WLAN Toolbox™

User's Guide

MATLAB®

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™ User's Guide

Trademarks

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

Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

Contents

Signal Reception

4

Propagation Channel Models

5

End-to-End Simulation

6

System-Level Simulation

7

Test and Measurement

8

Code Generation and Deployment

9

Software-Defined Radio

10

PHY Modeling

802.11az Waveform Generation

This example shows how to parameterize and generate IEEE® 802.11az™ high-efficiency (HE) ranging null data packet (NDP) waveforms and highlights some of the key features of the standard.

Introduction

The 802.11az standard [1 on page 1-0   ], commonly referred to as next-generation positioning (NGP), enables a station to identify its position relative to other stations. This standard supports two HE ranging physical layer (PHY) protocol data unit (PPDU) formats:

- HE ranging NDP
- HE trigger-based (TB) ranging NDP

The HE ranging NDP and HE TB ranging NDP are the respective analogues of the HE sounding NDP and HE TB NDP feedback PPDU formats, as defined in the 802.11ax™ standard. For more information on these HE PPDU formats, see [2 on page 1-0   ].

The HE ranging NDP supports the positioning of one or more users with an optional secure HE long training field (HE-LTF) sequence. The single-user HE ranging waveform contains HE-LTF symbols for a single user, which also support an optional secure HE-LTF sequence. The multi-user HE ranging waveform permits only secure HE-LTF symbols for multiple users. Single-user and multi-user waveforms can contain multiple repetitions of the HE-LTF symbols. This feature can help improve distance estimation accuracy.

Because the 802.11az standard uses the same underlying PHY technologies as the 802.11ax standard, the processing chains are very similar. This example shows how to generate 802.11az HE ranging NDP waveforms with secure and nonsecure HE-LTF sequences.

HE Ranging NDP Without Secure HE-LTF

The HE Ranging NDP contains HE-LTF symbols for a single user and uses the regular HE-LTF sequence defined in [2 on page 1-0   ]. The number of HE-LTF symbols is the product of the number of HE-LTF repetitions and the number of HE-LTF symbols per repetition. The number of HE-LTF symbols depends on the number of space-time streams as specified in Table 21-13 of [3 on page 1-0   ]. The construction of HE-LTF symbols in an HE Ranging NDP follows the steps defined in section 27.3.10.10 of [2 on page 1-0   ] for all repeated HE-LTF symbols in an HE-LTF.

Single-User HE Ranging NDP Generation

Configure a transmission with two antennas, two space-time streams, and two HE-LTF repetitions.

```
cfg = heRangingConfig('NumTransmitAntennas',2);
cfg.User{1}.NumSpaceTimeStreams = 2;
cfg.User{1}.NumHELTFRepetition = 2;
```

Generate the HE ranging NDP waveform for the specified configuration.

```
tx = heRangingWaveformGenerator(cfg);
```

Plot the transmission power on the first antenna.

```
heRangingWavGenPlot(tx,cfg);
```

Power of HE Ranging Packet
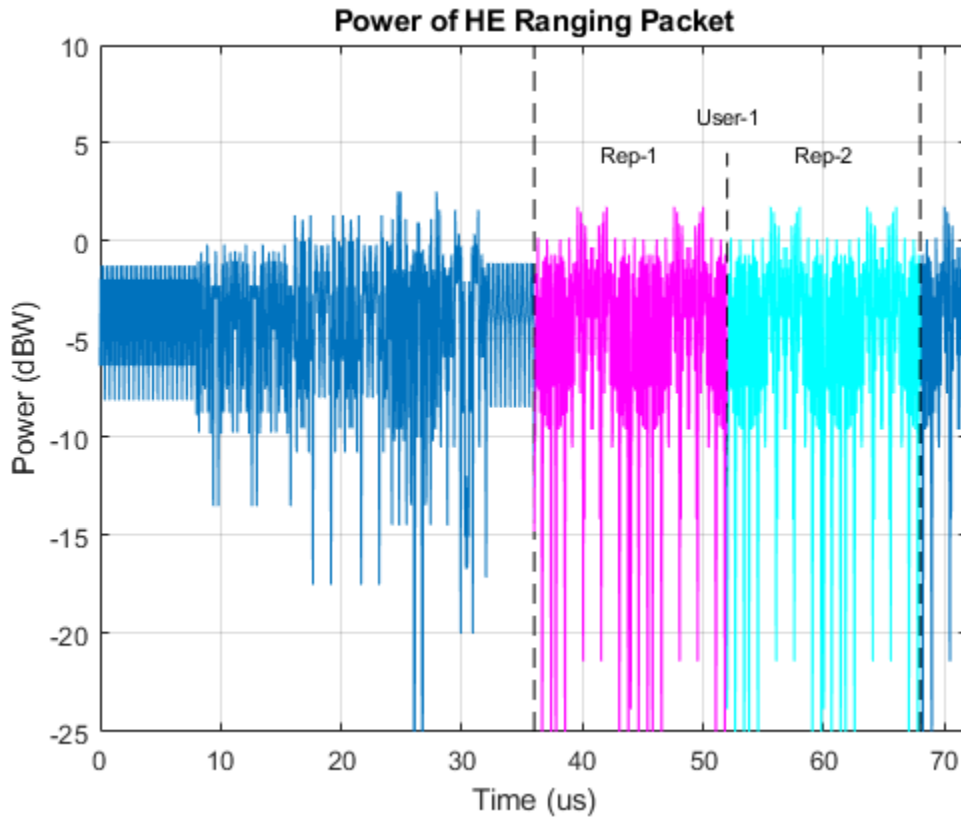
HE Ranging NDP With Secure HE-LTF

To generate an HE ranging NDP with secure HE-LTF symbols, as defined in [2 on page 1-0], specify either of these transmission parameter combinations.

- A single-user `heRangingConfig` object with its `SecureHELTF` property set to `1` (`true`)
- A multi-user `heRangingConfig` object

The secure HE-LTF comprises a randomized LTF sequence as defined in Section 27.3.17c of [1 on page 1-0]. To specify this sequence for a chosen user, set the `SecureHELTFSequence` property of the corresponding `User` property of the `heRangingConfig` object. If the number of bits in the `SecureHELTFSequence` property is less than the required number of bits for the given user configuration, the object cyclically extends the secure sequence. If the number of bits in `SecureHELTFSequence` is more than the required number of bits for the given user configuration, the object uses only the required number of bits. The object extracts the required bits from the specified hexadecimal sequence. When the transmission contains a secure HE-LTF sequence, the sequence must use a zero-power guard interval for the HE-LTF symbols. The packet extension (PE) starts with a zero-power guard interval.

Single-User HE Ranging NDP with Secure HE-LTF Generation

Configure a transmission with two antennas, two space-time streams, three HE-LTF repetitions, and secure HE-LTF symbols.

```
cfg = heRangingConfig('NumTransmitAntennas',2,'SecureHELTF',true);
cfg.User{1}.NumSpaceTimeStreams = 2;
```

```
cfg.User{1}.NumHELTFRepetition = 3;
cfg.User{1}.SecureHELTFSequence = '12345678ABCDEF1234';
```

Generate the HE ranging NDP waveform for the specified configuration.

```
tx = heRangingWaveformGenerator(cfg);
```

Plot the transmission power on the first antenna.

```
heRangingWavGenPlot(tx,cfg);
```
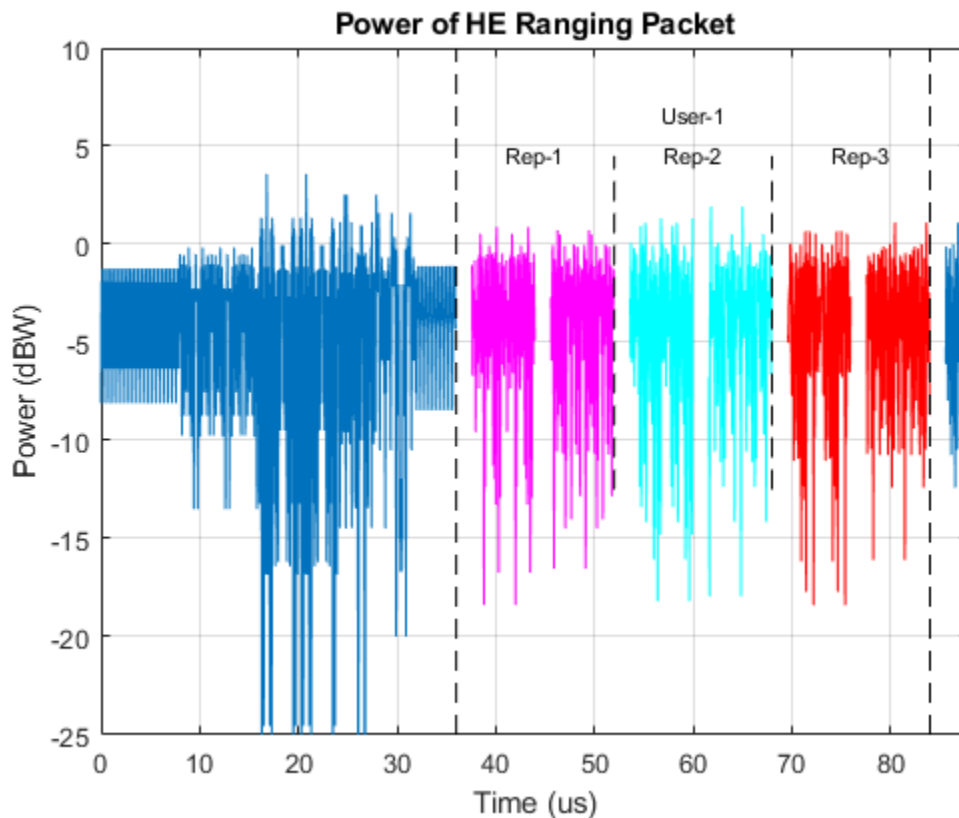

Multi-User HE Ranging NDP Generation

A multi-user HE ranging NDP waveform contains secure HE-LTF symbols for multiple users. The transmission concatenates HE-LTF symbols for each user up to a maximum of 64 consecutive symbols. This example demonstrates waveform generation with a secure HE-LTF sequence shown for two users. Configure transmission parameters, specifying two users and the number of space-time streams and HE-LTF repetitions for each user.

```
cfg = heRangingConfig(2);
cfg.User{1}.NumSpaceTimeStreams = 1;
cfg.User{1}.NumHELTFRepetition = 2;
cfg.User{2}.NumSpaceTimeStreams = 1;
cfg.User{2}.NumHELTFRepetition = 3;
```

Determine the number of secure HE-LTF bits required to generate the secure HE-LTF symbols for each user by using the `numSecureHELTFBits` object function.

```
numNibbles = numSecureHELTFBits(cfg)/4; % 4 bits per nibble
```

Set the secure HE-LTF sequences for each user.

```
secureSeqUser1 = 'a12c67f8b90dc56e78a2b3f1';
cfg.User{1}.SecureHELTFSequence = secureSeqUser1(1:numNibbles(1));
secureSeqUser2 = 'b3a49c5e6c1a2d35ed47c2d915f';
cfg.User{2}.SecureHELTFSequence = secureSeqUser2(1:numNibbles(2));
```

Generate the HE ranging NDP waveform for the specified configuration.

```
tx = heRangingWaveformGenerator(cfg);
```

Plot the transmission power on the first antenna.

```
heRangingWavGenPlot(tx,cfg);
```


References

1 IEEE P802.11az™/D2.0 Draft Standard for Information technology — Telecommunications and information exchange between systems Local and metropolitan area networks — Specific requirements - Amendment 3: Enhancements for positioning.

2 IEEE P802.11ax™/D4.1 Draft 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 6: Enhancements for High Efficiency WLAN.

3 IEEE Std 802.11™-2016 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.

Build VHT PPDU

Build VHT PPDUs by using the waveform generator function or by building each field individually.

Waveform Generator

Create a VHT configuration object.

```
vht = wlanVHTConfig;
```

Generate the VHT PPDU. The length of the input data sequence in bits must be 8 times the length of the PSDU, which is expressed in bytes. Turn off windowing.

```
x = randi([0 1],vht.PSDULength*8,1);
y = wlanWaveformGenerator(x,vht,'WindowTransitionTime',0);
```

Plot the magnitude of the waveform.

```
t = ((1:length(y))'-1)/80e6;
plot(t,abs(y))
xlabel('Time (s)')
ylabel('Magnitude (V)')
```
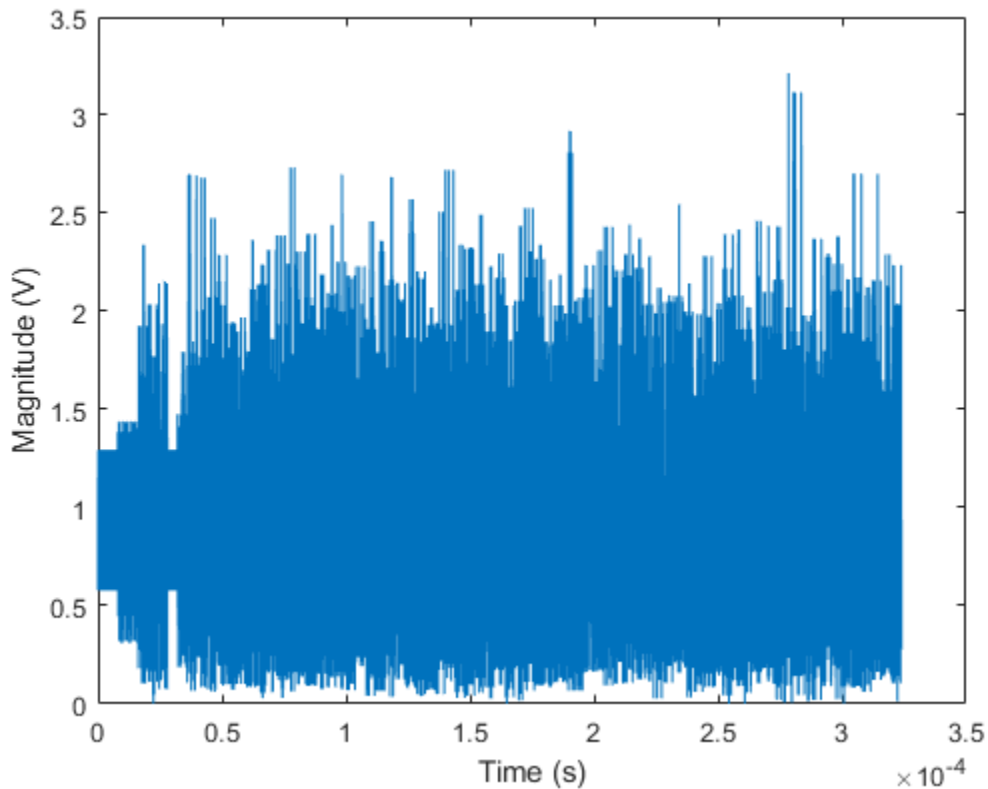

Individual PPDU Fields

Create L-STF, L-LTF, L-SIG, VHT-SIG-A, VHT-STF, VHT-LTF, and VHT-SIG-B preamble fields.

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

Generate the VHT-Data field using input data field x, which was used as an input to the waveform generator.

```
vhtData = wlanVHTData(x,vht);
```

Concatenate the individual fields to create a single PPDU.

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

Verify that the PPDUs created by the two methods are identical.

```
isequal(y,z)
```

```
ans = logical
   1
```

See Also

"Build S1G PPDU" on page 1-13 | "Build DMG PPDU" on page 1-14 | "Build HT PPDU" on page 1-16 | "Build Non-HT PPDU" on page 1-18

Generate VHT Multi-User Waveform

This example shows how to generate a VHT multi-user waveform from individual components. It also shows how to generate the same waveform by using the `wlanWaveformGenerator` function.

Create a VHT configuration object, specifying three users and three transmit antennas.

```
vht = wlanVHTConfig('NumUsers',3,'NumTransmitAntennas',3);
```

Set the number of space-time streams to the vector [1 1 1], which indicates that each user is assigned one space-time stream. Set the user positions to [0 1 2]. Set the group ID to 5. Group ID values from 1 to 62 apply for multiuser operation.

```
vht.NumSpaceTimeStreams = [1 1 1];
vht.UserPositions = [0 1 2];
vht.GroupID = 5;
```

Set a different MCS value for each user.

```
vht.MCS = [0 2 4];
```

Set the APEP length to 2000, 1400, and 1800 bytes. Each element corresponds to the number of bytes assigned to each user.

```
vht.APEPLength = [2000 1400 1800]

vht =
  wlanVHTConfig with properties:

       ChannelBandwidth: 'CBW80'
               NumUsers: 3
          UserPositions: [0 1 2]
     NumTransmitAntennas: 3
    NumSpaceTimeStreams: [1 1 1]
         SpatialMapping: 'Direct'
                    MCS: [0 2 4]
         ChannelCoding: 'BCC'
             APEPLength: [2000 1400 1800]
          GuardInterval: 'Long'
                GroupID: 5

  Read-only properties:
             PSDULength: [2000 6008 12019]
```

Display the PSDU lengths for the three users. The PSDU length is a function of both the APEP length and the MCS value.

```
vht.PSDULength

ans = 1×3

        2000        6008       12019
```

Display the field indices for the VHT waveform.

```
ind = wlanFieldIndices(vht)
```

```
ind = struct with fields:
        LSTF: [1 640]
        LLTF: [641 1280]
        LSIG: [1281 1600]
     VHTSIGA: [1601 2240]
      VHTSTF: [2241 2560]
      VHTLTF: [2561 3840]
     VHTSIGB: [3841 4160]
     VHTData: [4161 48000]
```

Create the individual fields that comprise the VHT waveform.

```
lstf = wlanLSTF(vht);
lltf = wlanLLTF(vht);
lsig = wlanLSIG(vht);
[vhtsigA,sigAbits] = wlanVHTSIGA(vht);
vhtstf = wlanVHTSTF(vht);
vhtltf = wlanVHTLTF(vht);
[vhtsigB,sigBbits] = wlanVHTSIGB(vht);
```

Extract the first two VHT-SIG-A information bits and convert them to their decimal equivalent.

```
bw = bi2de(double(sigAbits(1:2)'))
```

```
bw = 2
```

The value, 2, corresponds to an 80 MHz bandwidth (see `wlanVHTSIGA`).

Extract VHT-SIG-A information bits 5 through 10, and convert them to their decimal equivalent.

```
groupid = bi2de(double(sigAbits(5:10)'))
```

```
groupid = 5
```

The extracted group ID, 5, matches the corresponding property in the VHT configuration object.

Extract the packet length from the VHT-SIG-B information bits. For multiuser operation with an 80 MHz bandwidth, the first 19 bits contain the APEP length information. Convert the field lengths to their decimal equivalents. Multiply them by 4 because the length of the VHT-SIG-B field is expressed in units of 4 bytes.

```
pktLen = bi2de(double(sigBbits(1:19,:)'))*4
```

```
pktLen = 3×1

        2000
        1400
        1800
```

Confirm that the extracted APEP length matches the value set in the configuration object.

```
isequal(pktLen',vht.APEPLength)
```

```
ans = logical
    1
```

Extract the MCS values from the VHT-SIG-B information bits. The MCS component is specified by bits 20 to 23.

```
mcs = bi2de(double(sigBbits(20:23,:)'))
```

mcs = *3×1*

```
    0
    2
    4
```

The values correspond to those set in the VHT configuration object.

Create three data sequences, one for each user.

```
d1 = randi([0 1],vht.PSDULength(1)*8,1);
d2 = randi([0 1],vht.PSDULength(2)*8,1);
d3 = randi([0 1],vht.PSDULength(3)*8,1);
```

Generate a VHT data field using these data sequences.

```
vhtdata = wlanVHTData({d1 d2 d3},vht);
```

Generate a multiuser VHT waveform with windowing is disabled. Extract the data field from the waveform.

```
wv = wlanWaveformGenerator({d1 d2 d3},vht,'WindowTransitionTime',0);
wvdata = wv(ind.VHTData(1):ind.VHTData(2),:);
```

Confirm that the two generation approaches produce identical results.

```
isequal(vhtdata,wvdata)
```

ans = *logical*
```
   1
```

Visualize the waveform by plotting its magnitude.

```
t = ((1:length(wv))'-1)/80e6;
plot(t,abs(wv))
xlabel('Time (s)')
ylabel('Magnitude')
```
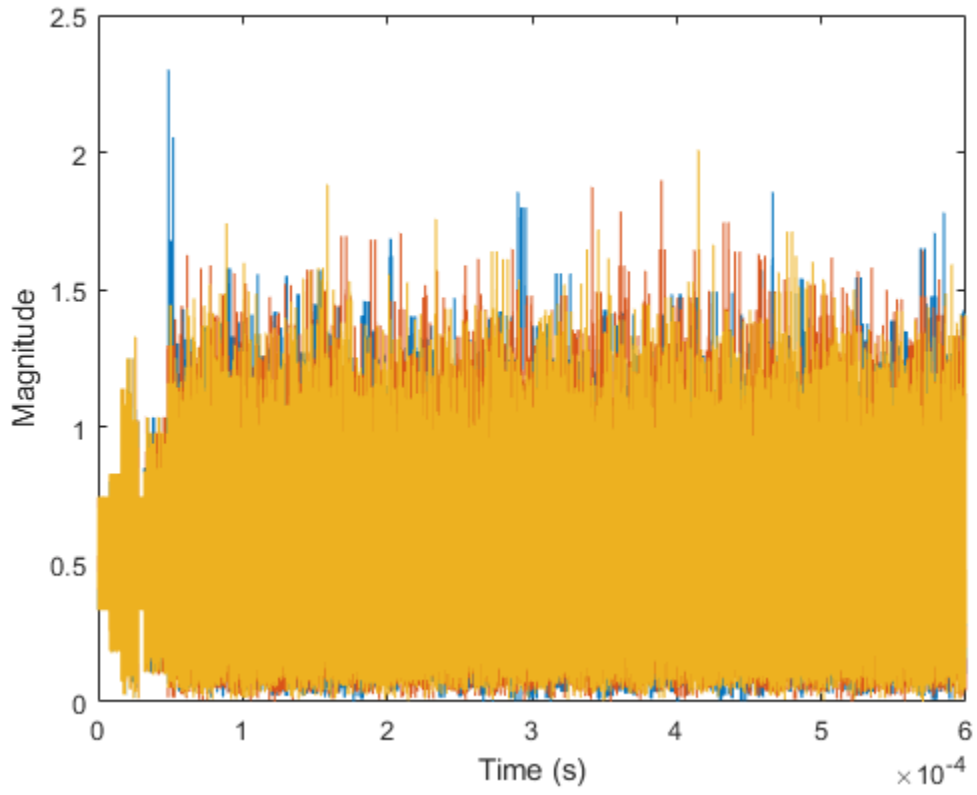
Build S1G PPDU

Build S1G PPDUs by using the waveform generator function.

Waveform Generator

Create an S1G configuration object.

```
s1g = wlanS1GConfig;
```

Generate the S1G PPDU. The length of the input data sequence in bits must be 8 times the length of the PSDU, which is expressed in bytes. Turn off windowing.

```
x = randi([0 1],s1g.PSDULength*8,1);
y = wlanWaveformGenerator(x,s1g,'WindowTransitionTime',0);
```

Plot the magnitude of the waveform.

```
t = ((1:length(y))'-1)/80e6;
plot(t,abs(y))
xlabel('Time (s)')
ylabel('Magnitude (V)')
```
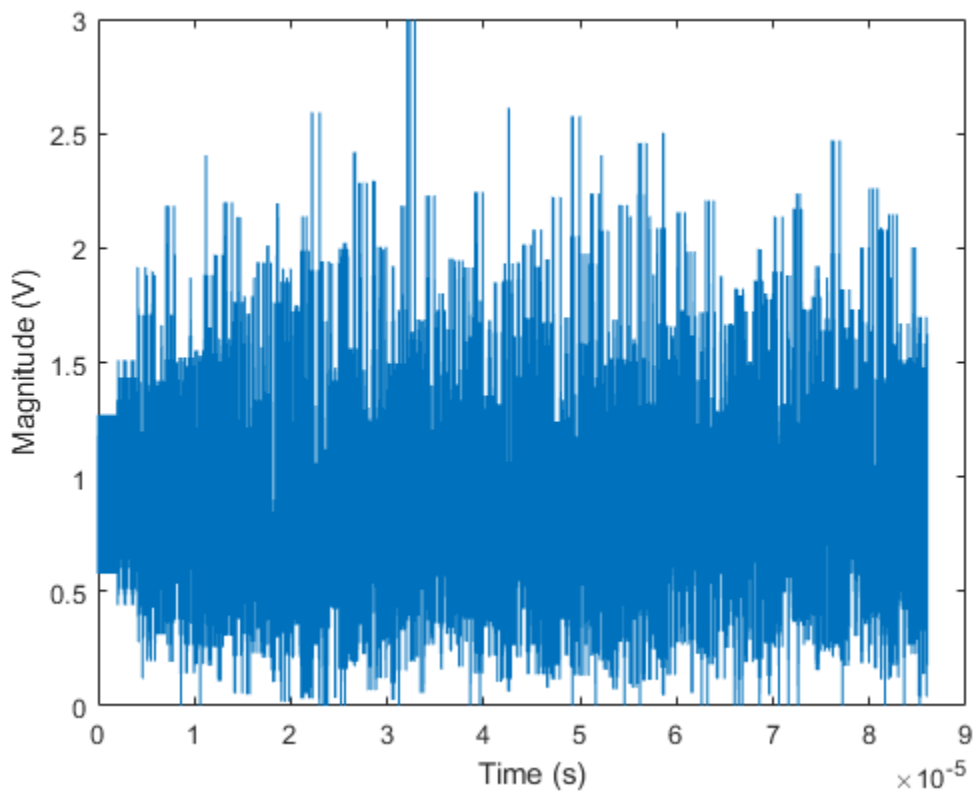

See Also

Build DMG PPDU

Build DMG PPDUs by using the waveform generator function.

Waveform Generator

Create an DMG configuration object.

```
dmg = wlanDMGConfig
```

```
dmg =
  wlanDMGConfig with properties:

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

Generate the DMG PPDU. The length of the input data sequence in bits must be 8 times the length of the PSDU, which is expressed in bytes. Turn off windowing.

```
psdu = randi([0 1],dmg.PSDULength*8,1);
tx = wlanWaveformGenerator(psdu,dmg,'WindowTransitionTime',0);
```

The waveform has MCS=0, which is single carrier and DBPSK modulated. Plot the constellation of the waveform.

```
scatterplot(tx)
```
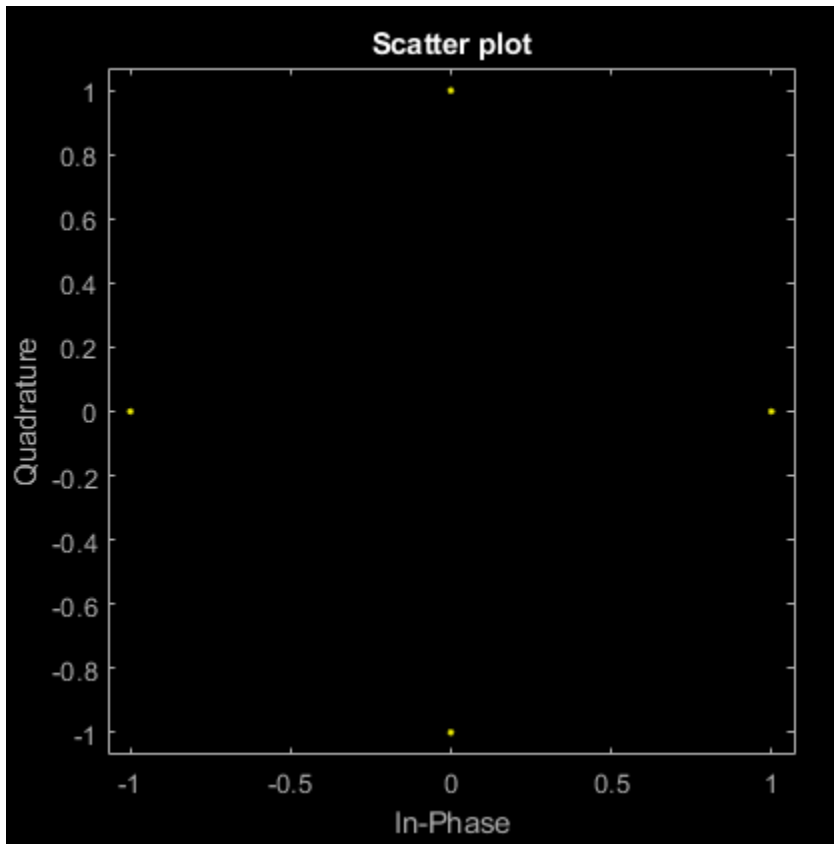
See Also

Build HT PPDU

Build HT PPDUs by using the waveform generator function or by building each field individually.

Waveform Generator

Create an HT configuration object.

```
ht = wlanHTConfig;
```

Generate the HT PPDU. The length of the input data sequence in bits must be 8 times the length of the PSDU, which is expressed in bytes. Turn windowing off.

```
x = randi([0 1],ht.PSDULength*8,1);
y = wlanWaveformGenerator(x,ht,'WindowTransitionTime',0);
```

Plot the magnitude of the waveform.

```
t = ((1:length(y))'-1)/20e6;
plot(t,abs(y))
xlabel('Time (s)')
ylabel('Magnitude (V)')
```
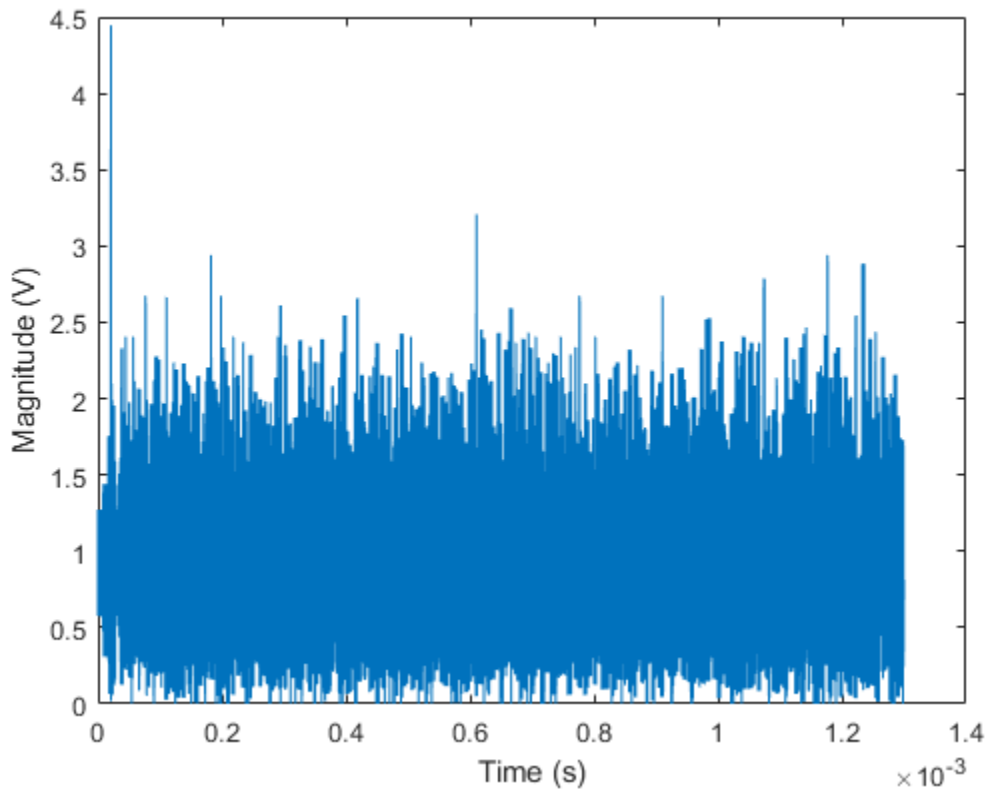

Individual PPDU Fields

Create L-STF, L-LTF, L-SIG, HT-SIG, HT-STF, and HT-LTF preamble fields.

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

Generate the HT-Data field using input data field x, which is the same input signal that was used with the waveform generator.

```
htData = wlanHTData(x,ht);
```

Concatenate the individual fields to create a single PPDU.

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

Verify that the PPDUs created by the two methods are identical.

```
isequal(y,z)
```

```
ans = logical
   1
```

See Also

"Build VHT PPDU" on page 1-7 | "Build S1G PPDU" on page 1-13 | "Build DMG PPDU" on page 1-14 | "Build Non-HT PPDU" on page 1-18

Build Non-HT PPDU

Build non-HT PPDUs by using the waveform generator function or by building each field individually.

Waveform Generator

Create a non-HT configuration object.

```
nht = wlanNonHTConfig;
```

Generate the non-HT PPDU. The length of the input data sequence in bits must be 8 times the length of the PSDU, which is expressed in bytes. Turn off windowing.

```
x = randi([0 1],nht.PSDULength*8,1);
y = wlanWaveformGenerator(x,nht,'WindowTransitionTime',0);
```

Plot the magnitude of the waveform.

```
t = ((1:length(y))'-1)/20e6;
plot(t,abs(y))
xlabel('Time (s)')
ylabel('Magnitude (V)')
```
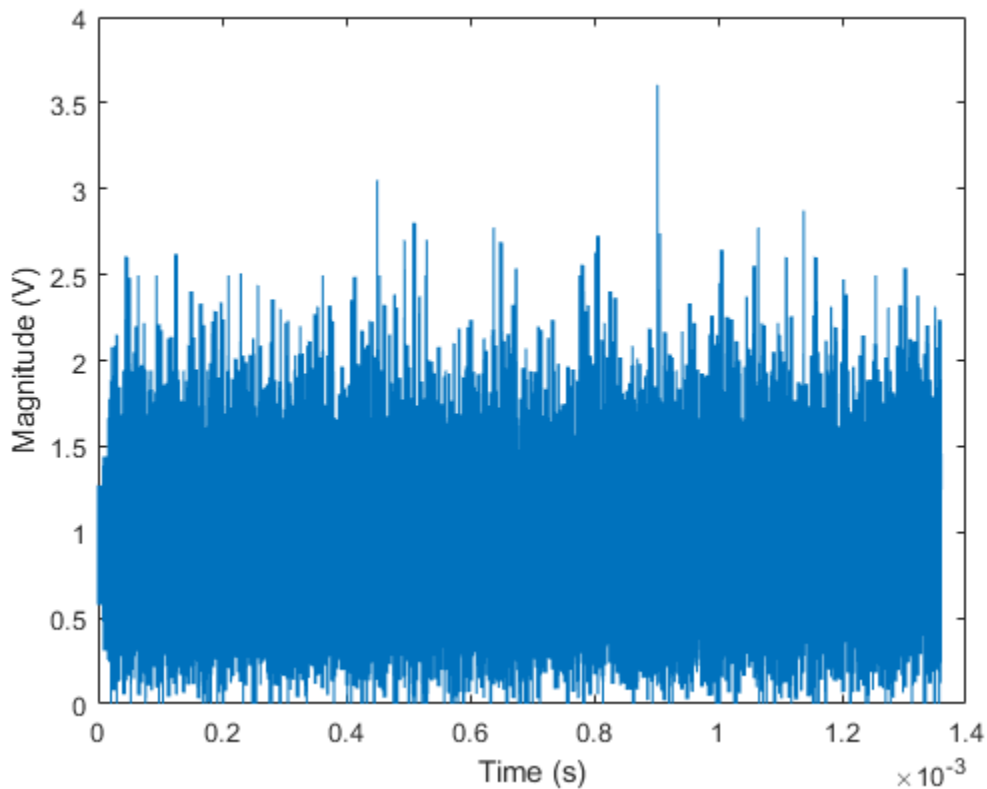

Individual PPDU Fields

Create L-STF, L-LTF, and L-SIG preamble fields.

