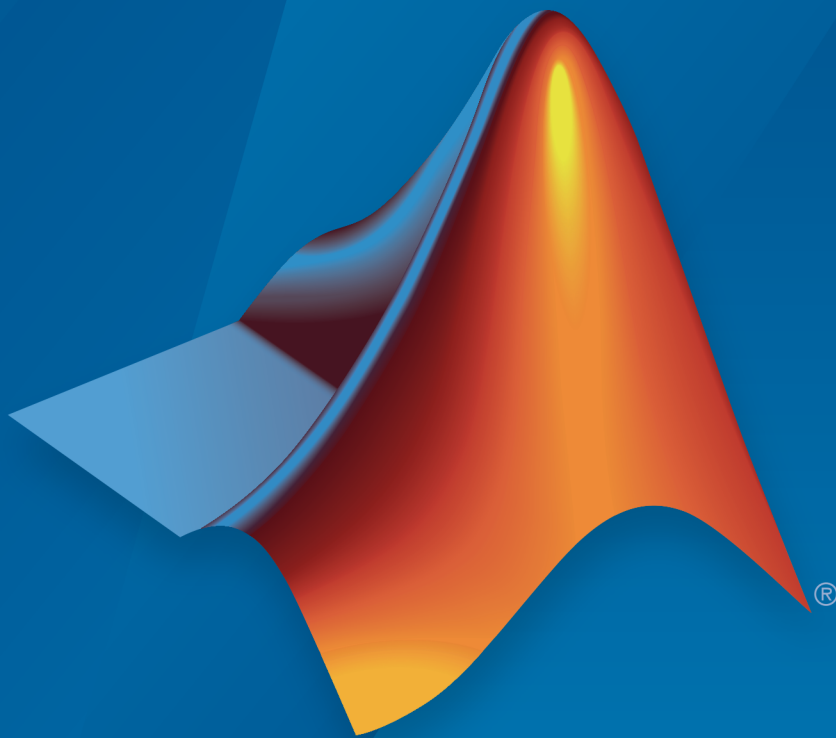


# WLAN System Toolbox™

## Getting Started Guide



# MATLAB®

R2015b

 MathWorks®

## How to Contact MathWorks



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



Phone: 508-647-7000



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

*WLAN System Toolbox™ Getting Started Guide*

© COPYRIGHT 2015 by MathWorks, Inc.

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

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

### **Trademarks**

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

### **Patents**

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

### **Revision History**

October 2015      Online only      New for Version 1.0 (R2015b)

## Introduction

1

<b>WLAN System Toolbox Product Description</b> .....	1-2
Key Features .....	1-2
<b>Expected Background</b> .....	1-3

## Tutorials

2

<b>Create Configuration Objects</b> .....	2-2
Create VHT Configuration Object .....	2-2
Create HT Configuration Object .....	2-4
Create Non-HT Configuration Object .....	2-6
Create Waveform Generator Configuration Object .....	2-7
Create Recovery Configuration Object .....	2-9
<b>Waveform Generation</b> .....	2-11
Generate WLAN Waveforms .....	2-12
<b>WLAN Channel Models</b> .....	2-21
<b>Packet Recovery</b> .....	2-32
VHT Packet Recovery .....	2-32
HT Packet Recovery .....	2-37
Non-HT Packet Recovery .....	2-42
<b>End-to-End Simulation</b> .....	2-48
<b>Test and Measurement</b> .....	2-49

<b>What Is WLAN?</b> .....	3-2
Network Architecture .....	3-2
Physical Layer Evolution .....	3-4
WLAN Protocol Stack .....	3-5
<b>Limitations</b> .....	3-7
MATLAB Compiler Support .....	3-7
Code Generation Support .....	3-7
Fixed-Point Support .....	3-7
Block Support .....	3-7
<b>Acknowledgments</b> .....	3-8
<b>Terminology</b> .....	3-9

# 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 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 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.





# Tutorials

---

- “Create Configuration Objects” on page 2-2
- “Waveform Generation” on page 2-11
- “WLAN Channel Models” on page 2-21
- “Packet Recovery” on page 2-32
- “End-to-End Simulation” on page 2-48
- “Test and Measurement” on page 2-49

## 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: false  
            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: false
            MCS: 0
    ChannelCoding: 'BCC'
        APEPLength: 1024
    GuardInterval: 'Long'
        GroupID: 63
        PartialAID: 275
```

### Create Object and Override 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: false
            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: false  
        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: true
```

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: true
```

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: true
```

### 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'  
    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'  
            MCS: 3  
        PSDULength: 1000  
        NumTransmitAntennas: 4
```

### Create Object and Override 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: true  
        PSDULength: 1000
```

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

### Create Waveform Generator Configuration Object

This example shows how to create waveform generator 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 generator configuration object and view the default settings.

```
cfg = wlanGeneratorConfig

cfg =
    wlanGeneratorConfig with properties:
        NumPackets: 1
        IdleTime: 0
        ScramblerInitialization: 93
        Windowing: true
        WindowTransitionTime: 1e-07
```

Modify the defaults to specify ten packets and an idle time of 20 microseconds.

```
cfg.NumPackets = 10;
cfg.IdleTime = 20e-6

cfg =
    wlanGeneratorConfig with properties:
        NumPackets: 10
        IdleTime: 2e-05
        ScramblerInitialization: 93
        Windowing: true
        WindowTransitionTime: 1e-07
```

### Create Object and Override Default Property Values

Create a generator configuration object. Use Name, Value pairs to turn off windowing and to change the scrambler initialization state.

```
cfg2 = wlanGeneratorConfig('Windowing',false,'ScramblerInitialization',17)

cfg2 =
```



```
wlanGeneratorConfig with properties:
```

```
    NumPackets: 1
      IdleTime: 0
  ScramblerInitialization: 17
    Windowing: false
```

## 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: 0.75
    EqualizationMethod: 'MMSE'
    PilotPhaseTracking: 'PreEQ'
```

Modify the default to specify no pilot phase tracking.

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

```
cfgRec =
```

```
    wlanRecoveryConfig with properties:
```

```
      OFDMSymbolOffset: 0.75
    EqualizationMethod: 'MMSE'
    PilotPhaseTracking: 'None'
```

### Create Object and Override 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: 0.6  
        EqualizationMethod: 'ZF'  
        PilotPhaseTracking: 'PreEQ'
```

### See Also

[wlanGeneratorConfig Properties](#) | [wlanHTConfig Properties](#) | [wlanNonHTConfig Properties](#) | [wlanRecoveryConfig Properties](#) | [wlanVHTConfig Properties](#)

