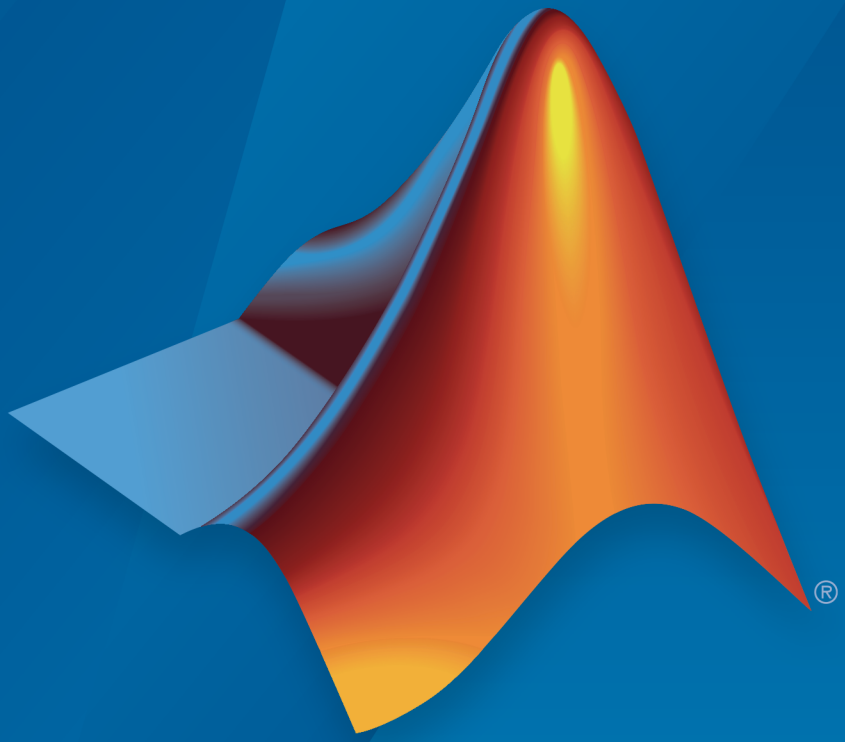
# WLAN System Toolbox™

## Getting Started Guide



MATLAB®

R2016b

MathWorks®

# How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

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

Phone: 508-647-7000

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

*WLAN System Toolbox™ Getting Started Guide*



#### Revision History

| | | |
|---|---|---|
| October 2015 | Online only | New for Version 1.0 (R2015b) |
| March 2016 | Online only | Revised for Version 1.1 (Release 2016a) |
| September 2016 | Online only | Revised for Version 1.2 (Release 2016b) |

# Contents

## 1 Introduction



## 2 Tutorials

# 3 About WLAN

1

# Introduction

# WLAN System Toolbox Product Description

**Simulate, analyze, and test the physical layer of WLAN communications systems**

WLAN System Toolbox provides standard-compliant functions for the design, simulation, analysis, and testing of wireless LAN communications systems. The system toolbox provides configurable physical layer waveforms for IEEE® 802.11ac™ and 802.11b/a/g/n/j/p standards. It also provides transmitter, channel modeling, and receiver operations, including channel coding, modulation (OFDM, DSSS, and CCK), spatial stream mapping, channel models (TGac and TGn), and MIMO receivers.

You can generate very high throughput (VHT), high throughput (HT-mixed), and legacy (non-HT) signals, and demodulate VHT, HT-mixed, and non-HT OFDM signals. You can also perform measurements such as channel power, spectrum mask, and occupied bandwidth, and create test benches for the end-to-end simulation of WLAN communications links.

The system toolbox provides reference designs to help you explore baseband specifications and study the effects of RF designs and interference sources on system performance. Using WLAN System Toolbox with hardware support packages, you can connect your transmitter and receiver models to radio devices and verify your designs via over-the-air transmission and reception.

## Key Features

- IEEE 802.11ac and 802.11b/a/g/n/j/p standard-compliant physical layer models
- Very high throughput (VHT), high throughput (HT-mixed), and legacy (non-HT) waveform generation
- Channel coding, modulation (OFDM, DSSS, CCK), spatial stream mapping, and MIMO receivers
- Channel models, including TGac and TGn
- Measurements including channel power, spectrum mask, EVM, PER, and occupied bandwidth
- Waveform transmission and reception with radio devices and instruments
- C code generation support

# Expected Background

This documentation assumes that you already have background knowledge in the subject of digital communications, WLAN, and IEEE 802.11™ standards. A standard communications text, in addition to the books and web links listed in the Selected Bibliography subsections accompanying many topics can be used to acquire sufficient background understanding.

Continue reading and try the examples. Then, read subsequent content pertaining to your specific areas of interest. As you learn which functions you want to use, refer to the online reference pages for more information.

# C Code Generation with Functions and System Objects

These functions and System objects support code generation from MATLAB® code. Code generation from MATLAB code requires the MATLAB Coder™ software.

| Name | Remarks and Limitations |
|---|---|
| **WLAN Modeling** | |
| `wlanHTConfig` | — |
| `wlanNonHTConfig` | — |
| `wlanRecoveryConfig` | — |
| `wlanS1GConfig` | — |
| `wlanVHTConfig` | — |
| **Signal Transmission** | |
| `wlanHTData` | — |
| `wlanHTLTF` | — |
| `wlanHTSIG` | — |
| `wlanHTSTF` | — |
| `wlanLLTF` | — |
| `wlanLSIG` | — |
| `wlanLSTF` | — |
| `wlanNonHTData` | — |
| `wlanVHTData` | — |
| `wlanVHTLTF` | — |
| `wlanVHTSIGA` | — |
| `wlanVHTSIGB` | — |
| `wlanVHTSTF` | — |
| `wlanWaveformGenerator` | — |
| **Signal Reception** | |
| `wlanCoarseCFOEstimate` | — |
| `wlanFormatDetect` | — |

| Name | Remarks and Limitations |
|---|---|
| `wlanFieldIndices` | — |
| `wlanFineCFOEstimate` | — |
| `wlanHTDataRecover` | — |
| `wlanHTLTFChannelEstimate` | — |
| `wlanHTLTFDemodulate` | — |
| `wlanHTSIGRecover` | — |
| `wlanLLTFChannelEstimate` | — |
| `wlanLLTFDemodulate` | — |
| `wlanLSIGRecover` | — |
| `wlanNonHTDataRecover` | — |
| `wlanPacketDetect` | — |
| `wlanVHTDataRecover` | — |
| `wlanVHTLTFChannelEstimate` | — |
| `wlanVHTLTFDemodulate` | — |
| `wlanVHTSIGARecover` | — |
| `wlanVHTSIGBRecover` | — |
| **Propagation Channel** | |
| wlanTGacChannel | "System Objects in MATLAB Code Generation" |
| wlanTGnChannel | "System Objects in MATLAB Code Generation" |

**Note:** WLAN System Toolbox functionality with the `MATLAB Function` block is not supported.

# Tutorials

# Create Configuration Objects

WLAN System Toolbox uses value objects to organize properties required for generation of IEEE 802.11 b/a/g/n/ac waveforms and to recover signal data from such waveforms. After you create the various configuration objects described here, you can use them to generate waveforms.

## Create VHT Configuration Object

This example shows how to create VHT configuration objects. It also shows how to change the default property settings by using dot notation or by overriding the default settings by using `Name,Value` pairs when creating the object.

### Create Object and Then Modify Properties

Create a VHT configuration object and view the default settings.

```
vht = wlanVHTConfig
```

```
vht =

  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
```

Modify the defaults to specify a 160 MHz channel bandwidth, two transmit antennas, and two space-time streams.

```
vht.ChannelBandwidth = 'CBW160';
```

```
vht.NumTransmitAntennas = 2;
vht.NumSpaceTimeStreams = 2
```

```
vht =

  wlanVHTConfig with properties:

       ChannelBandwidth: 'CBW160'
                NumUsers: 1
    NumTransmitAntennas: 2
    NumSpaceTimeStreams: 2
         SpatialMapping: 'Direct'
                   STBC: 0
                    MCS: 0
          ChannelCoding: 'BCC'
             APEPLength: 1024
          GuardInterval: 'Long'
                GroupID: 63
             PartialAID: 275
```

### Create Object and Overide Default Property Values

Create a VHT configuration object. Use `Name,Value` pairs to set the MCS to 7 and to specify two transmit antennas.

```
vht2 = wlanVHTConfig('MCS',7,'NumTransmitAntennas',2)
```

```
vht2 =

  wlanVHTConfig with properties:

       ChannelBandwidth: 'CBW80'
                NumUsers: 1
    NumTransmitAntennas: 2
    NumSpaceTimeStreams: 1
         SpatialMapping: 'Direct'
                   STBC: 0
                    MCS: 7
          ChannelCoding: 'BCC'
             APEPLength: 1024
          GuardInterval: 'Long'
                GroupID: 63
```

```
            PartialAID: 275
```

As currently configured, this object is not a valid VHT configuration. Validation of the object occurs when it is input to a calling function. When spatial mapping is `Direct`, the number of space-time streams must equal the number of transmit antennas. Change the number of space time streams to match the number of transmit antennas is one option to make the configuration of the object valid.

```
vht2.NumSpaceTimeStreams = 2
```

```
vht2 =

  wlanVHTConfig with properties:

      ChannelBandwidth: 'CBW80'
               NumUsers: 1
    NumTransmitAntennas: 2
    NumSpaceTimeStreams: 2
         SpatialMapping: 'Direct'
                   STBC: 0
                    MCS: 7
          ChannelCoding: 'BCC'
             APEPLength: 1024
          GuardInterval: 'Long'
                GroupID: 63
             PartialAID: 275
```

## Create HT Configuration Object

This example shows how to create HT configuration objects. It also shows how to change the default property settings by using dot notation or by overriding the default settings by using `Name,Value` pairs when creating the object.

### Create Object and Then Modify Properties

Create an HT configuration object and view the default settings.

```
ht = wlanHTConfig
```

```
ht =

  wlanHTConfig with properties:

       ChannelBandwidth: 'CBW20'
    NumTransmitAntennas: 1
    NumSpaceTimeStreams: 1
         SpatialMapping: 'Direct'
                    MCS: 0
          GuardInterval: 'Long'
          ChannelCoding: 'BCC'
             PSDULength: 1024
     RecommendSmoothing: 1
```

Modify the defaults to specify three transmit antennas and two space-time streams.

```
ht.NumTransmitAntennas = 3;
ht.NumSpaceTimeStreams = 2
```

```
ht =

  wlanHTConfig with properties:

       ChannelBandwidth: 'CBW20'
    NumTransmitAntennas: 3
    NumSpaceTimeStreams: 2
    NumExtensionStreams: 0
         SpatialMapping: 'Direct'
                    MCS: 0
          GuardInterval: 'Long'
          ChannelCoding: 'BCC'
             PSDULength: 1024
     RecommendSmoothing: 1
```

As the settings of the object are modified, the set of properties that apply for the current configuration are shown. When the number of transmit antennas is more than the number of space-time streams, the number of extension streams property applies and is shown. Also, as currently configured, this object is not a valid HT configuration because the default `'Direct'` spatial mapping requires the number of space-time streams to equal the number of transmit antennas. Validation of the object occurs when it is input to a calling function.

### Create Object and Override Default Property Values

Create an HT configuration object. Use `Name,Value` pairs to define a sounding packet by specifying `PSDULength` = 0, and set the number of transmit antennas and space-time streams to 3.

```
ht2 = wlanHTConfig('PSDULength',0,'NumTransmitAntennas',3,'NumSpaceTimeStreams',3)
```

```
ht2 =

  wlanHTConfig with properties:

       ChannelBandwidth: 'CBW20'
    NumTransmitAntennas: 3
    NumSpaceTimeStreams: 3
         SpatialMapping: 'Direct'
                    MCS: 0
          GuardInterval: 'Long'
          ChannelCoding: 'BCC'
              PSDULength: 0
     RecommendSmoothing: 1
```

## Create Non-HT Configuration Object

This example shows how to create non-HT configuration objects. It also shows how to change the default property settings by using dot notation or by overriding the default settings by using `Name,Value` pairs when creating the object.

### Create Object and Then Modify Properties

Create a non-HT configuration object and view the default settings.

```
nonHT = wlanNonHTConfig
```

```
nonHT =

  wlanNonHTConfig with properties:

              Modulation: 'OFDM'
       ChannelBandwidth: 'CBW20'
                    MCS: 0
              PSDULength: 1000
```

```
    NumTransmitAntennas: 1
```

Modify the defaults to specify four transmit antennas and to set the MCS to 3.

```
nonHT.NumTransmitAntennas = 4;
nonHT.MCS = 3
```

```
nonHT =

  wlanNonHTConfig with properties:

            Modulation: 'OFDM'
      ChannelBandwidth: 'CBW20'
                   MCS: 3
             PSDULength: 1000
    NumTransmitAntennas: 4
```

### Create Object and Overide Default Property Values

Create a non-HT configuration object. Use a `Name,Value` pair change the modulation scheme to `DSSS`.

```
nonHT2 = wlanNonHTConfig('Modulation','DSSS')
```

```
nonHT2 =

  wlanNonHTConfig with properties:

      Modulation: 'DSSS'
        DataRate: '1Mbps'
    LockedClocks: 1
      PSDULength: 1000
```

For the DSSS modulation scheme, a different set of properties apply and are shown for the non-HT configuration object.

## Create Recovery Configuration Object

Recovery configuration objects are used to specify receiver algorithms and settings to use for recovery. This example shows how to create recovery configuration objects. It also

shows how to change the default property settings by using dot notation or by overriding the default settings by using `Name,Value` pairs when creating the object.

