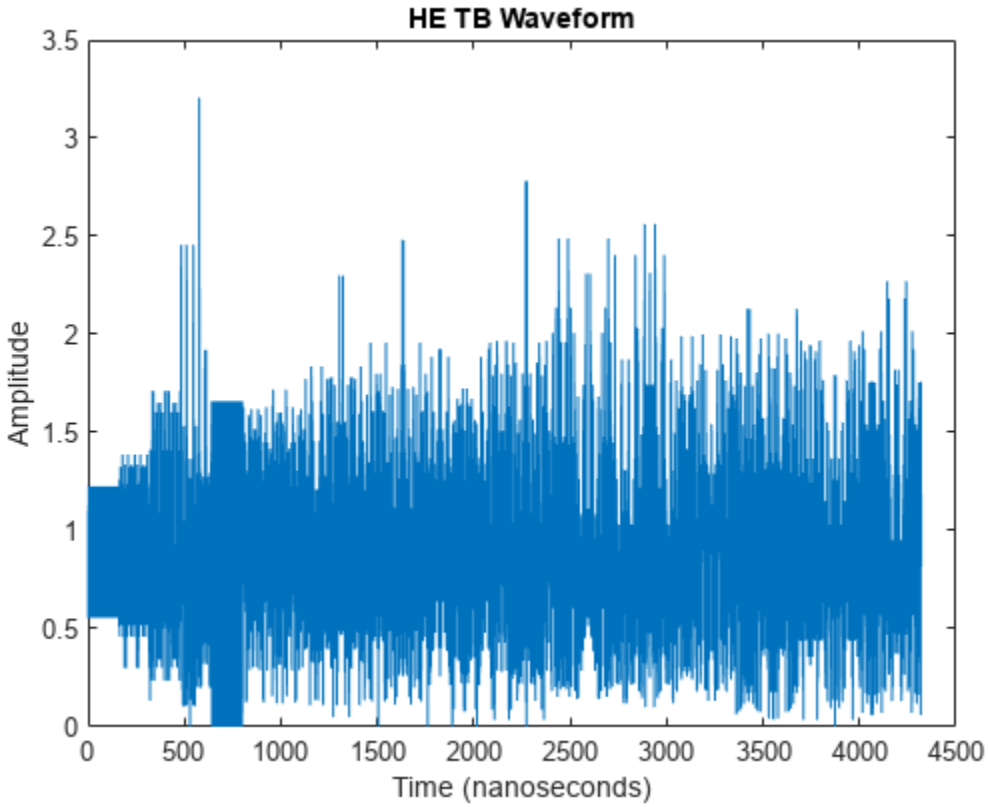
```
psduLength = getPSDULength(cfgHETB);
```

Generate a PSDU of the relevant length.

```
psdu = randi([0 1],8*psduLength,1);
```

Generate and plot the waveform.

```
waveform = wlanWaveformGenerator(psdu,cfgHETB);
figure;
plot(abs(waveform));
title('HE TB Waveform');
xlabel('Time (nanoseconds)');
ylabel('Amplitude');
```



Recover Feedback Status from HE TB Feedback NDP

Configure an uplink HE TB feedback NDP transmission with four stations (STAs), a channel bandwidth of 20 MHz, and a signal-to-noise ratio (SNR) of 20 dB.

```
numSTA = 4;
cbw = 'CBW20';
snr = 20;
cfgSTA = cell(1,numSTA);
```

Specify the resource unit (RU) tone set index, starting space-time stream, and feedback status for all STAs.

```
ruToneSetIndex = repmat([1 2],1,round(numSTA/2));
startingSTS = repmat([1 2],1,round(numSTA/2));
feedbackStatus = repmat([1 0],1,round(numSTA/2));
```

Create a valid HE TB feedback NDP configuration.

```
cfg = wlanHETBConfig;
cfg = getNDPFeedbackConfiguration(cfg);
```

Configure the channel for transmission, assuming no variation across STAs.

```
tgax = wlanTGaxChannel('ChannelBandwidth',cbw, ...
    'TransmissionDirection','Uplink', ...
```

```
    'SampleRate',wlanSampleRate(cfg));
chanInfo = info(tgax);
awgn = comm.AWGNChannel('NoiseMethod','Signal to noise ratio (SNR)', ...
    'SignalPower',1/tgax.NumReceiveAntennas);
```

Configure STAs and generate an HE TB feedback NDP waveform.

```
rx = 0;
for idx = 1:numSTA

    % Configure STAs

    cfg.RUToneSetIndex = ruToneSetIndex(idx);
    cfg.StartingSpaceTimeStream = startingSTS(idx);
    cfg.FeedbackStatus = feedbackStatus(idx);
    cfgSTA{idx} = cfg;

    % Generate transmit waveform

    waveform = wlanWaveformGenerator([],cfg);

    % Pass waveform through TGax channel

    rx = rx + tgax([waveform; zeros(15,size(waveform,2))]);
end
```

Pass the waveform through the AWGN channel, accounting for the noise energy in nulls to ensure the SNR is defined per active and complementary subcarrier.

```
field = 'HE-LTF';
ofdmInfo = wlanHEOFDMInfo(field,cbw,cfg.GuardInterval);
awgn.SNR = snr - 10*log10(ofdmInfo.FFTLength/12);
rx = awgn(rx);
```

Get the field indices and extract the HE-LTF.

```
ind = wlanFieldIndices(cfgSTA{1});
offset = chanInfo.ChannelFilterDelay;
heltf = rx(offset+(ind.HELTF(1):ind.HELTF(2)),:);
```

Demodulate the HE-LTF.

```
rxSym = wlanHEDemodulate(heltf,field,cbw,cfg.GuardInterval,cfg.HELTFType);
```

Recover the feedback status for the STAs.

```
status = zeros(1,numSTA);
for n = 1:numSTA
    status(n) = wlanHETBNDPFeedbackStatus(rxSym,cfgSTA{n});
end
```

Compare the transmitted and received feedback status for the STAs.

```
disp(isequal(feedbackStatus(1:numSTA),status))
```

```
   1
```

Generate HE TB Waveform in Response to Frame Containing TRS Control Subfield

Configure and generate a WLAN HE TB waveform to be transmitted in response to a frame containing a TRS Control subfield.

Create an HE TB configuration object, specifying the triggering frame type.

```
cfgHETB = wlanHETBConfig('TriggerMethod','TRS');
```

Generate a valid configuration by using the `getTRSConfiguration` object function, displaying the result.

```
cfgTRS = getTRSConfiguration(cfgHETB)
```

```
cfgTRS =
  wlanHETBConfig with properties:

               FeedbackNDP: 0
             TriggerMethod: 'TRS'
          ChannelBandwidth: 'CBW20'
                    RUSize: 242
                   RUIndex: 1
    PreHEPowerScalingFactor: 1
        NumTransmitAntennas: 1
        NumSpaceTimeStreams: 1
    StartingSpaceTimeStream: 1
             SpatialMapping: 'Direct'
                       STBC: 0
                        MCS: 0
                        DCM: 0
             ChannelCoding: 'BCC'
        PreFECPaddingFactor: 4
             NumDataSymbols: 10
          DefaultPEDuration: 0
             GuardInterval: 3.2000
                 HELTFType: 4
            NumHELTFSymbols: 1
         SingleStreamPilots: 1
                   BSSColor: 0
             SpatialReuse1: 15
             SpatialReuse2: 15
             SpatialReuse3: 15
             SpatialReuse4: 15
              TXOPDuration: 127
               HighDoppler: 0
          HESIGAReservedBits: [9x1 double]
       PostFECPaddingSource: 'mt19937ar with seed'
         PostFECPaddingSeed: 73
```

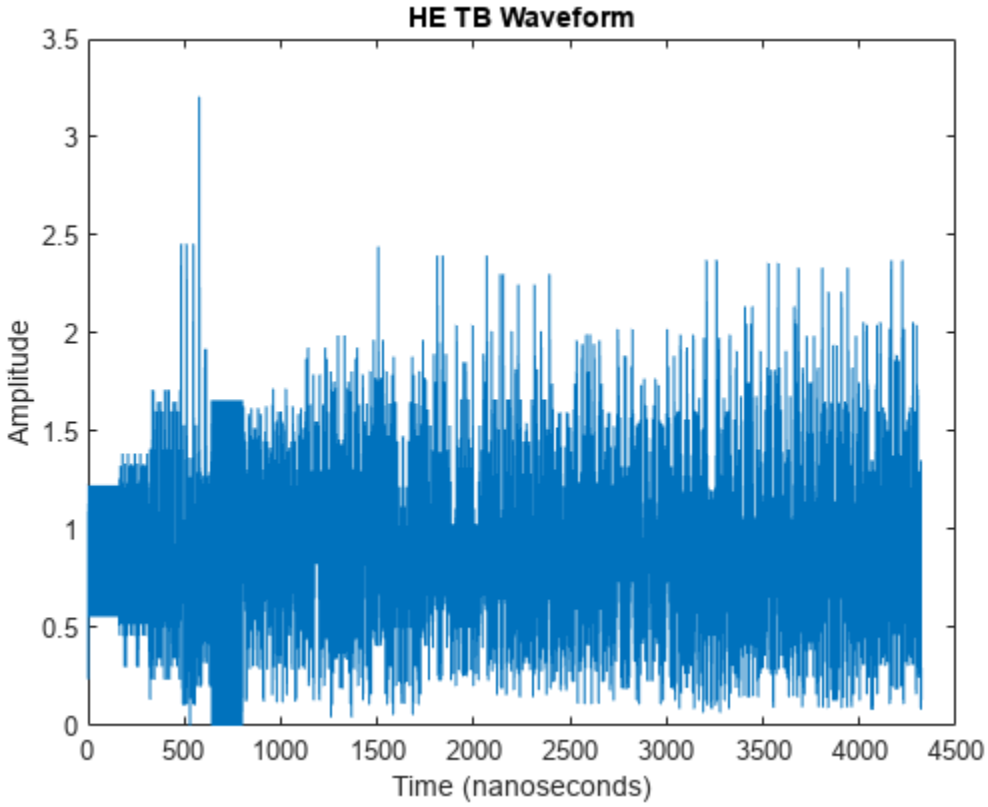
Get the PSDU length in bytes and generate a PSDU for transmission.

```
psduLength = getPSDULength(cfgTRS);
psdu = randi([0 1],8*psduLength,1);
```

Generate and plot the waveform.

```
waveform = wlanWaveformGenerator(psdu,cfgTRS);
figure;
```

```
plot(abs(waveform));
title('HE TB Waveform');
xlabel('Time (nanoseconds)');
ylabel('Amplitude');
```



More About

PPDU

The physical layer (PHY) protocol data unit (PPDU) is the complete physical layer convergence procedure (PLCP) frame, including PLCP headers, MAC headers, the MAC data field, and the MAC and PLCP trailers.

NDP Feedback Report Procedure

To allocate resources to a station (STA) for transmission of an HE TB PPDU, an HE access point (AP) requires simultaneous resource request information (feedback status) from multiple uplink STAs. The AP obtains this information by using the HE TB NDP feedback report procedure specified in section 26.5.7 of [2]. This procedure improves the power and system efficiency of the IEEE 802.11ax network.

The AP solicits an NDP feedback report response from the STAs by sending an NDP feedback report poll (NFRP) trigger frame. The NFRP trigger frame contains scheduling information for the STAs. Each STA transmits a response only if it satisfies all of these conditions.

- The STA is associated with the basic service set identifiers (BSSIDs) indicated by the transmitter address (TA) subfield of the NFRP trigger frame
- $AID_{start} \le AID_{STA} \le AID_{start} + N_{STA}$, where AID_{STA} is the STA association identifier (AID), AID_{start} is the value of the starting AID subfield of the NFRP trigger frame, and N_{STA} is the total number of non-AP STAs scheduled to respond to the NFRP trigger frame.

The STAs that transmit a response use the information contained in the NFRP trigger frame to derive these parameters for the HE TB feedback NDP response.

- The RU tone set index (corresponding to the `RUToneSetIndex` property of this object), which defines the small allocation tone sets in the HE-LTF
- The starting space-time stream number, which defines the orthogonal allocation of tone sets for multiplexing the feedback status of different STAs

All participating STAs respond within a short interframe space (SIFS) interval after receiving the NFRP trigger frame.

Version History

Introduced in R2020a

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

[2] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

You must specify the data type of the `postFECPaddingBits` property as `int8`.

See Also

Objects

`wlanDMGConfig` | `wlanHEMUConfig` | `wlanHERecoveryConfig` | `wlanHTConfig` | `wlanNonHTConfig` | `wlanS1GConfig` | `wlanVHTConfig`

Functions

`transmitTime` | `wlanHEDemodulate` | `wlanHELTFChannelEstimate` | `wlanHETBNDPFeedbackStatus` | `wlanWaveformGenerator`

Apps
WLAN Waveform Generator

wlanHTConfig

Configure HT transmission

Description

The `wlanHTConfig` object is a configuration object for the WLAN high throughput (HT) packet format.

Creation

Syntax

```
cfgHT = wlanHTConfig
cfgHT = wlanHTConfig(Name,Value)
```

Description

`cfgHT = wlanHTConfig` creates a configuration object that initializes parameters for an IEEE 802.11 HT “PPDU” on page 4-145.

`cfgHT = wlanHTConfig(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotation marks. For example, `wlanHTConfig('GuardInterval','Short')` specifies a 400 nanosecond guard interval (cyclic prefix) duration.

At runtime, the calling function validates object settings for properties relevant to the operation of the function.

Properties

`ChannelBandwidth` — Channel bandwidth of PPDU transmission

`'CBW20'` (default) | `'CBW40'`

Channel bandwidth of PPDU transmission, specified as one of these values:

- `'CBW20'` – Channel bandwidth of 20 MHz
- `'CBW40'` – Channel bandwidth of 40 MHz

Data Types: `char` | `string`

`NumTransmitAntennas` — Number of transmit antennas

`1` (default) | positive integer

Number of transmit antennas, specified as a positive integer.

Data Types: `double`

wlanMACTriggerUserConfig

Configure User Info field of WLAN MAC trigger frame

Description

The `wlanMACTriggerUserConfig` object configures the User Info field of an IEEE 802.11 medium access control (MAC) trigger frame body.

Creation

Syntax

```
cfgUser = wlanMACTriggerUserConfig
cfgUser = wlanMACTriggerUserConfig(Name,Value)
```

Description

`cfgUser = wlanMACTriggerUserConfig` creates a default configuration object for the User Info field of a WLAN MAC trigger frame. For more information on the trigger frame format and its User Info field, see section 9.3.1.22 of [1].

`cfgUser = wlanMACTriggerUserConfig(Name,Value)` sets property values by using one or more name-value arguments. Enclose each property name in quotes. For example, `'TriggerType','Basic'` specifies a basic trigger frame type.

Properties

`TriggerType` — Trigger frame type

`'Basic'` (default) | `'MU-BAR'` | `'MU-RTS'`

Trigger frame type, specified as one of these values.

- `'Basic'` — Basic trigger frame
- `'MU-BAR'` — Multi-user block ack request (MU-BAR) frame
- `'MU-RTS'` — Multi-user request to send (MU-RTS) frame

For more information, see section 9.3.1.22 of [1].

Data Types: `char` | `string`

`AID12` — Value of AID12 subfield

1 (default) | integer in the interval [0, 2007] | `2045` | `2046`

Value of the AID12 subfield, specified as one of these values.

- `0` — The User Info field allocates contiguous random access resource units (RA-RUs) for associated stations (STAs).

- An integer in the interval [1, 2007] — This property represents the STA association identifier (AID) for the User Info field.
- `2045` — The User Info field allocates one or more contiguous RA-RUs for unassociated STAs.
- `2046` — The User Info field identifies an unallocated RU.

Data Types: `double`

`RUAllocationRegion` — RU allocation region
`'primary 80MHz'` (default) | `'secondary 80MHz'`

RU allocation region in the 80+80 MHz or 160 MHz channel, specified as `'primary 80MHz'` or `'secondary 80MHz'`.

Dependencies

- To enable this property, set the `RUSize` property to a value other than `1992`.
- To enable this property when this object is an element of the `UserInfo` property of a `wlanMACTriggerConfig` object, set the `ChannelBandwidth` property of the `wlanMACTriggerConfig` object to `'CBW80+80 or CBW160'`.

Data Types: `char` | `string`

`RUSize` — RU size
`242` (default) | `26` | `52` | `106` | `484` | `996` | `1992`

RU size, specified as `26`, `52`, `106`, `242`, `484`, `996`, or `1992`.

Data Types: `double`

`RUIndex` — RU index for subcarrier allocation
`1` (default) | integer in the interval [1, 37]

RU index for subcarrier allocation, specified as an integer in the interval [1, 37]. The RU index specifies the location of the RU within the channel. For example, an 80 MHz transmission contains four 242-tone RUs (one for each 20 MHz subchannel). RU number 242-1 (size 242, index 1) is the lowest absolute frequency within the 80 MHz channel. Similarly, RU number 242-4 is the highest absolute frequency. For a 160 MHz transmission, this property indicates the RU index value for the 80 MHz segment specified by the `RUAllocationRegion` property.

Data Types: `double`

`ChannelCoding` — FEC coding type
`'LDPC'` (default) | `'BCC'`

Forward-error-correction (FEC) coding type for the HE-Data field of the HE TB PPDU response, specified as one of these values.

- `'LDPC'` — Low-density parity-check (LDPC) coding
- `'BCC'` — Binary convolutional coding (BCC)

Data Types: `char` | `string`

`MCS` — Modulation and coding scheme
`0` (default) | integer in the interval [0, 11]

Modulation and coding scheme (MCS) used in transmitting the HE TB PPDU response, specified as an integer in the interval [0, 11]. This table shows the modulation type and coding rate for each valid value of this property.

Value of MCS	Modulation Type	Dual Carrier Modulation	Coding Rate
0	Binary phase-shift keying (BPSK)	0 or 1	1/2
1	Quadrature phase-shift keying (QPSK)		1/2
2		Not applicable	3/4
3	16-point quadrature amplitude modulation (16-QAM)	0 or 1	1/2
4			3/4
5	64-QAM	Not applicable	2/3
6			3/4
7			5/6
8	256-QAM		3/4
9			5/6
10	1024-QAM		3/4
11			5/6

When you set the DCM property to 1 (true), you must set this property to 0, 1, 3, or 4.

Data Types: double

DCM — DCM indicator

false or 0 (default) | true or 1

Dual carrier modulation (DCM) indicator, specified as 1 (true) or 0 (false). To use DCM for the HE-Data field of the HE TB PPDU response, set this property to 1 (true). Otherwise, set this property to 0 (false).

When this object is an element of the UserInfo property of a wlanMACTriggerConfig object, you can set this property to 1 (true) only when the STBC property of the wlanMACTriggerConfig object is 0 (false).

Data Types: logical

StartingSpatialStream — Starting spatial stream index

1 (default) | integer in the interval [1, 8]

Starting spatial stream index, in one-based form, specified as an integer in the interval [1, 8].

Dependencies

To enable this property, set the AID12 property to a value other than 0 or 2045.

Data Types: double

NumSpatialStreams — Number of spatial streams

1 (default) | integer in the interval [1, 8]

Number of spatial streams, specified as an integer in the interval [1, 8].

Dependencies

To enable this property, set the `AID12` property to a value other than `0` or `2045`.

Data Types: `double`

`NumRARU` — Number of allocated contiguous RA-RUs

`1` (default) | integer in the interval [1, 32]

Number of allocated contiguous RA-RUs, specified as an integer in the interval [1, 32].

Dependencies

To enable this property, set the `AID12` property to `0` or `2045`.

Data Types: `double`

`MoreRARU` — Indication of more RA-RU allocations in subsequent trigger frames

`false` or `0` (default) | `true` or `1`

Indication of more RA-RU allocations in subsequent trigger frames, specified as `1` (`true`) or `0` (`false`). To indicate more RA-RU allocations in subsequent trigger frames, set this property to `1` (`true`). Otherwise, set this property to `0` (`false`).

Dependencies

- To enable this property, set the `AID12` property to `0` or `2045`.
- To enable this property when this object is an element of the `UserInfo` property of a `wlanMACTriggerConfig` object, set the `MoreTF` property of the `wlanMACTriggerConfig` object to `1` (`true`).

Data Types: `logical`

`UseMaxTransmitPower` — Maximum transmit power indicator

`true` or `1` (default) | `false` or `0`

Maximum transmit power indicator, specified as `1` (`true`) or `0` (`false`). To solicit maximum transmit power of the HE TB PPDU response from the receiving station for the assigned HE MCS value, set this property to `1` (`true`). Otherwise, set this property to `0` (`false`).

Data Types: `logical`

`TargetRSSI` — Expected power of received signal

`-110` (default) | integer in the interval [–110, –20]

Expected power of the received signal, in dBm, specified as an integer in the interval [–110, –20]. This property represents the expected power of the HE TB PPDU response transmitted on the assigned RU averaged over the AP antenna connectors and rounded to the nearest integer.

Dependencies

To enable this property, set the `UseMaxTransmitPower` property to `0` (`false`).

Data Types: `double`

`MPDUMUSpacingFactor` — Value of MU MPDU Spacing Factor subfield

`0` (default) | `1` | `2` | `3`

Value of the MU MPDU Spacing Factor subfield in a basic trigger frame, specified as `0`, `1`, `2`, or `3`. This property indicates the minimum MPDU start spacing multiplication factor. For more information, see sections 9.3.1.22.2 and 10.13.3 of [1].

Dependencies

To enable this property, set the `TriggerType` property to `'Basic'`.

Data Types: `double`

`TIDAggregationLimit` — Value of TID Aggregation Limit subfield

`0` (default) | integer in the interval [0, 7]

Value of the TID Aggregation Limit subfield, specified as an integer in the interval [0, 7]. This subfield indicates the maximum number of traffic identifiers (TIDs) that a station can aggregate.

Dependencies

To enable this property, set the `TriggerType` property to `'Basic'`.

Data Types: `double`

`PreferredAC` — Value of Preferred AC subfield

`0` (default) | `1` | `2` | `3`

Value of the Preferred AC subfield, specified as `0`, `1`, `2`, or `3`. This subfield indicates the lowest recommended access category for aggregation of MAC protocol data units (MPDUs) in the HE TB PPDU response.

Dependencies

To enable this property, set the `TriggerType` property to `'Basic'`.

Data Types: `double`

`TID` — Traffic identifier

`0` (default) | integer in the interval [0, 7]

Traffic identifier, specified as an integer in the interval [0, 7].

Dependencies

To enable this property, set the `TriggerType` property to `'MU-BAR'`.

Data Types: `double`

`StartingSequenceNum` — Starting MSDU or A-MSDU sequence number

`0` (default) | integer in the interval [0, 4095]

Starting MAC service data unit (MSDU) or aggregate MSDU (A-MSDU) sequence number, specified as an integer in the interval [0, 4095].

Dependencies

To enable this property, set the `TriggerType` property to `'MU-BAR'`.

Data Types: `double`

Examples

Create Basic MAC Trigger Frame

Create a basic MAC trigger frame to carry information for two users.

Create a MAC trigger frame-body configuration object, specifying a channel bandwidth of 40 MHz.

```
cfgTrigger = wlanMACTriggerConfig(ChannelBandwidth="CBW40");
```

Create configuration objects for the User Info fields of the trigger frame.

```
cfgUser1 = wlanMACTriggerUserConfig(AID12=1, ...
    RUSize=242,RUIndex=1);
cfgUser2 = wlanMACTriggerUserConfig(AID12=2, ...
    RUSize=242,RUIndex=2);
```

Add the User Info fields to the trigger frame.

```
cfgTrigger = addUserInfo(cfgTrigger,cfgUser1);
cfgTrigger = addUserInfo(cfgTrigger,cfgUser2);
```

Configure the trigger frame by creating a MAC frame-body configuration object, specifying the frame type and the trigger frame-body configuration.

```
cfgMAC = wlanMACFrameConfig(FrameType="Trigger", ...
    TriggerConfig=cfgTrigger);
```

Specify a non-HT PHY configuration by creating a default non-HT configuration object.

```
cfgPHY = wlanNonHTConfig;
```

Create the MAC trigger frame and display its length.

```
[frame,frameLength] = wlanMACFrame(cfgMAC,cfgPHY);
disp(frameLength)
```

```
    40
```

Version History

Introduced in R2021a

References

[1] IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`wlanMACFrame`

Objects

`wlanMACFrameConfig` | `wlanMACTriggerConfig`

Topics

“Generate and Parse WLAN MAC Frames”

wlanNonHTConfig

Configure non-HT transmission

Description

The `wlanNonHTConfig` object is a configuration object for the WLAN non-high-throughput (non-HT) packet format.

Creation

Syntax

```
cfgNonHT = wlanNonHTConfig
cfgNonHT = wlanNonHTConfig(Name,Value)
```

Description

`cfgNonHT = wlanNonHTConfig` creates a configuration object that initializes parameters for an IEEE 802.11 non-HT "PPDU" on page 4-186.

`cfgNonHT = wlanNonHTConfig(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotation marks. For example, `wlanNonHTConfig('Modulation','DSSS')` specifies the modulation type as direct-sequence spread spectrum (DSSS).

At runtime, the calling function validates object settings for properties relevant to the operation of the function.

Properties

`Modulation` — Modulation type for non-HT transmission

`'OFDM'` (default) | `'DSSS'`

Modulation type for the non-HT transmission, specified as `'OFDM'` for orthogonal frequency division multiplexing (OFDM) or `'DSSS'` for direct-sequence spread spectrum (DSSS).

Data Types: `char` | `string`

`ChannelBandwidth` — Channel bandwidth of PPDU transmission

`'CBW20'` (default) | `'CBW5'` | `'CBW10'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth of PPDU transmission, specified as one of these values.

- `'CBW5'` — Channel bandwidth of 5 MHz
- `'CBW10'` — Channel bandwidth of 10 MHz
- `'CBW20'` — Channel bandwidth of 20 MHz

- `'CBW40'` — Channel bandwidth of 40 MHz for non-HT duplicate
- `'CBW80'` — Channel bandwidth of 80 MHz for non-HT duplicate
- `'CBW160'` — Channel bandwidth of 160 MHz for non-HT duplicate

Data Types: `char` | `string`

`InactiveSubchannels` — Indicate inactive 20 MHz subchannels in non-HT duplicate packet

`false` or `0` (default) | logical vector

Indicate inactive 20 MHz subchannels in a non-HT duplicate packet, specified as a numeric or logical `0` (`false`) or a logical vector with at least one element set to `0` (`false`). When specifying a vector, the elements correspond to subchannels in order of increasing absolute frequency. Each element indicates whether the corresponding 20 MHz subchannel is inactive. To indicate an inactive 20 MHz subchannel, set the corresponding element to `1` (`true`). If you set this property to `0` (`false`), the `wlanNonHTConfig` object applies that value to all 20 MHz subchannels, indicating that all subchannels are active.