```
lstf = wlanLSTF(nht);
lltf = wlanLLTF(nht);
lsig = wlanLSIG(nht);
```

Generate the Non-HT-data field using input data field x, which was used as the input to the waveform generator.

```
nhtData = wlanNonHTData(x,nht);
```

Concatenate the individual fields to create a single PPDU.

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

Verify that the PPDUs created by the two methods are identical.

```
isequal(y,z)
```

```
ans = logical
   1
```

See Also

"Build VHT PPDU" on page 1-7 | "Build S1G PPDU" on page 1-13 | "Build DMG PPDU" on page 1-14 | "Build HT PPDU" on page 1-16

Basic VHT Data Recovery

This example shows how to perform basic VHT data recovery. It also shows how to recover VHT data when the received signal has a carrier frequency offset. Similar procedures can be used to recover data with the HT and non-HT formats.

Basic Data Recovery

WLAN Toolbox™ provides functions to generate and recover IEEE® 802.11ac™ standards-compliant waveforms. The data recovery process comprises these steps.

1 Generate a VHT waveform

2 Pass the waveform through a channel

3 Extract the VHT-LTF and demodulate

4 Estimate the channel by using the demodulated VHT-LTF

5 Extract the data field

6 Recover the data by using the channel and noise variance estimates

The block diagram shows these steps, along with their corresponding commands.
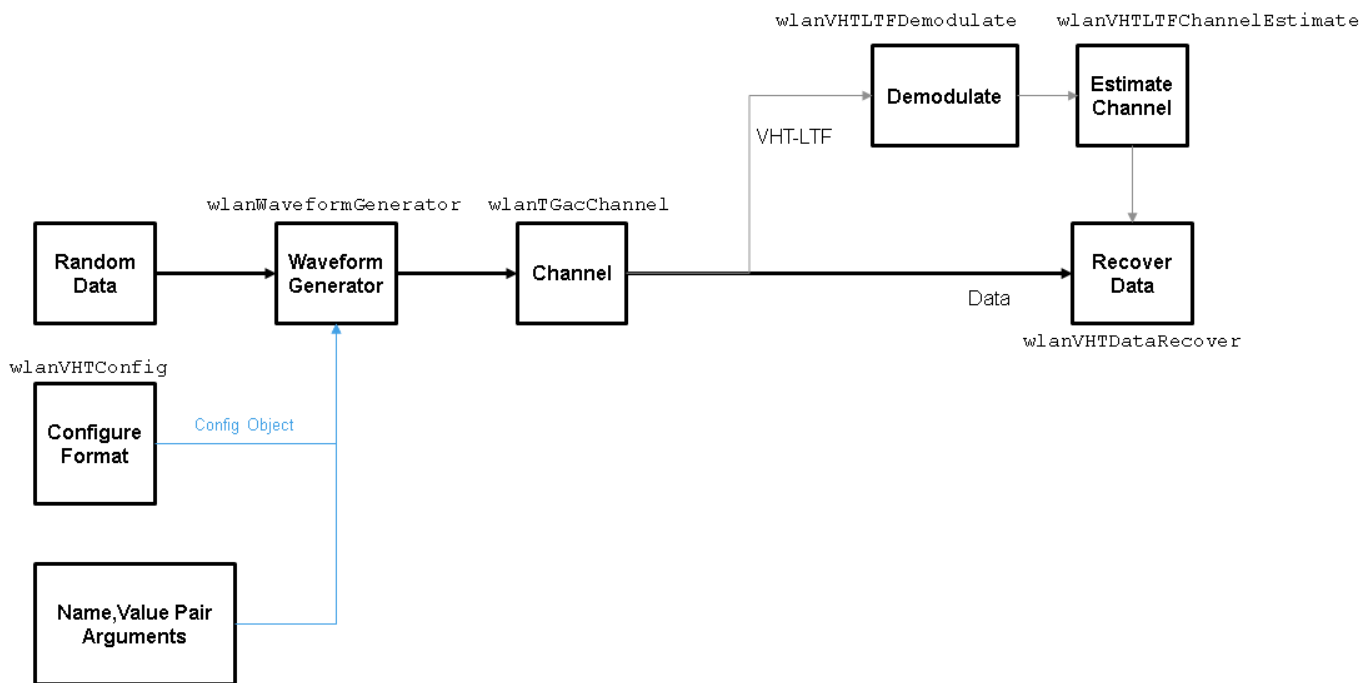
Create VHT configuration object.

```
cfg = wlanVHTConfig;
```

Create a VHT transmit waveform by using the VHT configuration object. Set the data sequence to `[1;0;1;1]`. The waveform generator function repeats the data sequence to generate the specified number of packets.

```
txSig = wlanWaveformGenerator([1;0;1;1],cfg);
```

Pass the received signal through an AWGN channel.

```
rxSig = awgn(txSig,10);
```

Determine the field indices of the waveform.

```
ind = wlanFieldIndices(cfg);
```

Extract the VHT-LTF from the received signal.

```
rxVHTLTF = rxSig(ind.VHTLTF(1):ind.VHTLTF(2),:);
```

Demodulate the VHT-LTF. Estimate the channel response by using the demodulated signal.

```
demodVHTLTF = wlanVHTLTFDemodulate(rxVHTLTF,cfg);
chEst = wlanVHTLTFChannelEstimate(demodVHTLTF,cfg);
```

Extract the VHT data field.

```
rxData = rxSig(ind.VHTData(1):ind.VHTData(2),:);
```

Recover the information bits by using the channel and noise variance estimates. Confirm that the first 8 bits match two repetitions of the input data sequence of [1;0;1;1].

```
rxBits = wlanVHTDataRecover(rxData,chEst,0.1,cfg);
```

```
rxBits(1:8)
```

```
ans = 8x1 int8 column vector

   1
   0
   1
   1
   1
   0
   1
   1
```

Data Recovery with Frequency Correction

Data recovery when a carrier frequency offset is present is accomplished by these steps.

1   Generate a VHT waveform

2   Pass the waveform through a channel

3   Extract the L-STF and perform a coarse frequency offset estimate

4   Correct for the offset by using the coarse estimate

5   Extract the L-LTF and perform a fine frequency offset estimate

6   Correct for the offset by using the fine estimate

7   Extract the VHT-LTF and demodulate

8   Estimate the channel by using the demodulated VHT-LTF

9   Extract the data field

10   Recover the data by using the channel and noise variance estimates

The block diagram shows these steps, along with their corresponding commands.
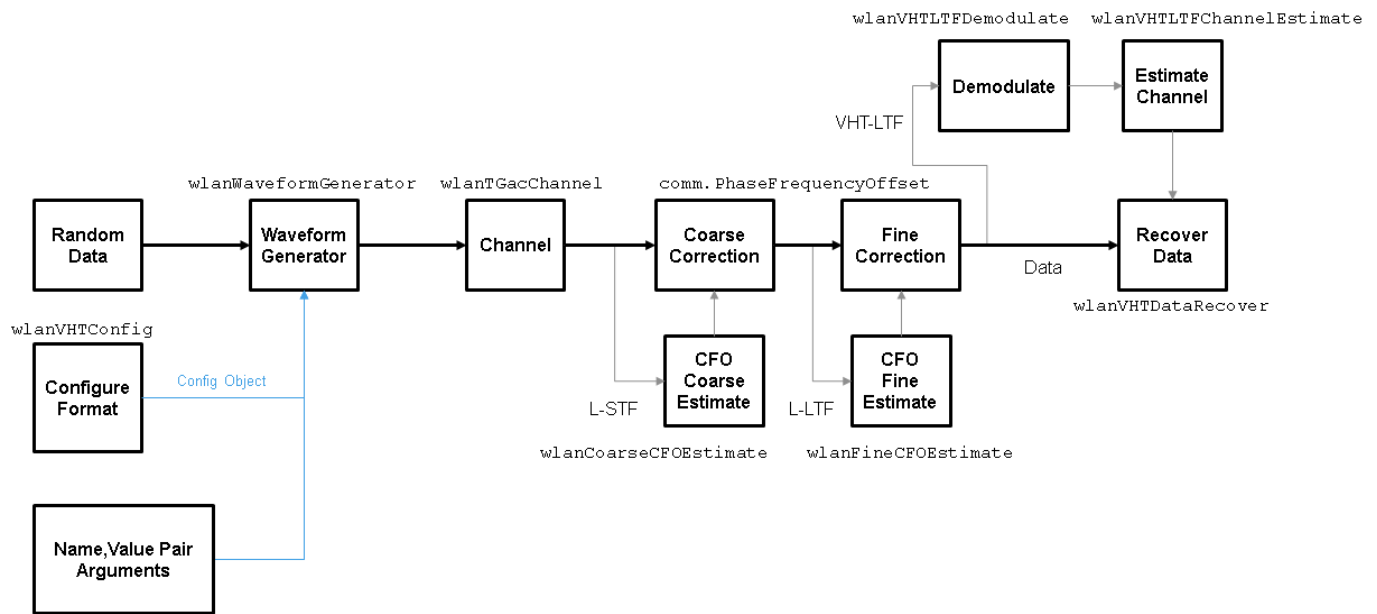
Set the channel bandwidth and sample rate.

```
cbw = 'CBW160';
fs = 160e6;
```

Create a VHT configuration object that supports a 2x2 MIMO transmission.

```
cfg = wlanVHTConfig('ChannelBandwidth',cbw, ...
    'NumTransmitAntennas',2,'NumSpaceTimeStreams',2);
```

Generate a VHT waveform containing a random PSDU.

```
txPSDU = randi([0 1],cfg.PSDULength*8,1);
txSig = wlanWaveformGenerator(txPSDU,cfg);
```

Create a 2x2 TGac channel.

```
tgacChan = wlanTGacChannel('SampleRate',fs,'ChannelBandwidth',cbw, ...
    'NumTransmitAntennas',2,'NumReceiveAntennas',2);
```

Create a phase and frequency offset object.

```
pfOffset = comm.PhaseFrequencyOffset('SampleRate',fs,'FrequencyOffsetSource','Input port');
```

Pass the transmitted waveform through the noisy TGac channel.

```
rxSigNoNoise = tgacChan(txSig);
rxSig = awgn(rxSigNoNoise,15);
```

Introduce a frequency offset of 500 Hz to the received signal.

```
rxSigFreqOffset = pfOffset(rxSig,500);
```

Find the start and stop indices for all component fields of the PPDU.

```
ind = wlanFieldIndices(cfg);
```

Extract the L-STF. Estimate and correct for the carrier frequency offset.

```
rxLSTF = rxSigFreqOffset(ind.LSTF(1):ind.LSTF(2),:);

foffset1 = wlanCoarseCFOEstimate(rxLSTF,cbw);
rxSig1 = pfOffset(rxSigFreqOffset,-foffset1);
```

Extract the L-LTF from the corrected signal. Estimate and correct for the residual frequency offset.

```
rxLLTF = rxSig1(ind.LLTF(1):ind.LLTF(2),:);

foffset2 = wlanFineCFOEstimate(rxLLTF,cbw);
rxSig2 = pfOffset(rxSig1,-foffset2);
```

Extract and demodulate the VHT-LTF. Estimate the channel coefficients.

```
rxVHTLTF = rxSig2(ind.VHTLTF(1):ind.VHTLTF(2),:);
demodVHTLTF = wlanVHTLTFDemodulate(rxVHTLTF,cfg);
chEst = wlanVHTLTFChannelEstimate(demodVHTLTF,cfg);
```

Extract the VHT data field from the received and frequency-corrected PPDU. Recover the data field.

```
rxData = rxSig2(ind.VHTData(1):ind.VHTData(2),:);
rxPSDU = wlanVHTDataRecover(rxData,chEst,0.03,cfg);
```

Calculate the number of bit errors in the received packet.

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

```
numErr = 2
```

802.11ax Parameterization for Waveform Generation and Simulation

This example shows how to parameterize and generate different IEEE® 802.11ax™ high efficiency (HE) format packets.

Introduction

IEEE P802.11ax/D4.1 [1] specifies four high efficiency (HE) packet formats:

1 Single-user

2 Extended-range single-user

3 Multi-user

4 Trigger-based

This example shows how packets can be generated for these different formats, and demonstrates some of the key features of the draft standard [1].

HE Single-User Format

An HE single-user (SU) packet is a full-band transmission to a single user. The transmit parameters for the HE SU format are configured using a `wlanHESUConfig` object. The `wlanHESUConfig` object can be configured to operate in extended-range mode. To enable or disable this mode, set the `ExtendedRange` property to `true` or `false`. In this example we create a configuration for an HE SU transmission and configure transmission properties.

```
cfgSU = wlanHESUConfig;
cfgSU.ExtendedRange = false;      % Do not use extended-range format
cfgSU.ChannelBandwidth = 'CBW20'; % Channel bandwidth
cfgSU.APEPLength = 1000;          % Payload length in bytes
cfgSU.MCS = 0;                    % Modulation and coding scheme
cfgSU.ChannelCoding = 'LDPC';     % Channel coding
cfgSU.NumSpaceTimeStreams = 1;    % Number of space-time streams
cfgSU.NumTransmitAntennas = 1;    % Number of transmit antennas
```

A single-user packet can be generated with the waveform generator, `wlanWaveformGenerator`. The `getPSDULength()` method returns the required PSDU length given the transmission configuration. This length is used to create a random PSDU for transmission.

```
psdu = randi([0 1],getPSDULength(cfgSU)*8,1,'int8'); % Random PSDU
txSUWaveform = wlanWaveformGenerator(psdu,cfgSU);     % Create packet
```

HE Extended-Range Single-User Format

An extended-range single-user packet has the same fields as the standard single-user format, but the powers of some fields are boosted, and some fields are repeated to improve performance at low SNRs. An extended-range packet can be configured using a `wlanHESUConfig` object with `ChannelBandwidth` set to `'CBW20'` and `ExtendedRange` set to `true`. An extended-range packet has an option to only transmit in the upper 106-tone resource unit (RU) within the 20 MHz channel, or over the entire bandwidth. This can be configured with the `Upper106ToneRU` property:

```
cfgExtSU = cfgSU;
cfgExtSU.ExtendedRange = true;  % Enable extended-range format
cfgExtSU.Upper106ToneRU = true; % Use only upper 106-tone RU
```

```
% Generate a packet
psdu = randi([0 1],getPSDULength(cfgExtSU)*8,1,'int8'); % Random PSDU
txExtSUWaveform = wlanWaveformGenerator(psdu,cfgExtSU);   % Create packet
```

Look at the spectrum and spectrogram of the generated signal. In the spectrogram you can see that the packet headers use the available bandwidth, however, the data portion only occupies the upper half of the channel.

```
fs = wlanSampleRate(cfgExtSU); % Get baseband sample rate
ofdmInfo = wlanHEOFDMInfo('HE-Data',cfgExtSU);
fftsize = ofdmInfo.FFTLength; % Use the data field fft size
rbw = fs/fftsize; % Resoluton bandwidth
spectrumAnalyzer = dsp.SpectrumAnalyzer('SampleRate',fs,...
    'Method','Filter bank','RBWSource','Property','RBW',rbw,...
    'AveragingMethod','Exponential','ForgettingFactor',0.25,...
    'ReducePlotRate',false,'YLimits',[-50,20],...
    'Title','HE Extended-Range SU with Active Upper 106-Tone RU');
spectrumAnalyzer.ViewType = 'Spectrum and Spectrogram';
spectrumAnalyzer.TimeSpanSource = 'Property';
spectrumAnalyzer.TimeSpan = length(txExtSUWaveform)/fs;
spectrumAnalyzer(txExtSUWaveform)
```
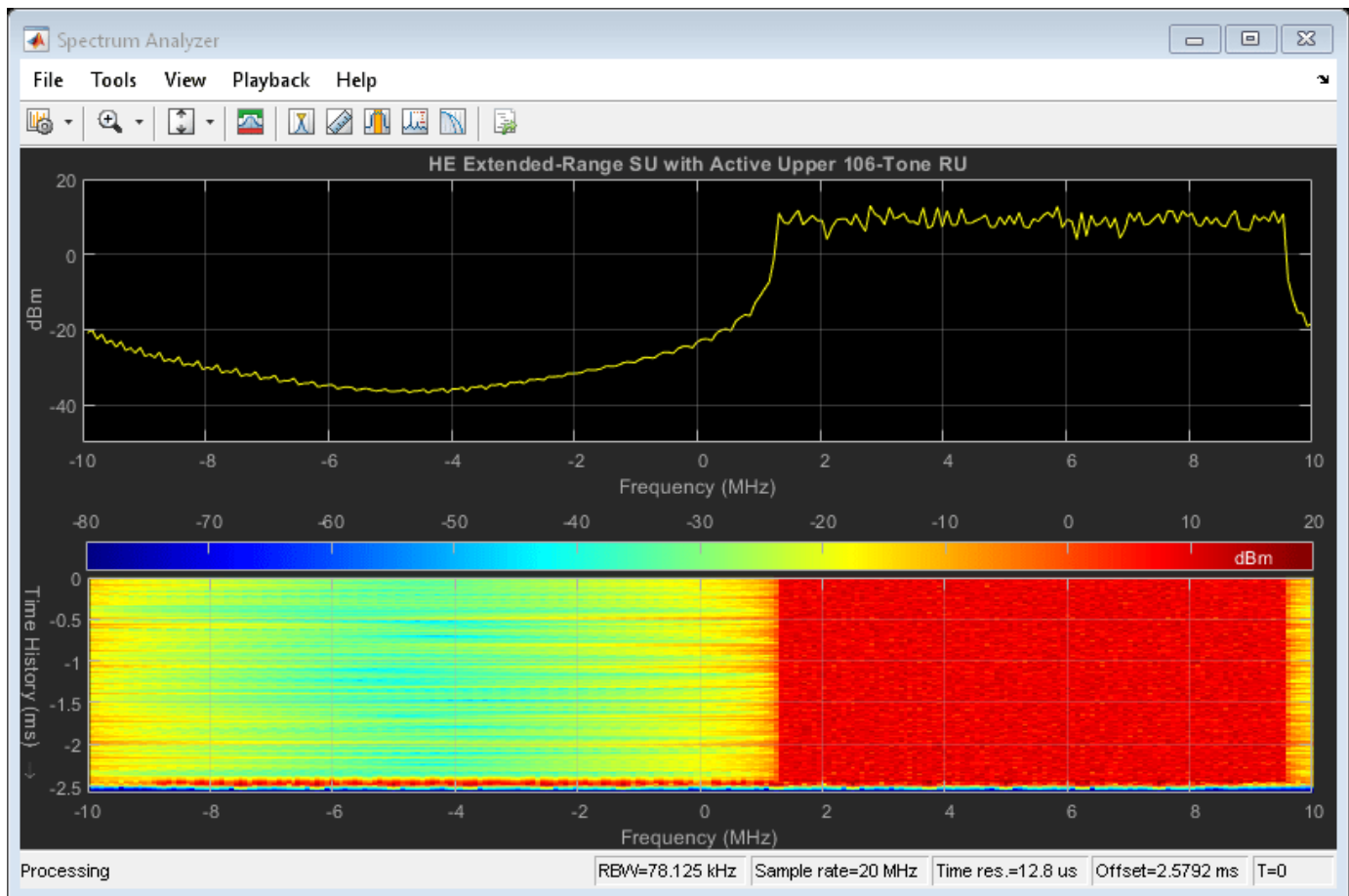

If you compare the power of the L-STF and L-LTF fields you can see that the extended-range transmission is boosted by 3 dB.

```
figure;
ind = wlanFieldIndices(cfgExtSU);
t = (0:(ind.LLTF(2)-1))/fs*1e6;
plot(t,20*log10(movmean(abs(txSUWaveform(1:ind.LLTF(2))),20)),'-b')
hold on;
plot(t,20*log10(movmean(abs(txExtSUWaveform(1:ind.LLTF(2))),20)),'-r')
grid on;
title('Power of L-STF and L-LTF (1 us Moving Average)');
xlabel('Time (us)');
ylabel('Power (dBW)');
legend('HE SU','HE Extended-Range SU','Location','SouthWest');
```


Power of L-STF and L-LTF (1 us Moving Average)
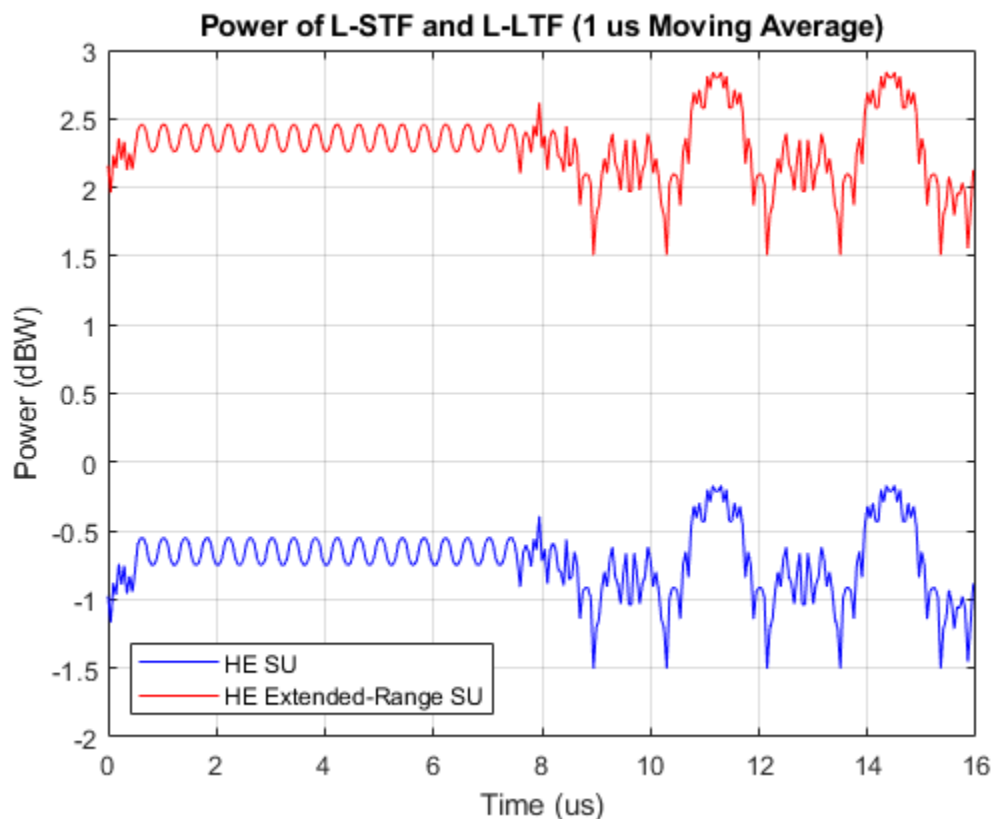
HE Multi-User Format - OFDMA

The HE multi-user (HE MU) format can be configured for an OFDMA transmission, a MU-MIMO transmission, or a combination of the two. This flexibility allows an HE MU packet to transmit to a single user over the whole band, multiple users over different parts of the band (OFDMA), or multiple users over the same part of the band (MU-MIMO).

For an OFDMA transmission, the channel bandwidth is divided into resource units (RUs). An RU is a group of subcarriers assigned to one or more users. An RU is defined by a size (the number of subcarriers) and an index. The RU index specifies the location of the RU within the channel. For example, in an 80 MHz transmission there are four possible 242-tone RUs, one in each 20 MHz subchannel. RU# 242-1 (size 242, index 1) is the RU occupying the lowest absolute frequency within the 80 MHz, and RU# 242-4 (size 242, index 4) is the RU occupying the highest absolute frequency. The draft standard defines possible sizes and locations of RUs in Section 27.3.2.2 of [1].

The assignment of RUs in a transmission is defined by the allocation index. The allocation index is defined in Table 28-24 of [1]. For each 20 MHz subchannel, an 8-bit index describes the number and size of RUs, and the number of users transmitted on each RU. The allocation index also determines which content channel is used to signal a user in HE-SIG-B. The allocation indices within Table 28-24, and the corresponding RU assignments, are provided in the table returned by the function `heRUAllocationTable`. The first 10 allocations within the table are shown below. For each allocation index, the 8-bit allocation index, the number of users, number of RUs, RU indices, RU sizes, and number of users per RU are displayed. A note is also provided about allocations which are reserved, or serve a special purpose. The allocation table can also be viewed in the Appendix.

```
allocationTable = heRUAllocationTable;
disp('First 10 entries in the allocation table: ')
disp(allocationTable(1:10,:));
```

```
First 10 entries in the allocation table:
    Allocation    BitAllocation    NumUsers    NumRUs        RUIndices                    RUS:
    _____    _____    _____    _____    _____          _____

        0          "00000000"          9          9      {[1 2 3 4 5 6 7 8 9]}    {[26 26 26 26 2
        1          "00000001"          8          8      {[  1 2 3 4 5 6 7 4]}    {[   26 26 26 2
        2          "00000010"          8          8      {[  1 2 3 4 5 3 8 9]}    {[   26 26 26 2
        3          "00000011"          7          7      {[     1 2 3 4 5 3 4]}   {[      26 26 2
        4          "00000100"          8          8      {[  1 2 2 5 6 7 8 9]}    {[   26 26 52 2
        5          "00000101"          7          7      {[     1 2 2 5 6 7 4]}   {[      26 26 5
        6          "00000110"          7          7      {[     1 2 2 5 3 8 9]}   {[      26 26 5
        7          "00000111"          6          6      {[        1 2 2 5 3 4]}  {[         26 2
        8          "00001000"          8          8      {[  1 3 4 5 6 7 8 9]}    {[   52 26 26 2
        9          "00001001"          7          7      {[     1 3 4 5 6 7 4]}   {[      52 26 2
```

A `wlanHEMUConfig` object is used to configure the transmission of an HE MU packet. The allocation index for each 20 MHz subchannel must be provided when creating an HE MU configuration object, `wlanHEMUConfig`. An integer between 0 and 223, corresponding to the 8-bit number in Table 28-24 of [1], must be provided for each 20 MHz subchannel.

The allocation index can be provided as a decimal or 8-bit binary sequence. In this example, a 20 MHz HE MU configuration is created with 8-bit allocation index "10000000". This is equivalent to the decimal allocation index 128. This configuration specifies 3 RUs, each with one user.

```
allocationIndex = "10000000"; % 3 RUs, 1 user per RU
cfgMU = wlanHEMUConfig(allocationIndex);
```

The `showAllocation` method visualizes the occupied RUs and subcarriers for the specified configuration. The colored blocks illustrate the occupied subcarriers in the pre-HE and HE portions of the packet. White indicates subcarriers are unoccupied. The pre-HE portion illustrates the occupied subcarriers in the fields preceding HE-STF. The HE portion illustrates the occupied subcarriers in the HE-STF, HE-LTF and HE-Data field and therefore shows the RU allocation. Clicking on an RU will display information about the RU. The RU number corresponds to the i-th RU element of the `cfgMU.RU` property. The size and index are the details of the RU. The RU index is the i-th possible RU of the corresponding RU size within the channel bandwidth, for example Index 2 is the 2nd possible 106-tone RU within the 20 MHz channel bandwidth. The user number corresponds to the i-th User element of the `cfgMU.User` property, and the user field in HE-SIG-B. Note the middle RU (RU #2) is split across the DC subcarriers.

```
showAllocation(cfgMU);
axAlloc = gca; % Get axis handle for subsequent plotting
```
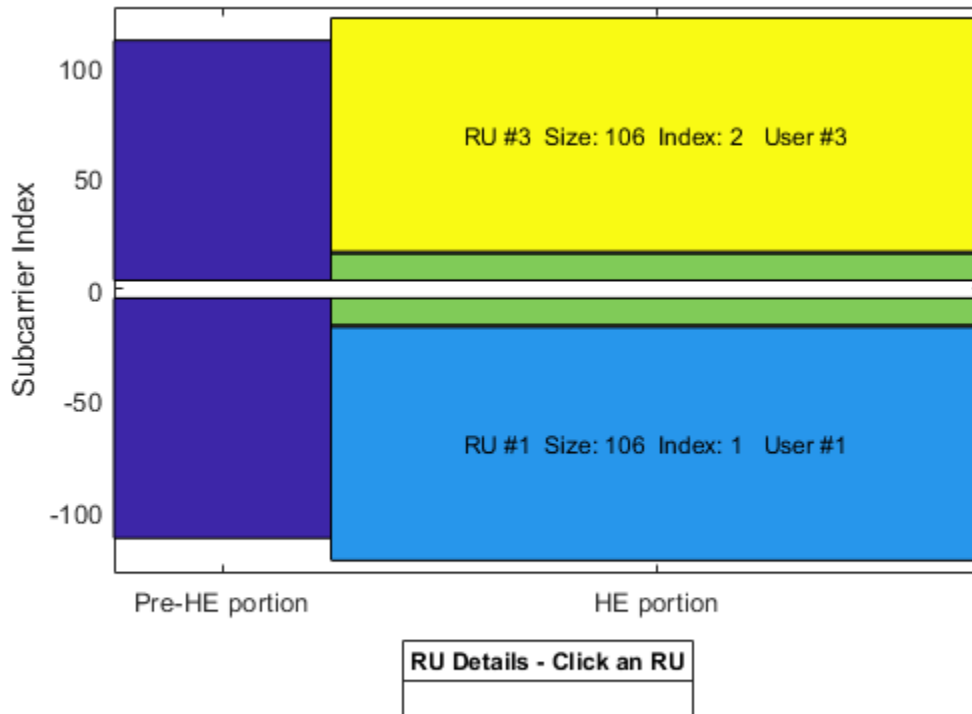
The `ruInfo` method provides details of the RUs in the configuration. In this case we can see three users and three RUs.

```
allocInfo = ruInfo(cfgMU);
disp('Allocation info:')
disp(allocInfo)

Allocation info:
                        NumUsers: 3
                          NumRUs: 3
                       RUIndices: [1 5 2]
                         RUSizes: [106 26 106]
                  NumUsersPerRU: [1 1 1]
      NumSpaceTimeStreamsPerRU: [1 1 1]
        PowerBoostFactorPerRU: [1 1 1]
                       RUNumbers: [1 2 3]
```

The properties of `cfgMU` describe the transmission configuration. The `cfgMU.RU` and `cfgMU.User` properties of `cfgMU` are cell arrays. Each element of the cell arrays contains an object which configures an RU or a User. When the `cfgMU` object is created, the elements of `cfgMU.RU` and `cfgMU.User` are configured to create the desired number of RUs and users. Each element of `cfgMU.RU` is a `wlanHEMURU` object describing the configuration of an RU. Similarly, each element of `cfgMU.User` is a `wlanHEMUUser` object describing the configuration of a User. This object hierarchy is shown below:
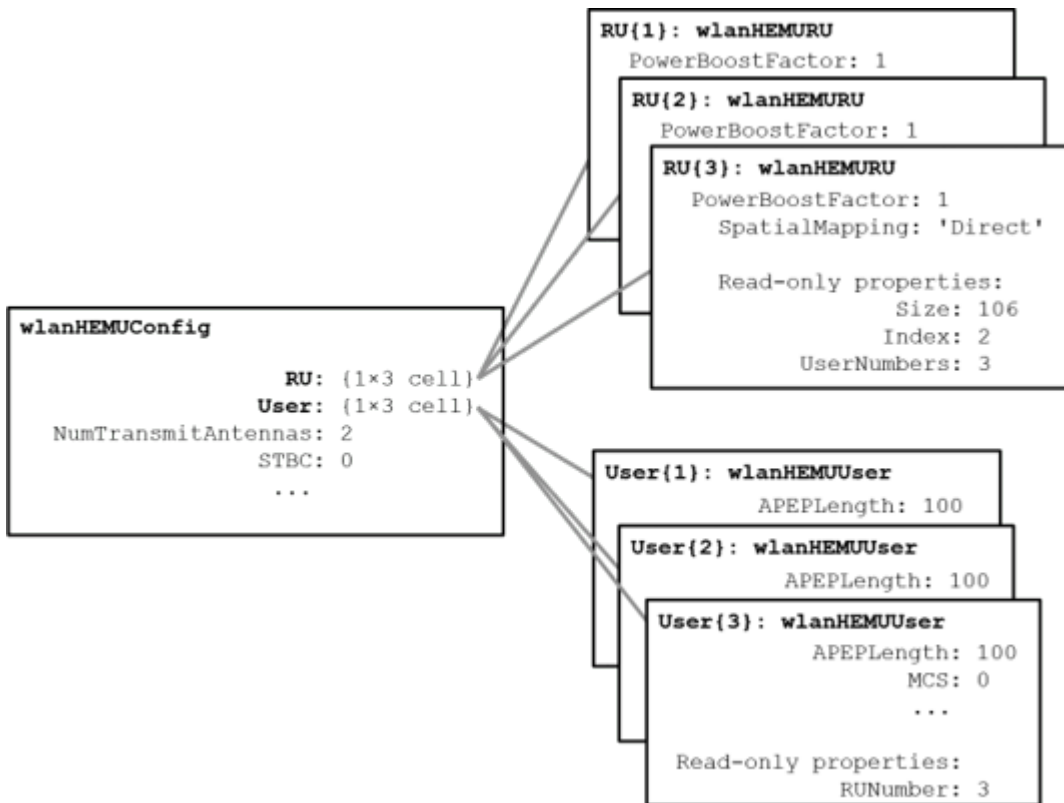
In this example, three RUs are specified by the allocation index 128, therefore `cfgMU.RU` is a cell array with three elements. The index and size of each RU are configured according to the allocation index used to create `cfgMU`. After the object is created, each RU can be configured to create the desired transmission configuration, by setting the properties of the appropriate RU object. For example, the spatial mapping and power boost factor can be configured per RU. The `Size` and `Index` properties of each RU are fixed once the object is created, and therefore are read-only properties. Similarly, the `UserNumbers` property is read-only and indicates which user is transmitted on the RU. For this configuration the first RU is size 106, index 1.

```
disp('First RU configuration:')
disp(cfgMU.RU{1})
```

```
First RU configuration:
  wlanHEMURU with properties:

    PowerBoostFactor: 1
      SpatialMapping: 'Direct'

  Read-only properties:
                Size: 106
               Index: 1
         UserNumbers: 1
```

In this example, the allocation index specifies three users in the transmission, therefore, `cfgMU.User` contains three elements. The transmission properties of users can be configured by modifying individual user objects, for example the MCS, APEP length and channel coding scheme. The read-only `RUNumber` property indicates which RU is used to transmit this user.

