WLAN Toolbox™

Getting Started Guide

MATLAB®

R2020a

MathWorks®

How to Contact MathWorks

Latest news:	`www.mathworks.com`
Sales and services:	`www.mathworks.com/sales_and_services`
User community:	`www.mathworks.com/matlabcentral`
Technical support:	`www.mathworks.com/support/contact_us`
Phone:	508-647-7000

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

WLAN Toolbox™ Getting Started Guide

Revision History

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

Contents

Introduction

WLAN Toolbox Product Description

Simulate, analyze, and test WLAN communications systems

WLAN Toolbox provides standards-compliant functions for the design, simulation, analysis, and testing of wireless LAN communications systems. It includes configurable physical layer waveforms for IEEE® 802.11ax/ac/ad/ah and 802.11b/a/g/n/j/p standards. It also provides transmitter, channel modeling, and receiver operations, including channel coding (BCC and LDPC), modulation (OFDM, DSSS, and CCK), spatial stream mapping, channel models (TGay, TGax, TGac, TGah, and TGn), and MIMO receivers.

The toolbox provides reference designs to help you perform baseband link-level simulations and multi-node system-level simulations. You can generate and parse common MAC frames. You can also perform signal measurements such as channel power, spectrum mask, and occupied bandwidth, and create test benches for the end-to-end simulation of WLAN communications links.

You can study the effects of RF designs and interference on system performance. Using WLAN Toolbox with RF instruments or 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.

Tutorials

Create Configuration Objects

WLAN Toolbox uses value objects to organize properties required for generation of IEEE 802.11™ b/a/g/n/j/p/ac/ah/ad/ax 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 HE MU Configuration Object

This example shows how to create HE MU 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 HE MU configuration object with the `AllocationIndex` set to 0 and view the default settings.

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

```
cfgHEMU = 
  wlanHEMUConfig with properties:

                     RU: {1x9 cell}
                   User: {1x9 cell}
    NumTransmitAntennas: 1
                   STBC: 0
          GuardInterval: 3.2000
              HELTFType: 4
                SIGBMCS: 0
                SIGBDCM: 0
       UplinkIndication: 0
               BSSColor: 0
          SpatialReuse: 0
           TXOPDuration: 127
            HighDoppler: 0

   Read-only properties:
       ChannelBandwidth: 'CBW20'
        AllocationIndex: 0
```

Modify the defaults to specify four transmit antennas.

```
cfgHEMU.NumTransmitAntennas = 4
```

```
cfgHEMU = 
  wlanHEMUConfig with properties:

                     RU: {1x9 cell}
                   User: {1x9 cell}
    NumTransmitAntennas: 4
                   STBC: 0
          GuardInterval: 3.2000
              HELTFType: 4
                SIGBMCS: 0
                SIGBDCM: 0
```

```
       UplinkIndication: 0
               BSSColor: 0
           SpatialReuse: 0
           TXOPDuration: 127
            HighDoppler: 0

   Read-only properties:
       ChannelBandwidth: 'CBW20'
        AllocationIndex: 0
```

Create Object and Override Default Property Values

Create an HE MU configuration object with `AllocationIndex` set to 192. Use `Name,Value` pairs to set the spatial reuse to `3`.

```
cfgHEMU = wlanHEMUConfig(192,'SpatialReuse',3)
```

```
cfgHEMU = 
  wlanHEMUConfig with properties:

                     RU: {[1x1 wlanHEMURU]}
                   User: {[1x1 wlanHEMUUser]}
    NumTransmitAntennas: 1
                   STBC: 0
          GuardInterval: 3.2000
              HELTFType: 4
        SIGBCompression: 1
                SIGBMCS: 0
                SIGBDCM: 0
       UplinkIndication: 0
               BSSColor: 0
           SpatialReuse: 3
           TXOPDuration: 127
            HighDoppler: 0

   Read-only properties:
       ChannelBandwidth: 'CBW20'
        AllocationIndex: 192
```

Create Single User HE Configuration Object

This example shows how to create single user HE 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 single user HE configuration object and view the default settings.

```
hesu = wlanHESUConfig
```

```
hesu = 
  wlanHESUConfig with properties:

        ChannelBandwidth: 'CBW20'
```

```
           ExtendedRange: 0
      NumTransmitAntennas: 1
      NumSpaceTimeStreams: 1
           SpatialMapping: 'Direct'
      PreHESpatialMapping: 0
                     STBC: 0
                      MCS: 0
                      DCM: 0
            ChannelCoding: 'LDPC'
               APEPLength: 100
            GuardInterval: 3.2000
               HELTFType: 4
         UplinkIndication: 0
                 BSSColor: 0
             SpatialReuse: 0
             TXOPDuration: 127
              HighDoppler: 0
     NominalPacketPadding: 0
```

Modify the defaults to specify an four transmit antennas.

```
hesu.NumTransmitAntennas = 4
```

```
hesu = 
  wlanHESUConfig with properties:

         ChannelBandwidth: 'CBW20'
            ExtendedRange: 0
      NumTransmitAntennas: 4
      NumSpaceTimeStreams: 1
           SpatialMapping: 'Direct'
      PreHESpatialMapping: 0
                     STBC: 0
                      MCS: 0
                      DCM: 0
            ChannelCoding: 'LDPC'
               APEPLength: 100
            GuardInterval: 3.2000
                HELTFType: 4
         UplinkIndication: 0
                 BSSColor: 0
             SpatialReuse: 0
             TXOPDuration: 127
              HighDoppler: 0
     NominalPacketPadding: 0
```

Create Object and Override Default Property Values

Create a single user HE configuration object. Use `Name,Value` pairs to set the modulation and coding scheme to 9 and to enable space-time block coding.

```
hesu2 = wlanHESUConfig('MCS',9,'STBC',true)
```

```
hesu2 = 
  wlanHESUConfig with properties:

         ChannelBandwidth: 'CBW20'
```

```
          ExtendedRange: 0
    NumTransmitAntennas: 1
    NumSpaceTimeStreams: 1
         SpatialMapping: 'Direct'
    PreHESpatialMapping: 0
                   STBC: 1
                    MCS: 9
                    DCM: 0
          ChannelCoding: 'LDPC'
             APEPLength: 100
          GuardInterval: 3.2000
              HELTFType: 4
       UplinkIndication: 0
               BSSColor: 0
           SpatialReuse: 0
           TXOPDuration: 127
            HighDoppler: 0
   NominalPacketPadding: 0
```

Create DMG Configuration Object

This example shows how to create DMG 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 DMG configuration object and view the default settings. By default, the configuration object creates properties to model the DMG control PHY.

```
dmg = wlanDMGConfig
```

```
dmg = 
  wlanDMGConfig with properties:

                      MCS: '0'
           TrainingLength: 0
               PSDULength: 1000
    ScramblerInitialization: 2
               Turnaround: 0
```

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

```
dmg.MCS = 5
```

```
dmg = 
  wlanDMGConfig with properties:

                      MCS: 5
           TrainingLength: 0
               PSDULength: 1000
    ScramblerInitialization: 2
            AggregatedMPDU: 0
                  LastRSSI: 0
               Turnaround: 0
```

For the various configurations, different sets of configuration fields apply and are visible. By changing the MCS setting from 0 to 5, we see that the configured object includes the `AggregationMPDU` and `LastRSSI` fields.

Create Object and Override Default Property Values

Create a DMG configuration object for OFDM PHY. Use `Name,Value` pairs to set the MCS to `14` and specify four training fields.

```
dmg2 = wlanDMGConfig('MCS',14,'TrainingLength',4)
```

```
dmg2 = 
  wlanDMGConfig with properties:

                        MCS: 14
             TrainingLength: 4
                 PacketType: 'TRN-R'
        BeamTrackingRequest: 0
            TonePairingType: 'Static'
                 PSDULength: 1000
    ScramblerInitialization: 2
             AggregatedMPDU: 0
                   LastRSSI: 0
                 Turnaround: 0
```

Create S1G Configuration Object

This example shows how to create S1G 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 S1G configuration object and view the default settings.

```
s1g = wlanS1GConfig
```

```
s1g = 
  wlanS1GConfig with properties:

       ChannelBandwidth: 'CBW2'
               Preamble: 'Short'
               NumUsers: 1
    NumTransmitAntennas: 1
    NumSpaceTimeStreams: 1
         SpatialMapping: 'Direct'
                   STBC: 0
                    MCS: 0
             APEPLength: 256
          GuardInterval: 'Long'
             PartialAID: 37
        UplinkIndication: 0
                  Color: 0
         TravelingPilots: 0
     ResponseIndication: 'None'
     RecommendSmoothing: 1
```

```
   Read-only properties:
          ChannelCoding: 'BCC'
             PSDULength: 258
```

Modify the defaults to specify an 8 MHz channel bandwidth, three transmit antennas, and three space-time streams.

```
s1g.ChannelBandwidth = 'CBW8';
s1g.NumTransmitAntennas = 3;
s1g.NumSpaceTimeStreams = 3
```

```
s1g = 
  wlanS1GConfig with properties:

       ChannelBandwidth: 'CBW8'
               Preamble: 'Short'
               NumUsers: 1
    NumTransmitAntennas: 3
    NumSpaceTimeStreams: 3
         SpatialMapping: 'Direct'
                   STBC: 0
                    MCS: 0
             APEPLength: 256
          GuardInterval: 'Long'
             PartialAID: 37
       UplinkIndication: 0
                  Color: 0
        TravelingPilots: 0
     ResponseIndication: 'None'
     RecommendSmoothing: 1

   Read-only properties:
          ChannelCoding: 'BCC'
             PSDULength: 261
```

Create Object and Override Default Property Values

Create a S1G configuration object. Use `Name,Value` pairs to set the MCS to 5 and to specify two transmit antennas.

```
s1g2 = wlanS1GConfig('MCS',5,'NumTransmitAntennas',2)
```

```
s1g2 = 
  wlanS1GConfig with properties:

       ChannelBandwidth: 'CBW2'
               Preamble: 'Short'
               NumUsers: 1
    NumTransmitAntennas: 2
    NumSpaceTimeStreams: 1
         SpatialMapping: 'Direct'
                   STBC: 0
                    MCS: 5
             APEPLength: 256
          GuardInterval: 'Long'
```

```
             PartialAID: 37
        UplinkIndication: 0
                   Color: 0
         TravelingPilots: 0
      ResponseIndication: 'None'
      RecommendSmoothing: 1

   Read-only properties:
           ChannelCoding: 'BCC'
              PSDULength: 258
```

As currently configured, this object is not a valid S1G configuration. Validation of the object occurs when it is the input to a calling function. When spatial mapping is `'Direct'`, the number of space-time streams must equal the number of transmit antennas. Changing the number of space time streams to match the number of transmit antennas is one option to make the configuration of the object valid.

```
s1g2.NumSpaceTimeStreams = 2
```

```
s1g2 = 
  wlanS1GConfig with properties:

        ChannelBandwidth: 'CBW2'
                Preamble: 'Short'
                NumUsers: 1
     NumTransmitAntennas: 2
     NumSpaceTimeStreams: 2
          SpatialMapping: 'Direct'
                    STBC: 0
                     MCS: 5
              APEPLength: 256
           GuardInterval: 'Long'
              PartialAID: 37
        UplinkIndication: 0
                   Color: 0
         TravelingPilots: 0
      ResponseIndication: 'None'
      RecommendSmoothing: 1

   Read-only properties:
           ChannelCoding: 'BCC'
              PSDULength: 258
```

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

   Read-only properties:
             PSDULength: 1035
```

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

   Read-only properties:
             PSDULength: 1050
```

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: 0
                   MCS: 7
         ChannelCoding: 'BCC'
            APEPLength: 1024
         GuardInterval: 'Long'
               GroupID: 63
            PartialAID: 275

  Read-only properties:
            PSDULength: 1167
```

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

  Read-only properties:
            PSDULength: 1166
```