Example: `[0 0 0 1]` indicates a non-HT duplicate packet such that the subchannel with the highest absolute frequency value is inactive.

Dependencies

To enable this property, set the `ChannelBandwidth` property to either `'CBW80'` or `'CBW160'`.

Data Types: `logical`

`MCS` — OFDM MCS used for transmission

`0` (default) | integer in the interval [0, 7]

OFDM MCS used for transmission, specified as an integer in the interval [0, 7]. This table shows the modulation type and coding rate for each valid value of `MCS`.

Value of MCS	Modulation	Coding Rate	Coded Bits Per Subcarrier	Coded Bits Per OFDM Symbol	Data Bits Per OFDM Symbol	Data Rate in Mbps		
						5-MHz Channel Bandwidth	10-MHz Channel Bandwidth	20-MHz Channel Bandwidth
0	Binary phase-shift keying (BPSK)	1/2	1	48	24	1.5	3	6
1	BPSK	3/4	1	48	36	2.25	4.5	9
2	quadrature phase-shift keying (QPSK)	1/2	2	96	48	3	6	12
3	QPSK	3/4	2	96	72	4.5	9	18

Value of MCS	Modulation	Coding Rate	Coded Bits Per Subcarrier	Coded Bits Per OFDM Symbol	Data Bits Per OFDM Symbol	Data Rate in Mbps		
						5-MHz Channel Bandwidth	10-MHz Channel Bandwidth	20-MHz Channel Bandwidth
4	16-point quadrature amplitude modulation (16-QAM)	1/2	4	192	96	6	12	24
5	16-QAM	3/4	4	192	144	9	18	36
6	64-QAM	2/3	6	288	192	12	24	48
7	64-QAM	3/4	6	288	216	13.5	27	54

For more information, see Table 17-4 of [1].

Data Types: `double`

`DataRate` — Data rate for DSSS modulation

`'1Mbps'` (default) | `'2Mbps'` | `'5.5Mbps'` | `'11Mbps'`

Data rate for DSSS modulation, specified as a one of these values:

- `'1Mbps'` — Differential binary phase-shift keying (DBPSK) with a data rate of 1 Mbps
- `'2Mbps'` — Differential quadrature phase-shift keying (DQPSK) with a data rate of 2 Mbps
- `'5.5Mbps'` — Complementary code keying (CCK) with a data rate of 5.5 Mbps
- `'11Mbps'` — CCK with a data rate of 11 Mbps

Data Types: `char` | `string`

`Preamble` — DSSS modulation preamble type

`'Long'` (default) | `'Short'`

DSSS modulation preamble type, specified as `'Long'` or `'Short'`.

Dependencies

The `'Short'` value of this property does not apply when you set the `DataRate` property to `'1Mbps'`.

Data Types: `char` | `string`

`LockedClocks` — Clock locking indicator for DSSS modulation

`true` or `1` (default) | `false` or `0`

Clock locking indicator for DSSS modulation, specified as a numeric or logical `1` (`true`) or `0` (`false`). This property corresponds to the locked clocks bit (bit b2) of the *SERVICE* field as specified in section 16.2.3.5 of [1]. To indicate that the physical layer (PHY) implementation derives its transmit frequency clock and symbol clock from the same oscillator, set this property to `1` (`true`). For more information, see sections 16.2.3.5 and 18.1.3 of [1].

Note Section 18.3.2.2 of [1] specifies that the locked clocks bit must be 1 for all extended rate PHY (ERP) systems when transmitting at any of these rates:

- An optional ERP-packet binary convolutional coding (ERP-PBCC) rate
- A data rate described in Section 16 of [1]

Therefore, to model ERP systems, you must set this property to `1` (`true`).

Data Types: `logical`

`PSDULength` — PSDU length

`1000` (default) | integer in the interval [0, 4095]

Physical layer convergence procedure (PLCP) service data unit (PSDU) length, in bytes, specified as an integer in the interval [0, 4095].

Data Types: `double`

`NumTransmitAntennas` — Number of transmit antennas

`1` (default) | positive integer

Number of transmit antennas, specified as a positive integer.

Dependencies

To enable this property, set the `ChannelBandwidth` property to `'CBW20'`.

Data Types: `double`

`CyclicShifts` — Cyclic shift values of additional transmit antennas

`-75` (default) | integer in the interval [–200, 0] | row vector

Cyclic shift values, in nanoseconds, of additional transmit antennas. The first eight antennas use the cyclic shift values specified in Table 21-10 of [1]. The remaining L antennas use the values you specify in this property, where L = `NumTransmitAntennas` – 8. Specify this property as one of these values:

- An integer in the interval [–200, 0] – the `wlanNonHTConfig` object uses this cyclic shift value for each of the L additional antennas.
- A row vector of length L of integers in the interval [–200, 0] – the `wlanNonHTConfig` object uses the kth element as the cyclic shift value for the (k + 8)th transmit antenna.

Note If you specify this property as a row vector of length greater than L, the `wlanNonHTConfig` object uses only the first L elements. For example, if you set the `NumTransmitAntennas` property to `16`, the `wlanNonHTConfig` object uses only the first L= 16 – 8 = 8 elements of this vector.

Dependencies

To enable this property, set the `NumTransmitAntennas` property to a value greater than `8`.

Data Types: `double`

`SignalChannelBandwidth` — Signal channel bandwidth in scrambler sequence

`false` or `0` (default) | `true` or `1`

Signal channel bandwidth in the scrambler sequence, specified as a numeric or logical `1` (`true`) or `0` (`false`). To signal the channel bandwidth, set this property to `1` (`true`). Otherwise, set this property to `0` (`false`). For more information, see section 17.3.5.5 of [1].

Dependencies

To enable this property, set the `Modulation` property to `'OFDM'`.

Data Types: `logical`

`BandwidthOperation` — Signal bandwidth operation in scrambler sequence
`'Absent'` (default) | `'Static'` | `'Dynamic'`

Signal bandwidth operation in the scrambler sequence, specified as one of these values.

- `'Absent'` — Disable bandwidth operation signaling
- `'Static'` — Signal static bandwidth operation
- `'Dynamic'` — Signal dynamic bandwidth operation

For more information, see section 17.3.5.5 of [1].

Data Types: `char` | `string`

Object Functions

scramblerRange	Get scrambler initialization range
transmitTime	Packet transmission time

Examples

Generate Data Field Signal of Non-HT Transmission

Configure transmission parameters by creating a `wlanNonHTConfig` object, specifying a channel bandwidth of 80 MHz and static bandwidth operation.

```
cfg = wlanNonHTConfig('ChannelBandwidth','CBW80','SignalChannelBandwidth',true, ...
     'BandwidthOperation','Static');
```

Generate a random PSDU of the appropriate length.

```
psdu = randi([0 1],8*cfg.PSDULength,1,'int8');
```

Generate the initial pseudorandom scrambler sequence.

```
[range,numBits] = scramblerRange(cfg);
scramInit = randi(range);
```

Generate the non-HT Data field signal.

```
y = wlanNonHTData(psdu,cfg,scramInit);
```

Create Non-HT Format Configuration Object for DSSS Modulation

Create a `wlanNonHTConfig` object for DSSS operation for a PSDU length of 2048 bytes.

```
cfgNHT = wlanNonHTConfig('Modulation','DSSS','PSDULength',2048)

cfgNHT =
  wlanNonHTConfig with properties:

      Modulation: 'DSSS'
        DataRate: '1Mbps'
    LockedClocks: 1
      PSDULength: 2048
```

More About

OFDM PLCP Timing Parameters

Section 17 of [1] specifies OFDM PLCP 5 MHz, 10 MHz, and 20 MHz channel bandwidth operation.

This table lists timing parameters associated with the OFDM PLCP.[27].

Parameter	Value	20 MHz Channel Bandwidth	10 MHz Channel Bandwidth	5 MHz Channel Bandwidth
N_{SD}: Number of data subcarriers	48	48	48	48
N_{SP}: Number of pilot subcarriers	4	4	4	4
N_{ST}: Total number of subcarriers	$N_{SD} + N_{SP}$	52	52	52
Δ_F: Subcarrier frequency spacing, in MHz	(Channel bandwidth) / 64	0.3125	0.15625	0.078125
T_{FFT}: Inverse Fast Fourier Transform (IFFT) / Fast Fourier Transform (FFT) period, in µs	$1 / \Delta_F$	3.2	6.4	12.8
$T_{PREAMBLE}$: PLCP preamble duration, in µs	$T_{SHORT} + T_{LONG}$	16	32	64
T_{SIGNAL}: Duration of the L-SIG symbol, in µs	$T_{GI} + T_{FFT}$	4	8	16
T_{GI}: GI duration, in µs	$T_{FFT}/4$	0.8	1.6	3.2
T_{GI2}: Training symbol GI duration, in	$T_{FFT}/2$	1.6	3.2	6.4

27 IEEE Std 802.11-2016 Adapted and reprinted with permission from IEEE. Copyright IEEE 2016. All rights reserved.

Parameter	Value	20 MHz Channel Bandwidth	10 MHz Channel Bandwidth	5 MHz Channel Bandwidth
T_{SYM}: Symbol interval, in µs	$T_{GI} + T_{FFT}$	4	8	16
T_{SHORT}: L-STF duration, in µs	$10 \times T_{FFT}/4$	8	16	32
T_{LONG}: L-LTF duration, in µs	$T_{GI2} + 2 \times T_{FFT}$	8	16	32

Note The standard refers to operation at 10 MHz as “half-clocked” and operation at 5 MHz as “quarter-clocked”.

PPDU

The physical layer (PHY) protocol data unit (PPDU) is the complete physical layer convergence procedure (PLCP) frame, including PLCP headers, MAC headers, the MAC data field, and the MAC and PLCP trailers.

Version History

Introduced in R2015b

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`wlanDMGConfig` | `wlanHESUConfig` | `wlanHEMUConfig` | `wlanHETBConfig` | `wlanHTConfig` | `wlanS1GConfig` | `wlanVHTConfig`

Functions

`transmitTime` | `wlanNonHTData` | `wlanNonHTDataRecover` | `wlanNonHTOFDMInfo` | `wlanWaveformGenerator`

Apps

WLAN Waveform Generator

Topics

“Packet Size and Duration Dependencies”

wlanPCAPWriter

PCAP or PCAPNG file writer of WLAN MAC packets

Description

The `wlanPCAPWriter` object writes generated and recovered WLAN medium access control (MAC) packets to a packet capture (PCAP) or packet capture next generation (PCAPNG) file (`.pcap` or `.pcapng`, respectively).

Creation

Syntax

```
obj = wlanPCAPWriter
obj = wlanPCAPWriter(Name,Value)
```

Description

`obj = wlanPCAPWriter` creates a default WLAN PCAP or PCAPNG file writer object that writes WLAN MAC packets to a PCAP or PCAPNG file, respectively.

`obj = wlanPCAPWriter(Name,Value)` sets "Properties" on page 4-187 using one or more name-value pairs. Enclose each property name in quotes. For example, `('FileExtension','pcapng')` sets the extension of the file as `.pcapng`.

Properties

Note The `wlanPCAPWriter` object does not overwrite the existing PCAP or PCAPNG file. During each call of this object, specify a unique PCAP or PCAPNG file name.

`FileName` — Name of the PCAP or PCAPNG file
`'wlanCapture'` (default) | character row vector | string scalar

Name of the PCAP or PCAPNG file, specified as a character row vector or a string scalar.

Data Types: `char` | `string`

`ByteOrder` — Byte order
`'little-endian'` (default) | `'big-endian'`

Byte order, specified as `'little-endian'` or `'big-endian'`.

Data Types: `char` | `string`

`FileExtension` — Type of file
`'pcap'` (default) | `'pcapng'`

Type of file, specified as `'pcap'` or `'pcapng'`.

Data Types: `char` | `string`

`FileComment` — Comment for PCAPNG file
`''` (default) | character vector | string scalar

Comment for the PCAPNG file, specified as a character vector or a string scalar.

Data Types: `char` | `string`

`Interface` — Name of interface on which WLAN packets are captured
`'WLAN'` (default) | character vector | string scalar

Name of the interface on which WLAN packets are captured, specified as a character vector or a string scalar.

Data Types: `char` | `string`

`RadiotapPresent` — Flag to indicate presence of radiotap
`false` or `0` (default) | `true` or `1`

Flag to indicate the presence of radiotap, specified as a `logical` `1` (`true`) or `0` (`false`).

Data Types: `logical`

`PCAPWriter` — PCAP or PCAPNG file writer object
`pcapWriter` object | `pcapngWriter` object

PCAP or PCAPNG file writer object, specified as `pcapWriter` or `pcapngWriter` object.

Object Functions

Specific to This Object

write Write protocol packet data to PCAP or PCAPNG file

Examples

Write WLAN MAC Packet to PCAP File

Create a WLAN PCAP file writer object, specifying the name of the PCAP file.

```
pcapObj = wlanPCAPWriter('FileName','wlanExample', ...
    'FileExtension','pcap');
```

Generate a WLAN MAC packet of type QoS Data.

```
macConfig = wlanMACFrameConfig('FrameType','QoS Data');
payload ='00576000103afffe80';
mpdu = wlanMACFrame(payload,macConfig);
```

Write the WLAN MAC packet to the PCAP file.

```
timestamp = 0; % Number of microseconds
write(pcapObj,mpdu,timestamp);
```

Write WLAN MAC Packet to PCAPNG File

Create a WLAN PCAP file writer object, specifying the name of the PCAPNG file.

```
pcapObj = wlanPCAPWriter('FileName','wlanExample2', ...
    'FileExtension','pcapng');
```

Generate a WLAN MAC packet of type QoS Data.

```
macConfig = wlanMACFrameConfig('FrameType','QoS Data');
payload = '00576000103afffe80';
mpdu = wlanMACFrame(payload,macConfig,'OutputFormat','bits');
```

Write the WLAN MAC packet to the PCAPNG format file.

```
timestamp = 12800000; % Number of microseconds
write(pcapObj,mpdu,timestamp,'PacketFormat','bits');
```

Write WLAN MAC Packet to PCAPNG File Using PCAP File Writer

Create a PCAPNG file writer object, specifying the name of the PCAPNG file.

```
pcapObj = pcapngWriter('FileName','wlanExample3', ...
    'FileComment','This is a sample file');
```

Create a WLAN PCAP file writer object, specifying the PCAP file writer and the presence of radiotap header.

```
wlanPCAP = wlanPCAPWriter('PCAPWriter',pcapObj,'RadiotapPresent', ...
    true);
```

Generate a WLAN MAC packet of type QoS Data.

```
macConfig = wlanMACFrameConfig('FrameType','QoS Data');
payload = '00576000103afffe80';
mpdu = hex2dec(wlanMACFrame(payload,macConfig));
```

Write the WLAN MAC packet to the PCAPNG file.

```
radiotapBytes = [0 0 24 0 2 0 40 0 16 3 0 0 2 192 0 0 0 0 63 1 19 0 0 0];
timestamp = 18912345; % Number of microseconds
write(wlanPCAP,mpdu,timestamp,'Radiotap',radiotapBytes, ...
    'PacketComment','This is the first packet');
```

Version History

Introduced in R2021a

References

[1] Tuexen, M. “PCAP Next Generation (Pcapng) Capture File Format.” 2020. https://www.ietf.org/.

[2] “Radiotap - Introduction.” Accessed May 20, 2020. https://www.radiotap.org/.

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

[4] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`pcapWriter` | `pcapngWriter`

wlanS1GConfig

Configure S1G transmission

Description

The `wlanS1GConfig` object is a sub-1-GHz-format (S1G-format) configuration object for the WLAN S1G packet format.

Creation

Syntax

```
cfgS1G = wlanS1GConfig
cfgS1G = wlanS1GConfig(Name,Value)
```

Description

`cfgS1G = wlanS1GConfig` creates a configuration object that initializes parameters for an IEEE 802.11 S1G-format "PPDU" on page 4-199.

`cfgS1G = wlanS1GConfig(Name,Value)` sets properties using one or more name-value pair arguments. Enclose each property name in quotation marks. For example, `wlanS1GConfig('ChannelBandwidth','CBW4','STBC',true)` specifies an S1G format with a channel bandwidth of 4 MHz and space-time block coding enabled.

Properties

`ChannelBandwidth` — Channel bandwidth of PPDU transmission
`'CBW2'` (default) | `'CBW1'` | `'CBW4'` | `'CBW8'` | `'CBW16'`

Channel bandwidth of PPDU transmission, specified as one of these values:

- `'CBW1'` – Channel bandwidth of 1 MHz
- `'CBW2'` – Channel bandwidth of 2 MHz
- `'CBW4'` – Channel bandwidth of 4 MHz
- `'CBW8'` – Channel bandwidth of 8 MHz
- `'CBW16'` – Channel bandwidth of 16 MHz

Data Types: `char` | `string`

`Preamble` — Preamble type
`'Short'` (default) | `'Long'`

Preamble type, specified as `'Short'` or `'Long'`.

Dependencies

This property applies only when you set the `ChannelBandwidth` property to a value other than `'CBW1'`.

Data Types: `char` | `string`

`NumUsers` — Number of users

`1` (default) | `2` | `3` | `4`

Number of users, specified as `1`, `2`, `3`, or `4`.

Data Types: `double`

`UserPositions` — User positions

`[0 1]` (default) | vector of integers

User positions, specified as a 1-by-`NumUsers` vector of integers in the interval [0, 3] in strictly increasing order.

Example: `[0 2 3]` specifies the positions for three users. The first user occupies position 0, the second user occupies position 2, and the third user occupies position 3.

Dependencies

This property applies only when you specify the `NumUsers` property as a value greater than `1`.

Data Types: `double`

`NumTransmitAntennas` — Number of transmit antennas

`1` (default) | `2` | `3` | `4`

Number of transmit antennas, specified as `1`, `2`, `3`, or `4`.

Data Types: `double`

`NumSpaceTimeStreams` — Number of space-time streams

`1` (default) | integer in the interval [1, 4] | row vector of integers

Number of space-time streams in the transmission, specified as a 1-by-`NumUsers` vector of integers in the interval [1, 4].

Data Types: `double`

`SpatialMapping` — Spatial mapping scheme

`'Direct'` (default) | `'Hadamard'` | `'Fourier'` | `'Custom'`

Spatial mapping scheme, specified as `'Direct'`, `'Hadamard'`, `'Fourier'`, or `'Custom'`.

Dependencies

The default value, `'Direct'`, applies only when you set the `NumTransmitAntennas` and `NumSpaceTimeStreams` properties to the same value.

Data Types: `char` | `string`

`SpatialMappingMatrix` — Spatial mapping matrix

`1` (default) | complex-valued scalar | complex-valued matrix | complex-valued 3-D array

Spatial mapping matrix, specified as one of these values:

- A complex-valued scalar. This value applies to all the subcarriers.
- A complex-valued matrix of size N_{STS} -by-N_T, where:
  - N_{STS} is the number of space-time streams;
  - N_T is the number of transmit antennas.

  In this case, the spatial mapping matrix applies to all the subcarriers.
- A complex-valued 3-D array of size N_{ST}-by-N_{STS}-by-N_T, where N_{ST} is the number of occupied subcarriers. The value of N_{ST} is the sum of the occupied data and pilot subcarriers. The `ChannelBandwidth` property determines the value of N_{ST}. In this case, each occupied subcarrier has its own spatial mapping matrix.

  This table shows the `ChannelBandwidth` setting and the corresponding N_{ST}:

`ChannelBandwidth`	Number of Occupied Subcarriers, N_{ST}	Number of Data Subcarriers	Number of Pilot Subcarriers
`'CBW1'`	26	24	2
`'CBW2'`	56	52	4
`'CBW4'`	114	108	6
`'CBW8'`	242	234	8
`'CBW16'`	484	468	16

Use this property to rotate and scale the output vector of the constellation mapper. The spatial mapping matrix is used for beamforming and mixing space-time streams over the transmit antennas. For more information, see Section 20.3.11.11.2 of [1]. The calling function normalizes the spatial mapping matrix for each subcarrier.

Example: `[0.5 0.3; 0.4 0.4; 0.5 0.8]` represents a spatial mapping matrix with three space-time streams and two transmit antennas.

Dependencies

This property applies only when you set the `SpatialMapping` property to `'Custom'`.

Data Types: `double`
Complex Number Support: Yes

`Beamforming` — Enable beamforming in a long-preamble packet

`true` or `1` (default) | `false` or `0`

Enable beamforming in a long-preamble packet, specified as a numeric or logical value of `1` (`true`) or `0` (`false`). To apply a beamforming steering matrix, set this property to `1` (`true`). The `SpatialMappingMatrix` property specifies the beamforming steering matrix.

Dependencies

This property applies only when all of these conditions are satisfied:

- The `Preamble` property is set to `'Long'`.
- The `NumUsers` property is set to `1`.
- The `SpatialMapping` property is set to `'Custom'`.

Data Types: `logical`

STBC — Enable space-time block coding

`false` or `0` (default) | `true` or `1`

Enable space-time block coding (STBC) of the PPDU data field for all users, specified as a numeric or logical value of `1` (`true`) or `0` (`false`). STBC transmits multiple copies of the data stream across assigned antennas.

- When you set this property to `0` (`false`), STBC is not applied to the data field. The number of space-time streams is equal to the number of spatial streams.
- When you set this property to `1` (`true`), STBC is applied to the data field. The number of space-time streams is twice the number of spatial streams.

For more information, see Section 22.3.10.9.4 of [2].

Dependencies

This property applies only when the `NumUsers` property is `1`.

Data Types: `logical`

MCS — Modulation and coding scheme

`0` (default) | integer in the interval [0, 10] | vector of integers

Modulation and coding scheme specified as one of these values:

- an integer in the interval [0, 10], applicable when the `NumUsers` property is `1`
- a 1-by-`NumUsers` vector of integers in the interval [0, 10].

This table shows the modulation type and coding rate for each valid value of `MCS`:

MCS	Modulation	Coding Rate
0	Binary phase-shift keying (BPSK)	1/2
1	Quadrature phase-shift keying (QPSK)	1/2
2	QPSK	3/4
3	16-point quadrature amplitude modulation (16-QAM)	1/2
4	16-QAM	3/4
5	64-QAM	2/3
6	64-QAM	3/4
7	64-QAM	5/6
8	256-QAM	3/4
9	256-QAM	5/6
10	BPSK	1/2

Data Types: `double`

ChannelCoding — FEC coding type

`'BCC'` (default)

This property is read-only.

Forward error correction (FEC) coding type, specified as `'BCC'`. The `wlanS1GConfig` object supports only binary convolutional coding (BCC).

Data Types: `char`

`APEPLength` — APEP length

`256` (default) | nonnegative integer | vector of nonnegative integers

Aggregated MPDU (A-MPDU) pre-end-of-frame (pre-EOF) padding (APEP) length, in bytes.

- When the `NumUsers` property is `1`, specify this property as a nonnegative integer in the interval $[0, 2^{16} - 1]$.
- When the `NumUsers` property is a value other than `1`, specify this property as a 1-by-`NumUsers` vector of integers in the interval $[0, 2^{16} - 1]$.
- For a null data packet (NDP), set this property to `0`.

The `wlanS1GConfig` uses this property to determine the number of OFDM symbols in the data field. For more information, see Table 22-1 of [2].

Note This object supports only aggregated data transmission.

Data Types: `double`

`PSDULength` — PSDU length

nonnegative integer

This property is read-only.

Physical layer convergence procedure (PLCP) service data unit (PSDU) length, in bytes, specified as an integer. The `wlanS1GConfig` object calculates this property internally based on other properties.

Data Types: `double`

`GuardInterval` — Guard interval (cyclic prefix) duration

`'Long'` (default) | `'Short'`

Guard interval (cyclic prefix) duration for the data field within a packet, specified as one of these values:

- `'Long'` – Guard interval duration of 800 ns
- `'Short'` – Guard interval duration of 400 ns

Note For S1G format, the first OFDM symbol within the data field always has a long guard interval, even when you set this property to `'Short'`.

Data Types: `char` | `string`

`GroupID` — Group identification number

`1` (default) | integer in the interval [1, 62]

Group identification number, specified as an integer in the interval [1, 62]. The group identification number is signaled during a multiuser transmission.

Dependencies

This property applies only when you set the `Preamble` property to `'Long'` and the `NumUsers` property to a value greater than 1.

Data Types: `double`

`PartialAID` — Abbreviated indication of PSDU recipients

`37` (default) | integer in the interval [0, 511]

Abbreviated indication of the PSDU recipients, specified as an integer in the interval [0, 511].

- When you set the `UplinkIndication` property to `1` (`true`), the partial identification number is the last nine bits of the basic service set identifier (BSSID). This property must be an integer in the interval [0, 511].
- When you set the `UplinkIndication` property to `0` (`false`), the partial identification number is an identifier that combines the association ID with the BSSID of its serving AP. This property must be an integer in the interval [0, 63].

For more information, see Table 22-1 of [2].

Data Types: `double`

`UplinkIndication` — Uplink indication

`false` or `0` (default) | `true` or `1`

Uplink indication, specified as a numeric or logical value of `1` (`true`) or `0` (`false`). To indicate that the PPDU is sent on a downlink transmission, set this property to `0` (`false`). To indicate that the PPDU is sent on an uplink transmission, set this property to `1` (`true`).

Dependencies

This property applies only when you set the `ChannelBandwidth` property to a value other than `'CBW1'` and the `NumUsers` property to `1`.

Data Types: `logical`

`Color` — AP color identifier

`0` (default) | integer in the interval [0, 7]

Access point (AP) color identifier, specified as an integer in the interval [0, 7]. An AP includes a color number for the basic service set (BSS). An S1G station (STA) can use the color setting to determine if the transmission is within a BSS with which it is associated. The STA can terminate the reception process for transmissions received from a BSS with which it is not associated.

Dependencies

This property applies only when these conditions are satisfied:

- The `ChannelBandwidth` property is not `'CBW1'`.
- The `NumUsers` property is `1`.
- The `UplinkIndication` property is `0` (`false`).

Data Types: `double`

`TravelingPilots` — Enable traveling pilots

`false` or `0` (default) | `true` or `1`

Enable traveling pilots, specified as a numeric or logical value of `1` (`true`) or `0` (`false`). To specify non-constant pilot locations, set this property to `1` (`true`). Traveling pilots allow a receiver to track a changing channel due to Doppler spread.

Data Types: `logical`

`ResponseIndication` — Response indication type

`'None'` (default) | `'NDP'` | `'Normal'` | `'Long'`

Response indication type, specified as `'None'`, `'NDP'`, `'Normal'`, or `'Long'`. This information is used to indicate the presence and type of frame that will be sent a short interframe space (SIFS) after the current frame transmission. The value to which you set this property sets the response indication field, which is transmitted in these fields:

- The SIG2 field of the S1G_SHORT preamble
- The SIG-A-2 field of the S1G_LONG preamble
- The SIG field of the S1G_1M preamble

Data Types: `char` | `string`

`RecommendSmoothing` — Recommend smoothing for channel estimation

`true` or `1` (default) | `false` or `0`

Recommend smoothing for channel estimation, specified as a numeric or logical value of `1` (`true`) or `0` (`false`).