```
disp('First user configuration:')
disp(cfgMU.User{1})

First user configuration:
  wlanHEMUUser with properties:

              APEPLength: 100
                     MCS: 0
      NumSpaceTimeStreams: 1
                     DCM: 0
          ChannelCoding: 'LDPC'
                   STAID: 0
      NominalPacketPadding: 0
      PostFECPaddingSource: 'mt19937ar with seed'
        PostFECPaddingSeed: 1

    Read-only properties:
                RUNumber: 1
```

The number of users per RU, and mapping of users to RUs is determined by the allocation index. The `UserNumbers` property of an RU object indicates which users (elements of the `cfgMU.User` cell array) are transmitted on that RU. Similarly, the `RUNumber` property of each User object, indicates which RU (element of the `cfgMU.RU` cell array) is used to transmit the user:
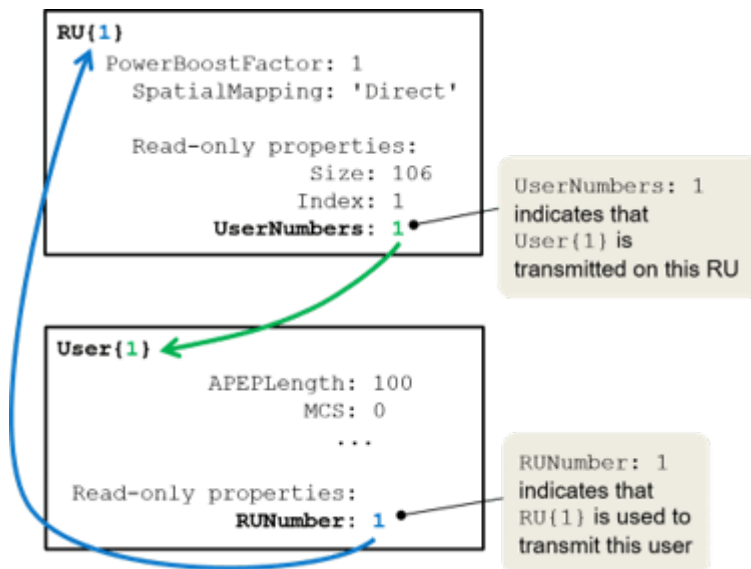
This allows the properties of an RU associated with a User to be accessed easily:

```
ruNum = cfgMU.User{2}.RUNumber; % Get the RU number associated with user 2
disp(cfgMU.RU{ruNum}.SpatialMapping); % Display the spatial mapping
```

```
Direct
```

When an RU serves multiple users, in a MU-MIMO configuration, the `UserNumbers` property can index multiple users:
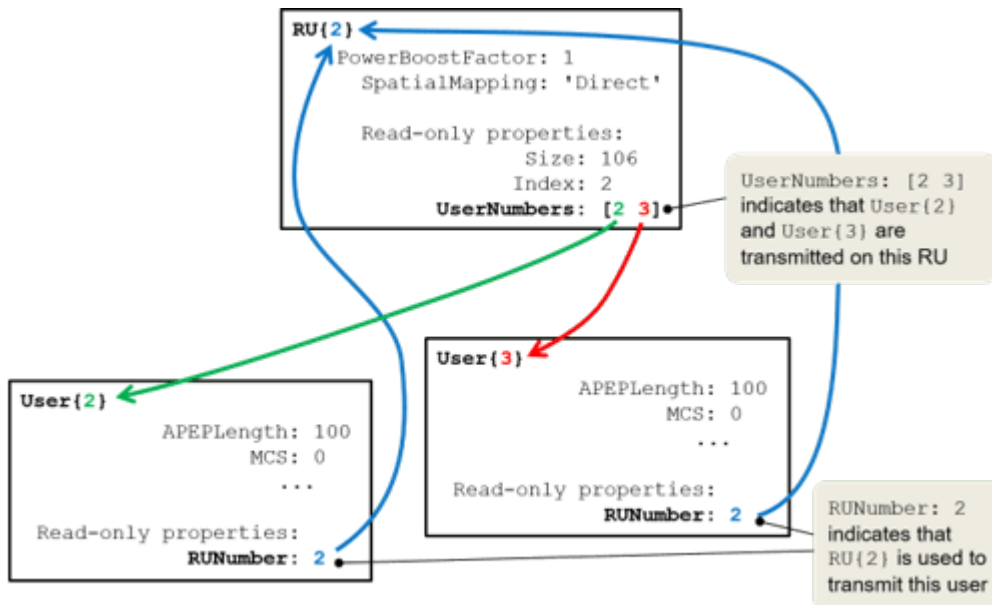
Once the `cfgMU` object is created, transmission parameters can be set as demonstrated below.

```
% Configure RU 1 and user 1
cfgMU.RU{1}.SpatialMapping = 'Direct';
cfgMU.User{1}.APEPLength = 1e3;
cfgMU.User{1}.MCS = 2;
cfgMU.User{1}.NumSpaceTimeStreams = 4;
cfgMU.User{1}.ChannelCoding = 'LDPC';

% Configure RU 2 and user 2
cfgMU.RU{2}.SpatialMapping = 'Fourier';
cfgMU.User{2}.APEPLength = 500;
cfgMU.User{2}.MCS = 3;
cfgMU.User{2}.NumSpaceTimeStreams = 2;
cfgMU.User{2}.ChannelCoding = 'LDPC';

% Configure RU 3 and user 3
cfgMU.RU{3}.SpatialMapping = 'Fourier';
cfgMU.User{3}.APEPLength = 100;
cfgMU.User{3}.MCS = 4;
cfgMU.User{3}.DCM = true;
cfgMU.User{3}.NumSpaceTimeStreams = 1;
cfgMU.User{3}.ChannelCoding = 'BCC';
```

Some transmission parameters are common for all users in the HE MU transmission.

```
% Configure common parameters for all users
cfgMU.NumTransmitAntennas = 4;
cfgMU.SIGBMCS = 2;
```

To generate the HE MU waveform, we first create a random PSDU for each user. A cell array is used to store the PSDU for each user as the PSDU lengths differ. The `getPSDULength()` method returns a vector with the required PSDU per user given the configuration. The waveform generator is then used to create a packet.

```
psduLength = getPSDULength(cfgMU);
psdu = cell(1,allocInfo.NumUsers);
for i = 1:allocInfo.NumUsers
    psdu{i} = randi([0 1],psduLength(i)*8,1,'int8'); % Generate random PSDU
end

% Create MU packet
txMUWaveform = wlanWaveformGenerator(psdu,cfgMU);
```

To configure an OFDMA transmission with a channel bandwidth greater than 20 MHz, an allocation index must be provided for each 20 MHz subchannel. For example, to configure an 80 MHz OFDMA transmission, four allocation indices are required. In this example four 242-tone RUs are configured. The allocation index 192 specifies one 242-tone RU with a single user in a 20 MHz subchannel, therefore the allocation indices [192 192 192 192] are used to create four of these RUs, over 80 MHz:

```
% Display 192 allocation index properties in the table (the 193rd row)
disp('Allocation #192 table entry:')
disp(allocationTable(193,:))

% Create 80 MHz MU configuration, with four 242-tone RUs
cfgMU80MHz = wlanHEMUConfig([192 192 192 192]);
```

```
Allocation #192 table entry:
    Allocation    BitAllocation    NumUsers    NumRUs    RUIndices    RUSizes    NumUsersPerRU
    _____    _____    _____    _____    _____    _____    _____

       192          "11000000"        1          1         {[1]}      {[242]}        {[1]}
```

When multiple 20 MHz subchannels are specified, the ChannelBandwidth property is set to the appropriate value. For this configuration it is set to 'CBW80' as four 20 MHz subchannels are specified. This is also visible in the allocation plot.

```
disp('Channel bandwidth for HE MU allocation:')
disp(cfgMU80MHz.ChannelBandwidth)
showAllocation(cfgMU80MHz,axAlloc)

Channel bandwidth for HE MU allocation:
CBW80
```
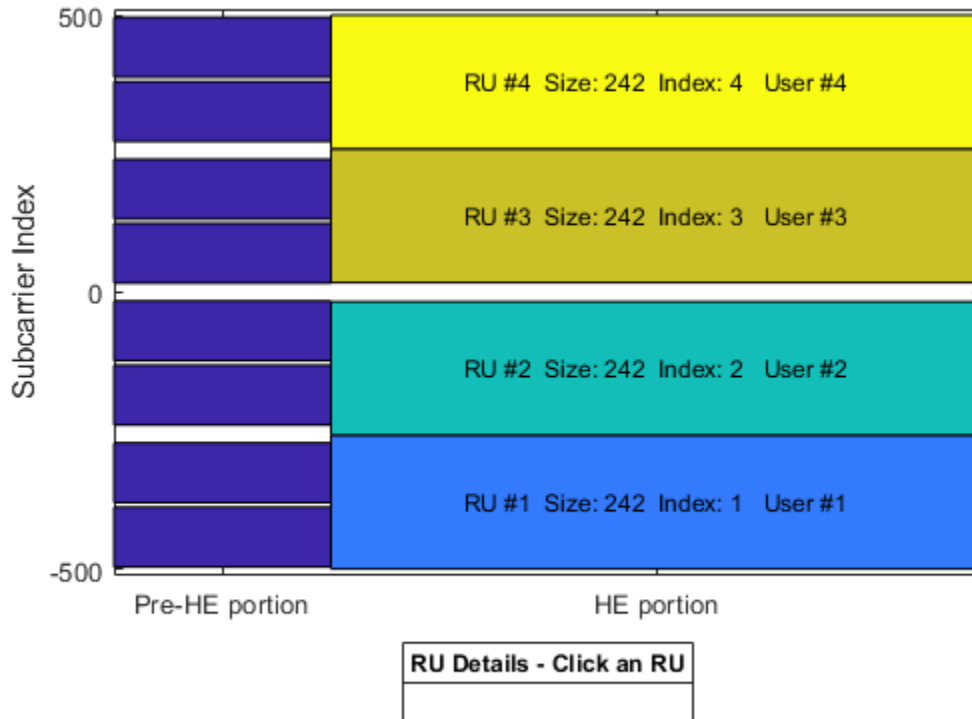
HE Multi-User Format - MU-MIMO

An HE MU packet can also transmit an RU to multiple users using MU-MIMO. For a full band MU-MIMO allocation, the allocation indices between 192 and 199 configure a full-band 20 MHz allocation (242-tone RU). The index within this range determines how many users are configured. The allocation details can be viewed in the allocation table. Note the `NumUsers` column in the table grows with index but the `NumRUs` is always 1. The allocation table can also be viewed in the Appendix.

```
disp('Allocation #192-199 table entries:')
disp(allocationTable(193:200,:)) % Indices 192-199 (rows 193 to 200)
```

```
Allocation #192-199 table entries:
    Allocation    BitAllocation    NumUsers    NumRUs    RUIndices    RUSizes    NumUsersPerRU
    _____    _____    _____    _____    _____    _____    _____

       192         "11000000"         1          1         {[1]}      {[242]}        {[1]}
       193         "11000001"         2          1         {[1]}      {[242]}        {[2]}
       194         "11000010"         3          1         {[1]}      {[242]}        {[3]}
       195         "11000011"         4          1         {[1]}      {[242]}        {[4]}
       196         "11000100"         5          1         {[1]}      {[242]}        {[5]}
       197         "11000101"         6          1         {[1]}      {[242]}        {[6]}
       198         "11000110"         7          1         {[1]}      {[242]}        {[7]}
       199         "11000111"         8          1         {[1]}      {[242]}        {[8]}
```

The allocation index 193 transmits a 20 MHz 242-tone RU to two users. In this example, we will create a transmission with a random spatial mapping matrix which maps a single space-time stream for each user, onto two transmit antennas.

```
% Configure 2 users in a 20 MHz channel
cfgMUMIMO = wlanHEMUConfig(193);

% Set the transmission properties of each user
cfgMUMIMO.User{1}.APEPLength = 100; % Bytes
cfgMUMIMO.User{1}.MCS = 2;
cfgMUMIMO.User{1}.ChannelCoding = 'LDPC';
cfgMUMIMO.User{1}.NumSpaceTimeStreams = 1;

cfgMUMIMO.User{2}.APEPLength = 1000; % Bytes
cfgMUMIMO.User{2}.MCS = 6;
cfgMUMIMO.User{2}.ChannelCoding = 'LDPC';
cfgMUMIMO.User{2}.NumSpaceTimeStreams = 1;

% Get the number of occupied subcarriers in the RU
ruIndex = 1; % Get the info for the first (and only) RU
ofdmInfo = wlanHEOFDMInfo('HE-Data',cfgMUMIMO,ruIndex);
numST = ofdmInfo.NumTones; % Number of occupied subcarriers

% Set the number of transmit antennas and generate a random spatial mapping
% matrix
numTx = 2;
allocInfo = ruInfo(cfgMUMIMO);
numSTS = allocInfo.NumSpaceTimeStreamsPerRU(ruIndex);
cfgMUMIMO.NumTransmitAntennas = numTx;
cfgMUMIMO.RU{ruIndex}.SpatialMapping = 'Custom';
cfgMUMIMO.RU{ruIndex}.SpatialMappingMatrix = rand(numST,numSTS,numTx);

% Create packet with a repeated bit sequence as the PSDU
txMUMIMOWaveform = wlanWaveformGenerator([1 0 1 0],cfgMUMIMO);
```

A full band MU-MIMO transmission with a channel bandwidth greater than 20 MHz is created by providing a single RU allocation index within the range 200-223 when creating the wlanHEMUConfig object. For these allocations HE-SIG-B compression is used.

The allocation indices between 200 and 207 configure a full-band MU-MIMO 40 MHz allocation (484-tone RU). The index within this range determines how many users are configured. The allocation details can be viewed in the allocation table. Note the NumUsers column in the table grows with index but the NumRUs is always 1.

```
disp('Allocation #200-207 table entries:')
disp(allocationTable(201:208,:)) % Indices 200-207 (rows 201 to 208)
```

```
Allocation #200-207 table entries:
    Allocation    BitAllocation    NumUsers    NumRUs    RUIndices    RUSizes    NumUsersPerRU
    _____    _____    _____    _____    _____    _____    _____

       200         "11001000"          1          1        {[1]}      {[484]}        {[1]}
       201         "11001001"          2          1        {[1]}      {[484]}        {[2]}
       202         "11001010"          3          1        {[1]}      {[484]}        {[3]}
       203         "11001011"          4          1        {[1]}      {[484]}        {[4]}
       204         "11001100"          5          1        {[1]}      {[484]}        {[5]}
       205         "11001101"          6          1        {[1]}      {[484]}        {[6]}
       206         "11001110"          7          1        {[1]}      {[484]}        {[7]}
```

| 207 | "11001111" | 8 | 1 | {[1]} | {[484]} | {[8]} |

Similarly, the allocation indices between 208 and 215 configure a full-band MU-MIMO 80 MHz allocation (996-tone RU), and the allocation indices between 216 and 223 configure a full-band MU-MIMO 160 MHz allocation (2x996-tone RU).

As an example, the allocation index 203 specifies a 484-tone RU with 4 users:

```
cfg484MU = wlanHEMUConfig(203);
showAllocation(cfg484MU,axAlloc)
```
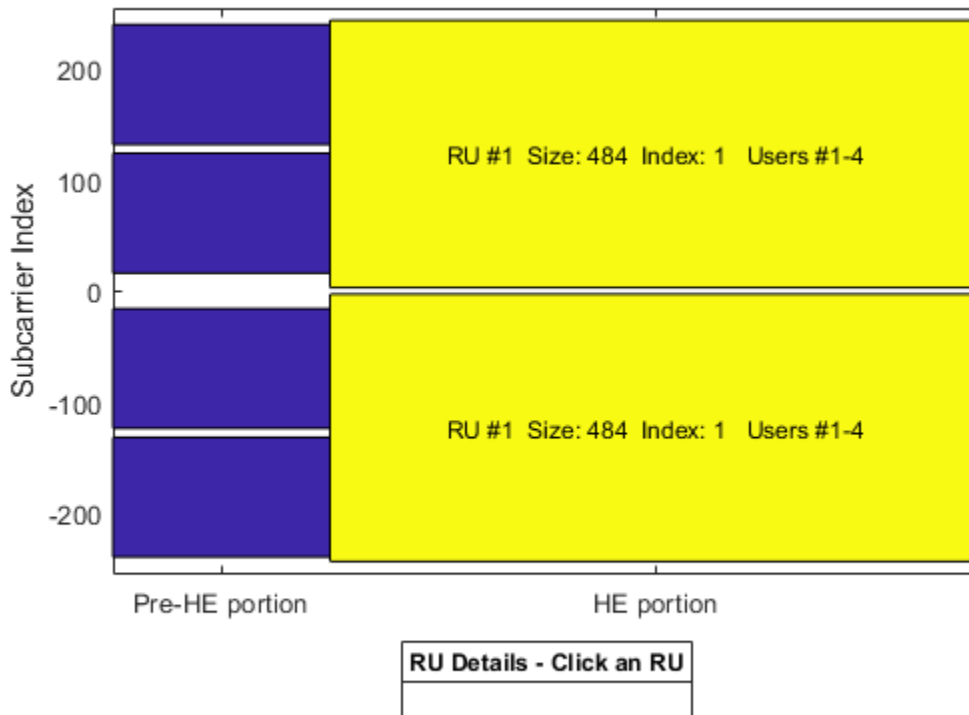

HE Multi-User Format - OFDMA with RU Sizes Greater Than 242 Subcarriers

For an HE MU transmission with a channel bandwidth greater than 20 MHz, two HE-SIG-B content channels are used to signal user configurations. These content channels are duplicated over each 40 MHz subchannel for larger channel bandwidths, as described in Section 27.3.10.8.3 of [1]. When an RU size greater than 242 is specified as part of an OFDMA system, the users assigned to the RU can be signaled on either of the two HE-SIG-B content channels. The allocation index provided when creating an wlanHEMUConfig object controls which content channel each user is signaled on. The allocation table in the Appendix shows the relevant allocation indices.

As an example, consider the following 80 MHz configuration which serves 7 users:

- One 484-tone RU (RU #1) with four users (users #1-4)
- One 242-tone RU (RU #2) with one user (user #5)

1-35

- Two 106-tone RUs (RU #3 and #4), each with one user (users #6 and #7)

To configure an 80 MHz OFDMA transmission, four allocation indices are required, one for each 20 MHz subchannel. To configure the above scenario the allocation indices below are used:

```
[X Y 192 96]
```

- X and Y configure the 484-tone RU, with users #1-4. The possible values of X and Y are discussed below.
- 192 configures a 242-tone RU with one user, user #5.
- 96 signals two 106-tone RUs, each with one user, users #6 and #7.

The selection of X and Y configures the appropriate number of users in the 242-tone RU, and determines which HE-SIG-B content channel is used to signal the users. A 484-tone RU spans two 20 MHz subchannels, therefore two allocation indices are required. All seven users from the four RUs will be signaled on the HE-SIG-B content channels, but for now we will only consider the signaling of users on the 484-tone RU. For the 484-tone RU, the four users can be signaled on the two HE-SIG-B content channels in different combinations as shown in Table 1.

Combination	Content Channel 1	Content Channel 2
A	User 1	Users 2-4
B	Users 1-2	Users 3-4
C	Users 1-3	User 4
D	Users 1-4	No users
E	No users	Users 1-4

Table 1

An allocation index within the range 200-207 specifies 1-8 users on a 484-tone RU. To signal no users on a content channel, the allocation index 114 or 115 can be used, for a 448-tone or 996-tone RU. Therefore, the combinations in Table 1 can be defined using two allocation indices as shown in Table 2. The two allocation indices in each row of Table 2 are X and Y.

Combination	484-Tone RU Allocation Index	Number of Users Per Content Channel	
		Content Channel 1	Content Channel 2
A	[200 202]	1	3
B	[201 201]	2	2
C	[202 200]	3	1
D	[203 114]	4	0
E	[114 203]	0	4

Table 2

Therefore, to configure 'Combination E' the following 80 MHz allocation indices are used:

```
[114 203 192 96]
```

- 114 and 203 configure the 484-tone RU, with users #1-4.
- 192 configures a 242-tone RU with one user, user #5.

- **96** signals two 106-tone RUs, each with one user, users #6 and #7.

```
cfg4840FDMA = wlanHEMUConfig([114 203 192 96]);
showAllocation(cfg4840FDMA,axAlloc);
```
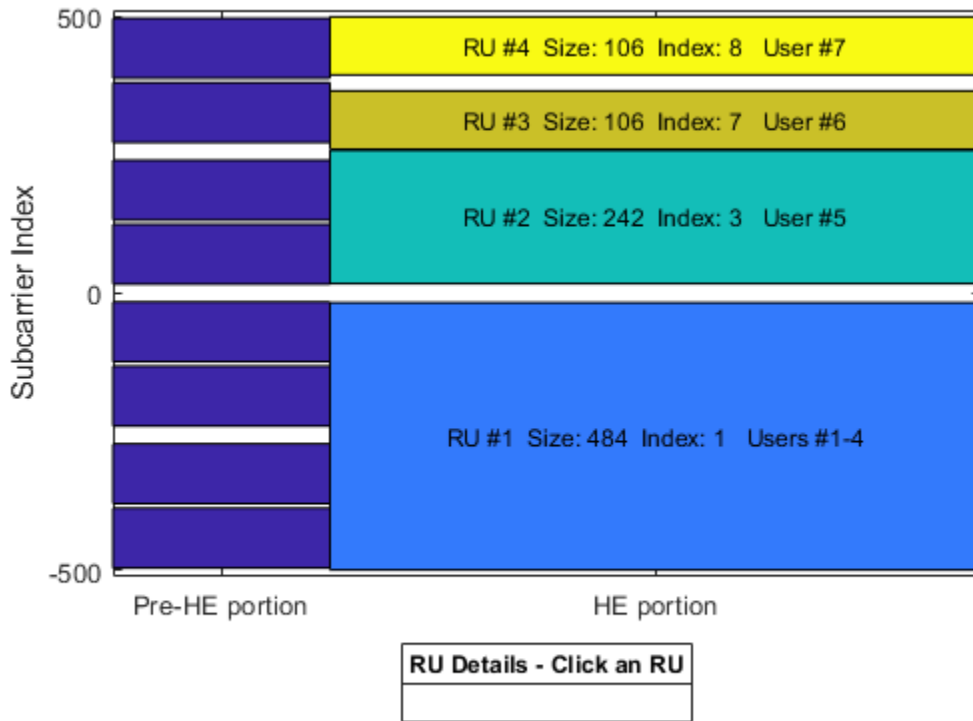

To view the HE-SIG-B allocation signaling, use the `hePlotHESIGBAllocationMapping` function. This shows the user fields signaled on each HE-SIG-B content channel, and which RU and user in the `wlanHEMUConfig` object, each user field signals. In this case we can see the users on RU #1, 3 and 4 are all signaled on content channel 2, and the user of RU #2 is signaled on content channel 1. The second content channel signals six users, while the first content channel only signals one user. Therefore, the first content channel will be padded up to the length of the second for transmission. In the diagram, the RU allocation information is provided in the form index-size, e.g. RU8-106 is the 8th 106-tone RU.

```
figure;
hePlotHESIGBAllocationMapping(cfg4840FDMA);
axSIGB = gca; % Get axis handle for subsequent plotting
```
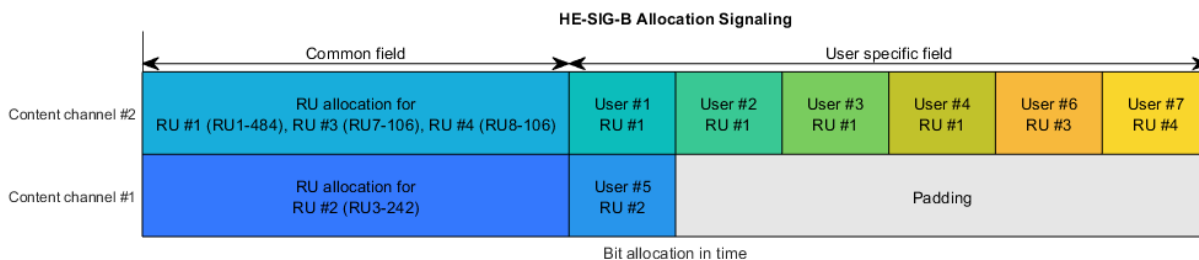
To balance the user field signaling in HE-SIG-B, we can use 'Combination B' in Table 2 when creating the allocation index for the 484-tone RU. This results in two users being signaled on each content channel of HE-SIG-B, creating a better balance of user fields, and potentially fewer HE-SIG-B symbols in the transmission.

```
cfg484OFDMABalanced = wlanHEMUConfig([201 201 96 192]);
hePlotHESIGBAllocationMapping(cfg484OFDMABalanced,axSIGB);
```

HE-SIG-B Allocation Signaling

	Common field		User specific field			
Content channel #2	RU allocation for RU #1 (RU1-484), RU #4 (RU4-242)		User #3 RU #1	User #4 RU #1	User #7 RU #4	Padding
Content channel #1	RU allocation for RU #1 (RU1-484), RU #2 (RU5-106), RU #3 (RU6-106)		User #1 RU #1	User #2 RU #1	User #5 RU #2	User #6 RU #3

Bit allocation in time
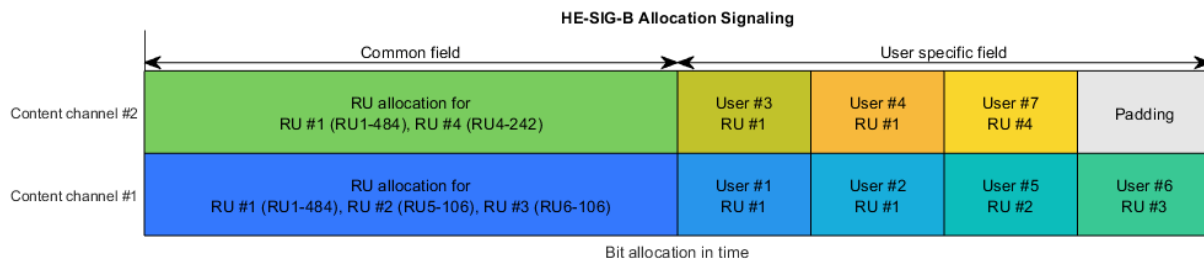
HE Multi-User Format - Central 26-Tone RU

In an 80 MHz transmission, when a full band RU is not used, the central 26-tone RU can be optionally active. The central 26-tone RU is enabled using a name-value pair when creating the `wlanHEMUConfig` object.

```
% Create a configuration with no central 26-tone RU
cfgNoCentral = wlanHEMUConfig([192 192 192 192],'LowerCenter26ToneRU',false);
showAllocation(cfgNoCentral,axAlloc);

% Create a configuration with a central 26-tone RU
cfgCentral = wlanHEMUConfig([192 192 192 192],'LowerCenter26ToneRU',true);
showAllocation(cfgCentral,axAlloc);
```
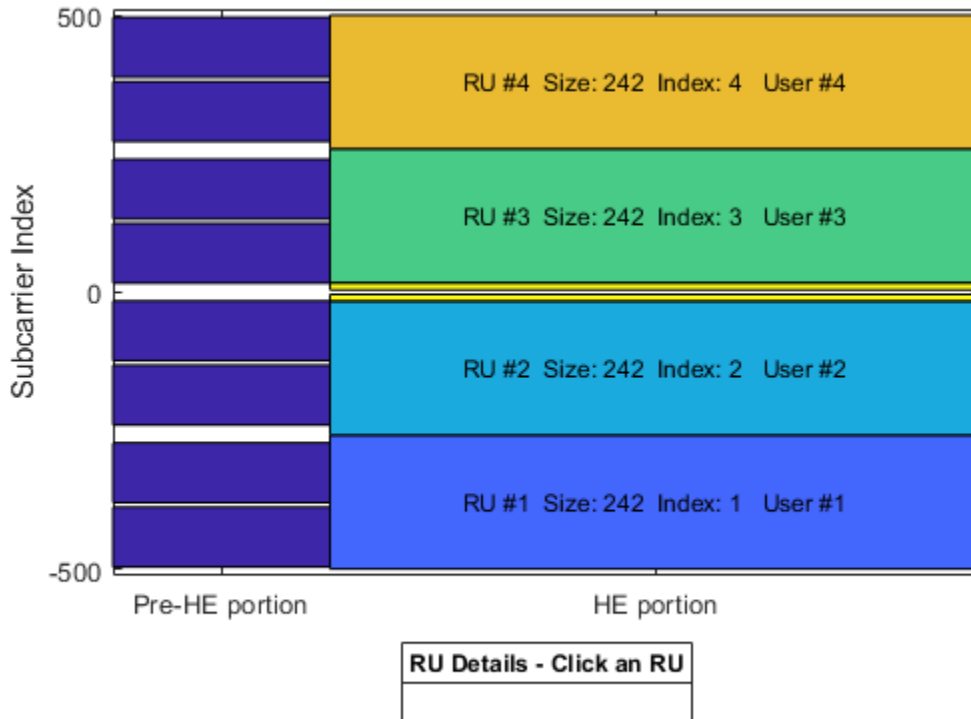
Similarly, for a 160 MHz transmission, the central 26-tone RU in each 80 MHz segment can be optionally used. Each central 26-tone RU can be enabled using name-value pairs when creating the `wlanHEMUConfig` object. In this example only the upper central 26-tone RU is created. Four 242-tone RUs, each with one user are specified with the allocation index [200 114 114 200 200 114 114 200].

```
cfgCentral160MHz = wlanHEMUConfig([200 114 114 200 200 114 114 200],'UpperCenter26ToneRU',true);
disp(cfgCentral160MHz)
```

```
  wlanHEMUConfig with properties:

                   RU: {1x5 cell}
                 User: {1x5 cell}
     PrimarySubchannel: 1
   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: 'CBW160'
         AllocationIndex: [200 114 114 200 200 114 114 200]
     LowerCenter26ToneRU: 0
     UpperCenter26ToneRU: 1
```

HE Multi-User Format - Preamble Puncturing

In an 80 MHz or 160 MHz transmission, 20 MHz subchannels can be punctured to allow a legacy system to operate in the punctured channel. This method is also described as channel bonding. To null a 20 MHz subchannel the 20 MHz subchannel allocation index `113` can be used. The punctured 20 MHz subchannel can be viewed with the `showAllocation` method.

```
% Null second lowest 20 MHz subchannel in a 160 MHz configuration
cfgNull = wlanHEMUConfig([192 113 114 200 208 115 115 115]);

% Plot the allocation
showAllocation(cfgNull,axAlloc);
```
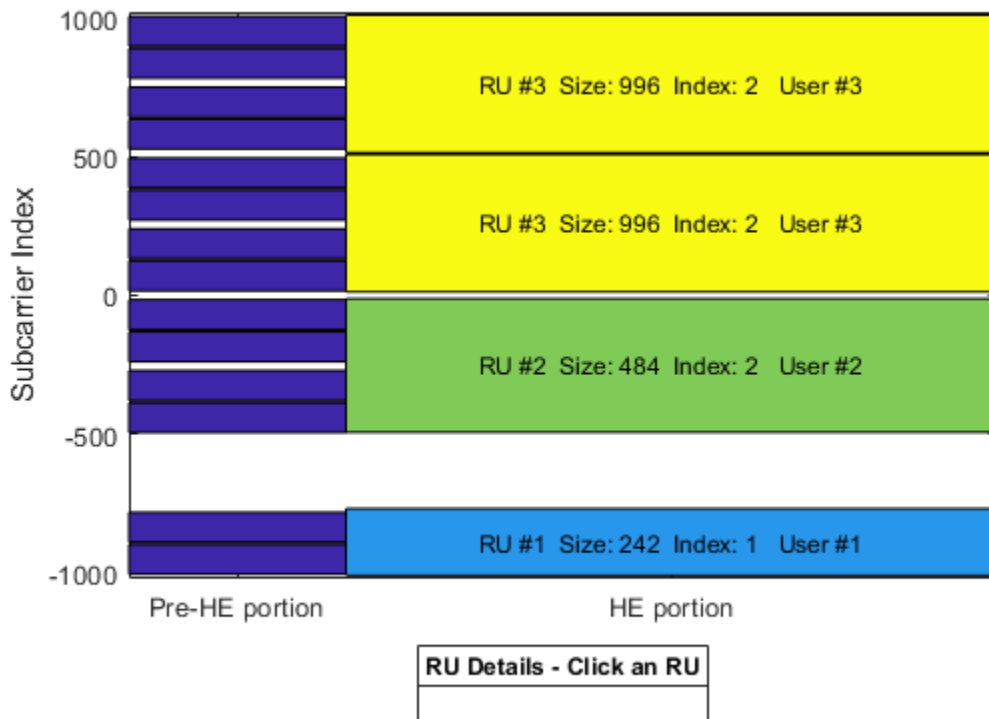

The punctured 20 MHz can also be viewed with the generated waveform and the spectrum analyzer.

```
% Set the transmission properties of each user in all RUs
cfgNull.User{1}.APEPLength = 100;
cfgNull.User{1}.MCS = 2;
cfgNull.User{1}.ChannelCoding = 'LDPC';
cfgNull.User{1}.NumSpaceTimeStreams = 1;

cfgNull.User{2}.APEPLength = 1000;
```

```
cfgNull.User{2}.MCS = 6;
cfgNull.User{2}.ChannelCoding = 'LDPC';
cfgNull.User{2}.NumSpaceTimeStreams = 1;

cfgNull.User{3}.APEPLength = 100;
cfgNull.User{3}.MCS = 1;
cfgNull.User{3}.ChannelCoding = 'LDPC';
cfgNull.User{3}.NumSpaceTimeStreams = 1;

% Create packet
txNullWaveform = wlanWaveformGenerator([1 0 1 0],cfgNull);

% Visualize signal spectrum
fs = wlanSampleRate(cfgNull);
ofdmInfo = wlanHEOFDMInfo('HE-Data',cfgNull,1);
fftsize = ofdmInfo.FFTLength;
spectrumAnalyzer = dsp.SpectrumAnalyzer('SampleRate',fs,...
                'AveragingMethod','Exponential','ForgettingFactor',0.99,...
                'RBWSource','Property','RBW',fs/fftsize,...
                'Title','160 MHz HE MU Transmission with Punctured 20 MHz Channel');
spectrumAnalyzer(txNullWaveform);
```
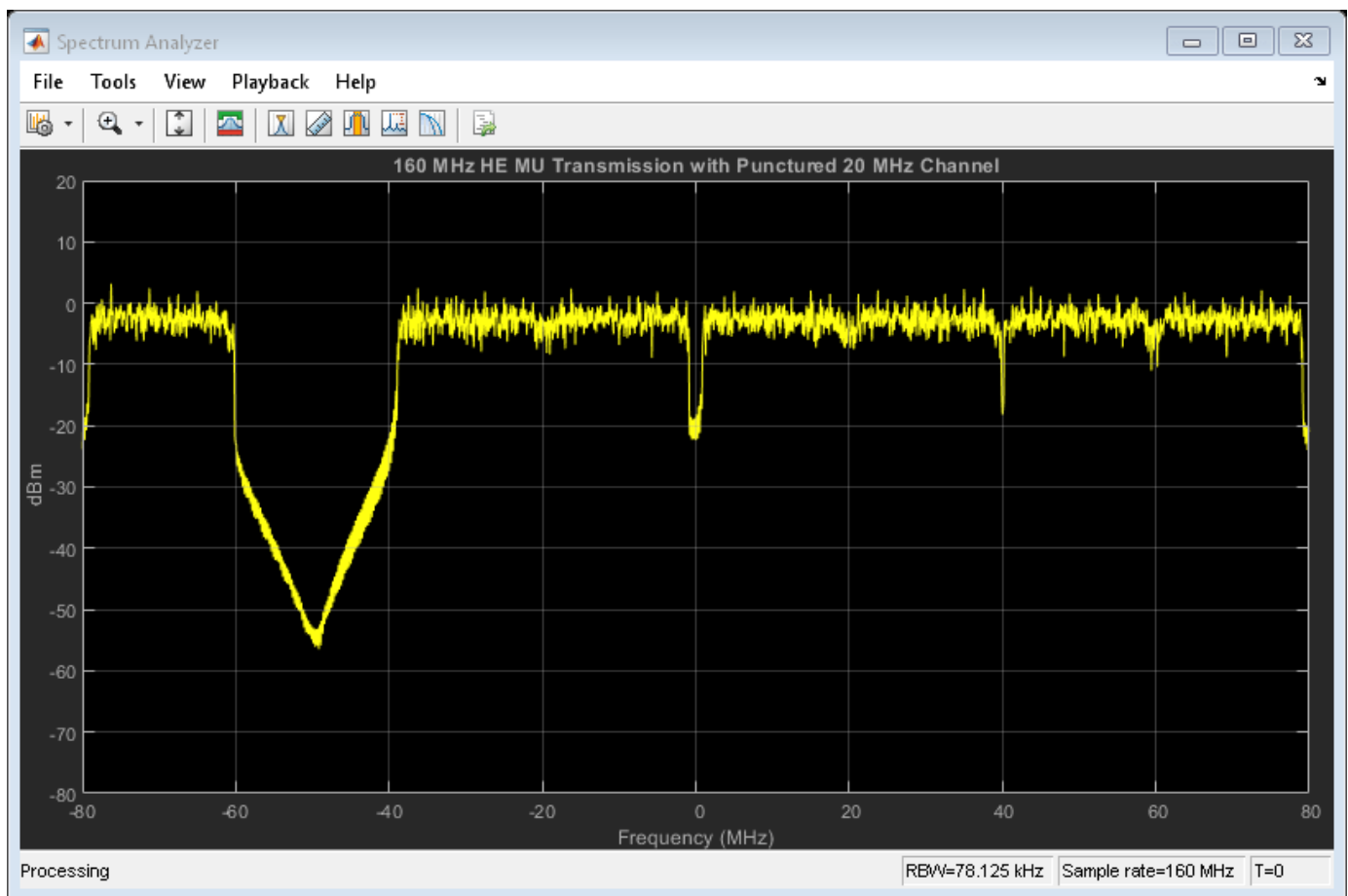
Trigger-Based MU Format

The HE trigger-based (TB) format allows for OFDMA or MU-MIMO transmission in the uplink. Each station (STA) transmits a TB packet simultaneously, when triggered by the access point (AP). A TB transmission is controlled entirely by the AP. All the parameters required for the transmission are provided in a trigger frame to all STAs participating in the TB transmission. In this example a TB transmission in response to a trigger frame for three users in an OFDMA/MU-MIMO system is configured; three STAs will transmit simultaneously to an AP.

The 20 MHz allocation 97 is used which corresponds to two RUs, one of which serves two users in MU-MIMO.