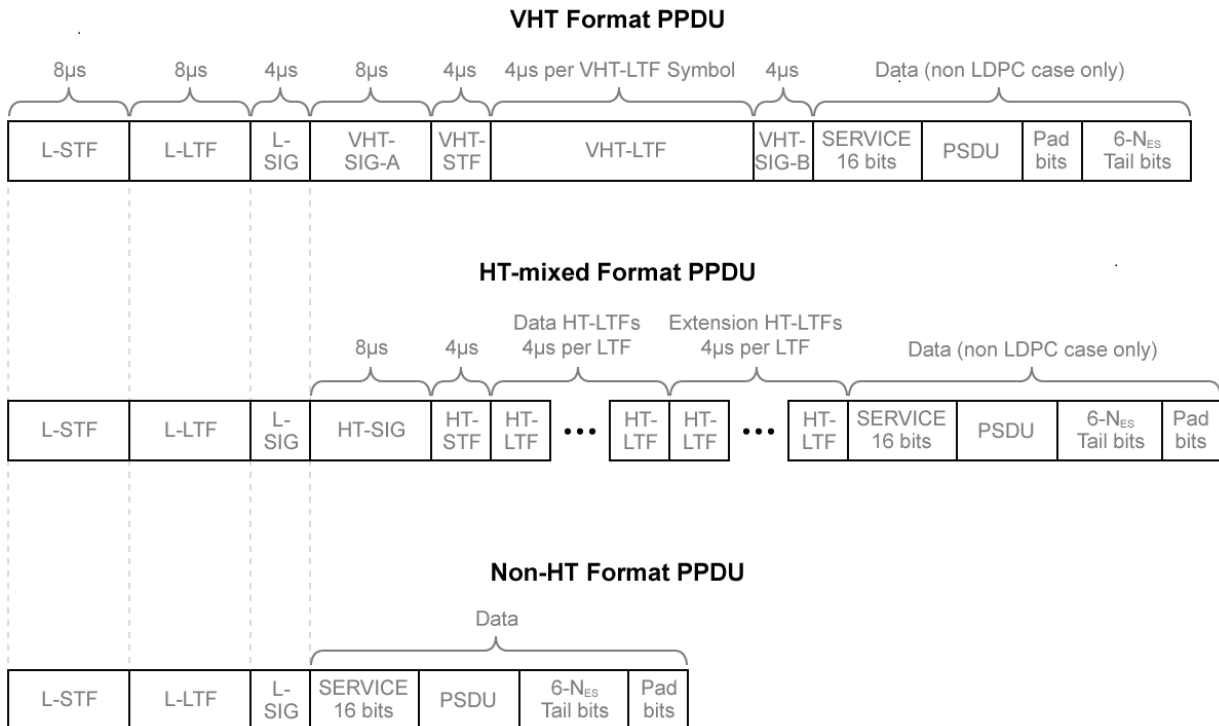
### Related Examples

- “Waveform Generation” on page 2-11
- “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<sup>12</sup> 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.



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.

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

Use `wlanWaveformGenerator` to populate all the PPDU fields (preamble and data) per the configuration objects. The waveform is generated in single call. `wlanWaveformGenerator` accepts a bit stream and configuration objects for the desired format (`wlanVHTConfig`, `wlanHTConfig`, or `wlanNonHTConfig`) and waveform generation (`wlanGeneratorConfig`) configurations.

You can also create a PPDU waveform by generating and concatenating waveforms for individual PPDU fields.

PPDU Format	Individual Field Functions
VHT	<code>wlanLSTF</code> , <code>wlanLLTF</code> , <code>wlanLSIG</code> , <code>wlanVHTSTF</code> , <code>wlanVHTLTF</code> , <code>wlanVHTSIGA</code> , <code>wlanVHTSIGB</code> , and <code>wlanVHTData</code>
HT	<code>wlanLSTF</code> , <code>wlanLLTF</code> , <code>wlanLSIG</code> , <code>wlanHTSTF</code> , <code>wlanHTLTF</code> , <code>wlanHTSIG</code> , and <code>wlanHTData</code>
Non-HT for OFDM modulation	<code>wlanLSTF</code> , <code>wlanLLTF</code> , <code>wlanLSIG</code> , and <code>wlanNonHTData</code>

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

## Generate WLAN Waveforms

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

### Create Waveform Generator Configuration Object

Create a generator configuration object. Using `Name`, `Value` pairs to the specify five packets and 20 microseconds of idle time.

```
wgc = wlanGeneratorConfig('NumPackets',5,'IdleTime',20e-6)
```

```
wgc =
```

wlanGeneratorConfig with properties:

```

        NumPackets: 5
        IdleTime: 2e-05
    ScramblerInitialization: 93
        Windowing: true
    WindowTransitionTime: 1e-07

```

You can now use this object to generate the VHT, HT, and non-HT WLAN format waveforms.

### Generate VHT Format Waveform

Create a VHT configuration object and waveform. Plot the VHT format waveform, scaling the  $x$ -axis relative to the channel bandwidth.

```

vht = wlanVHTConfig
bits = [1;0;0;1;1];
fs = 80e6;
vhtWaveform = wlanWaveformGenerator(bits,vht,wgc);
time = ([0:length(vhtWaveform)-1]/fs)*1e6;
plot(time,abs(vhtWaveform))
xlabel('Time (microseconds)');
ylabel('Magnitude');

```

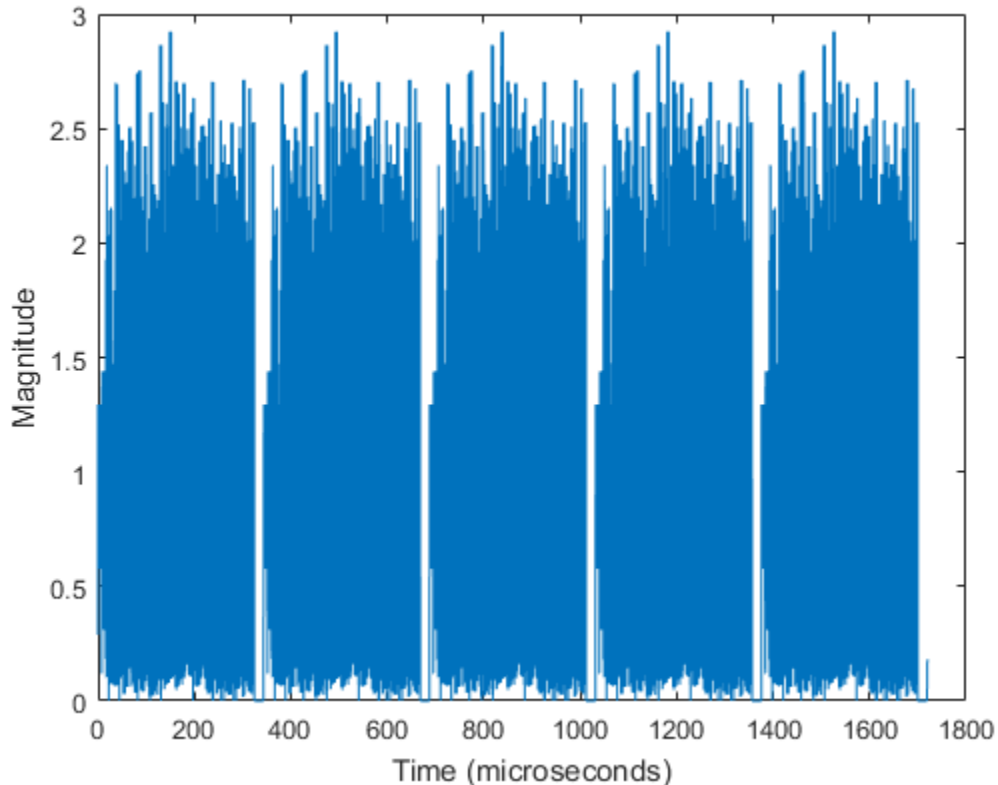
vht =

wlanVHTConfig with properties:

```

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

```



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, modify the idle time, and generate an HT format waveform.

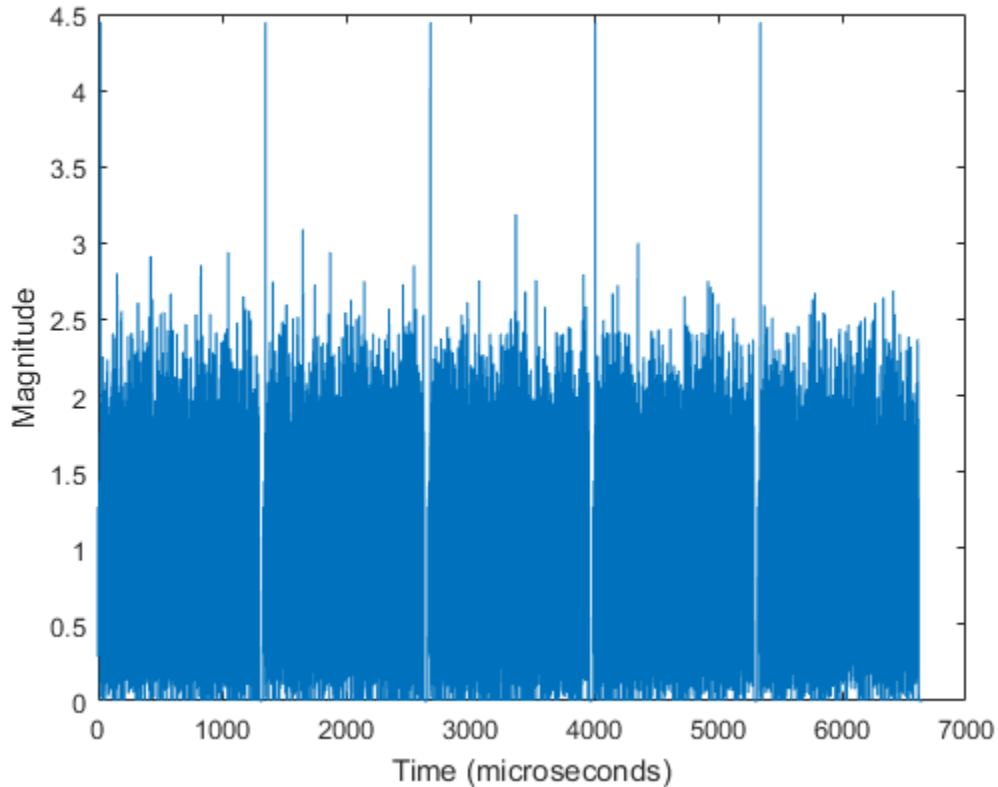
```
ht = wlanHTConfig
fs = 20e6;
wgc.IdleTime = 30e-6;
htWaveform = wlanWaveformGenerator(bits,ht,wgc);
time = ([0:length(htWaveform)-1]/fs)*1e6;
plot(time,abs(htWaveform))
```

```
xlabel ('Time (microseconds)');  
ylabel ('Magnitude');
```

```
ht =
```

```
wlanHTConfig with properties:
```