### Create Object and Then Modify Properties

Create a recovery configuration object and view default settings.

```
cfgRec = wlanRecoveryConfig
```

```
cfgRec =

  wlanRecoveryConfig with properties:

            OFDMSymbolOffset: 0.7500
          EqualizationMethod: 'MMSE'
          PilotPhaseTracking: 'PreEQ'
    MaximumLDPCIterationCount: 12
             EarlyTermination: 0
```

Modify the default to specify no pilot phase tracking.

```
cfgRec.PilotPhaseTracking = 'None'
```

```
cfgRec =

  wlanRecoveryConfig with properties:

            OFDMSymbolOffset: 0.7500
          EqualizationMethod: 'MMSE'
          PilotPhaseTracking: 'None'
    MaximumLDPCIterationCount: 12
             EarlyTermination: 0
```

### Create Object and Overide Default Property Values

Use `Name,Value` pairs to create a recovery configuration object to perform zero-forcing equalization, using an OFDM symbol sampling offset of 0.6 in the recovery process.

```
cfgRec = wlanRecoveryConfig('OFDMSymbolOffset',0.6,'EqualizationMethod','ZF')
```

```
cfgRec =

  wlanRecoveryConfig with properties:

            OFDMSymbolOffset: 0.6000
          EqualizationMethod: 'ZF'
          PilotPhaseTracking: 'PreEQ'
    MaximumLDPCIterationCount: 12
             EarlyTermination: 0
```

## See Also

wlanHTConfig Properties | wlanNonHTConfig Properties | wlanRecoveryConfig Properties | wlanVHTConfig Properties

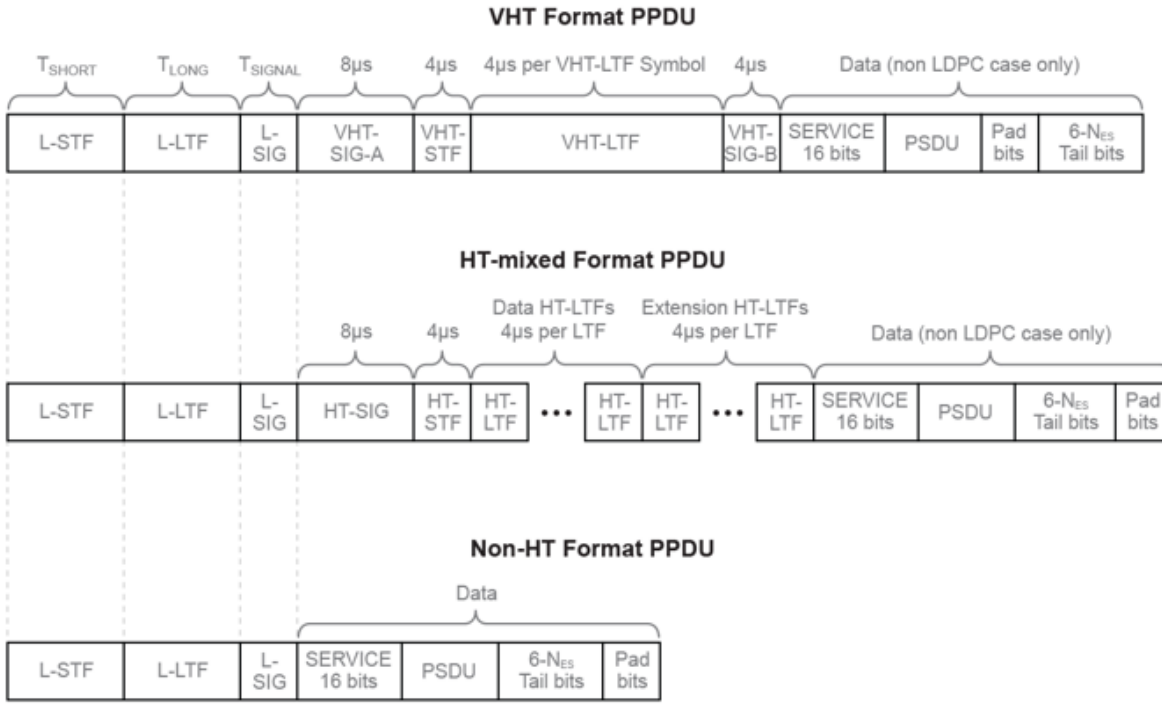
## Related Examples

- “Waveform Generation” on page 2-10
- “What Is WLAN?” on page 3-2

# Waveform Generation

After you create the necessary configuration objects described in "Create Configuration Objects" on page 2-2, you can use the objects to generate the desired WLAN format waveform.

The IEEE 802.11 [12] standards define a physical layer conformance procedure (PLCP) protocol data unit (PPDU) as the transmission unit at the physical layer. See "WLAN Packet Structure" for more details. The VHT, HT, and non-HT PPDU formats consist of preamble and data fields.




**S1G Format PPDU**

1. IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.
2. IEEE Std 802.11ac-2013 Adapted and reprinted with permission from IEEE. Copyright IEEE 2013. All rights reserved.

The S1G format has there three PPDU types:

- *S1G_LONG* PPDU for S1G ≥ 2 MHz transmissions with a long preamble
- *S1G_SHORT* PPDU for S1G ≥ 2 MHz transmissions with a short preamble
- *S1G_1M* PPDU for 1 MHz transmissions

Each S1G PPDU type has a specific preamble structure. The field structure for S1G PPDUs consist of preamble and data portions:

- An S1G ≥ 2 MHz long preamble PPDU supports single-user and multi-user transmissions. The long preamble PPDU consists of an omnidirectional portion and a beam-changeable portion.
  - The omnidirectional portion is transmitted without beamforming to all users . It consists of three fields:
    - STF — The short training field is 2 symbols long. It is used for coarse synchronization.
    - LTF1 — The first long training field is 2 symbols long. It is used for fine synchronization and initial channel estimation.
    - SIG-A — The signaling A field is 2 symbols long. The receiver decodes the signaling A field to determine transmission parameters relevant to all users.
  - The beam-changeable portion can be beamformed to each user. It consists of four fields:
    - D-STF - The beamformed short training field is 1 symbol long. It is used by the receiver for automatic gain control.
    - D-LTF-N - The beamformed long training fields are each 1 symbol long per D-LTF. They are used for MIMO channel estimation.
    - SIG-B - The signaling B field is 1 symbol long. In a multi-user transmission, the SIG-B field signals the MCS for each user. In a single-user transmission, the MCS is signaled in the SIG-A field of the omnidirectional portion of the preamble. Therefore, in a single-user transmission, the SIG-B symbol transmitted is an exact repetition of the first D-LTF. This repetition allows for improved channel estimation.
    - Data - The data field is variable in length. It carries the user data payload.
- An S1G ≥ 2 MHz short preamble PPDU supports single-user transmissions. All fields in the PPDU can be beamformed. The PPDU consists of five fields:

  - STF — The short training field is 2 symbols long. It is used for coarse synchronization.
  - LTF1 — The first long training field is 2 symbols long. It is used for fine synchronization and initial channel estimation.
  - SIG — The signaling field is 2 symbols long. The receiver decodes the signaling field to determine transmission parameters.
  - LTF2-N — The subsequent long training fields are 1 symbol long per LTF. They are used for MIMO channel estimation. $N_{SYMBOLS} = 1$ per subsequent LTF.
  - Data — The data field is variable in length. It carries the user data payload.
- An S1G 1 MHz PPDU supports single-user transmissions. It is composed of the same five fields as the S1G ≥ 2 MHz short preamble PPDU. All fields can be beamformed. An S1G 1 MHz PPDU has longer STF, LTF1, and SIG fields than other S1G format PPDUs. This narrower bandwidth mode can achieve similar sensitivity to the S1G ≥ 2 MHz short preamble PPDU transmissions. The STF is 4 symbols, the LTF1 is 4 symbols, and the SIG is 6 symbols.

Use WLAN System Toolbox functions to generate a full PPDU waveform or individual PPDU field waveforms.

Generate a full PPDU waveform using the `wlanWaveformGenerator` function to populate all PPDU fields (preamble and data) in a single call. `wlanWaveformGenerator` accepts a bit stream, a format configuration object (`wlanS1GConfig` `wlanVHTConfig`, `wlanHTConfig`, or `wlanNonHTConfig`) and `Name,Value` pairs to configure the waveform.

## Generate WLAN Waveforms

Generate S1G, VHT, HT-mixed, and non-HT format waveforms. Vary configuration parameters and plot the waveforms to highlight differences in waveforms and sample rates.

In each section of this example, you:

- Create a format-specific configuration object.
- Create a vector of information bits for the packet data payload. Internally, the `wlanWaveformGeneration` function loops through the bits vector as many times as needed to generate the specified number of packets.

- Generate the format-specific waveform and plot it. For plotting, because no filtering is applied to the waveform and the oversampling rate is 1, set the sampling rate equal to the channel bandwidth.

### Generate S1G Format Waveform

Create an S1G configuration object and waveform. Using `Name,Value` pairs, specify 4 MHz channel bandwidth, 3 packets, and 15 microseconds of idle time. Display the configuration object and inspect its properties and settings.

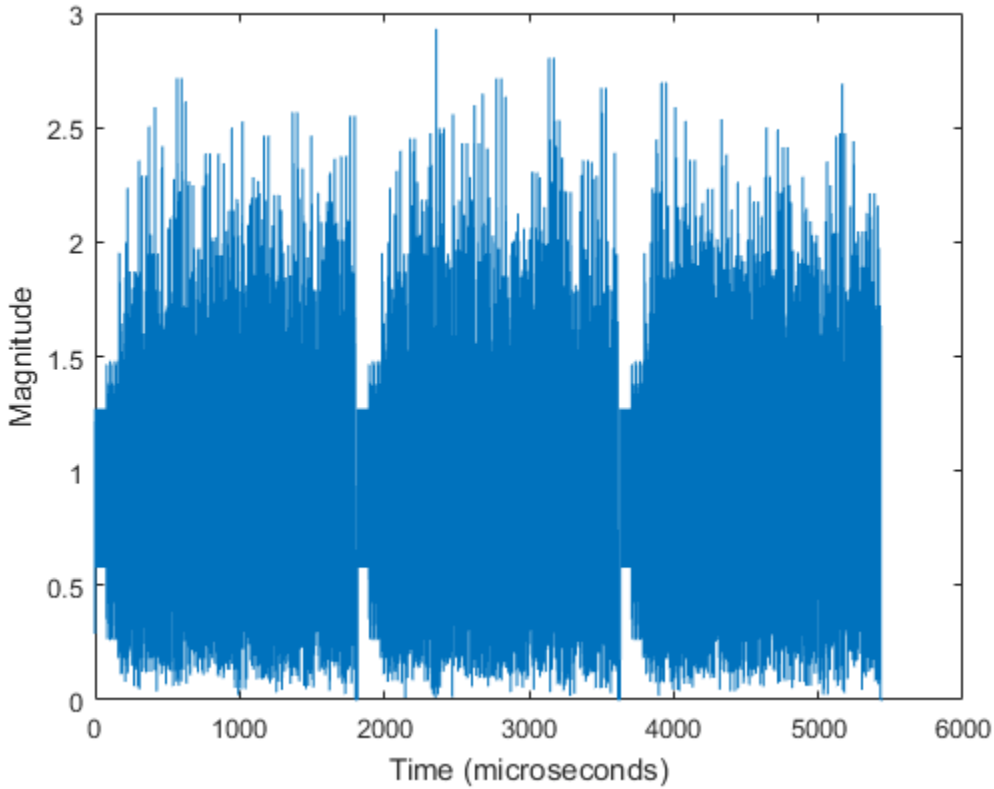
```
s1g = wlanS1GConfig('ChannelBandwidth','CBW4')
bits = [1;0;0;1;1];

s1gWaveform = wlanWaveformGenerator(bits,s1g, ...
    'NumPackets',3,'IdleTime',15e-6);
```

```
s1g =

  wlanS1GConfig with properties:

       ChannelBandwidth: 'CBW4'
              Preamble: 'Short'
              NumUsers: 1
    NumTransmitAntennas: 1
    NumSpaceTimeStreams: 1
         SpatialMapping: 'Direct'
                   STBC: 0
                    MCS: 0
          ChannelCoding: 'BCC'
             APEPLength: 256
          GuardInterval: 'Long'
             PartialAID: 37
       UplinkIndication: 0
                  Color: 0
         TravelingPilots: 0
     ResponseIndication: 'None'
     RecommendSmoothing: 1
```

Plot the S1G format waveform, scaling the *x-axis* relative to the channel bandwidth.

```
fs = 4e6; % Set sampling frequency equal to the channel bandwidth
time = ([0:length(s1gWaveform)-1]/fs)*1e6;
```

```
plot(time,abs(s1gWaveform))
xlabel ('Time (microseconds)');
ylabel('Magnitude');
```




The plot shows three S1G format packets, with each packet separated by 15 microseconds of idle time.

### Generate VHT Format Waveform

Create a VHT configuration object and waveform. Using `Name,Value` pairs, specify 5 packets and 20 microseconds of idle time. Display the configuration object and inspect its properties and settings.

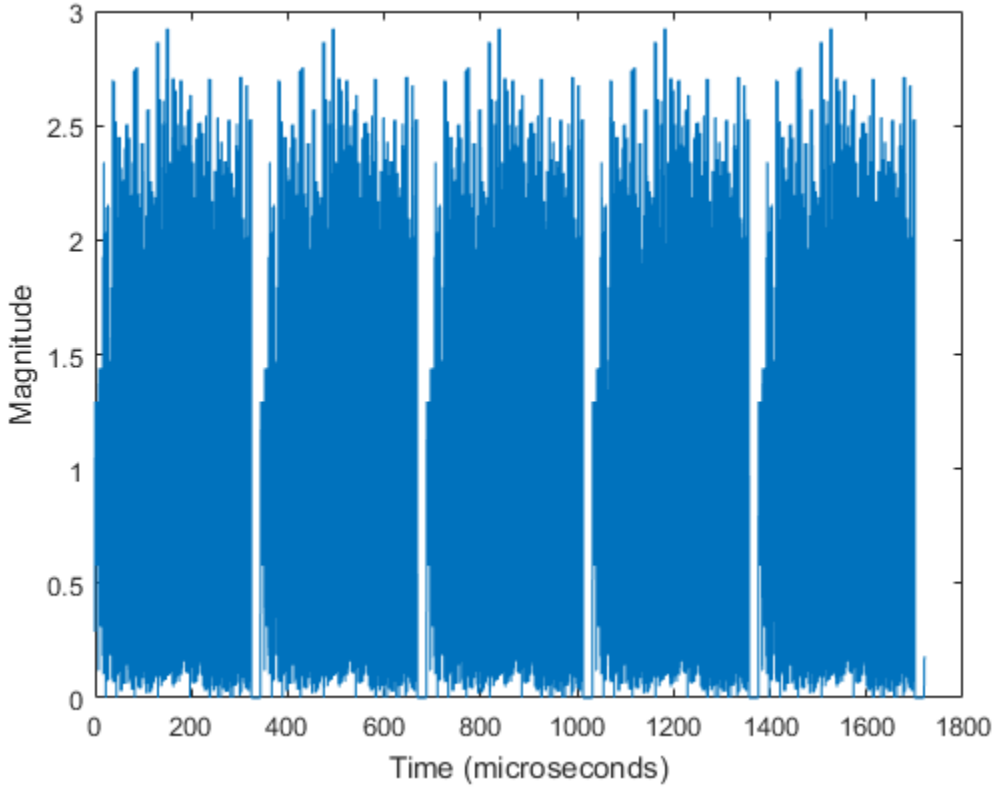
```
vht = wlanVHTConfig
```

```
bits = [1;0;0;1;1];
vhtWaveform = wlanWaveformGenerator(bits,vht, ...
    'NumPackets',5,'IdleTime',20e-6);
```

```
vht =

  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
```

Plot the VHT format waveform, scaling the *x-axis* relative to the channel bandwidth.

```
fs = 80e6; % Set sampling frequency equal to the channel bandwidth
time = ([0:length(vhtWaveform)-1]/fs)*1e6;
plot(time,abs(vhtWaveform))
xlabel ('Time (microseconds)');
ylabel('Magnitude');
```




The plot shows five VHT format packets, with each packet separated by 20 microseconds of idle time.