- If the frequency profile does not vary across the channel, the receiver sets this property to `1` (`true`). In this case, frequency-domain smoothing is recommended as part of channel estimation.
- If the frequency profile varies across the channel, the receiver sets this property to `0` (`false`). In this case, frequency-domain smoothing is not recommended as part of channel estimation.

Data Types: `logical`

Object Functions

packetFormat	WLAN packet format
transmitTime	Packet transmission time

Examples

Create wlanS1GConfig Object for Single User

Create an S1G configuration object with default settings for a single user. Override the default by specifying a 4 MHz channel bandwidth and short preamble configuration.

```
cfgS1G = wlanS1GConfig;
cfgS1G.ChannelBandwidth = 'CBW4';
cfgS1G.Preamble = 'Short';
cfgS1G
```

```
cfgS1G =
  wlanS1GConfig with properties:
```

```
       ChannelBandwidth: 'CBW4'
              Preamble: 'Short'
              NumUsers: 1
    NumTransmitAntennas: 1
    NumSpaceTimeStreams: 1
         SpatialMapping: 'Direct'
                   STBC: 0
                    MCS: 0
              APEPLength: 256
          GuardInterval: 'Long'
             PartialAID: 37
       UplinkIndication: 0
                  Color: 0
          TravelingPilots: 0
     ResponseIndication: 'None'
     RecommendSmoothing: 1

   Read-only properties:
          ChannelCoding: 'BCC'
             PSDULength: 261
```

Create wlanS1GConfig Object for Two Users

Create an S1G configuration object that assigns a 2 MHz bandwidth and two users. Use a combination of Name,Value pairs and in-line initialization to change default settings. In vector-valued properties, each element applies to a specific user.

```
cfgMU = wlanS1GConfig('ChannelBandwidth','CBW2', ...
    'Preamble','Long', ...
    'NumUsers',2, ...
    'GroupID',2, ...
    'NumTransmitAntennas', 2);
cfgMU.NumSpaceTimeStreams = [1 1];
cfgMU.MCS = [4 8];
cfgMU.APEPLength = [1024 2048];
cfgMU
```

```
cfgMU =
  wlanS1GConfig with properties:

       ChannelBandwidth: 'CBW2'
              Preamble: 'Long'
              NumUsers: 2
         UserPositions: [0 1]
    NumTransmitAntennas: 2
    NumSpaceTimeStreams: [1 1]
         SpatialMapping: 'Direct'
                   MCS: [4 8]
              APEPLength: [1024 2048]
          GuardInterval: 'Long'
                GroupID: 2
          TravelingPilots: 0
     ResponseIndication: 'None'
```

```
   Read-only properties:
          ChannelCoding: 'BCC'
             PSDULength: [1031 2065]
```

`NumUsers` is set to 2 and the user-dependent properties are two-element vectors.

Create WLAN S1G Configuration Object and Return Packet Format

Create an S1G configuration object with default property values.

```
cfgS1G = wlanS1GConfig;
```

Compute and display the packet format. The default properties specify a transmission with short preamble.

```
format = packetFormat(cfgS1G);
disp(format)
```

```
S1G-Short
```

Now create an S1G configuration object, specifying a long preamble.

```
cfgS1GLongPreamble = wlanS1GConfig('Preamble','Long');
```

Compute and display the packet format.

```
format = packetFormat(cfgS1GLongPreamble);
disp(format)
```

```
S1G-Long
```

More About

PPDU

The physical layer (PHY) protocol data unit (PPDU) is the complete physical layer convergence procedure (PLCP) frame, including PLCP headers, MAC headers, the MAC data field, and the MAC and PLCP trailers.

Version History

Introduced in R2016b

References

[1] IEEE Std 802.11-2012. "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications." IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements.

[2] IEEE 802.11ac-2013. "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. — Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz." IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations: After the first use of this object for an S1G MU-MIMO transmission, you cannot change the number of elements in any of these properties.

- `UserPositions`
- `NumSpaceTimeStreams`
- `MCS`
- `APEPLength`

See Also

Objects

`wlanDMGConfig` | `wlanHESUConfig` | `wlanHEMUConfig` | `wlanHETBConfig` | `wlanHTConfig` | `wlanNonHTConfig` | `wlanVHTConfig`

Functions

`transmitTime` | `wlanS1GDemodulate` | `wlanS1GOFDMInfo` | `wlanWaveformGenerator`

Apps

WLAN Waveform Generator

Topics

“Packet Size and Duration Dependencies”

wlanTGacChannel

Filter signal through 802.11ac multipath fading channel

Description

The `wlanTGacChannel` System object filters an input signal through an 802.11ac (TGac) multipath fading channel.

The fading processing assumes the same parameters for all N_T-by-N_R links of the TGac channel, where N_T is the number of transmit antennas and N_R is the number of receive antennas. Each link comprises all multipaths for that link.

To filter an input signal using a TGac multipath fading channel:

1. Create the `wlanTGacChannel` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
tgac = wlanTGacChannel
tgac = wlanTGacChannel(Name,Value)
```

Description

`tgac = wlanTGacChannel` creates a TGac fading channel System object, `tgac`. This object filters a real or complex input signal through the TGac channel to obtain the channel-impaired signal.

`tgac = wlanTGacChannel(Name,Value)` creates a TGac channel object, `tgac`, and sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `wlanTGacChannel('NumReceiveAntennas',2,'SampleRate',10e6)` creates a TGac channel with two receive antennas and a 10 MHz sample rate.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`SampleRate` — Sample rate of the input signal
`80e6` (default) | positive scalar

Sample rate of the input signal in Hz, specified as a positive scalar.

Data Types: `double`

`DelayProfile` — Delay profile model
`'Model-B'` (default) | `'Model-A'` | `'Model-C'` | `'Model-D'` | `'Model-E'` | `'Model-F'`

Delay profile model, specified as `'Model-A'`, `'Model-B'`, `'Model-C'`, `'Model-D'`, `'Model-E'`, or `'Model-F'`. To enable the `FluorescentEffect` property, select either `'Model-D'` or `'Model-E'`.

The table summarizes the models properties before the bandwidth reduction factor.

Parameter	Model					
	A	B	C	D	E	F
Breakpoint distance (m)	5	5	5	10	20	30
RMS delay spread (ns)	0	15	30	50	100	150
Maximum delay (ns)	0	80	200	390	730	1050
Rician K-factor (dB)	0	0	0	3	6	6
Number of taps	1	9	14	18	18	18
Number of clusters	1	2	2	3	4	6

The number of clusters represents the number of independently modeled propagation paths.

Data Types: `char` | `string`

`ChannelBandwidth` — Channel bandwidth
`'CBW80'` (default) | `'CBW20'` | `'CBW40'` | `'CBW160'`

Channel bandwidth, specified as `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`. The default is `'CBW80'`, which corresponds to an 80 MHz channel bandwidth.

Data Types: `char` | `string`

`CarrierFrequency` — RF carrier frequency
`5.25e9` (default) | positive scalar

RF carrier frequency in Hz, specified as a positive scalar.

Data Types: `double`

`EnvironmentalSpeed` — Speed of the scatterers
`0.089` (default) | positive scalar

Speed of the scatterers in km/h, specified as a positive scalar.

Data Types: `double`

`TransmitReceiveDistance` — Distance between transmitter and receiver
`3` (default) | positive scalar

Distance between the transmitter and receiver in meters, specified as a positive scalar.

`TransmitReceiveDistance` is used to compute the path loss, and to determine whether the channel has a line of sight (LOS) or non line of sight (NLOS) condition. The path loss and standard deviation of shadow fading loss depend on the separation between the transmitter and the receiver.

Data Types: `double`

`NormalizePathGains` — Normalize path gains
`true` or `1` (default) | `false` or `0`

Normalize path gains, specified as a numeric or logical `1` (`true`) or `0` (`false`). To normalize the fading processes such that the total power of the path gains, averaged over time, is 0 dB, set this property to `1` (`true`). Otherwise, set this property to `0` (`false`).

Data Types: `logical`

`UserIndex` — User index for single or multi-user scenario
`0` (default) | positive integer

User index, specified as a nonnegative integer. If the property is set to `0`, the angles of arrival and departure from the TGn channel model are used in the calculation of the spatial correlation matrix. If the property is set to a positive integer, pseudorandom offsets are applied to the TGn angles of arrival and departure before the calculation of the spatial correlation matrix. For more details, see the section on “MIMO Enhancements” on page 4-212.

Data Types: `double`

`TransmissionDirection` — Transmission direction
`'Downlink'` (default) | `'Uplink'`

Transmission direction of the active link, specified as either `'Downlink'` or `'Uplink'`.The default value, `'Downlink'`, specifies transmission from an access point to a user station.

Data Types: `char` | `string`

`NumTransmitAntennas` — Number of transmit antennas
`1` (default) | positive integer

Number of transmit antennas, specified as a positive integer.

Data Types: `double`

`TransmitAntennaSpacing` — Distance between transmit antenna elements
`0.5` (default) | positive scalar

Distance between transmit antenna elements, specified as a positive scalar expressed in wavelengths.

`TransmitAntennaSpacing` supports uniform linear arrays only.

Dependencies

To enable this property, set the `NumTransmitAntennas` property to a value greater than `1`.

Data Types: `double`

`NumReceiveAntennas` — Number of receive antennas
`1` (default) | positive integer

Number of receive antennas, specified as a positive integer.

Data Types: `double`

`ReceiveAntennaSpacing` — Distance between receive antenna elements
`0.5` (default) | positive scalar

Distance between receive antenna elements, specified as a positive scalar expressed in wavelengths.

`ReceiveAntennaSpacing` supports uniform linear arrays only.

Dependencies

To enable this property, set the `NumReceiveAntennas` property to a value greater than `1`.

Data Types: `double`

`LargeScaleFadingEffect` — Large-scale fading effects

`'None'` (default) | `'Pathloss'` | `'Shadowing'` | `'Pathloss and shadowing'`

Large-scale fading effects applied in the channel, specified as `'None'`, `'Pathloss'`, `'Shadowing'`, or `'Pathloss and shadowing'`.

Data Types: `char` | `string`

`FluorescentEffect` — Fluorescent effect

`true` or `1` (default) | `false` or `0`

Fluorescent effect, specified as a numeric or logical `1` (`true`) or `0` (`false`). To include Doppler effects from fluorescent lighting, set this property to `1` (`true`).

Dependencies

To enable this property, set the `DelayProfile` property to `'Model-D'` or `'Model-E'`.

Data Types: `logical`

`PowerLineFrequency` — Power line frequency

`'60Hz'` (default) | `'50Hz'`

Power line frequency in Hz, specified as `'50Hz'` or `'60Hz'`.

The power line frequency is 60 Hz in the United States and 50 Hz in Europe.

Dependencies

To enable this property, set the `FluorescentEffect` property to `1` (`true`) and the `DelayProfile` property to `'Model-D'` or `'Model-E'`.

Data Types: `char` | `string`

`NormalizeChannelOutputs` — Normalize channel outputs

`true` or `1` (default) | `false` or `0`

Normalize channel outputs by the number of receive antennas, specified as a numeric or logical `1` (`true`) or `0` (`false`).

Data Types: `logical`

`ChannelFiltering` — Enable channel filtering

`true` or `1` (default) | `false` or `0`

Enable channel filtering, specified as a numeric or logical `1` (`true`) or `0` (`false`). To enable channel filtering, set this property to `1` (`true`). To disable channel filtering, set this property to `0` (`false`).

Note If you set this property to `0` (`false`), the `step` object function does not accept an input signal. In this case, the `NumSamples` and `SampleRate` properties determine the duration of the fading process realization.

Data Types: `logical`

`NumSamples` — Number of time-domain samples

`320` (default) | positive integer

Number of time-domain samples used to get path gain samples, specified as a positive integer.

Dependencies

To enable this property, set the `ChannelFiltering` property to `0` (`false`).

Data Types: `double`

`OutputDataType` — Data type of impaired signal

`'double'` (default) | `'single'`

Data type of impaired signal, specified as one of these values:

- `'double'` – Return the `pathGains` output as a double-precision matrix
- `'single'` – Return the `pathGains` output as a single-precision matrix

Dependencies

To enable this property, set the `ChannelFiltering` property to `0` (`false`).

Data Types: `char` | `string`

`RandomStream` — Source of random number stream

`'Global stream'` (default) | `'mt19937ar with seed'`

Source of random number stream, specified as `'Global stream'` or `'mt19937ar with seed'`.

If you set this property to `'Global stream'`, the current global random number stream is used for random number generation. In this case, the `reset` function resets the filters and creates a new channel realization.

If you set this property to `'mt19937ar with seed'`, the mt19937ar algorithm generates random numbers. In this case, the `reset` function also reinitializes the random number stream to the value of the `Seed` property.

Note The random numbers of the channel components are distributed as follows:

- The random phase of the Doppler component due to fluorescent lights is uniformly distributed. See equation 27 of *TGn Channel Models* for more information.
- In multi-user scenarios using the TGac on page 4-201, TGah on page 4-214, or TGax on page 4-227 channel models, the per-user angle-of-arrival (AoA) and angle-of-departure (AoD) rotations discussed in the “MIMO Enhancements” on page 4-238 section are uniformly distributed.
- The fading samples are generated from a normally-distributed complex uncorrelated Gaussian process with zero mean and unit variance in discrete time.

Data Types: `char` | `string`

Seed — Initial seed of mt19937ar random number stream

73 (default) | nonnegative integer

Initial seed of an mt19937ar random number stream, specified as a nonnegative integer. The `Seed` property reinitializes the mt19937ar random number stream in the `reset` function.

Dependencies

To enable this property, set the `RandomStream` property to `'mt19937ar with seed'`.

Data Types: `double`

PathGainsOutputPort — Enable path gain output

`false` or `0` (default) | `true` or `1`

Enable path gain output computation, specified as a numeric or logical `1` (`true`) or `0` (`false`).

Data Types: `logical`

Usage

Syntax

```
y = tgac(x)
[y,pathGains] = tgac(x)
pathGains = tgac(x)
```

Description

`y = tgac(x)` filters the input signal `x` through the TGac fading channel defined by the `wlanTGacChannel` System object `tgac` and returns the result in `y`.

`[y,pathGains] = tgac(x)` also returns in `pathGains` the TGac channel path gains of the underlying fading process.

This syntax applies when you set the `PathGainsOutputPort` property to `1` (`true`).

`pathGains = tgac(x)` returns the path gains. The `NumSamples` property determines the duration of the fading process.

This syntax applies when you set the `ChannelFiltering` property to `0` (`false`).

Input Arguments

x — Input signal

complex matrix

Input signal, specified as a real or complex N_S-by-N_T matrix, where:

- N_S is the number of samples.
- N_T is the number of transmit antennas and must be equal to the `NumTransmitAntennas` property value.

Data Types: `single` | `double`

Complex Number Support: Yes

Output Arguments

y — Output signal
complex matrix

Output signal, returned as an N_S-by-N_R complex matrix, where:

- N_S is the number of samples.
- N_R is the number of receive antennas and is equal to the `NumReceiveAntennas` property value.

Data Types: `single` | `double`

`pathGains` — Path gains of the fading process
complex array

Path gains of the fading process, returned as an N_S-by-N_P-by-N_T-by-N_R complex array, where:

- N_S is the number of samples.
- N_P is the number of resolvable paths, that is, the number of paths defined for the case specified by the `DelayProfile` property.
- N_T is the number of transmit antennas and is equal to the `NumTransmitAntennas` property value.
- N_R is the number of receive antennas and is equal to the `NumReceiveAntennas` property value.

Data Types: `single` | `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to wlanTGacChannel

info Characteristic information about multipath fading channels

Common to All System Objects

step Run System object algorithm
release Release resources and allow changes to System object property values and input characteristics
reset Reset internal states of System object

Note If the `RandomStream` property of the System object is set to `'Global stream'`, the `reset` function resets the filters only. If you set `RandomStream` to `'mt19937ar with seed'`, the `reset` function also reinitializes the random number stream to the value of the `Seed` property.

Examples

Transmit VHT Waveform Through TGac Channel

Generate a VHT waveform and pass it through a TGac SISO channel. Display the spectrum of the resultant signal.

Set the channel bandwidth and the corresponding sample rate.

```
bw = 'CBW80';
fs = 80e6;
```

Generate a VHT waveform.

```
cfg = wlanVHTConfig;
txSig = wlanWaveformGenerator(randi([0 1],1000,1),cfg);
```

Create a TGac SISO channel with path loss and shadowing enabled.

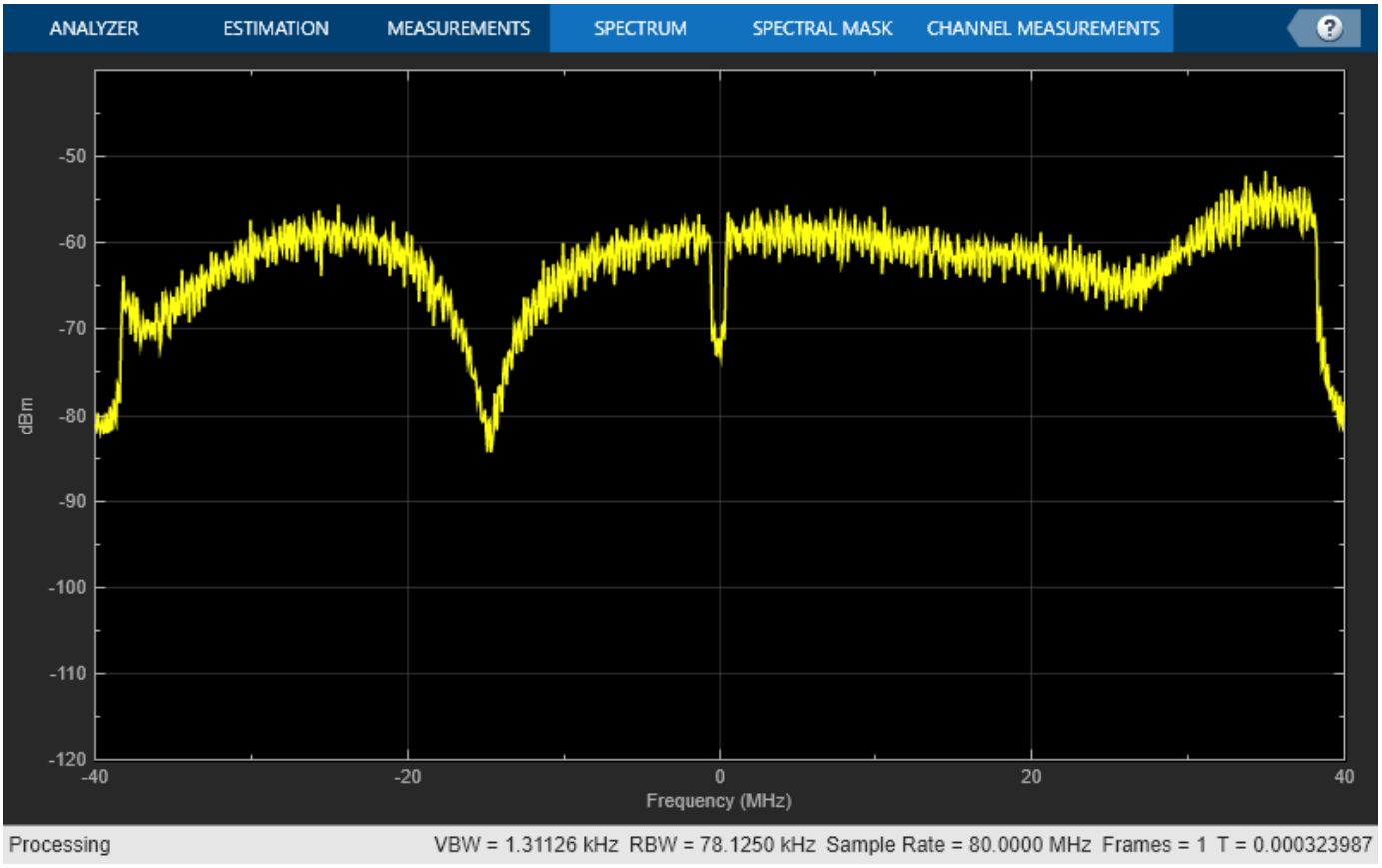
```
tgacChan = wlanTGacChannel('SampleRate',fs,'ChannelBandwidth',bw, ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
```

Pass the VHT waveform through the channel.

```
rxSig = tgacChan(txSig);
```

Plot the spectrum of the received waveform.

```
saScope = spectrumAnalyzer(SampleRate=fs,YLimits=[-120 -40]);
saScope(rxSig)
```



Because path loss and shadowing are enabled, the mean received power across the spectrum is approximately –60 dBm.

Transmit VHT Waveform Through 4x2 MIMO Channel

Create a VHT waveform with four transmit antennas and two space-time streams.

```
cfg = wlanVHTConfig('NumTransmitAntennas',4,'NumSpaceTimeStreams',2, ...
    'SpatialMapping','Fourier');
txSig = wlanWaveformGenerator([1;0;0;1],cfg);
```

Create a 4x2 MIMO TGac channel and disable large-scale fading effects.

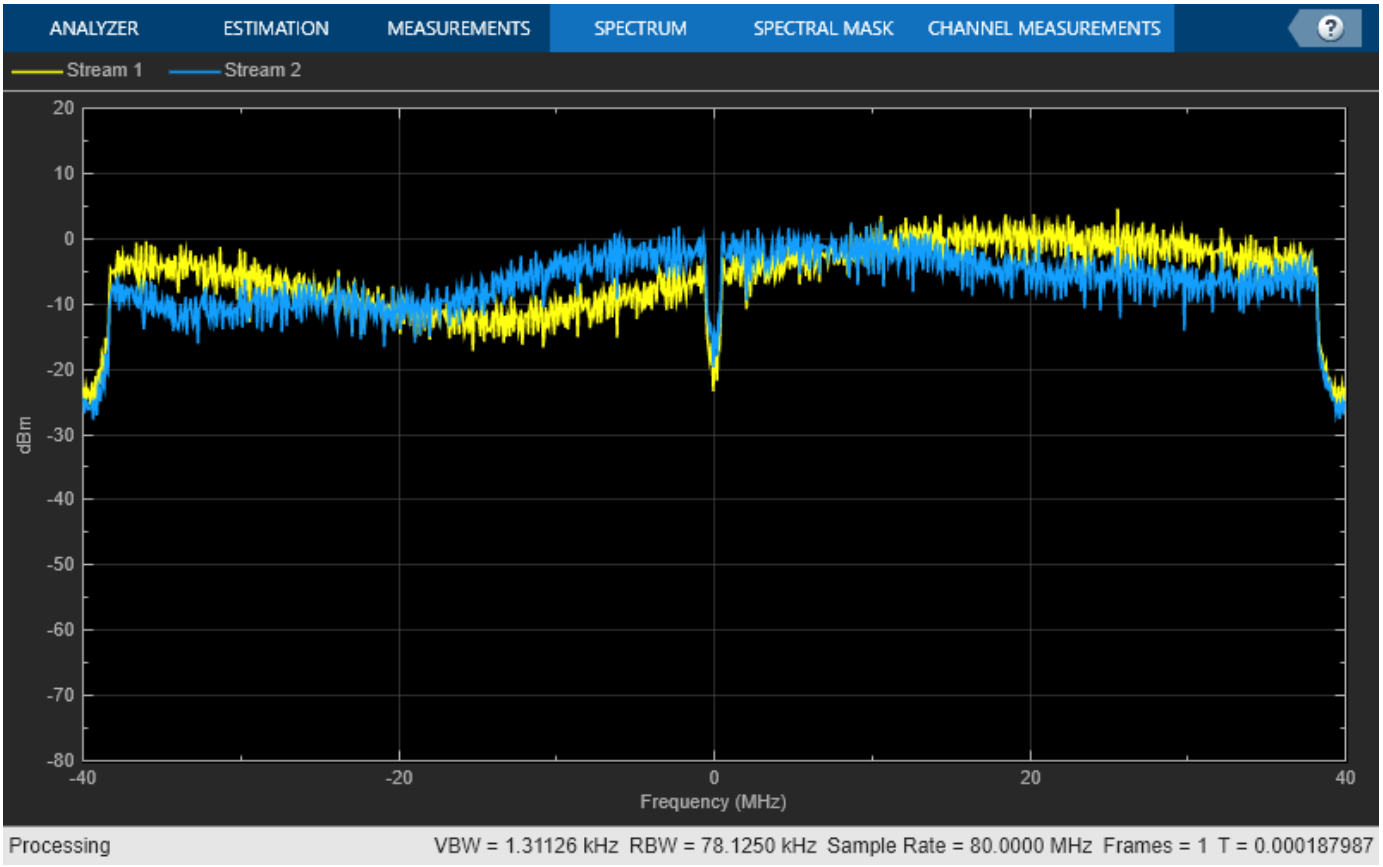
```
tgacChan = wlanTGacChannel('SampleRate',80e6,'ChannelBandwidth','CBW80', ...
    'NumTransmitAntennas',4,'NumReceiveAntennas',2, ...
    'LargeScaleFadingEffect','None');
```

Pass the transmit waveform through the channel.

```
rxSig = tgacChan(txSig);
```

Display the spectrum of the two received space-time streams.

```
saScope = spectrumAnalyzer(SampleRate=80e6, ...
    ShowLegend=true, ...
    ChannelNames={'Stream 1','Stream 2'});
saScope(rxSig)
```



Recover VHT Data from 2x2 MIMO Channel

Transmit a VHT-LTF and a VHT data field through a noisy 2x2 MIMO channel. Demodulate the received VHT-LTF to estimate the channel coefficients. Recover the VHT data and determine the number of bit errors.

Set the channel bandwidth and corresponding sample rate.

```
bw = 'CBW160';
fs = 160e6;
```

Create VHT-LTF and VHT data fields with two transmit antennas and two space-time streams.

```
cfg = wlanVHTConfig('ChannelBandwidth',bw, ...
    'NumTransmitAntennas',2,'NumSpaceTimeStreams',2);
txPSDU = randi([0 1],8*cfg.PSDULength,1);
txLTF = wlanVHTLTF(cfg);
txDataSig = wlanVHTData(txPSDU,cfg);
```

Create a 2x2 MIMO TGac channel.

```
tgacChan = wlanTGacChannel('SampleRate',fs,'ChannelBandwidth',bw, ...
    'NumTransmitAntennas',2,'NumReceiveAntennas',2);
```

Create AWGN channel noise, setting the SNR to 15 dB.

```
chNoise = comm.AWGNChannel('NoiseMethod','Signal to noise ratio (SNR)',...
    'SNR',15);
```

Pass the signals through the TGac channel and noise models.