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



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, modify the number of packets, and the idle time. Generate a non-HT format DSSS waveform.

```
nht = wlanNonHTConfig('Modulation','DSSS')
fs = 11e6;
wgc.NumPackets = 2;
wgc.IdleTime = 5e-6;
nhtDSSSWaveform = wlanWaveformGenerator(bits,nht,wgc);
time = ([0:length(nhtDSSSWaveform)-1]/fs)*1e6;
```

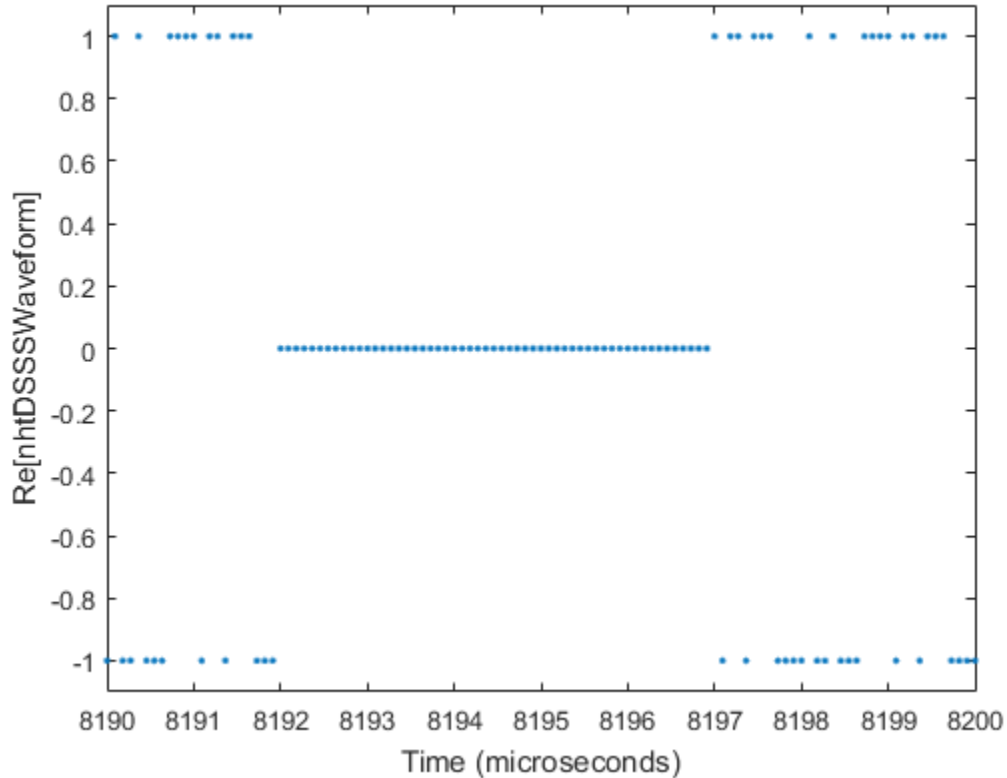


```
plot(time,real(nhtDSSSWaveform),'.')
xlabel ('Time (microseconds)');
ylabel('Re[nhtDSSSWaveform]');
axis([8190,8200,-1.1,1.1])
```

```
nht =
```

```
wlanNonHTConfig with properties:
```

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



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 packet one, the 5 microsecond idle period, and beginning of packet two for the non-HT format DSSS modulated waveform.

### Generate Non-HT Format OFDM Waveform

Create a non-HT configuration object, modify the number of packets, and the idle time. Generate a non-HT format OFDM waveform.

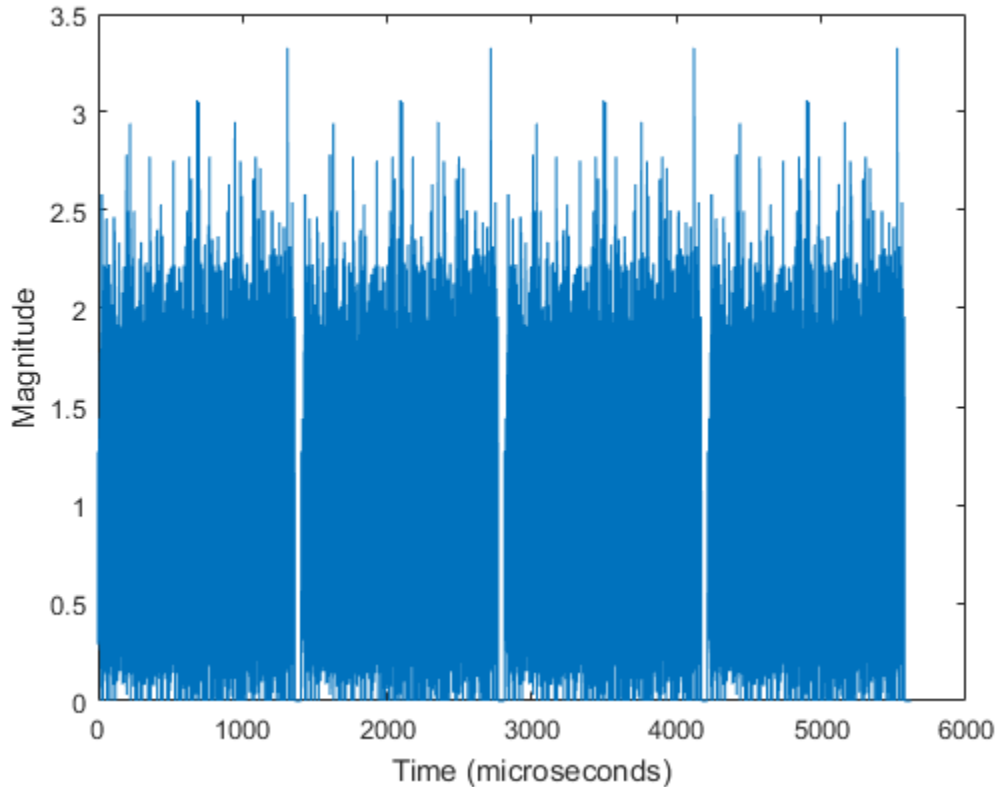
```
nht = wlanNonHTConfig
fs = 20e6;
wgc.IdleTime = 45e-6;
wgc.NumPackets = 4;
nhtWaveform = wlanWaveformGenerator(bits,nht,wgc);
```

```
time = ([0:length(nhtWaveform)-1]/fs)*1e6;  
plot(time,abs(nhtWaveform))  
xlabel('Time (microseconds)');  
ylabel('Magnitude');
```

```
nht =
```

```
wlanNonHTConfig with properties:
```

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



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

### See Also

[wlanGeneratorConfig Properties](#) | [wlanHTConfig Properties](#) | [wlanNonHTConfig Properties](#) | [wlanRecoveryConfig Properties](#) | [wlanVHTConfig Properties](#)

### Related Examples

- “Create Configuration Objects” on page 2-2
- “WLAN Channel Models” on page 2-21
- “What Is WLAN?” on page 3-2

## WLAN Channel Models

Options for channel modeling include addition of AWGN, channel models from the Communications System Toolbox™, plus TGn and TGac models from the WLAN System Toolbox™. This example shows how to pass WLAN waveforms through different channels. Because no front end filtering has been added and the oversampling rate is one, it is sufficient to set the channel model sampling frequency to match the channel bandwidth. Use a spectrum analyzer to display waveforms before and after they pass through the channel.