### Generate HT Format Waveform

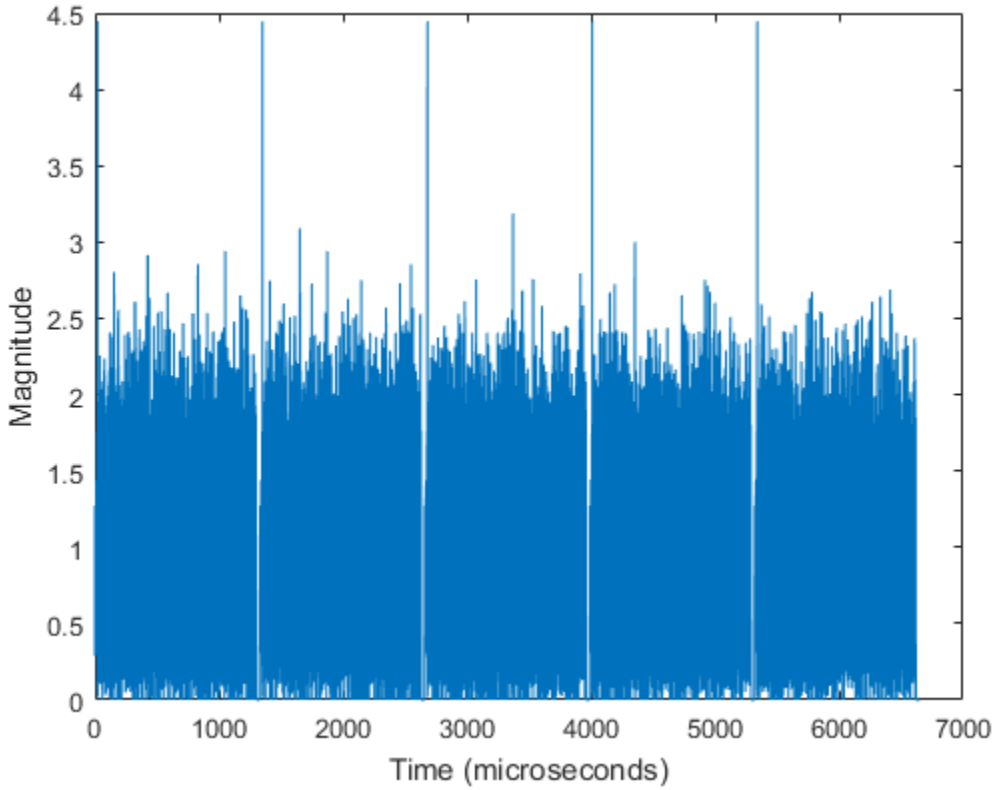
Create an HT configuration object and waveform. Using `Name,Value` pairs, specify 5 packets and 30 microseconds of idle time. Display the configuration object and inspect its properties and settings.

```
ht = wlanHTConfig
bits = [1;0;0;1;1];
htWaveform = wlanWaveformGenerator(bits,ht, ...
    'NumPackets',5,'IdleTime',30e-6);
```

```
ht =

  wlanHTConfig with properties:

      ChannelBandwidth: 'CBW20'
    NumTransmitAntennas: 1
    NumSpaceTimeStreams: 1
         SpatialMapping: 'Direct'
                    MCS: 0
          GuardInterval: 'Long'
          ChannelCoding: 'BCC'
             PSDULength: 1024
     RecommendSmoothing: 1
```

Plot the HT format waveform, scaling the *x-axis* relative to the channel bandwidth.

```
fs = 20e6; % Set sampling frequency equal to the channel bandwidth
time = ([0:length(htWaveform)-1]/fs)*1e6;
plot(time,abs(htWaveform))
xlabel ('Time (microseconds)');
ylabel('Magnitude');
```




The plot shows five HT format packets, with 30 microseconds of idle time separating each packet.

### Generate Non-HT Format DSSS Waveform

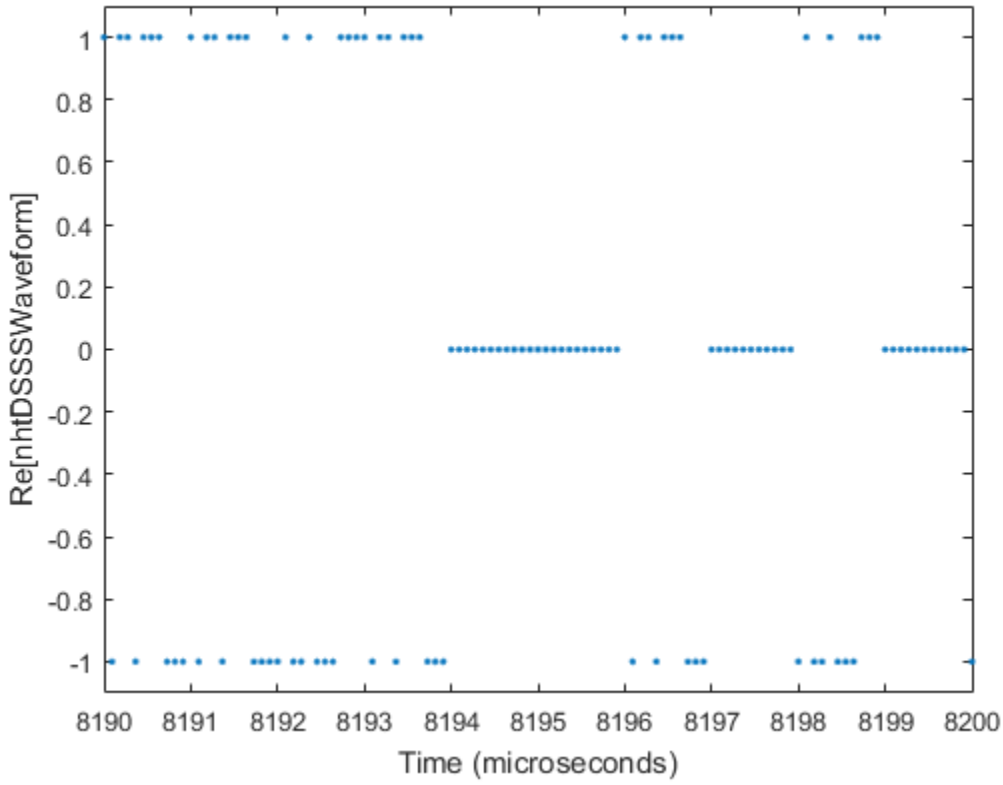
Create a non-HT configuration object and generate a non-HT format DSSS waveform with a 2 Mbps data rate. Using `Name,Value` pairs, specify 2 packets and 5 microseconds of idle time. Display the configuration object and inspect its properties and settings.

```
nht = wlanNonHTConfig('Modulation','DSSS','DataRate','2Mbps')
bits = [1;0;0;1;1];
nhtDSSSWaveform = wlanWaveformGenerator(bits,nht, ...
    'NumPackets',2,'IdleTime',5e-6);
```

```
nht =

  wlanNonHTConfig with properties:

      Modulation: 'DSSS'
        DataRate: '2Mbps'
        Preamble: 'Long'
    LockedClocks: 1
      PSDULength: 1000
```

Plot the non-HT Format DSSS waveform, scaling the *x-axis* relative to the channel bandwidth. As specified in IEEE 802.11-2012, Section 17.1.1, the channel bandwidth is 11 MHz for DSSS.

```
fs = 11e6; % Set sampling frequency equal to the channel bandwidth
time = ([0:length(nhtDSSSWaveform)-1]/fs)*1e6;
plot(time,real(nhtDSSSWaveform),'.')
xlabel ('Time (microseconds)');
ylabel('Re[nhtDSSSWaveform]');
axis([8190,8200,-1.1,1.1])
```




Sample values in DSSS modulation are –1 or 1. The plot shows the real values for a section of the waveform that includes the tail end of the first packet, the 5 microsecond idle period, and the beginning of the second packet for the non-HT format DSSS modulated waveform.

### Generate Non-HT Format OFDM Waveform

Create a non-HT configuration object and waveform. Using `Name,Value` pairs, specify 4 packets and 45 microseconds of idle time. Display the configuration object and inspect its properties and settings.

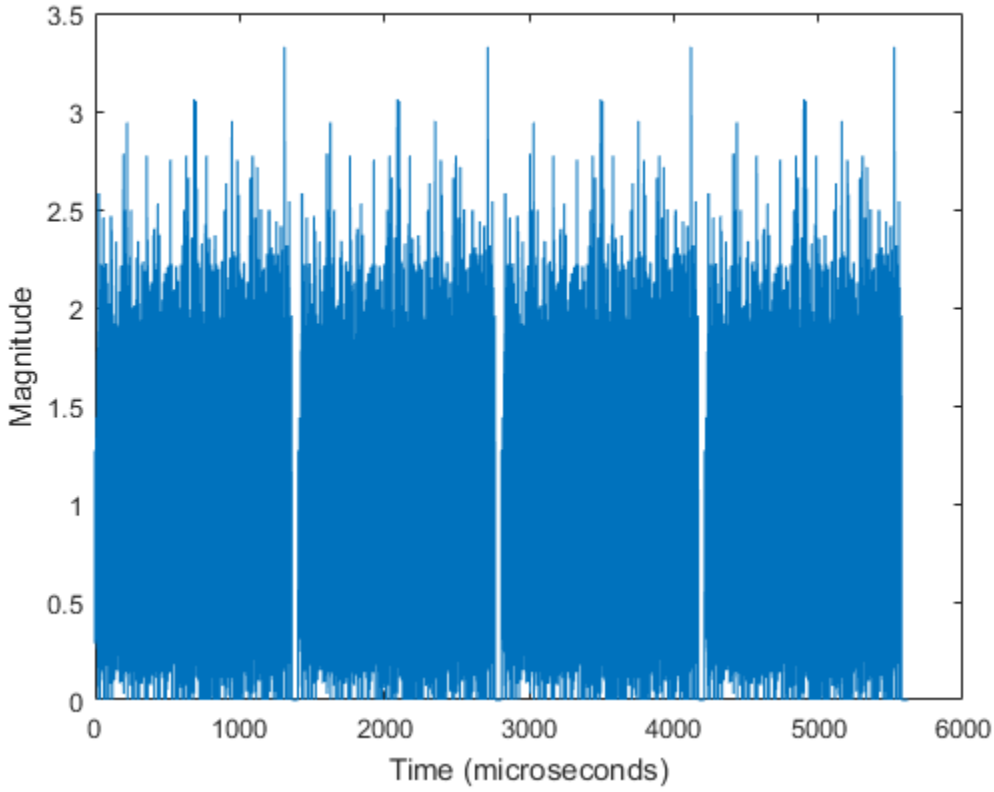
```
nht = wlanNonHTConfig
bits = [1;0;0;1;1];
nhtWaveform = wlanWaveformGenerator(bits,nht, ...
```

```
    'NumPackets',4,'IdleTime',45e-6);
```

```
nht =

  wlanNonHTConfig with properties:

            Modulation: 'OFDM'
      ChannelBandwidth: 'CBW20'
                   MCS: 0
            PSDULength: 1000
    NumTransmitAntennas: 1
```

Plot the non-HT format OFDM waveform, scaling the *x-axis* relative to the channel bandwidth.

```
fs = 20e6; % Set sampling frequency equal to the channel bandwidth
time = ([0:length(nhtWaveform)-1]/fs)*1e6;
plot(time,abs(nhtWaveform))
xlabel ('Time (microseconds)');
ylabel('Magnitude');
```




The plot shows four non-HT format OFDM modulated packets, with 45 microseconds of idle time separating each packet.

## Waveforms of Individual PPDU Fields

You can also create a VHT, HT, or non-HT PPDU waveform by generating and concatenating waveforms for individual PPDU fields.

| PPDU Format | Individual Field Functions |
|---|---|
| VHT | `wlanLSTF`, `wlanLLTF`, `wlanLSIG`, `wlanVHTSTF`, `wlanVHTLTF`, `wlanVHTSIGA`, `wlanVHTSIGB`, and `wlanVHTData` |

| PPDU Format | Individual Field Functions |
|---|---|
| HT | `wlanLSTF`, `wlanLLTF`, `wlanLSIG`, `wlanHTSTF`, `wlanHTLTF`, `wlanHTSIG`, and `wlanHTData` |
| Non-HT for OFDM modulation | `wlanLSTF`, `wlanLLTF`, `wlanLSIG`, and `wlanNonHTData` |

Generating individual PPDU field waveforms, enables you to experiment with the individual fields without generating an entire PPDU.

## See Also

wlanHTConfig Properties | wlanNonHTConfig Properties | wlanRecoveryConfig Properties | wlanVHTConfig Properties

## More About

- “Create Configuration Objects” on page 2-2
- “WLAN Channel Models” on page 2-24
- “What Is WLAN?” on page 3-2

# WLAN Channel Models

This example demonstrates passing WLAN VHT, HT, and non-HT format waveforms through appropriate fading channel models. When simulating a WLAN communications link, viable options for channel modeling include the TGn and TGac models from WLAN System Toolbox™ and the AWGN and 802.11g models from Communications System Toolbox™. In this example, it is sufficient to set the channel model sampling frequency to match the channel bandwidth because no front-end filtering is applied to the signal and the oversampling rate is 1.