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
         AggregatedMPDU: 0
     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
         AggregatedMPDU: 0
     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
          AggregatedMPDU: 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 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: 1
      PSDULength: 1000
```

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

See Also

Objects
`wlanDMGConfig` | `wlanHEMUConfig` | `wlanHERecoveryConfig` | `wlanHESUConfig` | `wlanHTConfig` | `wlanNonHTConfig` | `wlanS1GConfig` | `wlanVHTConfig`

Related Examples

- "Waveform Generation" on page 2-14
- "What Is WLAN?" on page 3-2

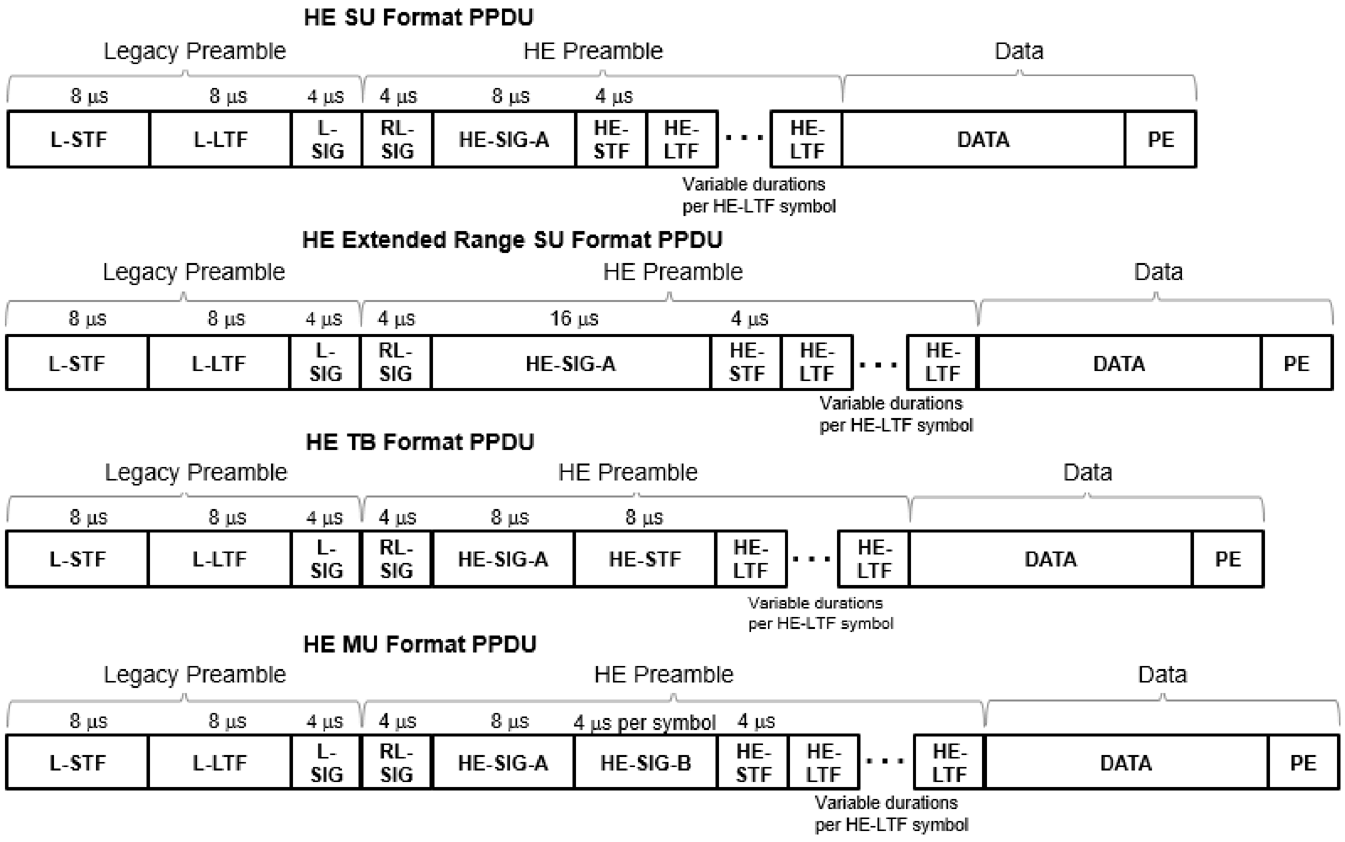
Waveform Generation

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[1] standards define a physical layer conformance procedure (PLCP) protocol data unit (PPDU) as the transmission unit at the physical layer. For a detailed description of the PPDU field structures for each transmission format, see “WLAN PPDU Structure”.

HE Format PPDU

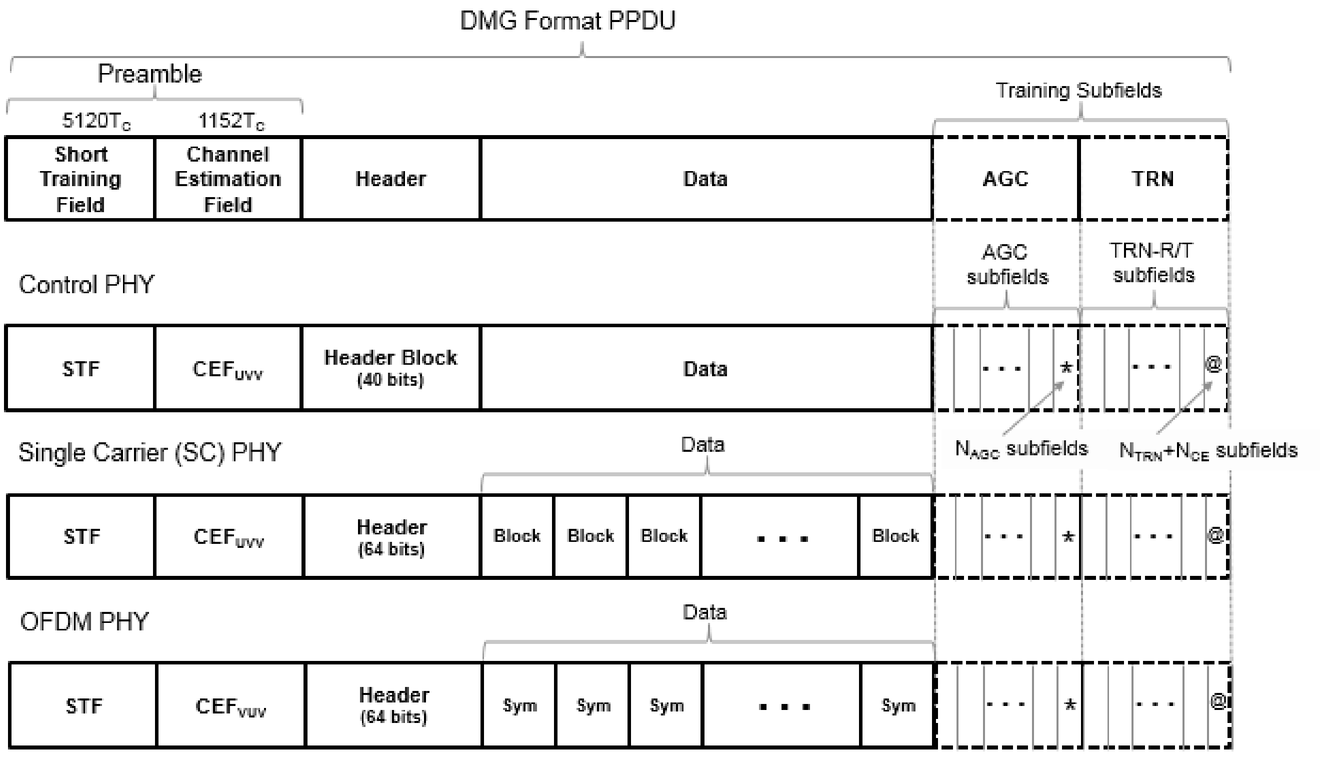
In HE, there are four transmission modes supported: single user, single user extended range, trigger-based, and multi-user.



DMG Format PPDU

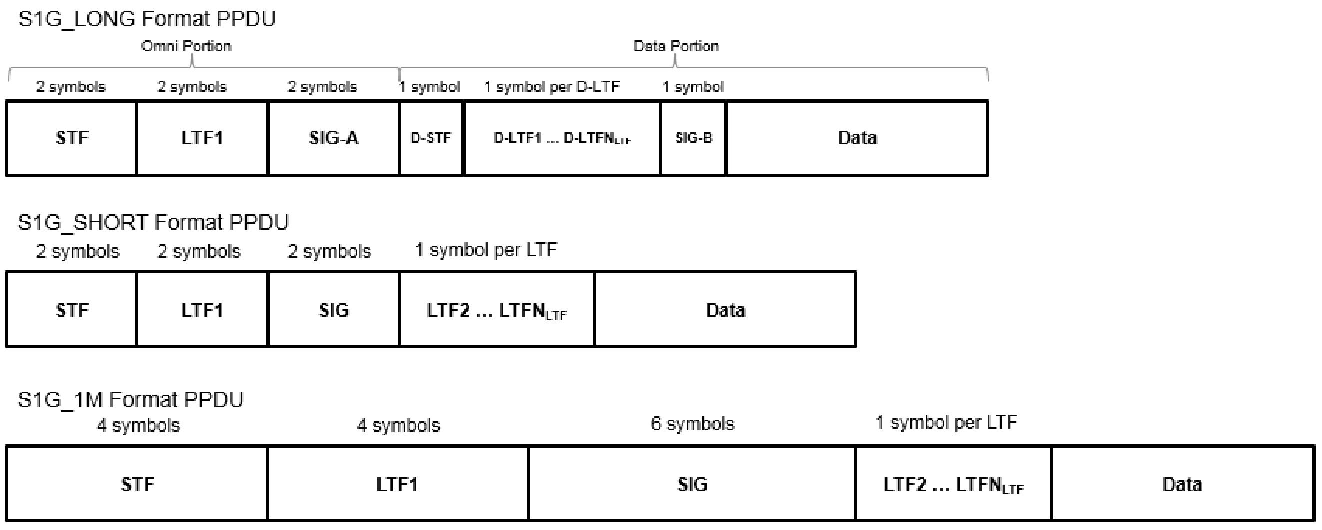
In DMG, there are three physical layer (PHY) modulation schemes supported: control, single carrier, and OFDM.

1. IEEE Std 802.11-2016 Adapted and reprinted with permission from IEEE. Copyright IEEE 2016. All rights reserved.



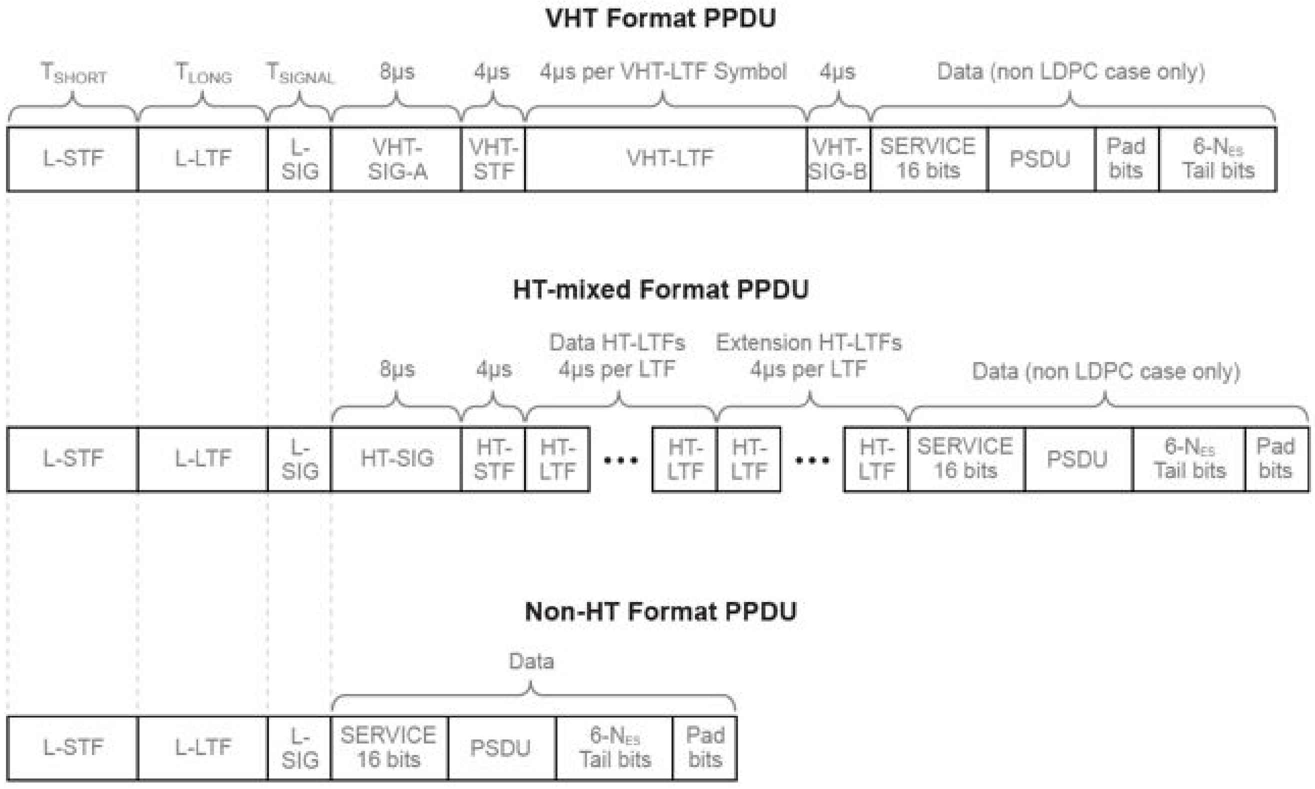
S1G Format PPDU

In S1G, there are three transmission modes: S1G_LONG, S1G_SHORT, and S1G_1M. Each transmission mode has a specific PPDU preamble structure.



VHT, HT, and non-HT Format PPDUs

The VHT, HT, and non-HT PPDU formats consist of preamble and data fields.



Use WLAN 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 (`wlanHESUConfig`, `wlanHEMUConfig`, `wlanDMGConfig`, `wlanS1GConfig` `wlanVHTConfig`, `wlanHTConfig`, or `wlanNonHTConfig`) and `Name,Value` pairs to configure the waveform.

Generate WLAN Waveforms

Generate HE, DMG, 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.

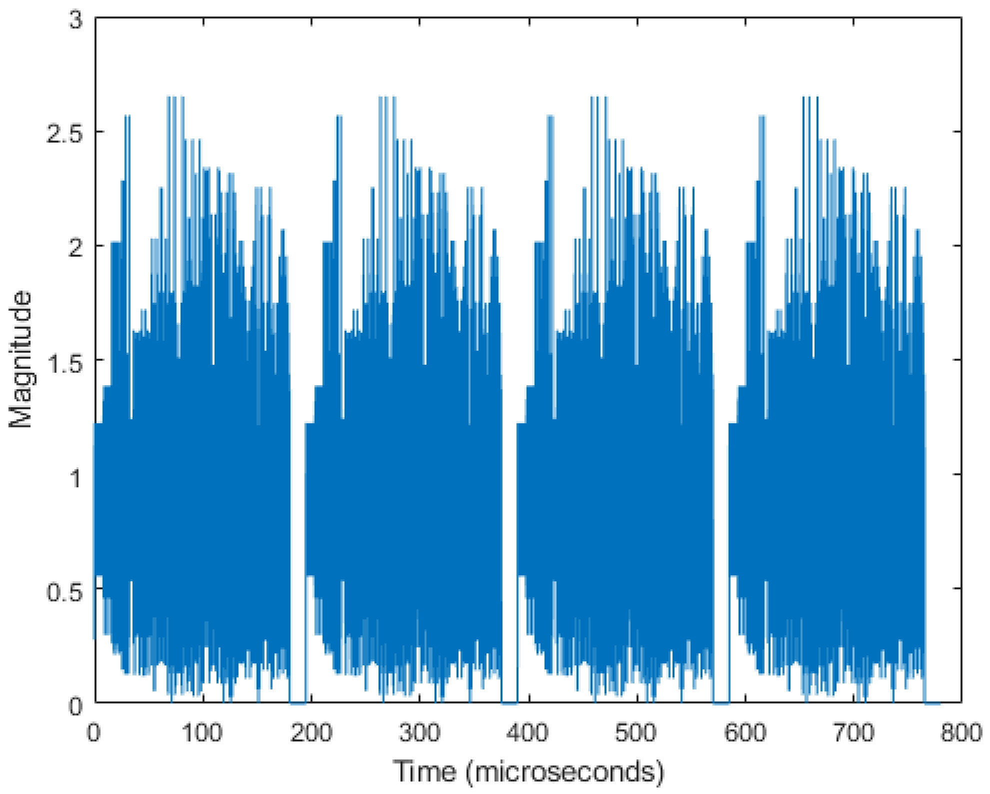
Generate Single User HE Format Waveform

Create an HE single-user (HE SU) configuration object and waveform. Using `Name,Value` pairs, specify 4 packets and 15 microseconds of idle time. Display the configuration object and inspect its properties and settings.