### Create Waveform Generator Configuration Object

Create a generator configuration object and bit stream. Use them when generating VHT, HT and non-HT WLAN format waveforms.

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

```
wgc =
```

```
wlanGeneratorConfig with properties:
```

```
    NumPackets: 1
      IdleTime: 0
ScramblerInitialization: 93
      Windowing: true
WindowTransitionTime: 1e-07
```

### Pass VHT Waveform Through TGac SISO Channel.

Create a VHT configuration object and generate an 80 MHz VHT waveform.

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

Create a TGac SISO channel and AWGN channel. Use Name, Value pairs to set the sample rate and channel bandwidth, to enable path loss and shadowing, and to use the Model-D delay profile.

```
bw = vht.ChannelBandwidth;
fs = 80e6;
chTGac = wlanTGacChannel('SampleRate',fs,'ChannelBandwidth',bw,...
    'LargeScaleFadingEffect','Pathloss and shadowing','DelayProfile','Model-D')
```

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

```
chTGac =
```

```
System: wlanTGacChannel
```

```
Properties:
```

```
    SampleRate: 8000000
    DelayProfile: 'Model-D'
    ChannelBandwidth: 'CBW80'
    CarrierFrequency: 5250000000
    TransmitReceiveDistance: 3
    NormalizePathGains: true
    UserIndex: 0
    TransmissionDirection: 'Downlink'
    NumTransmitAntennas: 1
    NumReceiveAntennas: 1
    LargeScaleFadingEffect: 'Pathloss and shadowing'
    FluorescentEffect: true
    PowerLineFrequency: '60Hz'
    RandomStream: 'Global stream'
    NormalizeChannelOutputs: true
```

```
chAWGN =
```

```
System: comm.AWGNChannel
```

```
Properties:
```

```
    NoiseMethod: 'Variance'
    VarianceSource: 'Input port'
    RandomStream: 'Global stream'
```

Calculate the noise variance for a receiver 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. Pass the transmitted waveform through the noisy, TGac channel.

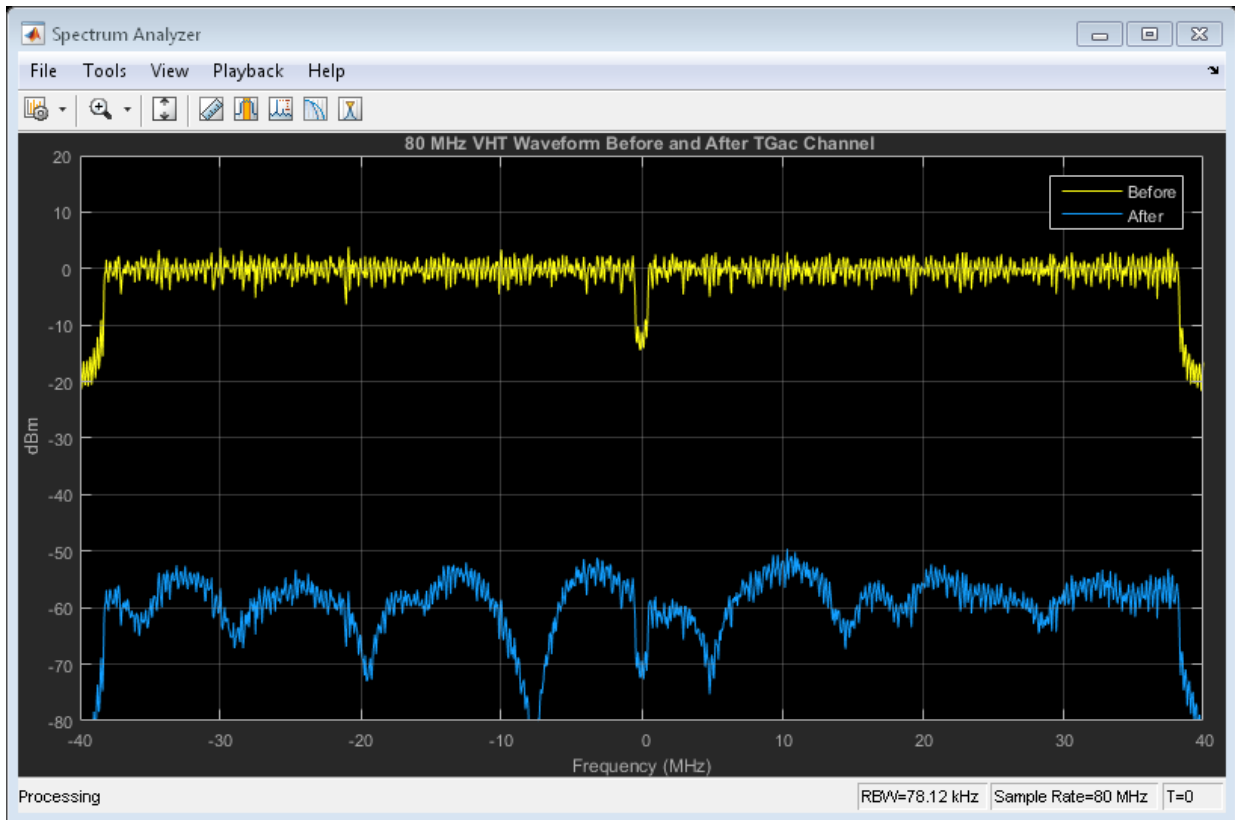
```
nVar = 10^((-228.6 + 10*log10(290) + 10*log10(fs) + 9)/10);
postChVHT = step(chAWGN, step(chTGac, preChVHT), nVar);
```

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';
sa = dsp.SpectrumAnalyzer('SampleRate',fs,'ShowLegend',true,...
    'SpectralAverages',10,'Title',title,'ChannelNames',{'Before','After'});
step(sa,[preChVHT,postChVHT]);

```



The path loss results from the default transmitter-to-receiver distance of three meters, and from shadowing effects. This path loss accounts for the roughly 50 to 60 dB of separation between the waveform before and after passing through the TGac channel. The frequency selectivity of the delay profile is seen in the signal level variation across the frequency spectrum.

### Pass HT Waveform Through TGn SISO Channel

Create an HT configuration object and generate an HT waveform.

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

Create a TGn SISO channel and AGWN channel. Use **Name, Value** pairs to set the sample rate, to enable path loss and shadowing, and to use the Model-F delay profile.