In each section, you create a waveform and transmit it through a fading channel with noise added. Then you use a spectrum analyzer to display the waveform before and after it passes through the noisy fading channel.

### Pass VHT Waveform Through TGac SISO Channel

Create a bit stream to use when generating the WLAN VHT format waveform.

```
bits = randi([0 1],1000,1);
```

Create a VHT configuration object, and generate an 80 MHz VHT waveform. Calculate the signal power.

```
vht = wlanVHTConfig;
preChVHT = wlanWaveformGenerator(bits,vht);
```

Pass the signal through a TGac SISO channel with AWGN noise (SNR=10 dB) and a receiver with a 9 dB noise figure. Recall that the channel model sampling frequency is equal to the bandwidth in this example. Set parameters using `Name,Value` pairs.

Create a TGac channel object. Set the channel model sampling frequency and channel bandwidth, enable path loss and shadowing, and use the Model-D delay profile.

```
cbw = vht.ChannelBandwidth;
fs = 80e6; % Channel model sampling frequency equals the channel bandwidth
tgacChan = wlanTGacChannel('SampleRate',fs,'ChannelBandwidth',cbw, ...
    'LargeScaleFadingEffect','Pathloss and shadowing', ...
    'DelayProfile','Model-D');
```

Create an `AWGN Channel` object with SNR = 10 dB. Determine the signal power in Watts, accounting for the TGac large scale fading pathloss.

```
preChSigPwr_dB = 10*log10(mean(abs(preChVHT)));
sigPwr = 10^((preChSigPwr_dB-tgacChan.info.Pathloss)/10);

chNoise = comm.AWGNChannel('NoiseMethod','Signal to noise ratio (SNR)',...
    'SNR',10,'SignalPower', sigPwr);
```

Pass the VHT waveform through a 2x2 TGac channel and add the AWGN channel noise.

```
postChVHT = chNoise(tgacChan(preChVHT));
```

Create another `AWGN Channel` object to add receiver noise.

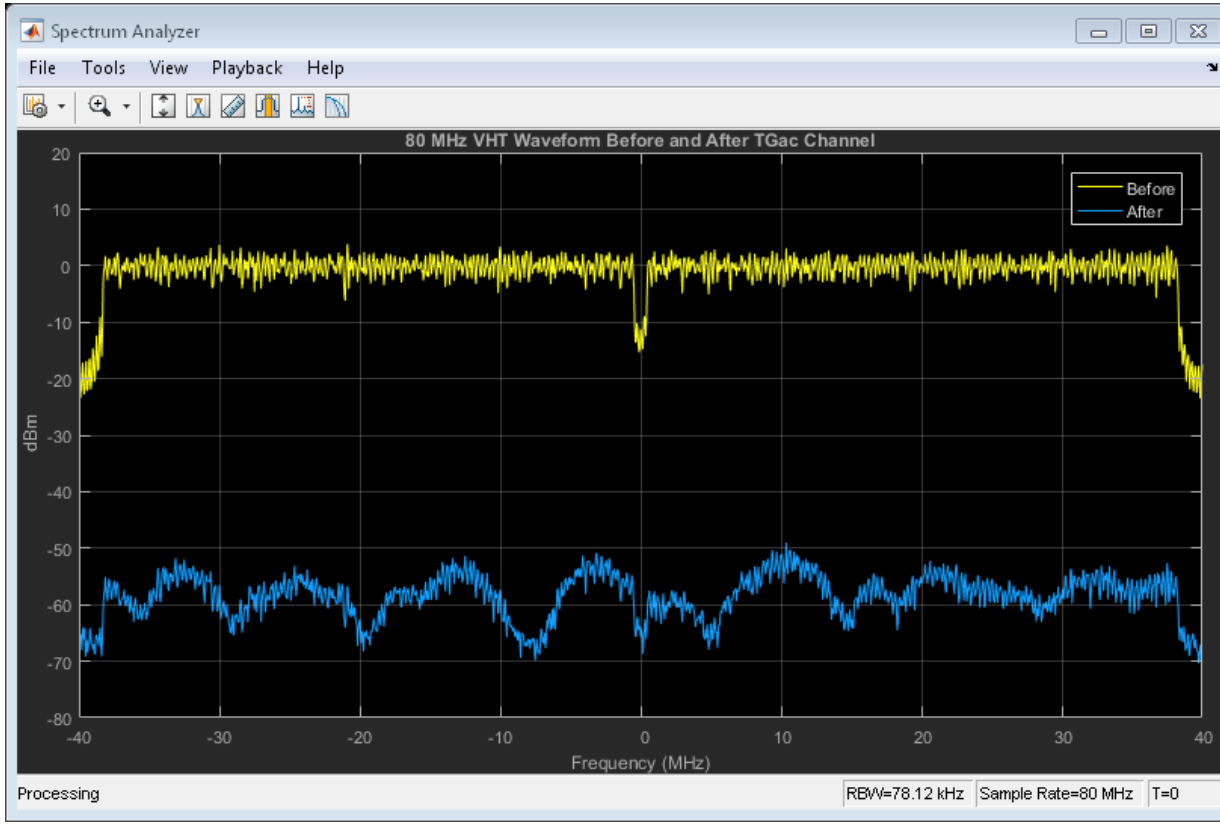
```
rxNoise = comm.AWGNChannel('NoiseMethod','Variance', ...
    'VarianceSource','Input port');
```

Pass the VHT waveform through the receiver. Choose an appropriate noise variance, nVar, to set the receiver noise level. Here, the receiver noise level is based on the noise variance for a receiver with a 9 dB noise figure. `nVar` = $kTBF$, where $k$ is Boltzmann's constant, $T$ is the ambient temperature of 290 K, $B$ is the bandwidth, and $F$ is the receiver noise figure.

```
nVar = 10^((-228.6 + 10*log10(290) + 10*log10(fs) + 9)/10);

rxVHT = rxNoise(postChVHT,nVar);
```

Display a spectrum analyzer with before-channel and after-channel waveforms. Use `SpectralAverages` = 10 to reduce noise in the plotted signals.

```
title = '80 MHz VHT Waveform Before and After TGac Channel';
saScope = dsp.SpectrumAnalyzer('SampleRate',fs,'ShowLegend',true,...
    'SpectralAverages',10,'Title',title,'ChannelNames',{'Before','After'});
saScope([preChVHT,rxVHT])
```




Path loss accounts for the roughly 50 to 60 dB of separation between the waveform before and after it passes through the TGac channel. The path loss results from the default transmitter-to-receiver distance of 3 meters, and from shadowing effects. The signal level variation shows the frequency selectivity of the delay profile across the frequency spectrum.

### Pass HT Waveform Through TGn SISO Channel

Create a bit stream to use when generating the WLAN HT format waveform.

```
bits = randi([0 1],1000,1);
```

Create an HT configuration object, and generate an HT waveform.

```
ht = wlanHTConfig;
```

```
preChHT = wlanWaveformGenerator(bits,ht);
```

Pass the signal through a TGn SISO channel with AWGN noise (SNR=10 dB) and a receiver with a 9 dB noise figure. Recall that the channel model sampling frequency is equal to the bandwidth in this example. Set parameters using `Name,Value` pairs.

Create a TGn channel object. Set the channel model sampling frequency and channel bandwidth, enable path loss and shadowing, and use the Model-F delay profile.

```
fs = 20e6; % Channel model sampling frequency equals the channel bandwidth
tgnChan = wlanTGnChannel('SampleRate',fs,'LargeScaleFadingEffect', ...
    'Pathloss and shadowing','DelayProfile','Model-F');
```

Pass the HT waveform through a TGn channel. Use the `awgn` function to add channel noise at an SNR level of 10 dB.

```
postChHT = awgn(tgnChan(preChHT),10,'measured');
```

Create an `AWGN Channel` object to add receiver noise.

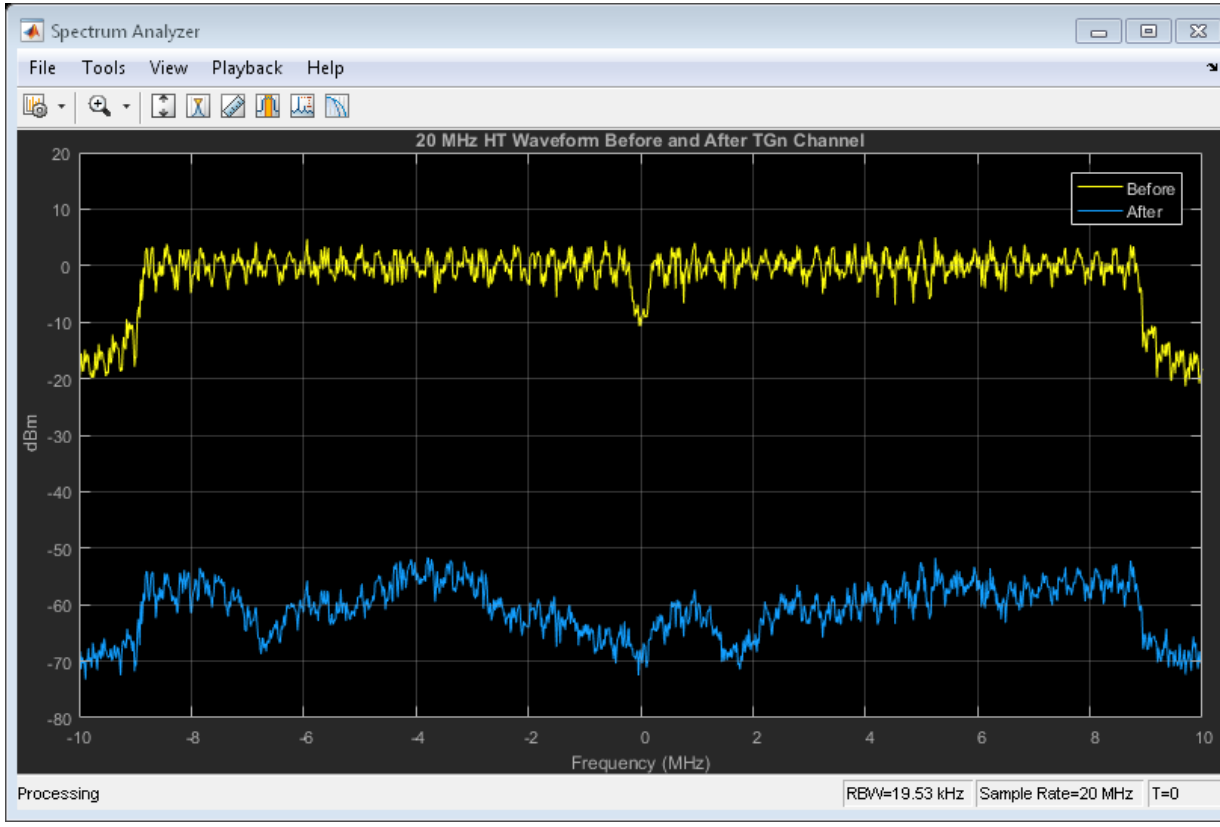
```
rxNoise = comm.AWGNChannel('NoiseMethod','Variance', ...
    'VarianceSource','Input port');
```

Pass the HT waveform through the receiver. Choose an appropriate noise variance, nVar, for setting the receiver noise level. Here, the receiver noise is based on the noise variance for a receiver with a 9 dB noise figure. `nVar` = $kTBF$, where $k$ is Boltzmann's constant, $T$ is the ambient temperature of 290 K, $B$ is the bandwidth, and $F$ is the receiver noise figure.

```
nVar = 10^((-228.6 + 10*log10(290) + 10*log10(fs) + 9)/10);

rxHT = rxNoise(postChHT, nVar);
```

Display a spectrum analyzer with before-channel and after-channel waveforms. Use `SpectralAverages` = 10 to reduce noise in the plotted signals.

```
title = '20 MHz HT Waveform Before and After TGn Channel';
saScope = dsp.SpectrumAnalyzer('SampleRate',fs,'ShowLegend',true,...
    'SpectralAverages',10,'Title',title,'ChannelNames',{'Before','After'});
saScope([preChHT,postChHT])
```




Path loss accounts for the roughly 50 to 60 dB of separation between the waveform before and after it passes through the TGn channel. The path loss results from the default transmitter-to-receiver distance of 3 meters, and from shadowing effects. The signal level variation shows the frequency selectivity of the delay profile across the frequency spectrum.

### Pass Non-HT Waveform Through 802.11g Channel

Create a bit stream to use when generating the WLAN Non-HT format waveform.

```
bits = randi([0 1],1000,1);
```

Create a non-HT configuration object, and generate a non-HT waveform.

```
nht = wlanNonHTConfig;
```

```
preChNonHT = wlanWaveformGenerator(bits,nht);
```

Calculate free-space path loss for a transmitter-to-receiver separation distance of 3 meters. Create an 802.11g channel object with a 3 Hz maximum Doppler shift and an RMS path delay equal to two times the sample time. Recall that the channel model sampling frequency is equal to the bandwidth in this example. Create an AWGN channel object.

```
dist = 3;
fc = 2.4e9;
pathLoss = 10^(-log10(4*pi*dist*(fc/3e8)));
fs = 20e6; % Channel model sampling frequency equals the channel bandwidth
trms = 2/fs;
ch802 = stdchan(1/fs,dist,'802.11g',trms);
```

Pass the non-HT waveform through an 802.11g channel. Use the `awgn` function to add channel noise at an SNR level of 10 dB.

```
postChNonHT = awgn(filter(ch802,preChNonHT),10,'measured');
```

Create an `AWGN Channel` object to add receiver noise.