```
cfgHESU = wlanHESUConfig;
bits = [1;0;0;1;1];
hesuWaveform = wlanWaveformGenerator(bits,cfgHESU, ...
    'NumPackets',4,'IdleTime',15e-6);
```

Plot the single user HE format waveform, scaling the *x-axis* relative to the channel bandwidth.

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



The plot shows four single user HE format packets, with each packet separated by 15 microseconds of idle time.

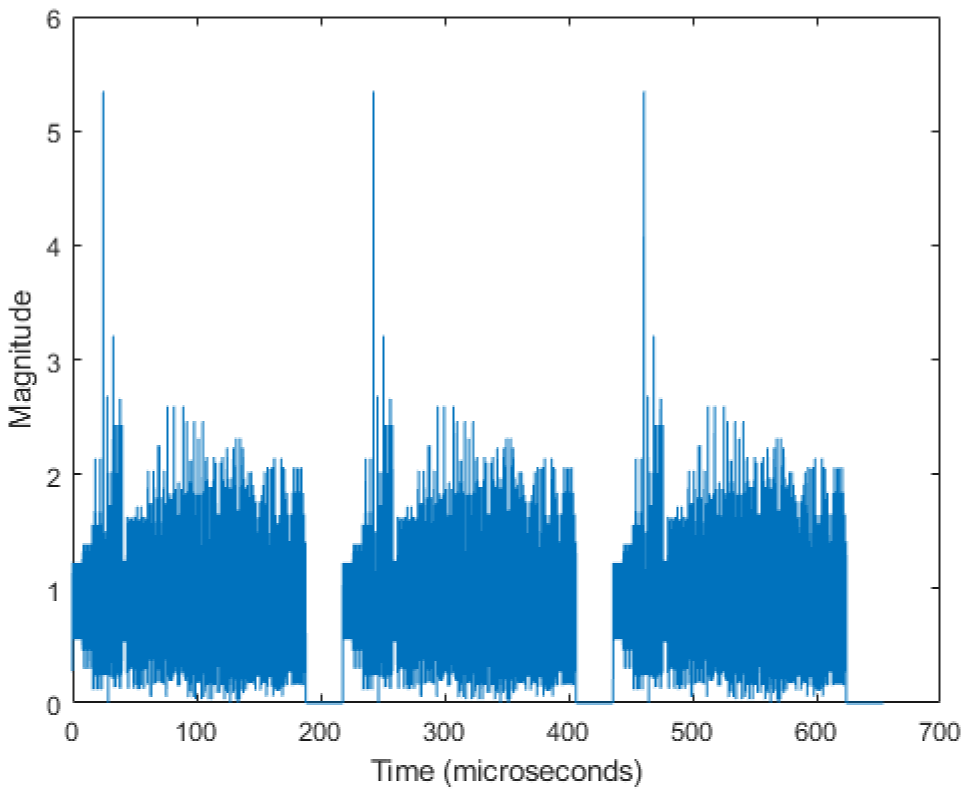
Generate Multiuser HE Format Waveform

Create an HE multiuser (HE MU) configuration object and waveform. Using `Name,Value` pairs, specify 3 packets and 30 microseconds of idle time. Display the configuration object and inspect its properties and settings.

```
cfgHEMU = wlanHEMUConfig(192);
bits = [1;0;0;1;1];
hemuWaveform = wlanWaveformGenerator(bits,cfgHEMU, ...
    'NumPackets',3,'IdleTime',30e-6);
```

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

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



The plot shows three multiuser HE format packets, with each packet separated by 30 microseconds of idle time.

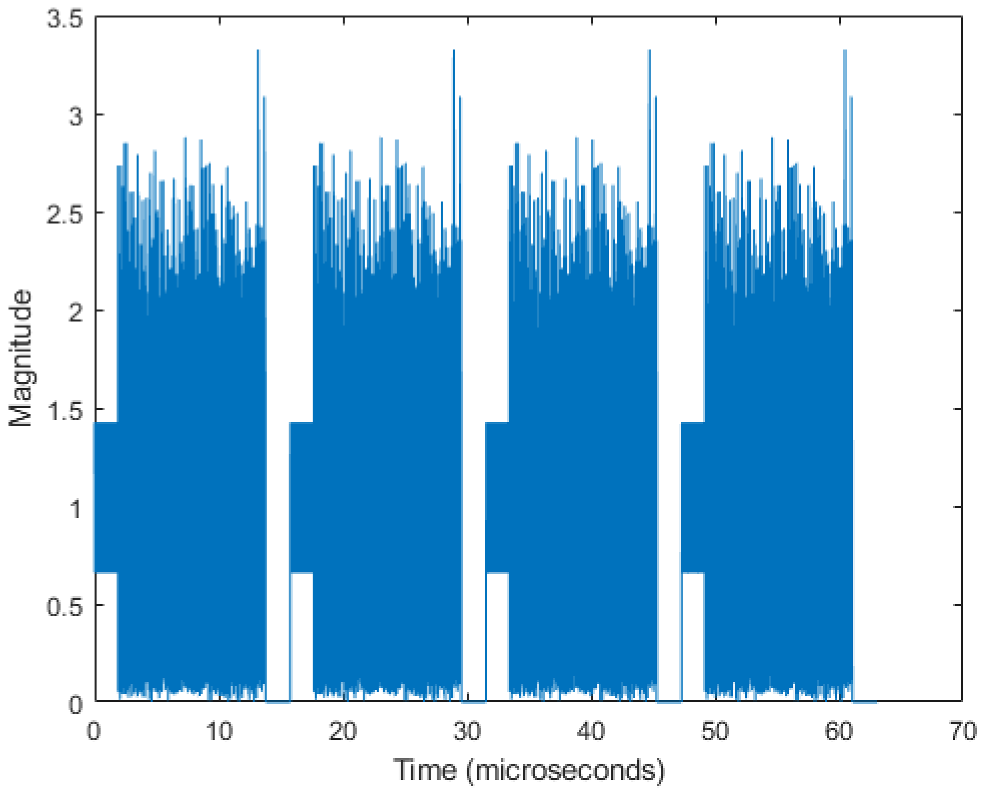
Generate DMG Format Waveform

Create a DMG configuration object and waveform. Using `Name,Value` pairs, assign 13 for the MCS which specifies an OFDM waveform, 4 packets, and 2 microseconds of idle time. Display the configuration object and inspect its properties and settings.

```
cfgDMG = wlanDMGConfig('MCS',13);
bits = [1;0;0;1;1];
dmgWaveform = wlanWaveformGenerator(bits,cfgDMG, ...
    'NumPackets',4,'IdleTime',2e-6);
```

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

```
fs = 2640e6; % Set sampling frequency equal to the channel bandwidth
time = ((0:length(dmgWaveform)-1)/fs)*1e6;
plot(time,abs(dmgWaveform))
xlabel ('Time (microseconds)');
ylabel('Magnitude');
```



The plot shows four DMG format packets, with each packet separated by 2 microseconds of idle time.

Generate S1G Format Waveform

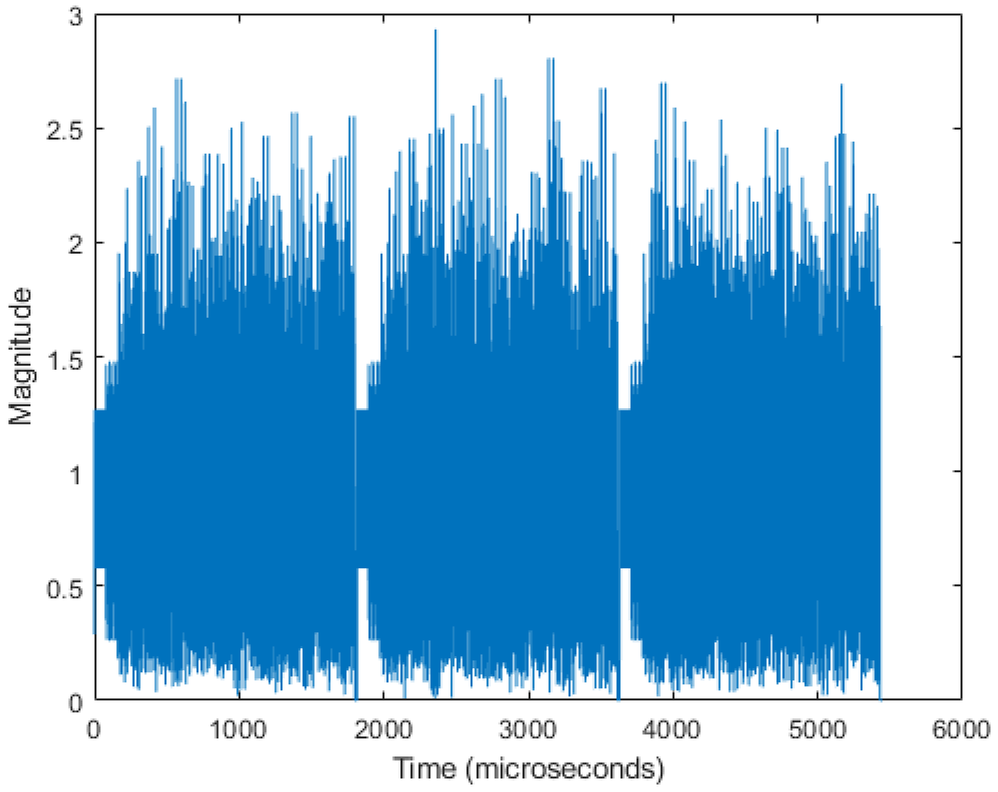
Create a sub-1-GHz (S1G) configuration object and waveform. Using `Name,Value` pairs, specify 4 MHz channel bandwidth, 3 packets, and 15 microseconds of idle time. Display the configuration object and inspect its properties and settings.