```
rxLTF = chNoise(tgacChan(txLTF));
rxDataSig = chNoise(tgacChan(txDataSig));
```

Create an AWGN channel for a 160 MHz channel with a 9 dB noise figure. The noise variance, `nVar`, is equal to *kTBF*, where *k* is Boltzmann's constant, *T* is the ambient temperature of 290 K, *B* is the bandwidth (sample rate), and *F* is the receiver noise figure.

```
nVar = 10^((-228.6 + 10*log10(290) + 10*log10(fs) + 9)/10);
rxNoise = comm.AWGNChannel('NoiseMethod','Variance','Variance',nVar);
```

Pass the signals through the receiver noise model.

```
rxLTF = rxNoise(rxLTF);
rxDataSig = rxNoise(rxDataSig);
```

Demodulate the VHT-LTF. Use the demodulated signal to estimate the channel coefficients.

```
dLTF = wlanVHTLTFDemodulate(rxLTF,cfg);
chEst = wlanVHTLTFChannelEstimate(dLTF,cfg);
```

Recover the data and determine the number of bit errors.

```
rxPSDU = wlanVHTDataRecover(rxDataSig,chEst,nVar,cfg);
numErr = biterr(txPSDU,rxPSDU)
```

```
numErr = 0
```

Algorithms

The algorithms used to model the TGac channel are based on those used for the TGn channel and are described in `wlanTGnChannel` and [1]. The changes to support the TGac channel include:

- Increased bandwidth
- Higher-order MIMO
- Multi-user MIMO
- Reduced Doppler

Complete information on the changes required to support TGac channels can be found in [2].

Increased Bandwidth

TGac channels support bandwidths of up to 1.28 GHz, whereas TGn channels have a maximum bandwidth of 40 MHz. By increasing the sampling rate and decreasing the tap spacing of the power delay profile (PDP), the TGn model is used as the basis for TGac. The channel sampling rate is increased by a factor of $2^{\lceil \log_2(W/40) \rceil}$, where W is the bandwidth. The PDP tap spacing is reduced by the same factor.

Bandwidth, *W*	Sampling Rate Expansion Factor	PDP Tap Spacing (ns)
$W \le 40$ MHz	1	10
40 MHz < $W \le 80$ MHz	2	5
80 MHz < $W \le 160$ MHz	4	2.5
160 MHz < $W \le 320$ MHz	8	1.25
320 MHz < $W \le 640$ MHz	16	0.625
640 MHz < $W \le 1280$ MHz	32	0.3125

MIMO Enhancements

The TGn channel model supports no more than 4x4 MIMO, while the TGac model supports 8x8 MIMO.

The TGac model uses per-user angle diversity to support simultaneous communication between multiple user stations and an access point. For each channel realization, the model achieves this by adding pseudorandom offsets to the angles of arrival and departure of each cluster before the calculation of the spatial correlation matrix. Different offsets are chosen for each positive value of the `UserIndex` property. This causes the angles of arrival and departure for each cluster to differ between users.

Changing the `TransmissionDirection` property swaps the angle of arrival of each cluster with its corresponding angle of departure. For more details, see the Appendix of [2].

To see how the `UserIndex` property can be used in an application, see this example: “802.11ax Packet Error Rate Simulation for Uplink Trigger-Based Format”.

Reduced Doppler

Indoor channel measurements indicate that the magnitude of Doppler assumed in the TGn channel model is too high for stationary users. As such, the TGac channel model uses a reduced environment velocity of 0.089 km/hr. This model assumes a coherence time of 800 ms or, equivalently, an RMS Doppler spread of 0.4 Hz for a 5 GHz carrier frequency.

Version History

Introduced in R2015b

References

[1] Erceg, V., L. Schumacher, P. Kyritsi, et al. *TGn Channel Models*. Version 4. IEEE 802.11-03/940r4, May 2004.

[2] Breit, G., H. Sampath, S. Vermani, et al.*TGac Channel Model Addendum*. Version 12. IEEE 802.11-09/0308r12, March 2010.

[3] Kermoal, J. P., L. Schumacher, K. I. Pedersen, P. E. Mogensen, and F. Frederiksen. “A Stochastic MIMO Radio Channel Model with Experimental Validation”. *IEEE Journal on Selected Areas in Communications* 20, No. 6 (August 2002): pp. 1211–1226.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

Objects

`wlanTGayChannel` | `wlanTGaxChannel` | `wlanTGahChannel` | `wlanTGnChannel`

wlanTGahChannel

Filter signal through 802.11ah multipath fading channel

Description

The `wlanTGahChannel` System object filters an input signal through an 802.11ah (TGah) indoor MIMO channel as specified in [1], following the MIMO modeling approach described in [4].

The fading processing assumes the same parameters for all N_T-by-N_R links of the TGah channel, where N_T is the number of transmit antennas and N_R is the number of receive antennas. Each link comprises all multipaths for that link.

To filter an input signal using a TGah multipath fading channel:

1. Create the `wlanTGahChannel` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
tgah = wlanTGahChannel
tgah = wlanTGahChannel(Name,Value)
```

Description

`tgah = wlanTGahChannel` creates a TGah channel System object, `tgah`. This object filters a real or complex input signal through the TGah channel to obtain the channel-impaired signal.

`tgah = wlanTGahChannel(Name,Value)` creates a TGah channel object, `tgah`, and sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `wlanTGahChannel('NumReceiveAntennas',4,'SampleRate',4e6)` creates a TGah channel with four receive antennas and a 4 MHz sample rate.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`SampleRate` — Sample rate of the input signal
`2e6` (default) | positive scalar

Sample rate of the input signal in Hz, specified as a real positive scalar.

Data Types: `double`

`DelayProfile` — Delay profile model
`'Model-B'` (default) | `'Model-A'` | `'Model-C'` | `'Model-D'` | `'Model-E'` | `'Model-F'`

Delay profile model, specified as `'Model-A'`, `'Model-B'`, `'Model-C'`, `'Model-D'`, `'Model-E'`, or `'Model-F'`.

The table summarizes the models properties before the bandwidth reduction factor.

Parameter	Model					
	A	B	C	D	E	F
Breakpoint distance (m)	5	5	5	10	20	30
RMS delay spread (ns)	0	15	30	50	100	150
Maximum delay (ns)	0	80	200	390	730	1050
Rician K-factor (dB)	0	0	0	3	6	6
Number of taps	1	9	14	18	18	18
Number of clusters	1	2	2	3	4	6

The number of clusters represents the number of independently modeled propagation paths.

Data Types: `char` | `string`

`ChannelBandwidth` — Channel bandwidth
`'CBW2'` (default) | `'CBW1'` | `'CBW4'` | `'CBW8'` | `'CBW16'`

Channel bandwidth, specified as `'CBW1'`, `'CBW2'`, `'CBW4'`, `'CBW8'`, or `'CBW16'`. The default is `'CBW2'`, which corresponds to a 2 MHz channel bandwidth.

For channel bandwidths greater than 4 MHz, the TGah channel applies a reduction factor to the multipath spacing of the power delay profile. The reduction factor applied is $2^{\text{ceil}(\log2(BW/4))}$, where BW is the channel bandwidth in MHz. For more information, see *TGac Channel Model Addendum* [3].

Data Types: `char` | `string`

`CarrierFrequency` — RF carrier frequency
`915e6` (default) | positive scalar

RF carrier frequency in Hz, specified as a positive scalar.

Data Types: `double`

`EnvironmentalSpeed` — Speed of the scatterers
`0.089` (default) | positive scalar

Speed of the scatterers in km/h, specified as a positive scalar.

Data Types: `double`

`TransmitReceiveDistance` — Distance between transmitter and receiver
`3` (default) | positive scalar

Distance between the transmitter and receiver in meters, specified as a positive scalar.

`TransmitReceiveDistance` is used to compute the path loss, and to determine whether the channel has a line of sight (LOS) or no line of sight (NLOS) condition. The path loss and standard deviation of shadow fading loss depend on the separation between the transmitter and the receiver.

Data Types: `double`

`NormalizePathGains` — Normalize path gains
`true` or `1` (default) | `false` or `0`

Normalize path gains, specified as a numeric or logical `1` (`true`) or `0` (`false`). To normalize the fading processes such that the total power of the path gains, averaged over time, is 0 dB, set this property to `1` (`true`). Otherwise, set this property to `0` (`false`).

Data Types: `logical`

`UserIndex` — User index for single or multi-user scenario
`0` (default) | positive integer

User index, specified as a nonnegative integer. If the property is set to `0`, the angles of arrival and departure from the TGn channel model are used in the calculation of the spatial correlation matrix. If the property is set to a positive integer, pseudorandom offsets are applied to the TGn angles of arrival and departure before the calculation of the spatial correlation matrix. For more details, see the section on “MIMO Enhancements” on page 4-225.

Data Types: `double`

`TransmissionDirection` — Transmission direction
`'Downlink'` (default) | `'Uplink'`

Transmission direction of the active link, specified as either `'Downlink'` or `'Uplink'`. The default value, `'Downlink'`, specifies transmission from an access point to a user station.

Data Types: `char` | `string`

`NumTransmitAntennas` — Number of transmit antennas
`1` (default) | `2` | `3` | `4`

Number of transmit antennas, specified as `1`, `2`, `3`, or `4`.

Data Types: `double`

`TransmitAntennaSpacing` — Distance between transmit antenna elements
`0.5` (default) | positive scalar

Distance between transmit antenna elements, specified as a positive scalar expressed in wavelengths.

`TransmitAntennaSpacing` supports uniform linear arrays only.

Dependencies

To enable this property, set the `NumTransmitAntennas` property to a value greater than `1`.

Data Types: `double`

`NumReceiveAntennas` — Number of receive antennas
`1` (default) | `2` | `3` | `4`

Number of receive antennas, specified as `1`, `2`, `3`, or `4`.

Data Types: `double`

`ReceiveAntennaSpacing` — Distance between receive antenna elements

`0.5` (default) | positive scalar

Distance between receive antenna elements, specified as a positive scalar expressed in wavelengths.

`ReceiveAntennaSpacing` supports uniform linear arrays only.

Dependencies

To enable this property, set the `NumReceiveAntennas` property to a value greater than `1`.

Data Types: `double`

`LargeScaleFadingEffect` — Large-scale fading effects

`'None'` (default) | `'Pathloss'` | `'Shadowing'` | `'Pathloss and shadowing'`

Large-scale fading effects applied in the channel, specified as `'None'`, `'Pathloss'`, `'Shadowing'`, or `'Pathloss and shadowing'`.

Data Types: `char` | `string`

`NumPenetratedFloors` — Number of building floors

`0` (default) | positive integer

Number of building floors between the transmitter and the receiver, specified as a positive integer. Use this property in multiple floor scenarios to account for the floor attenuation loss in the path loss calculation. The default is `0`, which represents a communication link between a transmitter and a receiver located on the same floor.

Dependencies

The `NumPenetratedFloors` property applies only when `DelayProfile` is `'Model-A'` or `'Model-B'`.

Data Types: `double`

`FluorescentEffect` — Fluorescent effect

`true` or `1` (default) | `false` or `0`

Fluorescent effect, specified as a numeric or logical `1` (`true`) or `0` (`false`). To include Doppler effects from fluorescent lighting, set this property to `1` (`true`).

Dependencies

To enable this property, set the `DelayProfile` property to `'Model-D'` or `'Model-E'`.

Data Types: `logical`

`PowerLineFrequency` — Power line frequency

`'60Hz'` (default) | `'50Hz'`

Power line frequency in Hz, specified as `'50Hz'` or `'60Hz'`.

The power line frequency is 60 Hz in the United States and 50 Hz in Europe.

Dependencies

To enable this property, set the `FluorescentEffect` property to `1` (`true`) and the `DelayProfile` property to `'Model-D'` or `'Model-E'`.

Data Types: `char` | `string`

`NormalizeChannelOutputs` — Normalize channel outputs

`true` or `1` (default) | `false` or `0`

Normalize channel outputs by the number of receive antennas, specified as a numeric or logical `1` (`true`) or `0` (`false`).

Data Types: `logical`

`ChannelFiltering` — Enable channel filtering

`true` or `1` (default) | `false` or `0`

Enable channel filtering, specified as a numeric or logical `1` (`true`) or `0` (`false`). To enable channel filtering, set this property to `1` (`true`). To disable channel filtering, set this property to `0` (`false`).

Note If you set this property to `0` (`false`), the `step` object function does not accept an input signal. In this case, the `NumSamples` and `SampleRate` properties determine the duration of the fading process realization.

Data Types: `logical`

`NumSamples` — Number of time-domain samples

`80` (default) | positive integer

Number of time-domain samples used to get path gain samples, specified as a positive integer.

Dependencies

To enable this property, set the `ChannelFiltering` property to `0` (`false`).

Data Types: `double`

`OutputDataType` — Data type of impaired signal

`'double'` (default) | `'single'`

Data type of impaired signal, specified as one of these values:

- `'double'` – Return the `pathGains` output as a double-precision matrix
- `'single'` – Return the `pathGains` output as a single-precision matrix

Dependencies

To enable this property, set the `ChannelFiltering` property to `0` (`false`).

Data Types: `char` | `string`

`RandomStream` — Source of random number stream

`'Global stream'` (default) | `'mt19937ar with seed'`

Source of random number stream, specified as `'Global stream'` or `'mt19937ar with seed'`.

If you set this property to `'Global stream'`, the current global random number stream is used for random number generation. In this case, the `reset` function resets the filters and creates a new channel realization.

If you set this property to `'mt19937ar with seed'`, the mt19937ar algorithm generates random numbers. In this case, the `reset` function also reinitializes the random number stream to the value of the `Seed` property.

Note The random numbers of the channel components are distributed as follows:

- The random phase of the Doppler component due to fluorescent lights is uniformly distributed. See equation 27 of *TGn Channel Models* for more information.
- In multi-user scenarios using the TGac on page 4-201, TGah on page 4-214, or TGax on page 4-227 channel models, the per-user angle-of-arrival (AoA) and angle-of-departure (AoD) rotations discussed in the “MIMO Enhancements” on page 4-238 section are uniformly distributed.
- The fading samples are generated from a normally-distributed complex uncorrelated Gaussian process with zero mean and unit variance in discrete time.

Data Types: `char` | `string`

Seed — Initial seed of mt19937ar random number stream

73 (default) | nonnegative integer

Initial seed of an mt19937ar random number stream, specified as a nonnegative integer. The `Seed` property reinitializes the mt19937ar random number stream in the `reset` function.

Dependencies

To enable this property, set the `RandomStream` property to `'mt19937ar with seed'`.

Data Types: `double`

PathGainsOutputPort — Enable path gain output

`false` or `0` (default) | `true` or `1`

Enable path gain output computation, specified as a numeric or logical `1` (`true`) or `0` (`false`).

Data Types: `logical`

Usage

Syntax

```
y = tgah(x)
[y,pathGains] = tgah(x)
pathGains = tgah(x)
```

Description

`y = tgah(x)` filters input signal `x` through the TGah fading channel defined by the `wlanTGahChannel` System object, `tgah`, and returns the result in `y`.

`[y,pathGains] = tgah(x)` also returns in `pathGains` the TGah channel path gains of the underlying fading process.

This syntax applies when you set the `PathGainsOutputPort` property to `1` (`true`).

`pathGains = tgah(x)` returns the path gains. The `NumSamples` property determines the duration of the fading process.

This syntax applies when you set the `ChannelFiltering` property to `0` (`false`).

Input Arguments

x — Input signal
complex matrix

Input signal, specified as a real or complex N_S-by-N_T matrix, where:

- N_S is the number of samples.
- N_T is the number of transmit antennas and must be equal to the `NumTransmitAntennas` property value.

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

Output Arguments

y — Output signal
complex matrix

Output signal, returned as an N_S-by-N_R complex matrix, where:

- N_S is the number of samples.
- N_R is the number of receive antennas and is equal to the `NumReceiveAntennas` property value.

Data Types: `single` | `double`

`pathGains` — Path gains of the fading process
complex array

Path gains of the fading process, returned as an N_S-by-N_P-by-N_T-by-N_R complex array, where:

- N_S is the number of samples.
- N_P is the number of resolvable paths, that is, the number of paths defined for the case specified by the `DelayProfile` property.
- N_T is the number of transmit antennas and is equal to the `NumTransmitAntennas` property value.
- N_R is the number of receive antennas and is equal to the `NumReceiveAntennas` property value.

Data Types: `double` | `single`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to wlanTGahChannel

info Characteristic information about multipath fading channels

Common to All System Objects

step Run System object algorithm
release Release resources and allow changes to System object property values and input characteristics
reset Reset internal states of System object

Note `reset`: If the `RandomStream` property of the System object is set to `'Global stream'`, the `reset` function resets the filters only. If you set `RandomStream` to `'mt19937ar with seed'`, the `reset` function also reinitializes the random number stream to the value of the `Seed` property.

Examples

Pass S1G Waveform Through TGah Channel

Filter an 802.11ah waveform through a TGah channel.

```
cfgS1G = wlanS1GConfig(APEPLength=1000);
txWaveform = wlanWaveformGenerator([1;0;0;1],cfgS1G);
```

Create a TGah channel object and adjust some default properties. Specify a seed value to produce a repeatable channel output. Create an S1G configuration object and waveform. Pass the S1G waveform through the channel by supplying it as an input to the TGah channel object.

```
tgah = wlanTGahChannel;
tgah.LargeScaleFadingEffect = 'PathLoss and shadowing';
tgah.FloorSeparation = 2;
tgah.RandomStream = 'mt19937ar with seed';
tgah.Seed = 10;

channelOutput = tgah(txWaveform);
```

Confirm the channel bandwidth and set the corresponding sample rate.

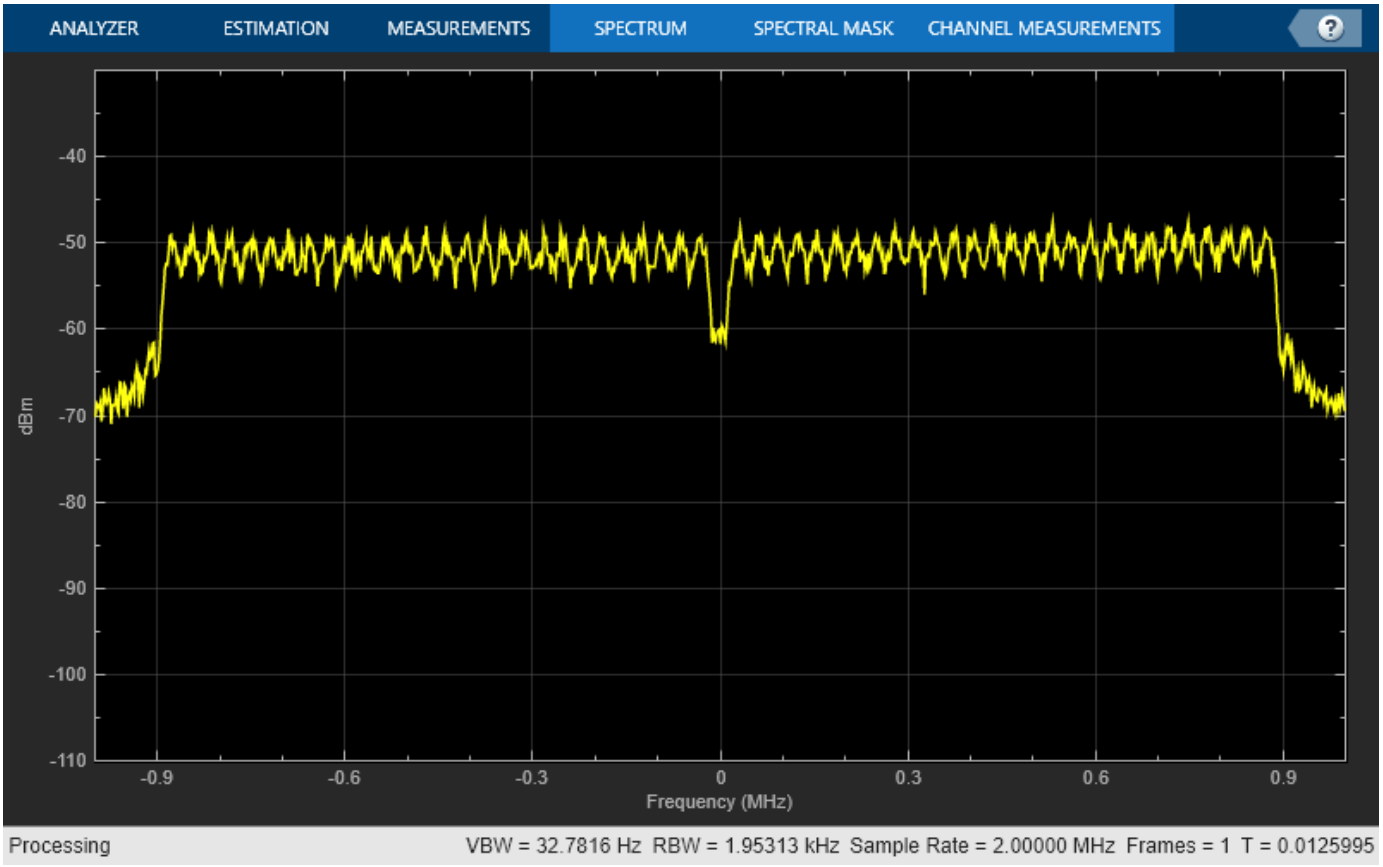
```
cfgS1G.ChannelBandwidth
```

```
ans =
'CBW2'
```

```
fs = 2e6;
```

Plot the spectrum of the channel output waveform.

```
saScope = spectrumAnalyzer(SampleRate=fs,YLimits=[-110 -30]);
saScope(channelOutput)
```



Across the spectrum, the mean power of the channel output waveform is approximately -50 dBm.

TGah Channel Model-B Delay Profile

Plot the delay profile for an impulse waveform passed through a TGah channel.

Create an impulse waveform. Delay the impulse by 10 samples, which is equivalent to 10 ns in time.

```
txWaveform = zeros(100,1);
txWaveform(11) = 1;
```

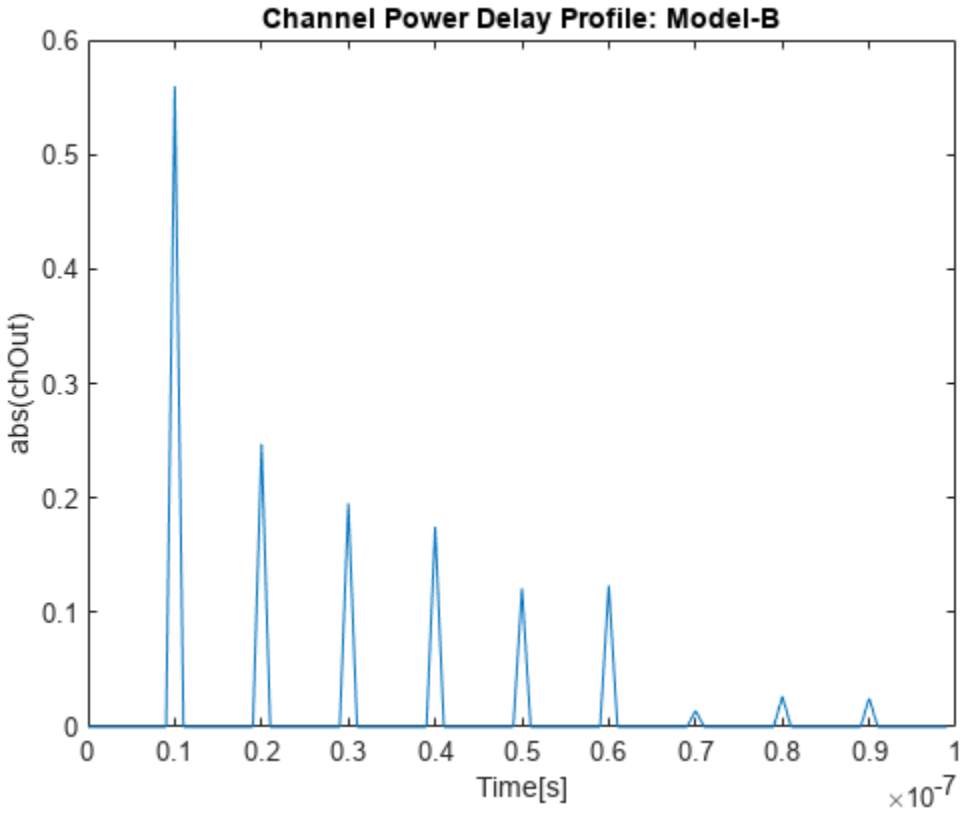
Create a TGah channel object. Specify the seed for reproducible results.

```
tgah = wlanTGahChannel;
tgah.RandomStream = 'mt19937ar with seed';
tgah.Seed = 10;
```

Set the sample rate so that sampling of the channel multipaths are integer multiples of integer sampling delay.

```
tgah.SampleRate = 1e9;

chOut = tgah(txWaveform);
plot((0:length(chOut)-1)*1/tgah.SampleRate,abs(chOut));
```

```
xlabel('Time[s]'); ylabel('abs(chOut)');
title('Channel Power Delay Profile: Model-B')
```



Transmit S1G Waveform Through 4x2 MIMO Channel

Create a S1G waveform generated using four transmit antennas and two spatial streams.

```
cfg = wlanS1GConfig(NumTransmitAntennas=4,NumSpaceTimeStreams=2, ...
    SpatialMapping='Fourier',APEPLength=1000);
txSig = wlanWaveformGenerator([1;0;0;1],cfg);
```

Create a 4x2 MIMO TGah channel and disable large-scale fading effects.

```
tgahChan = wlanTGahChannel('SampleRate',1e6,'ChannelBandwidth','CBW1', ...
    'NumTransmitAntennas',4,'NumReceiveAntennas',2, ...
    'LargeScaleFadingEffect','None');
```

Pass the transmit waveform through the channel.

```
rxSig = tgahChan(txSig);
```

Display the spectrum of the two received space-time streams.

```
saScope = spectrumAnalyzer(SampleRate=1e6, ...
    ShowLegend=true, ...
```

```
    ChannelNames={'Stream 1','Stream 2'});
saScope(rxSig)
```

Algorithms

The algorithms used to model the TGah channel are based on those used for the TGn channel (as described in `wlanTGnChannel` and *TGn Channel Models* [2]) and the TGac channel (as described in `wlanTGacChannel` and *TGac Channel Model Addendum* [3]). Complete information on the changes required to support TGah channels can be found in *TGah Channel Model* [1]. The changes to support the TGah channel include lower bandwidths, floor separation attenuation, Wall Separation Attenuation, and path loss and shadowing.

Lower Bandwidths

The TGah channel model supports channel bandwidths down to 1 MHz.