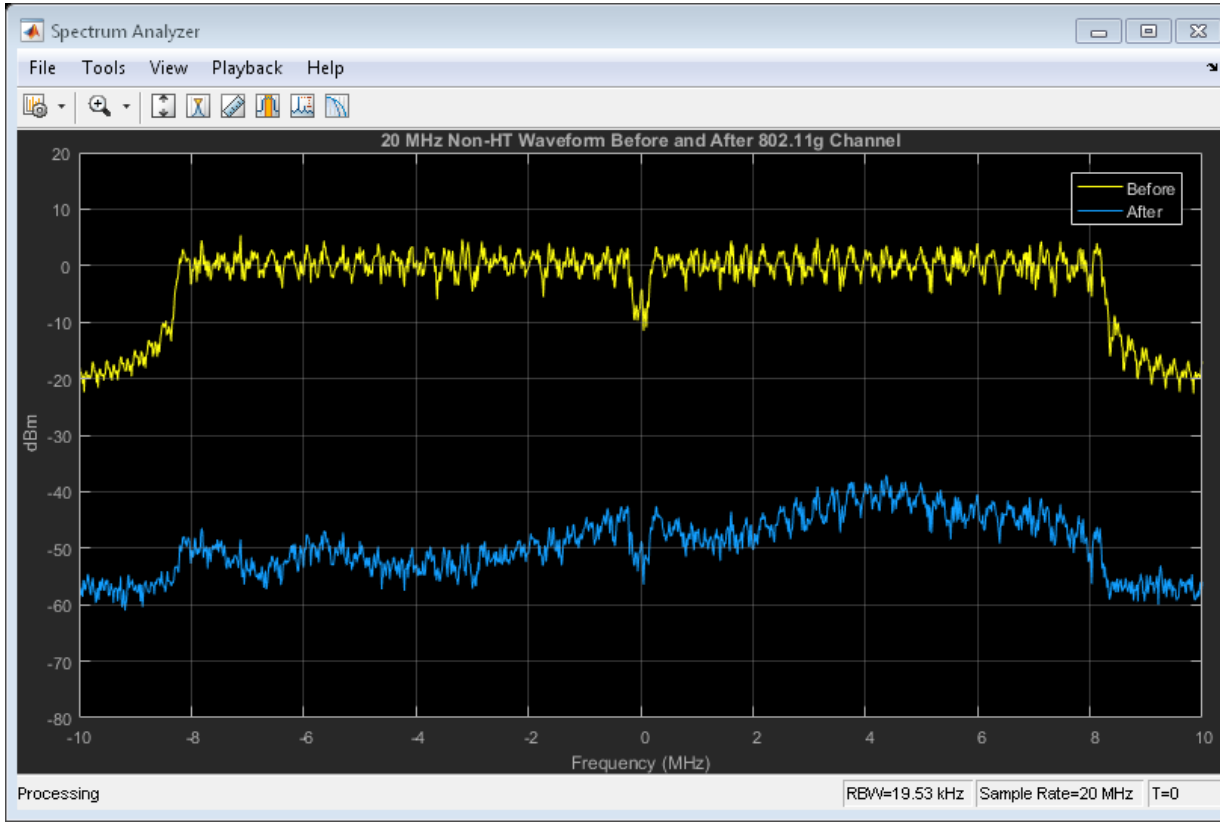
```
rxNoise = comm.AWGNChannel('NoiseMethod','Variance', ...
    'VarianceSource','Input port');
```

Pass the non-HT waveform through the receiver. Choose an appropriate noise variance, `nVar`, for setting the receiver noise level. Here, the receiver noise is based on the noise variance for a receiver with a 9 dB noise figure. `nVar` = $kTBF$, where $k$ is Boltzmann's constant, $T$ is the ambient temperature of 290 K, $B$ is the bandwidth, and $F$ is the receiver noise figure.

```
nVar = 10^((-228.6 + 10*log10(290) + 10*log10(fs) + 9)/10);

rxNonHT = rxNoise(postChNonHT, nVar)* pathLoss;
```

Display a spectrum analyzer with before-channel and after-channel waveforms. Use `SpectralAverages` = 10 to reduce noise in the plotted signals.

```
title = '20 MHz Non-HT Waveform Before and After 802.11g Channel';
saScope = dsp.SpectrumAnalyzer('SampleRate',fs,'ShowLegend',true,...
    'SpectralAverages',10,'Title',title,'ChannelNames',{'Before','After'});
saScope([preChNonHT,rxNonHT])
```




Free-space path loss accounts for the roughly 50 to 60 dB of separation between the waveform before and after it passes through the 802.11g channel. The path loss results from the specified transmitter-to-receiver distance of 3 meters, and from shadowing effects. The signal level variation shows the frequency selectivity of the delay profile across the frequency spectrum.

### Pass VHT Waveform Through TGac MIMO Channel

Create a bit stream to use when generating the WLAN VHT format waveform.

```
bits = randi([0 1],1000,1);
```

Create a multi-user VHT configuration object, and generate a VHT waveform. Set the number of transmit antennas to four. Set the number of space-time streams and the

number of receive antennas to 3. Because the number of transmit antennas is not equal to the number of space-time streams, the spatial mapping is not direct. Set the spatial mapping to Hadamard.

```
ntx = 4;
nsts = 3;
nrx = 3;
vht = wlanVHTConfig('NumTransmitAntennas',ntx, ...
    'NumSpaceTimeStreams',nsts,'SpatialMapping','Hadamard');
preChVHT = wlanWaveformGenerator(bits,vht);
```

Create TGac MIMO channel and AWGN channel objects. Recall that the channel model sampling frequency is equal to the bandwidth in this example. Disable large-scale fading effects.

```
cbw = vht.ChannelBandwidth;
fs = 80e6; % Channel model sampling frequency equals the channel bandwidth
tgacChan = wlanTGacChannel('SampleRate',fs,'ChannelBandwidth',cbw,...
    'NumTransmitAntennas',ntx,'NumReceiveAntennas',nrx);
tgacChan.LargeScaleFadingEffect = 'None';
```

Pass the VHT waveform through a TGac channel. Use the `awgn` function to add channel noise at an SNR level of 10 dB.

```
postChVHT = awgn(tgacChan(preChVHT),10,'measured');
```

Create an `AWGN Channel` object to add receiver noise.

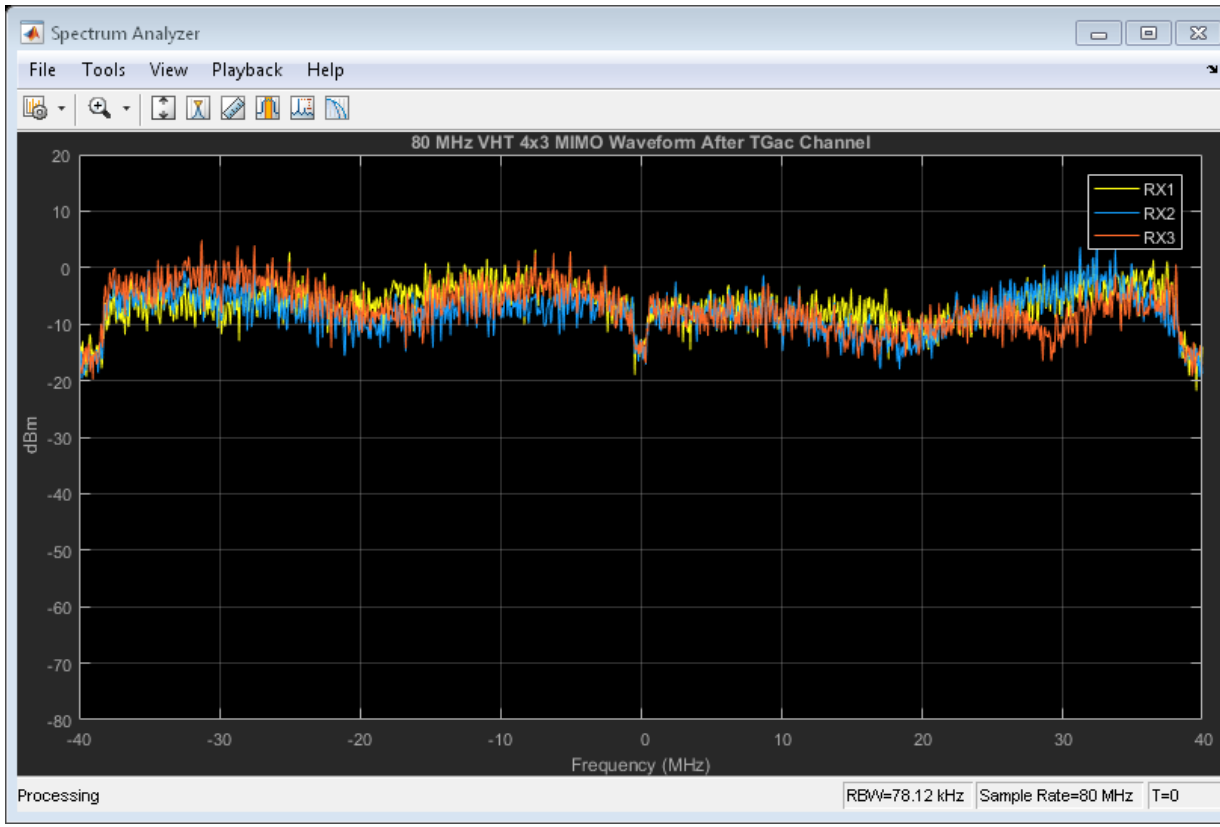
```
rxNoise = comm.AWGNChannel('NoiseMethod','Variance', ...
    'VarianceSource','Input port');
```

Pass the multi-user VHT waveform through a noisy TGac channel. Choose an appropriate noise variance, nVar, for setting the AWGN level. Here, the AWGN level is based on the noise variance for a receiver with a 9 dB noise figure. `nVar` = $kTBF$, where $k$ is Boltzmann's constant, $T$ is the ambient temperature of 290 K, $B$ is the bandwidth, and $F$ is the receiver noise figure.

```
nVar = 10^((-228.6 + 10*log10(290) + 10*log10(fs) + 9)/10);

rxVHT = rxNoise(postChVHT,nVar);
```

Display a spectrum analyzer showing the multiple streams after the channel effects have been added. Use `SpectralAverages` = 10 to reduce noise in the plotted signals.

```
title = '80 MHz VHT 4x3 MIMO Waveform After TGac Channel';
saScope = dsp.SpectrumAnalyzer('SampleRate',fs,'ShowLegend',true,...
    'SpectralAverages',10,'Title',title,'ChannelNames', ...
    {'RX1','RX2','RX3'});
saScope(rxVHT)
```




The overlaid signals show the TGac channel variation between the received streams.

## 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.

## See Also

wlanTGacChannel | wlanTGnChannel | `stdchan` | wlanHTConfig Properties | wlanNonHTConfig Properties | wlanRecoveryConfig Properties | wlanVHTConfig Properties

## Related Examples

- “Waveform Generation” on page 2-10
- “Packet Recovery” on page 2-34
- “What Is WLAN?” on page 3-2

# Packet Recovery

Received packets are degraded due to radio and channel impairments. Recovery of packet contents requires symbol timing and frequency offset correction, channel estimation, and demodulation and recovery of the preamble and payload. WLAN System Toolbox functions perform these operations on VHT, HT-mixed, and non-HT PPDU fields.

## VHT Packet Recovery

This example shows how to recover contents from a VHT format waveform.

### Generate 80 MHz VHT Waveform

Create a VHT configuration object. Set `APEPLength` to `3200` and `MCS` to `5`. Later these settings are compared to recovered signal information. Create a transmission bit stream for the data field. For a VHT waveform, the data field is `PSDULength*8` bits.

```
vht = wlanVHTConfig('APEPLength',3200,'MCS',5);
txBits = randi([0 1],vht.PSDULength*8,1);
```

Create the PPDU fields individually. Create L-STF, L-LTF, L-SIG, VHT-SIG-A, VHT-STF, VHT-LTF, and VHT-SIG-B preamble fields and the VHT-Data field.

```
lstf = wlanLSTF(vht);
lltf = wlanLLTF(vht);
lsig = wlanLSIG(vht);
vhtSigA = wlanVHTSIGA(vht);
vhtstf = wlanVHTSTF(vht);
vhtltf = wlanVHTLTF(vht);
vhtSigB = wlanVHTSIGB(vht);
vhtData = wlanVHTData(txBits,vht);
```

Concatenate the individual fields to create a single PPDU waveform.

```
txPPDU = [lstf; lltf; lsig; vhtSigA; vhtstf; vhtltf; vhtSigB; vhtData];
```

### Pass VHT Waveform Through TGac SISO Channel

Create TGac SISO and AWGN channel objects.

```
chBW = vht.ChannelBandwidth;
fs = 80e6;
tgacChan = wlanTGacChannel('SampleRate',fs,'ChannelBandwidth',chBW,...
    'LargeScaleFadingEffect','Pathloss and shadowing');
awgnChan = comm.AWGNChannel('NoiseMethod','Variance','VarianceSource','Input port');
```

Calculate the noise variance for a receiver with a 9 dB noise figure. The noise variance, `noiseVar`, 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. Pass the transmitted waveform through the noisy TGac channel.

```
noiseVar = 10^((-228.6 + 10*log10(290) + 10*log10(fs) + 9)/10)
rxPPDU = awgnChan(tgacChan(txPPDU),noiseVar);
```

```
noiseVar =

   2.5438e-12
```

### Recover VHT Preamble Contents from PPDU

In general, the L-STF and L-LTF are processed to perform frequency offset estimation and correction, and symbol timing. For this example, the carrier frequency is not offset and the packet timing is 'on-time'. Therefore, for accurate demodulation, determination of carrier frequency offset and symbol timing is not required.