```
cfgS1G = wlanS1GConfig('ChannelBandwidth','CBW4');
bits = [1;0;0;1;1];

s1gWaveform = wlanWaveformGenerator(bits,cfgS1G, ...
    'NumPackets',3,'IdleTime',15e-6);
```

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.

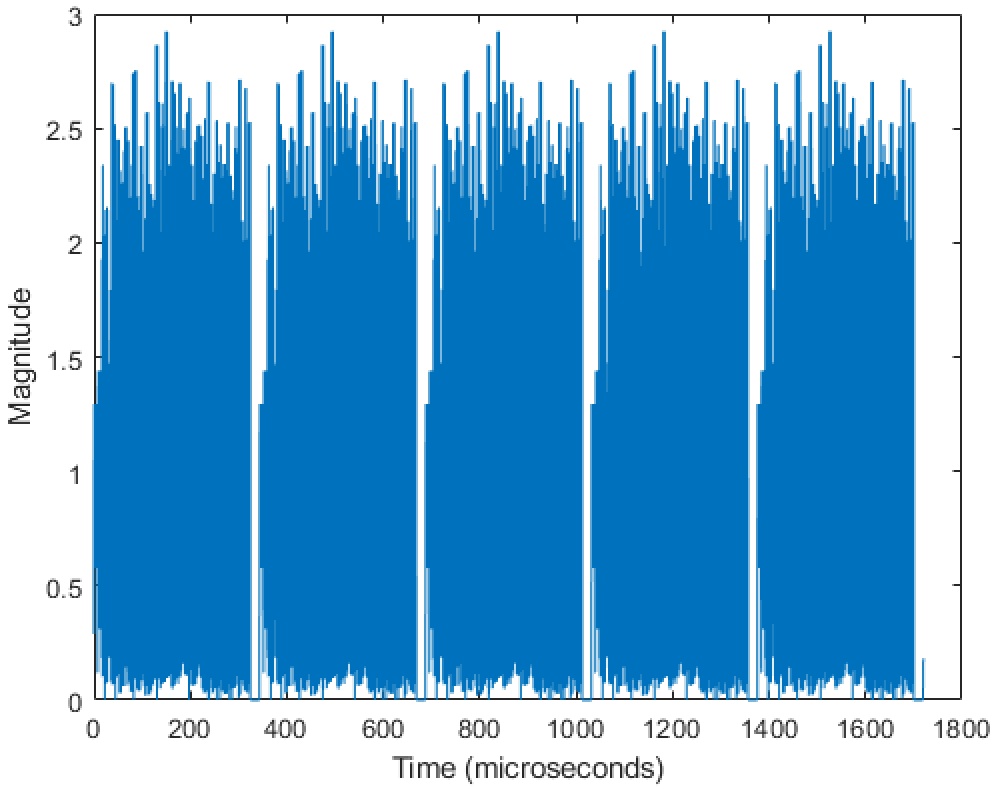
Generate VHT Format Waveform

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.

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

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.

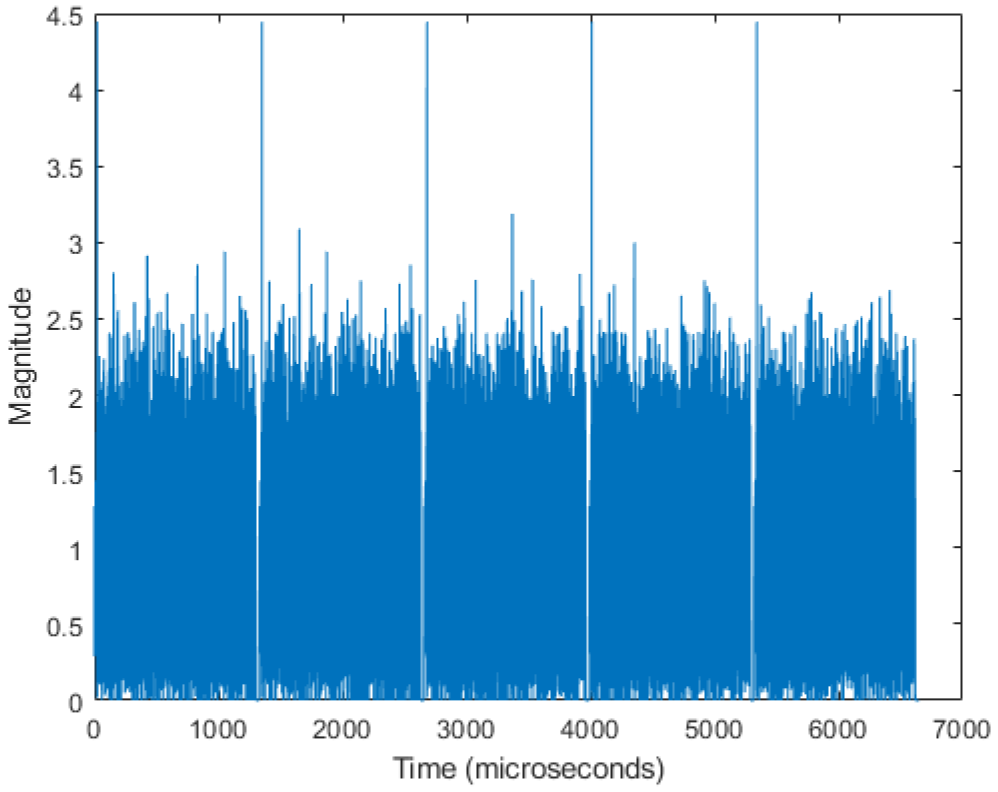
Generate HT Format Waveform

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.

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

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.

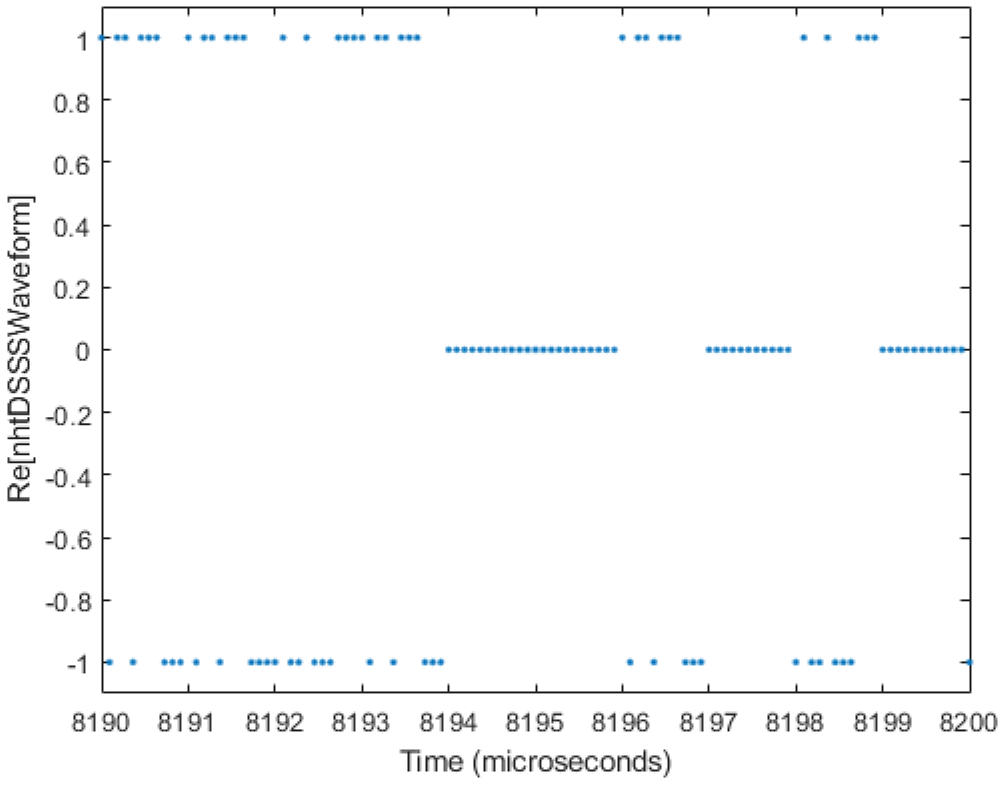
Generate Non-HT Format DSSS Waveform

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.

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

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.

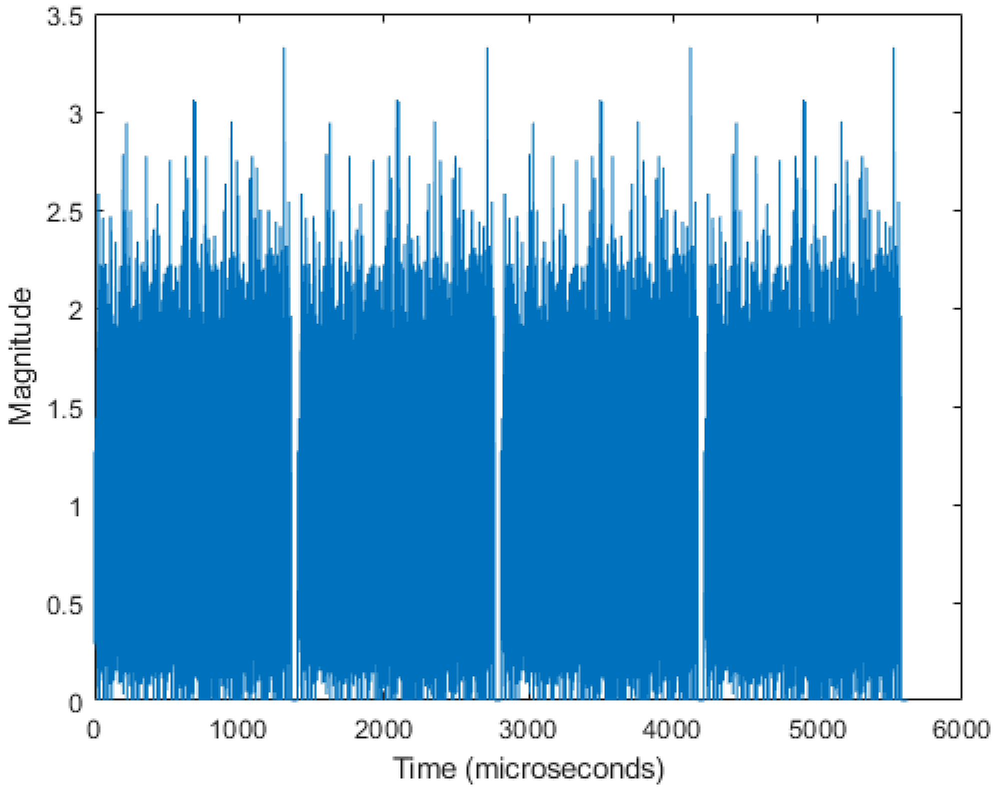
Generate Non-HT Format OFDM Waveform

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.

```
cfgNonHT = wlanNonHTConfig;
bits = [1;0;0;1;1];
nhtWaveform = wlanWaveformGenerator(bits,cfgNonHT, ...
    'NumPackets',4,'IdleTime',45e-6);
```

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`
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` | `wlanNonHTConfig` | `wlanVHTConfig`

More About

- “Create Configuration Objects” on page 2-2
- “WLAN Channel Models” on page 2-28
- “What Is WLAN?” on page 3-2

Generate and Parse WLAN MAC Frames

This example shows how to configure and generate WLAN MAC frames, then recover the payload of MSDUs by parsing the MAC frame.

Introduction

The IEEE® 802.11™ family of standards supports four types of MAC frame: control, data, management, and extension. Within each of these types, the standard defines a range of subtypes, each of which serves a specific purpose in an 802.11™ network.

This example demonstrates how to configure, generate, and parse MPDUs and A-MPDUs by using WLAN Toolbox™ configuration objects and functions.

Generate and Decode MPDU

Create a MAC frame configuration object for a Data frame, specifying a high-efficiency single-user (HE SU) physical layer (PHY) configuration.

```
cfgMPDU = wlanMACFrameConfig('FrameType','Data','FrameFormat','HE-SU');
```

Specify an MSDU as a numeric vector of octets in bit format. You can also specify MSDUs as a character vector or string of octets in hexadecimal format.