Floor Separation Attenuation

In the TGah channel, the path loss model used to compute the spatial correlation accounts for floor separation attenuation effects. The floor separation loss depends on the number of floors penetrated as shown in the equation:

$$PEL_{floor} = 18.3n^{(n+2)/(n+1)-0.46},$$

where n is the number of floors, represented by `NumPenetratedFloors` property of the System object. For more information, see *TGah Channel Model* [1].

MIMO Enhancements

The TGn channel model supports no more than 4x4 MIMO, while the TGah model supports 8x8 MIMO.

The TGah model uses per-user angle diversity to support simultaneous communication between multiple user stations and an access point. For each channel realization, the model achieves this by adding pseudorandom offsets to the angles of arrival and departure of each cluster before the calculation of the spatial correlation matrix. Different offsets are chosen for each positive value of the `UserIndex` property. This causes the angles of arrival and departure for each cluster to differ between users.

Changing the `TransmissionDirection` property swaps the angle of arrival of each cluster with its corresponding angle of departure. For more details, see the Appendix of [3].

To see how the `UserIndex` property can be used in an application, see this example: “802.11ax Packet Error Rate Simulation for Uplink Trigger-Based Format”.

Path Loss and Shadowing

TGah Channel Model [1], Table 2 defines path loss parameters that are slightly modified from those defined for TGn. Specifically, the shadow fading values corresponding to breakpoint distance are 1 dB less for all TGah channel models.

The path loss exponent and the standard deviation of the shadow fading loss characterize each model. The two parameters depend on the presence of a line of sight (LOS) between the transmitter and receiver. For paths with a transmitter-to-receiver distance, d, less that the breakpoint distance, d_{BP}, the LOS parameters apply. For $d > d_{BP}$, the non line of sight (NLOS) parameters apply. The table summarizes the path loss and shadow fading parameters.

Parameter	Model					
	A	B	C	D	E	F
Breakpoint distance, d_{BP} (m)	5	5	5	10	20	30
Path loss exponent for $d \le d_{BP}$	2	2	2	2	2	2
Path loss exponent for $d > d_{BP}$	3.5	3.5	3.5	3.5	3.5	3.5
Shadow fading σ (dB) for $d \le d_{BP}$	2	2	2	2	2	2
Shadow fading σ (dB) for $d > d_{BP}$	3	3	4	4	5	5

Version History

Introduced in R2017a

References

[1] Porat R., S. K. Yong, and K. Doppler. *TGah Channel Model*. IEEE 802.11-11/0968r4, March 2015.

[2] Erceg, V., L. Schumacher, P. Kyritsi, et al. *TGn Channel Models*. Version 4. IEEE 802.11-03/940r4, May 2004.

[3] Breit, G., H. Sampath, S. Vermani, et al. *TGac Channel Model Addendum*. Version 12. IEEE 802.11-09/0308r12, March 2010.

[4] Kermoal, J. P., L. Schumacher, K. I. Pedersen, P. E. Mogensen, and F. Frederiksen. "A Stochastic MIMO Radio Channel Model with Experimental Validation." *IEEE Journal on Selected Areas in Communications*. Vol. 20, No. 6, August 2002, pp. 1211–1226.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also

Objects

`wlanTGayChannel` | `wlanTGaxChannel` | `wlanTGacChannel` | `wlanTGnChannel`

wlanTGaxChannel

Filter signal through 802.11ax multipath fading channel

Description

The `wlanTGaxChannel` System object filters an input signal through an 802.11ax (TGax) indoor MIMO channel as specified in [1], following the MIMO modeling approach described in [4].

The fading processing assumes the same parameters for all N_T-by-N_R links of the TGax channel, where N_T is the number of transmit antennas and N_R is the number of receive antennas. Each link comprises all multipaths for that link.

To filter an input signal using a TGax multipath fading channel:

1. Create the `wlanTGaxChannel` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
tgax = wlanTGaxChannel
tgax = wlanTGaxChannel(Name,Value)
```

Description

`tgax = wlanTGaxChannel` creates a TGax channel System object, `tgax`. This object filters a real or complex input signal through the TGax channel to obtain the channel-impaired signal.

`tgax = wlanTGaxChannel(Name,Value)` creates a TGax channel object, `tgax`, and sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `wlanTGaxChannel('NumReceiveAntennas',2,'SampleRate',10e6)` creates a TGax channel with two receive antennas and a 10 MHz sample rate.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`SampleRate` — Sample rate of the input signal
`80e6` (default) | positive scalar

Sample rate of the input signal in Hz, specified as a positive scalar.

Data Types: `double`

`DelayProfile` — Delay profile model

`'Model-B'` (default) | `'Model-A'` | `'Model-C'` | `'Model-D'` | `'Model-E'` | `'Model-F'`

Delay profile model, specified as `'Model-A'`, `'Model-B'`, `'Model-C'`, `'Model-D'`, `'Model-E'`, or `'Model-F'`.

The table summarizes the models properties before the bandwidth reduction factor.

Parameter	Model					
	A	B	C	D	E	F
Breakpoint distance (m)	5	5	5	10	20	30
RMS delay spread (ns)	0	15	30	50	100	150
Maximum delay (ns)	0	80	200	390	730	1050
Rician K-factor (dB)	0	0	0	3	6	6
Number of taps	1	9	14	18	18	18
Number of clusters	1	2	2	3	4	6

The number of clusters represents the number of independently modeled propagation paths.

Data Types: `char` | `string`

`ChannelBandwidth` — Channel bandwidth

`'CBW80'` (default) | `'CBW20'` | `'CBW40'` | `'CBW160'` | `'CBW320'`

Channel bandwidth, specified as one of these values.

- `'CBW20'` — Channel bandwidth of 20 MHz
- `'CBW40'` — Channel bandwidth of 40 MHz
- `'CBW80'` — Channel bandwidth of 80 MHz
- `'CBW160'` — Channel bandwidth of 160 MHz
- `'CBW320'` — Channel bandwidth of 320 MHz

Data Types: `char` | `string`

`CarrierFrequency` — RF carrier frequency

`5.25e9` (default) | positive scalar

RF carrier frequency, in Hz, specified as a positive scalar.

Data Types: `double`

`EnvironmentalSpeed` — Speed of the scatterers

`0.089` (default) | positive scalar

Speed of the scatterers, in km/h, specified as a positive scalar.

Data Types: `double`

`TransmitReceiveDistance` — Distance between transmitter and receiver

`3` (default) | positive scalar

Distance between the transmitter and receiver in meters, specified as a positive scalar.

`TransmitReceiveDistance` is used to compute the path loss, and to determine whether the channel has a line of sight (LOS) or non line of sight (NLOS) condition. The path loss and standard deviation of shadow fading loss depend on the separation between the transmitter and the receiver.

Data Types: `double`

`NormalizePathGains` — Normalize path gains

`true` or `1` (default) | `false` or `0`

Normalize path gains, specified as a numeric or logical `1` (`true`) or `0` (`false`). To normalize the fading processes such that the total power of the path gains, averaged over time, is 0 dB, set this property to `1` (`true`). Otherwise, set this property to `false`.

Data Types: `logical`

`UserIndex` — User index for single or multi-user scenario

`0` (default) | positive integer

User index, specified as a nonnegative integer. If the property is set to `0`, the angles of arrival and departure from the TGn channel model are used in the calculation of the spatial correlation matrix. If the property is set to a positive integer, pseudorandom offsets are applied to the TGn angles of arrival and departure before the calculation of the spatial correlation matrix. For more details, see the section on “MIMO Enhancements” on page 4-238.

Data Types: `double`

`TransmissionDirection` — Transmission direction

`'Downlink'` (default) | `'Uplink'`

Transmission direction of the active link, specified as either `'Downlink'` or `'Uplink'`. The default value, `'Downlink'`, specifies transmission from an access point to a user station.

Data Types: `char` | `string`

`NumTransmitAntennas` — Number of transmit antennas

`1` (default) | positive integer

Number of transmit antennas, specified as a positive integer.

Data Types: `double`

`TransmitAntennaSpacing` — Distance between transmit antenna elements

`0.5` (default) | positive scalar

Distance between transmit antenna elements, specified as a positive scalar expressed in wavelengths.

`TransmitAntennaSpacing` supports uniform linear arrays only.

Dependencies

To enable this property, set the `NumTransmitAntennas` property to a value greater than `1`.

Data Types: `double`

`NumReceiveAntennas` — Number of receive antennas

`1` (default) | positive integer

Number of receive antennas, specified as a positive integer.

Data Types: `double`

`ReceiveAntennaSpacing` — Distance between receive antenna elements

`0.5` (default) | positive scalar

Distance between receive antenna elements, specified as a positive scalar expressed in wavelengths.

`ReceiveAntennaSpacing` supports uniform linear arrays only.

Dependencies

To enable this property, set the `NumReceiveAntennas` property to a value greater than `1`.

Data Types: `double`

`LargeScaleFadingEffect` — Large-scale fading effects

`'None'` (default) | `'Pathloss'` | `'Shadowing'` | `'Pathloss and shadowing'`

Large-scale fading effects applied in the channel, specified as `'None'`, `'Pathloss'`, `'Shadowing'`, or `'Pathloss and shadowing'`.

Data Types: `char` | `string`

`NumPenetratedFloors` — Number of building floors

`0` (default) | positive integer

Number of building floors between the transmitter and the receiver, specified as a positive integer. Use this property in multiple floor scenarios to account for the floor attenuation loss in the path loss calculation. The default is `0`, which represents a communication link between a transmitter and a receiver located on the same floor.

Dependencies

The `NumPenetratedFloors` property applies only when `DelayProfile` is `'Model-A'` or `'Model-B'`.

Data Types: `double`

`NumPenetratedWalls` — Number of walls

`0` (default) | positive integer

Number of walls between the transmitter and receiver, specified as a positive integer. Use this property to account for the wall penetration loss in the path loss calculation.

The default is `0`, which represents a communication link between a transmitter and a receiver without wall penetration loss.

Data Types: `double`

`WallPenetrationLoss` — Penetration loss of a single wall

`5` (default) | real scalar

Penetration loss of a single wall in dB, specified as a real scalar.

Dependencies

The `WallPenetrationLoss` property applies only when `NumPenetratedWalls` is greater than 0.

Data Types: `double`

`FluorescentEffect` — Fluorescent effect

`true` or `1` (default) | `false` or `0`

Fluorescent effect, specified as a numeric or logical `1` (`true`) or `0` (`false`). To include Doppler effects from fluorescent lighting, set this property to `1` (`true`).

Dependencies

To enable this property, set the `DelayProfile` property to `'Model-D'` or `'Model-E'`.

Data Types: `logical`

`PowerLineFrequency` — Power line frequency

`'60Hz'` (default) | `'50Hz'`

Power line frequency in Hz, specified as `'50Hz'` or `'60Hz'`.

The power line frequency is 60 Hz in the United States and 50 Hz in Europe.

Dependencies

To enable this property, set the `FluorescentEffect` property to `1` (`true`) and the `DelayProfile` property to `'Model-D'` or `'Model-E'`.

Data Types: `char` | `string`

`NormalizeChannelOutputs` — Normalize channel outputs

`true` or `1` (default) | `false` or `0`

Normalize channel outputs by the number of receive antennas, specified as a numeric or logical `1` (`true`) or `0` (`false`).

Data Types: `logical`

`ChannelFiltering` — Enable channel filtering

`true` or `1` (default) | `false` or `0`

Enable channel filtering, specified as a numeric or logical `1` (`true`) or `0` (`false`). To enable channel filtering, set this property to `1` (`true`). To disable channel filtering, set this property to `0` (`false`).

Note If you set this property to `0` (`false`), the `step` object function does not accept an input signal. In this case, the `NumSamples` and `SampleRate` properties determine the duration of the fading process realization.

Data Types: `logical`

`NumSamples` — Number of time-domain samples

`1280` (default) | positive integer

Number of time-domain samples used to get path gain samples, specified as a positive integer.

Dependencies

To enable this property, set the `ChannelFiltering` property to `0` (`false`).

Data Types: `double`

`OutputDataType` — Data type of impaired signal

`'double'` (default) | `'single'`

Data type of impaired signal, specified as one of these values:

- `'double'` – Return the `pathGains` output as a double-precision matrix
- `'single'` – Return the `pathGains` output as a single-precision matrix

Dependencies

To enable this property, set the `ChannelFiltering` property to `0` (`false`).

Data Types: `char` | `string`

`RandomStream` — Source of random number stream

`'Global stream'` (default) | `'mt19937ar with seed'`

Source of random number stream, specified as `'Global stream'` or `'mt19937ar with seed'`.

If you set this property to `'Global stream'`, the current global random number stream is used for random number generation. In this case, the `reset` function resets the filters and creates a new channel realization.

If you set this property to `'mt19937ar with seed'`, the mt19937ar algorithm generates random numbers. In this case, the `reset` function also reinitializes the random number stream to the value of the `Seed` property.

Note The random numbers of the channel components are distributed as follows:

- The random phase of the Doppler component due to fluorescent lights is uniformly distributed. See equation 27 of *TGn Channel Models* for more information.
- In multi-user scenarios using the TGac on page 4-201, TGah on page 4-214, or TGax on page 4-227 channel models, the per-user angle-of-arrival (AoA) and angle-of-departure (AoD) rotations discussed in the “MIMO Enhancements” on page 4-238 section are uniformly distributed.
- The fading samples are generated from a normally-distributed complex uncorrelated Gaussian process with zero mean and unit variance in discrete time.

Data Types: `char` | `string`

`Seed` — Initial seed of mt19937ar random number stream

73 (default) | nonnegative integer

Initial seed of an mt19937ar random number stream, specified as a nonnegative integer. The `Seed` property reinitializes the mt19937ar random number stream in the `reset` function.

Dependencies

To enable this property, set the `RandomStream` property to `'mt19937ar with seed'`.

Data Types: `double`

`PathGainsOutputPort` — Enable path gain output
`false` or `0` (default) | `true` or `1`

Enable path gain output computation, specified as a numeric or logical `1` (`true`) or `0` (`false`).

Data Types: `logical`

Usage

Syntax

```
y = tgax(x)
[y,pathGains] = tgax(x)
pathGains = tgax(x)
```

Description

`y = tgax(x)` filters input signal `x` through the TGax fading channel defined by the `wlanTGaxChannel` System object, `tgax`, and returns the result in `y`.

`[y,pathGains] = tgax(x)` also returns in `pathGains` the TGax channel path gains of the underlying fading process.

This syntax applies when you set the `PathGainsOutputPort` property of `tgax` to `1` (`true`).

`pathGains = tgax(x)` returns the path gains. The `NumSamples` property determines the duration of the fading process.

This syntax applies when you set the `ChannelFiltering` property to `0` (`false`).

Input Arguments

`x` — Input signal
complex matrix

Input signal, specified as a real or complex N_S-by-N_T matrix, where:

- N_S is the number of samples.
- N_T is the number of transmit antennas and must be equal to the `NumTransmitAntennas` property value of `tgax`.

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

Output Arguments

`y` — Output signal
complex matrix

Output signal, returned as an N_S-by-N_R complex matrix, where:

- N_S is the number of samples.

- N_R is the number of receive antennas and is equal to the `NumReceiveAntennas` property value of `tgax`.

Data Types: `single` | `double`

`pathGains` — Path gains of the fading process
complex array

Path gains of the fading process, returned as an N_S-by-N_P-by-N_T-by-N_R complex array, where:

- N_S is the number of samples.
- N_P is the number of resolvable paths, that is, the number of paths defined for the case specified by the `DelayProfile` property.
- N_T is the number of transmit antennas and is equal to the `NumTransmitAntennas` property value of `tgax`.
- N_R is the number of receive antennas and is equal to the `NumReceiveAntennas` property value of `tgax`.

Data Types: `single` | `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to wlanTGaxChannel

info Characteristic information about multipath fading channels

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Note `reset`: If the `RandomStream` property of the System object is set to `'Global stream'`, the `reset` function resets the filters only. If you set `RandomStream` to `'mt19937ar with seed'`, the `reset` function also reinitializes the random number stream to the value of the `Seed` property.

Examples

TGax Channel Impulse Response

Obtain a channel impulse response by filtering an impulse through a TGax channel.

Create an impulse.

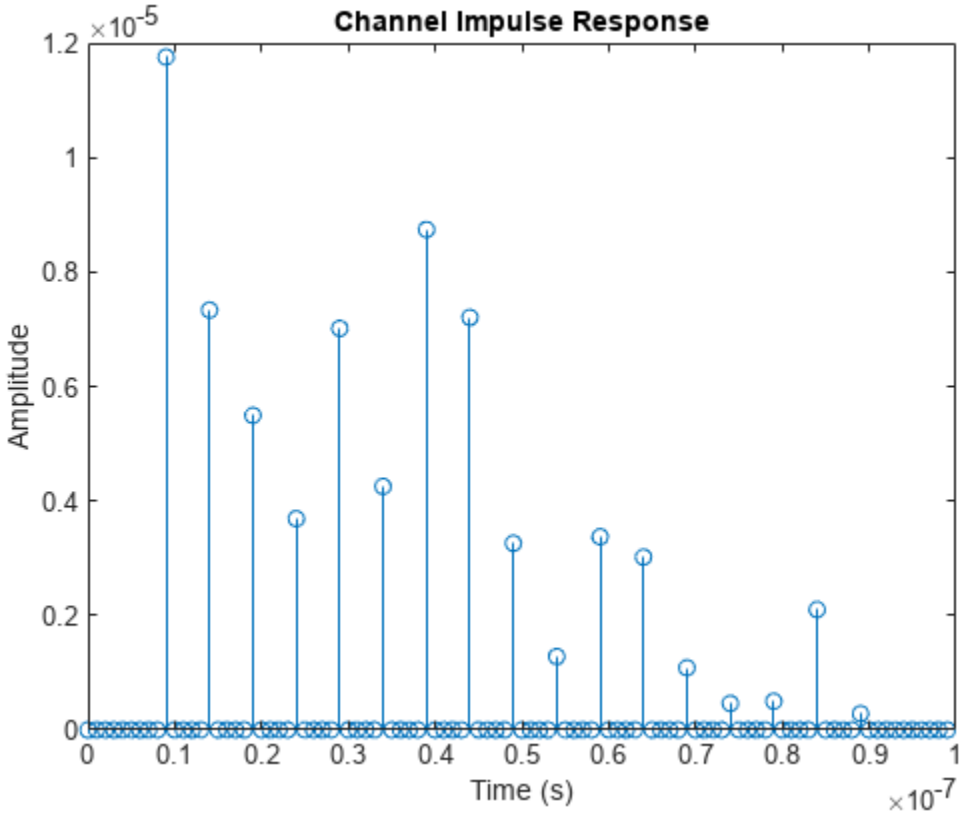
```
input = zeros(100,1);
input(10) = 1;
```

Create the TGax channel System Object with path loss and shadowing, two penetrated floors, and a sampling rate of 1 GHz.

```
tgax = wlanTGaxChannel;
tgax.LargeScaleFadingEffect = 'Pathloss and shadowing';
tgax.NumPenetratedFloors = 2;
tgax.RandomStream = 'mt19937ar with seed';
tgax.Seed = 10;
tgax.SampleRate = 1e9;
```

Plot the output impulse response of the channel.

```
figure
time = (1/tgax.SampleRate)*(0:length(input)-1);
stem(time,abs(tgax(input)))
xlabel('Time (s)')
ylabel('Amplitude')
title('Channel Impulse Response')
```



TGax Channel Delay Profile and Path Gains

Plot the delay profile and path gains of a TGax channel.

Create an impulse.

```
input = zeros(100,4);
input(10) = 1;
```

Create the TGax channel System Object. Enable path gains at the output, and specify path loss, 20 MHz of channel bandwidth, a 4x2 MIMO channel, four penetrated floors, and a sampling rate of 1 GHz.

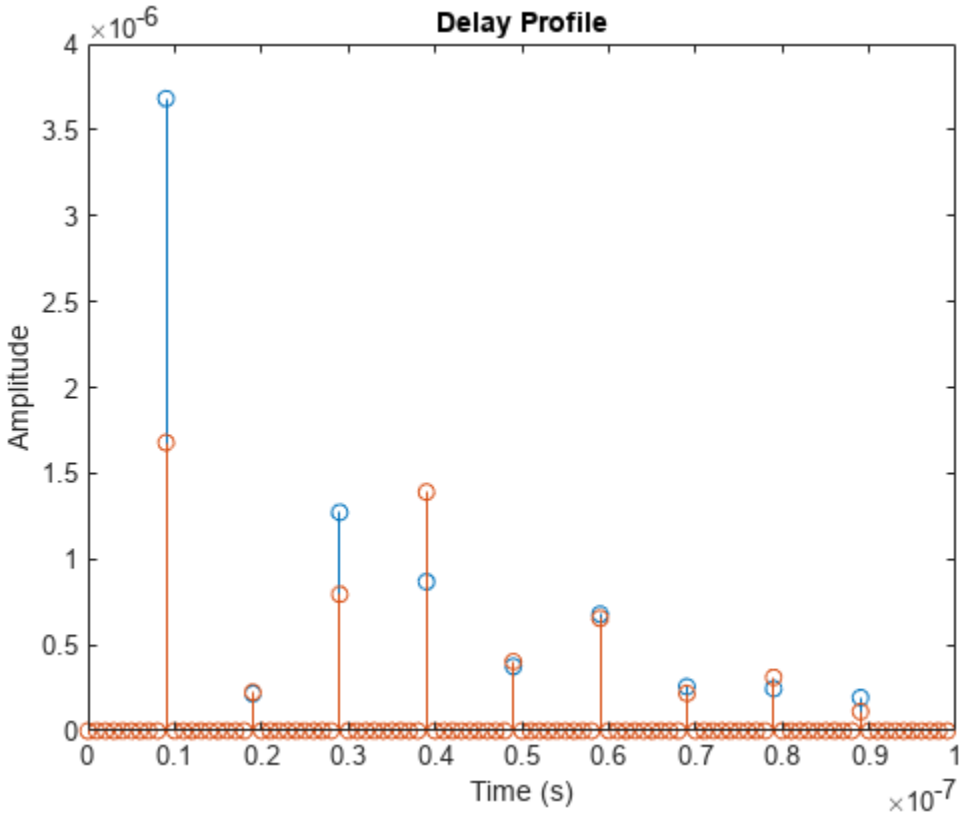
```
tgax = wlanTGaxChannel;
tgax.LargeScaleFadingEffect = 'Pathloss';
tgax.ChannelBandwidth = 'CBW20';
tgax.NumTransmitAntennas = 4;
tgax.NumReceiveAntennas = 2;
tgax.NumPenetratedFloors = 4;
tgax.RandomStream = 'mt19937ar with seed';
tgax.Seed = 10;
tgax.SampleRate = 1e9;
tgax.PathGainsOutputPort = true;
```

Filter the input impulse. Use the TGax channel object to generate the output response and the path gains.

```
[out,pathgains]= tgax(input);
```

Plot the output impulse response of the channel. The channel has two delay profiles, one per each receive antenna.

```
figure
time = (1/tgax.SampleRate)*(0:length(input)-1);
stem(time,abs(out))
xlabel('Time (s)')
ylabel('Amplitude')
title('Delay Profile')
```



The path gains of the channel are contained in a four dimensional array since the channel has nine resolvable paths, four transmit antennas and two receive antennas.

```
size(pathgains)
```

```
ans = 1×4

   100     9     4     2
```

Algorithms

The algorithms used to model the TGax channel are based on those used for the TGn channel (as described in `wlanTGnChannel` and *TGn Channel Models* [2]) and the TGac channel (as described in `wlanTGacChannel` and *TGac Channel Model Addendum* [3]). Complete information on the changes required to support TGax channels can be found in *TGax Channel Model* [1]. The changes to support the TGax channel include lower bandwidths, floor separation attenuation, wall separation attenuation, and path loss and shadowing.

Floor Separation Attenuation

In the TGax channel, the path loss model used to compute the spatial correlation accounts for floor separation attenuation effects. The floor separation loss depends on the number of floors penetrated, as shown in the equation:

$$PEL_{\text{floor}} = 18.3n^{(n+2)/(n+1)-0.46},$$

where n is the number of floors, represented by the `NumPenetratedFloors` property of the System object. For more information, see *TGax Channel Model* [1].

Wall Separation Attenuation

In the TGax channel, the path loss model used to compute the spatial correlation accounts for wall separation attenuation effects. The wall separation loss is defined by the following equation:

$PEL_{\text{wall}} = m \times L_{\text{iw}}$.

Where m is the number of walls penetrated, and L_{iw} is the penetration loss for a single wall. The variables m and L_{iw} are represented by the `NumPenetratedWalls` and `WallPenetrationLoss` properties of the System object, respectively. For more information, see *TGax Channel Model* [1].

MIMO Enhancements

The TGn channel model supports no more than 4x4 MIMO, while the TGax model supports 8x8 MIMO.

The TGax model uses per-user angle diversity to support simultaneous communication between multiple user stations and an access point. For each channel realization, the model achieves this by adding pseudorandom offsets to the angles of arrival and departure of each cluster before the calculation of the spatial correlation matrix. Different offsets are chosen for each positive value of the `UserIndex` property. This causes the angles of arrival and departure for each cluster to differ between users.

Changing the `TransmissionDirection` property swaps the angle of arrival of each cluster with its corresponding angle of departure. For more details, see the Appendix of [3].

To see how the `UserIndex` property can be used in an application, see this example: “802.11ax Packet Error Rate Simulation for Uplink Trigger-Based Format”.

Path Loss and Shadowing

In *TGax Channel Model* [1], Table 3 defines path loss parameters that are slightly modified from those defined for TGn. The floor penetration loss and wall penetration loss are added to this path loss.

The path loss exponent and the standard deviation of the shadow fading loss characterize each model. The two parameters depend on the presence of a line of sight (LOS) between the transmitter and receiver. For paths with a transmitter-to-receiver distance, d, less than the breakpoint distance, d_{BP}, the LOS parameters apply. For $d > d_{\text{BP}}$, the non line of sight (NLOS) parameters apply. The table summarizes the path loss and shadow fading parameters.

Parameter	Model	
	B	D
Breakpoint distance, d_{BP} (m)	5	10
Path loss exponent for $d \le d_{\text{BP}}$	2	2
Path loss exponent for $d > d_{\text{BP}}$	3.5	3.5
Shadow fading σ (dB) for $d \le d_{\text{BP}}$	3	3
Shadow fading σ (dB) for $d > d_{\text{BP}}$	4	5

Version History

Introduced in R2018a

References

[1] Jianhan, L., Ron, P. *et al*. *TGax Channel Model*. IEEE 802.11-14/0882r4, September 2014.

[2] Erceg, V., Schumacher, L., Kyritsi, P. *et al*. *TGn Channel Models*. Version 4. IEEE 802.11-03/940r4, May 2004.

[3] Breit, G., Sampath, H., Vermani, S. *et al*. *TGac Channel Model Addendum*. Version 12. IEEE 802.11-09/0308r12, March 2010.

[4] Kermoal, J. P., L. Schumacher, K. I. Pedersen, P. E. Mogensen, and F. Frederiksen. "A Stochastic MIMO Radio Channel Model with Experimental Validation." *IEEE Journal on Selected Areas in Communications*. Vol. 20, No. 6, August 2002, pp. 1211–1226.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also

Objects

`wlanTGayChannel` | `wlanTGacChannel` | `wlanTGahChannel` | `wlanTGnChannel`

wlanTGayChannel

Filter signal through 802.11ay multipath fading channel

Description

The `wlanTGayChannel` System object filters an input signal through an IEEE 802.11ay (TGay) multipath fading channel. The channel model follows the quasi-deterministic (Q-D) approach specified in [1].

To filter an input signal by using a TGay multipath fading channel:

1. Create the `wlanTGayChannel` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
tgay = wlanTGayChannel
tgay = wlanTGayChannel(Name,Value)
```

Description

`tgay = wlanTGayChannel` creates a TGay channel System object. This System object filters a real or complex input signal through the TGay channel to obtain a channel-impaired signal.