Find the start and stop indices for the PPDU fields.

```
fieldInd = wlanFieldIndices(vht)
```

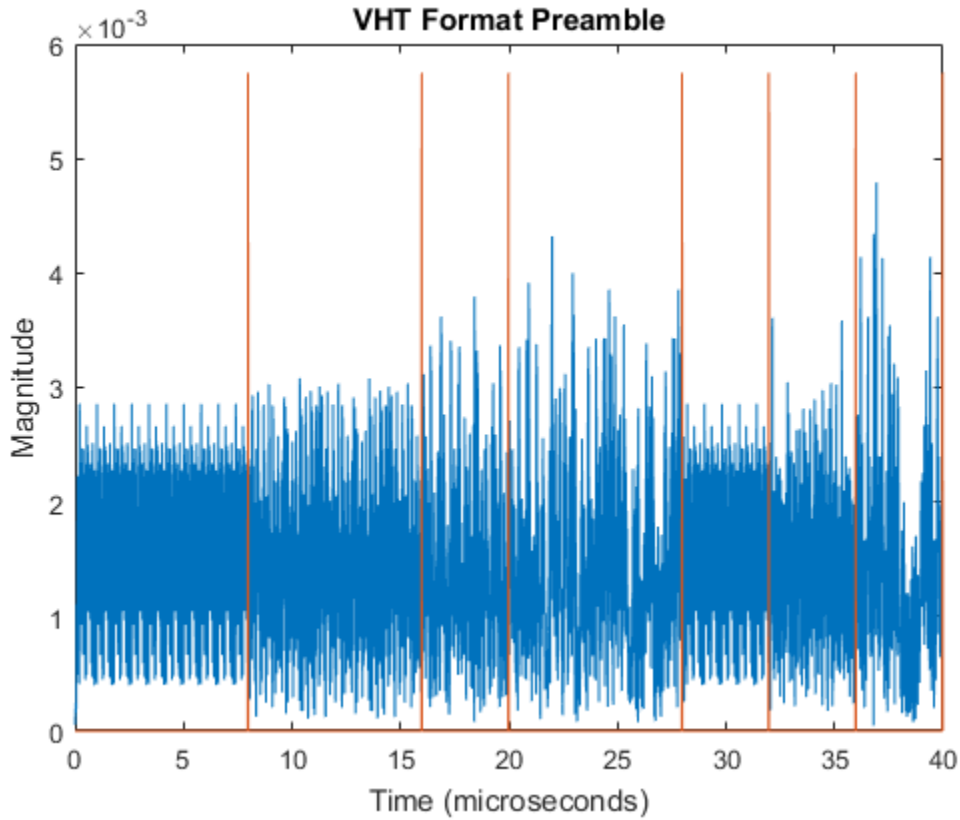
```
fieldInd =

  struct with fields:

       LSTF: [1 640]
       LLTF: [641 1280]
       LSIG: [1281 1600]
    VHTSIGA: [1601 2240]
     VHTSTF: [2241 2560]
     VHTLTF: [2561 2880]
    VHTSIGB: [2881 3200]
    VHTData: [3201 12160]
```

The stop index of VHT-SIG-B indicates the preamble length in samples.

```
numSamples = fieldInd.VHTSIGB(2);
```

Plot the preamble and the beginning of the packet data. Add markers to and plot to delineate the packet field boundaries.

```
time = ([0:double(numSamples)-1]/fs)*1e6;
peak = 1.2*max(abs(rxPPDU(1:numSamples)));
fieldMarkers = zeros(numSamples,1);
fieldMarkers(fieldInd.LSTF(2)-1,1) = peak;
fieldMarkers(fieldInd.LLTF(2)-1,1) = peak;
fieldMarkers(fieldInd.LSIG(2)-1,1) = peak;
fieldMarkers(fieldInd.VHTSIGA(2)-1,1) = peak;
fieldMarkers(fieldInd.VHTSTF(2)-1,1) = peak;
fieldMarkers(fieldInd.VHTLTF(2)-1,1) = peak;
fieldMarkers(fieldInd.VHTSIGB(2)-1,1) = peak;
plot(time,abs(rxPPDU(1:numSamples)),time,fieldMarkers)
xlabel ('Time (microseconds)')
ylabel('Magnitude')
title('VHT Format Preamble')
```



Demodulate the L-LTF and estimate the channel.

```
rxLLTF = rxPPDU(fieldInd.LLTF(1):fieldInd.LLTF(2),:);
demodLLTF = wlanLLTFDemodulate(rxLLTF,vht);
chEstLLTF = wlanLLTFChannelEstimate(demodLLTF,vht);
```

Extract the L-SIG field from the received PPDU, recover its information bits and check the CRC.

```
rxLSIG = rxPPDU(fieldInd.LSIG(1):fieldInd.LSIG(2),:);
[recLSIG,failCRC] = wlanLSIGRecover(rxLSIG,chEstLLTF,noiseVar,chBW);
failCRC
```

```
failCRC =

  logical

   0
```

`failCRC = 0` indicates that CRC passed.

For the VHT format, the L-SIG rate bits are constant and set to `[1 1 0 1]`. Inspect the L-SIG rate information and confirm that this constant sequence is recovered. For the VHT format, the MCS setting in VHT-SIG-A2 determines the actual data rate.

```
rate = recLSIG(1:4)'
```

```
rate =

  1×4 int8 row vector

   1   1   0   1
```

Extract the VHT-SIG-A and confirm that the CRC check passed.

```
rxVHTSIGA = rxPPDU(fieldInd.VHTSIGA(1):fieldInd.VHTSIGA(2),:);
[recVHTSIGA,failCRC] = wlanVHTSIGARecover(rxVHTSIGA, ...
    chEstLLTF,noiseVar,chBW);
failCRC
```

```
failCRC =

  logical

   0
```

Extract the MCS setting from the VHT-SIG-A. For single user VHT, the MCS is located in VHT-SIG-A2 bits 4 through 7.

```
recMCSbits = (recVHTSIGA(29:32))';
recMCS = bi2de(double(recMCSbits))
isequal(recMCS,vht.MCS)
```

```
recMCS =

     5


ans =

  logical

   1
```

The recovered MCS setting matches the MCS value in the configuration object.

Extract and demodulate the VHT-LTF. Use the demodulated signal to perform channel estimation. Use the channel estimate to recover the VHT-SIG-B and VHT-Data fields.

```
rxVHTLTF = rxPPDU(fieldInd.VHTLTF(1):fieldInd.VHTLTF(2),:);
demodVHTLTF = wlanVHTLTFDemodulate(rxVHTLTF,vht);
chEstVHTLTF = wlanVHTLTFChannelEstimate(demodVHTLTF,vht);
```

Extract and recover the VHT-SIG-B.

```
rxVHTSIGB = rxPPDU(fieldInd.VHTSIGB(1):fieldInd.VHTSIGB(2),:);
recVHTSIGB = wlanVHTSIGBRecover(rxVHTSIGB,chEstVHTLTF,noiseVar,chBW);
```

As described in IEEE Std 802.11ac-2013, Table 22-1, the value in the VHT-SIG-B Length field multiplied by 4 is the recovered APEP length for packets carrying data. Verify that the APEP length, contained in the first 19 bits of the VHT-SIG-B, corresponds to the specified APEP length.

```
sigbAPEPbits = recVHTSIGB(1:19)';
sigbAPEPlength = bi2de(double(sigbAPEPbits))*4
isequal(sigbAPEPlength,vht.APEPLength)
```

```
sigbAPEPlength =

        3200


ans =

  logical

   1
```

The recovered value matches the configured APEP Length.

#### Recover VHT-Data Contents from PPDU

Construct a recovery configuration object.

```
cfgRec = wlanRecoveryConfig;
```

Recover receive equalized symbols using channel estimates from VHT-LTF.

```
recPSDU = wlanVHTDataRecover(rxPPDU(fieldInd.VHTData(1):fieldInd.VHTData(2),:),...
    chEstVHTLTF,noiseVar,vht,cfgRec);
```

Compare transmission and receive PSDU bits.

```
numErr = biterr(txBits,recPSDU)
```

```
numErr =

     0
```

The number of bit errors is zero.

## HT Packet Recovery

This example shows how to recover content from a HT format waveform.

### Generate 20 MHz HT Waveform

Create an HT configuration object and transmission PSDU. Set `MCS` to `2`. Later these settings are compared to recovered signal information. For an HT waveform, the data field is `PSDULength*8` bits.

```
ht = wlanHTConfig('MCS',2);
txPSDU = randi([0 1],ht.PSDULength*8,1);
```

Create the PPDU fields individually. Create L-STF, L-LTF, L-SIG, HT-SIG, HT-STF, and HT-LTF preamble fields and the HT-Data field.

```
lstf = wlanLSTF(ht);
lltf = wlanLLTF(ht);
lsig = wlanLSIG(ht);
htsig = wlanHTSIG(ht);
htstf = wlanHTSTF(ht);
htltf = wlanHTLTF(ht);
htData = wlanHTData(txPSDU,ht);
```

Concatenate the individual fields to create a single PPDU waveform.

```
txPPDU = [lstf; lltf; lsig; htsig; htstf; htltf; htData];
```

### Pass HT Waveform Through TGn SISO Channel

Create TGn SISO channel and AWGN channel objects.

```
fs = 20e6;
tgnChan = wlanTGnChannel('SampleRate',fs,'LargeScaleFadingEffect','Pathloss and shadow
awgnChan = comm.AWGNChannel('NoiseMethod','Variance','VarianceSource','Input port');
```

Calculate the noise variance for a receiver with a 9 dB noise figure. The noise variance, `noiseVar`, 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. Pass the transmitted waveform through the noisy TGn channel.

```
noiseVar = 10^((-228.6 + 10*log10(290) + 10*log10(fs) + 9)/10);
rxPPDU = awgnChan(tgnChan(txPPDU),noiseVar);
```

### Recover HT Preamble Contents from PPDU

In general, the L-STF and L-LTF are processed to perform frequency offset estimation and correction, and symbol timing. For this example, the carrier frequency is not offset

and the packet timing is 'on-time'. Therefore, for accurate demodulation, determination of carrier frequency offset and symbol timing is not required.

Find the start and stop indices for the PPDU fields.

```
fieldInd = wlanFieldIndices(ht)
```

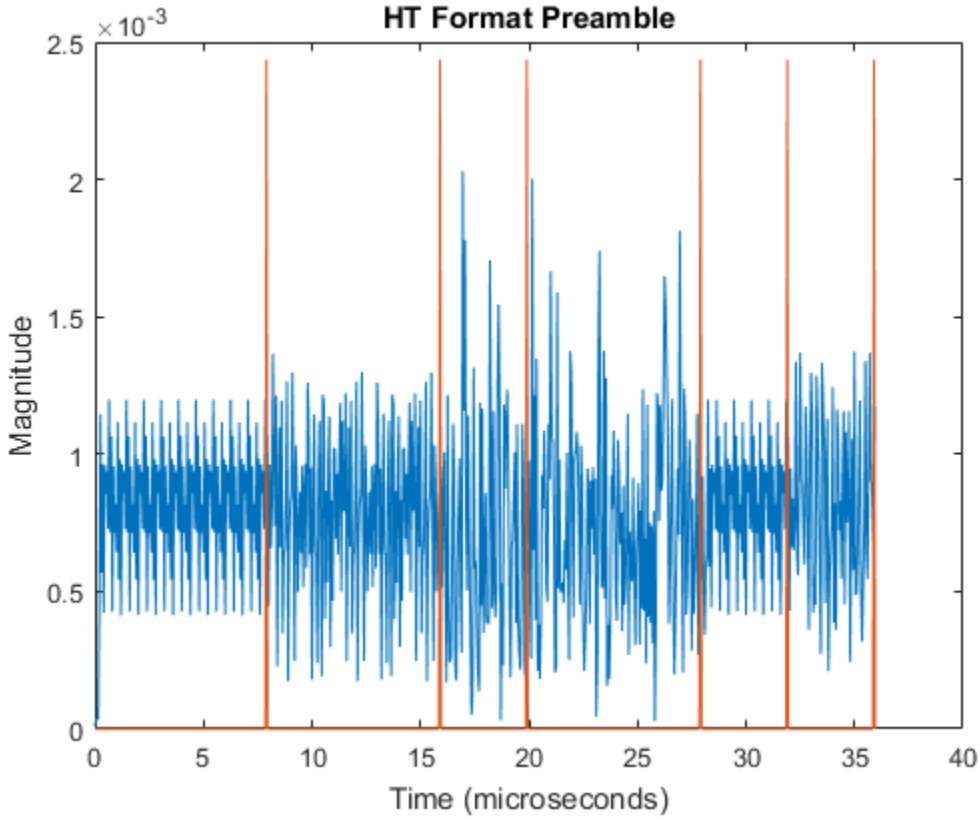
```
fieldInd =

  struct with fields:

      LSTF: [1 160]
      LLTF: [161 320]
      LSIG: [321 400]
     HTSIG: [401 560]
     HTSTF: [561 640]
     HTLTF: [641 720]
    HTData: [721 9200]
```

The stop index of HT-LTF indicates the preamble length in samples.

```
numSamples = fieldInd.HTLTF(2);
```

Plot the preamble and the beginning of the packet data. Add markers to and plot to delineate the packet field boundaries.

```
time = ([0:double(numSamples)-1]/fs)*1e6;
peak = 1.2*max(abs(rxPPDU(1:numSamples)));
fieldMarkers = zeros(numSamples,1);
fieldMarkers(fieldInd.LSTF(2)-1,1) = peak;
fieldMarkers(fieldInd.LLTF(2)-1,1) = peak;
fieldMarkers(fieldInd.LSIG(2)-1,1) = peak;
fieldMarkers(fieldInd.HTSIG(2)-1,1) = peak;
fieldMarkers(fieldInd.HTSTF(2)-1,1) = peak;
fieldMarkers(fieldInd.HTLTF(2)-1,1) = peak;
plot(time,abs(rxPPDU(1:numSamples)),time,fieldMarkers)
xlabel ('Time (microseconds)')
ylabel('Magnitude')
title('HT Format Preamble')
```




Demodulate the L-LTF and estimate the channel.

```
rxLLTF = rxPPDU(fieldInd.LLTF(1):fieldInd.LLTF(2),:);
demodLLTF = wlanLLTFDemodulate(rxLLTF,ht);
chEstLLTF = wlanLLTFChannelEstimate(demodLLTF,ht);
```

Extract the L-SIG field from the received PPDU and recover its information bits.

```
rxLSIG = rxPPDU(fieldInd.LSIG(1):fieldInd.LSIG(2),:);
[recLSIG,failCRC] = wlanLSIGRecover(rxLSIG,chEstLLTF,noiseVar,ht.ChannelBandwidth);
failCRC
```

```
failCRC =
```

```
  logical

   0
```

`failCRC = 0` indicates that CRC passed.

For the HT format, the L-SIG rate bits are constant and set to `[1 1 0 1]`. Inspect the L-SIG rate information and confirm that this constant sequence is recovered. For the HT format, the MCS setting in HT-SIG determines the actual data rate.

```
rate = recLSIG(1:4)'
```

```
rate =

  1×4 int8 row vector

   1   1   0   1
```

Extract the HT-SIG and confirm that the CRC check passed.

```
recHTSIG = rxPPDU(fieldInd.HTSIG(1):fieldInd.HTSIG(2),:);
[recHTSIG,failCRC] = wlanHTSIGRecover(recHTSIG,chEstLLTF,noiseVar,ht.ChannelBandwidth);
failCRC
```

```
failCRC =

  logical

   0
```

Extract the MCS setting from the HT-SIG. For HT, the MCS is located in HT-SIG bits 0 through 6.

```
recMCSbits = (recHTSIG(1:7))';
recMCS = bi2de(double(recMCSbits))
isequal(recMCS,ht.MCS)
```

```
recMCS =
```

```
     2


ans =

  logical

   1
```

The recovered MCS setting matches the MCS value in the configuration object.