```
msdu = randi([0 255],32,1);
```

Generate the MPDU by calling the `wlanMACFrame` function, specifying bits as the output format.

```
[mpdu,mpduLength] = wlanMACFrame(msdu,cfgMPDU,'OutputFormat','bits');
```

Recover the MSDU by calling the `wlanMPDUDecode` function. The function also returns the MAC frame configuration object and the status of the decoding. Check that the decoding operation returns the correct frame format and display the status.

```
[rxCfgMPDU,payload,status] = wlanMPDUDecode(mpdu,wlanHESUConfig);
disp(isequal(cfgMPDU.FrameFormat,rxCfgMPDU.FrameFormat))
```

```
   1
```

```
disp(status)
```

```
    Success
```

Generate and Parse A-MPDU

Create a configuration object for a QoS Data MAC frame, specifying an HE SU PHY configuration. Enable MPDU aggregation and disable MSDU aggregation.

```
cfgAMPDU = wlanMACFrameConfig('FrameType','QoS Data','FrameFormat','HE-SU',...
    'MPDUAggregation',true,'MSDUAggregation',false);
```

Specify a cell array of MSDUs, specifying each MSDU as a numeric vector of octets in bit format. You can also specify MSDUs as a character vector or string of octets in hexadecimal format.

```
msduList = repmat({randi([0 255],32,1)},1,4);
```

Generate the MPDU for a HE SU PHY configuration by calling the `wlanMACFrame` function.

```
cfgPHY = wlanHESUConfig('MCS',5);
[ampdu,ampduLength] = wlanMACFrame(msduList,cfgAMPDU,cfgPHY,'OutputFormat','bits');
```

Deaggregate the A-MPDU to return the MPDU list by calling the `wlanAMPDUDeaggregate` function. The function also returns the result of the delimiter cyclic redundancy check (CRC) and the status of A-MPDU deaggregation.

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

Display the number of delimiter CRC failures and the status of deaggregation.

```
disp(nnz(delimiterCRCFailure))
```

```
     0
```

```
disp(status)
```

```
    Success
```

Obtain the MSDUs by decoding the deaggregated MPDUs with the `wlanMPDUDecode` function and display the status of the decoding process.

```
if strcmp(status,'Success')
    for i = 1:numel(mpduList)
        if ~delimiterCRCFailure(i)
            [cfg,msdu,decodeStatus] = wlanMPDUDecode(mpduList{i},cfgPHY,'DataFormat','octets');
            disp(['MPDU ' num2str(i) ' decoding status: ' char(decodeStatus)])
        end
    end
end
```

```
MPDU 1 decoding status: Success
MPDU 2 decoding status: Success
MPDU 3 decoding status: Success
MPDU 4 decoding status: Success
```

See Also

More About

- “802.11ac Waveform Generation with MAC Frames”
- “802.11 MAC Frame Generation”
- “802.11 MAC Frame Decoding”

WLAN Channel Models

This example demonstrates passing WLAN S1G, 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 TGah,TGn and TGac models from WLAN Toolbox™ and the AWGN and 802.11g models from Communications 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 of this example, you:

- Create a waveform.
- Transmit it through a fading channel with noise added.
- Use a spectrum analyzer to display the waveform before and after it passes through the noisy fading channel.

Pass S1G Waveform Through TGah SISO Channel

Create a bit stream to use when generating the WLAN S1G format waveform.

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

Create a S1G configuration object, and generate an 2 MHz S1G waveform. Calculate the signal power.

```
s1g = wlanS1GConfig;
preChS1G = wlanWaveformGenerator(bits,s1g);
```

Pass the signal through a TGah 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 TGah channel object. Set the channel model sampling frequency and channel bandwidth, enable path loss and shadowing, and use the Model-D delay profile.

```
cbw = s1g.ChannelBandwidth;
fs = 2e6; % Channel model sampling frequency equals the channel bandwidth
tgahChan = wlanTGahChannel('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 TGah large scale fading pathloss.

```
preChSigPwr_dB = 10*log10(mean(abs(preChS1G)));
sigPwr = 10^((preChSigPwr_dB-tgahChan.info.Pathloss)/10);

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

Pass the S1G waveform through a SISO TGah channel and add the AWGN channel noise.

```
postChS1G = chNoise(tgahChan(preChS1G));
```

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

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

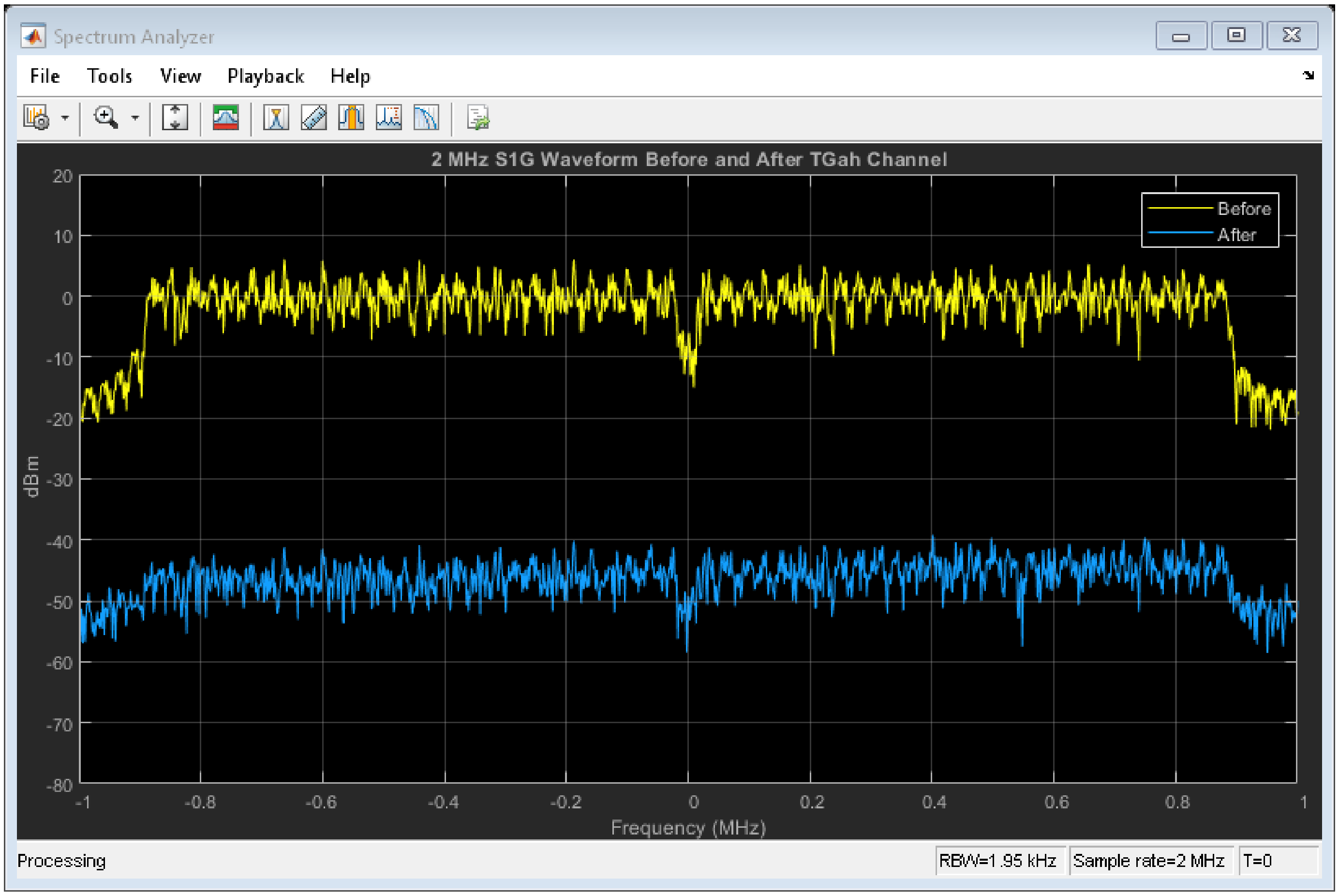
Pass the S1G waveform through the receiver. Choose an appropriate noise variance, nVar, to set the receiver noise level. Here, the receiver noise level is based on the noise variance for a receiver with a 9 dB noise figure. `nVar` = *kTBF*, where *k* is Boltzmann's constant, *T* is the ambient temperature 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);
```

```
rxS1G = rxNoise(postChS1G,nVar);
```

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

```
title = '2 MHz S1G Waveform Before and After TGah Channel';
saScope = dsp.SpectrumAnalyzer('SampleRate',fs,'ShowLegend',true,...
    'SpectralAverages',10,'Title',title,'ChannelNames',{'Before','After'});
saScope([preChS1G,rxS1G])
```



Path loss accounts for the roughly 50 dB of separation between the waveform before and after it passes through the TGah 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 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 SISO 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');
```

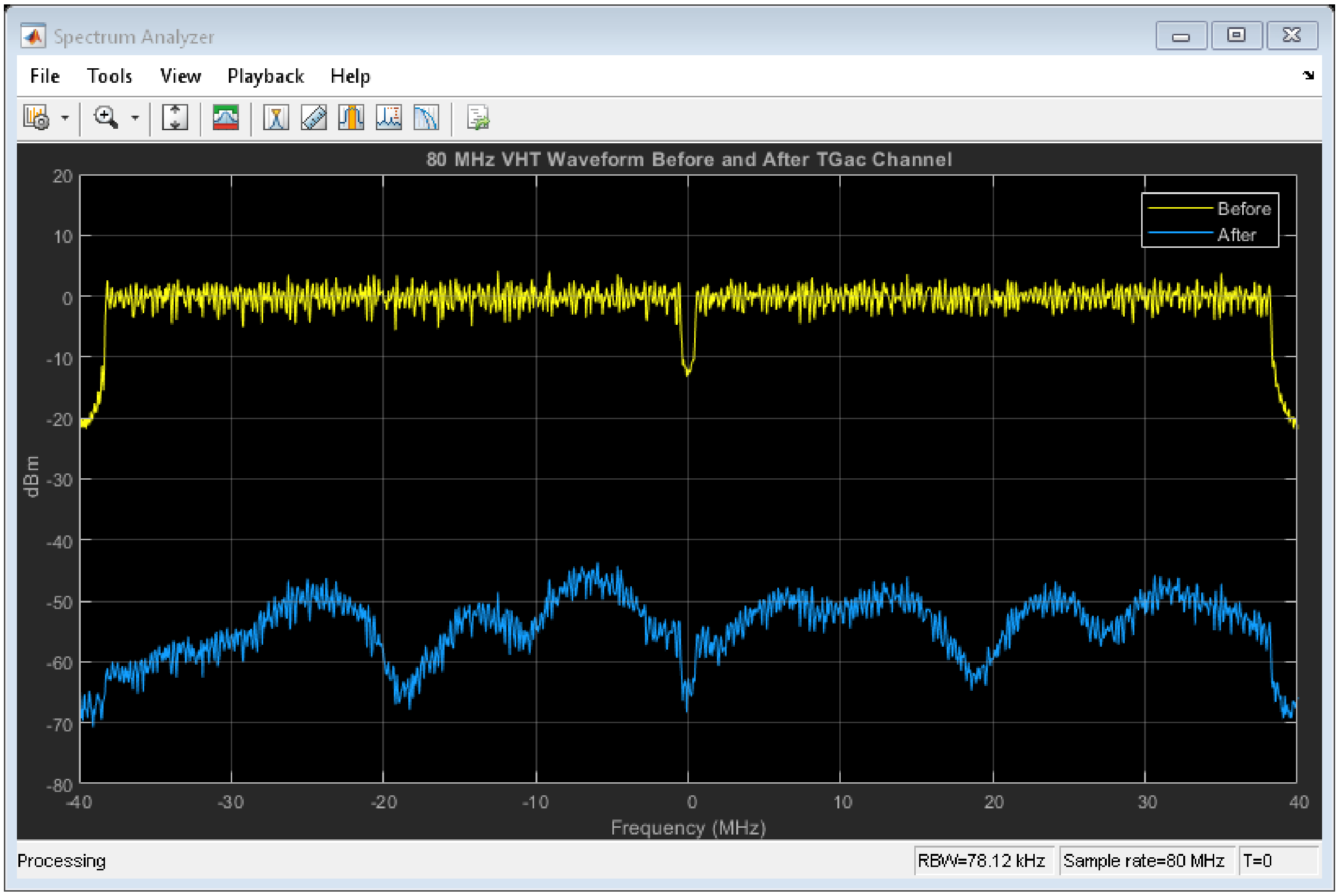
Pass the VHT waveform through the receiver. Choose an appropriate noise variance, nVar, to set the receiver noise level. Here, the receiver noise level is based on the noise variance for a receiver with a 9 dB noise figure. `nVar` = *kTBF*, where *k* is Boltzmann's constant, *T* is the ambient temperature 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');
```