```
disp('Allocation #97 table entry:')
disp(allocationTable(98,:)) % Index 97 (row 98)
```

Allocation #97 table entry:

Allocation	BitAllocation	NumUsers	NumRUs	RUIndices	RUSizes	NumUsersPerF
97	"01100001"	3	2	{[1 2]}	{[106 106]}	{[1 2]}

The allocation information is obtained by creating a MU configuration with `wlanHEMUConfig`.

```
% Generate an OFDMA allocation
cfgMU = wlanHEMUConfig(97);
allocationInfo = ruInfo(cfgMU);
```

In a TB transmission several parameters are the same for all users in the transmission. Some of these are specified below:

```
% These parameters are the same for all users in the OFDMA system
trgMethod = 'TriggerFrame'; % Method used to trigger an HE TB PPDU
channelBandwidth = cfgMU.ChannelBandwidth; % Bandwidth of OFDMA system
lsigLength = 142;           % L-SIG length
preFECPaddingFactor = 2;    % Pre-FEC padding factor
ldpcExtraSymbol = false;    % LDPC extra symbol
numHELTFSymbols = 2;        % Number of HE-LTF symbols
```

A TB transmission for a single user within the system is configured with a `wlanHETBConfig` object. In this example, a cell array of three objects is created to describe the transmission of the three users.

```
% Create a trigger configuration for each user
numUsers = allocationInfo.NumUsers;
cfgTriggerUser = repmat({wlanHETBConfig},1,numUsers);
```

The non-default system-wide properties are set for each user.

```
for userIdx = 1:numUsers
    cfgTriggerUser{userIdx}.TriggerMethod = trgMethod;
    cfgTriggerUser{userIdx}.ChannelBandwidth = channelBandwidth;
    cfgTriggerUser{userIdx}.LSIGLength = lsigLength;
    cfgTriggerUser{userIdx}.PreFECPaddingFactor = preFECPaddingFactor;
    cfgTriggerUser{userIdx}.LDPCExtraSymbol = ldpcExtraSymbol;
    cfgTriggerUser{userIdx}.NumHELTFSymbols = numHELTFSymbols;
end
```

Next the per-user properties are set. When multiple users are transmitting in the same RU, in a MU-MIMO configuration, each user must transmit on different space-time stream indices. The properties `StartingSpaceTimeStream` and `NumSpaceTimeStreamSteams` must be set for each user to make sure different space-time streams are used. In this example user 1 and 2 are in a MU-MIMO configuration, therefore `StartingSpaceTimeStream` for user two is set to 2, as user one is configured to transmit 1 space-time stream with `StartingSpaceTimeStream = 1`.

```
% These parameters are for the first user - RU#1 MU-MIMO user 1
cfgTriggerUser{1}.RUSize = allocationInfo.RUSizes(1);
cfgTriggerUser{1}.RUIndex = allocationInfo.RUIndices(1);
cfgTriggerUser{1}.MCS = 4;                          % Modulation and coding scheme
cfgTriggerUser{1}.NumSpaceTimeStreams = 1;      % Number of space-time streams
cfgTriggerUser{1}.NumTransmitAntennas = 1;      % Number of transmit antennas
cfgTriggerUser{1}.StartingSpaceTimeStream = 1; % The starting index of the space-time streams
cfgTriggerUser{1}.ChannelCoding = 'LDPC';       % Channel coding

% These parameters are for the second user - RU#1 MU-MIMO user 2
cfgTriggerUser{2}.RUSize = allocationInfo.RUSizes(1);
cfgTriggerUser{2}.RUIndex = allocationInfo.RUIndices(1);
cfgTriggerUser{2}.MCS = 3;                          % Modulation and coding scheme
cfgTriggerUser{2}.NumSpaceTimeStreams = 1;      % Number of space-time streams
cfgTriggerUser{2}.StartingSpaceTimeStream = 2; % The starting index of the space-time streams
cfgTriggerUser{2}.NumTransmitAntennas = 1;      % Number of transmit antennas
cfgTriggerUser{2}.ChannelCoding = 'LDPC';       % Channel coding

% These parameters are for the third user - RU#2
cfgTriggerUser{3}.RUSize = allocationInfo.RUSizes(2);
cfgTriggerUser{3}.RUIndex = allocationInfo.RUIndices(2);
cfgTriggerUser{3}.MCS = 4;                          % Modulation and coding scheme
cfgTriggerUser{3}.NumSpaceTimeStreams = 2;      % Number of space-time streams
cfgTriggerUser{3}.StartingSpaceTimeStream = 1; % The starting index of the space-time streams
cfgTriggerUser{3}.NumTransmitAntennas = 2;      % Number of transmit antennas
cfgTriggerUser{3}.ChannelCoding = 'BCC';        % Channel coding
```

A packet containing random data is now transmitted by each user with `wlanWaveformGenerator`. The waveform transmitted by each user is stored for analysis.

```
trigInd = wlanFieldIndices(cfgTriggerUser{1}); % Get the indices of each field
txTrigStore = zeros(trigInd.HEData(2),numUsers);
for userIdx = 1:numUsers
    % Generate waveform for a user
    cfgTrigger = cfgTriggerUser{userIdx};
    txPSDU = randi([0 1],getPSDULength(cfgTrigger)*8,1);
    txTrig = wlanWaveformGenerator(txPSDU,cfgTrigger);

    % Store the transmitted STA waveform for analysis
    txTrigStore(:,userIdx) = sum(txTrig,2);
end
```

The spectrum of the transmitted waveform from each STA shows the different portions of the spectrum used, and the overlap in the MU-MIMO RU.

```
fs = wlanSampleRate(cfgTriggerUser{1});
ofdmInfo = wlanHEOFDMInfo('HE-Data',cfgTriggerUser{1});
spectrumAnalyzer = dsp.SpectrumAnalyzer('SampleRate',fs,...
            'AveragingMethod','Running','SpectralAverages',1,...
            'ChannelNames', {'RU#1 User 1','RU#1 User 2','RU#2'},...
```

```
                  'ShowLegend',true,'Title','Transmitted HE TB Waveform per User');
spectrumAnalyzer(txTrigStore);
```
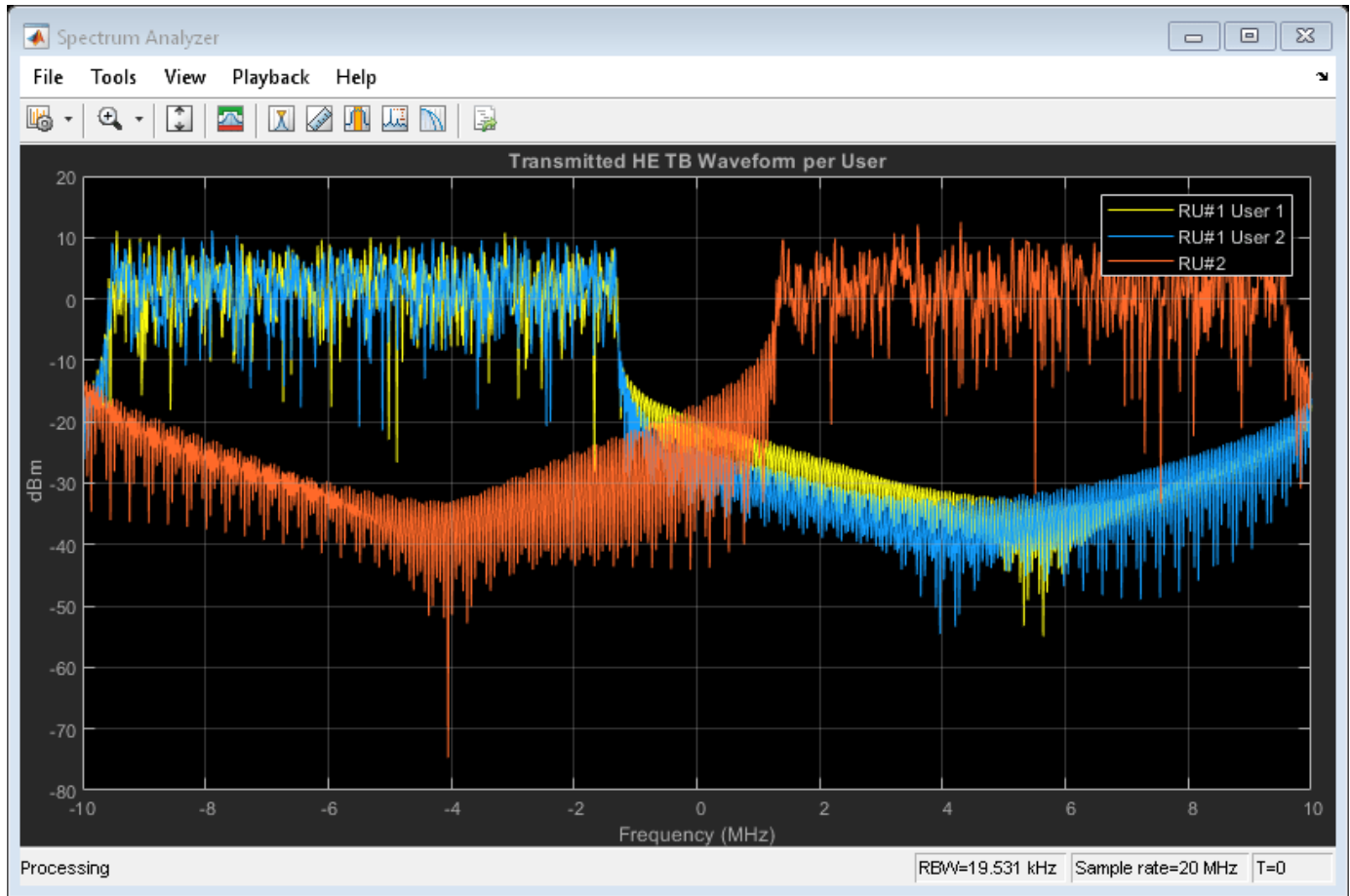

Appendix

The RU allocation table for allocations <= 20 MHz is shown below, with annotated descriptions.

Allocation Index	20 MHz Subchannel Resource Unit (RU) Assignment								
0	26	26	26	26	26	26	26	26	26
1	26	26	26	26	26	26	26	52	
2	26	26	26	26	26	52		26	26
3	26	26	26	26	26	52		52	
4	26	26	52		26	26	26	26	26
5	26	26	52		26	26	26	52	
6	26	26	52		26	52		26	26
7	26	26	52		26	52		52	
8	52		26	26	26	26	26	26	26
9	52		26	26	26	26	26	52	
10	52		26	26	26	52		26	26
11	52		26	26	26	52		52	
12	52		52		26	26	26	26	26
13	52		52		26	26	26	52	
14	52		52		26	52		26	26
15	52		52		26	52		52	
16-23 (15 + N)	52		52		-	106 (N users)			
24-31 (23 + N)	106 (N users)				-	52		52	
32-39 (31 + N)	26	26	26	26	26	106 (N users)			
40-47 (39 + N)	26	26	52		26	106 (N users)			
48-55 (47 + N)	52		26	26	26	106 (N users)			
56-63 (55 + N)	52		52		26	106 (N users)			
64-71 (63 + N)	106 (N users)				26	26	26	26	26
72-79 (71 + N)	106 (N users)				26	26	26	52	
80-87 (79 + N)	106 (N users)				26	52		26	26
88-95 (87 + N)	106 (N users)				26	52		52	
96-99 (95 + M)	106				-	106 (M users)			
100-103 (99 + M)	106 (2 users)				-	106 (M users)			
104-107 (103 + M)	106 (3 users)				-	106 (M users)			
108-111 (107 + M)	106 (4 users)				-	106 (M users)			
112	52		52		-	52		52	
113	Empty 242-tone RU - No user assigned								
116-127	Reserved								
128-135 (127 + N)	106				26	106 (N users)			
136-143 (135 + N)	106 (2 users)				26	106 (N users)			
144-151 (143 + N)	106 (3 users)				26	106 (N users)			
152-159 (151 + N)	106 (4 users)				26	106 (N users)			
160-167 (159 + N)	106 (5 users)				26	106 (N users)			
168-175 (167 + N)	106 (6 users)				26	106 (N users)			
176-183 (175 + N)	106 (7 users)				26	106 (N users)			
184-191 (183 + N)	106 (8 users)				26	106 (N users)			
192-199 (191 + N)	242 (N users)								

Annotations:

- 26 tone RU assigned to 1 user as part of a 20 MHz subchannel assignment of 9 26-tone RUs
- No users assigned to this RU; no data field transmitted on these subcarriers
- The number of users (N) assigned to this 106-tone RU depends on the allocation index and must be 1-8.
- The number of users (M) assigned to this 106-tone RU depends on the allocation index and must be 1-4.
- The number of users assigned to the upper 106-tone RU depends on the allocation index, but 2 users are always assigned to the lower 106-tone RU
- If selected, this 20 MHz subchannel is unused; the subchannel is punctured

Legend:
- RU assigned to 1 user
- RU assigned to 1-4/8 users, depending on the allocation index
- RU assigned to specified number of users, irrespective of the allocation index

The RU allocation and HE-SIG-B user signaling for allocations > 20 MHz is shown in the table below, with annotated descriptions.

Allocation Index	RU Allocation & Number of Users on the Corresponding HE-SIG-B Content Channel for RU Size > 242
114	484-tone RU with no users signaled on the corresponding HE-SIG-B content channel
115	996-tone RU with no users signaled on the corresponding HE-SIG-B content channel
200-207 (199 + N)	Full band 40 MHz (N users), or 484-tone RU with N users signaled in the corresponding HE-SIG-B content channel
208-215 (207 + N)	Full band 80 MHz (N users), or 996-tone RU with N users signaled in the corresponding HE-SIG-B content channel
216-223 (215 + N)	Full band 160 MHz (N users)
224-255	Reserved

Annotations:
- Must be used with other allocation indices. Signifies a 484-tone RU with zero users signaled on the corresponding HE-SIG-B content channel
- A single allocation index between 200-207 configures a full-band 40 MHz 484-tone RU with N users. N must be 1-8.

Legend:
- >242-tone RU allocation

Selected Bibliography

1 IEEE P802.11ax™/D4.1 Draft 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 6: Enhancements for High Efficiency WLAN.

Basic WLAN Link Modeling

This example shows how to create a basic WLAN link model using WLAN Toolbox™. An IEEE® 802.11ac™ [1] VHT packet is created, passed through a TGac channel. The received signal is equalized and decoded in order to recover the transmitted bits.

Introduction

This example shows how a simple transmitter-channel-receiver simulation may be created using functions from WLAN Toolbox. A VHT transmit and receive link is implemented as shown in the figure below. A VHT packet is transmitted through a TGac channel, demodulated and the equalized symbols are recovered. The equalized symbols are decoded to recover the transmitted bits.

Waveform Generation

An 802.11ac VHT transmission is simulated in this example. The transmit parameters for the VHT format of the 802.11™ standard are configured using a VHT configuration object. The `wlanVHTConfig` creates a VHT configuration object. In this example the object is configured for a 20 MHz channel bandwidth, MCS 5 and single transmit antenna.

```
% Create a format configuration object for a SISO VHT transmission
cfgVHT = wlanVHTConfig;
cfgVHT.NumTransmitAntennas = 1;    % Transmit antennas
cfgVHT.NumSpaceTimeStreams = 1;    % Space-time streams
cfgVHT.APEPLength = 4096;          % APEP length in bytes
cfgVHT.MCS = 5;                    % Single spatial stream, 64-QAM
cfgVHT.ChannelBandwidth = 'CBW20'; % Transmitted signal bandwidth
Rs = wlanSampleRate(cfgVHT);       % Sampling rate
```

A single VHT packet is generated consisting of training, signal and data fields:

- Non-HT Short Training Field (L-STF)
- Non-HT Long Training Field (L-LTF)
- Non-HT Signal (L-SIG) field
- VHT Signal A (VHT-SIG-A) field
- VHT Short Training Field (VHT-STF)
- VHT Long Training Field (VHT-LTF)
- VHT Signal B (VHT-SIG-B) field
- Data field

These fields are generated separately using functions from WLAN Toolbox and are concatenated to produce a VHT transmit packet.

The first field in the PPDU is the L-STF and is used for the start of packet detection and automatic gain control (AGC) setting. It is also used for initial frequency offset estimation and coarse timing synchronization. The `wlanLSTF` function generates the L-STF field in the time-domain using some of the parameters included in configuration object `cfgVHT`.

```
lstf = wlanLSTF(cfgVHT);
```

The L-LTF is used for fine time synchronization, channel estimation and fine frequency offset estimation. The docid:wlan_ref#buz04s0 wlanLLTF> function generates the L-LTF in the time-domain.

```
lltf = wlanLLTF(cfgVHT);
```

The L-SIG field carries packet configuration such as data rate, modulation and code rate for non-HT format. The `wlanLSIG` function generates the L-SIG field in the time-domain.

```
lsig = wlanLSIG(cfgVHT);
```

The figure below shows the L-STF, L-LTF and L-SIG fields. These fields are common to the VHT, HT-Mixed and non-HT OFDM transmission formats.

```
nonHTfield = [lstf;lltf;lsig]; % Combine the non-HT preamble fields
```
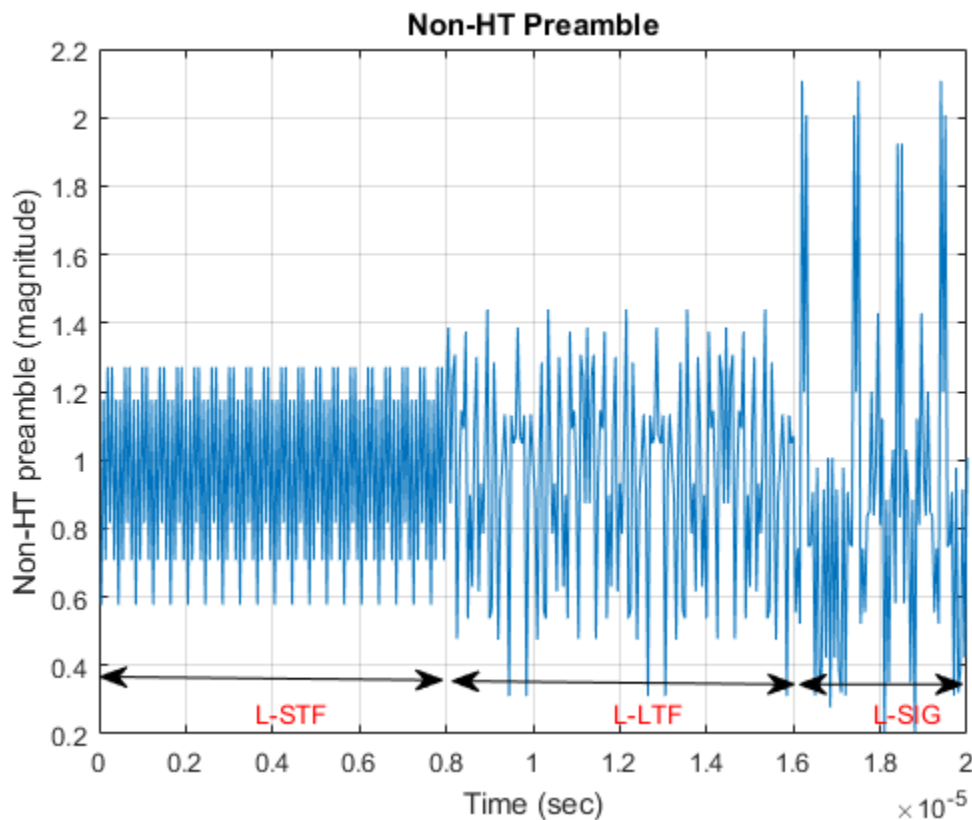

The VHT specific signal and training fields are generated after the non-HT preamble fields. The purpose of the VHT-SIG-A field is to provide information to allow the receiver to decode the data payload. The VHT-SIG-A is composed of two symbols VHT-SIG-A1 and VHT-SIG-A2. The `wlanVHTSIGA` function generates the VHT-SIG-A field in the time-domain.

```
vhtsiga = wlanVHTSIGA(cfgVHT);
```

The purpose of the VHT-STF is to improve the gain control estimation in a MIMO transmission and help the receiver detect the repeating pattern similar to the L-STF field. The `wlanVHTSTF` function generates the VHT-STF field in the time-domain.

```
vhtstf = wlanVHTSTF(cfgVHT);
```

The VHT-LTF provides a mean for the receiver to estimate the channel between the transmitter and the receiver. Depending on the number of space time streams, it consists of 1,2,4,6 or 8 VHT-LTF symbols. The `wlanVHTLTF` function generates the VHT-LTF in the time-domain.

```
vhtltf = wlanVHTLTF(cfgVHT);
```

The VHT-SIG-B field is used to set the data rate and the length of the data field payload of the transmitted packet. The `wlanVHTSIGB` function generates the VHT-SIG-B field in the time-domain.

```
vhtsigb = wlanVHTSIGB(cfgVHT);
```

Construct the preamble with the generated signal and training fields for the VHT format.

```
preamble = [lstf;lltf;lsig;vhtsiga;vhtstf;vhtltf;vhtsigb];
```

The `wlanVHTData` function generates the time-domain VHT data field. The VHT format configuration `cfgVHT` specifies the parameters for generating the data field from the PSDU bits. The `cfgVHT.PSDULength` property gives the number of bytes to be transmitted in the VHT data field. This property is used to generate the random PSDU bits `txPSDU`.

```
rng(0) % Initialize the random number generator
txPSDU = randi([0 1],cfgVHT.PSDULength*8,1); % Generate PSDU data in bits
data = wlanVHTData(txPSDU,cfgVHT);

% A VHT waveform is constructed by prepending the non-HT and VHT
% preamble fields with data
txWaveform = [preamble;data]; % Transmit VHT PPDU
```

Alternatively the waveform for a given format configuration can also be generated using a single function call `wlanWaveformGenerator` function. This function can produce one or more VHT packets. By default OFDM windowing is applied to the generated waveform. For more information on OFDM windowing, see the reference page for the `wlanWaveformGenerator` function.

Channel Impairments

This section simulates the effects of over-the-air transmission. The transmitted signal is impaired by the channel and AWGN. The level of the AWGN is given in dBs. In this example the TGac channel model [2] is used with delay profile Model-B. For this delay profile when the distance between transmitter and receiver is greater than or equal to 5 meters, the model is in Non-Line-of-Sight (N-LOS) configuration. This is described further in the help for `wlanTGacChannel`.

```
% Parameterize the channel
tgacChannel = wlanTGacChannel;
tgacChannel.DelayProfile = 'Model-B';
tgacChannel.NumTransmitAntennas = cfgVHT.NumTransmitAntennas;
tgacChannel.NumReceiveAntennas = 1;
tgacChannel.LargeScaleFadingEffect = 'None';
tgacChannel.ChannelBandwidth = 'CBW20';
tgacChannel.TransmitReceiveDistance = 5;
tgacChannel.SampleRate = Rs;
tgacChannel.RandomStream = 'mt19937ar with seed';
tgacChannel.Seed = 10;

% Pass signal through the channel. Append zeroes to compensate for channel
% filter delay
```

```
txWaveform = [txWaveform;zeros(10,1)];
chanOut = tgacChannel(txWaveform);

snr = 40; % In dBs
rxWaveform = awgn(chanOut,snr,0);

% Display the spectrum of the transmitted and received signals. The
% received signal spectrum is affected by the channel
spectrumAnalyzer  = dsp.SpectrumAnalyzer('SampleRate',Rs, ...
            'AveragingMethod','Exponential','ForgettingFactor',0.99, ...
            'YLimits',[-30 10],'ShowLegend',true, ...
            'ChannelNames',{'Transmitted waveform','Received waveform'});
spectrumAnalyzer([txWaveform rxWaveform]);
```
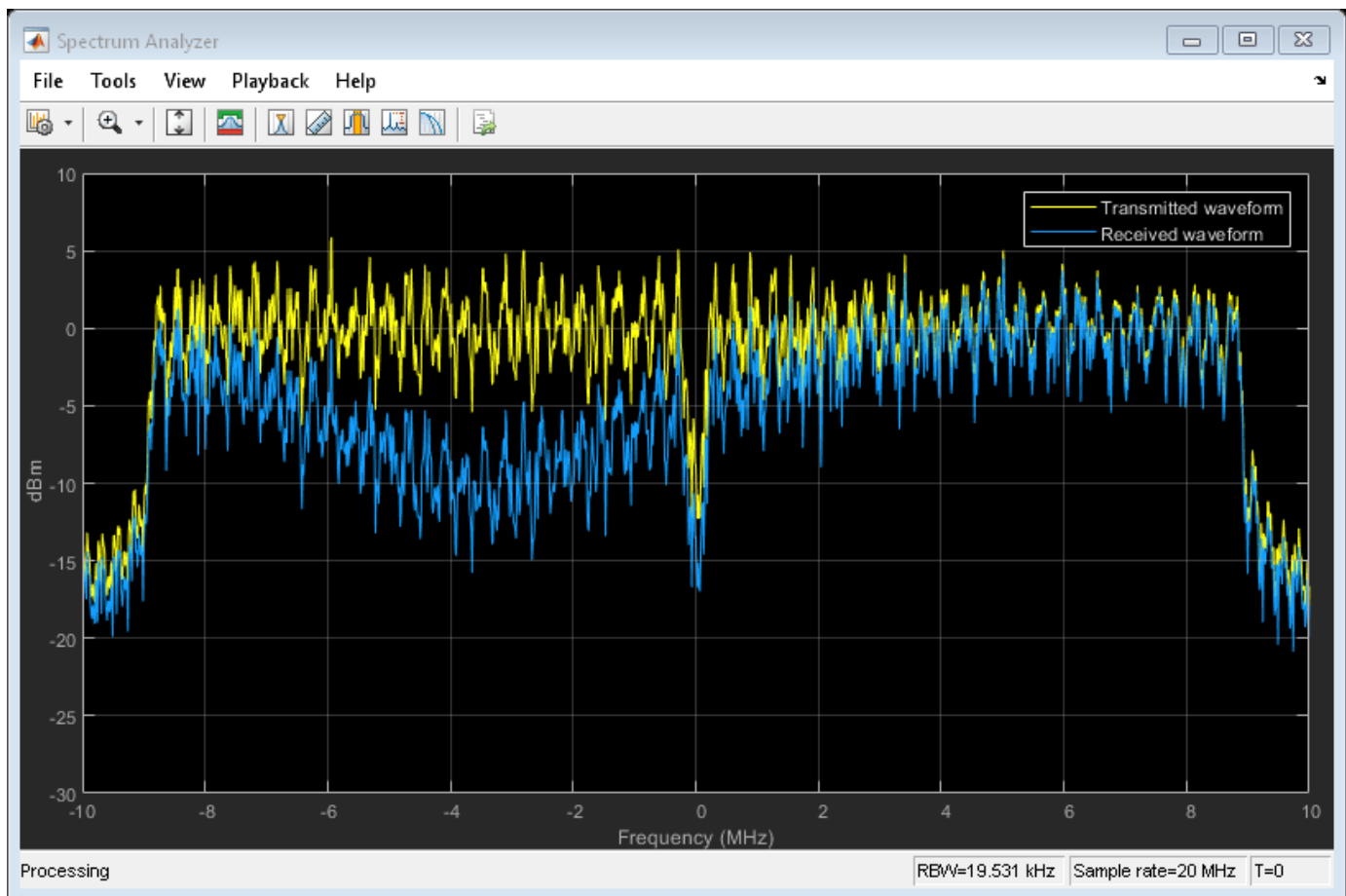

Channel Estimation and Equalization

In this section the time-domain VHT-LTF is extracted from the received waveform. The waveform is assumed to be synchronized to the start of the packet by taking the channel filter delay into account. The VHT-LTF is demodulated and is used to estimate the channel. The received signal is then equalized using the channel estimate obtained from the VHT-LTF.

In this example the received signal is synchronized to the start of the packet by compensating for a known channel filter delay. For more information on how to automatically detect and synchronize to the received signal see the following examples:

- "802.11n Packet Error Rate Simulation for 2x2 TGn Channel" on page 5-17
- "802.11ac Packet Error Rate Simulation for 8x8 TGac Channel" on page 5-11

```
chInfo = info(tgacChannel); % Get characteristic information
% Channel filter delay, measured in samples
chDelay  = chInfo.ChannelFilterDelay;
rxWaveform = rxWaveform(chDelay+1:end,:);
```

After synchronization the receiver has to extract the relevant fields from the received packet. The `wlanFieldIndices` function is used to return the start and end time-domain sample indices of all fields relative to the first sample in a packet. These indices are used to extract the required fields for further processing.

```
indField = wlanFieldIndices(cfgVHT);
```

An estimate of the noise power after OFDM demodulation is required to perform MMSE equalization on the received OFDM symbols. In this example the noise power in the VHT fields is estimated using the demodulated L-LTF symbols.The L-LTF is extracted from the received waveform and is demodulated using the `wlanLLTFDemodulate` function.

```
indLLTF = indField.LLTF(1):indField.LLTF(2);
demodLLTF = wlanLLTFDemodulate(rxWaveform(indLLTF),cfgVHT);
% Estimate noise power in VHT fields
nVar = helperNoiseEstimate(demodLLTF,cfgVHT.ChannelBandwidth,cfgVHT.NumSpaceTimeStreams);
```

To extract the VHT-LTF from the received signal the start and end indices are used to generate a vector of indices.

```
indVHTLTF = indField.VHTLTF(1):indField.VHTLTF(2);
```

The VHT-LTF is used to estimate the channel between all space-time streams and receive antennas. The VHT-LTF is extracted from the received waveform and is demodulated using the `wlanVHTLTFDemodulate` function.

```
demodVHTLTF = wlanVHTLTFDemodulate(rxWaveform(indVHTLTF,:),cfgVHT);
```

The channel estimate includes the effect of the applied spatial mapping and cyclic shifts at the transmitter for a multi antenna configuration. The `wlanVHTLTFChannelEstimate` function returns the estimated channel between all space-time streams and receive antennas.

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

The transmit signal encounters a deep fade as shown in the channel frequency response in the figure below. The effect of channel fades can also be seen in the spectrum plot shown previously.

```
figure
plot(20*log10(abs(chanEstVHTLTF)));
grid on;
title('Estimated Channel Response');
xlabel('Subcarrier index');
ylabel('Power (dB)');
```

Estimated Channel Response
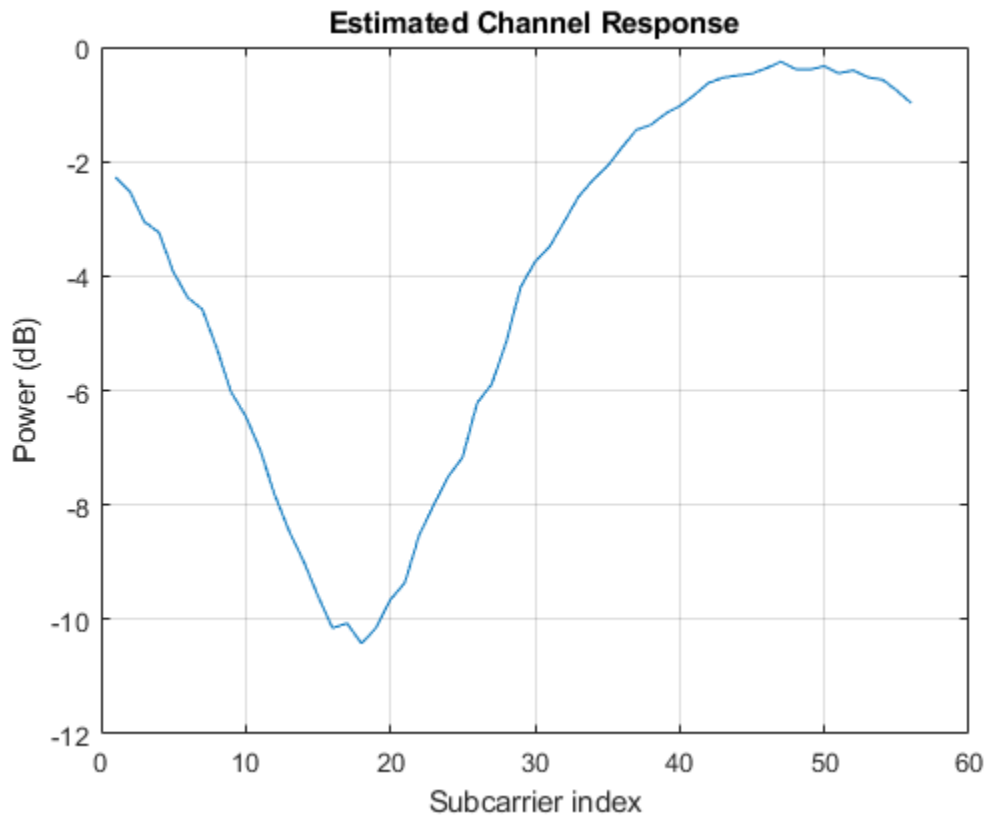
To extract the data field from the received signal the start and end indices for the data field are used to generate a vector of indices.

```
indData = indField.VHTData(1):indField.VHTData(2);

% Recover the bits and equalized symbols in the VHT Data field using the
% channel estimates from VHT-LTF
[rxPSDU,~,eqSym] = wlanVHTDataRecover(rxWaveform(indData,:),chanEstVHTLTF,nVar,cfgVHT);

% Compare transmit and receive PSDU bits
numErr = biterr(txPSDU,rxPSDU);
```

The following plot shows the constellation of the equalized symbols at the output of the `wlanVHTDataRecover` function compared against the reference constellation. Increasing the channel noise should begin to spread the distinct constellation points.