Extract and demodulate the HT-LTF. Use the demodulated signal to perform channel estimation. Use the channel estimate to recover the HT-Data field.

```
rxHTLTF = rxPPDU(fieldInd.HTLTF(1):fieldInd.HTLTF(2),:);
demodHTLTF = wlanHTLTFDemodulate(rxHTLTF,ht);
chEstHTLTF = wlanHTLTFChannelEstimate(demodHTLTF,ht);
```

#### Recover HT-Data Contents from PPDU

Create a recovery configuration object.

```
cfgRec = wlanRecoveryConfig;
```

Recover the received equalized symbols using channel estimates from the HT-LTF.

```
[recPSDU] = wlanHTDataRecover(rxPPDU(fieldInd.HTData(1):fieldInd.HTData(2),:),...
    chEstHTLTF,noiseVar,ht,cfgRec);
```

Compare the transmitted and received PSDU bits, and confirm that the number of bit errors is zero.

```
numErr = biterr(txPSDU,recPSDU)


numErr =

     0
```

## Non-HT Packet Recovery

This example steps through recovery of non-HT format waveform content.

### Generate 20 MHz Non-HT Waveform

Create a non-HT configuration object and transmission PSDU. Set `MCS` to `4`. Later these settings are compared to recovered signal information. For a non-HT waveform, the data field is `PSDULength*8` bits.

```
nht = wlanNonHTConfig('MCS',4);
txPSDU = randi([0 1],nht.PSDULength*8,1);
```

Create the PPDU fields individually. Use the non-HT-Data contents to check the bit error rate after recovery. Create L-STF, L-LTF, and L-SIG preamble fields and non-HT data field.

```
lstf = wlanLSTF(nht);
lltf = wlanLLTF(nht);
lsig = wlanLSIG(nht);
nhtData = wlanNonHTData(txPSDU,nht);
```

Concatenate the individual fields to create a single PPDU waveform.

```
txPPDU = [lstf; lltf; lsig; nhtData];
```

### Pass Non-HT Waveform Through 802.11g SISO Channel

Calculate the free-space path loss for a transmitter-to-receiver separation distance of 3 meters. Create an 802.11g channel with a 3 Hz maximum Doppler shift and an RMS path delay equal to two times the sample time. Create an AWGN channel.

```
dist = 3;
pathLoss = 10^(-log10(4*pi*dist*(2.4e9/3e8)));
fs = 20e6;
trms = 2/fs;
ch802 = stdchan(1/fs,3,'802.11g',trms);
awgnChan = comm.AWGNChannel('NoiseMethod','Variance','VarianceSource','Input port');
```

Calculate the noise variance for a receiver with a 9 dB noise figure. The noise variance, `noiseVar`, 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. Pass the transmitted waveform through the noisy, lossy 802.11g channel.

```
noiseVar = 10^((-228.6 + 10*log10(290) + 10*log10(fs) + 9)/10);
rxPPDU = awgnChan(filter(ch802,txPPDU),noiseVar) * pathLoss;
```

### Recover Non-HT Preamble Contents from PPDU

In general, the L-STF and L-LTF are processed to perform frequency offset estimation and correction, and symbol timing. For this example, the carrier frequency is not offset and the packet timing is 'on-time'. Therefore, for accurate demodulation, determination of carrier frequency offset and symbol timing is not required.

Find the start and stop indices for the PPDU fields.

```
fieldInd = wlanFieldIndices(nht)
```

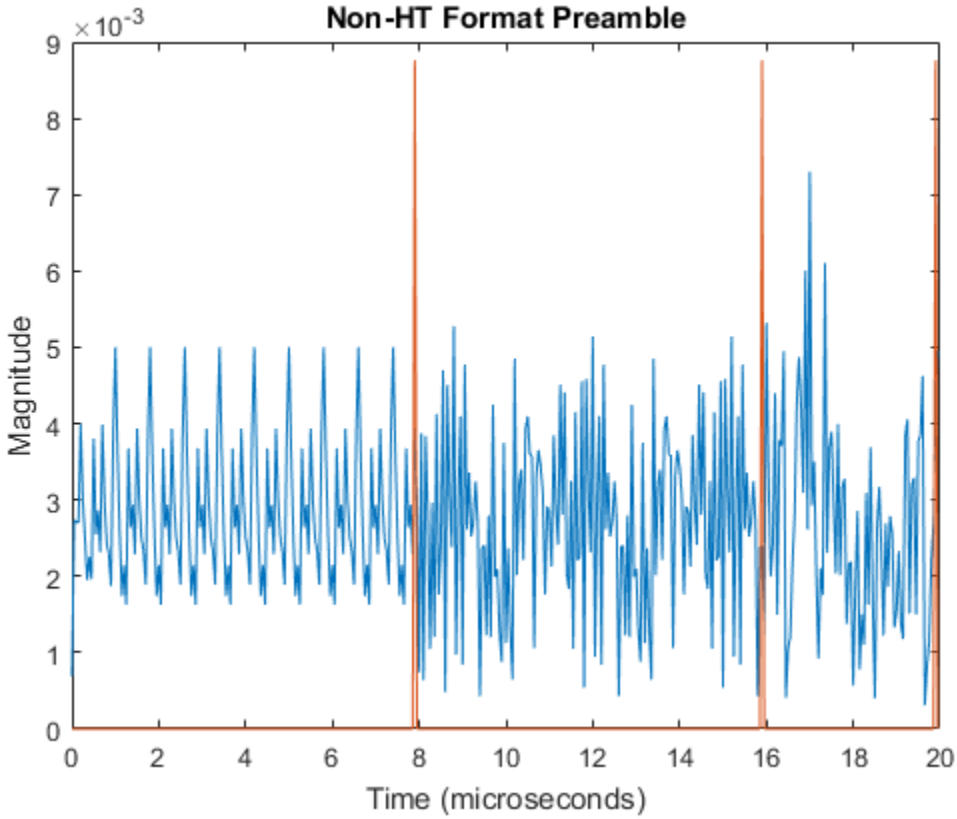
```
fieldInd =

  struct with fields:

        LSTF: [1 160]
        LLTF: [161 320]
        LSIG: [321 400]
    NonHTData: [401 7120]
```

The stop index of the L-SIG field indicates the preamble length in samples.

```
numSamples = fieldInd.LSIG(2);
```

Plot the preamble and the beginning of the packet data. Add markers to and plot to delineate the packet field boundaries.

```
time = ([0:double(numSamples)-1]/fs)*1e6;
peak = 1.2*max(abs(rxPPDU(1:numSamples)));
fieldMarkers = zeros(numSamples,1);
fieldMarkers(fieldInd.LSTF(2)-1,1)  = peak;
fieldMarkers(fieldInd.LLTF(2)-1,1) = peak;
fieldMarkers(fieldInd.LSIG(2)-1,1) = peak;
plot(time,abs(rxPPDU(1:numSamples)),time,fieldMarkers)
xlabel ('Time (microseconds)')
ylabel('Magnitude')
title('Non-HT Format Preamble')
```




Demodulate the L-LTF and estimate the channel.

```
rxLLTF = rxPPDU(fieldInd.LLTF(1):fieldInd.LLTF(2),:);
demodLLTF = wlanLLTFDemodulate(rxLLTF,nht);
chEstLLTF = wlanLLTFChannelEstimate(demodLLTF,nht);
```

Extract the L-SIG field from the received PPDU and recover its information bits.

```
rxLSIG = rxPPDU(fieldInd.LSIG(1):fieldInd.LSIG(2),:);
recLSIG = wlanLSIGRecover(rxLSIG,chEstLLTF,noiseVar,'CBW20');
```

The first four bits of the L-SIG field, bits 0 through 3, contain the rate information. Confirm that the sequence `[1 0 0 1]` is recovered. This sequence corresponds to the 24 MHz data rate for the non-HT MCS setting of 4.

```
rate = recLSIG(1:4)'
```

```
rate =

  1×4 int8 row vector

   1   0   0   1
```

Extract and demodulate the L-LTF. Use the demodulated signal to perform channel estimation. Use the channel estimate to recover the non-HT-Data field.

```
rxLLTF = rxPPDU(fieldInd.LLTF(1):fieldInd.LLTF(2),:);
demodLLTF = wlanLLTFDemodulate(rxLLTF,nht);
chEstLLTF = wlanLLTFChannelEstimate(demodLLTF,nht);
```

#### Recover Non-HT-Data Contents from PPDU

Create a recovery configuration object, with its equalization method set to zero forcing.

```
cfgRec = wlanRecoveryConfig('EqualizationMethod','ZF');
```

Recover equalized symbols using channel estimates from HT-LTF.

```
rxPSDU = rxPPDU(fieldInd.NonHTData(1):fieldInd.NonHTData(2),:);
[recPSDU,~,eqSym] = wlanNonHTDataRecover(rxPSDU,chEstLLTF,noiseVar,nht,cfgRec);
```

Compare the transmitted and received PSDU bits, and confirm that the number of bit errors is zero.

```
numErr = biterr(txPSDU,recPSDU)
```

```
numErr =

     0
```

## See Also

wlanHTConfig Properties | wlanNonHTConfig Properties | wlanRecoveryConfig Properties | wlanVHTConfig Properties

## Related Examples

- “WLAN Channel Models” on page 2-24

- “What Is WLAN?” on page 3-2
- “Build VHT PPDU”
- “Build HT PPDU”
- “Build Non-HT PPDU”

3

# About WLAN

# What Is WLAN?

In general, a wireless local area network (WLAN) refers to a wireless computer network. More commonly, WLAN is equated with the implementation specified by the IEEE 802.11[34] group of standards and branded as Wi-Fi® by the Wi-Fi Alliance. The Wi-Fi Alliance certifies interoperability between IEEE 802.11 devices from different manufacturers. WLAN System Toolbox functionality enables you to model IEEE 802.11 standardized implementations of the WLAN physical layer (PHY). It also enables you to explore variations on implementations for future evolution of the standard.

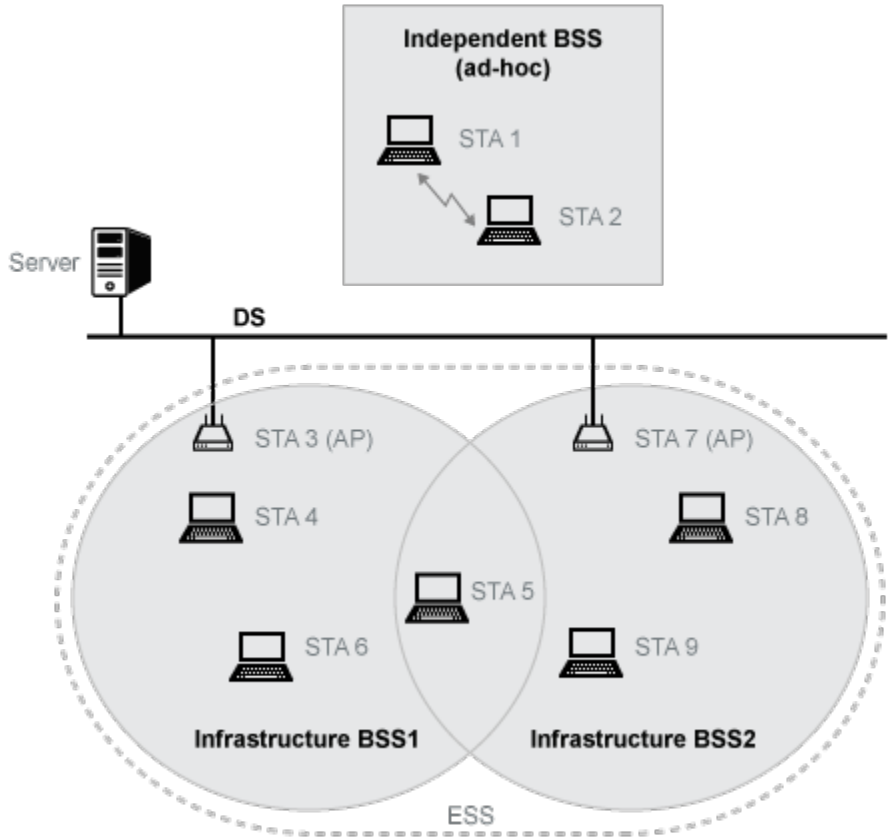
| In this section... |
| --- |
|  |
|  |
|  |
|  |

## Network Architecture

IEEE 802.11 defines the network architectures. In IEEE 802.11, a group of stations (STAs) within a defined coverage area and with appropriate association to each other form a basic service set (BSS). The BSS is a basic building block for 802.11 network architecture. A basic service area (BSA) defines an area containing STAs within a BSS. STAs can be associated in overlapping BSSs. In terms of mobility, STAs are either fixed, portable, or mobile. Any compliant STA can serve as an access point (AP).

The figure depicts WLAN components and network architectures built up from BSSs.

3. IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.
4. IEEE Std 802.11ac-2013 Adapted and reprinted with permission from IEEE. Copyright IEEE 2013. All rights reserved.



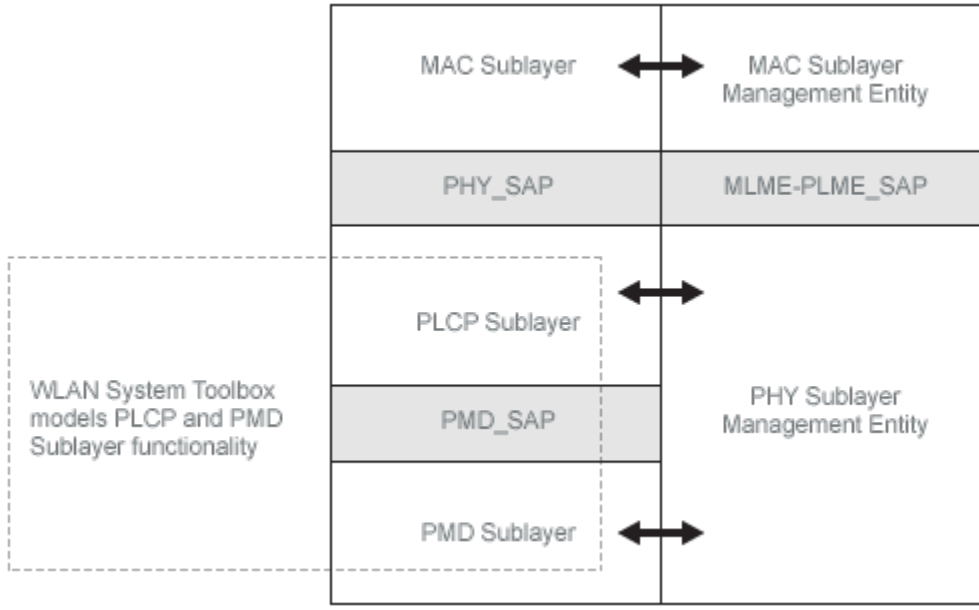

- Independent BSS (IBSS) describes STAs communicating directly with one another in an ad-hoc fashion. An IBSS has no connection to the wired network.
- Infrastructure BSS describes STAs associated with a central STA that manages the BSS. The central STA is referred to as an access point (AP). This deployment is commonly used in home, office, and hotspot network installations. Generally speaking, the AP connects wirelessly with associated STAs and is wired to the Internet. This connection enables associated STAs to communicate beyond the local BSS. The APs also wirelessly serve STAs in a BSA, providing internet connectivity for those STAs.
- Distributed systems (DS) interconnect infrastructure BSSs via their APs. Typically the DS backbone is an 802.3 Ethernet LAN.