`tgay = wlanTGayChannel(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotation marks. For example, `wlanTGayChannel('SampleRate',1e9,'Environment','Large hotel lobby')` creates a TGay channel with a 1-GHz sample rate in a large hotel lobby environment.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`SampleRate` — Sample rate of input signal

`2.64e9` (default) | positive scalar

Sample rate of the input signal, in Hz, specified as a positive scalar.

Data Types: `double`

`CarrierFrequency` — Center frequency of input signal
`6e10` (default) | positive scalar

Center frequency of the input signal, in Hz, specified as a positive scalar.

Data Types: `double`

`Environment` — Channel model environment
`'Open area hotspot'` (default) | `'Street canyon hotspot'` | `'Large hotel lobby'`

Channel model environment, specified as `'Open area hotspot'`, `'Street canyon hotspot'`, or `'Large hotel lobby'`. For more information, see [1].

Data Types: `char` | `string`

`RoadWidth` — Street canyon road width
`16` (default) | positive scalar

Street canyon road width, in meters, specified as a positive scalar. The road is parallel to the *y*-axis, on which it has its center.

Dependencies

To enable this property, set `Environment` to `'Street canyon hotspot'`.

Data Types: `double`

`SidewalkWidth` — Street canyon sidewalk width
`6` (default) | positive scalar

Street canyon sidewalk width, in meters, specified as a positive scalar.

Dependencies

To enable this property, set `Environment` to `'Street canyon hotspot'`.

Data Types: `double`

`RoomDimensions` — Hotel lobby dimensions
`[20 15 6]` (default) | 1-by-3 vector of positive values

Hotel lobby dimensions, in meters, specified as a 1-by-3 vector of positive values. Each element of `RoomDimensions` specifies the length of the hotel lobby measured along an axis of the Cartesian coordinate system (*x*,*y*,*z*). The first element specifies the length along the *x*-axis. The second element specifies the length along the *y*-axis. The third element specifies the length along the *z*-axis. The origin of the coordinate system is on the floor of the hotel lobby, at the midpoint between the bounding walls.

Dependencies

To enable this property, set `Environment` to `'Large hotel lobby'`.

Data Types: `double`

`UserConfiguration` — User configuration
`'SU-SISO'` (default) | `'SU-MIMO 1x1'` | `'SU-MIMO 2x2'`

User configuration, specified as one of these values:

- `'SU-SISO'` – specify one transmit antenna array, one receive antenna array, and one data stream
- `'SU-MIMO 1x1'` – specify one transmit antenna array, one receive antenna array, and two data streams
- `'SU-MIMO 2x2'` – specify two transmit antenna arrays, two receive antenna arrays, and either two or four data streams, depending on the value of the `ArrayPolarization` property. You can check the number of data streams by using the `info` object function.

Use this property to specify the number of transmit and receive antenna arrays and the number of data streams at the transmitter and receiver. For more information, see Table 3-2 in [1].

Data Types: `char` | `string`

`ArraySeparation` — Separation between transmit arrays and receive arrays

`[0.5 0.5]` (default) | 1-by-2 vector of positive values

Separation between transmit arrays and receive arrays, in meters, specified as a 1-by-2 vector of positive values. The first element specifies the separation between centers of the transmit arrays. The second element specifies the separation between centers of the receive arrays. The distances between the relevant array centers are measured along the *x*-axes of the local array coordinate systems, in accordance with Figure 3-10 in [1].

Dependencies

To enable this property, set `UserConfiguration` to `'SU-MIMO 2x2'`.

Data Types: `double`

`ArrayPolarization` — Transmit and receive antenna array polarization type for SU-MIMO

`'Single, Single'` (default) | `'Single, Dual'` | `'Dual, Dual'`

Transmit and receive antenna array polarization type for SU-MIMO, specified as `'Single, Single'`, `'Single, Dual'`, or `'Dual, Dual'`. For more information, refer to Table 3-2 in [1].

Dependencies

To enable this property, set `UserConfiguration` to `'SU-MIMO 1x1'` or `'SU-MIMO 2x2'`.

Data Types: `char` | `string`

`TransmitArray` — Transmit antenna array

2-by-2 URA with 0.2 m element spacing (default) | `wlanURAConfig` object

Transmit antenna array, specified as a `wlanURAConfig` object. You can specify `TransmitArray` as a uniform rectangular array (URA), uniform linear array (ULA), or single element by setting the `Size` property of the `wlanURAConfig` object.

`TransmitArrayPosition` — Center of transmit antenna array

`[0; 0; 5]` (default) | 3-by-1 real-valued vector

Center of transmit antenna array, specified as a 3-by-1 real-valued vector. This property specifies the displacement, in meters, from the origin of the Cartesian coordinate system to the center of the transmit antenna array.

Data Types: `double`

`TransmitArrayOrientation` — Transmit antenna array orientation

`[0; 0; 0]` (default) | 3-by-1 real-valued vector

Transmit antenna array orientation, in degrees, specified as a 3-by-1 real-valued vector. Each element specifies the angle by which the local coordinate system of the transmit antenna array is rotated with respect to an axis of the global Cartesian coordinate system. The first element is the angle of rotation about the *z*-axis, and determines the target azimuthal angle. The second element is the angle of rotation about the rotated *x*-axis, and determines the target elevation angle. The third element is the angle of rotation about the rotated *z*-axis, and is specified for the non-symmetric azimuth distribution of the antenna gain. A positive value indicates a counterclockwise rotation. For more information, refer to Section 6.3.3 in [2].

Data Types: `double`

`TransmitArrayPolarization` — Transmit antenna array polarization type

`'None'` (default) | `'Vertical'` | `'Horizontal'` | `'LHCP'` | `'RHCP'`

Transmit antenna array polarization type, specified as one of these values:

- `'None'` – An unpolarized transmit antenna array
- `'Vertical'` – A vertically polarized transmit antenna array
- `'Horizontal'` – A horizontally polarized transmit antenna array
- `'LHCP'` – A left-hand circularly polarized transmit antenna array
- `'RHCP'` – A right-hand circularly polarized transmit antenna array

Dependencies

To enable this property, set `UserConfiguration` to `'SU-SISO'`.

Data Types: `char` | `string`

`ReceiveArray` — Receive antenna array

2-by-2 URA with element spacing of 0.2 m (default) | `wlanURAConfig` object

Receive antenna array, specified as a `wlanURAConfig` object. You can specify `ReceiveArray` as a URA, ULA, or single element by setting the `Size` property of the `wlanURAConfig` object.

`ReceiveArrayPosition` — Center of receive antenna array

`[8; 0; 1.5]` (default) | 3-by-1 real-valued vector

Center of receive antenna array, specified as a 3-by-1 real-valued vector. This property specifies the displacement, in meters, from the origin of the Cartesian coordinate system to the center of the receive antenna array.

Data Types: `double`

`ReceiveArrayOrientation` — Receive antenna array orientation

`[0; 0; 0]` (default) | 3-by-1 real-valued vector

Receive antenna array orientation, in degrees, specified as a 3-by-1 real-valued vector. Each element specifies the angle by which the local coordinate system of the receive antenna array is rotated with respect to an axis of the global Cartesian coordinate system. The first element is the angle of rotation about the *z*-axis, and determines the target azimuthal angle. The second element is the angle of rotation about the rotated *x*-axis, and determines the target elevation angle. The third element is the angle of rotation about the rotated *z*-axis, and is specified for the non-symmetric azimuth distribution

of the antenna gain. A positive value indicates a counterclockwise rotation. For more information, refer to Section 6.3.3 in [2].

Data Types: `double`

`ReceiveArrayPolarization` — Receive antenna array polarization type
`'None'` (default) | `'Vertical'` | `'Horizontal'` | `'LHCP'` | `'RHCP'`

Receive antenna array polarization type, specified as one of these values:

- `'None'` – An unpolarized receive antenna array
- `'Vertical'` – A vertically polarized receive antenna array
- `'Horizontal'` – A horizontally polarized receive antenna array
- `'LHCP'` – A left-hand circularly polarized receive antenna array
- `'RHCP'` – A right-hand circularly polarized receive antenna array

Dependencies

To enable this property, set `UserConfiguration` to `'SU-SISO'`.

Data Types: `char` | `string`

`ReceiveArrayVelocitySource` — Receive antenna array velocity source
`'Auto'` (default) | `'Custom'`

Receive antenna array velocity source, specified as `'Auto'` or `'Custom'`. To specify a randomly generated receive array velocity, as defined in [1], set this property to `'Auto'`.

Data Types: `char` | `string`

`ReceiveArrayVelocity` — Receive antenna array velocity
`[1; 1; 0]` (default) | 3-by-1 real-valued vector

Receive antenna array velocity, in meters per second, specified as a 3-by-1 real-valued vector.

Data Types: `double`

`RandomRays` — Generate random rays
`true` (default) | `false`

Generate random rays (R-Rays), specified as a logical value of `true` or `false`.

Data Types: `logical`

`IntraClusterRays` — Generate intra-cluster rays
`true` (default) | `false`

Generate intra-cluster rays, specified as a logical value of `true` or `false`.

Data Types: `logical`

`OxygenAbsorption` — Power losses due to oxygen absorption
`0.015` (default) | nonnegative scalar

Power losses due to oxygen absorption, in dB/m, specified as a nonnegative scalar.

Data Types: `double`

`BeamformingMethod` — Beamforming method

`'Maximum power ray'` (default) | `'Custom'`

Beamforming method, specified as `'Maximum power ray'` or `'Custom'`. For more information, see Section 6.5 in [2].

Data Types: `char` | `string`

`TransmitBeamformingVectors` — Transmit beamforming vectors

`[0.5; 0.5; 0.5; 0.5]` (default) | N_{TE}-by-N_{TS} complex-valued matrix

Transmit beamforming vectors, specified as an N_{TE}-by-N_{TS} complex-valued matrix.

- N_{TE} is the number of elements in each transmit antenna array.
- N_{TS} is the number of input data streams.

You can obtain N_{TE} and N_{TS} by using the `info` object function.

Tunable: Yes

Dependencies

To enable this property, set `BeamformingMethod` to `'Custom'`.

Data Types: `double`
Complex Number Support: Yes

`ReceiveBeamformingVectors` — Receive beamforming vectors

`[0.5; 0.5; 0.5; 0.5]` (default) | N_{RE}-by-N_{RS} complex-valued matrix

Receive beamforming vectors, specified as an N_{RE}-by-N_{RS} complex-valued matrix.

- N_{RE} is the number of elements in each receive antenna array.
- N_{RS} is the number of output data streams.

You can obtain N_{RE} and N_{RS} by using the `info` object function.

Tunable: Yes

Dependencies

To enable this property, set `BeamformingMethod` to `'Custom'`.

Data Types: `double`

`NormalizeImpulseResponses` — Normalize channel impulse responses

`true` (default) | `false`

Normalize channel impulse responses (CIRs), specified as a logical value of `true` or `false`. To normalize CIRs to 0 dB per stream, set this property to `true`.

Data Types: `logical`

`NormalizeChannelOutputs` — Normalize output by number of output streams

`true` (default) | `false`

Normalize output by number of output streams, specified as a logical value of `true` or `false`.

Data Types: `logical`

`RandomStream` — Source of random number stream
`'Global stream'` (default) | `'mt19937ar with seed'`

Source of random number stream, specified as `'Global stream'` or `'mt19937ar with seed'`. To use the current global random number stream for random number generation, set this property to `'Global stream'`. Using the `reset` object function when this property is set to `'Global stream'`:

- Regenerates R-Rays when `RandomRays` is set to `true`
- Regenerates intra-cluster rays when `IntraClusterRays` is set to `true`
- Regenerates the receive antenna array velocity when `ReceiveArrayVelocitySource` is set to `'Auto'`

To use the mt19937ar algorithm for self-contained random number generation, set this property to `'mt19937ar with seed'`.

Data Types: `char` | `string`

Seed — Initial seed of random number generator
73 (default) | nonnegative integer

Initial seed of random number generator, specified as a nonnegative integer.

Dependencies

To enable this property, set `RandomStream` to `'mt19937ar with seed'`.

Data Types: `double`

Usage

Syntax

```
y = tgay(x)
[y,CIR] = tgay(x)
```

Description

`y = tgay(x)` returns output signal `y` by filtering input signal `x` through the TGay fading channel defined by `wlanTGayChannel` System object `tgay`.

`[y,CIR] = tgay(x)` also returns the TGay channel impulse response of the underlying fading process for all simulated rays.

Input Arguments

`x` — Input signal
complex-valued matrix

Input signal, specified as a complex-valued matrix of size N_s-by-N_{TS}.

- N_s is the number time-domain samples.

- N_{TS} is the number of input data streams.

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

Output Arguments

y — Output signal
complex-valued matrix

Output signal, returned as a complex-valued matrix of size N_s-by-N_{RS}.

- N_s is the number time-domain samples.
- N_{RS} is the number of output data streams.

The `wlanTGayChannel` System object returns this output with the same data type as that of the `x` input.

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

CIR — Channel impulse response for all simulated rays
complex-valued matrix

Channel impulse response for all simulated rays, returned as a complex-valued matrix of size N_s-by-N_{ray}-by-N_{TS}-by-N_{RS}.

- N_s is the number time-domain samples.
- N_{ray} is the number of simulated rays.
- N_{TS} is the number of input data streams.
- N_{RS} is the number of output data streams.

The `wlanTGayChannel` System object returns this output with the same data type as that of the `x` input.

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

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to `wlanTGayChannel`

info	Characteristic information about multipath fading channels
showEnvironment	Display channel environment with D-Rays from ray tracing

Common to All System Objects

step	Run System object algorithm

release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Note `reset`: If you set the `RandomStream` property of the `wlanTGayChannel` System object to `'Global stream'`, using `reset`:

- Regenerates R-Rays when you set the `RandomRays` property to `true`
- Regenerates intra-cluster rays when you set the `IntraClusterRays` property to `true`
- Regenerates the receive antenna array velocity when you set the `ReceiveArrayVelocitySource` property to `'Auto'`

Examples

Return Characteristic Information of WLAN TGay Channel

Create a WLAN TGay multipath fading channel System object with default property values.

```
chan = wlanTGayChannel;
```

Return and display the characteristic information of the TGay channel.

```
s = info(chan)
```

```
s = struct with fields:
     ChannelFilterDelay: 7
    NumSamplesProcessed: 0
           NumTxStreams: 1
           NumRxStreams: 1
          NumTxElements: 4
          NumRxElements: 4
```

Filter 802.11ad Waveform Through TGay Channel

Filter an 802.11ad™ single-carrier unpolarized waveform through an SU-SISO 802.11ay™ channel, specifying a large hotel lobby environment. Check that the output signal is consistent when the same input waveform is filtered through the channel.

Create a directional-multi-gigabit-format (DMG-format) configuration object with the specified modulation and coding scheme (MCS).

```
cfgDMG = wlanDMGConfig('MCS','4');
```

Generate a DMG waveform for a randomly generated PSDU.

```
psdu = randi([0 1], 8*cfgDMG.PSDULength, 1);
txWaveform = wlanWaveformGenerator(psdu,cfgDMG);
```

Configure a TGay channel System object for a large hotel lobby environment, specifying the sample rate, transmit and receive antenna arrays, and source of the random number stream.

```
tgay =  wlanTGayChannel('SampleRate',wlanSampleRate(cfgDMG),'Environment','Large hotel lobby', .
    'TransmitArray',wlanURAConfig('Size',[4 4]),'ReceiveArray',wlanURAConfig('Size',[3 3]), ...
    'RandomStream','mt19937ar with seed','Seed',100);
```

Filter the waveform through the TGay channel.

```
rxWaveform1 = tgay(txWaveform);
```

Reset the channel and filter the waveform through the TGay channel again. Check that the output waveform is consistent when the same input waveform is filtered through the TGay channel after calling the `reset` object function.

```
reset(tgay);
rxWaveform2 =  tgay(txWaveform);
isequal(rxWaveform1,rxWaveform2)
```

```
ans = logical
   1
```

Filter Dual-Polarized Signal Through 802.11ay Channel

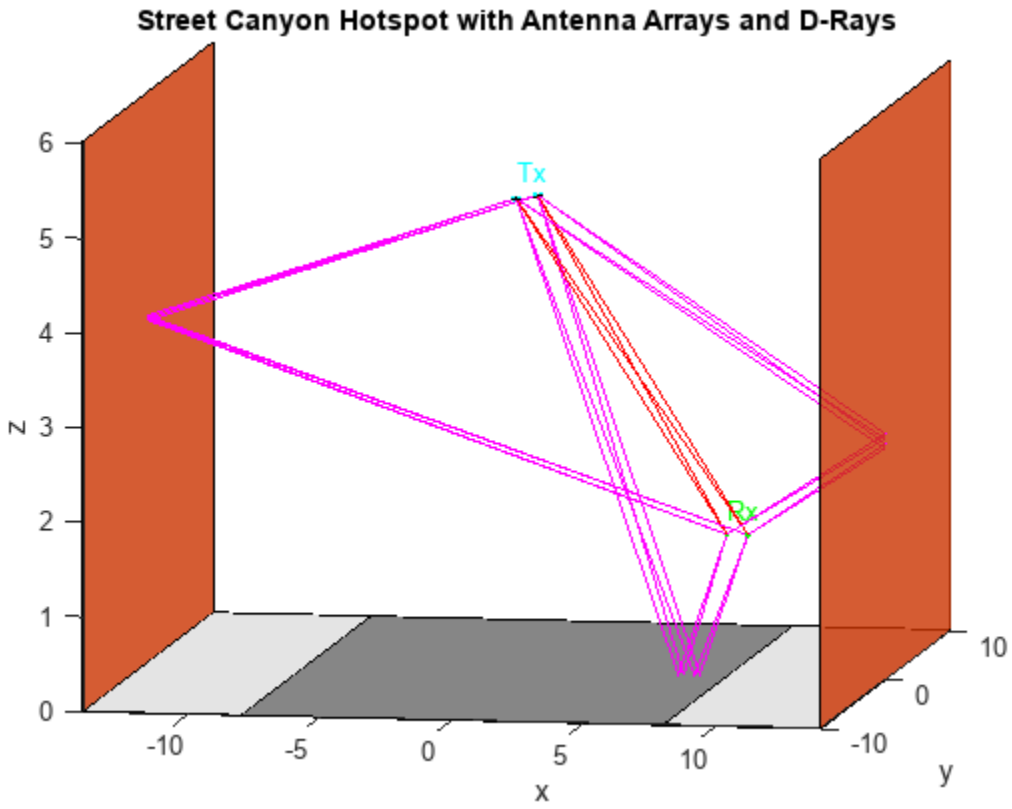
Filter a dual-polarized signal through a WLAN 802.11ay™ channel, specifying a street canyon environment.

Configure a TGay channel System object for a street canyon environment, specifying a user configuration of single-user multiple-input/multiple-output (SU-MIMO) with two transmit antenna arrays and two receive antenna arrays. Specify the transmit antenna arrays as two-element uniform linear arrays (ULAs) and the receive antenna arrays as single isotropic elements. Use a custom beamforming method to specify the transmit and receive beamforming vectors, and specify the source of the random number stream.

```
tgay = wlanTGayChannel('SampleRate',2e9,'Environment','Street canyon hotspot', ...
    'UserConfiguration','SU-MIMO 2x2','ArraySeparation',[0.8 0.8],'ArrayPolarization','Dual, Dua
    'TransmitArray',wlanURAConfig('Size',[1 2]),'TransmitArrayOrientation',[10; 10; 10], ...
    'ReceiveArray',wlanURAConfig('Size',[1 1]),'BeamformingMethod','Custom','NormalizeImpulseRes
    'RandomStream','mt19937ar with seed','Seed',100);
```

Display the environment of the TGay channel.

```
showEnvironment(tgay);
title('Street Canyon Hotspot with Antenna Arrays and D-Rays');
```



Retrieve channel characteristics by using the `info` object function.

```
tgayInfo = tgay.info;
```

Formulate the beamforming vectors in terms of the number of transmit elements, receive elements, transmit streams, and receive streams obtained from `tgayInfo`.

```
NTE = tgayInfo.NumTxElements;
NTS = tgayInfo.NumTxStreams;
NRE = tgayInfo.NumRxElements;
NRS = tgayInfo.NumRxStreams;
tgay.TransmitBeamformingVectors = ones(NTE,NTS)/sqrt(NTE);
tgay.ReceiveBeamformingVectors = ones(NRE,NRS)/sqrt(NRE);
```

Create a random input signal and filter it through the TGay channel.

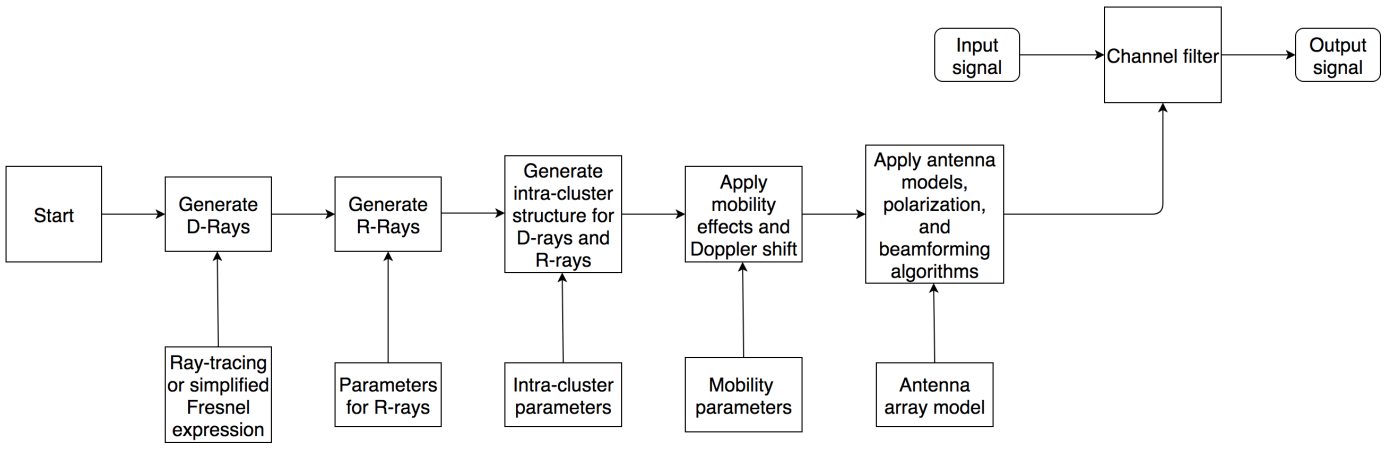
```
txSignal = complex(rand(100,NTS),rand(100,NTS));
rxSignal = tgay(txSignal);
```

Algorithms

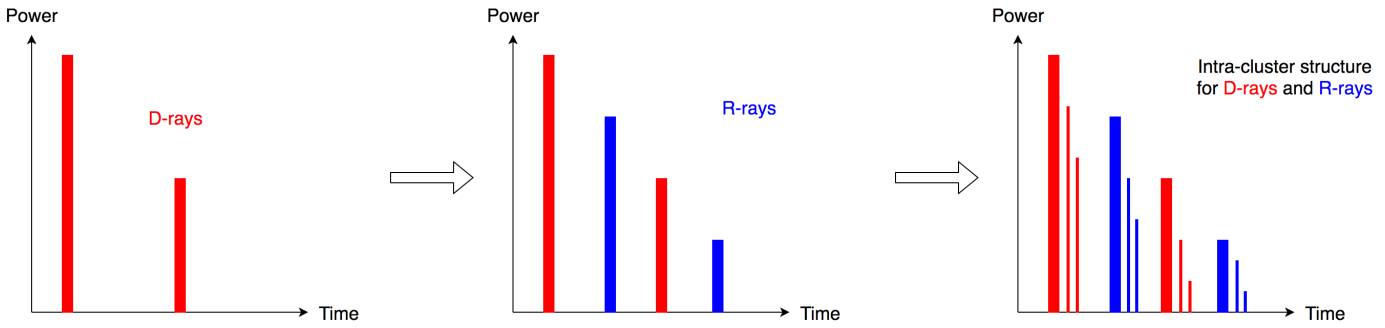
Channel Impulse Response

These diagrams show the Q-D algorithm and base steps for generating the channel impulse response. For more information, see Section 4 of [1].

Q-D Algorithm



Base Steps



Version History

Introduced in R2019a

References

[1] Maltsev, A., *et al. Channel Models for 802.11ay*. IEEE 802.11-15/1150r9, March 2017.

[2] Maltsev, A., *et al.. Channel Models for 60GHz WLAN Systems*. IEEE 802.11-09/0334r8, May 2010.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

Objects
`wlanTGaxChannel` | `wlanTGacChannel` | `wlanTGahChannel` | `wlanTGnChannel`

wlanTGnChannel

Filter signal through 802.11n multipath fading channel

Description

The `wlanTGnChannel` System object filters an input signal through an 802.11n (TGn) multipath fading channel.

The fading processing assumes the same parameters for all N_T-by-N_R links of the TGn channel. N_T is the number of transmit antennas and N_R is the number of receive antennas. Each link comprises all multipaths for that link.