```
fs = 20e6;  
chTGn = wlanTGnChannel('SampleRate',fs,'LargeScaleFadingEffect',...  
    'Pathloss and shadowing','DelayProfile','Model-F')  
chAWGN = comm.AWGNChannel('NoiseMethod','Variance','VarianceSource','Input port')
```

```
chTGn =
```

```
System: wlanTGnChannel
```

```
Properties:
```

```
    SampleRate: 20000000  
    DelayProfile: 'Model-F'  
    CarrierFrequency: 5250000000  
    TransmitReceiveDistance: 3  
    NormalizePathGains: true  
    NumTransmitAntennas: 1  
    NumReceiveAntennas: 1  
    LargeScaleFadingEffect: 'Pathloss and shadowing'  
    RandomStream: 'Global stream'  
    NormalizeChannelOutputs: true
```

```
chAWGN =
```

```
System: comm.AWGNChannel
```

```
Properties:
```

```
    NoiseMethod: 'Variance'  
    VarianceSource: 'Input port'  
    RandomStream: 'Global stream'
```

Calculate the noise variance for a receiver 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. Pass the transmitted waveform through the noisy TGn channel.

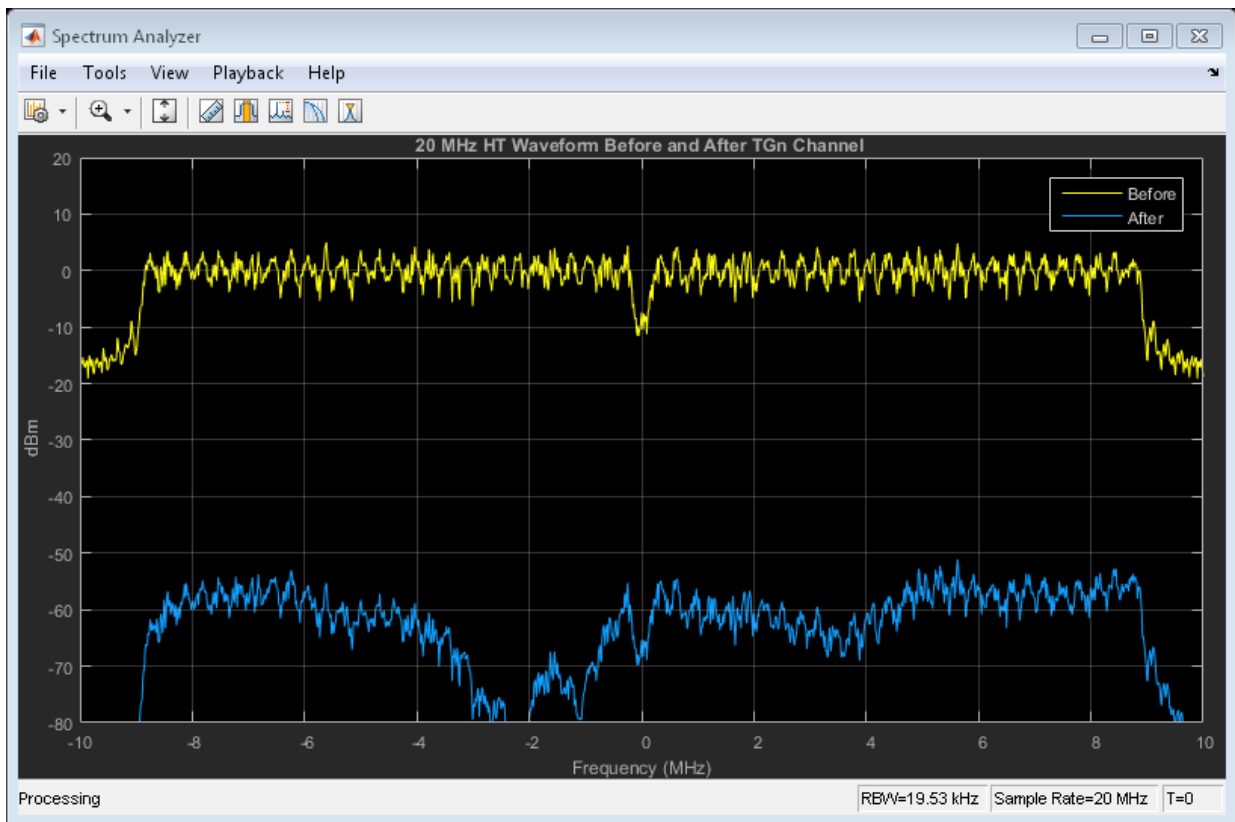
```
nVar = 10^((-228.6 + 10*log10(290) + 10*log10(fs) + 9)/10);
```



```
postChHT = step(chAWGN, step(chTGN,preChHT), 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';
sa = dsp.SpectrumAnalyzer('SampleRate',fs,'ShowLegend',true,...
    'SpectralAverages',10,'Title',title,'ChannelNames',{'Before','After'});
step(sa,[preChHT,postChHT]);
```



The path loss results from the default transmitter-to-receiver distance of three meters, and from shadowing effects. This path loss accounts for the roughly 50 to 60 dB of separation between the waveform before and after passing through the TGN channel. The frequency selectivity of the delay profile is seen in the signal level variation across the frequency spectrum.

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

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

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

Calculate free-space path loss for a transmitter-to-receiver separation distance of three 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;  
fc = 2.4e9;  
pathLoss = 10^(-log10(4*pi*dist*(fc/3e8)));  
fs = 20e6;  
trms = 2/fs;  
ch802 = stdchan(1/fs,dist,'802.11g',trms)  
chAWGN = comm.AWGNChannel('NoiseMethod','Variance','VarianceSource','Input port')
```

```
ch802 =
```

```
    ChannelType: 'Rayleigh'  
    InputSamplePeriod: 5.0000e-08  
    DopplerSpectrum: [1x1 doppler.jakes]  
    MaxDopplerShift: 3  
    PathDelays: [1x21 double]  
    AvgPathGaindB: [1x21 double]  
    NormalizePathGains: 1  
    StoreHistory: 0  
    StorePathGains: 0  
    PathGains: [1x21 double]  
    ChannelFilterDelay: 0  
    ResetBeforeFiltering: 1  
    NumSamplesProcessed: 0
```

```
chAWGN =
```

```
System: comm.AWGNChannel
```

```
Properties:
```

```
    NoiseMethod: 'Variance'  
    VarianceSource: 'Input port'
```