- Extended service set (ESS) describes a set of infrastructure BSSs interconnected by a DS. In an ESS, APs communicate among themselves to forward traffic from one BSS to another and to facilitate the movement of mobile station from one BSS to another.

## WLAN Protocol Stack

The interworking reference model shown here includes a subset of the network components associated with the data link layer (DLL) and physical layer (PHY). IEEE Std 802.11-2012 [2], Section 4.9.2 describes the interworking reference model for 802.11. the medium access control (MAC) is a sublayer of the DLL.

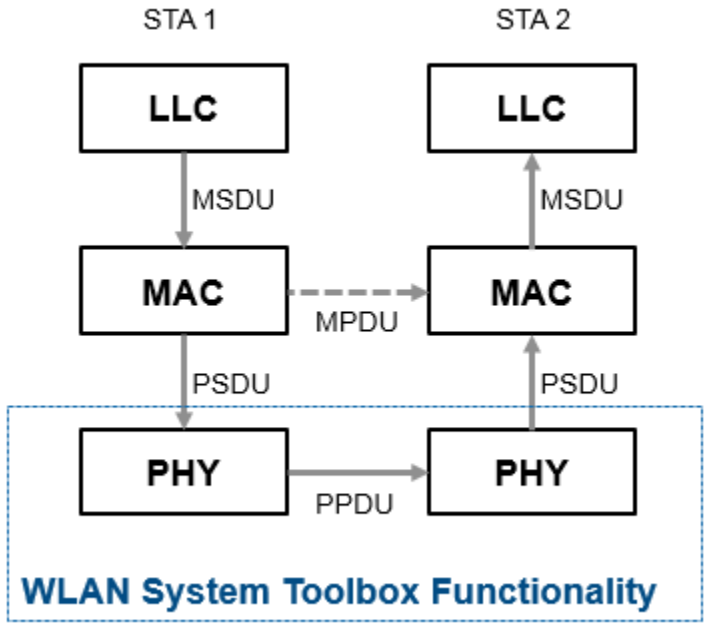
The 802.11 standards focus on the MAC and PHY as a whole. The WLAN System Toolbox PHY modeling functionality focuses on the physical medium dependent (PMD) and the physical layer convergence procedure (PLCP) sublayers of the PHY, and their interfaces.




## WLAN Protocol Layers

Data and control information messages are exchanged between layers of the protocol stack within an individual STA and between peer layers in communicating STAs.

- Data and control information exchanged between peer STA layers are *protocol information transfers*. See MPDU and PPDU in the figure.
- Data and control information exchanged between layers within an STA are *service information transfers*. See MSDU and PSDU in the figure.

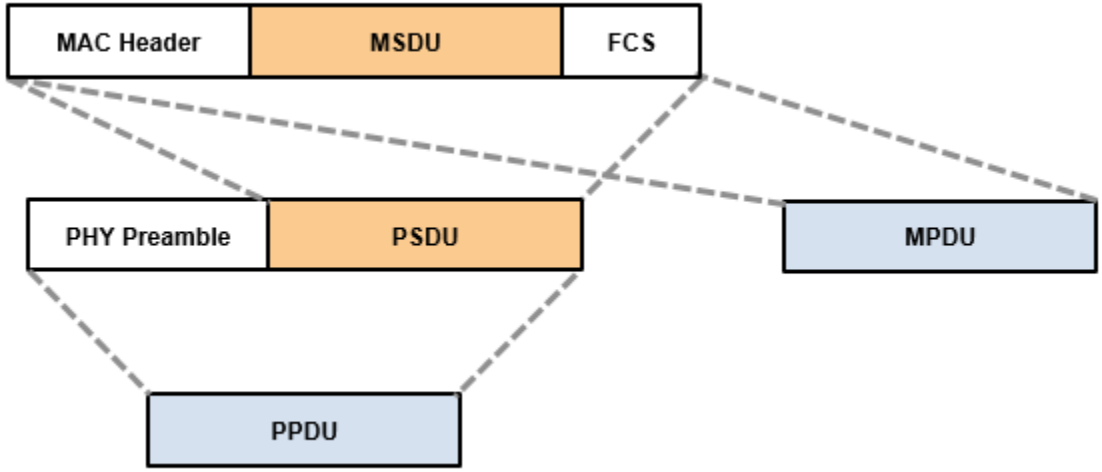



WLAN System Toolbox functionality focuses on PHY implementations and the exchange of PPDUs between PHY peers. Messages exchanged between protocol stack layers are briefly described here. For more information on these messages, see IEEE Std 802.11-2012 [2].

| Message | Description |
|---|---|
| MSDU — MAC service data unit | Messages that transfer information between the logical link control (LLC) layer and the MAC layer within an STA |

| Message | Description |
|---|---|
| MPDU — MAC protocol data unit | Messages that transfer information between MAC layer peers in communicating STAs |
| PSDU — PLCP service data unit | Messages that transfer information between the MAC and PHY layers within an STA |
| PPDU — PLCP protocol data unit | Messages that transfer information between PHY layer peers in communicating STAs |

This figure shows the distinction between these WLAN message data units for a nonaggregated MAC frame.




**Note:** In reference to PSDU, the terms PLCP SDU and PHY SDU appear in the 802.11 standard. PLCP is the physical layer convergence procedure sublayer of the PHY. No distinction is made when the terms are used between layers.

## Physical Layer Evolution

The IEEE 802.11 standardized implementation of WLAN has evolved since its first release in 1997. Today, it is deployed worldwide in unlicensed regions of the radio frequency spectrum. Since the first release, the 802.11 standard has progressed to include several physical layer implementations and has ensured backward compatibility with legacy releases. Over time, the maximum achievable transmission data rate has grown from 1 Mbps to nearly 7 Gbps.

WLAN System Toolbox provides native support for the various 802.11 standard versions listed here. The toolbox focuses on the physical layer and enables adaptation of standards-based functionality to explore custom implementations.

| Standard | Release Year | Modulation | Base Frequency (GHz) | Bandwidth (MHz) | Maximum Throughput (Mbps) | Antenna Scheme | PPDU Format |
|---|---|---|---|---|---|---|---|
| 802.11 | 1997 | DSSS | 2.4 | 11 | 2 | SISO | non-HT |
| 802.11b™ | 1999 | HR/ DSSS/ CCK | 2.4 | 11 | 11 | SISO | non-HT |
| 802.11a™ | 1999 | OFDM | 5 | 5, 10, 20 | 54 | SISO | non-HT |
| 802.11g™ | 2003 | 802.11b and 802.11a @ 2.4 GHz | | | | | |
| 802.11j™ | 2004 | OFDM | 5 | 10 | 27 | SISO | non-HT |
| 802.11n™ | 2009 | OFDM | 2.4 and 5 | 20, 40 | < 600 | MIMO, up to four streams | HT |
| 802.11p™ | 2010 | OFDM | 5 | 5, 10 | 27 | SISO | non-HT |
| 802.11ac | 2013 | OFDM | 5 | 20, 40, 80, 160, 80+80 | < 7000 | DL MU-MIMO up to eight streams | VHT |
| 802.11ah™ | 2016 | OFDM | < 1 | 1, 2, 4, 8, 16 | 346 | DL MU-MIMO up to four streams | S1G |

Deployment and commercial uptake grew with the increased data rates offered by 802.11b direct sequence spread spectrum (DSSS) with complementary code keying (CCK). At that time, companies began offering 802.11b products and systems for WLAN.

802.11a increased data rates by introducing an orthogonal frequency division multiplexing (OFDM) physical layer. However, OFDM was deployed at only 5 GHz, so uptake was slow. A short time later, the FCC allowed the use of OFDM at 2.4 GHz. The adoption of the 802.11g amendment offered the opportunity to operate the PHY defined by 802.11a at 2.4 GHz, with backward compatibility to the 802.11b PHY.

With 802.11n, a data rate increase came by way of widened channel bandwidth and allowance of up to four input/output streams. For 802.11ac, wider channels and up to eight input/output streams offers higher maximum throughputs. This increased throughput capability enables users to stream video to mobile devices in the home or at public mobile hot spots. The demand for bandwidth continues to grow and the IEEE 802.11 working groups continue to advance standards to raise the throughput ceiling.

For the history of IEEE 802.11 and to monitor working group activities, consult the IEEE website [1].

## References

[1] IEEE 802.11™: Wireless LANs. http://standards.ieee.org/about/get/802/802.11.html

[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] 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.

[4] Perahia, E., and R. Stacey. *Next Generation Wireless LANs: 802.11n and 802.11ac*. 2nd Edition. United Kingdom: Cambridge University Press, 2013.

## Related Examples

- “Create Configuration Objects” on page 2-2

- "Waveform Generation" on page 2-10
- "WLAN Channel Models" on page 2-24
- "Packet Recovery" on page 2-34
- "WLAN Packet Structure"

## External Websites

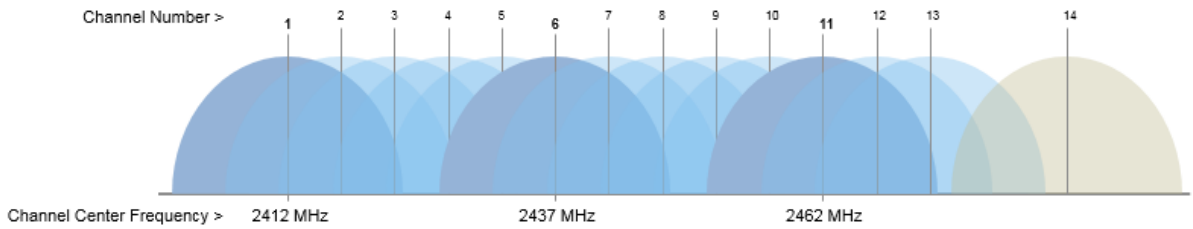
- http://standards.ieee.org/about/get/802/802.11.html

# WLAN Radio Frequency Channels

WLAN operates in unlicensed radio frequency (RF) spectrum allocated by governing bodies in individual countries for signal transmissions. Apropriate regulatory bodies specify maximum allowable output power.

Refer to IEEE Std 802.11-2012, Annex E for detailed description of country information, operating classes, and behavior limits. The discussion here is restricted to identification of the WLAN operating frequency channel designations.

In general, the 2.4 GHz and 5 GHz bands of operation designate channels spaced 5 MHz apart, with noted exceptions. As an example, the 2.4 GHz band designates channels 1 through 13 spaced 5 MHz apart plus a 14th channel 12 MHz from channel 13. Defined WLAN channel bandwidths are greater than 5 MHz, therefore cross-channel interference limits the number of designated usable channels. Access point deployments manage interference from neighboring cells by operating on non-overlapping channels. In the United States, the 2.4 GHz band designated usable non-overlapping channels are 1, 6, and 11.




The channel center frequency, $F_{CENTER}$, is calculated using the starting frequency, $F_{START}$, and the channel number.

$F_{CENTER}$ in MHz = $F_{START}$ + (5×*Channel Number*)

Example: Determine the center frequency for channel number 6 in the 2.4 GHz band.

$F_{CENTER}$ in MHz = 2407 + (5×6) = 2437 MHz.

| 802.11 channels | | |
|---|---|---|
| ***Channel Number*** | **$F_{START}$, Starting Frequency (MHz)** | **Comments** |
| 1, ..., 13 | 2407 | Refer to IEEE Std 802.11-2012 [1] and IEEE Std 802.11ac-2013 [2] for country and release specific restrictions. |
| 14 | 2414 | |
| 132, 133, 134, 136, 137, 138 | 3000 | |
| 131, ..., 138 | 3002.5 | |
| 183, ..., 197 | 4000 | |
| 182, ..., 189 | 4002.5 | |
| 21, 25 | 4850 | |
| 11, 13, 15, 17, 19 | 4890 | |
| 1, ..., 10 | 4937.5 | |
| 7, ..., 12, 16<br>34, ..., 60 in increments of 2<br>64<br>100, 104, 106, 108<br>112, 114, 116<br>120, 122, 124, 128<br>132, 136, 138<br>140, 144, 149<br>153, 155, 157<br>161, 165, 169<br>171, ..., 184 in increments of 1 | 5000 | |

| 802.11 channels | | |
|---|---|---|
| ***Channel Number*** | **$F_{START}$, Starting Frequency (MHz)** | **Comments** |
| 6, ..., 11<br><br>170, ..., 184 in increments of 1 | 5002.5 | |

## 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] 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.

# Limitations

| In this section... |
| --- |
|  |
|  |
|  |
|  |

## MATLAB Compiler Support

The WLAN System Toolbox product does not support the MATLAB Compiler™. You cannot compile functions in the toolbox.

## Code Generation Support

The WLAN System Toolbox product does not support automatic generation of HDL code.

## Fixed-Point Support

The WLAN System Toolbox product does not support fixed-point data types.

## Block Support

The WLAN System Toolbox product does not contain Simulink® blocks.

# Acknowledgments

This table lists the copyright owners of content used in the WLAN System Toolbox documentation.

| Source | Copyright Owner |
|---|---|
| Content from IEEE Std 802.11-2012 | Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved. |
| Content from IEEE Std 802.11ac-2013 | Adapted and reprinted with permission from IEEE. Copyright IEEE 2013. All rights reserved. |

# Terminology

This list defines commonly used WLAN abbreviations and acronyms. These terms may appear in the documents that describe MathWorks® products that model communications and WLAN.

| | |
|---|---|
| $N_T$ | Number of transmit antennas |