```
% Plot equalized symbols
constellationDiagram = comm.ConstellationDiagram;
constellationDiagram.ReferenceConstellation = wlanReferenceSymbols(cfgVHT);
% Compare received and reference constellation
constellationDiagram(reshape(eqSym,[],1));
constellationDiagram.Title = 'Equalized Data Symbols';
```
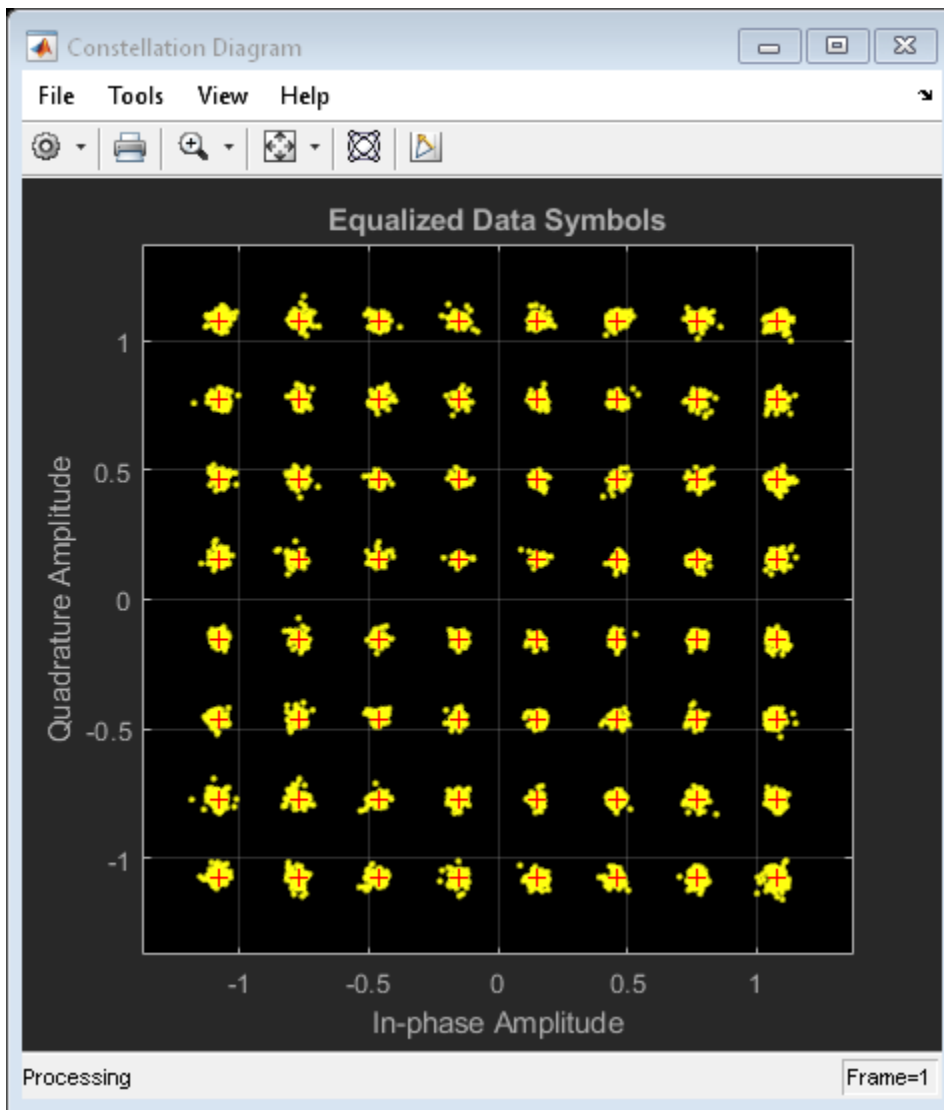
Appendix

This example uses this helper function.

- helperNoiseEstimate.m

Selected Bibliography

1 IEEE Std 802.11ac™-2013 IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz.

2 Breit, G., H. Sampath, S. Vermani, et al. TGac Channel Model Addendum. Version 12. IEEE 802.11-09/0308r12, March 2010.

802.11ac Multi-User MIMO Precoding

This example shows the transmit and receive processing for a 802.11ac™ multi-user downlink transmission over a fading channel. The example uses linear precoding techniques based on a singular-value-decomposition (SVD) of the channel.

Introduction

802.11ac supports downlink (access-point to station) multi-user transmissions for up to four users and up to eight transmit antennas to increase the aggregate throughput of the link [1]. Based on a scheduled transmission time for a user, the scheduler looks for other smaller packets ready for transmission to other users. If available, it schedules these users over the same interval, which reduces the overall time taken for multiple transmissions.

This simultaneous transmission comes at a higher complexity because successful reception of the individual user's payloads requires precoding, also known as transmit-end beamforming. Precoding assumes that channel state information (CSI) is known at the transmitter. A sounding packet, as described in the "802.11ac Transmit Beamforming" on page 3-21 example, is used to determine the CSI for each user in a multi-user transmission. Each of the users feed back their individual CSI to the beamformer. The beamformer uses the CSI from all users to set the precoding (spatial mapping) matrix for subsequent data transmission.

This example uses a channel inversion technique for a three-user transmission with a different number of spatial streams allocated per user and different rate parameters per user. The system can be characterized by the figure below.
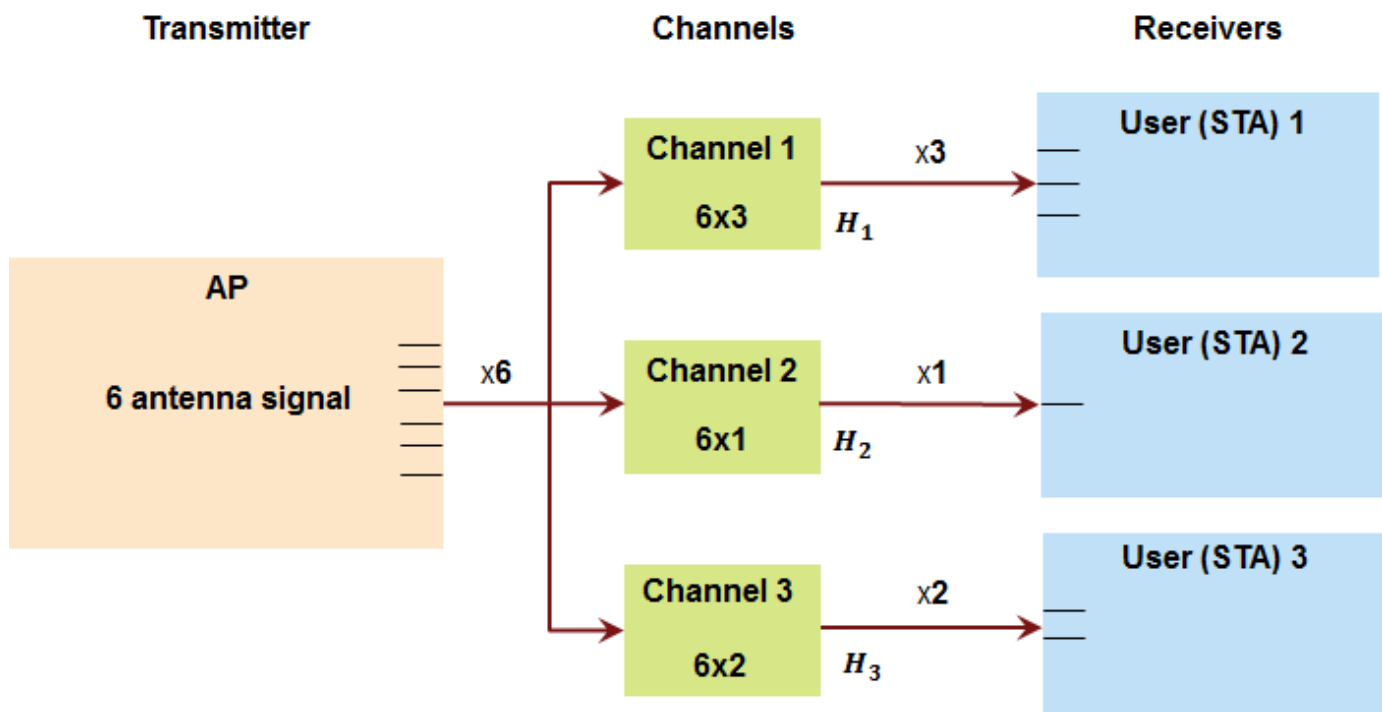
The example generates the multi-user transmit waveform, passes it through a channel per user and decodes the received signal for each user to calculate the bits in error. Prior to the data transmission,

the example uses a null-data packet (NDP) transmission to sound the different channels and determines the precoding matrix under the assumption of perfect feedback.

Simulation Parameters and Configuration

For 802.11ac, a maximum of eight spatial streams is allowed. A 6x6 MIMO configuration for three users is used in this example, where the first user has three streams, second has one, and the third has two streams allocated to it. Different rate parameters and payload sizes for up to four users are specified as vector parameters. These are indexed appropriately in the transmission configuration based on the number of active users.

```matlab
s = rng(21);                              % Set RNG seed for repeatability

% Transmission parameters
chanBW      = 'CBW80';                    % Channel bandwidth
numUsers    = 3;                          % Number of active users
numSTSAll   = [3 1 2 2];                  % Number of streams for 4 users
userPos     = [0 1 2 3];                  % User positions for maximum 4 users
mcsVec      = [4 6 2 2];                  % MCS for maximum 4 users
apepVec     = [15120 8192 5400 6000];     % Payload, in bytes, for 4 users
chCodingVec = {'BCC', 'LDPC', 'LDPC', 'BCC'};  % Channel coding for 4 users

% Channel and receiver parameters
chanMdl      = 'Model-A';                 % TGac fading channel model
precodingType = 'ZF';                     % Precoding type; ZF or MMSE
snr          = 38;                        % SNR in dB
eqMethod     = 'ZF';                      % Equalization method

% Create the multi-user VHT format configuration object, appropriately
% indexing into the vector values for the active users
if (numUsers==1)
    groupID = 0;
else
    groupID = 2;
end
numSTSVec = numSTSAll(1:numUsers);
numTx = sum(numSTSVec);
cfgVHTMU = wlanVHTConfig('ChannelBandwidth', chanBW,...
    'NumUsers', numUsers, ...
    'NumTransmitAntennas', numTx, ...
    'GroupID', groupID, ...
    'NumSpaceTimeStreams', numSTSVec,...
    'UserPositions', userPos(1:numUsers), ...
    'MCS', mcsVec(1:numUsers), ...
    'APEPLength', apepVec(1:numUsers), ...
    'ChannelCoding', chCodingVec(1:numUsers));
```

The number of transmit antennas is set to be the sum total of all the used space-time streams. This implies no space-time block coding (STBC) or spatial expansion is employed for the transmission.

Sounding (NDP) Configuration

For precoding, channel sounding is first used to determine the channel experienced by the users (receivers). This channel state information is sent back to the transmitter, for it to be used for subsequent data transmission. It is assumed that the channel varies slowly over the two transmissions. For multi-user transmissions, the same NDP (Null Data Packet) is transmitted to each of the scheduled users [2].

```
% VHT sounding (NDP) configuration, for same number of streams
cfgVHTNDP = wlanVHTConfig('ChannelBandwidth', chanBW,...
    'NumUsers', 1, ...
    'NumTransmitAntennas', numTx, ...
    'GroupID', 0, ...
    'NumSpaceTimeStreams', sum(numSTSVec),...
    'MCS', 0, ...
    'APEPLength', 0);
```

The number of streams specified is the sum total of all space-time streams used. This allows the complete channel to be sounded.

```
% Generate the null data packet, with no data
txNDPSig = wlanWaveformGenerator([], cfgVHTNDP);
```

Transmission Channel

The TGac multi-user channel consists of independent single-user MIMO channels between the access point and spatially separated stations [4]. In this example, the same delay profile Model-A channel is applied for each of the users, even though individual users can experience different conditions. The flat-fading channel allows a simpler receiver without front-end synchronization. It is also assumed that each user's number of receive antennas are equal to the number of space-time streams allocated to them.

Cell arrays are used in the example to store per-user elements which allow for a flexible number of users. Here, as an example, each instance of the TGac channel per user is stored as an element of a cell array.

```
% Create three independent channels
TGAC      = cell(numUsers, 1);
chanSeeds = [1111 2222 3333 4444];  % chosen for a maximum of 4 users
uIndex    = [10 5 2 1];             % chosen for a maximum of 4 users
chanDelay = zeros(numUsers, 1);
for uIdx = 1:numUsers
    TGAC{uIdx} = wlanTGacChannel(...
        'ChannelBandwidth', cfgVHTMU.ChannelBandwidth,...
        'DelayProfile', chanMdl, ...
        'UserIndex', uIndex(uIdx), ...
        'NumTransmitAntennas', numTx, ...
        'NumReceiveAntennas', numSTSVec(uIdx), ...
        'RandomStream', 'mt19937ar with seed', ...
        'Seed', chanSeeds(uIdx),...
        'SampleRate', wlanSampleRate(cfgVHTMU), ...
        'TransmitReceiveDistance',5);
    chanInfo = info(TGAC{uIdx});
    chanDelay(uIdx) = chanInfo.ChannelFilterDelay;
end
```

The channels for each individual user use different seeds for random number generation. A different user index is specified to allow for random angle offsets to be applied to the arrival (AoA) and departure (AoD) angles for the clusters. The channel filtering delay is stored to allow for its compensation at the receiver. In practice, symbol timing estimation would be used.

```
% Append zeroes to allow for channel filter delay
txNDPSig = [txNDPSig; zeros(10, numTx)];

% Sound the independent channels per user for all transmit streams
```

```
rxNDPSig = cell(numUsers, 1);
for uIdx = 1:numUsers
    rxNDPChan = TGAC{uIdx}(txNDPSig);

    % Add WGN per receiver
    rxNDPSig{uIdx} = awgn(rxNDPChan, snr);
end
```

Channel State Information Feedback

Each user estimates its own channel using the received NDP signal and computes the channel state information that it can send back to the transmitter. This example uses the singular value decomposition of the channel seen by each user to compute the CSI feedback.

```
mat = cell(numUsers,1);
for uIdx = 1:numUsers
    % Compute the feedback matrix based on received signal per user
    mat{uIdx} = vhtCSIFeedback(rxNDPSig{uIdx}(chanDelay(uIdx)+1:end,:), ...
        cfgVHTNDP, uIdx, numSTSVec);
end
```

Assuming perfect feedback, with no compression or quantization loss of the CSI, the transmitter computes the steering matrix for the data transmission using either Zero-Forcing or Minimum-Mean-Square-Error (MMSE) based precoding techniques. Both methods attempt to cancel out the intra-stream interference for the user of interest and interference due to other users. The MMSE-based approach avoids the noise enhancement inherent in the zero-forcing technique. As a result, it performs better at low SNRs.

```
% Pack the per user CSI into a matrix
numST = length(mat{1});          % Number of subcarriers
steeringMatrix = zeros(numST, sum(numSTSVec), sum(numSTSVec));
%   Nst-by-Nt-by-Nsts
for uIdx = 1:numUsers
    stsIdx = sum(numSTSVec(1:uIdx-1))+(1:numSTSVec(uIdx));
    steeringMatrix(:,:,stsIdx) = mat{uIdx};      % Nst-by-Nt-by-Nsts
end

% Zero-forcing or MMSE precoding solution
if strcmp(precodingType, 'ZF')
    delta = 0; % Zero-forcing
else
    delta = (numTx/(10^(snr/10))) * eye(numTx); % MMSE
end
for i = 1:numST
    % Channel inversion precoding
    h = squeeze(steeringMatrix(i,:,:));
    steeringMatrix(i,:,:) = h/(h'*h + delta);
end

% Set the spatial mapping based on the steering matrix
cfgVHTMU.SpatialMapping = 'Custom';
cfgVHTMU.SpatialMappingMatrix = permute(steeringMatrix,[1 3 2]);
```

Data Transmission

Random bits are used as the payload for the individual users. A cell array is used to hold the data bits for each user, txDataBits. For a multi-user transmission the individual user payloads are padded

such that the transmission duration is the same for all users. This padding process is described in Section 9.12.6 of [1]. In this example for simplicity the payload is padded with zeros to create a PSDU for each user.

```
% Create data sequences, one for each user
txDataBits = cell(numUsers, 1);
psduDataBits = cell(numUsers, 1);
for uIdx = 1:numUsers
    % Generate payload for each user
    txDataBits{uIdx} = randi([0 1], cfgVHTMU.APEPLength(uIdx)*8, 1, 'int8');

    % Pad payload with zeros to form a PSDU
    psduDataBits{uIdx} = [txDataBits{uIdx}; ...
        zeros((cfgVHTMU.PSDULength(uIdx)-cfgVHTMU.APEPLength(uIdx))*8, 1, 'int8')];
end
```

Using the format configuration, cfgVHTMU, with the steering matrix, the data is transmitted over the fading channel.

```
% Generate the multi-user VHT waveform
txSig = wlanWaveformGenerator(psduDataBits, cfgVHTMU);

% Transmit through per-user fading channel
rxSig = cell(numUsers, 1);
for uIdx = 1:numUsers
    % Append zeroes to allow for channel filter delay
    rxSig{uIdx} = TGAC{uIdx}([txSig; zeros(10, numTx)]);
end
```

Data Recovery Per User

The receive signals for each user are processed individually. The example assumes that there are no front-end impairments and that the transmit configuration is known by the receiver for simplicity.

A user number specifies the user of interest being decoded for the transmission. This is also used to index into the vector properties of the configuration object that are user-specific.

```
% Get field indices from configuration, assumed known at receiver
ind = wlanFieldIndices(cfgVHTMU);

% Single-user receivers recover payload bits
rxDataBits = cell(numUsers, 1);
scaler = zeros(numUsers, 1);
spAxes = gobjects(sum(numSTSVec), 1);
hfig = figure('Name','Per-stream equalized symbol constellation');
for uIdx = 1:numUsers
    % Add WGN per receiver
    rxNSig = awgn(rxSig{uIdx}, snr);
    rxNSig = rxNSig(chanDelay(uIdx)+1:end, :);

    % User space-time streams
    stsU = numSTSVec(uIdx);

    % Perform channel estimation based on VHT-LTF
    rxVHTLTF   = rxNSig(ind.VHTLTF(1):ind.VHTLTF(2),:);
    demodVHTLTF = wlanVHTLTFDemodulate(rxVHTLTF, chanBW, numSTSVec);
    chanEst = wlanVHTLTFChannelEstimate(demodVHTLTF, chanBW, numSTSVec);
```

```matlab
        % Get single stream channel estimate
        chanEstSSPilots = vhtSingleStreamChannelEstimate(demodVHTLTF,cfgVHTMU);

        % Extract VHT Data samples from the waveform
        rxVHTData = rxNSig(ind.VHTData(1):ind.VHTData(2),:);

        % Estimate the noise power in VHT data field
        nVar = vhtNoiseEstimate(rxVHTData,chanEstSSPilots,cfgVHTMU);

        % Recover information bits in VHT Data field
        [rxDataBits{uIdx}, ~, eqsym] = wlanVHTDataRecover(rxVHTData, ...
            chanEst, nVar, cfgVHTMU, uIdx, 'EqualizationMethod', eqMethod, ...
            'PilotPhaseTracking', 'None', 'LDPCDecodingMethod', 'layered-bp');

        % Plot equalized symbols for all streams per user
        scaler(uIdx) = ceil(max(abs([real(eqsym(:)); imag(eqsym(:))])));
        for i = 1:stsU
            subplot(numUsers, max(numSTSVec), (uIdx-1)*max(numSTSVec)+i);
            plot(reshape(eqsym(:,:,i), [], 1), '.');
            axis square
            spAxes(sum([0 numSTSVec(1:(uIdx-1))])+i) = gca; % Store axes handle
            title(['User ' num2str(uIdx) ', Stream ' num2str(i)]);
            grid on;
        end
    end

    % Scale axes for all subplots and scale figure
    for i = 1:numel(spAxes)
        xlim(spAxes(i),[-max(scaler) max(scaler)]);
        ylim(spAxes(i),[-max(scaler) max(scaler)]);
    end
    pos = get(hfig, 'Position');
    set(hfig, 'Position', [pos(1)*0.7 pos(2)*0.7 1.3*pos(3) 1.3*pos(4)]);
```
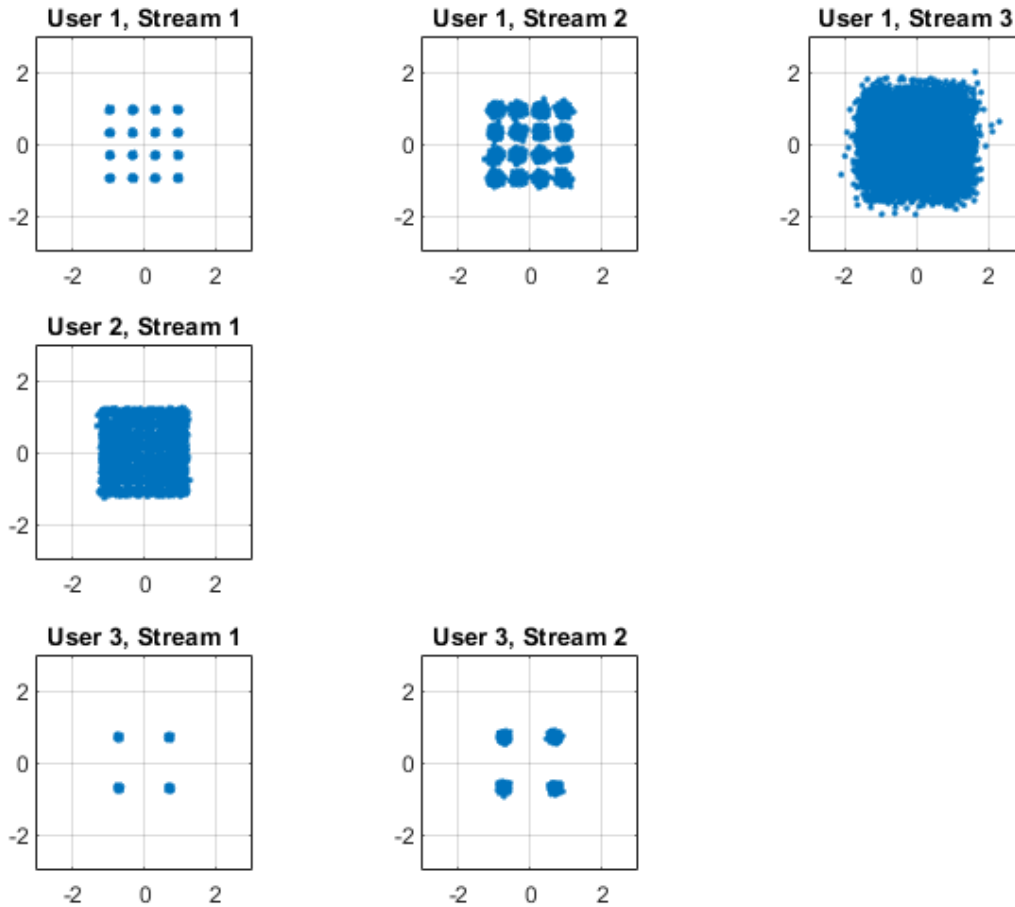
Per-stream equalized symbol constellation plots validate the simulation parameters and convey the effectiveness of the technique. Note the discernible 16QAM, 64QAM and QPSK constellations per user as specified on the transmit end. Also observe the EVM degradation over the different streams for an individual user. This is a representative characteristic of the channel inversion technique.

The recovered data bits are compared with the transmitted payload bits to determine the bit error rate.

```
% Compare recovered bits against per-user APEPLength information bits
ber = inf(1, numUsers);
for uIdx = 1:numUsers
    idx = (1:cfgVHTMU.APEPLength(uIdx)*8).';
    [~, ber(uIdx)] = biterr(txDataBits{uIdx}(idx), rxDataBits{uIdx}(idx));
    disp(['Bit Error Rate for User ' num2str(uIdx) ': ' num2str(ber(uIdx))]);
end

rng(s); % Restore RNG state
```

```
Bit Error Rate for User 1: 0.00013228
Bit Error Rate for User 2: 0
Bit Error Rate for User 3: 0
```

The small number of bit errors, within noise variance, indicate successful data decoding for all streams for each user, despite the variation in EVMs seen in individual streams.

Conclusion and Further Exploration

The example shows multi-user transmit configuration, independent per-user channel modeling, and the individual receive processing using the channel inversion precoding techniques.

Further exploration includes modifications to the transmission and channel parameters, alternate precoding techniques, more realistic receivers and feedback mechanism incorporating delays and quantization.

Appendix

This example uses the following helper functions:

- vhtBeamformingRemoveCSD.m
- vhtCSIFeedback.m
- vhtNoiseEstimate.m
- vhtSingleStreamChannelEstimate.m

Selected Bibliography

1 IEEE Std 802.11ac™-2013 IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz.

2 Perahia, E., R. Stacey, "Next Generation Wireless LANS: 802.11n and 802.11ac", Cambridge University Press, 2013.

3 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.

4 Breit, G., H. Sampath, S. Vermani, et al., "TGac Channel Model Addendum", Version 12. IEEE 802.11-09/0308r12, March 2010.

MAC Modeling

802.11 MAC Frame Generation

This example shows how to generate IEEE® 802.11™ MAC frames.

Introduction

This example shows how WLAN MAC frames specified in Section 9 of [1] and [2] can be generated and exported to a packet capture (PCAP) file for analysis with third-party packet analysis tools. In this example Wireshark [3] is used to verify the content of MAC frames is as expected.

The general MAC frame format consists of a header, frame-body, and frame check sequence (FCS). The header holds information about the frame. The frame-body carries data that needs to be transmitted. The transmitter calculates the FCS over the header and frame-body. The receiver uses the FCS to confirm that the header and frame-body are properly received. The following diagram shows the structure of a general MAC frame.

MAC Frame Format

MAC Header	Frame Body	FCS
Variable	Variable	4 octets

Frame Control	Duration/ ID	Address 1	Address 2	Address 3	Sequence Control	Address 4	QoS Control	HT Control
2 octets	2 octets	6 octets	6 octets	6 octets	2 octets	6 octets	2 octets	4 octets

Protocol Version	Type	Subtype	To DS	From DS	More Frag	Retry	Power Mgmt.	More Data	Protected Frame	+HTC/ Order
2 bits	2 bits	4 bits	1 bit	1 bit	1 bit	1 bit	1 bit	1 bit	1 bit	1 bit

- *Mandatory fields for all frame types*
- *Fields that are mandatory based on Type and Subtype of the frame*
- *Fields that are optionally present based on flags in the frame control field*

The standard specifies four types of frames: Management, Data, Control, and Extension. There are multiple subtypes of each frame type. These are identified by the *Type* and *Subtype* fields of the frame control field in the MAC header.

Management Frames:

- Frames that are used for connection establishment and maintenance.
- These frames carry the information fields and elements that indicate the capabilities and configuration of the device operating in the 802.11 network. While establishing the connection, these information fields and elements are communicated between the devices to match capabilities of both devices.
- The MAC layer adds header and FCS to the frame-body carrying the information to form a MAC protocol data unit (MPDU).

Data Frames:

- Frames that are used to transmit the higher-layer data.
- The payload given to the MAC layer is termed MAC service data unit (MSDU). The MAC layer adds header and FCS to the MSDU to form a MAC protocol data unit (MPDU).
- To improve the throughput, WLAN supports aggregated MSDUs (A-MSDUs) and aggregated MPDUs (A-MPDUs) as specified in Sections 9.3.2.2, 9.7 of [1] and [2].

- If MSDU aggregation is enabled, multiple MSDUs are aggregated to form an A-MSDU and then the MAC header and FCS are added to an A-MSDU to form an MPDU.
- If MPDU aggregation is enabled, multiple MPDUs are aggregated to form an A-MPDU.

Control Frames:

- Frames that are used to support the delivery of data, management, and extension frames.
- Each control frame has a specific functionality. For example, control frames like request-to-send (RTS) and clear-to-send (CTS) help in reserving the channel to avoid collisions, while Ack frames help in recognizing successful transmission.

Extension Frames:

- This frame type is an extension to the three frame types defined above.
- DMG Beacon is the only frame currently specified under this frame type in [1].

You can use the `wlanMACFrame` function to generate MAC frames. This function accepts a MAC frame configuration object `wlanMACFrameConfig` as an input. This object configures the fields in the MAC header. Set the `FrameType` property to the desired *Subtype description* in Table 9-1 of [1] to set the appropriate *Type* and *Subtype* fields in the MAC header. The `wlanMACFrame` function supports the generation of following MPDUs.

- **Management Frames:** Beacon
- **Data Frames:** Data, Null, QoS Data, QoS Null
- **Control Frames:** RTS, CTS, Ack, Block Ack

In addition to the above MPDUs, `wlanMACFrame` also supports generation of A-MPDUs containing MPDUs of type `QoS Data`.

Control Frame Generation

To generate an RTS frame, create a MAC frame configuration object with the `FrameType` set to `'RTS'`.

```
rtsCfg = wlanMACFrameConfig('FrameType', 'RTS');
disp(rtsCfg);

  wlanMACFrameConfig with properties:

          FrameType: 'RTS'
    PowerManagement: 0
           MoreData: 0
           Duration: 0
           Address1: 'FFFFFFFFFFFF'
           Address2: '00123456789B'

  Read-only properties:
            Decoded: 0
```

Configure the frame header fields.

```
% Duration
rtsCfg.Duration = 500;
% Receiver address
```

```
rtsCfg.Address1 = 'FCF8B0102001';
% Transmitter address
rtsCfg.Address2 = 'FCF8B0102002';
```

Generate an RTS frame using the configuration.

```
% Generate octets for an RTS frame
rtsFrame = wlanMACFrame(rtsCfg);
```

By default, the output of `wlanMACFrame` is a sequence of hexadecimal octets. If you want to generate the MAC frame as a sequence of bits, set the `OutputFormat` parameter to `bits`.

```
% Generate bits for an RTS frame
rtsFrameBits = wlanMACFrame(rtsCfg, 'OutputFormat', 'bits');
```

Data Frame Generation

To generate a QoS Data frame, create a MAC frame configuration object with the `FrameType` set to `'QoS Data'`.

```
qosDataCfg = wlanMACFrameConfig('FrameType', 'QoS Data');
disp(qosDataCfg);

  wlanMACFrameConfig with properties:

          FrameType: 'QoS Data'
        FrameFormat: 'Non-HT'
               ToDS: 0
             FromDS: 1
      Retransmission: 0
    PowerManagement: 0
           MoreData: 0
           Duration: 0
           Address1: 'FFFFFFFFFFFF'
           Address2: '00123456789B'
           Address3: '00123456789B'
     SequenceNumber: 0
                TID: 0
          AckPolicy: 'No Ack'
    MSDUAggregation: 0

  Read-only properties:
            Decoded: 0
```

Configure the frame header fields.

```
% From DS flag
qosDataCfg.FromDS = 1;
% To DS flag
qosDataCfg.ToDS = 0;
% Acknowledgment Policy
qosDataCfg.AckPolicy = 'Normal Ack';
% Receiver address
qosDataCfg.Address1 = 'FCF8B0102001';
% Transmitter address
qosDataCfg.Address2 = 'FCF8B0102002';
```

The QoS Data frame is used to transmit a payload from higher-layer. A 20-byte payload containing a repeating sequence of hexadecimal value '11' is used in this example.

```
payload = repmat('11', 1, 20);
```

Generate a QoS Data frame using payload and configuration.

```
% Generate octets for a QoS Data frame
qosDataFrame = wlanMACFrame(payload, qosDataCfg);
```

By default, the output of `wlanMACFrame` is a sequence of hexadecimal octets. If you want to generate the MAC frame as a sequence of bits, set the `OutputFormat` parameter to `bits`.

```
% Generate bits for a QoS Data frame
qosDataFrameBits = wlanMACFrame(payload, qosDataCfg, 'OutputFormat', 'bits');
```

The output MAC frame is an MPDU with a single MSDU. Refer to the example "802.11ac Waveform Generation with MAC Frames" on page 2-18 for A-MSDU and A-MPDU generation.

Management Frame Generation

To generate a Beacon frame, create a MAC frame configuration object with the `FrameType` set to `'Beacon'`.

```
beaconCfg = wlanMACFrameConfig('FrameType', 'Beacon');
disp(beaconCfg);

  wlanMACFrameConfig with properties:

           FrameType: 'Beacon'
                ToDS: 0
              FromDS: 1
      Retransmission: 0
     PowerManagement: 0
            MoreData: 0
            Duration: 0
            Address1: 'FFFFFFFFFFFF'
            Address2: '00123456789B'
            Address3: '00123456789B'
      SequenceNumber: 0
    ManagementConfig: [1x1 wlanMACManagementConfig]

   Read-only properties:
             Decoded: 0
```

Beacon frame-body consists of information fields and information elements as explained in Section 9.3.3.3 of [1]. You can configure these information fields and elements using `wlanMACManagementConfig`.

```
% Create a management frame-body configuration object
frameBodyCfg = wlanMACManagementConfig;
disp(frameBodyCfg);

  wlanMACManagementConfig with properties:

                   FrameType: 'Beacon'
                   Timestamp: 0
```

```
           BeaconInterval: 100
            ESSCapability: 1
           IBSSCapability: 0
                  Privacy: 0
            ShortPreamble: 0
       SpectrumManagement: 0
               QoSSupport: 1
          ShortSlotTimeUsed: 0
             APSDSupport: 0
         RadioMeasurement: 0
    DelayedBlockAckSupport: 0
  ImmediateBlockAckSupport: 0
                     SSID: 'default SSID'
               BasicRates: {'6 Mbps'  '12 Mbps'  '24 Mbps'}
          AdditionalRates: {}

  Read-only properties:
       InformationElements: {511x2 cell}
```

Configure the information fields and elements in the frame-body configuration. You can add information elements using `addIE(elementID, information)` method as shown below. Refer Section 9.4 in [1] for the list of information fields and information elements.

```
% Beacon Interval
frameBodyCfg.BeaconInterval = 100;
% Timestamp
frameBodyCfg.Timestamp = 123456;
% SSID
frameBodyCfg.SSID = 'TEST_BEACON';
% Add DS Parameter IE (element ID - 3) with channel number 11 (0x0b)
frameBodyCfg = frameBodyCfg.addIE(3, '0b');
```

Assign the updated frame-body configuration object to the `ManagementConfig` property in the MAC frame configuration.

```
% Update management frame-body configuration
beaconCfg.ManagementConfig = frameBodyCfg;
```

Generate the Beacon frame with the updated frame configuration.

```
% Generate octets for a Beacon frame
beaconFrame = wlanMACFrame(beaconCfg);
```

By default, the output of `wlanMACFrame` is a sequence of hexadecimal octets. If you want to generate the MAC frame as a sequence of bits, set the `OutputFormat` parameter to `bits`.

```
% Generate bits for a Beacon frame
beaconFrameBits = wlanMACFrame(beaconCfg, 'OutputFormat', 'bits');
```

Exporting to a PCAP File

This example uses `pcapWriter` object to export the generated MAC frames to a file with .pcap extension. To analyze and visualize this file, use a third-party packet analyzer such as Wireshark. To export the generated MAC frames to a file with .pcapng extension, use `pcapngWriter` object.

Create an object of type `pcapWriter` and specify the packet capture file name. The constants `wlanLinkType` and `timestamp` specifies the link layer header type [4] and the capture time of a

WLAN frame, respectively. In this example, the capture time is same for all the frames. Before writing packets to the file with .pcap or .pcapng extension, use `writeGlobalHeader` function to write a global header to the file.