To filter an input signal using a TGn multipath fading channel:

1. Create the `wlanTGnChannel` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
tgn = wlanTGnChannel
tgn = wlanTGnChannel(Name,Value)
```

Description

`tgn = wlanTGnChannel` creates a TGn fading channel System object, `tgn`. This object filters a real or complex input signal through the TGn channel to obtain the channel-impaired signal.

`tgn = wlanTGnChannel(Name,Value)` creates a TGn channel object, `tgn`, and sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `wlanTGnChannel('NumReceiveAntennas',2,'SampleRate',10e6)` creates a TGn channel with two receive antennas and a 10 MHz sample rate.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`SampleRate` — Sample rate of the input signal
`20e6` (default) | positive scalar

Sample rate of the input signal in Hz, specified as a positive scalar.

Data Types: `double`

`DelayProfile` — Delay profile model
`'Model-B'` (default) | `'Model-A'` | `'Model-C'` | `'Model-D'` | `'Model-E'` | `'Model-F'`

Delay profile model, specified as `'Model-A'`, `'Model-B'`, `'Model-C'`, `'Model-D'`, `'Model-E'`, or `'Model-F'`.

The table summarizes the models properties before the bandwidth reduction factor.

Parameter	Model					
	A	B	C	D	E	F
Breakpoint distance (m)	5	5	5	10	20	30
RMS delay spread (ns)	0	15	30	50	100	150
Maximum delay (ns)	0	80	200	390	730	1050
Rician K-factor (dB)	0	0	0	3	6	6
Number of clusters	1	2	2	3	4	6
Number of taps	1	9	14	18	18	18

Data Types: `char` | `string`

`CarrierFrequency` — RF carrier frequency
`5.25e9` (default) | positive scalar

RF carrier frequency in Hz, specified as a positive scalar.

Data Types: `double`

`EnvironmentalSpeed` — Speed of the scatterers
`1.2` (default) | positive scalar

Speed of the scatterers in km/h, specified as a positive scalar.

Data Types: `double`

`TransmitReceiveDistance` — Distance between transmitter and receiver
`3` (default) | positive scalar

Distance between the transmitter and receiver in meters, specified as a positive scalar.

`TransmitReceiveDistance` is used to compute the path loss, and to determine whether the channel has a line of sight (LOS) or non line of sight (NLOS) condition. The path loss and standard deviation of shadow fading loss depend on the separation between the transmitter and the receiver.

Data Types: `double`

`NormalizePathGains` — Normalize path gains
`true` or `1` (default) | `false` or `0`

Normalize path gains, specified as a numeric or logical `1` (`true`) or `0` (`false`). To normalize the fading processes such that the total power of the path gains, averaged over time, is 0 dB, set this property to `1` (`true`). Otherwise, set this property to `0` (`false`).

Data Types: `logical`

`NumTransmitAntennas` — Number of transmit antennas
`1` (default) | positive integer

Number of transmit antennas, specified as a positive integer.

Data Types: `double`

`TransmitAntennaSpacing` — Distance between transmit antenna elements
`0.5` (default) | positive scalar

Distance between transmit antenna elements, specified as a positive scalar expressed in wavelengths.

`TransmitAntennaSpacing` supports uniform linear arrays only.

Dependencies

To enable this property, set the `NumTransmitAntennas` property to a value greater than `1`.

Data Types: `double`

`NumReceiveAntennas` — Number of receive antennas
`1` (default) | positive integer

Number of receive antennas, specified as a positive integer.

Data Types: `double`

`ReceiveAntennaSpacing` — Distance between receive antenna elements
`0.5` (default) | positive scalar

Distance between receive antenna elements, specified as a positive scalar expressed in wavelengths.

`ReceiveAntennaSpacing` supports uniform linear arrays only.

Dependencies

To enable this property, set the `NumReceiveAntennas` property to a value greater than `1`.

Data Types: `double`

`LargeScaleFadingEffect` — Large-scale fading effects
`'None'` (default) | `'Pathloss'` | `'Shadowing'` | `'Pathloss and shadowing'`

Large-scale fading effects applied in the channel, specified as `'None'`, `'Pathloss'`, `'Shadowing'`, or `'Pathloss and shadowing'`.

Data Types: `char` | `string`

`FluorescentEffect` — Fluorescent effect
`true` or `1` (default) | `false` or `0`

Fluorescent effect, specified as a numeric or logical `1` (`true`) or `0` (`false`). To include Doppler effects from fluorescent lighting, set this property to `1` (`true`).

Dependencies

To enable this property, set the `DelayProfile` property to `'Model-D'` or `'Model-E'`.

Data Types: `logical`

`PowerLineFrequency` — Power line frequency

`'60Hz'` (default) | `'50Hz'`

Power line frequency in Hz, specified as `'50Hz'` or `'60Hz'`.

The power line frequency is 60 Hz in the United States and 50 Hz in Europe.

Dependencies

To enable this property, set the `FluorescentEffect` property to `1` (`true`) and the `DelayProfile` property to `'Model-D'` or `'Model-E'`.

Data Types: `char` | `string`

`NormalizeChannelOutputs` — Normalize channel outputs

`true` or `1` (default) | `false` or `0`

Normalize channel outputs by the number of receive antennas, specified as a numeric or logical `1` (`true`) or `0` (`false`).

Data Types: `logical`

`ChannelFiltering` — Enable channel filtering

`true` or `1` (default) | `false` or `0`

Enable channel filtering, specified as a numeric or logical `1` (`true`) or `0` (`false`). To enable channel filtering, set this property to `1` (`true`). To disable channel filtering, set this property to `0` (`false`)..

Note If you set this property to `0` (`false`), the `step` object function does not accept an input signal. In this case, the `NumSamples` and `SampleRate` properties determine the duration of the fading process realization.

Data Types: `logical`

`NumSamples` — Number of time-domain samples

`80` (default) | positive integer

Number of time-domain samples used to get path gain samples, specified as a positive integer.

Dependencies

To enable this property, set the `ChannelFiltering` property to `false`.

Data Types: `double`

`OutputDataType` — Data type of impaired signal

`'double'` (default) | `'single'`

Data type of impaired signal, specified as one of these values:

- `'double'` – Return the `pathGains` output as a double-precision matrix
- `'single'` – Return the `pathGains` output as a single-precision matrix

Dependencies

To enable this property, set the `ChannelFiltering` property to `0` (`false`).

Data Types: `char` | `string`

`RandomStream` — Source of random number stream

`'Global stream'` (default) | `'mt19937ar with seed'`

Source of random number stream, specified as `'Global stream'` or `'mt19937ar with seed'`.

If you set this property to `'Global stream'`, the current global random number stream is used for random number generation. In this case, the `reset` function resets the filters and creates a new channel realization.

If you set this property to `'mt19937ar with seed'`, the mt19937ar algorithm generates random numbers. In this case, the `reset` function also reinitializes the random number stream to the value of the `Seed` property.

Note The random numbers of the channel components are distributed as follows:

- The random phase of the Doppler component due to fluorescent lights is uniformly distributed. See equation 27 of *TGn Channel Models* for more information.
- In multi-user scenarios using the TGac on page 4-201, TGah on page 4-214, or TGax on page 4-227 channel models, the per-user angle-of-arrival (AoA) and angle-of-departure (AoD) rotations discussed in the “MIMO Enhancements” on page 4-238 section are uniformly distributed.
- The fading samples are generated from a normally-distributed complex uncorrelated Gaussian process with zero mean and unit variance in discrete time.

Data Types: `char` | `string`

`Seed` — Initial seed of mt19937ar random number stream

73 (default) | nonnegative integer

Initial seed of an mt19937ar random number stream, specified as a nonnegative integer. The `Seed` property reinitializes the mt19937ar random number stream in the `reset` function.

Dependencies

To enable this property, set the `RandomStream` property to `'mt19937ar with seed'`.

Data Types: `double`

`PathGainsOutputPort` — Enable path gain output

`false` or `0` (default) | `true` or `1`

Enable path gain output computation, specified as a numeric or logical `1` (`true`) or `0` (`false`).

Data Types: `logical`

Usage

Syntax

```
y = tgn(x)
[y,pathGains] = tgn(x)
pathGains = tgac(x)
```

Description

`y = tgn(x)` filters input signal `x` through the TGn fading channel defined by the `wlanTGnChannel` System object, `tgn`, and returns the result in `y`.

`[y,pathGains] = tgn(x)` also returns in `pathGains` the TGn channel path gains of the underlying fading process.

This syntax applies when you set the `PathGainsOutputPort` property to `1` (`true`).

`pathGains = tgac(x)` returns the path gains. The `NumSamples` property determines the duration of the fading process.

This syntax applies when you set the `ChannelFiltering` property to `0` (`false`).

Input Arguments

x — Input signal

complex matrix

Input signal, specified as a real or complex N_S-by-N_T matrix, where:

- N_S is the number of samples.
- N_T is the number of transmit antennas and must be equal to the `NumTransmitAntennas` property value.

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

Output Arguments

y — Output signal

complex matrix

Output signal, returned as an N_S-by-N_R complex matrix, where:

- N_S is the number of samples.
- N_R is the number of receive antennas and is equal to the `NumReceiveAntennas` property value.

Data Types: `single` | `double`

pathGains — Path gains of the fading process

complex array

Path gains of the fading process, returned as an N_S-by-N_P-by-N_T-by-N_R complex array, where:

- N_S is the number of samples.
- N_P is the number of resolvable paths, that is, the number of paths defined for the case specified by the `DelayProfile` property.
- N_T is the number of transmit antennas and is equal to the `NumTransmitAntennas` property value.
- N_R is the number of receive antennas and is equal to the `NumReceiveAntennas` property value.

Data Types: `single` | `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to wlanTGnChannel

info Characteristic information about multipath fading channels

Common to All System Objects

step Run System object algorithm
release Release resources and allow changes to System object property values and input characteristics
reset Reset internal states of System object

Note `reset`: If the `RandomStream` property of the System object is set to `'Global stream'`, the `reset` function resets the filters only. If you set `RandomStream` to `'mt19937ar with seed'`, the `reset` function also reinitializes the random number stream to the value of the `Seed` property.

Examples

Transmit HT Waveform Through TGn Channel

Generate an HT waveform and pass it through a TGn SISO channel. Display the spectrum of the resultant signal.

Set the channel bandwidth and the corresponding sample rate.

```
bw = 'CBW40';
fs = 40e6;
```

Generate an HT waveform for a 40 MHz channel.

```
cfg = wlanHTConfig('ChannelBandwidth',bw);
txSig = wlanWaveformGenerator(randi([0 1],1000,1),cfg);
```

Create a TGn SISO channel with path loss and shadowing enabled.

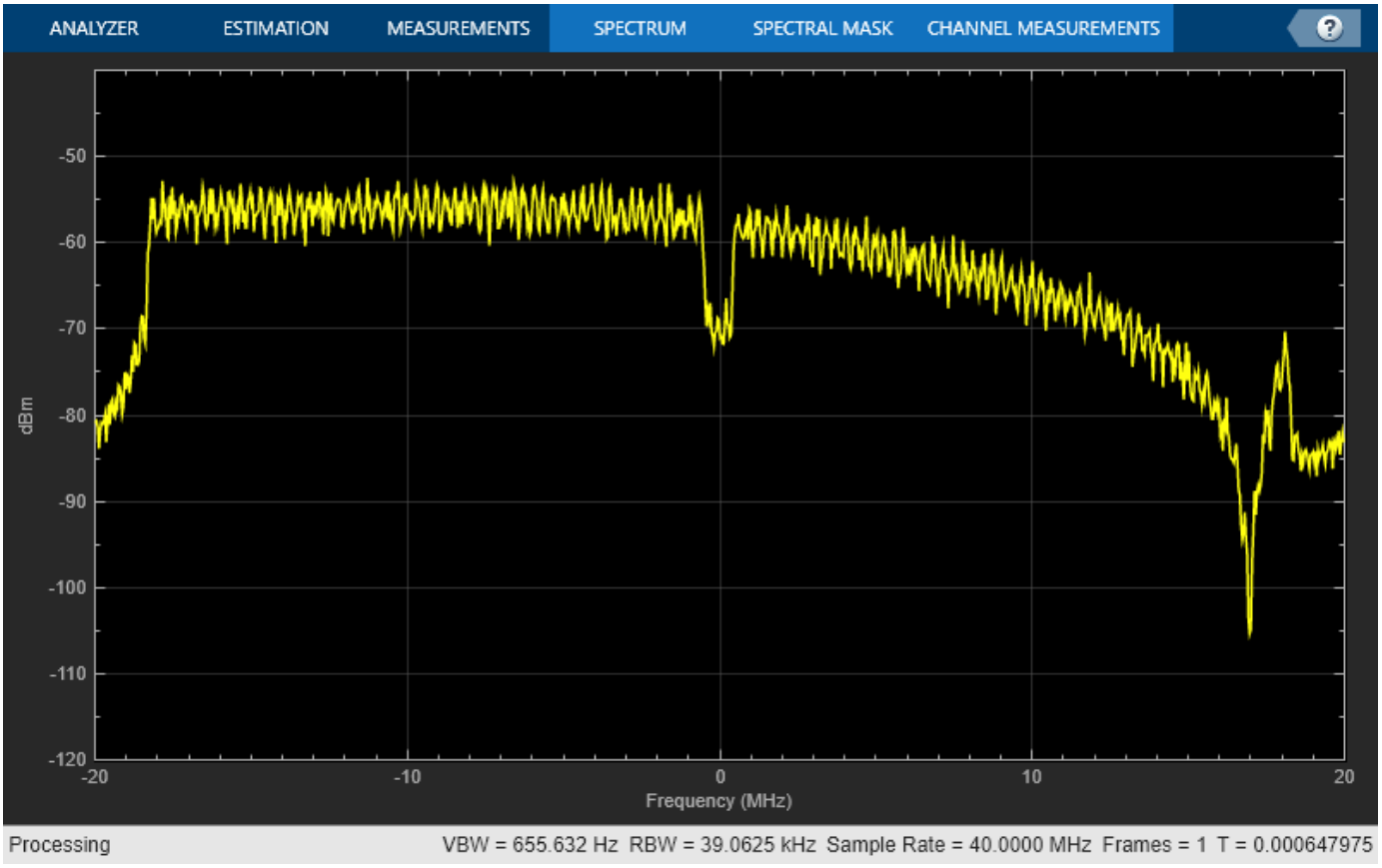
```
tgnChan = wlanTGnChannel('SampleRate',fs, ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
```

Pass the HT waveform through the channel.

```
rxSig = tgnChan(txSig);
```

Plot the spectrum of the received waveform.

```
saScope = spectrumAnalyzer(SampleRate=fs,YLimits=[-120 -40]);
saScope(rxSig)
```



Because path loss and shadowing are enabled, the mean received power across the spectrum is approximately -60 dBm.

Transmit HT Waveform Through 4x2 MIMO Channel

Create an HT waveform having four transmit antennas and two space-time streams.

```
cfg = wlanHTConfig('NumTransmitAntennas',4,'NumSpaceTimeStreams',2, ...
    'SpatialMapping','Fourier');
txSig = wlanWaveformGenerator([1;0;0;1],cfg);
```

Create a 4x2 MIMO TGn channel and disable large-scale fading effects.

```
tgnChan = wlanTGnChannel('SampleRate',20e6, ...
    'NumTransmitAntennas',4, ...
```

```
    'NumReceiveAntennas',2, ...
    'LargeScaleFadingEffect','None');
```

Pass the transmit waveform through the channel.

```
rxSig = tgnChan(txSig);
```

Display the spectrum of the two received space-time streams.

```
saScope = spectrumAnalyzer(SampleRate=20e6, ...
    ShowLegend=true, ...
    ChannelNames={'Stream 1','Stream 2'});
saScope(rxSig)
```

Recover HT Data from 2x2 MIMO Channel

Transmit an HT-LTF and an HT data field through a noisy 2x2 MIMO channel. Demodulate the received HT-LTF to estimate the channel coefficients. Recover the HT data and determine the number of bit errors.

Set the channel bandwidth and corresponding sample rate.

```
bw = 'CBW40';
fs = 40e6;
```

Create HT-LTF and HT data fields having two transmit antennas and two space-time streams.

```
cfg = wlanHTConfig('ChannelBandwidth',bw, ...
    'NumTransmitAntennas',2,'NumSpaceTimeStreams',2);
txPSDU = randi([0 1],8*cfg.PSDULength,1);
txLTF = wlanHTLTF(cfg);
txDataSig = wlanHTData(txPSDU,cfg);
```

Create a 2x2 MIMO TGn channel with path loss and shadowing enabled.

```
tgnChan = wlanTGnChannel('SampleRate',fs, ...
    'NumTransmitAntennas',2,'NumReceiveAntennas',2, ...
    'LargeScaleFadingEffect','None');
```

Create AWGN channel noise, setting SNR = 15 dB.

```
chNoise = comm.AWGNChannel('NoiseMethod','Signal to noise ratio (SNR)',...
    'SNR',15);
```

Pass the signals through the TGn channel and noise models.

```
rxLTF = chNoise(tgnChan(txLTF));
rxDataSig = chNoise(tgnChan(txDataSig));
```

Create an AWGN channel for a 40 MHz channel with a 9 dB noise figure. The noise variance, `nVar`, is equal to *kTBF*, where *k* is Boltzmann's constant, *T* is the ambient temperature of 290 K, *B* is the bandwidth (sample rate), and *F* is the receiver noise figure.

```
nVar = 10^((-228.6 + 10*log10(290) + 10*log10(fs) + 9)/10);
awgnChan = comm.AWGNChannel('NoiseMethod','Variance','Variance',nVar);
```

Pass the signals through the channel.

```
rxLTF = awgnChan(rxLTF);
rxDataSig = awgnChan(rxDataSig);
```

Demodulate the HT-LTF. Use the demodulated signal to estimate the channel coefficients.

```
dLTF = wlanHTLTFDemodulate(rxLTF,cfg);
chEst = wlanHTLTFChannelEstimate(dLTF,cfg);
```

Recover the data and determine the number of bit errors.

```
rxPSDU = wlanHTDataRecover(rxDataSig,chEst,nVar,cfg);
numErr = biterr(txPSDU,rxPSDU)
```

```
numErr = 0
```

Algorithms

The 802.11n channel object uses a filtered Gaussian noise model in which the path delays, powers, angular spread, angles of arrival, and angles of departure are determined empirically. The specific modeling approach is described in [1].

Multipath Parameters

The channel is modeled as several clusters, each of which represents an independent propagation path between the transmitter and the receiver. A cluster is composed of subpaths, or taps, which

share angular spreads, angles of arrival, and angles of departure. Delay and power level vary from tap to tap. Within the TGn model, clusters comprise 1–7 taps. The cluster parameters for cluster 1 of model B are shown in the table.

Parameter	Tap				
	1	2	3	4	5
Delay (ns)	0	10	20	30	40
Power (dB)	0	–5.4	–10.8	–16.2	–21.7
Angle of arrival (°)	4.3	4.3	4.3	4.3	4.3
Receiver angular spread (°)	14.4	14.4	14.4	14.4	14.4
Angle of departure (°)	225.1	225.1	225.1	225.1	225.1
Transmitter angular spread (°)	14.4	14.4	14.4	14.4	14.4

For each model, the first tap has a line of sight (LOS) between the transmitter and receiver, whereas all other taps are non line of sight (NLOS). As a result, the first tap exhibits Rician behavior, while the others exhibit Rayleigh behavior. The Rician K-factor is the ratio between the power in the first tap and the power in the other taps. A large K-factor indicates a strong LOS component.

The angles of arrival and departure for each cluster are randomly selected from a uniform distribution over [0, 2π]. These angles are independent of each other and are fixed for all channel realizations. By fixing the values, the transmit and receive correlation matrices are computed only once. Angular spread values were indirectly determined from empirical data and fall within the 20° to 40° range.

Path Loss and Shadowing

The path loss exponent and the standard deviation of the shadow fading loss characterize each model. The two parameters depend on the presence of a LOS between the transmitter and receiver. For paths with a transmitter-to-receiver distance, d, less that the breakpoint distance, d_{BP}, the LOS parameters apply. For $d > d_{BP}$, the NLOS parameters apply. The table summarizes the path loss and shadow fading parameters.

Parameter	Model					
	A	B	C	D	E	F
Breakpoint distance, d_{BP} (m)	5	5	5	10	20	30
Path loss exponent for $d \le d_{BP}$	2	2	2	2	2	2
Path loss exponent for $d > d_{BP}$	3.5	3.5	3.5	3.5	3.5	3.5
Shadow fading σ (dB) for $d \le d_{BP}$	3	3	3	3	3	3
Shadow fading σ (dB) for $d > d_{BP}$	4	4	5	5	6	6

Doppler Effects

In indoor environments, the transmitter and receiver are stationary, and Doppler effects arise from people moving between them. The TGn model employs a bell-shaped Doppler spectrum in which the environmental speed, ν_0, is 1.2 km/h by default (it is specified by the `EnvironmentalSpeed` property). The Doppler spread, f_d, is calculated as $f_d = \nu_0/\lambda$, where λ is the carrier wavelength.

The channel sampling rate, F_s, must be lower than the input sampling rate to avoid aliasing. It is calculated as:

$$F_s = (\nu_0 \times F_c)/(300 \times c)$$

where F_c is the carrier frequency, specified by the `CarrierFrequency` property, c is the speed of light and ν_0 is defined in m/s.

In addition to basic Doppler effects resulting from environmental motion, fluorescent lights introduce signal fading at twice the power line frequency. The effects show up as frequency-selective amplitude modulation. Again, to avoid aliasing, the Nyquist frequency of the first interpolation factor must be greater than the highest harmonic.

The effect is included in models D and E. To disable this effect, set the `FluorescentEffect` property to `false`.

Version History

Introduced in R2015b

References

[1] Erceg, V., L. Schumacher, P. Kyritsi, et al. *TGn Channel Models*. Version 4. IEEE 802.11-03/940r4, May 2004.

[2] Kermoal, J. P., L. Schumacher, K. I. Pedersen, P. E. Mogensen, and F. Frederiksen, “A Stochastic MIMO Radio Channel Model with Experimental Validation”. *IEEE Journal on Selected Areas in Communications*., Vol. 20, No. 6, August 2002, pp. 1211–1226.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

Objects

`wlanTGayChannel` | `wlanTGaxChannel` | `wlanTGacChannel` | `wlanTGahChannel`

wlanURAConfig

Create antenna array configuration object for 802.11ay channel model

Description

The `wlanURAConfig` object is a uniform rectangular array (URA) configuration object. It models a URA consisting of isotropic antenna elements. Use a `wlanURAConfig` object to specify the `TransmitArray` and `ReceiveArray` properties of the `wlanTGayChannel` System object.

Creation

Syntax

```
URA = wlanURAConfig
URA = wlanURAConfig(Name,Value)
```

Description

`URA = wlanURAConfig` creates a URA object, `URA`, to be used as the transmit and receive antenna arrays of a `wlanTGayChannel` System object.

`URA = wlanURAConfig(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotation marks. For example, `wlanURAConfig('Size',[4 4])` creates a 4-by-4 URA.

Properties

`Element` — Array element
`'isotropic'` (default)

This property is read-only.

Array element, specified as `'isotropic'`.

Data Types: `char`

`Size` — Antenna array size
`[2 2]` (default) | 1-by-2 vector of positive integers

Antenna array size, specified as a 1-by-2 vector of positive integers. Each element in the vector specifies the number of antenna elements along an axis of the Cartesian coordinate system (x,y,z). The first element specifies the number of elements along the y-axis. The second element specifies the number of elements along the x-axis. The normal to the antenna array points along the z-axis.

When one of the elements in `Size` is `1`, the array is a uniform linear array (ULA). When `Size` is `[1 1]`, the antenna array is a single antenna element.

Data Types: `double`

`ElementSpacing` — Spacing between array elements
`[0.2 0.2]` (default) | 1-by-2 vector of positive scalars

Spacing between array elements, in meters, specified as a 1-by-2 vector of positive scalars. Each element in the vector specifies the spacing between antenna elements along an axis of the Cartesian coordinate system. The first element specifies the spacing between antenna elements along the *y*-axis. The second element specifies the spacing between antenna elements along the *x*-axis.

Data Types: `double`

Examples

Create 3-by-3 URA

Create a 3-by-3 URA with an element spacing of 0.5 m for use with a WLAN TGay channel model.

Create a configuration object for a 3-by-3 URA with an element spacing of 0.5 m.

```
URA = wlanURAConfig('Size',[3 3],'ElementSpacing',[0.5 0.5]);
```

Create a WLAN TGay channel System object, specifying the URA as the transmit array.

```
tgay =  wlanTGayChannel('TransmitArray',URA);
disp(tgay.TransmitArray);
```

```
  wlanURAConfig with properties:

              Size: [3 3]
    ElementSpacing: [0.5000 0.5000]

   Constant properties:
           Element: 'isotropic'
```

Create ULA

Create an eight-element ULA for use with a WLAN TGay channel model.

Define an eight-element ULA along the *x*-axis by creating a configuration object for a 1-by-8 URA. Specify the element spacing as 0.1 m.

```
ULA = wlanURAConfig('Size',[1 8],'ElementSpacing',[0.1 0.1]);
```

Create a WLAN TGay channel System object, specifying the ULA as the receive antenna array.