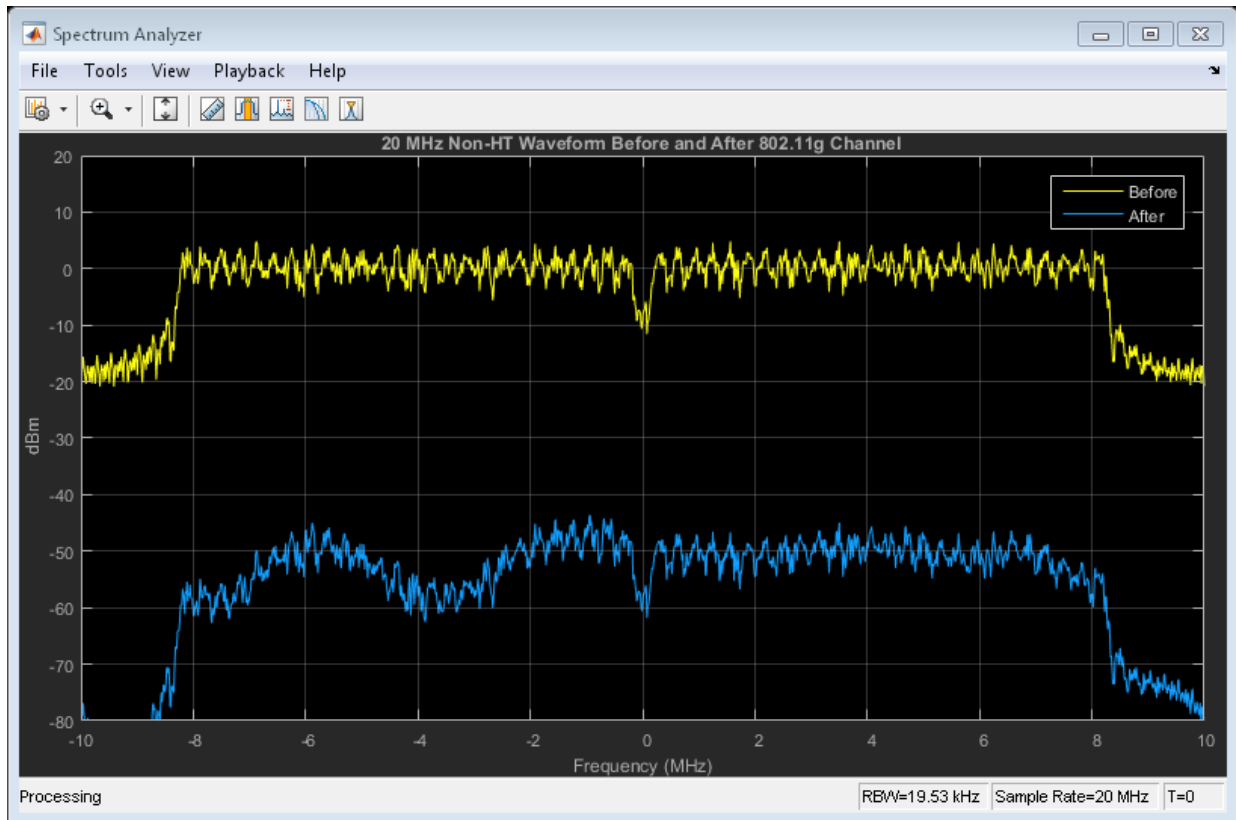
```
RandomStream: 'Global stream'
```

Calculate the noise variance for a receiver 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. Pass the transmitted waveform through the noisy, lossy 802.11g channel.

```
nVar = 10^((-228.6 + 10*log10(290) + 10*log10(fs) + 9)/10);  
postChNonHT = step(chAWGN, filter(ch802,preChNonHT), 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';  
sa = dsp.SpectrumAnalyzer('SampleRate',fs,'ShowLegend',true,...  
    'SpectralAverages',10,'Title',title,'ChannelNames',{'Before','After'});  
step(sa,[preChNonHT,postChNonHT]);
```



The free-space path loss accounts for the approximately 50 dB of separation between the waveform before and after it passes through the 802.11g channel.

### Pass VHT Waveform Through TGac MIMO Channel.

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 three. Since 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,wgc);
```

Create a TGac MIMO channel and an AWGN channel. Disable large-scale fading effects.

```
bw = vht.ChannelBandwidth;
fs = 80e6;
chTGac = wlanTGacChannel('SampleRate',fs,'ChannelBandwidth',bw,...
'NumTransmitAntennas',ntx,'NumReceiveAntennas',nrx);
chTGac.LargeScaleFadingEffect = 'None'
chAWGN = comm.AWGNChannel('NoiseMethod','Variance','VarianceSource','Input port')
```

```
chTGac =
```

```
System: wlanTGacChannel
```

```
Properties:
```

```
    SampleRate: 80000000
    DelayProfile: 'Model-B'
    ChannelBandwidth: 'CBW80'
    CarrierFrequency: 5250000000
    TransmitReceiveDistance: 3
    NormalizePathGains: true
    UserIndex: 0
    TransmissionDirection: 'Downlink'
    NumTransmitAntennas: 4
    TransmitAntennaSpacing: 0.5
    NumReceiveAntennas: 3
    ReceiveAntennaSpacing: 0.5
    LargeScaleFadingEffect: 'None'
    RandomStream: 'Global stream'
    NormalizeChannelOutputs: true
```

```
chAWGN =
```

```
System: comm.AWGNChannel
```

```
Properties:
```

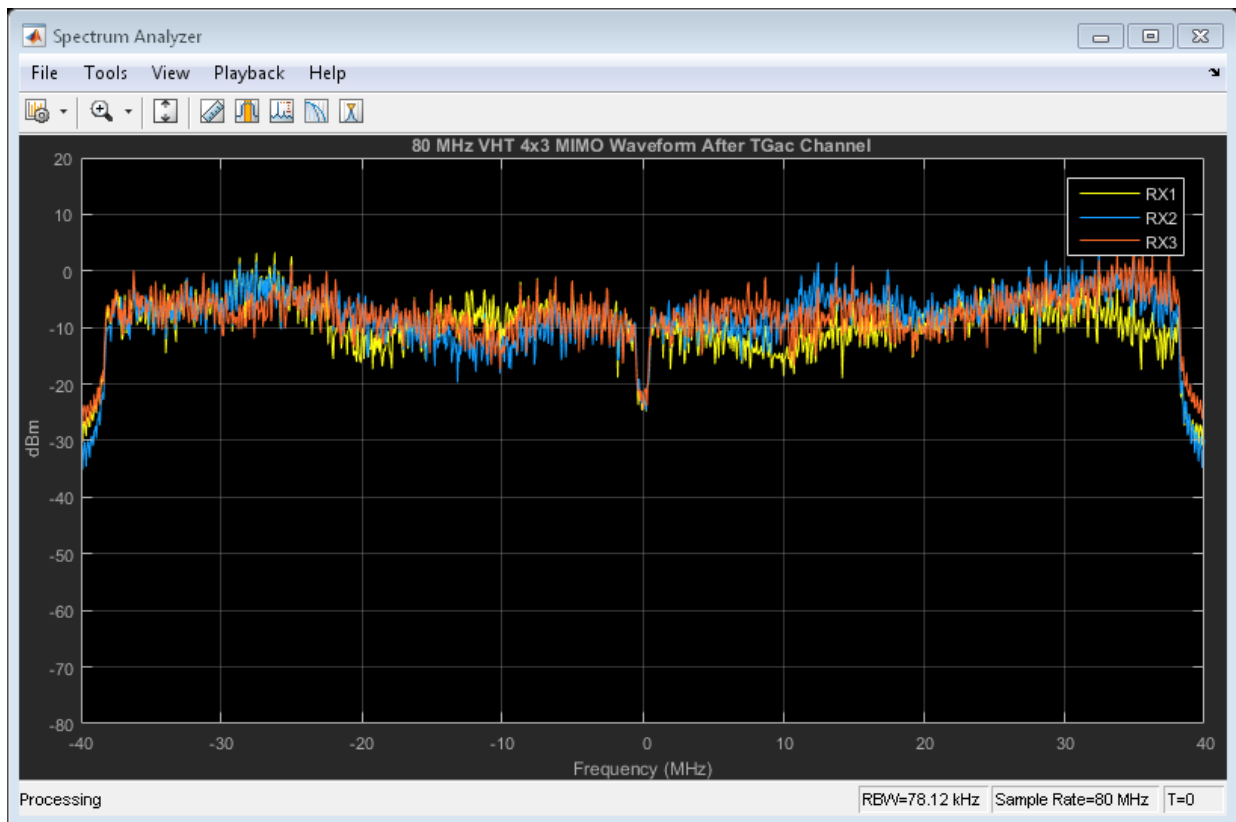
```
    NoiseMethod: 'Variance'
    VarianceSource: 'Input port'
    RandomStream: 'Global stream'
```

Calculate the noise variance for a receiver 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. Pass the transmitted waveform through the noisy TGac channel.