```matlab
timestamp = 124800; % Timestamp (in microseconds)
wlanLinkType = 105; % Link-layer header type
fileName = 'macFrames';
% Delete if a file with the same name already exists in the current
% directory
if isfile([fileName, '.pcap'])
    delete([fileName, '.pcap']);
end
pcap = pcapWriter('FileName', 'macFrames');
writeGlobalHeader(pcap, wlanLinkType); % Global header in pcap file
```

Use the `write` function to write all the MAC frames to a PCAP file

```matlab
% MAC frames to be exported to a PCAP file
frames = {rtsFrame, qosDataFrame, beaconFrame};

% Write all the packets to the PCAP file
for idx = 1:numel(frames)
    write(pcap, frames{idx}, timestamp);
end

% Clear the object
clear pcap;
```

Visualization of the Generated MAC Frames

You can open the PCAP files containing the generated MAC frames in a packet analyzer. The frames decoded by the Wireshark match the standard compliant MAC frames generated using the WLAN Toolbox. This figure shows the analysis of the captured MAC frames in Wireshark.

- **RTS frame**

```
⌄ IEEE 802.11 Request-to-send, Flags: ........C
      Type/Subtype: Request-to-send (0x001b)
  ⌄ Frame Control Field: 0xb400
        .... ..00 = Version: 0
        .... 01.. = Type: Control frame (1)
        1011 .... = Subtype: 11
    ⌄ Flags: 0x00
          .... ..00 = DS status: Not leaving DS or network is operating in AD-HOC mode (To DS: 0 From DS: 0) (0x0)
          .... .0.. = More Fragments: This is the last fragment
          .... 0... = Retry: Frame is not being retransmitted
          ...0 .... = PWR MGT: STA will stay up
          ..0. .... = More Data: No data buffered
          .0.. .... = Protected flag: Data is not protected
          0... .... = Order flag: Not strictly ordered
      .000 0001 1111 0100 = Duration: 500 microseconds
      Receiver address: fc:f8:b0:10:20:01 (fc:f8:b0:10:20:01)
      Transmitter address: fc:f8:b0:10:20:02 (fc:f8:b0:10:20:02)
      Frame check sequence: 0x57334a70 [correct]
      [FCS Status: Good]
```

- **QoS Data frame**

```
∨ IEEE 802.11 QoS Data, Flags: ......F.C
      Type/Subtype: QoS Data (0x0028)
    ∨ Frame Control Field: 0x8802
         .... ..00 = Version: 0
         .... 10.. = Type: Data frame (2)
         1000 .... = Subtype: 8
    ∨ Flags: 0x02
         .... ..10 = DS status: Frame from DS to a STA via AP(To DS: 0 From DS: 1) (0x2)
         .... .0.. = More Fragments: This is the last fragment
         .... 0... = Retry: Frame is not being retransmitted
         ...0 .... = PWR MGT: STA will stay up
         ..0. .... = More Data: No data buffered
         .0.. .... = Protected flag: Data is not protected
         0... .... = Order flag: Not strictly ordered
      .000 0000 0000 0000 = Duration: 0 microseconds
      Receiver address: fc:f8:b0:10:20:01 (fc:f8:b0:10:20:01)
      Destination address: fc:f8:b0:10:20:01 (fc:f8:b0:10:20:01)
      Transmitter address: fc:f8:b0:10:20:02 (fc:f8:b0:10:20:02)
      Source address: CamilleB_56:78:9b (00:12:34:56:78:9b)
      BSS Id: fc:f8:b0:10:20:02 (fc:f8:b0:10:20:02)
      STA address: fc:f8:b0:10:20:01 (fc:f8:b0:10:20:01)
      .... .... .... 0000 = Fragment number: 0
      0000 0000 0000 .... = Sequence number: 0
      Frame check sequence: 0xc85aa49b [correct]
      [FCS Status: Good]
    ∨ Qos Control: 0x0000
         .... .... .... 0000 = TID: 0
         [.... .... .... .000 = Priority: Best Effort (Best Effort) (0)]
         .... .... ...0 .... = EOSP: Service period
         .... .... .00. .... = Ack Policy: Normal Ack (0x0)
         .... .... 0... .... = Payload Type: MSDU
       > 0000 0000 .... .... = QAP PS Buffer State: 0x00
> Logical-Link Control
∨ Data (16 bytes)
      Data: 11111111111111111111111111111111
      [Length: 16]
```

- **Beacon frame**

```
∨ IEEE 802.11 Beacon frame, Flags: ......F.C
      Type/Subtype: Beacon frame (0x0008)
   ∨ Frame Control Field: 0x8002
         .... ..00 = Version: 0
         .... 00.. = Type: Management frame (0)
         1000 .... = Subtype: 8
      > Flags: 0x02
      .000 0000 0000 0000 = Duration: 0 microseconds
      Receiver address: Broadcast (ff:ff:ff:ff:ff:ff)
      Destination address: Broadcast (ff:ff:ff:ff:ff:ff)
      Transmitter address: CamilleB_56:78:9b (00:12:34:56:78:9b)
      Source address: CamilleB_56:78:9b (00:12:34:56:78:9b)
      BSS Id: CamilleB_56:78:9b (00:12:34:56:78:9b)
      STA address: Broadcast (ff:ff:ff:ff:ff:ff)
      .... .... .... 0000 = Fragment number: 0
      0000 0000 0000 .... = Sequence number: 0
      Frame check sequence: 0x5e56cf45 [correct]
      [FCS Status: Good]
∨ IEEE 802.11 wireless LAN
   ∨ Fixed parameters (12 bytes)
      Timestamp: 0x000000000001e240
      Beacon Interval: 0.102400 [Seconds]
   ∨ Capabilities Information: 0x0201
         .... .... .... ...1 = ESS capabilities: Transmitter is an AP
         .... .... .... ..0. = IBSS status: Transmitter belongs to a BSS
         .... ..1. .... 00.. = CFP participation capabilities: QAP (HC) does not use CFP for delivery of unicast data type frames (0x80)
         .... .... ...0 .... = Privacy: AP/STA cannot support WEP
         .... .... ..0. .... = Short Preamble: Not Allowed
         .... .... .0.. .... = PBCC: Not Allowed
         .... .... 0... .... = Channel Agility: Not in use
         .... ...0 .... .... = Spectrum Management: Not Implemented
         .... .0.. .... .... = Short Slot Time: Not in use
         .... 0... .... .... = Automatic Power Save Delivery: Not Implemented
         ...0 .... .... .... = Radio Measurement: Not Implemented
         ..0. .... .... .... = DSSS-OFDM: Not Allowed
         .0.. .... .... .... = Delayed Block Ack: Not Implemented
         0... .... .... .... = Immediate Block Ack: Not Implemented
   ∨ Tagged parameters (21 bytes)
      ∨ Tag: SSID parameter set: TEST_BEACON
         Tag Number: SSID parameter set (0)
         Tag length: 11
         SSID: TEST_BEACON
      ∨ Tag: Supported Rates 6(B), 12(B), 24(B), [Mbit/sec]
         Tag Number: Supported Rates (1)
         Tag length: 3
         Supported Rates: 6(B) (0x8c)
         Supported Rates: 12(B) (0x98)
         Supported Rates: 24(B) (0xb0)
      ∨ Tag: DS Parameter set: Current Channel: 11
         Tag Number: DS Parameter set (3)
         Tag length: 1
         Current Channel: 11
```

Conclusion and Further Exploration

This example demonstrated generation of MAC frames for the IEEE 802.11 standard. You can use a packet analyzer to view the generated MAC frames. To transmit the generated MAC frames over air, refer "802.11 OFDM Beacon Frame Generation" on page 2-24 and "802.11ac Waveform Generation with MAC Frames" on page 2-18 examples.

Selected Bibliography

1 IEEE Std 802.11™-2016 IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications

2 IEEE P802.11ax™/D4.1 Draft 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 6: Enhancements for High Efficiency WLAN

3   Wireshark · Go Deep. https://www.wireshark.org/. Accessed 30 June 2020

4   Group, The Tcpdump. Tcpdump/Libpcap Public Repository. https://www.tcpdump.org. Accessed 30 June 2020

802.11 MAC Frame Decoding

This example shows how to decode IEEE® 802.11™ MAC frames.

Background

The general MAC frame format consists of a header, frame-body, and frame check sequence (FCS). The header holds information about the frame. The frame-body carries data that needs to be transmitted. The transmitter calculates the FCS over the header and frame-body. The receiver uses the FCS to confirm that the header and frame-body are properly received. The following diagram shows the structure of a general MAC frame.

MAC Frame Format

The standard specifies four types of frames: Management, Data, Control, and Extension. There are multiple subtypes of each frame type. These are identified by the *Type* and *Subtype* fields of the frame control field in the MAC header.

Management Frames:

- Frames that are used for connection establishment and maintenance.
- These frames carry the information fields and elements that indicate the capabilities and configuration of the device operating in the 802.11 network. While establishing the connection,

these information fields and elements are communicated between the devices to match capabilities of both devices.

- The MAC layer adds header and FCS to the frame-body carrying the information to form a MAC protocol data unit (MPDU).

Data Frames:

- Frames that are used to transmit the higher-layer data.
- The payload given to the MAC layer is termed MAC service data unit (MSDU). The MAC layer adds header and FCS to the MSDU to form an MPDU.
- To improve the throughput, WLAN supports aggregated MSDUs (A-MSDUs) and aggregated MPDUs (A-MPDUs) as specified in Sections 9.3.2.2, 9.7 of [1] and [2].
- If MSDU aggregation is enabled, multiple MSDUs are aggregated to form an A-MSDU and then the MAC header and FCS are added to an A-MSDU to form an MPDU.
- If MPDU aggregation is enabled, multiple MPDUs are aggregated to form an A-MPDU.

Control Frames:

- Frames that are used to support the delivery of data, management and extension frames.
- Each control frame has a specific functionality. For instance, control frames like request-to-send (RTS) and clear-to-send (CTS) help in reserving the channel to avoid collisions, while Ack frames help in recognizing successful transmission.

Extension Frames:

- This frame type is an extension to the three frame types defined above.
- DMG Beacon is the only frame currently specified under this frame type in [1].

Introduction

This example shows how WLAN MAC frames specified in Section 9.3 of [1] or [2] can be decoded. It also shows how aggregated MAC frames specified in Section 9.7 of [1] or [2] can be deaggregated.

WLAN Toolbox™ supports MPDU decoding for the following MAC frames:

- **Management Frames:** Beacon
- **Data Frames:** Data, Null, QoS Data, QoS Null
- **Control Frames:** RTS, CTS, Ack, Block Ack

In addition to MPDU decoding, WLAN Toolbox also supports deaggregation of an A-MPDU.

MPDU Decoding

An MPDU can be a data, control or management frame type. `wlanMPDUDecode` can be used to decode an MPDU. This function processes the given MPDU and a physical layer configuration object to output the decoded MAC parameters.

To illustrate MPDU decoding, a valid MPDU is created using `wlanMACFrame`. The created MPDU is passed to the `wlanMPDUDecode` function and the outputs are observed.
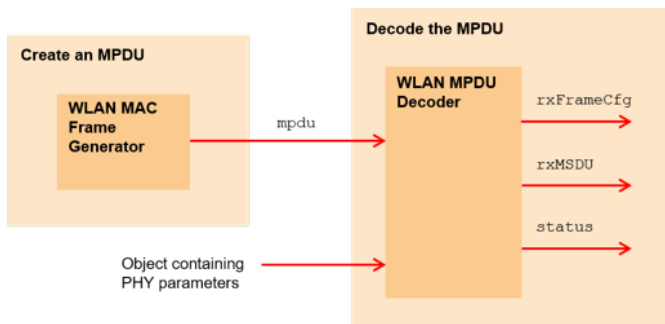
Create an MPDU

A QoS Data frame is created for this example using `wlanMACFrame`. The following inputs are required to form a Non-HT format QoS Data frame containing a 40-octet payload:

1 `txFrameCfg` : A MAC frame configuration object of type `wlanMACFrameConfig`.

2 `txMSDU` : A 40-octet payload (MSDU) to be included in the QoS Data frame.

```
% Create a MAC frame configuration object
txFrameCfg = wlanMACFrameConfig('FrameType', 'QoS Data', ...
                                'FrameFormat', 'Non-HT');

% 40-octet payload for each 'QoS Data' frame
txMSDU = randi([0, 255], 40, 1);

% Physical layer configuration
phyCfg = wlanNonHTConfig;

% Create the MPDU
mpdu = wlanMACFrame(txMSDU, txFrameCfg);
```

Decode the MPDU

`wlanMPDUDecode` consumes an MPDU, a PHY configuration object of type `wlanNonHTConfig`, `wlanHTConfig`, `wlanVHTConfig`, or `wlanHESUConfig` and optionally a (Name, Value) pair for `DataFormat` specifying the input format of the MPDU. Since the MPDU generated using `wlanMACFrame` is in terms of octets, `DataFormat` is set to `octets`. `wlanMPDUDecode` decodes the MPDU and outputs the following information:

1 `rxFrameCfg` : A MAC frame configuration object of type `wlanMACFrameConfig`, containing the decoded MAC parameters.

2 `rxMSDU` : A cell array, where each element is an n-by-2 character array representing the decoded MSDU. Multiple MSDUs are returned when the MPDU contains an aggregated MSDU (A-MSDU) as the payload.

3 `status` : An enumeration of type "status", which indicates whether the MPDU decoding was successful.

```
% Decode the MPDU.
[rxFrameCfg, rxMSDU, status] = wlanMPDUDecode(mpdu, phyCfg, ...
                                    'DataFormat', 'octets');

% Check if the MPDU is decoded successfully
disp(['Status of the MPDU decoding: ' char(status)])
```

```matlab
% Observe the outputs, if the MPDU is decoded successfully
if strcmp(status, 'Success')
    disp(['Type of the decoded MPDU: ' rxFrameCfg.FrameType])
    disp(['Number of MSDUs in the MPDU: ' num2str(numel(rxMSDU))])
    for i = 1:numel(rxMSDU)
        disp(['Size of MSDU-' num2str(i) ': ' num2str(size(rxMSDU{i}, 1)) ' octets'])
    end
end
```

```
Status of the MPDU decoding: Success
Type of the decoded MPDU: QoS Data
Number of MSDUs in the MPDU: 1
Size of MSDU-1: 40 octets
```

A-MPDU Deaggregation

An A-MPDU is an aggregation of multiple MPDUs. The type of MPDUs in an A-MPDU are restricted as specified in Section 9.7.3 of [1].

`wlanAMPDUDeaggregate` can be used to deaggregate an A-MPDU. This function processes the given A-MPDU and the corresponding physical layer configuration object to output the deaggregated list of MPDUs. `wlanAMPDUDeaggregate` is capable of decoding HT (High Throughput), VHT (Very High Throughput), HE-SU (High Efficiency Single User) and HE-EXT-SU (High Efficiency Extended Range Single User) format A-MPDUs as specified in [1] and [2].

To illustrate the A-MPDU deaggregation, a valid A-MPDU containing five MPDUs is created using `wlanMACFrame`. The created A-MPDU is passed to the `wlanAMPDUDeaggregate` function and the outputs are observed.
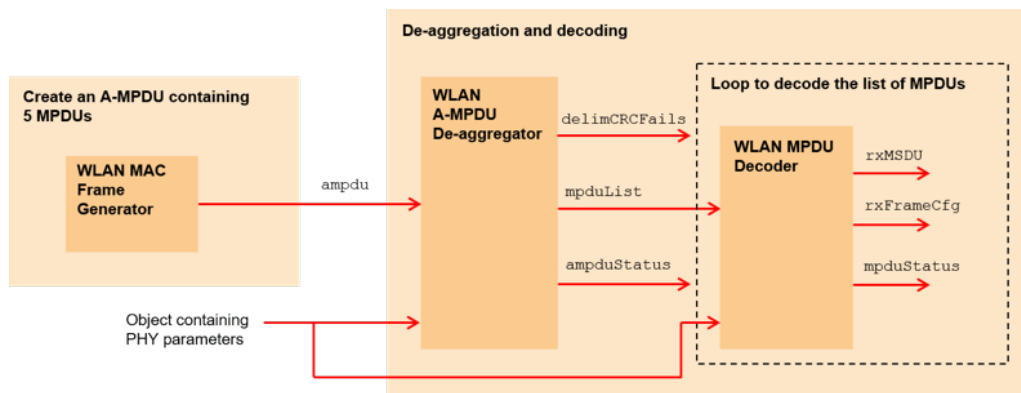
Create an A-MPDU

The following inputs are required to form an HE-SU format A-MPDU containing five MPDUs (QoS Data frames), each MPDU containing a 40-octet payload:

1 `txFrameCfg` : A MAC frame configuration object of type `wlanMACFrameConfig`.

2 `txMSDUList` : A five element cell array containing payload (MSDU) for five MPDUs. Since `MSDUAggregation` is set to false in the `txFrameCfg`, a separate MPDU is created for each MSDU.

3 `phyCfg` : A physical layer configuration object of type `wlanHESUConfig`.

```
% Create a MAC frame configuration object
txFrameCfg = wlanMACFrameConfig('FrameType', 'QoS Data', ...
                                'FrameFormat', 'HE-SU', ...
                                'MPDUAggregation', true, ...
                                'MSDUAggregation', false);

% 40-octet payload for each 'QoS Data' frame
txMSDUList = repmat({randi([0, 255], 40, 1)}, 1, 5);

% Physical layer configuration
phyCfg = wlanHESUConfig('MCS', 3);

% Create the A-MPDU containing 5 MPDUs
ampdu = wlanMACFrame(txMSDUList, txFrameCfg, phyCfg);
```

Deaggregate the A-MPDU

wlanAMPDUDeaggregate consumes an A-MPDU, a PHY configuration object of type wlanHTConfig, wlanVHTConfig, or wlanHESUConfig and optionally a (Name, Value) pair for DataFormat specifying the input format of the A-MPDU. It finds and validates the MPDU delimiters, extracts the MPDUs and outputs the following information that can be used for further processing the MPDUs:

1   mpduList : A cell array containing the list of MPDUs extracted from the A-MPDU.

2   delimCRCFails : A logical row vector representing delimiter CRC validity for the corresponding index in mpduList. A value of true represents that the MPDU present in mpduList at the corresponding index may not be properly extracted.

3   ampduStatus : An enumeration of type "status", which indicates whether the A-MPDU deaggregation was successful.

```
% Deaggregate the A-MPDU
[mpduList, delimCRCFails, ampduStatus] = wlanAMPDUDeaggregate(ampdu, phyCfg, ...
                                                 'DataFormat', 'octets');

% Observe the outputs
disp(['Status of A-MPDU deaggregation: ' char(ampduStatus)])
disp(['Number of MPDUs extracted from the A-MPDU: ' num2str(numel(mpduList))])
disp(['Number of MPDUs with delimiter CRC fails: ' num2str(nnz(delimCRCFails))])

Status of A-MPDU deaggregation: Success
Number of MPDUs extracted from the A-MPDU: 5
Number of MPDUs with delimiter CRC fails: 0
```

Decode the list of MPDUs

The mpduList contains the list of MPDUs extracted from the A-MPDU. Each of the MPDUs present in the list can be decoded separately. However, if the delimCRCFails contains any true values, the MPDU present in mpduList at the corresponding index can be considered invalid as it may not be properly extracted because of the delimiter CRC failure.

```
% Decode the list of MPDUs
if strcmp(ampduStatus, 'Success')
    % Number of MPDUs in the list
    numMPDUs = numel(mpduList);

    for i = 1:numMPDUs
        % Decode the MPDU only if the corresponding delimiter CRC is valid
```

```
        if ~delimCRCFails(i)
            [rxFrameCfg, rxMSDU, mpduStatus] = wlanMPDUDecode(mpduList{i}, phyCfg, ...
                                                'DataFormat', 'octets');
            disp(['MPDU-' num2str(i) ' decoding status: ' char(mpduStatus)])
            disp(['MPDU-' num2str(i) ' type: ' rxFrameCfg.FrameType])
            disp(['MPDU-' num2str(i) ' payload size: ' num2str(size(rxMSDU{1}, 1)) ' octets'])
            disp(' ')
        end
    end
end

MPDU-1 decoding status: Success
MPDU-1 type: QoS Data
MPDU-1 payload size: 40 octets

MPDU-2 decoding status: Success
MPDU-2 type: QoS Data
MPDU-2 payload size: 40 octets

MPDU-3 decoding status: Success
MPDU-3 type: QoS Data
MPDU-3 payload size: 40 octets

MPDU-4 decoding status: Success
MPDU-4 type: QoS Data
MPDU-4 payload size: 40 octets

MPDU-5 decoding status: Success
MPDU-5 type: QoS Data
MPDU-5 payload size: 40 octets
```

Conclusion and Further Exploration

This example demonstrated how to deaggregate and decode IEEE 802.11 MAC frames. You can also explore "802.11 OFDM Beacon Receiver with Captured Data" on page 4-41 and "Recovery Procedure for an 802.11ac Packet" on page 4-30 examples for decoding the MAC frames retrieved from the captured waveforms.

Selected Bibliography

1 IEEE Std 802.11™-2016 IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications

2 IEEE P802.11ax™/D4.1 Draft 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 6: Enhancements for High Efficiency WLAN

802.11ac Waveform Generation with MAC Frames

This example shows how to generate an IEEE® 802.11ac™ transmission containing MAC frames suitable for performing radio packet error rate (PER) receiver tests.

Introduction

WLAN Toolbox™ can be used to generate standard compliant waveforms for performing receiver tests. A basic WLAN receiver test scenario is shown in the diagram below.
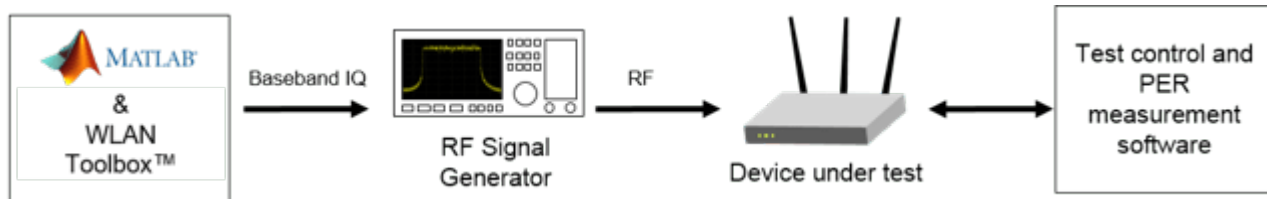
The device under test (DUT) is stimulated with RF test vectors, usually through a wired link. The packet error rate (PER) is a metric used to test the performance of a receiver at a given receive signal power in the presence of noise, interference, or other impairments. The PER is defined as the number of incorrectly decoded packets divided by the total number of transmitted packets.

The frame check sequence (FCS) within a MAC frame is used to determine whether a MAC frame has been decoded correctly by the receiver, and therefore whether the packet has been received in error. The general MAC frame for IEEE 802.11ac contains the following fields:

- MAC header
- Frame body
- FCS

The data to transmit from a higher layer is contained within the frame body of the MAC frame. The transmitter uses a cyclic redundancy check over the MAC header and frame body field to generate the FCS value. The receiver calculates the CRC and compares this to the received FCS field to determine if an error has occurred during transmission.

In this example an IEEE 802.11ac waveform consisting of multiple VHT format packets is generated. The `wlanWaveformGenerator` function can be used to generate a waveform containing one or more packets. The `wlanWaveformGenerator` function consumes physical layer service data units (PSDUs) for each packet and performs the appropriate physical layer processing to create the waveform. A PSDU containing a MAC header and valid FCS can be generated using the `wlanMACFrame` function. In this example a multi-packet baseband waveform containing MAC packets is synthesized. This waveform may be downloaded to a signal generator for RF transmission and used for receiver PER testing. Source code is provided to download and play the waveform using a Keysight Technologies™ N5172B signal generator. The example processing is illustrated in the following diagram:
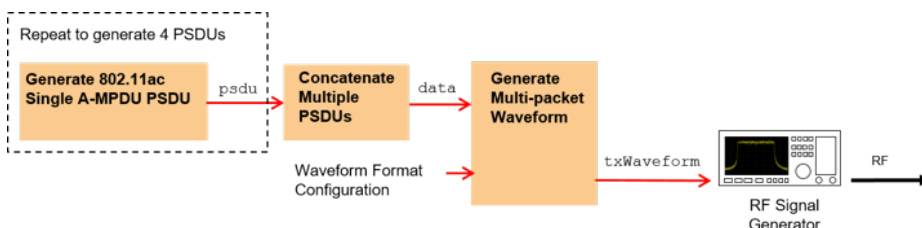
IEEE 802.11ac VHT Format Configuration

The format-specific configuration of a VHT waveform synthesized with the `wlanWaveformGenerator` function is described by the VHT format configuration object, `wlanVHTConfig`. The properties of the object contain the configuration. In this example an object is configured for a 160 MHz bandwidth, 1 transmit antenna, 1 space-time stream and QPSK rate 1/2 (MCS 1).

```
vhtCfg = wlanVHTConfig;               % Create packet configuration
vhtCfg.ChannelBandwidth = 'CBW160';   % 160 MHz channel bandwidth
vhtCfg.NumTransmitAntennas = 1;       % 1 transmit antenna
vhtCfg.NumSpaceTimeStreams = 1;       % 1 space-time stream
vhtCfg.MCS = 1;                       % Modulation: QPSK Rate: 1/2
```

Waveform Generation Configuration

The `wlanWaveformGenerator` function can be configured to generate one or more packets and add an idle time between each packet. In this example four packets with a 20 microsecond idle period will be created.

```
numPackets = 4;   % Generate 4 packets
idleTime = 20e-6; % 20 microseconds idle period after packet
```

The PSDU transmitted in each packet is scrambled using a random seed for each packet. This is accomplished by specifying a vector of scrambler initialization seeds. The valid range of the seed is between 1 and 127 inclusive.

```
% Initialize the scrambler with a random integer for each packet
scramblerInitialization = randi([1 127],numPackets,1);
```

Create a PSDU for Each Packet

For an IEEE 802.11ac data transmission the MAC frame is termed a MAC protocol data unit (MPDU), the MAC header is termed the MPDU header, and the frame body is an aggregated MAC service data unit (A-MSDU). One or more MPDUs are delimited, padded and aggregated to create an aggregated MPDU (A-MPDU). The A-MPDU is delimited and padded to form the physical layer service data unit (PSDU) which is coded and modulated to create the transmitted packet. This process of encapsulation is shown in the following diagram:
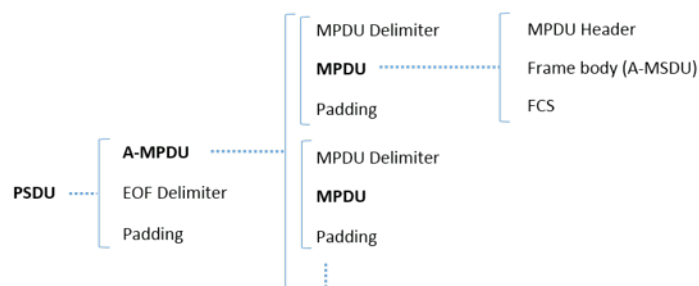
In this example a PSDU is created containing a single MPDU for each packet. The MPDU consists of an MPDU header, A-MSDU frame containing concatenated A-MSDU subframes with random data and valid FCS. The `wlanMACFrame` function creates an A-MPDU with EOF delimiters and padding, i.e. the PSDU, as specified in [1]. It also returns the length of the A-MPDU, termed as the APEP Length, which is used to set the `APEPLength` property of the VHT configuration object. A PSDU is generated for each packet and is concatenated into a vector `data` for transmission with the

`wlanWaveformGenerator` function. The processing to create the concatenated PSDU bits `data` is shown in the diagram below:
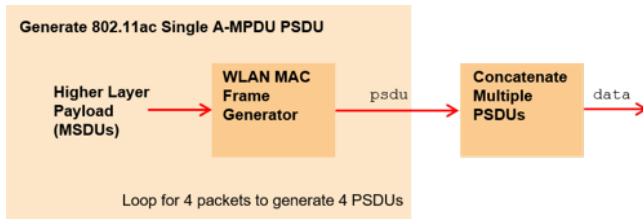

```matlab
% Create frame configuration
macCfg = wlanMACFrameConfig('FrameType', 'QoS Data');
macCfg.FrameFormat = 'VHT';        % Frame format
macCfg.MSDUAggregation = true;     % Form A-MSDUs internally
bitsPerByte = 8;                   % Number of bits in 1 byte
data = [];

for i=1:numPackets
    % Get MSDU lengths to create a random payload for forming an A-MPDU of
    % 4048 octets (pre-EOF padding)
    msduLengths = wlanMSDULengths(4048, macCfg, vhtCfg);
    msdu = cell(numel(msduLengths), 1);

    % Create MSDUs with the obtained lengths
    for j = 1:numel(msduLengths)
        msdu{j} = randi([0 255], 1, msduLengths(j));
    end

    % Generate PSDU bits containing A-MPDU with EOF delimiters and padding
    [psdu, apepLength] = wlanMACFrame(msdu, macCfg, vhtCfg, 'OutputFormat', 'bits');

    % Set the APEP length in the VHT configuration
    vhtCfg.APEPLength = apepLength;

    % Concatenate packet PSDUs for waveform generation
    data = [data; psdu]; %#ok<AGROW>
end
```

Generate a Baseband Waveform

The concatenated PSDU bits for all packets, `data`, are passed as an argument to the `wlanWaveformGenerator` function along with the VHT packet configuration object `vhtCfg`. This configures the waveform generator to synthesize an 802.11ac VHT waveform. To generate 802.11n™ HT or other format waveforms, use a different format configuration object, for example `wlanHTConfig` or `wlanNonHTConfig`. The waveform generator is additionally configured using name-value pairs to generate multiple packets with a specified idle time between packets, and initial scrambler states.

```matlab
% Generate baseband VHT packets
txWaveform = wlanWaveformGenerator(data,vhtCfg, ...
    'NumPackets',numPackets,'IdleTime',idleTime, ...
    'ScramblerInitialization',scramblerInitialization);

fs = wlanSampleRate(vhtCfg);
disp(['Baseband sampling rate: ' num2str(fs/1e6) ' Msps']);
```

Baseband sampling rate: 160 Msps

The magnitude of the baseband waveform is displayed below. Note the number of packets and idle time configured.

```
figure;
plot(abs(txWaveform));
xlabel('Sample index');
ylabel('Magnitude');
title('Baseband IEEE 802.11ac Waveform');
legend('Transmit antenna 1');
```
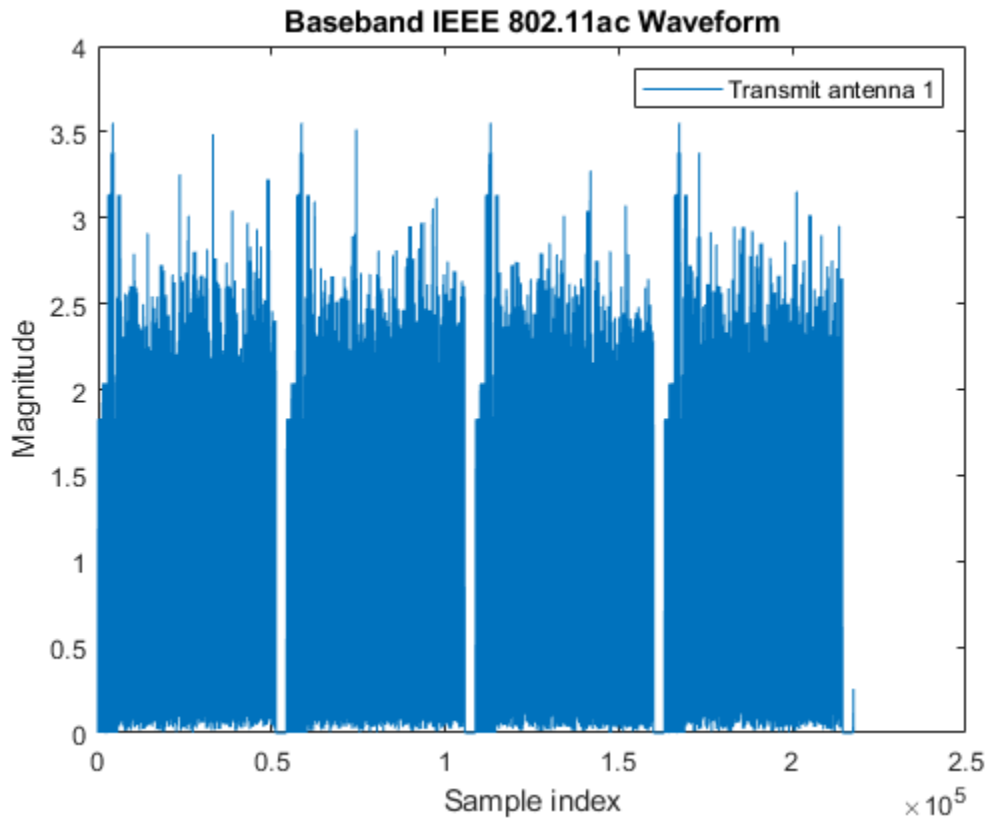

The frequency spectrum of the generated time domain waveform, txWaveform, can be viewed using the DSP System Toolbox™ dsp.SpectrumAnalyzer. As expected, the 160 MHz signal bandwidth is clearly visible at baseband.

```
spectrumAnalyzer = dsp.SpectrumAnalyzer;
spectrumAnalyzer.SampleRate = fs;
spectrumAnalyzer.SpectrumType = 'Power density';
spectrumAnalyzer.RBWSource = 'Property';
spectrumAnalyzer.RBW = 100e3;
spectrumAnalyzer.AveragingMethod = 'Exponential';
spectrumAnalyze.ForgettingFactor = 0.99;
spectrumAnalyzer.YLabel = 'PSD';
spectrumAnalyzer.YLimits = [-80 -40];
spectrumAnalyzer.Title = 'Baseband IEEE 802.11ac Waveform';
spectrumAnalyzer(txWaveform);
```
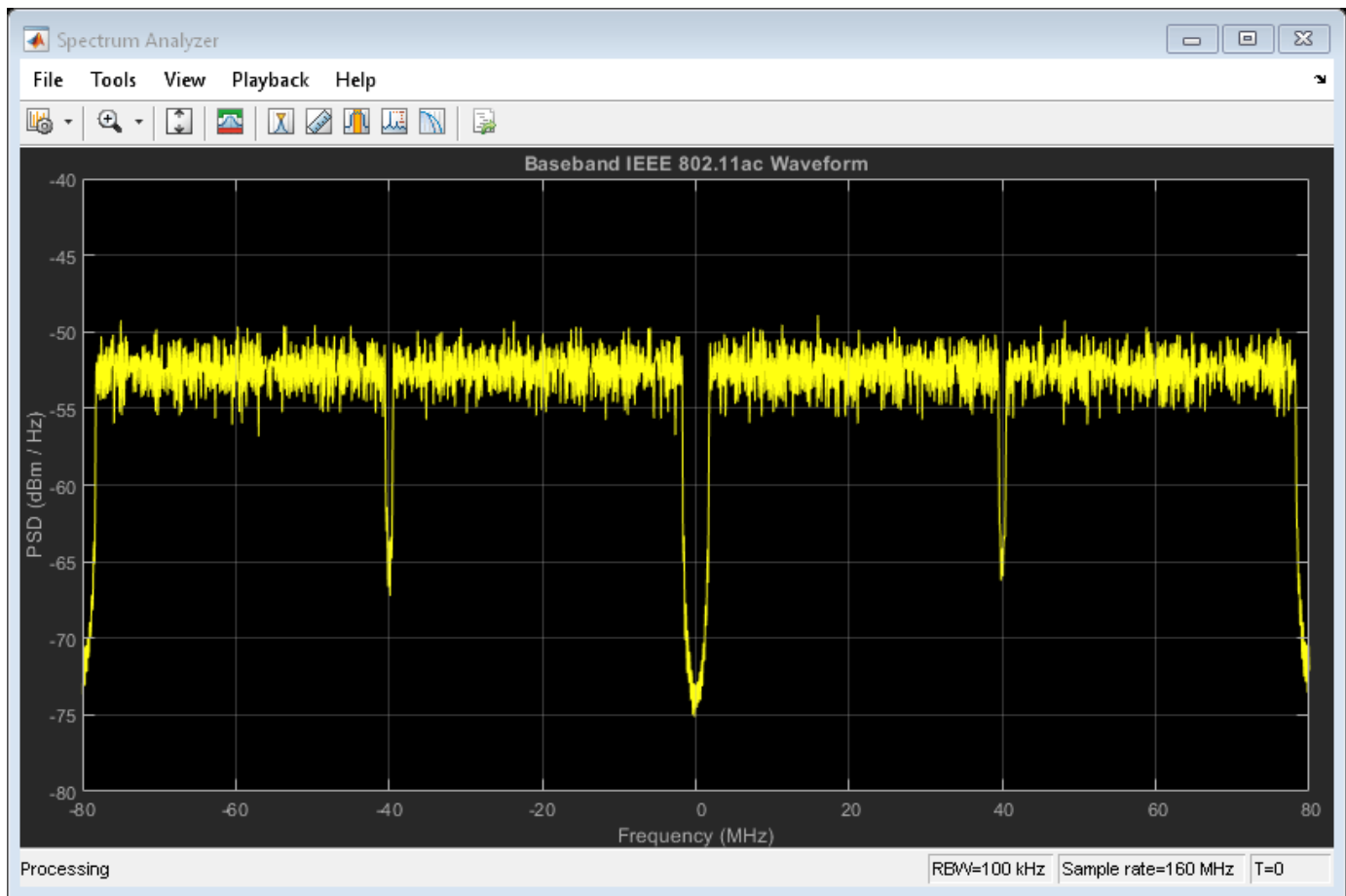
Generate an Over-the-Air Signal Using an RF Signal Generator

The baseband waveform created by WLAN Toolbox can now be downloaded to a signal generator to perform receiver tests. Instrument Control Toolbox™ is used to generate an RF signal with a center frequency of 5.25 GHz RF using the Keysight Technologies N5172B signal generator.