```
tgay = wlanTGayChannel('ReceiveArray',ULA);
disp(tgay.ReceiveArray);
```

```
  wlanURAConfig with properties:

              Size: [1 8]
    ElementSpacing: [0.1000 0.1000]

   Constant properties:
           Element: 'isotropic'
```

Version History

Introduced in R2019a

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`wlanTGayChannel`

wlanVHTConfig

Configure VHT transmission

Description

The `wlanVHTConfig` object is a configuration object for the WLAN very high throughput (VHT) packet format.

Creation

Syntax

```
cfgVHT = wlanVHTConfig
cfgVHT = wlanVHTConfig(Name,Value)
```

Description

`cfgVHT = wlanVHTConfig` creates a configuration object that initializes parameters for an IEEE 802.11 VHT “PPDU” on page 4-275.

`cfgVHT = wlanVHTConfig(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotation marks. For example, `wlanVHTConfig('GuardInterval','Short')` specifies a 400-nanosecond guard interval (cyclic prefix) duration.

At runtime, the calling function validates object settings for properties relevant to the operation of the function.

Properties

`ChannelBandwidth` — Channel bandwidth of PPDU transmission

`'CBW80'` (default) | `'CBW20'` | `'CBW40'` | `'CBW160'`

Channel bandwidth of PPDU transmission, specified as one of these values:

- `'CBW20'` – Channel bandwidth of 20 MHz
- `'CBW40'` – Channel bandwidth of 40 MHz
- `'CBW80'` – Channel bandwidth of 80 MHz
- `'CBW160'` – Channel bandwidth of 160 MHz

Data Types: `char` | `string`

`NumUsers` — Number of users

1 (default) | 2 | 3 | 4

Number of users, specified as 1, 2, 3, or 4.

Data Types: `double`

`UserPositions` — User positions

`[0 1]` (default) | vector of integers

User positions, specified as a 1-by-`NumUsers` vector of integers in the interval [0, 3] in strictly increasing order.

Example: `[0 2 3]` specifies the positions for three users. The first user occupies position 0, the second user occupies position 2, and the third user occupies position 3.

Dependencies

This property applies only when you specify the `NumUsers` property as a value greater than `1`.

Data Types: `double`

`NumTransmitAntennas` — Number of transmit antennas

`1` (default) | positive integer

Number of transmit antennas, specified as a positive integer.

Data Types: `double`

`PreVHTCyclicShifts` — Cyclic shift values of additional transmit antennas

`-75` (default) | integer in the interval [–200, 0] | row vector

Cyclic shift values, in nanoseconds, of additional transmit antennas for the pre-VHT fields of the waveform. The first eight antennas use the cyclic shift values specified in Table 21-10 of [1]. The remaining L antennas use the values you specify in this property, where L = `NumTransmitAntennas` – 8. Specify this property as one of these values:

- An integer in the interval [–200, 0] – the `wlanVHTConfig` object uses this cyclic shift value for each of the L additional antennas.
- A row vector of length L of integers in the interval [–200, 0] – the `wlanVHTConfig` object uses the kth element as the cyclic shift value for the (k + 8)th transmit antenna.

Note If you specify this property as a row vector of length greater than L, the `wlanVHTConfig` object uses only the first L elements. For example, if you set the `NumTransmitAntennas` property to `16`, the `wlanVHTConfig` object uses only the first L= 16 – 8 = 8 elements of this vector.

Dependencies

To enable this property, set the `NumTransmitAntennas` property to a value greater than `8`.

Data Types: `double`

`NumSpaceTimeStreams` — Number of space-time streams

`1` (default) | integer in the interval [1, 8] | vector of integers in the interval [1, 4]

Number of space-time streams in the transmission, specified as one of these values:

- An integer in the interval [1, 8], applicable when the `NumUsers` property is `1`
- A 1-by-`NumUsers` vector of integers in the interval [1, 4], applicable when the `NumUsers` property is greater than `1`.

Example: `[1 3 2]` is the number of space-time streams for each user in a three-user transmission.

Note The sum of the elements of this property must not exceed eight.

Data Types: `double`

SpatialMapping — Spatial mapping scheme

`'Direct'` (default) | `'Hadamard'` | `'Fourier'` | `'Custom'`

Spatial mapping scheme, specified as `'Direct'`, `'Hadamard'`, `'Fourier'`, or `'Custom'`.

Dependencies

The default value, `'Direct'`, applies only when you set the `NumTransmitAntennas` and `NumSpaceTimeStreams` properties to the same value.

Data Types: `char` | `string`

SpatialMappingMatrix — Spatial mapping matrix

1 (default) | complex-valued scalar | complex-valued matrix | complex-valued 3-D array

Spatial mapping matrix, specified as one of these values:

- A complex-valued scalar. This value applies to all the subcarriers.
- A complex-valued matrix of size N_{STS} -by-N_T, where:
  - N_{STS} is the number of space-time streams;
  - N_T is the number of transmit antennas.

  In this case, the spatial mapping matrix applies to all the subcarriers.
- A complex-valued 3-D array of size N_{ST}-by-N_{STS}-by-N_T, where N_{ST} is the number of occupied subcarriers. The value of N_{ST} is the number of occupied subcarriers. The `ChannelBandwidth` property determines the value of N_{ST}. In this case, each occupied subcarrier has its own spatial mapping matrix.

  This table shows the `ChannelBandwidth` setting and the corresponding N_{ST}:

`ChannelBandwidth`	Number of Occupied Subcarriers, N_{ST}	Number of Data Subcarriers	Number of Pilot Subcarriers
`'CBW20'`	56	52	4
`'CBW40'`	114	108	6
`'CBW80'`	242	234	8
`'CBW160'`	484	468	16

Use this property to rotate and scale the output vector of the constellation mapper. The spatial mapping matrix is used for beamforming and mixing space-time streams over the transmit antennas. For more information, see Section 19.3.11.11.2 of [1]. The calling function normalizes the spatial mapping matrix for each subcarrier.

Example: `[0.5 0.3; 0.4 0.4; 0.5 0.8]` represents a spatial mapping matrix with three space-time streams and two transmit antennas.

Dependencies

This property applies only when you set the `SpatialMapping` property to `'Custom'`.

Data Types: `double`
Complex Number Support: Yes

`Beamforming` — Enable beamforming

`true` or `1` (default) | `false` or `0`

Enable beamforming, specified as a numeric or logical value of `1` (`true`) or `0` (`false`). To apply a beamforming steering matrix, set this property to `1` (`true`). The `SpatialMappingMatrix` property specifies the beamforming steering matrix.

Dependencies

This property applies only when the `NumUsers` property is set to `1` and the `SpatialMapping` property is set to `'Custom'`.

Data Types: `logical`

`STBC` — Enable STBC

`false` or `0` (default) | `true` or `1`

Enable space-time block coding (STBC) of the PPDU data field, specified as a numeric or logical value of `1` (`true`) or `0` (`false`). STBC transmits multiple copies of the data stream across assigned antennas.

- When you set this property to `0` (`false`), STBC is not applied to the data field. The number of space-time streams is equal to the number of spatial streams.
- When you set this property to `1` (`true`), STBC is applied to the data field. The number of space-time streams is twice the number of spatial streams.

For more information, see Section 22.3.10.9.4 of

Dependencies

This property applies only when the `NumUsers` property is `1`.

Data Types: `logical`

`MCS` — Modulation and coding scheme used for transmission

`0` (default) | integer in the interval [0, 9] | vector of integers

Modulation and coding scheme used for transmission, specified as one of these values:

- an integer in the interval [0, 9], applicable when the `NumUsers` property is `1`
- a 1-by-`NumUsers` vector of integers in the interval [0, 9], applicable when the `NumUsers` property is greater than `1`.

This table shows the modulation type and coding rate for each valid value of `MCS`:

`MCS`	Modulation	Coding Rate
`0`	Binary phase-shift keying (BPSK)	1/2

MCS	Modulation	Coding Rate
1	Quadrature phase-shift keying (QPSK)	1/2
2	QPSK	3/4
3	16-point quadrature amplitude modulation (16-QAM)	1/2
4	16-QAM	3/4
5	64-QAM	2/3
6	64-QAM	3/4
7	64-QAM	5/6
8	256-QAM	3/4
9	256-QAM	5/6

Data Types: `double`

`ChannelCoding` — FEC coding type
`'BCC'` (default) | `'LDPC'` | cell array of character vectors | cell array of strings

Forward-error-correction (FEC) coding type for the VHT-Data field, specified as one of these values:

- `'LDPC'` – Low-density parity-check (LDPC) coding applies to all users in the transmission
- `'BCC'` – binary convolutional coding (BCC) applies to all users in the transmission
- A 1-by-`NumUsers` cell array containing the values `'LDPC'` and `'BCC'`, where the *k*th element specifies the channel coding for user *k*

For more information, see section 21.3.10.5 of [1].

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

`APEPLength` — APEP length
`1024` (default) | nonnegative integer | vector of nonnegative integers

Aggregated MPDU (A-MPDU) pre-end-of-frame (pre-EOF) padding (APEP) length, in bytes.

- When the `NumUsers` property is `1`, specify this property as a nonnegative integer in the interval [0, 2^{20} – 1].
- When the `NumUsers` property is a value other than `1`, specify this property as a 1-by-`NumUsers` vector of integers in the interval [0, 2^{20} – 1].
- For a null data packet (NDP), set this property to `0`.

The object uses this property to determine the number of OFDM symbols in the data field. For more information, see Table 21-1 of [1].

Note This object supports only aggregated data transmission.

Data Types: `double`

`PSDULength` — PSDU length
integer in the interval [0, 2^{20} – 1] | vector of integers in the interval [0, 2^{20}–1]

This property is read-only.

Physical layer convergence procedure (PLCP) service data unit (PSDU) length, in bytes, specified as one of these values:

- An integer in the interval [0, 2^{20} – 1], applicable when the `NumUsers` property is `1`. A value of `0` corresponds to a null data packet (NDP).
- A vector of integers in the interval [0, 2^{20} – 1], applicable when the `NumUsers` property is greater than `1`.
- An empty array, applicable when this property is undefined, for example, when the set of property values is invalid.

The object calculates this property based on the value of the `APEPLength` property and other coding related properties. For more information, see section 21.4.3 of [1].

Example: `[1035 4150]` is the PSDU length vector for a `wlanVHTConfig` object where the `NumUsers` property is `2` and the `MCS` property is `[0 3]`.

Data Types: `double`

`GuardInterval` — Guard interval (cyclic prefix) duration

`'Long'` (default) | `'Short'`

Guard interval (cyclic prefix) duration for the data field within a packet specified as one of these values:

- `'Long'` – Guard interval duration of 800 ns
- `'Short'` – Guard interval duration of 400 ns

Data Types: `char` | `string`

`GroupID` — Group identification number

`63` (default) | integer in the interval [0, 63]

Group identification number, specified as an integer in the interval [0, 63].

Dependencies

The values `0` and `63` apply only when you set the `NumUsers` property to `1`. Values in the interval [1, 62] apply only when you set the `NumUsers` property to a value other than `1`.

Data Types: `double`

`PartialAID` — Abbreviated indication of PSDU recipients

`275` (default) | integer in the interval [0, 511]

Abbreviated indication of the PSDU recipients, specified as an integer in the interval [0, 511].

- For an uplink transmission, the partial identification number is the last nine bits of the basic service set identifier (BSSID).
- For a downlink transmission, the partial identification number is an identifier that combines the association ID with the BSSID of its serving AP.

For more information, see Table 21-1 of [1].

Data Types: `double`

Object Functions

transmitTime Packet transmission time

Examples

Create wlanVHTConfig Object for Single User

Create a VHT configuration object with the default settings.

```
cfgVHT = wlanVHTConfig
```

```
cfgVHT =
  wlanVHTConfig with properties:

       ChannelBandwidth: 'CBW80'
                NumUsers: 1
    NumTransmitAntennas: 1
    NumSpaceTimeStreams: 1
         SpatialMapping: 'Direct'
                   STBC: 0
                    MCS: 0
          ChannelCoding: 'BCC'
             APEPLength: 1024
          GuardInterval: 'Long'
                GroupID: 63
             PartialAID: 275

   Read-only properties:
             PSDULength: 1035
```

Update the channel bandwidth.

```
cfgVHT.ChannelBandwidth = 'CBW40'
```

```
cfgVHT =
  wlanVHTConfig with properties:

       ChannelBandwidth: 'CBW40'
                NumUsers: 1
    NumTransmitAntennas: 1
    NumSpaceTimeStreams: 1
         SpatialMapping: 'Direct'
                   STBC: 0
                    MCS: 0
          ChannelCoding: 'BCC'
             APEPLength: 1024
          GuardInterval: 'Long'
                GroupID: 63
             PartialAID: 275

   Read-only properties:
             PSDULength: 1030
```

Create wlanVHTConfig Object for Two Users

Create a VHT configuration object for a 20 MHz two-user transmission with one antenna per user.

Create a `wlanVHTConfig` object using a combination of name-value pairs and inline initialization to change default settings. Vector-valued properties apply user-specific settings.

```
cfgMU = wlanVHTConfig('ChannelBandwidth','CBW20','NumUsers',2, ...
    'GroupID',2,'NumTransmitAntennas',2);
cfgMU.NumSpaceTimeStreams = [1 1];
cfgMU.MCS = [4 8];
cfgMU.APEPLength = [1024 2048];
cfgMU.ChannelCoding = {'BCC' 'LDPC'}
```

```
cfgMU =
  wlanVHTConfig with properties:

       ChannelBandwidth: 'CBW20'
               NumUsers: 2
          UserPositions: [0 1]
    NumTransmitAntennas: 2
    NumSpaceTimeStreams: [1 1]
         SpatialMapping: 'Direct'
                    MCS: [4 8]
          ChannelCoding: {'BCC'  'LDPC'}
             APEPLength: [1024 2048]
          GuardInterval: 'Long'
                GroupID: 2

   Read-only properties:
             PSDULength: [1030 2065]
```

The configuration object settings reflect the updates specified. Properties that aren't modified take their default values.

More About

PPDU

The physical layer (PHY) protocol data unit (PPDU) is the complete physical layer convergence procedure (PLCP) frame, including PLCP headers, MAC headers, the MAC data field, and the MAC and PLCP trailers.

Version History

Introduced in R2015b

References

[1] IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations: After the first use of this object for a VHT MU-MIMO transmission, you cannot change the number of elements in any of these properties.

- `UserPositions`
- `NumSpaceTimeStreams`
- `MCS`
- `ChannelCoding`
- `APEPLength`

In addition, if you specify the `ChannelCoding` property as a cell array, you cannot change any of the elements of this property after the first use of this object for a VHT MU-MIMO transmission.

See Also

Objects

`wlanDMGConfig` | `wlanHESUConfig` | `wlanHEMUConfig` | `wlanHETBConfig` | `wlanHTConfig` | `wlanNonHTConfig` | `wlanS1GConfig`

Functions

`transmitTime` | `wlanVHTData` | `wlanVHTDataRecover` | `wlanVHTLTF` | `wlanVHTLTFDemodulate` | `wlanVHTOFDMInfo` | `wlanVHTSIGA` | `wlanVHTSIGARecover` | `wlanVHTSIGB` | `wlanVHTSIGBRecover` | `wlanVHTSTF` | `wlanWaveformGenerator`

Apps

WLAN Waveform Generator

Topics

“Packet Size and Duration Dependencies”

wlanWURConfig

Configure WUR transmission

Description

The `wlanWURConfig` object is a configuration object for the WLAN wake-up radio (WUR) packet format.

Creation

Syntax

```
cfgWUR = wlanWURConfig
cfgWUR = wlanWURConfig(numSubchannels)
cfgWUR = wlanWURConfig( ___ ,Name=Value)
```

Description

`cfgWUR = wlanWURConfig` creates a configuration object that initializes transmit parameters for an IEEE 802.11ba™ WUR "PPDU" on page 4-280 with one 20 MHz subchannel. For a detailed description of the WUR WLAN format, see [1].

`cfgWUR = wlanWURConfig(numSubchannels)` parameterizes a WUR transmission with the specified number of 20 MHz subchannels. Specify `numSubchannels` as `1`, `2`, or `4`.

`cfgWUR = wlanWURConfig( ___ ,Name=Value)` sets "Properties" on page 4-277 using one or more name-value arguments. For example, `NumTransmitAntennas=3` specifies three transmit antennas.

Properties

`Subchannel` — Subchannel parameters

cell array of `wlanWURSubchannel` objects

Subchannel parameters, specified as a cell array of `wlanWURSubchannel` objects. Each element of the cell array contains properties to configure a 20 MHz subchannel. The default value is a 1-by-1 cell array containing a `wlanWURSubchannel` object with default property values.

`NumTransmitAntennas` — Number of transmit antennas

`1` (default) | integer in the interval [1, 8]

Number of transmit antennas, specified as an integer in the interval [1, 8].

Data Types: `double`

`ChannelBandwidth` — Channel bandwidth of PPDU transmission

`'CBW20'` (default) | `'CBW40'` | `'CBW80'`

This property is read-only.

Channel bandwidth of the PPDU transmission, returned as one of these values.

- `'CBW20'` — Channel bandwidth of 20 MHz
- `'CBW40'` — Channel bandwidth of 40 MHz
- `'CBW80'` — Channel bandwidth of 80 MHz

Data Types: `char` | `string`

`NumUsers` — Number of users in transmission
1 (default) | 2 | 3 | 4

This property is read-only.

Number of users in the transmission, returned as 1, 2, 3, or 4.

Data Types: `double`

Object Functions

getActiveSubchannelIndex	Active subchannel indices
getPSDULength	Calculate HE or WUR PSDU length
packetFormat	WLAN packet format
transmitTime	Packet transmission time

Examples

Generate Default WUR Waveform

Configure and generate a WLAN waveform containing a WUR packet with default settings.

Parameterize the transmission by creating a WUR configuration object with default property values.

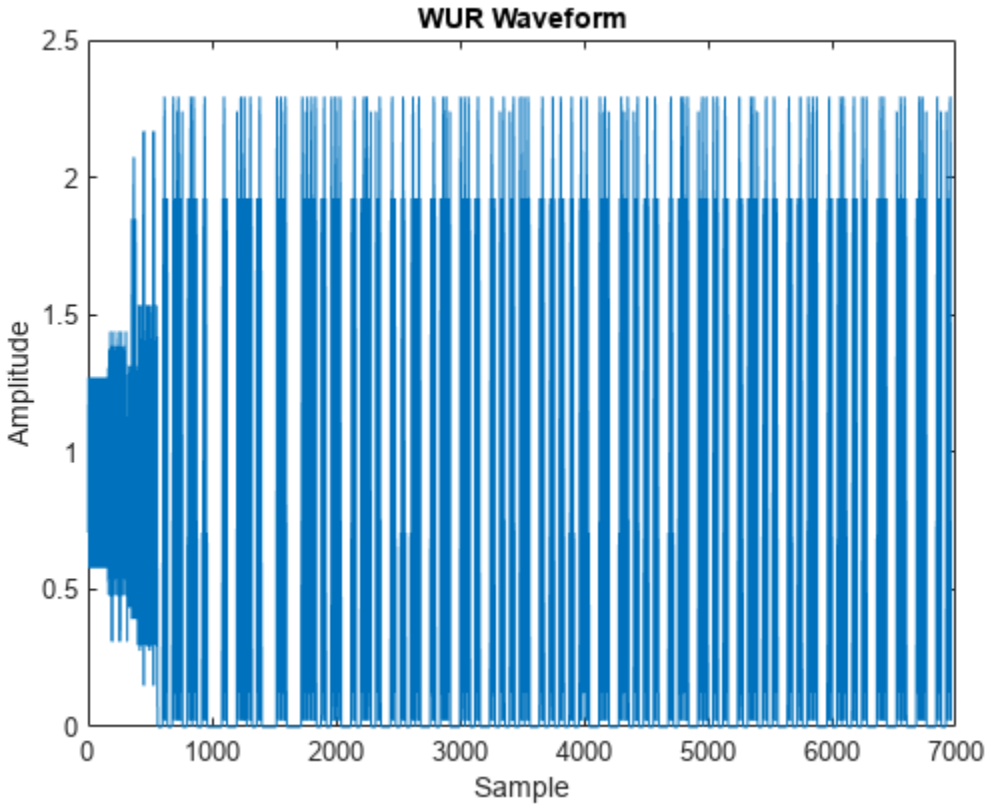
```
cfgWUR = wlanWURConfig;
```

Generate a PSDU of relevant length.

```
psduLength = getPSDULength(cfgWUR);
psdu = randi([0 1],8*psduLength,1);
```

Generate and plot the waveform.

```
waveform = wlanWaveformGenerator(psdu,cfgWUR);
figure
plot(abs(waveform))
title('WUR Waveform')
xlabel('Sample')
ylabel('Amplitude')
```



Generate Oversampled WUR Waveform

Create a WUR configuration object, specifying four 20 MHz subchannels.

```
numSubchannels = 4;
cfgWUR = wlanWURConfig(numSubchannels);
```

Configure each 20 MHz subchannel.

```
psduLength = [4 8 12 16];
dataRate = {'LDR','HDR','LDR','HDR'};
design = {'Example1','Example2','Example1','Example1'};
cfgSubchannel = cell(1,numSubchannels);
psdu = cell(1,cfgWUR.NumUsers);
for i = 1:cfgWUR.NumUsers
    cfgSubchannel{i} = wlanWURSubchannel(PSDULength=psduLength(i), ...
        DataRate=dataRate{i},SymbolDesign=design{i});
    psdu{i} = randi([0 1],8*psduLength(i),1,'int8');
end
cfgWUR.Subchannel = cfgSubchannel;
```

Specify an oversampling factor of two, and then generate the waveform.

```
osf = 2;
waveform = wlanWaveformGenerator(psdu,cfgWUR,NumPackets=4, ...
    IdleTime=1e-5,OversamplingFactor=osf);
```

Get Active Subchannel Indices of WUR Transmission

Create a WUR configuration object for a transmission with two subchannels and four transmit antennas.

```
numSubchannels = 2;
cfgWUR = wlanWURConfig(numSubchannels,NumTransmitAntennas=4);
```

Get the active subchannel indices for the transmission.

```
idx = getActiveSubchannelIndex(cfgWUR)
```

```
idx = 1×2

     1     2
```

More About

PPDU

The physical layer (PHY) protocol data unit (PPDU) is the complete physical layer convergence procedure (PLCP) frame, including PLCP headers, MAC headers, the MAC data field, and the MAC and PLCP trailers.

Version History

Introduced in R2021b

References

[1] IEEE Std 802.11ba-2021. “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 3: Wake-Up Radio Operation.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems. Local and Metropolitan Area Networks — Specific Requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`transmitTime` | `wlanWaveformGenerator` | `getActiveSubchannelIndex` | `getPSDULength` | `packetFormat`

Objects

`wlanWURSubchannel`

Apps
WLAN Waveform Generator

Topics
“802.11ba WUR Waveform Generation and Analysis”

wlanWURSubchannel

Configure WUR 20 MHz subchannel

Description

The `wlanWURSubchannel` object configures a 20 MHz subchannel of a WLAN wake-up radio (WUR) transmission.

Creation

Syntax

```
cfgSubchannel = wlanWURSubchannel
cfgSubchannel = wlanWURSubchannel(Name=Value)
```

Description

`cfgSubchannel = wlanWURSubchannel` configures a 20 MHz subchannel of an IEEE 802.11ba WUR transmission.

`cfgSubchannel = wlanWURSubchannel(Name=Value)` sets “Properties” on page 4-282 using one or more name-value arguments. For example, `DataRate='LDR'` specifies a data rate of 62.5 kb/s.

Properties

`DataRate` — Data rate
`'HDR'` (default) | `'LDR'`

Data rate, specified as one of these values.

- `'HDR'` — Data rate of 250 kb/s
- `'LDR'` — Data rate of 62.5 kb/s

Data Types: `char` | `string`

`PSDULength` — PSDU length
8 (default) | integer in the interval [1, 22]

PSDU length, in bytes, specified as an integer in the interval [1, 22].

Data Types: `double`

`SymbolDesign` — MC-OOK On symbol sequence and CSD values
`'Example1'` (default) | `'Example2'` | `'Example3'` | `'User-defined'`

Multicarrier on-off keying (MC-OOK) On symbol sequence and cyclic shift diversity (CSD) values for the WUR-Sync and WUR-Data fields, specified as one of these values.

- `'Example1'` — Use MC-OOK On symbol sequence and CSD values corresponding to Example 1 in Tables AC-1, AC-2, AC-3, and AC-4 of [1].
- `'Example2'` — Use MC-OOK On symbol sequence and CSD values corresponding to Example 2 in Tables AC-1, AC-2, AC-3, and AC-4 of [1].
- `'Example3'` — Use MC-OOK On symbol sequence and CSD values corresponding to Example 3 in Tables AC-1, AC-2, AC-3, and AC-4 of [1].
- `'User-defined'` — Use MC-OOK On symbol sequence and CSD values corresponding to the `HDRSequence`, `LDRSequence`, `HDRCSD`, and `LDRCSD` properties.

Data Types: `char` | `string`

`HDRSequence` — Normalized HDR MC-OOK On symbol sequence

complex-valued row vector

Normalized high-data-rate (HDR) MC-OOK On symbol sequence for the WUR-Sync and WUR-Data fields, specified as a complex-valued row vector of length 13.

Dependencies

To enable this property, set the `SymbolDesign` property to `'User-defined'`.

Data Types: `double`
Complex Number Support: Yes

`LDRSequence` — Normalized LDR MC-OOK On symbol sequence

complex-valued row vector

Normalized low-data-rate (LDR) MC-OOK On symbol sequence for the WUR-Data field, specified as a complex-valued row vector of length 13.

Dependencies

To enable this property, set the `SymbolDesign` property to `'User-defined'`.

Data Types: `double`
Complex Number Support: Yes

`HDRCSD` — HDR CSD values

real-valued row vector

HDR CSD values, in nanoseconds, for the WUR-Sync and WUR-Data fields, specified as a real-valued row vector of length N_T, the number of transmit antennas in the WUR transmission. When you use this object as an element of the `Subchannel` property of a `wlanWURConfig` object, N_T must be the value of the `NumTransmitAntennas` property of that object.

Each element of this vector must be nonpositive.

Note If you set this property as a row vector of length greater than N_T, the object uses only the first N_T elements. For example, if $N_T = 4$, the object uses only the first four elements of this vector.

Dependencies

To enable this property, set the `SymbolDesign` property to `'User-defined'`.

Data Types: `double`

`LDRCSD` — LDR CSD values

real-valued row vector

LDR CSD values, in nanoseconds, for the WUR-Data field, specified as a real-valued row vector of length N_T, the number of transmit antennas in the WUR transmission. When you use this object as an element of the `Subchannel` property of a `wlanWURConfig` object, N_T must be the value of the `NumTransmitAntennas` property of that object.

Each element of this vector must be nonpositive.

Note If you set this property as a row vector of length greater than N_T, the object uses only the first N_T elements. For example, if $N_T = 4$, the object uses only the first four elements of this vector.

Dependencies

To enable this property, set the `SymbolDesign` property to `'User-defined'`.

Data Types: `double`

`Enabled` — Subchannel puncturing indication

`true` or `1` (default) | `false` or `0`

Subchannel puncturing indication, specified as a numeric or logical `1` (`true`) or `0` (`false`). To puncture the subchannel, set this property to `false`. Otherwise, set this property to `true`.

When you use this object as an element of the `Subchannel` property of a `wlanWURConfig` object that configures a 20 or 40 MHz transmission, you must set this property to `true`.

When you use this object as an element of the `Subchannel` property of a `wlanWURConfig` object that configures an 80 MHz transmission, you must set this property to `true` in at least one of the `wlanWURSubchannel` objects.

Data Types: `double` | `logical`

Examples

Generate Oversampled WUR Waveform

Create a WUR configuration object, specifying four 20 MHz subchannels.

```
numSubchannels = 4;
cfgWUR = wlanWURConfig(numSubchannels);
```

Configure each 20 MHz subchannel.

```
psduLength = [4 8 12 16];
dataRate = {'LDR','HDR','LDR','HDR'};
design = {'Example1','Example2','Example1','Example1'};
cfgSubchannel = cell(1,numSubchannels);
psdu = cell(1,cfgWUR.NumUsers);
for i = 1:cfgWUR.NumUsers
    cfgSubchannel{i} = wlanWURSubchannel(PSDULength=psduLength(i), ...
        DataRate=dataRate{i},SymbolDesign=design{i});
    psdu{i} = randi([0 1],8*psduLength(i),1,'int8');
```

```
end
cfgWUR.Subchannel = cfgSubchannel;
```

Specify an oversampling factor of two, and then generate the waveform.

```
osf = 2;
waveform = wlanWaveformGenerator(psdu,cfgWUR,NumPackets=4, ...
    IdleTime=1e-5,OversamplingFactor=osf);
```

Version History

Introduced in R2021b

References

[1] IEEE Std 802.11ba-2021. “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 3: Wake-Up Radio Operation.” IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems. Local and Metropolitan Area Networks — Specific Requirements.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`wlanWURConfig`

Topics

“802.11ba WUR Waveform Generation and Analysis”