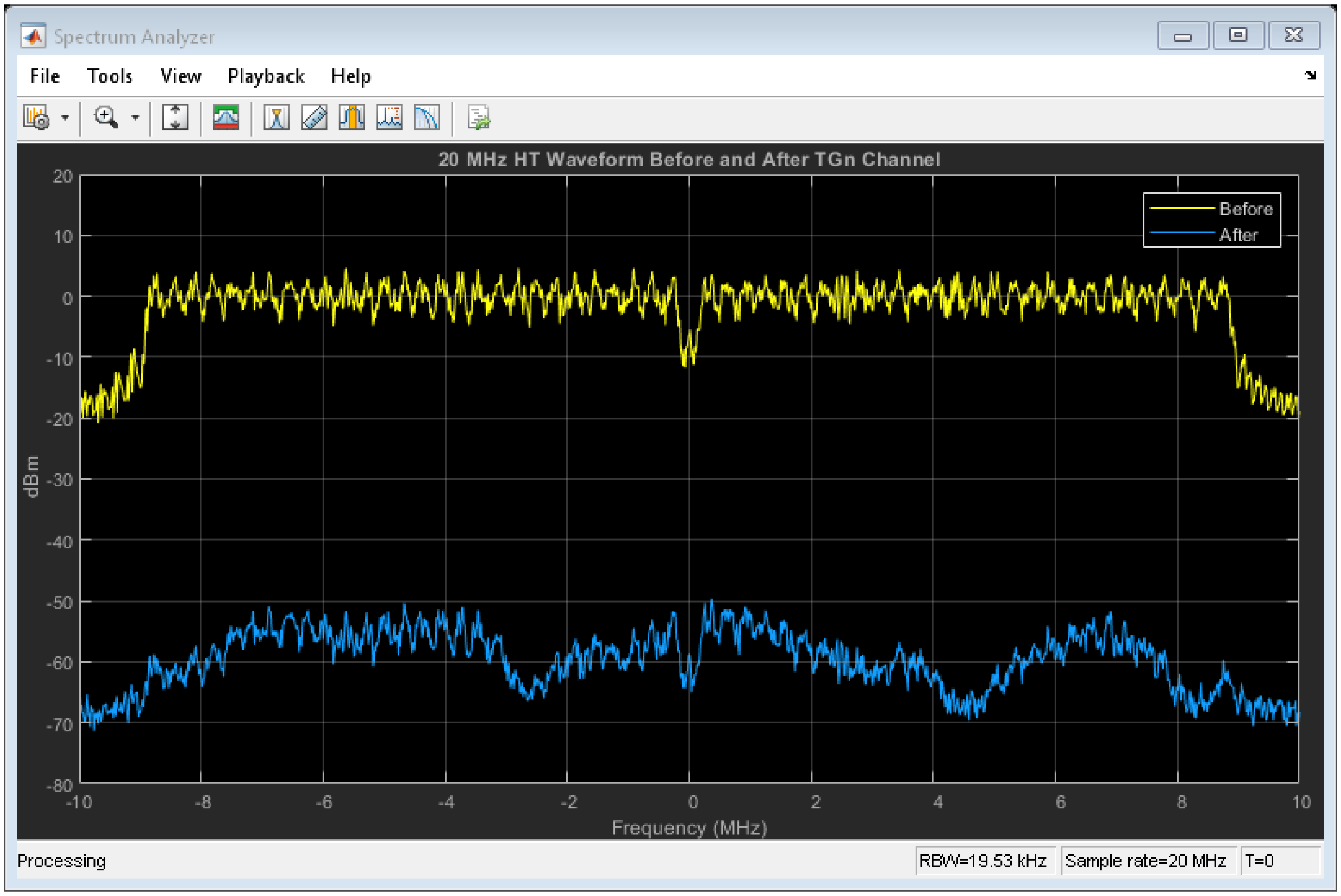
Pass the HT waveform through the receiver. Choose an appropriate noise variance, nVar, for setting the receiver noise level. Here, the receiver noise is based on the noise variance for a receiver with a 9 dB noise figure. `nVar` = $kTBF$, where k is Boltzmann's constant, T is the ambient temperature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

```
maxDoppShift = 3;
trms = 2/fs;
ch802 = comm.RayleighChannel('SampleRate',fs,'MaximumDopplerShift',maxDoppShift,'PathDelays',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(ch802(preChNonHT),10,'measured');
```

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

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

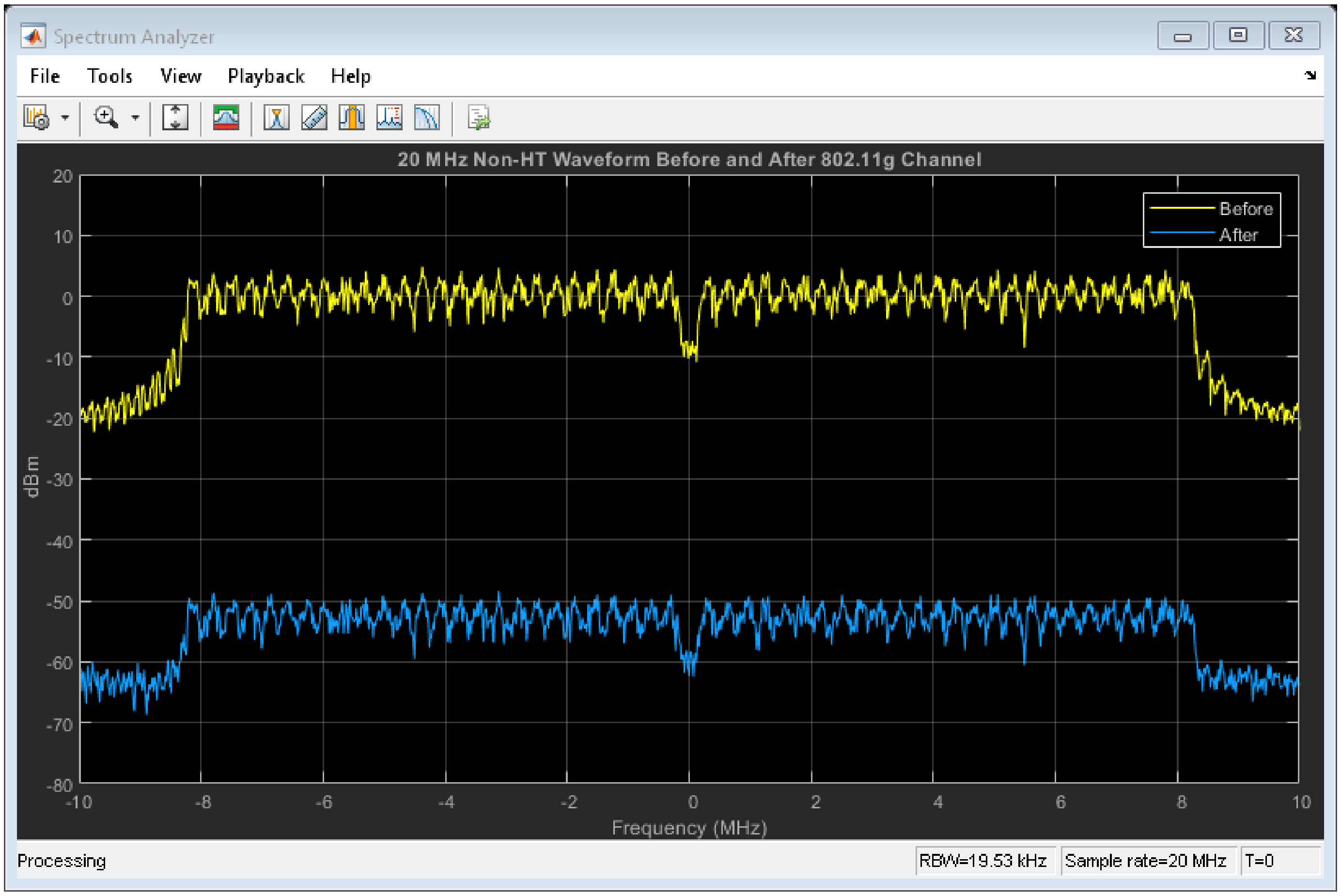
Pass the non-HT waveform through the receiver. Choose an appropriate noise variance, `nVar`, for setting the receiver noise level. Here, the receiver noise is based on the noise variance for a receiver with a 9 dB noise figure. `nVar` = $kTBF$, where k is Boltzmann's constant, T is the ambient temperature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

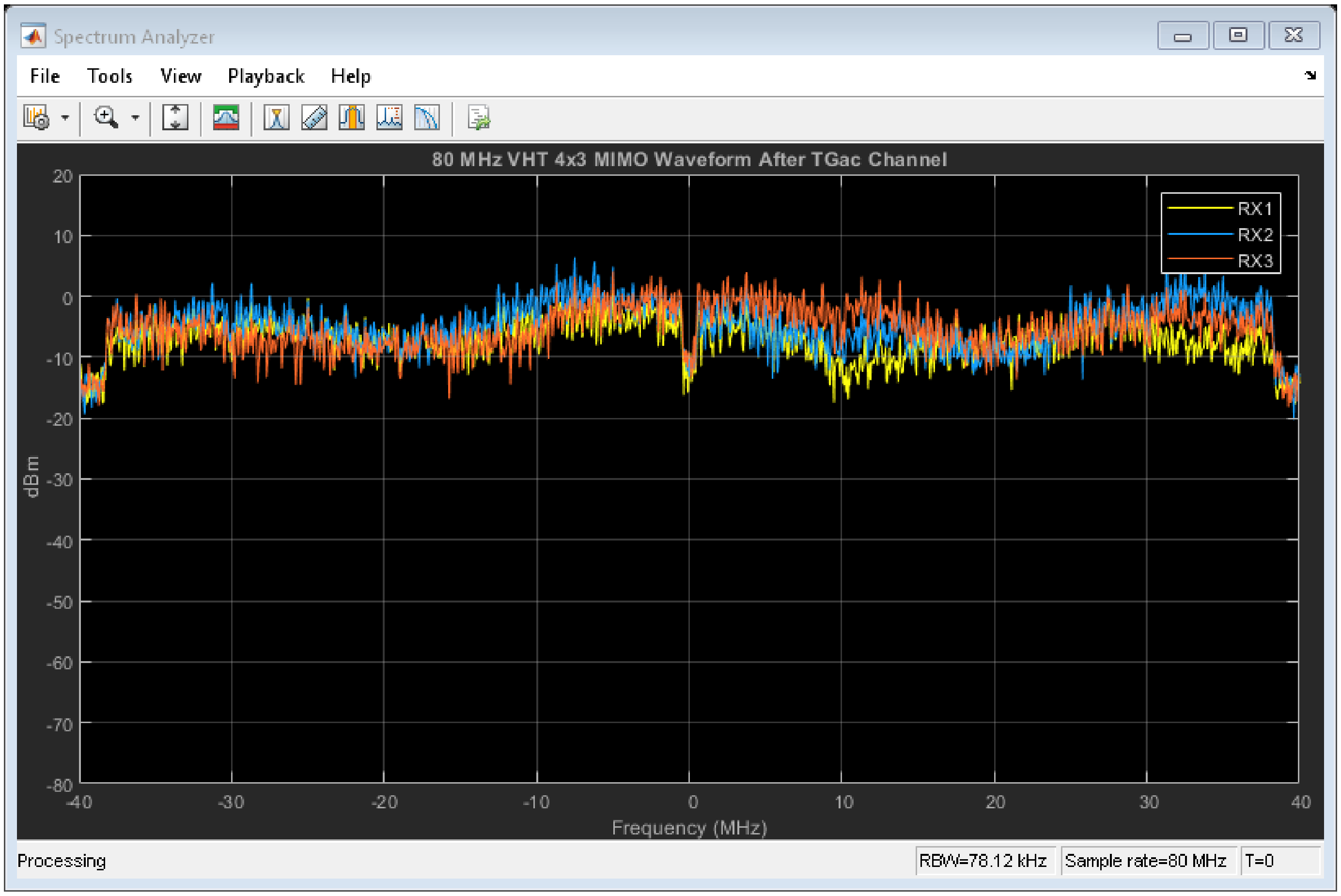
Pass the multi-user VHT waveform through a noisy TGac channel. Choose an appropriate noise variance, nVar, for setting the AWGN level. Here, the AWGN level is based on the noise variance for a receiver with a 9 dB noise figure. `nVar` = *kTBF*, where *k* is Boltzmann's constant, *T* is the ambient temperature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