```
% Control whether to download the waveform to the waveform generator
playOverTheAir = false;

% Download the baseband IQ waveform to the instrument. Generate the RF
% signal at a center frequency of 5.25 GHz and output power of -10 dBm.
if playOverTheAir
    fc = 5.25e9; %#ok<UNRCH> % Center frequency
    power = -10;      % Output power
    loopCount = Inf; % Number time to loop

    % Configure the signal generator, download the waveform and loop
    rf = rfsiggen();
    rf.Resource = 'TCPIP0::192.168.0.1::inst0::INSTR';
    rf.Driver = 'AgRfSigGen';
    connect(rf);                      % Connect to the instrument
    download(rf,txWaveform.',fs); % Download the waveform to the instrument
    start(rf,fc,power,loopCount); % Start transmitting waveform

    % When you have finished transmitting, stop the waveform output
```

```
    stop(rf);
    disconnect(rf);
end
```

Selected Bibliography

1   IEEE Std 802.11ac™-2013 IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz.

802.11 OFDM Beacon Frame Generation

This example shows how to generate packets containing MAC beacon frames suitable for baseband simulation or over-the-air transmission using an SDR platform.

Introduction

This example creates an IEEE® 802.11™ beacon frame as described in section 9.3.3.3 of [1]. A WiFi device can be used to view the beacon packet transmitted using SDR hardware as shown in the figure below.
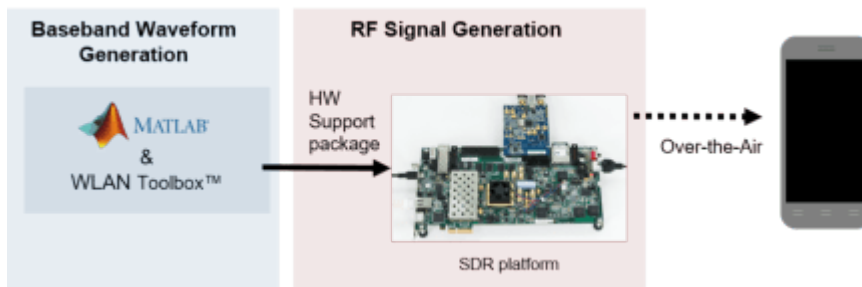
The beacon frame is a type of management frame, it identifies a basic service set (BSS) formed by a number of 802.11 devices. The access point of a BSS periodically transmits the beacon frame to establish and maintain the network. The beacon frame consists of a MAC header, a beacon frame body and a valid frame check sequence (FCS). The beacon frame body contains the information fields which allows stations to associate with the network. A WLAN beacon frame is created using the `wlanMACFrame` function. The beacon frame is encoded and modulated using the `wlanWaveformGenerator` function to create a baseband beacon packet. In this example the generated waveform can be:

- Stored in a baseband file format. The file format can be used with the example "802.11 OFDM Beacon Receiver with Captured Data" on page 4-41, which performs beacon packet decoding and describes the receiver processing.

- Transmitted over-the-air. The beacon packet is upconverted for RF transmission using Xilinx® Zynq-Based Radio SDR hardware. The radio hardware allows a waveform to be transmitted over-the-air.
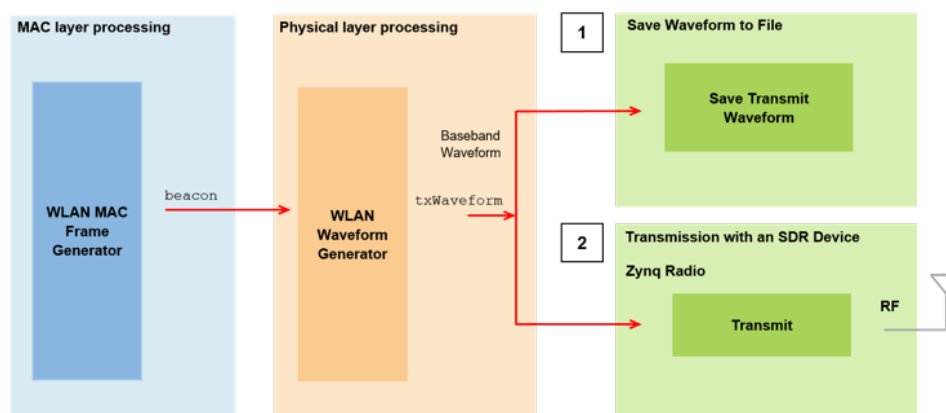
To transmit the beacon over-the-air, the Xilinx Zynq-based radio support package is required. This can be installed using the Add-On Explorer. More information about SDR platforms can be found here.

Example Setup

The beacon packet can be written to a baseband file and transmitted using an SDR platform. To transmit the beacon using the SDR platform set `useSDR` to true. To write to a baseband file set `saveToFile` to true.

```
useSDR = false;
saveToFile = false;
```

Create IEEE 802.11 Beacon Frame

The beacon packets are periodically transmitted as specified by the Target Beacon Transmission Time (TBTT) in the beacon interval field. The beacon interval represents the number of Time Units (TUs) between TBTT, where 1 TU represents 1024 microseconds. A beacon interval of 100 TU results in a 102.4 milliseconds time interval between successive beacons. A beacon frame is generated using the `wlanMACFrame` function. This function consumes the MAC frame configuration object `wlanMACFrameConfig`. This object accepts `wlanMACManagementConfig` as a property to configure the beacon frame-body.

```
SSID = 'TEST_BEACON'; % Network SSID
beaconInterval = 100; % In Time units (TU)
band = 5;             % Band, 5 or 2.4 GHz
chNum = 52;           % Channel number, corresponds to 5260MHz
bitsPerByte = 8;      % Number of bits in 1 byte

% Create Beacon frame-body configuration object
frameBodyConfig = wlanMACManagementConfig;
frameBodyConfig.BeaconInterval = beaconInterval;  % Beacon Interval in Time units (TUs)
frameBodyConfig.SSID = SSID;                       % SSID (Name of the network)
dsElementID = 3;                                   % DS Parameter IE element ID
dsInformation = dec2hex(chNum, 2);                 % DS Parameter IE information
frameBodyConfig = frameBodyConfig.addIE(dsElementID, dsInformation);  % Add DS Parameter IE to th

% Create Beacon frame configuration object
beaconFrameConfig = wlanMACFrameConfig('FrameType', 'Beacon');
beaconFrameConfig.ManagementConfig = frameBodyConfig;

% Generate Beacon frame bits
[beacon, mpduLength] = wlanMACFrame(beaconFrameConfig, 'OutputFormat', 'bits');

% Calculate center frequency for the given band and channel number
fc = helperWLANChannelFrequency(chNum, band);
```

Create IEEE 802.11 Beacon Packet

A beacon packet is synthesized using `wlanWaveformGenerator` with a non-HT format configuration object. In this example an object is configured to generate a beacon packet of 20 MHz bandwidth, 1 transmit antenna and BPSK rate 1/2 (MCS 0).

```
cfgNonHT = wlanNonHTConfig;            % Create a wlanNonHTConfig object
cfgNonHT.PSDULength = mpduLength;      % Set the PSDU length in bytes

% The idle time is the length in seconds of an idle period after each
% generated packet. The idle time is set to the beacon interval.
```

```
txWaveform = wlanWaveformGenerator(beacon, cfgNonHT, 'IdleTime', beaconInterval*1024e-6);
Rs = wlanSampleRate(cfgNonHT);              % Get the input sampling rate
```

Save Waveform to File

This section saves the waveform in a baseband file using `comm.BasebandFileWriter`.

```
if saveToFile
    % The waveform is stored in a baseband file
    BBW = comm.BasebandFileWriter('nonHTBeaconPacket.bb', Rs, fc); %#ok<UNRCH>
    BBW(txWaveform);
    release(BBW);
end
```

For information about automatically detecting and synchronizing the waveform stored in the baseband file format see "802.11 OFDM Beacon Receiver with Captured Data" on page 4-41.

Transmission with an SDR Device

This section demonstrates the transmission of the beacon packet over-the-air using an SDR device.

```
if useSDR
    % The SDR platform used must support |transmitRepeat|. Valid platforms
    % are 'AD936x', 'E3xx', and 'Pluto'.
    sdrPlatform = 'AD936x'; %#ok<UNRCH>
    tx = sdrtx(sdrPlatform);
    osf = 2; % OverSampling factor
    tx.BasebandSampleRate = Rs*osf;
    % The center frequency is set to the corresponding channel number
    tx.CenterFrequency = fc;
end
```

The transmitter gain `tx.Gain` parameter drives the power amplifier in the radio. This parameter is used to impair the quality of the waveform, you can change this parameter to reduce transmission quality, and impair the signal. These are suggested values, depending on your antenna configuration, you may have to tweak these values. The suggested values are:

1 Set to 0 for increased gain (0dB)

2 Set to -10 for default gain (-10dB)

3 Set to -20 for reduced gain (-20dB)

The `transmitRepeat` function transfers the baseband waveform to the SDR platform, and stores the signal samples in hardware memory. The example then repeatedly transmits this waveform over-the-air until the release method of the transmit object is called. Messages are displayed in the command window to confirm that transmission has started successfully.

```
if useSDR
    % Set transmit gain
    tx.Gain = 0;   %#ok<UNRCH>
    % Resample transmit waveform
    txWaveform = resample(txWaveform, osf, 1);
    % Transmit over-the-air
    transmitRepeat(tx, txWaveform);
end
```

Conclusion and Further Exploration

This example has demonstrated how to generate a beacon packet for the IEEE 802.11 standard. A Wi-Fi™ device can be used to view the beacon packet transmitted using SDR hardware. Alternatively, the stored baseband beacon packet can be processed to recover the transmitted information using the example "802.11 OFDM Beacon Receiver with Captured Data" on page 4-41.

Appendix

This example uses the following helper function:

- helperWLANChannelFrequency.m

Selected Bibliography

1   IEEE Std 802.11™-2016 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.

Signal Transmission

802.11be 4096-QAM 320 MHz Waveform Generation and Analysis

This example shows how to generate an IEEE® 802.11be™ single-user waveform with 4096-point quadrature amplitude modulation (4096-QAM) and a channel bandwidth of 320 MHz. The example demonstrates how to measure transmitter modulation accuracy, spectral mask, and spectral flatness.

Introduction

This example shows how to parameterize and generate an IEEE 802.11be extremely high throughput (EHT) single-user waveform specified in P802.11be Draft 0.1 [1] on page 3-0 . This example performs transmitter modulation accuracy, required spectral mask, and required spectral flatness measurements on the waveform for the configuration specified in Section 36.3.18 of [1 on page 3-0].

The example generates 20 single-user EHT packets with a 320 MHz channel bandwidth and a 10 microsecond gaps between packets. Each packet contains random data and uses 4096-QAM. The example oversamples the waveform using a larger IFFT than required for the nominal baseband rate and does not perform spectral filtering. The example uses a high-power amplifier (HPA) model, which introduces in-band distortion and spectral regrowth. The example performs spectral emission mask measurement on the upsampled waveform after the high-power amplifier modeling. The example decodes the EHT data field and measures the error vector magnitude (EVM) to determine the modulation accuracy after downsampling the waveform to baseband sampling rate. Additionally, the example measures the spectral flatness of the recovered waveform. This diagram shows the workflow contained in the example.
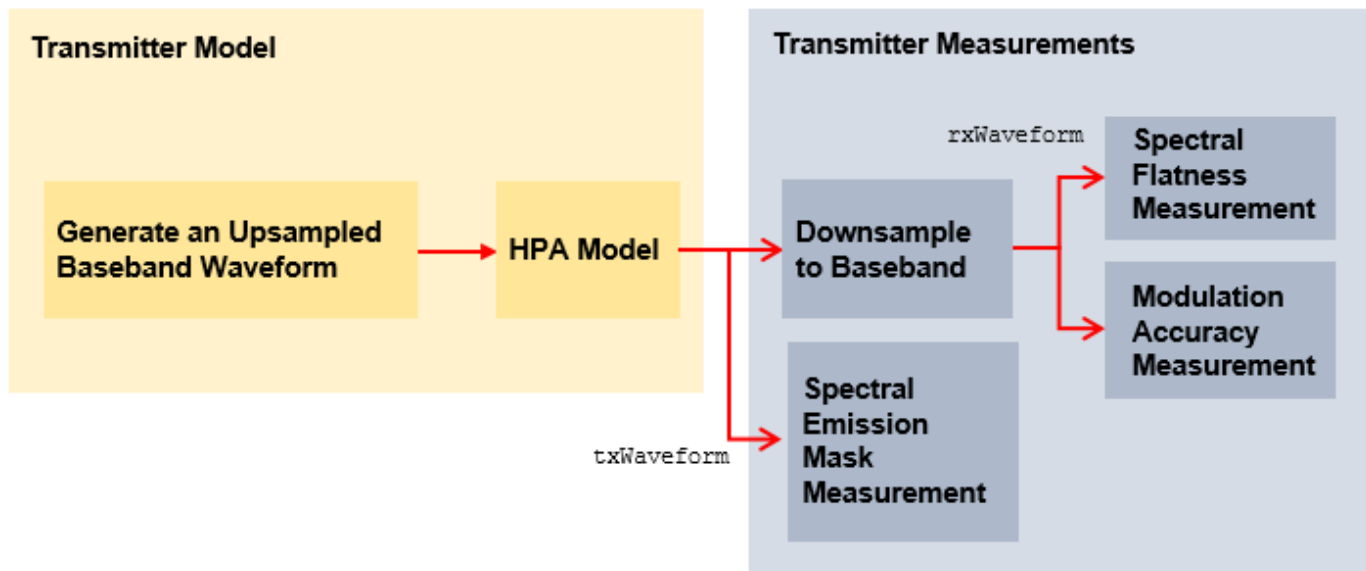
Simulation Setup

Configure the example to `generate` 20 single-user EHT packets with a 10 microsecond idle period between each packet.

```
numPackets =  20          ;
idleTime   =  10          ; % In microseconds
```

802.11be Waveform Configuration and Generation

The physical layer of 802.11be, defined in P802.11be Draft 0.1 [1] on page 3-0 , builds upon 802.11ax [2 on page 3-0] with these added capabilities and modifications.

1 Increased maximum channel bandwidth of 320 MHz

2 4096-QAM in MCS index options 12 and 13

3 Assignment of multiple resource unit (MRU) to a single station (STA), with optional puncturing of single-user packets

4 Increased maximum number of spatial streams for uplink MU-MIMO (16 spatial streams across up to 8 users)

5 Modified tone plan for 80 and 160 MHz bandwidths

6 Larger generator polynomial for Data field scrambler and descrambler

This example models these key 802.11be features.

1 320 MHz channel bandwidth

2 4096-QAM modulation

3 Modified tone plan for 80 and 160 MHz bandwidths

4 Larger generator polynomial for Data field scrambler and descrambler

The draft standard defines the EHT multi-user (EHT MU) format for the transmission of both single-user and multi-user packets. This example only supports generation of single-user packets. Use the EHT MU configuration object, `ehtMUConfig`, to configure transmission properties of an EHT MU packet. This example uses random bits to generate the U-SIG and EHT-SIG signaling fields. You can specify the random seed or field bit values by modifying the relevant properties of the `ehtMUConfig` object. The value of the U-SIG and EHT-SIG bits affects the peak-to-average-power ratio (PAPR) of the packet. Because this example does not use space-time block coding, it can measure the modulation accuracy per spatial stream.

```
chanBW     =  CBW320    ▼ ; % Channel bandwidth

mcs        =  12        ▼ ; % Modulation and coding scheme

numTx      =  1          ; % Number of transmit antennas

apepLength =  8000       ; % A-MPDU length pre-EOF padding in bytes

savedState = rng(0); % Set random state

cfgEHT = ehtMUConfig(chanBW); % EHT MU configuration object
cfgEHT.NumTransmitAntennas = numTx;
cfgEHT.User{1}.MCS = mcs;
cfgEHT.User{1}.NumSpaceTimeStreams = numTx;
cfgEHT.User{1}.APEPLength = apepLength;
cfgEHT.RU{1}.SpatialMapping = 'Direct';
```

To model the effect of a high-power amplifier on the waveform and view the out-of-band spectral emissions, the waveform must be oversampled. This example generates the waveform using a larger

IFFT than required for the nominal baseband rate, resulting in an oversampled waveform. The example does not perform spectral filtering.

```matlab
osf = [ 4            ]; % Oversampling factor
```

Create random bits for all packets.

```matlab
psduLength = cfgEHT.getPSDULength*8; % PSDU length in bits per packet
data = randi([0 1],psduLength*numPackets,1);
```

Generate the EHT MU waveform for the specified bits and configuration by using the `ehtWaveformGenerator` function, specifying the desired oversampling factor, number of packets, and idle time between each packet.

```matlab
txWaveform = ehtWaveformGenerator(data,cfgEHT,'NumPackets',numPackets,'IdleTime',idleTime*1e-6,'(
```

Get the baseband sampling rate of the waveform

```matlab
fs = ehtSampleRate(cfgEHT);
disp(['Baseband sampling rate: ' num2str(fs/1e6) ' Msps']);
```

```
Baseband sampling rate: 320 Msps
```

Prepend zeros to the waveform to allow for early timing synchronization.

```matlab
txWaveform = [zeros(round(idleTime*1e-6*fs),numTx); txWaveform];
```

Add Impairments

High-Power Amplifier Modeling

The high-power amplifier introduces nonlinear behavior in the form of in-band distortion and spectral regrowth. This example simulates the power amplifiers by using the Rapp model [3 on page 3-0   ], which introduces AM/AM distortion.

Model the amplifier by using the `dsp.FIRInterpolator` object, and configure reduced distortion by specifying a back-off, `hpaBackoff`, such that the amplifier operates below its saturation point. You can increase the backoff to reduce EVM for higher MCS values.

```matlab
pSaturation = 25; % Saturation power of a power amplifier in dBm
hpaBackoff = 16;  % Power amplifier backoff in dB
nonLinearity = comm.MemorylessNonlinearity;
nonLinearity.Method = 'Rapp model';
nonLinearity.Smoothness = 3; % p parameter
nonLinearity.LinearGain = -hpaBackoff;
nonLinearity.OutputSaturationLevel = db2mag(pSaturation-30);
```

Add the model to each transmit antenna.

```matlab
for i=1:cfgEHT.NumTransmitAntennas
    txWaveform(:,i) = nonLinearity(txWaveform(:,i));
end
```

Thermal Noise

Add thermal noise to the waveform, specifying a 6 dB receiver noise figure [4 on page 3-0   ].

```matlab
NF = 6;         % Noise figure (dB)
BW = fs*osf;    % Bandwidth (Hz)
```

```
k = 1.3806e-23; % Boltzman constant (J/K)
T = 290;        % Ambient temperature (K)
noisePower = 10*log10(k*T*BW)+NF;

awgnChannel = comm.AWGNChannel('NoiseMethod','Variance','Variance',10^(noisePower/10));
txWaveform = awgnChannel(txWaveform);
```

Modulation Accuracy (EVM) and Spectral Flatness Measurements

Downsampling and Filtering

Resample the oversampled waveform down to baseband for physical layer processing and EVM and spectral flatness measurements, applying a low-pass anti-aliasing filter before downsampling. The impact of the low-pass filter is visible in the spectral flatness measurement. The anti-aliasing filter is designed so that all active subcarriers are within the filter passband.

Design resampling filter.

```
aStop = 40; % Stopband attenuation
ofdmInfo = ehtOFDMInfo('EHT-Data',cfgEHT); % OFDM parameters
SCS = fs/ofdmInfo.FFTLength; % Subcarrier spacing
txbw = max(abs(ofdmInfo.ActiveFrequencyIndices))*2*SCS; % Occupied bandwidth
[L,M] = rat(osf);
maxLM = max([L M]);
R = (fs-txbw)/fs;
TW = 2*R/maxLM; % Transition width
b = designMultirateFIR(L,M,TW,aStop);
```

Resample the waveform to baseband.

```
firinterp = dsp.FIRRateConverter(M,L,b);
rxWaveform = firinterp(txWaveform);
```

Receiver Processing

This section detects, synchronizes, and extracts each packet in rxWaveform, then measures the EVM and spectral flatness. For each packet, the example performs these steps.

- Detect the start of the packet
- Extract the legacy fields
- Estimate and correct coarse carrier frequency offset (CFO)
- Perform fine symbol timing estimate by using the frequency-corrected legacy fields
- Extract the packet from the waveform by using the fine symbol timing offset
- Correct the extracted packet with the coarse CFO estimate
- Extract the L-LTF, then estimate and correct the fine CFO
- Extract the EHT-LTF and perform channel estimation for each of the transmit streams
- Measure the spectral flatness by using the channel estimate
- Extract and OFDM demodulate the EHT Data field
- Perform noise estimation by using the demodulated data field pilots and single-stream channel estimate at pilot subcarriers
- Phase-correct and equalize the EHT Data field by using the channel and noise estimates
- For each data-carrying subcarrier in each spatial stream, find the closest constellation point and measure the EVM

- Recover the PSDU by decoding the equalized symbols

This diagram shows the processing chain.
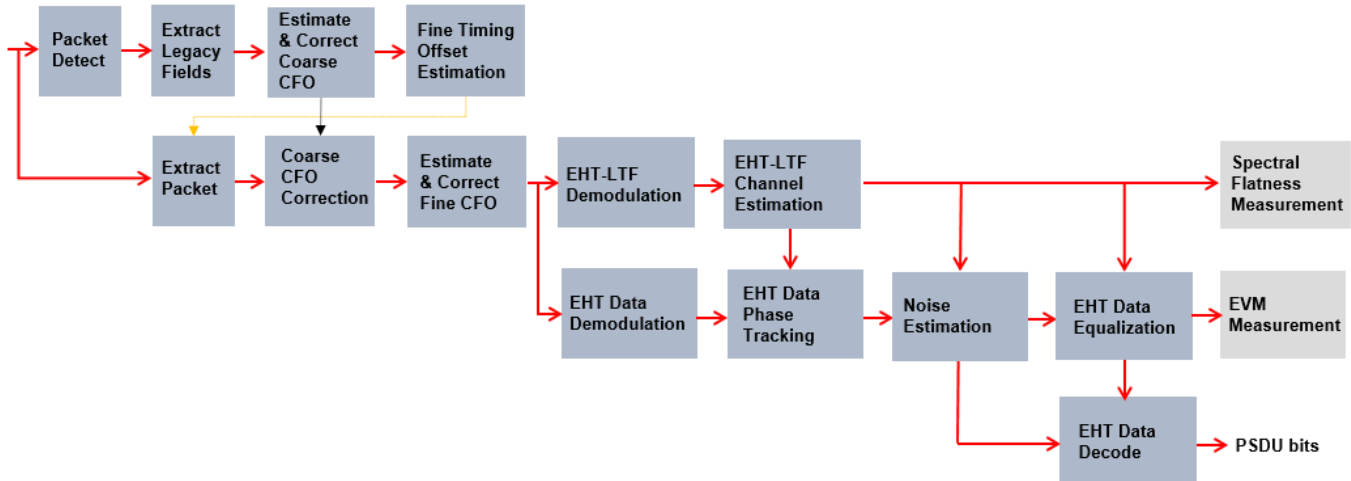
This example makes two different EVM measurements.

1 RMS EVM per packet, which comprises averaging the EVM over subcarriers, OFDM symbols, and spatial streams.

2 RMS EVM per subcarrier per spatial stream for a packet. Because this configuration maps spatial streams directly to antennas, this measurement can help detect frequency-dependent impairments, which may affect individual RF chains differently. This measurement averages the EVM over OFDM symbols only.

Setup EVM measurements.

```
[EVMPerPkt,EVMPerSC] = evmSetup(cfgEHT);
```

Get indices for accessing each field within the time-domain packet.

```
ind = ehtFieldIndices(cfgEHT);
```

Define the minimum detectable length of data, in samples.

```
minPktLen = double(ind.LSTF(2)-ind.LSTF(1))+1;
```

Detect and process packets within the received waveform by using a `while` loop, which performs these steps.

1 Detect a packet by indexing into `rxWaveform` with the sample offset, `searchOffset`

2 Detect and process the first packet within `rxWaveform`

3 Detect and process the next packet by incrementing the sample index offset

4 Repeat until no further packets are detected

```
rxWaveformLength = size(rxWaveform,1);
pktLength = double(ind.EHTData(2));
rmsEVM = zeros(numPackets,1);
pktOffsetStore = zeros(numPackets,1);
```

```matlab
rng(savedState); % Restore random state
pktNum = 0;
searchOffset = 0; % Start at first sample (no offset)
while (searchOffset+minPktLen)<=rxWaveformLength

    % Detect packet and determine coarse packet offset
    pktOffset = ehtPacketDetect(rxWaveform,cfgEHT.ChannelBandwidth,searchOffset);
    % Packet offset from start of the waveform
    pktOffset = searchOffset+pktOffset;
    % Skip packet if L-STF is empty
    if isempty(pktOffset) || (pktOffset<0) || ...
            ((pktOffset+ind.LSIG(2))>rxWaveformLength)
        break;
    end

    % Extract L-STF and perform coarse frequency offset correction
    nonht = rxWaveform(pktOffset+(ind.LSTF(1):ind.LSIG(2)),:);
    coarsefreqOff = ehtCoarseCFOEstimate(nonht,cfgEHT.ChannelBandwidth);
    nonht = helperFrequencyOffset(nonht,fs,-coarsefreqOff);

    % Extract the legacy fields and determine fine packet offset
    lltfOffset = ehtSymbolTimingEstimate(nonht,cfgEHT.ChannelBandwidth);
    pktOffset = pktOffset+lltfOffset; % Determine packet offset

    % If offset is outwith bounds of the waveform, then skip samples and
    % continue searching within remainder of the waveform
    if (pktOffset<0) || ((pktOffset+pktLength)>rxWaveformLength)
        searchOffset = pktOffset+double(ind.LSTF(2))+1;
        continue;
    end

    % Timing synchronization complete; extract the detected packet
    rxPacket = rxWaveform(pktOffset+(1:pktLength),:);
    pktNum = pktNum+1;
    disp(['  Packet ' num2str(pktNum) ' at index: ' num2str(pktOffset+1)]);

    % Apply coarse frequency correction to the extracted packet
    rxPacket = helperFrequencyOffset(rxPacket,fs,-coarsefreqOff);

    % Perform fine frequency offset correction on the extracted packet
    lltf = rxPacket(ind.LLTF(1):ind.LLTF(2),:); % Extract L-LTF
    fineFreqOff = ehtFineCFOEstimate(lltf,cfgEHT.ChannelBandwidth);
    rxPacket = helperFrequencyOffset(rxPacket,fs,-fineFreqOff);

    % Extract EHT-LTF samples, demodulate, and perform channel estimation
    ehtLTF = rxPacket(ind.EHTLTF(1):ind.EHTLTF(2),:);
    ehtLTFDemod = ehtDemodulate(ehtLTF,'EHT-LTF',cfgEHT);

    % Channel estimate
    [chanEst,pilotEst] = ehtLTFChannelEstimate(ehtLTFDemod,cfgEHT);

    % Spectral flatness measurement
    ehtTxSpectralFlatnessMeasurement(chanEst,cfgEHT,pktNum);

    % Data demodulate
    rxData = rxPacket(ind.EHTData(1):ind.EHTData(2),:);
    demodSym = ehtDemodulate(rxData,'EHT-Data',cfgEHT);
```

```matlab
        % Pilot phase tracking
        demodSym = ehtCommonPhaseErrorTracking(demodSym,chanEst,cfgEHT);

        % Estimate noise power in EHT fields
        nVarEst = ehtNoiseEstimate(demodSym(ofdmInfo.PilotIndices,:,:),pilotEst,cfgEHT);

        % Extract data subcarriers from demodulated symbols and channel
        % estimate
        demodDataSym = demodSym(ofdmInfo.DataIndices,:,:);
        chanEstData = chanEst(ofdmInfo.DataIndices,:,:);

        % Equalization and STBC combining
        [eqSym,csi] = ehtEqualizeCombine(demodDataSym,chanEstData,nVarEst,cfgEHT);

        % Compute RMS EVM over all spatial streams for a packet
        rmsEVM(pktNum) = EVMPerPkt(eqSym);
        fprintf('    RMS EVM: %2.2f%%, %2.2fdB\n',rmsEVM(pktNum),20*log10(rmsEVM(pktNum)/100));

        % Compute RMS EVM per subcarrier and spatial stream for the packet
        evmPerSC = EVMPerSC(eqSym); % Nst-by-1-by-Nss

        % Plot RMS EVM per subcarrier and equalized constellation
        ehtTxEVMConstellationPlots(eqSym,evmPerSC,cfgEHT,pktNum);

        % Recover data field bits
        rxPSDU = ehtDataBitRecover(eqSym,nVarEst,csi,cfgEHT);

        if isequal(rxPSDU,data((1:psduLength)+(pktNum-1)*psduLength))
            fprintf('    Decode success\n');
        else
            fprintf('    Decode failure\n');
        end

        % Store the offset of each packet within the waveform
        pktOffsetStore(pktNum) = pktOffset;

        % Increment waveform offset and search remaining waveform for a packet
        searchOffset = pktOffset+pktLength+minPktLen;
end
```