```
nVar = 10^((-228.6 + 10*log10(290) + 10*log10(fs) + 9)/10);
postChVHT = step(chAWGN, step(chTGac, preChVHT), 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';
sa = dsp.SpectrumAnalyzer('SampleRate', fs, 'ShowLegend', true, ...
    'SpectralAverages', 10, 'Title', title, 'ChannelNames', {'RX1', 'RX2', 'RX3'});
step(sa, postChVHT);
```



The overlaid signals show the TGac channel variation between 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` | wlanGeneratorConfig Properties  
| wlanHTConfig Properties | wlanNonHTConfig Properties | wlanRecoveryConfig  
Properties | wlanVHTConfig Properties

## Related Examples

- “Waveform Generation” on page 2-11
- “Packet Recovery” on page 2-32
- “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;  
chTGac = wlanTGacChannel('SampleRate',fs,'ChannelBandwidth',chBW,...  
    'LargeScaleFadingEffect','Pathloss and shadowing');
```



```
chAWGN = 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 = step(chAWGN,step(chTGac,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 =
    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.

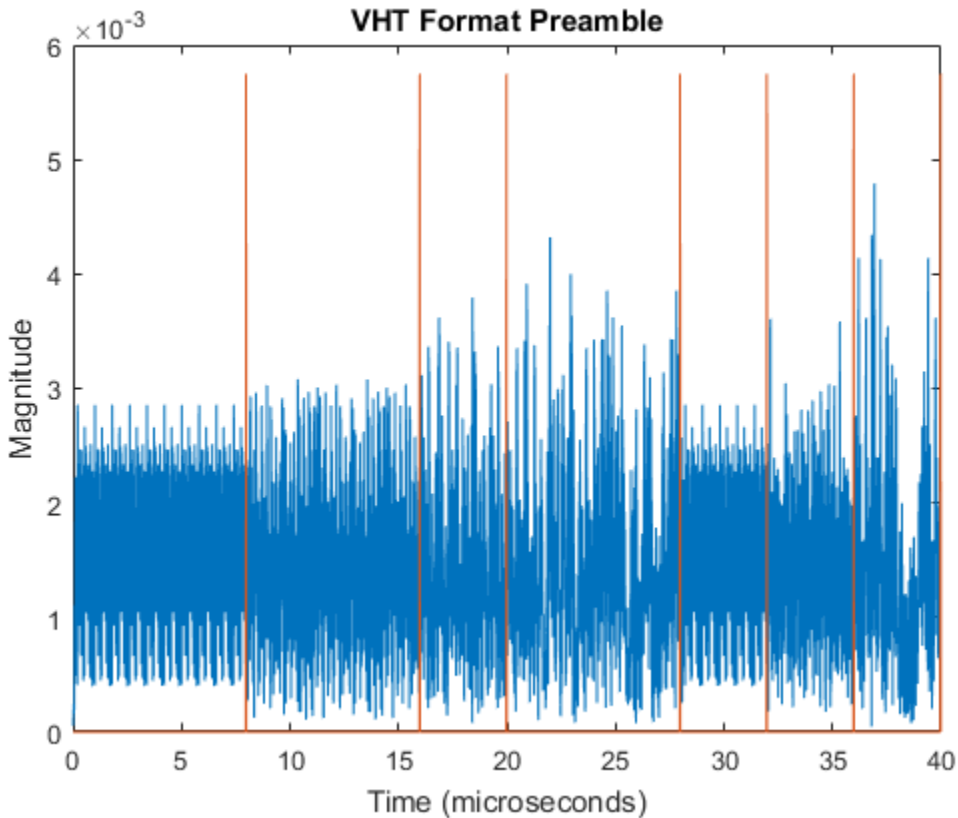
```
numSamples = fieldInd.VHTSIGB(2);
```

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 =
```

```
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 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 =
```

```
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 =
```

```
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 =
    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.

```
wgc = wlanGeneratorConfig;
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);
htlftf = wlanHTLTF(ht);
htData = wlanHTData(txPSDU,ht);
```

Concatenate the individual fields to create a single PPDU waveform.

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

### Pass HT Waveform Through TGn SISO Channel

Create TGn SISO channel and AWGN channel objects.

```
fs = 20e6;
chTGn = wlanTGnChannel('SampleRate',fs,'LargeScaleFadingEffect','Pathloss and shadowing');
chAWGN = 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 = step(chAWGN,step(chTGn,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 =
```

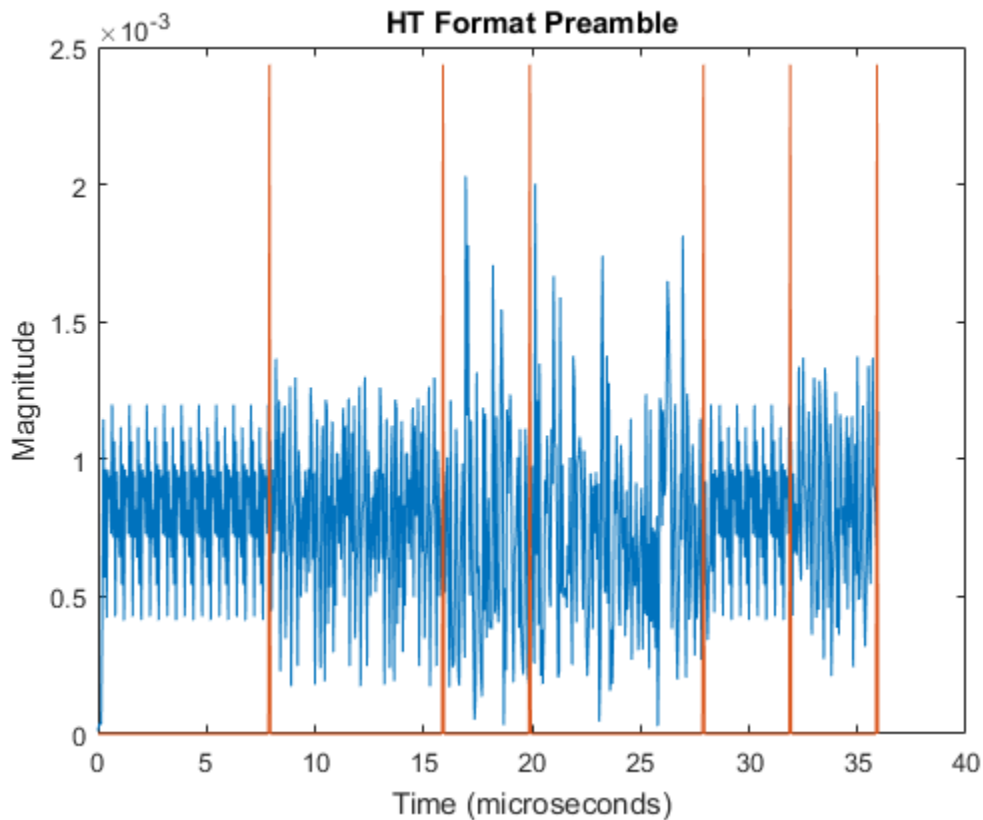
```
    LSTF: [1 160]
    LLTF: [161 320]
    LSIG: [321 400]
    HTSIG: [401 560]
    HTSTF: [561 640]
    HTLTF: [641 720]
    HTData: [721 9200]
```

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 =
```



```
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 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 =
```

```
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 =
```