`wlanHTConfig` | `wlanNonHTConfig` | `wlanTGacChannel` | `wlanTGnChannel` | `wlanVHTConfig`

Related Examples

- “Waveform Generation” on page 2-14
- “Packet Recovery” on page 2-38
- “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 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`. Create a transmission bit stream for the data field. For a VHT waveform, the data field contains `PSDULength*8` bits.

```
cfgVHT = wlanVHTConfig('APEPLength',3200,'MCS',5);
txBits = randi([0 1],cfgVHT.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(cfgVHT);
lltf = wlanLLTF(cfgVHT);
lsig = wlanLSIG(cfgVHT);
vhtSigA = wlanVHTSIGA(cfgVHT);
vhtstf = wlanVHTSTF(cfgVHT);
vhtltf = wlanVHTLTF(cfgVHT);
vhtSigB = wlanVHTSIGB(cfgVHT);
vhtData = wlanVHTData(txBits,cfgVHT);
```

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 = cfgVHT.ChannelBandwidth;
fs = 80e6;
tgac = 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)
```

```
noiseVar = 2.5438e-12
```

```
rxPPDU = awgnChan(tgac(txPPDU),noiseVar);
```

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(cfgVHT)
```

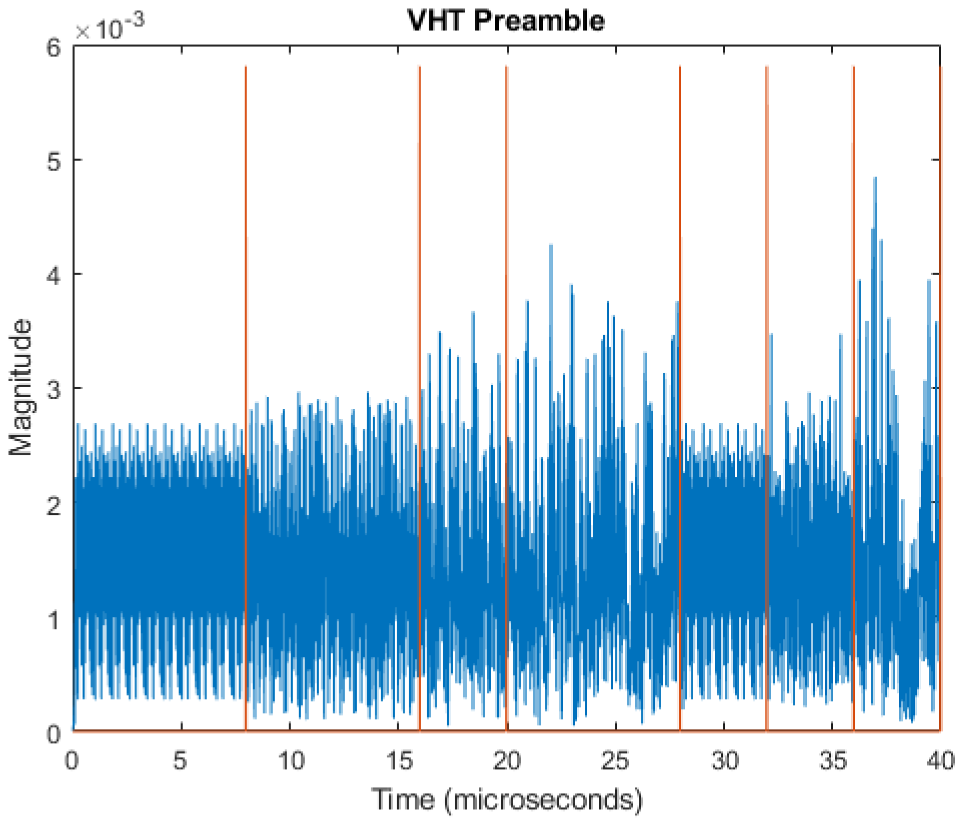
```
fieldInd = struct with fields:
       LSTF: [1 640]
       LLTF: [641 1280]
       LSIG: [1281 1600]
    VHTSIGA: [1601 2240]
     VHTSTF: [2241 2560]
     VHTLTF: [2561 2880]
    VHTSIGB: [2881 3200]
    VHTData: [3201 12160]
```

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 Preamble')
```



Demodulate the L-LTF and estimate the channel.

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

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 = 1x4 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))
```

```
recMCS = 5
```

```
isequal(recMCS,cfgVHT.MCS)
```

```
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,cfgVHT);
chEstVHTLTF = wlanVHTLTFChannelEstimate(demodVHTLTF,cfgVHT);
```

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
```

```
sigbAPEPlength = 3200
```

```
isequal(sigbAPEPlength,cfgVHT.APEPLength)
```

```
ans = logical
   1
```

The recovered value matches the configured APEP Length.

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

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

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 an HT-format waveform.

Generate 20 MHz HT Waveform

Create an HT configuration object and transmission PSDU. Set `MCS` to `2`. For an HT waveform, the data field is `PSDULength*8` bits.

```
cfgHT = wlanHTConfig('MCS',2);
txPSDU = randi([0 1],cfgHT.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(cfgHT);
lltf = wlanLLTF(cfgHT);
lsig = wlanLSIG(cfgHT);
htsig = wlanHTSIG(cfgHT);
htstf = wlanHTSTF(cfgHT);
htltf = wlanHTLTF(cfgHT);
htData = wlanHTData(txPSDU,cfgHT);
```

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 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 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(cfgHT)
```

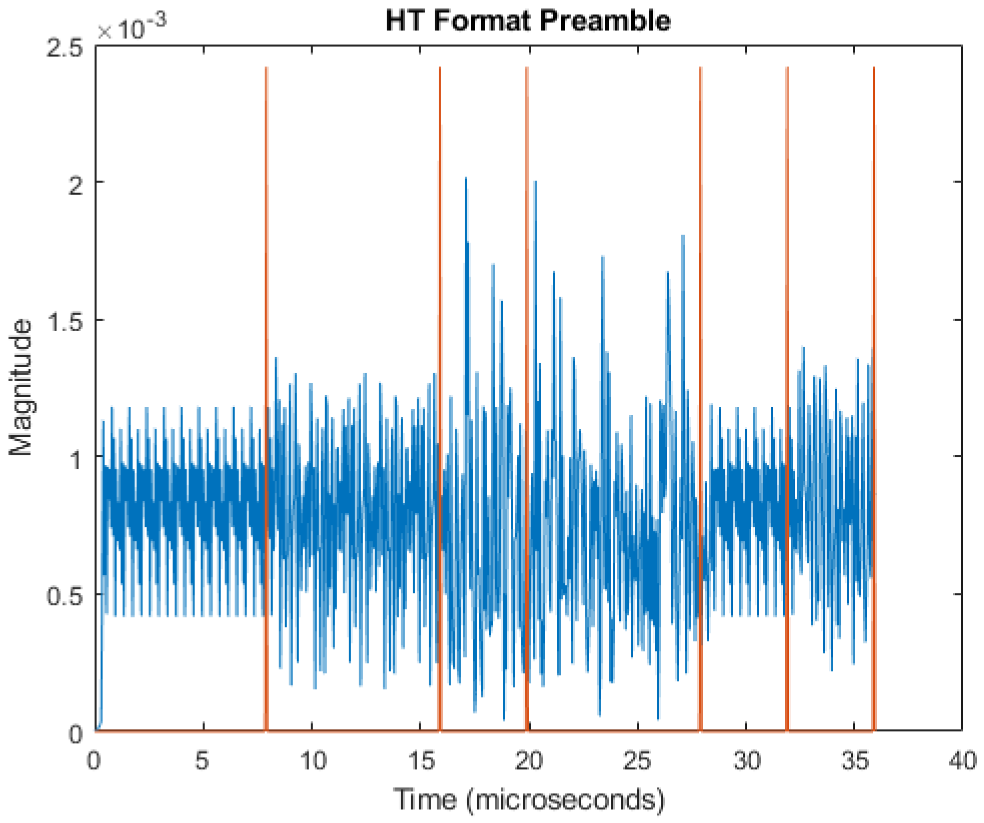
```
fieldInd = struct with fields:
      LSTF: [1 160]
      LLTF: [161 320]
      LSIG: [321 400]
     HTSIG: [401 560]
     HTSTF: [561 640]
     HTLTF: [641 720]
    HTData: [721 9200]
```

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,cfgHT);
chEstLLTF = wlanLLTFChannelEstimate(demodLLTF,cfgHT);
```

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,cfgHT.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 = 1x4 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,cfgHT.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))
```

```
recMCS = 2
```

```
isequal(recMCS,cfgHT.MCS)
```

```
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,cfgHT);
chEstHTLTF = wlanHTLTFChannelEstimate(demodHTLTF,cfgHT);
```

Recover HT-Data Contents from PPDU

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

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

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.For a non-HT waveform, the data field is `PSDULength*8` bits.

```
cfgNonHT = wlanNonHTConfig('MCS',4);
txPSDU = randi([0 1],cfgNonHT.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(cfgNonHT);
lltf = wlanLLTF(cfgNonHT);
lsig = wlanLSIG(cfgNonHT);
nhtData = wlanNonHTData(txPSDU,cfgNonHT);
```

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;
maxDoppShift = 3;
ch802 = comm.RayleighChannel('SampleRate',fs,'MaximumDopplerShift',maxDoppShift,'PathDelays',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(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(cfgNonHT)
```

```
fieldInd = struct with fields:
         LSTF: [1 160]
         LLTF: [161 320]
         LSIG: [321 400]
    NonHTData: [401 7120]
```

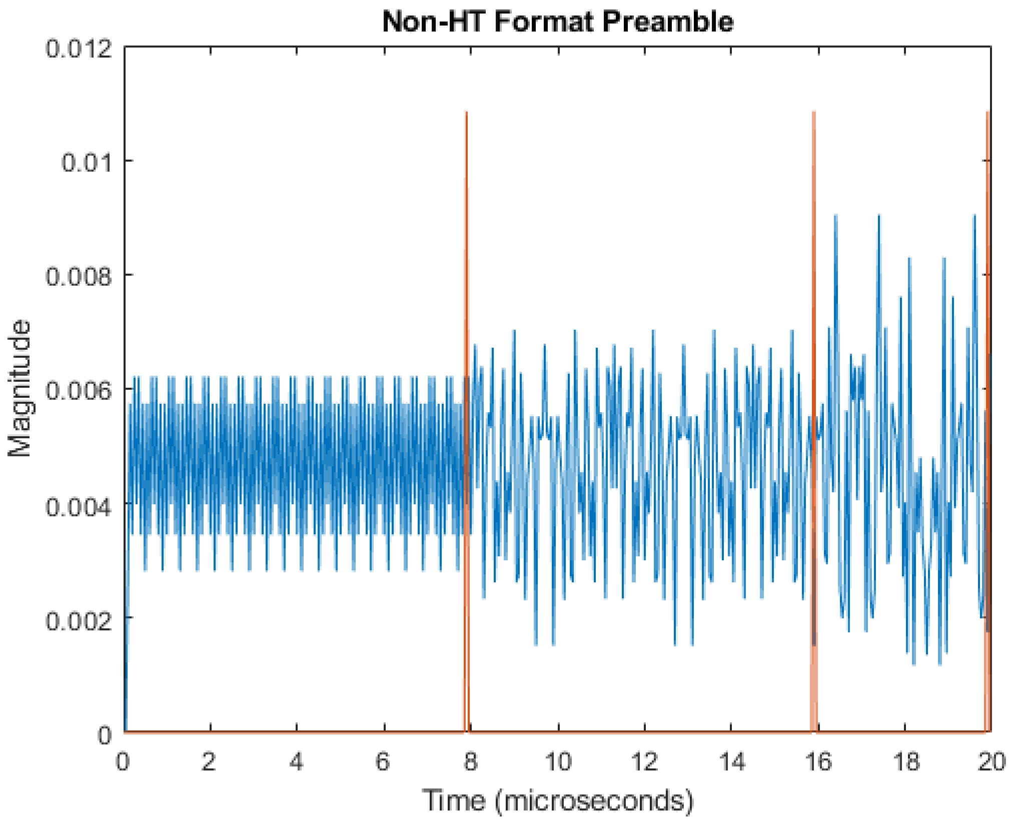
The stop index of the L-SIG field indicates the preamble length in samples.

```
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,cfgNonHT);
chEstLLTF = wlanLLTFChannelEstimate(demodLLTF,cfgNonHT);
```

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 = 1x4 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,cfgNonHT);
chEstLLTF = wlanLLTFChannelEstimate(demodLLTF,cfgNonHT);
```

Recover Non-HT-Data Contents from PPDU

Recover equalized symbols using channel estimates from HT-LTF, specifying a zero-forcing equalization method.