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 step through recovery 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);
chAWGN = 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 = step(chAWGN,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 =
```

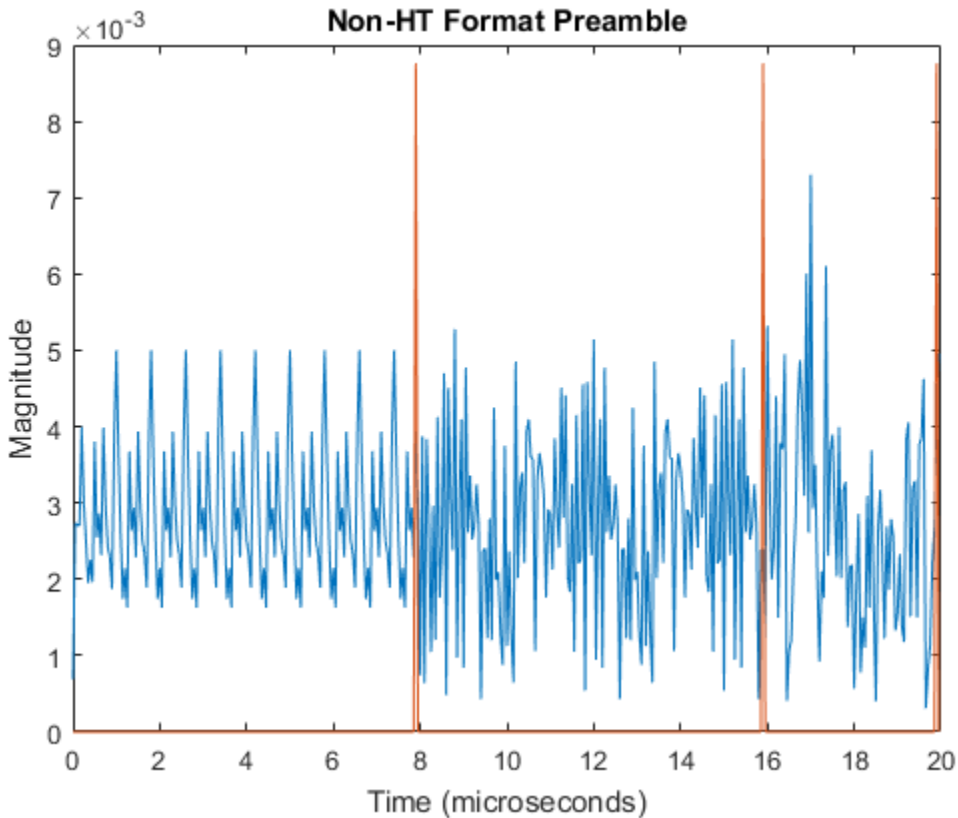
```
    LSTF: [1 160]  
    LLTF: [161 320]  
    LSIG: [321 400]  
    NonHTData: [401 7120]
```

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    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

[wlanGeneratorConfig Properties](#) | [wlanHTConfig Properties](#) | [wlanNonHTConfig Properties](#) | [wlanRecoveryConfig Properties](#) | [wlanVHTConfig Properties](#)

### Related Examples

- “WLAN Channel Models” on page 2-21

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

## **End-to-End Simulation**



## Test and Measurement



# About WLAN

---

- “What Is WLAN?” on page 3-2
- “Limitations” on page 3-7
- “Acknowledgments” on page 3-8
- “Terminology” on page 3-9

# 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<sup>34</sup> group of standards and branded as Wi-Fi<sup>®</sup> 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” on page 3-2

“Physical Layer Evolution” on page 3-4

“WLAN Protocol Stack” on page 3-5

## Network Architecture

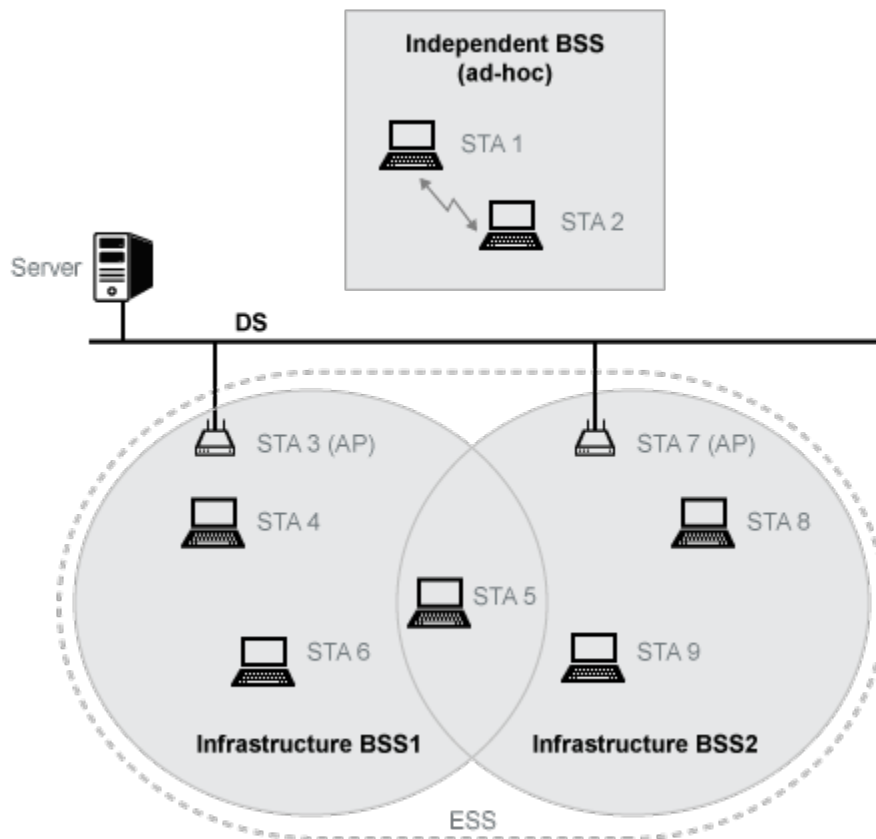
In IEEE 802.11, a group of stations (STAs) within a defined coverage area and with appropriate association to each other form a basic service set (BSS). The BSS is a basic building block for 802.11 network architecture. A basic service area (BSA) defines an area containing STAs within a BSS. STAs can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

In WLAN, uplink and downlink connections use the same protocol modulation and coding schemes. Also, any compliant STA can serve as an access point (AP). The figure depicts WLAN components and network architectures built up from BSSs. IEEE 802.11 defines the network architectures.

---

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.
- 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 BSSs interconnected by a DS.

## 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 (PHY) 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.

Standard	Features and Formats	Release Year
802.11	DSSS @ 2.4 GHz, 1 Mbps or 2 Mbps, SISO	1997
802.11b™	HR/DSSS/CCK @ 2.4 GHz, 5.5 Mbps or 11 Mbps, SISO	1999
802.11a™	OFDM @ 5 GHz, 54 Mbps, SISO	1999
802.11g™	802.11b and 802.11a @ 2.4 GHz	2003
802.11n™	2.4 GHz and 5 GHz, 20/40 MHz, MIMO up to four streams, < 600 Mbps	2009
802.11ac	5 GHz, 20/40/80/160/80+80 MHz, DL MU-MIMO up to eight streams, < 7 Gbps	2013

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 this time, companies began offering 802.11b products and systems for WLAN.

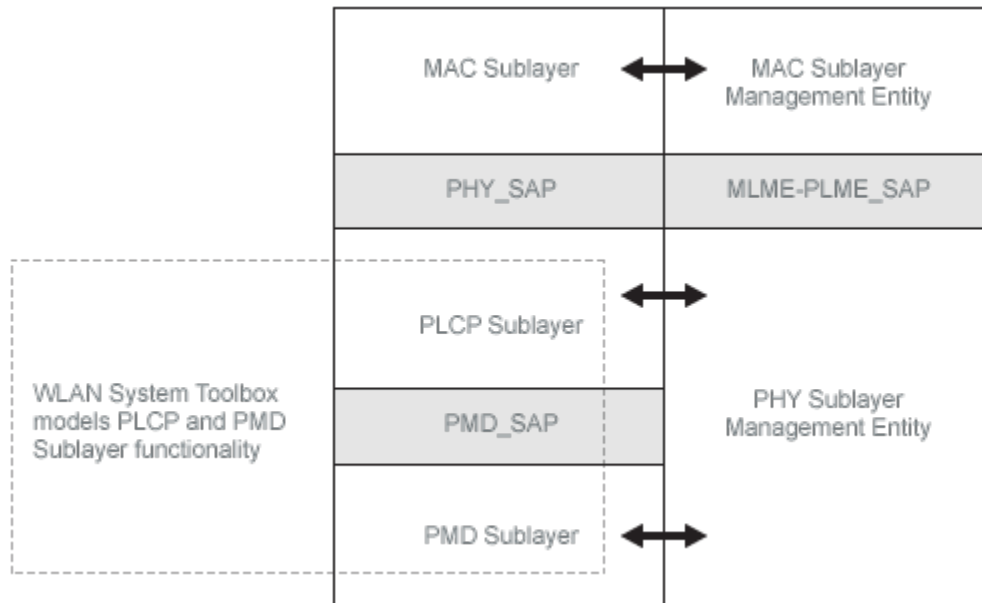
802.11a increased data rates by introducing an orthogonal frequency division multiplexing (OFDM) PHY. However, OFDM was deployed at only 5 GHz, so uptake was slow. A short time later, the FCC allowed the use of OFDM at 2.4 GHz. The adoption of 802.11g amendment offered the opportunity to operate the 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 future plans, consult the IEEE website [1].

## WLAN Protocol Stack

The interworking reference model shown here includes a subset of the network components associated with the data link layer (DLL) and PHY. IEEE Std 802.11-2012 [2], Section 4.9.2 describes the interworking reference model for 802.11. 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 System Toolbox functionality supports various 802.11 standard versions. It focuses on PHY implementations and enables adaptation of standards-based functionality to explore custom implementations.

## 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 2-11
- “WLAN Channel Models” on page 2-21
- “Packet Recovery” on page 2-32
- “WLAN Packet Structure”

### External Websites

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



## Limitations

In this section...
“MATLAB Compiler Support” on page 3-7
“Code Generation Support” on page 3-7
“Fixed-Point Support” on page 3-7
“Block Support” on page 3-7

### **MATLAB Compiler Support**

The WLAN System Toolbox product does not support the MATLAB<sup>®</sup> Compiler<sup>™</sup>. 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<sup>®</sup> blocks.

## Acknowledgments

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

<b>Source</b>	<b>Copyright Owner</b>
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.