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

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` | `wlanNonHTConfig` | `wlanVHTConfig`

Related Examples

- “WLAN Channel Models” on page 2-28
- “What Is WLAN?” on page 3-2
- “Build VHT PPDU”
- “Build HT PPDU”
- “Build Non-HT PPDU”

About WLAN

What Is WLAN?

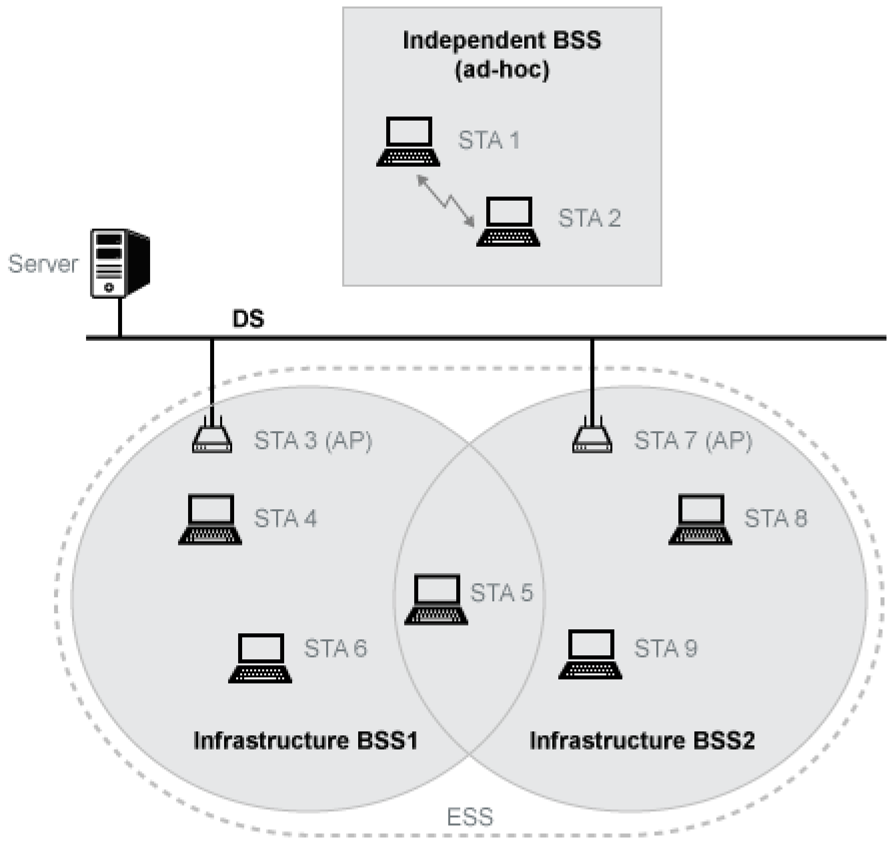
In this section...

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 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. With WLAN Toolbox, you can model IEEE 802.11 standardized implementations of the WLAN physical (PHY) and medium access control (MAC) layers. You can also explore variations on implementations for future evolution of the standard.

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).

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



- 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

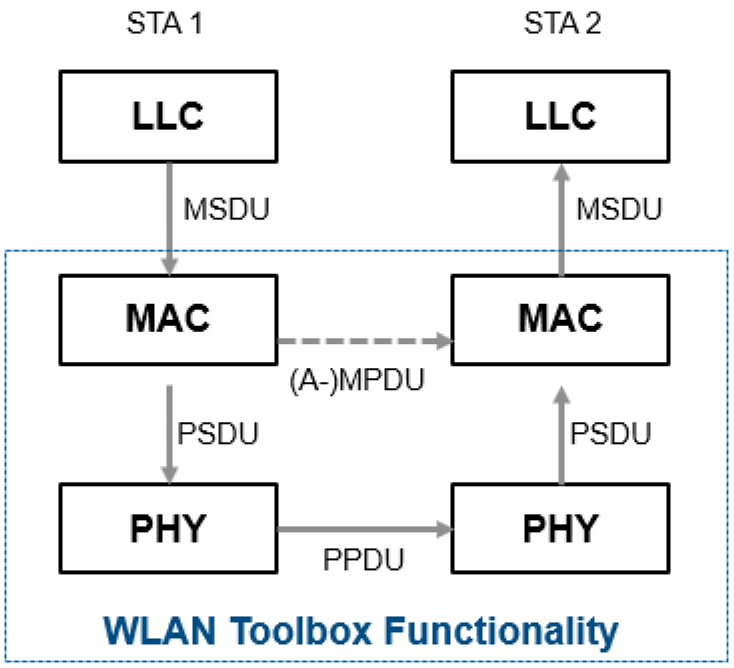
The interworking reference model shown here includes a subset of the network components associated with the data link layer (DLL) and physical layer (PHY). Section 4.9.2 of [2] describes the interworking reference model for 802.11. The medium access control (MAC) is a sublayer of the DLL.

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

WLAN Message Exchange

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

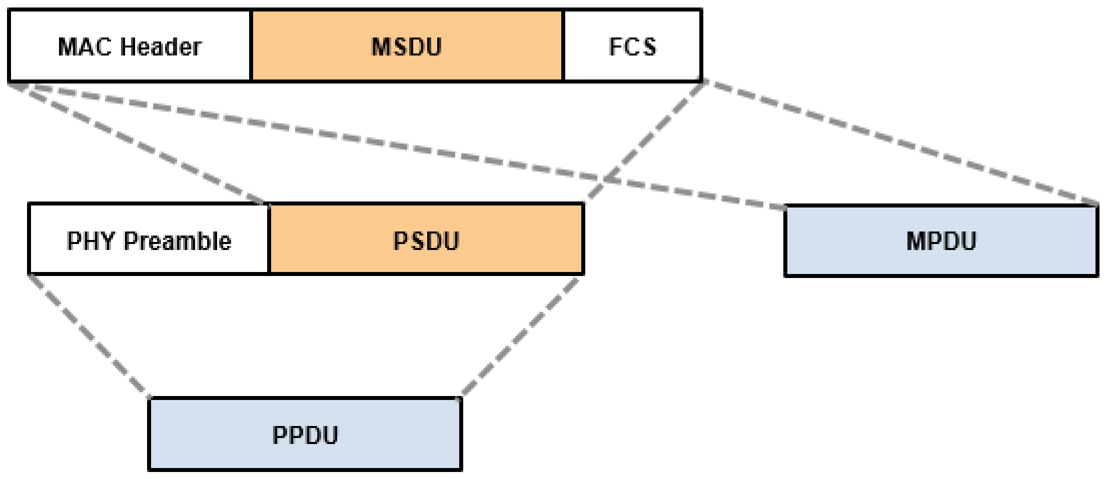
- Data and control information exchanged between peer STA layers are protocol information transfers. See (A-)MPDU and PPDU in the figure.
- Data and control information exchanged between layers within an STA are service information transfers. See MSDU and PSDU in the figure.



WLAN Toolbox functionality focuses on MAC and PHY implementations. Specifically, the toolbox models the exchange of PPDUs between PHY peers, and the exchange of MPDUs or A-MPDUs between MAC peers. Messages exchanged between protocol stack layers are briefly described here. For more information on these messages, see [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
MPDU or A-MPDU — MAC protocol data unit or aggregated 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 megabit per second (Mbps) to nearly 7 gigabit per second (Gbps).

WLAN Toolbox provides native support for the various 802.11 standard versions listed here. The toolbox focuses on the PHY and MAC layers, 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	4.9 and 5	10, 20	27	SISO	non-HT

Standard	Release Year	Modulation	Base Frequency (GHz)	Bandwidth (MHz)	Maximum Throughput (Mbps)	Antenna Scheme	PPDU Format
802.11n™ (Wi-Fi 4)	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
802.11ad™	2012	SC/OFDM	60 GHz	1760 (SC), 2640 (OFDM)	< 7000	MIMO single stream with beamforming	DMG
802.11ac™ (Wi-Fi 5)	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
802.11ax™ (Wi-Fi 6)	2020 (anticipated)	OFDMA	2.4 and 5	20, 40, 80, 160, 80+80	< 10,000	UL and DL MU-MIMO up to eight streams	HE

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.

The 802.11a amendment 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 Federal Communications Commission (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 802.11ad amendment specifies operation in the 60-GHz band.

The 802.11ah amendment uses sub-1-GHz frequencies (unlicensed 900-MHz bands) to provide extended range, and has low energy consumption to support the concepts involving the Internet of Things (IoT).

The 802.11ax amendment introduces orthogonal frequency-division multiple access (OFDMA) to improve overall spectral efficiency, and higher-order 1024-point quadrature amplitude modulation

(1024-QAM) support for increased throughput. 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.

References

[1] IEEE P802.11ax/D4.1. “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” Draft Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

[2] IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

[3] IEEE Std 802.11ah-2016 (Amendment to IEEE Std 802.11-2016 as amended by IEEE Std 802.11ai™-2016). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 2: Sub 1 GHz License Exempt Operation.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

[4] IEEE STD 802.11ac-2013 (Amendment to IEEE Std 802.11-2012, as amended by IEEE Std 802.11ae™-2012, IEEE Std 802.11a™-2012, and IEEE Std 802.11ad-2012). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 4: Enhancements for Very High Throughput Operation in Bands below 6 GHz.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

[5] IEEE STD 802.11ad-2012 (Amendment to IEEE Std 802.11-2012, as amended by IEEE Std 802.11ae™-2012 and IEEE Std 802.11a™-2012). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 4: Enhancements for Very High Throughput Operation in Bands below 6 GHz.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

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

See Also

Related Examples

- “Create Configuration Objects” on page 2-2
- “Waveform Generation” on page 2-14
- “WLAN Channel Models” on page 2-28
- “Packet Recovery” on page 2-38
- “WLAN PPDU Structure”

External Websites

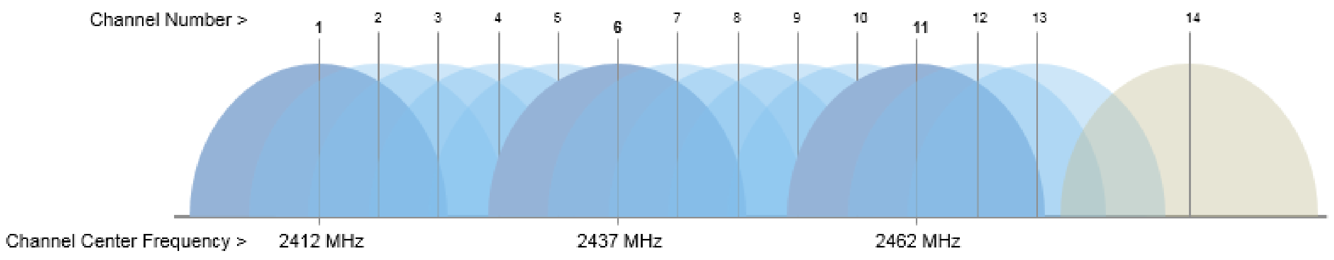
- https://standards.ieee.org/

WLAN Radio Frequency Channels

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

Refer to IEEE Std 802.11-2016, 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 = $F_{\text{START}} + (5\times \textit{Channel Number})$

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

F_{CENTER} in MHz = $2407 + (5\times6) = 2437$ MHz.

802.11 channels		
Channel Number	**F_{START}, Starting Frequency**	**Comments**
1, ..., 13	2407 MHz	For country- and release-specific restrictions, refer to [1]
14	2414 MHz	
132, 133, 134, 136, 137, 138	3000 MHz	
131, ..., 138	3002.5 MHz	
183, ..., 197	4000 MHz	
182, ..., 189	4002.5 MHz	
21, 25	4850 MHz	
11, 13, 15, 17, 19	4890 MHz	
1, ..., 10	4937.5 MHz	

802.11 channels		
Channel Number	**F_{START}, Starting Frequency**	**Comments**
7, ..., 12, 16 34, ..., 60 in increments of 2 64 100, 104, 106, 108 112, 114, 116 120, 122, 124, 128 132, 136, 138 140, 144, 149 153, 155, 157 161, 165, 169 171, ..., 184 in increments of 1	5000 MHz	
6, ..., 11 170, ..., 184 in increments of 1	5002.5 MHz	
1, 2, 3, 4	56.16 GHz	

References

[1] IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” IEEE Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

[2] IEEE P802.11ax/D4.1. “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN.” Draft Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

Acknowledgments

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

Source	Copyright Owner
Content from IEEE Std 802.11-2016	Adapted and reprinted with permission from IEEE. Copyright IEEE 2016. All rights reserved.
Content from IEEE Std 802.11ah-2016	Adapted and reprinted with permission from IEEE. Copyright IEEE 2016. All rights reserved.