```
  Packet 1 at index: 992

    Spectral flatness passed

    RMS EVM: 0.41%, -47.69dB

    Decode success

  Packet 2 at index: 32352

    Spectral flatness passed

    RMS EVM: 0.43%, -47.33dB

    Decode success

  Packet 3 at index: 63712

    Spectral flatness passed
```

```
  RMS EVM: 0.43%, -47.26dB

  Decode success

Packet 4 at index: 95072

  Spectral flatness passed

  RMS EVM: 0.47%, -46.62dB

  Decode success

Packet 5 at index: 126432

  Spectral flatness passed

  RMS EVM: 0.46%, -46.79dB

  Decode success

Packet 6 at index: 157792

  Spectral flatness passed

  RMS EVM: 0.45%, -46.91dB

  Decode success

Packet 7 at index: 189152

  Spectral flatness passed

  RMS EVM: 0.43%, -47.27dB

  Decode success

Packet 8 at index: 220512

  Spectral flatness passed

  RMS EVM: 0.37%, -48.68dB

  Decode success

Packet 9 at index: 251872

  Spectral flatness passed

  RMS EVM: 0.48%, -46.45dB

  Decode success

Packet 10 at index: 283232

  Spectral flatness passed

  RMS EVM: 0.42%, -47.49dB

  Decode success

Packet 11 at index: 314592

  Spectral flatness passed
```

```
  RMS EVM: 0.37%, -48.69dB

  Decode success

Packet 12 at index: 345952

  Spectral flatness passed

  RMS EVM: 0.42%, -47.53dB

  Decode success

Packet 13 at index: 377312

  Spectral flatness passed

  RMS EVM: 0.43%, -47.27dB

  Decode success

Packet 14 at index: 408672

  Spectral flatness passed

  RMS EVM: 0.45%, -46.90dB

  Decode success

Packet 15 at index: 440032

  Spectral flatness passed

  RMS EVM: 0.47%, -46.58dB

  Decode success

Packet 16 at index: 471392

  Spectral flatness passed

  RMS EVM: 0.45%, -46.99dB

  Decode success

Packet 17 at index: 502752

  Spectral flatness passed

  RMS EVM: 0.41%, -47.84dB

  Decode success

Packet 18 at index: 534112

  Spectral flatness passed

  RMS EVM: 0.41%, -47.69dB

  Decode success

Packet 19 at index: 565472

  Spectral flatness passed
```

```
RMS EVM: 0.54%, -45.29dB

Decode success
```
Packet 20 at index: 596832

Spectral Flatness, Packet:20
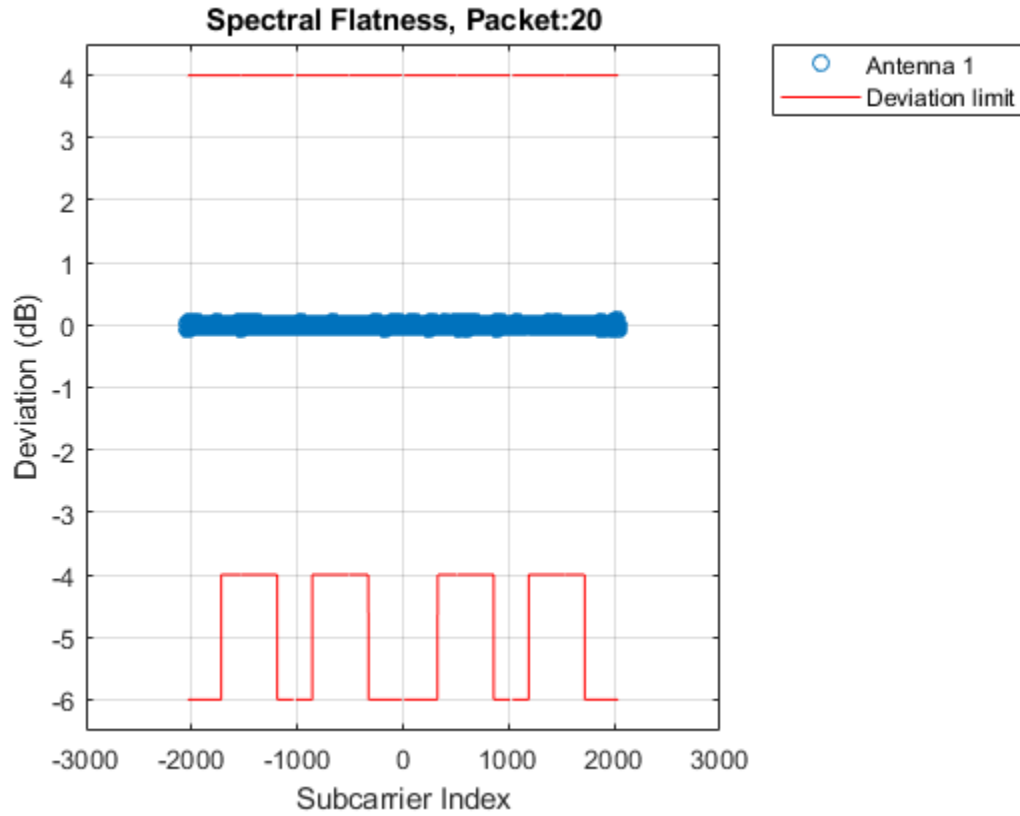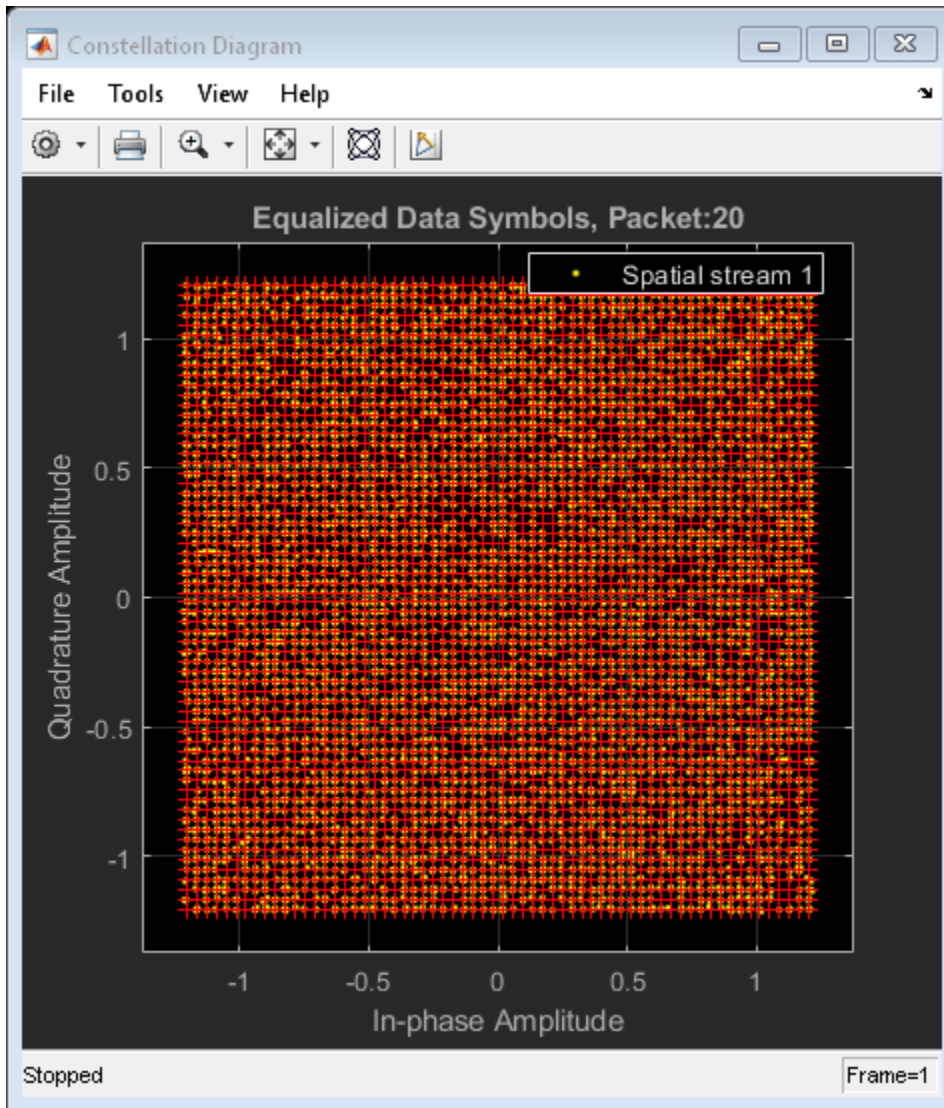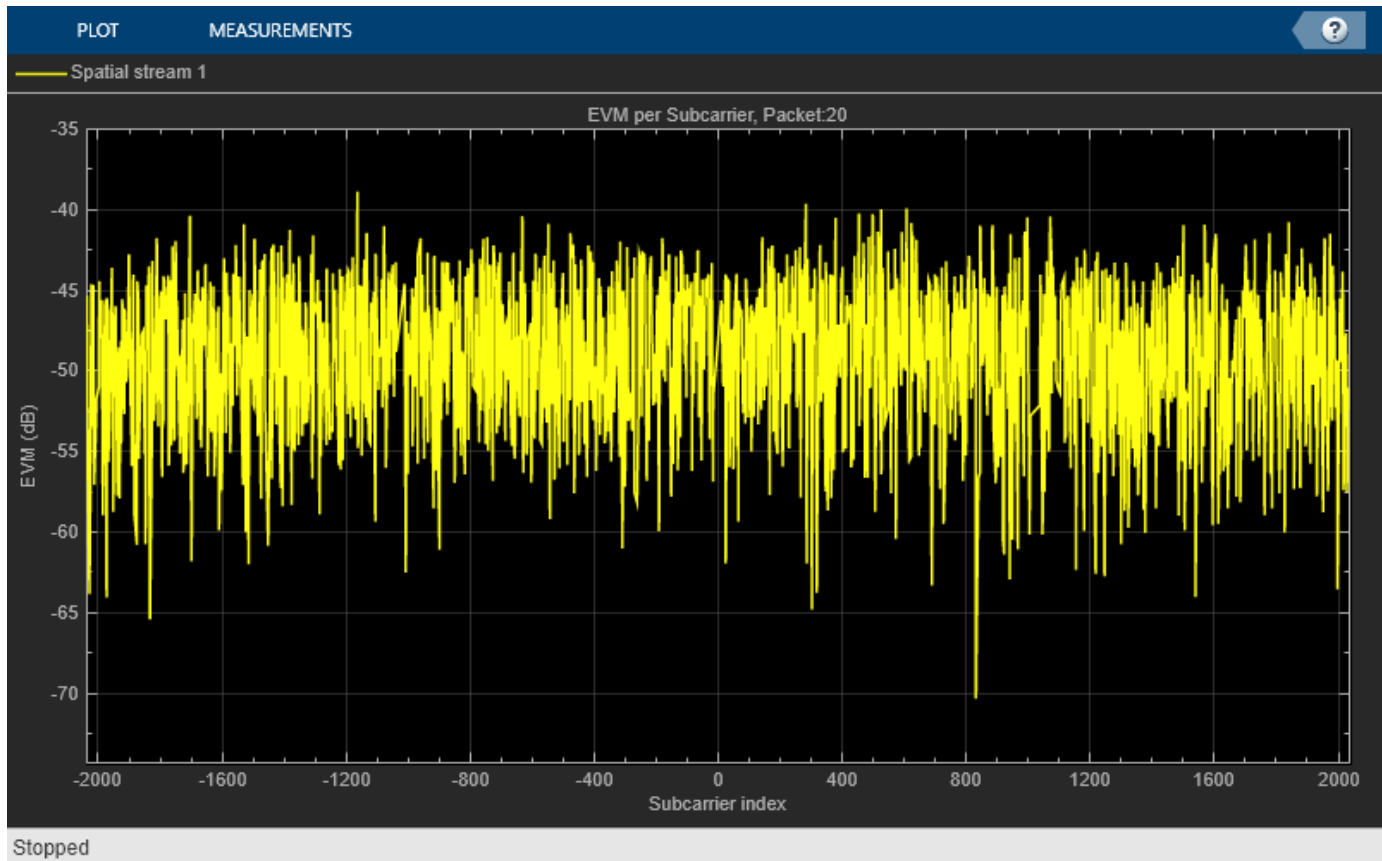
```
Spectral flatness passed

RMS EVM: 0.41%, -47.66dB
```

```
Decode success
```

```matlab
if pktNum>0
    fprintf('Average EVM for %d packets: %2.2f%%, %2.2fdB\n', ...
        pktNum,mean(rmsEVM(1:pktNum)),20*log10(mean(rmsEVM(1:pktNum))/100));
else
    disp('No complete packet detected');
end
```

```
Average EVM for 20 packets: 0.44%, -47.21dB
```

Spectral Mask Measurement

This section measures the spectral mask of the filtered and impaired waveform after high-power amplifier modeling. The transmitter spectral mask test [5 on page 3-0   ] uses a time-gated spectral measurement of the EHT Data field. The example extracts the EHT Data field of each packet from the oversampled waveform by using the start indices of each packet within the baseband waveform. Any delay introduced in the baseband processing chain used to determine the packet indices must be accounted for when gating the EHT data field within txWaveform. Concatenate the extracted EHT Data fields in preparation for measurement.

```matlab
startIdx = osf*(ind.EHTData(1)-1)+1; % Upsampled start of EHT Data
endIdx = osf*ind.EHTData(2);         % Upsampled end of EHT Data
delay = grpdelay(firinterp,1); % Group delay of downsampling filter
idx = zeros(endIdx-startIdx+1,pktNum);
for i = 1:pktNum
```

```matlab
    % Start of packet in txWaveform
    pktOffset = round(osf*pktOffsetStore(i))-delay;
    % Indices of EHT-Data in txWaveform
    idx(:,i) = (pktOffset+(startIdx:endIdx));
end
gatedEHTTData = txWaveform(idx(:),:);
```

The 802.11be standard specifies the spectral mask relative to the peak power spectral density. This generated plot overlays the required mask with the measured PSD.

```matlab
if pktNum>0
    ehtSpectralMaskTest(gatedEHTTData,fs,osf);
end
```

```
    Spectral mask passed
```
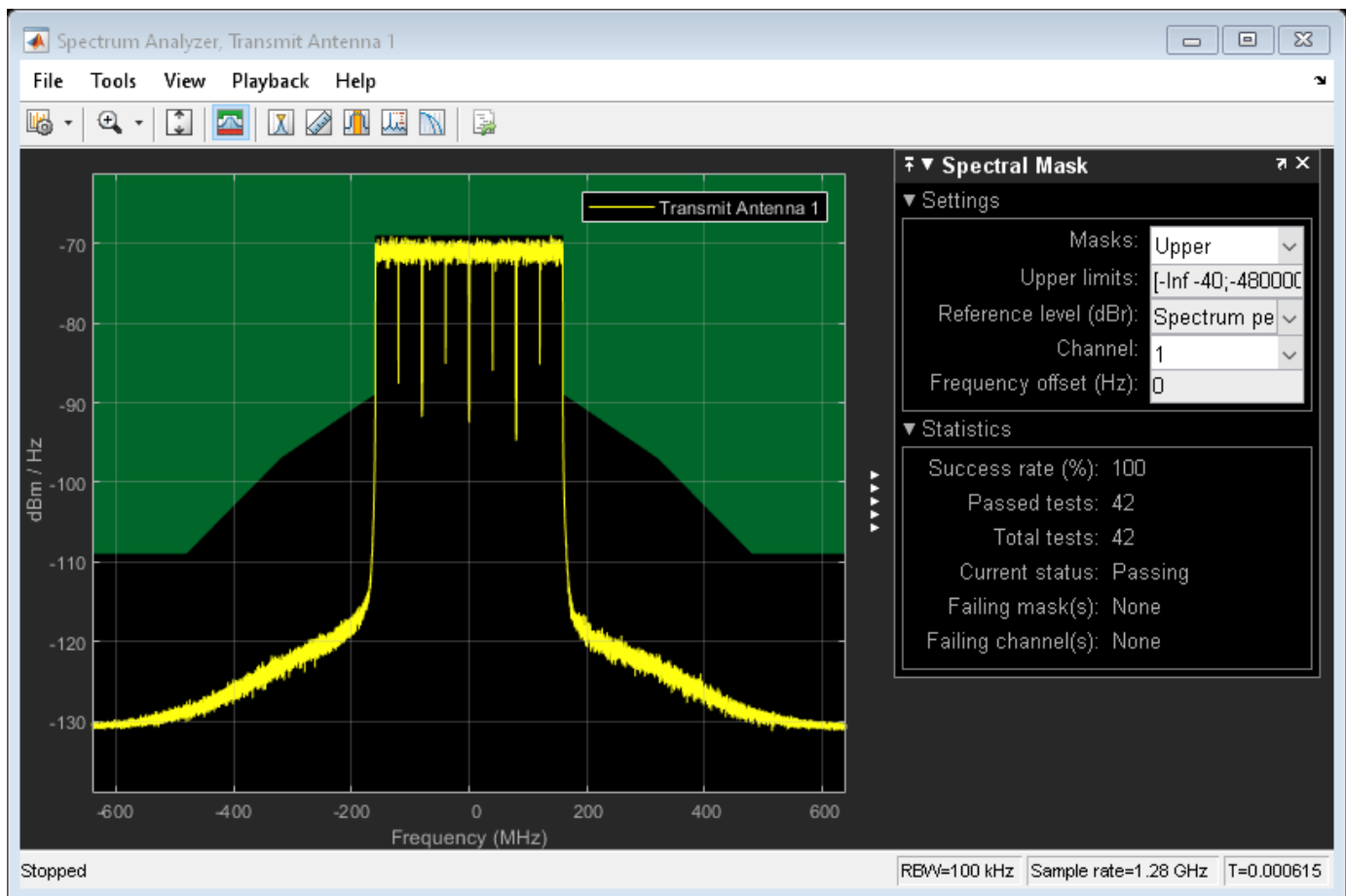

Conclusion and Further Exploration

This example measures and plots:

- Spectral flatness
- RMS EVM per subcarrier
- Equalized constellation

- Spectral mask

The high-power amplifier model introduces significant in-band distortion and spectral regrowth, which is visible in the EVM results, noisy constellation, and out-of-band emissions in the spectral mask plot. Try increasing the high-power amplifier backoff and observe the improved EVM, constellation, and lower out-of-band emissions. The downsampling (to bring the waveform to baseband for processing) stage includes filtering. The filter response affects the spectral flatness measurement. The ripple in the spectral flatness measurement is due to downsampling to baseband. Try using a different filter or changing the stop-band attenuation and observe the impact on the spectral flatness.

References

1 IEEE P802.11be™/D0.1 Draft 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 8: Enhancements for extremely high throughput (EHT).

2 IEEE P802.11ax™/D7.0 Draft 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 1: Enhancements for High Efficiency WLAN.

3 Loc and Cheong. IEEE P802.11 Wireless LANs. TGac Functional Requirements and Evaluation Methodology Rev. 16. 2011-01-19.

4 Perahia, E., and R. Stacey. Next Generation Wireless LANs: 802.11n and 802.11ac. 2nd Edition. United Kingdom: Cambridge University Press, 2013. Archambault, Jerry, and Shravan Surineni. "IEEE 802.11 spectral measurements using vector signal analyzers." RF Design 27.6 (2004): 38-49.

5 Archambault, Jerry, and Shravan Surineni. "IEEE 802.11 spectral measurements using vector signal analyzers." RF Design 27.6 (2004): 38-49.

802.11ad Waveform Generation with Beamforming

This example shows how to beamform an IEEE® 802.11ad™ DMG waveform with a phased array using WLAN Toolbox™ and Phased Array System Toolbox™.

Introduction

IEEE 802.11ad [1] defines the directional multi-gigabit (DMG) transmission format operating at 60 GHz. To overcome the large path loss experienced at 60 GHz, the IEEE 802.11ad standard is designed to support directional beamforming. By using phased antenna arrays you can apply an antenna weight vector (AWV) to focus the antenna pattern in the desired direction. Each packet is transmitted on all array elements, but the AWV applies a phase shift to each element to steer the transmission. The quality of a communication link can be improved by appending optional training fields to DMG packets, and testing different AWVs at the transmitter or receiver. This process is called beam refinement.

A DMG packet consists of the following fields:

1 STF - The short training field, which is used for synchronization.

2 CE - Channel estimation field, which is used for channel estimation.

3 Header - The signaling field, which the receiver decodes to determine transmission parameters.

4 Data - The data field, which carries the user data payload.

5 AGC Subfields - Optional automatic gain control (AGC) subfields, used for beam refinement.

6 Training Subfields - Optional training subfields, used for beam refinement.

The STF and CE fields form the preamble. The preamble, header, and data fields of a DMG packet are transmitted with the same AWV. For transmitter beam refinement training, up to 64 training (TRN) subfields can be appended to the packet. Each TRN subfield is transmitted using a different AWV. This allows the performance of up to 64 different AWVs to be measured, and the AWV for the preamble, header, and data fields to be refined for subsequent transmissions. CE subfields are periodically transmitted, one for every four TRN subfields, amongst the TRN subfields. Each CE subfield is transmitted using the same AWV as the preamble. To allow the receiver to reconfigure AGC before receiving the TRN subfields, the TRN subfields are preceded by AGC subfields. For each TRN subfield, an AGC subfield is transmitted using the same AWV applied to the individual TRN subfield. This allows a gain to be set at the receiver, suitable to measuring all TRN subfields. The diagram below shows the packet structure with four AGC and TRN subfields numbered and highlighted. Therefore, four AWVs are tested as part of beam refinement. The same AWVs are applied to AGC and TRN subfields with the same number.
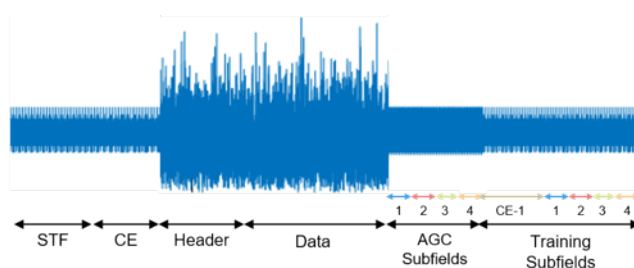
This example simulates transmitter training by applying different AWVs to each of the training subfields to steer the transmission in multiple directions. The strength of each training subfield is

evaluated at a receiver by evaluating the far-field plane wave to determine which transmission AWV is optimal. This simulation does not include a channel or path loss.

This example requires WLAN Toolbox and Phased Array System Toolbox.

Waveform Specification

The waveform is configured for a DMG packet transmission with the orthogonal frequency-division multiplexing (OFDM) physical layer, a 100-byte physical layer service data unit (PSDU), and four transmitter training subfields. The four training subfields allow four AWVs to be tested for beam refinement. Using the function `wlanDMGConfig`, create a DMG configuration object. A DMG configuration object specifies transmission parameters.

```matlab
dmg = wlanDMGConfig;
dmg.MCS = 13;              % OFDM
dmg.TrainingLength = 4;    % Use 4 training subfields
dmg.PacketType = 'TRN-T';  % Transmitter training
dmg.PSDULength = 100;      % Bytes
```

Beamforming Specification

The transmitter antenna pattern is configured as a 16-element uniform linear array with half-wavelength spacing. Using the objects `phased.ULA` (Phased Array System Toolbox) and `phased.SteeringVector` (Phased Array System Toolbox), create the phased array and the AWVs. The location of the receiver for evaluating the transmission is specified as an offset from the boresight of the transmitter.

```matlab
receiverAz = 6; % Degrees off the transmitter's boresight
```

A uniform linear phased array with 16 elements is created to steer the transmission.

```matlab
N = 16;                       % Number of elements
c = physconst('LightSpeed'); % Propagation speed (m/s)
fc = 60.48e9;                 % Center frequency (Hz)
lambda = c/fc;                % Wavelength (m)
d = lambda/2;                 % Antenna element spacing (m)
TxArray = phased.ULA('NumElements',N,'ElementSpacing',d);
```

The AWVs are created using a `phased.SteeringVector` (Phased Array System Toolbox) object. Five steering angles are specified to create five AWVs, one for the preamble and data fields, and one for each of the four the training subfields. The preamble and data fields are transmitted at boresight. The four training subfields are transmitted at angles around boresight.

```matlab
% Create a directional steering vector object
SteeringVector = phased.SteeringVector('SensorArray',TxArray);

% The directional angle for the preamble and data is 0 degrees azimuth, no
% elevation, therefore at boresight. [Azimuth; Elevation]
preambleDataAngle = [0; 0];

% Each of the four training fields uses a different set of weights to steer
% to a slightly different direction. [Azimuth; Elevation]
trnAngle = [[-10; 0] [-5; 0] [5; 0] [10; 0]];

% Generate the weights for all of the angles
weights = SteeringVector(fc,[preambleDataAngle trnAngle]);
```

```
% Each row of the AWV is a weight to apply to a different antenna element
preambleDataAWV = conj(weights(:,1)); % AWV used for preamble, data and CE fields
trnAWV = conj(weights(:,2:end));      % AWV used for each TRN subfield
```

Using the plotArrayResponse helper function, the array response shows the direction of the receiver is most aligned with the direction of training subfield TRN-SF3.

```
plotArrayResponse(TxArray,receiverAz,fc,weights);
```
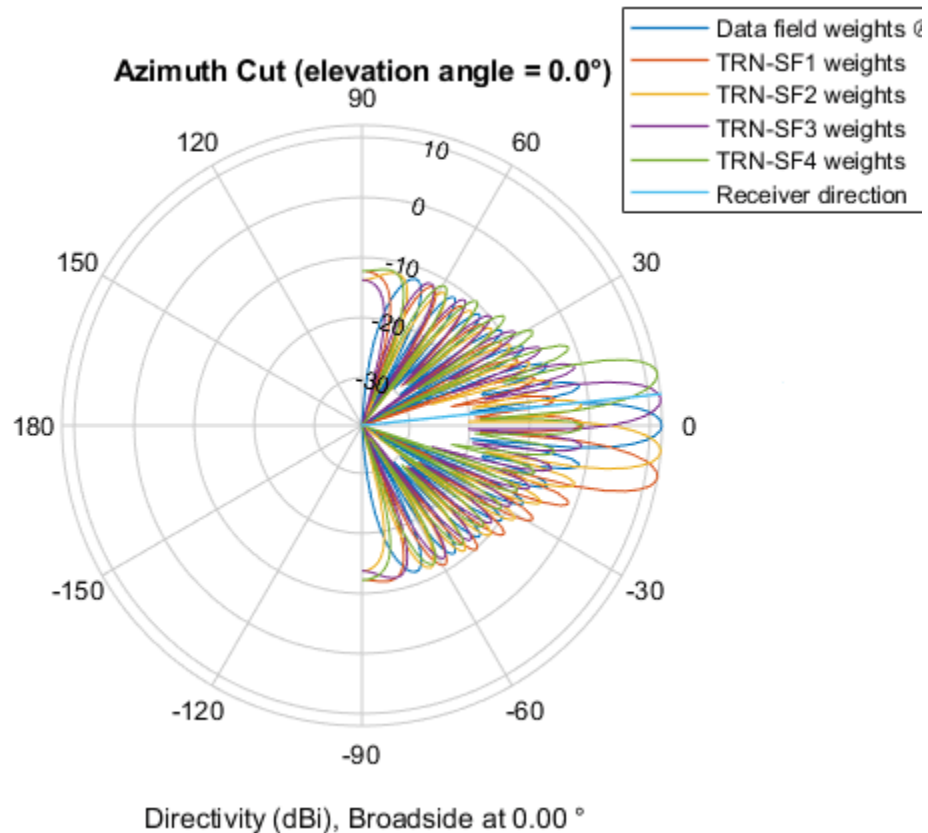

Generate Baseband Waveform

Use the configured DMG object and a PSDU filled with random data as inputs to the waveform generator, `wlanWaveformGenerator`. The waveform generator modulates PSDU bits according to a format configuration and also performs OFDM windowing.

```
% Create a PSDU of random bits
s = rng(0); % Set random seed for repeatable results
psdu = randi([0 1],dmg.PSDULength*8,1);

% Generate packet
tx = wlanWaveformGenerator(psdu,dmg);
```

Apply Weight Vectors to Each Field

A `phased.Radiator` (Phased Array System Toolbox) object is created to apply the AWVs to the waveform, combine the radiated signal from each element to form a plane wave, and determine the plane wave at the angle of interest, `receiverAz`. Each portion of the DMG waveform `tx` is passed through the Radiator with a specified set of AWVs, and the angle at which to evaluate the plane wave.

```
Radiator = phased.Radiator;
Radiator.Sensor = TxArray;          % Use the uniform linear array
Radiator.WeightsInputPort = true;   % Provide AWV as argument
Radiator.OperatingFrequency = fc;   % Frequency in Hertz
Radiator.CombineRadiatedSignals = true; % Create plane wave

% The plane wave is evaluated at a direction relative to the radiator
steerAngle = [receiverAz; 0]; % [Azimuth; Elevation]

% The beamformed waveform is evaluated as a plane wave at the receiver
planeWave = zeros(size(tx));

% Get indices for fields
ind = wlanFieldIndices(dmg);

% Get the plane wave while applying the AWV to the preamble, header, and data
idx = (1:ind.DMGData(2));
planeWave(idx) = Radiator(tx(idx),steerAngle,preambleDataAWV);

% Get the plane wave while applying the AWV to the AGC and TRN subfields
for i = 1:dmg.TrainingLength
    % AGC subfields
    agcsfIdx = ind.DMGAGCSubfields(i,1):ind.DMGAGCSubfields(i,2);
    planeWave(agcsfIdx) = Radiator(tx(agcsfIdx),steerAngle,trnAWV(:,i));
    % TRN subfields
    trnsfIdx = ind.DMGTRNSubfields(i,1):ind.DMGTRNSubfields(i,2);
    planeWave(trnsfIdx) = Radiator(tx(trnsfIdx),steerAngle,trnAWV(:,i));
end

% Get the plane wave while applying the AWV to the TRN-CE
for i = 1:dmg.TrainingLength/4
    trnceIdx = ind.DMGTRNCE(i,1):ind.DMGTRNCE(i,2);
    planeWave(trnceIdx) = Radiator(tx(trnceIdx),steerAngle,preambleDataAWV);
end
```

Evaluate the Beamformed Waveform

The helper function plotDMGWaveform plots the magnitude of the beamformed plane wave. When evaluating the magnitude of the beamformed plane wave we can see that the fields beamformed in the direction of the receiver are stronger than other fields.

```
plotDMGWaveform(planeWave,dmg,'Beamformed Plane Wave with Fields Highlighted');

rng(s); % Restore random state
```

Beamformed Plane Wave with Fields Highlighted
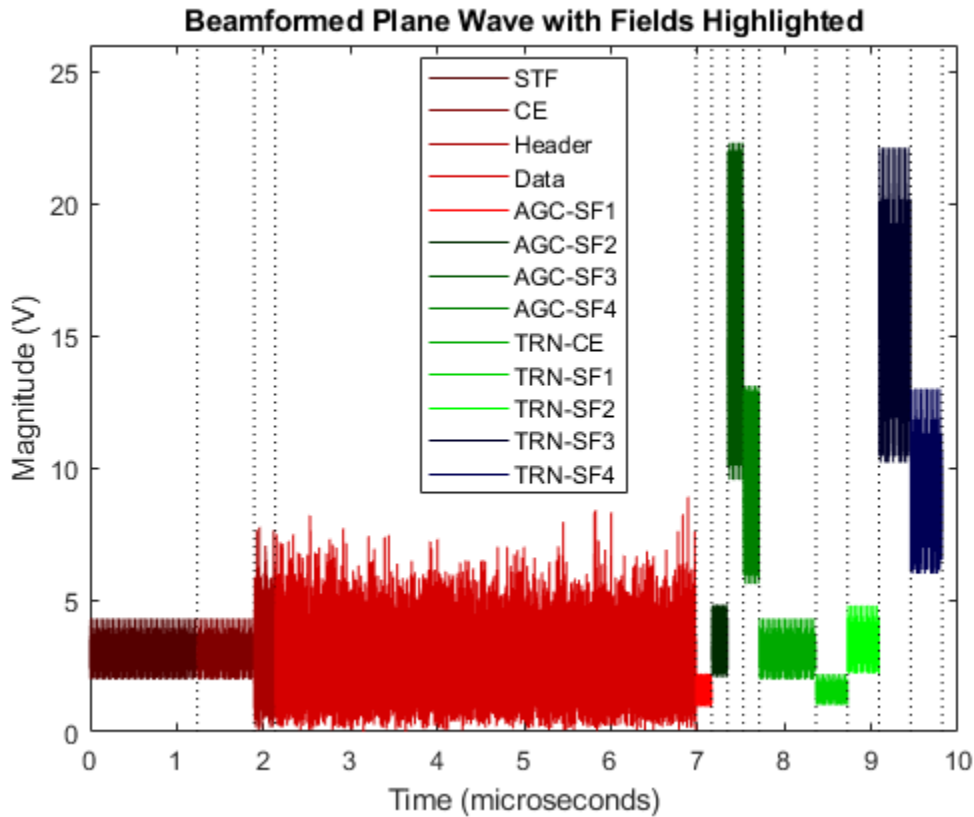
Conclusion

This example showed how to generate an IEEE 802.11ad DMG waveform and apply AWVs to different portions of the waveform. WLAN Toolbox was used to generate a standard compliant waveform, and Phased Array System Toolbox was used to apply the AWVs and evaluate the magnitude of the resultant plane wave in the direction of a receiver.

Appendix

This example uses the following helper functions:

- plotArrayResponse.m
- plotDMGWaveform.m

Selected Bibliography

1	IEEE Std 802.11ad™-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. Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band.

802.11ac Transmit Beamforming

This example shows how to improve the performance of an IEEE® 802.11ac™ link by beamforming the transmission when channel state information is available at the transmitter.

Introduction

Transmit beamforming focuses energy towards a receiver to improve the SNR of a link. In this scheme the transmitter is called a beamformer and the receiver is called a beamformee. A steering matrix is used by the beamformer to direct the energy to the beamformee. The steering matrix is calculated using channel state information obtained through channel measurements. In IEEE 802.11ac [1] these measurements are obtained by sounding the channel between beamformer and beamformee. To sound the channel the beamformer sends an NDP (Null Data Packet) to the beamformee. The beamformee uses the channel information provided by sounding to calculate a feedback matrix. This matrix is fed back to the beamformer in a compressed format. The beamformer can then use the feedback matrix to create a steerin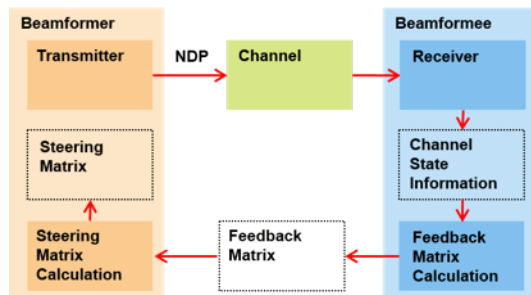g matrix and beamform transmissions to the beamformee. The process of forming the steering matrix is shown in this diagram.

In IEEE 802.11ac the single user beamformee capability is not mandatory. Therefore a multi-antenna transmitter may have to use a different scheme to transmit packets to a receiver which cannot act as a beamformee. One such scheme is spatial expansion. Spatial expansion allows a number of space-time streams to be transmitted on a greater number of transmit antennas. Using spatial expansion can provide a small transmit diversity gain in channels with flat fading when compared to directly mapping space-time streams to transmit antennas [2].

In this example a 4x2 MIMO configuration is considered between a transmitter and receiver, with two space-time streams used for a data packet transmission. First the scenario of a receiver which is not capable of being a beamformee is considered. A transmission is made using spatial expansion and the data symbols are recovered and the signal quality measured. To show the benefits of transmit beamforming the data packet is then transmitted over the same channel realization, but this time using transmit beamforming. The performance of the two schemes are then compared. These stages are shown in the diagram below.
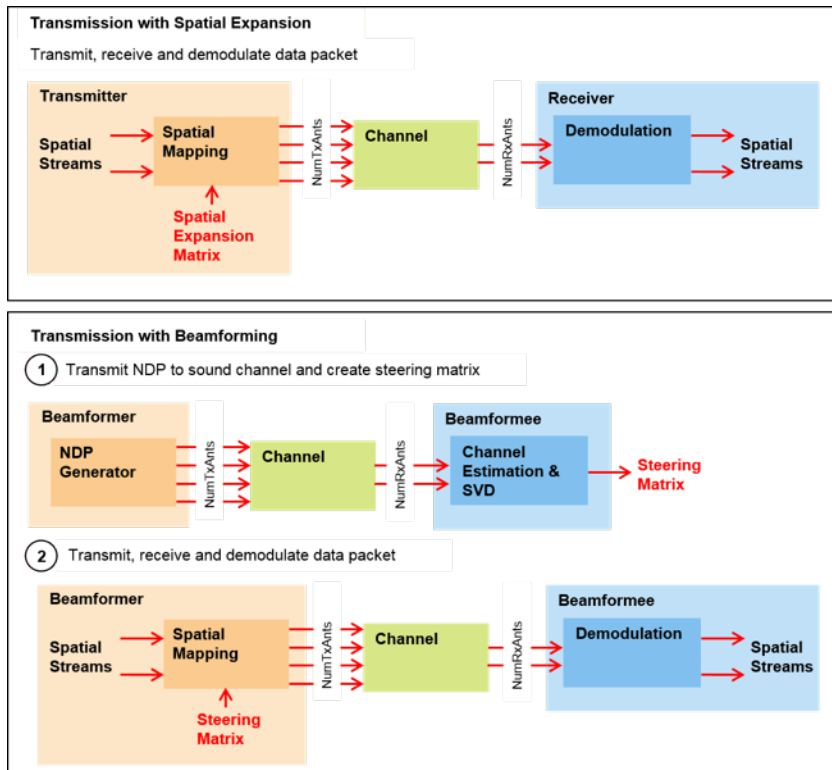
Waveform Configuration

A 4x2 MIMO configuration is used in this example with 2 space-time streams.

```
NumTxAnts = 4;   % Number of transmit antennas
NumSTS = 2;      % Number of space-time streams
NumRxAnts = 2;   % Number of receive antennas
```

The format specific configuration of a VHT waveform is described using a VHT format configuration object. In this example the waveform is configured with a 20 MHz bandwidth and the MIMO configuration specified above.

```
cfgVHT = wlanVHTConfig;
cfgVHT.ChannelBandwidth = 'CBW20';
cfgVHT.APEPLength = 4000;
cfgVHT.NumTransmitAntennas = NumTxAnts;
cfgVHT.NumSpaceTimeStreams = NumSTS;
cfgVHT.MCS = 4; % 16-QAM, rate 3/4
```

Channel Configuration

In this example a TGac channel model is used with delay profile Model-B. The channel realization is controlled with a seed to allow repeatability.

```
tgacChannel = wlanTGacChannel;
tgacChannel.DelayProfile = 'Model-B';
tgacChannel.ChannelBandwidth = cfgVHT.ChannelBandwidth;
tgacChannel.SampleRate = wlanSampleRate(cfgVHT);
tgacChannel.NumReceiveAntennas = NumRxAnts;
tgacChannel.NumTransmitAntennas = NumTxAnts;
```

```
tgacChannel.TransmitReceiveDistance = 100; % Meters
tgacChannel.RandomStream = 'mt19937ar with seed';
tgacChannel.Seed = 70; % Seed to allow repeatability
```

Noise is added to the time domain waveform at the output of the channel with a power, `noisePower`.

```
noisePower = -37; % dBW
```

Setup other objects and variables for simulation.

```
% Indices for extracting fields
ind = wlanFieldIndices(cfgVHT);

% AWGN channel to add noise with a specified noise power. The random
% process controlling noise generation is seeded to allow repeatability.
awgnChannel = comm.AWGNChannel;
awgnChannel.RandomStream = 'mt19937ar with seed';
awgnChannel.Seed = 5;
awgnChannel.NoiseMethod = 'Variance';
awgnChannel.Variance = 10^(noisePower/10);

% Calculate the expected noise variance after OFDM demodulation
noiseVar = vhtBeamformingNoiseVariance(noisePower,cfgVHT);

% Number of spatial streams
Nss = NumSTS/(cfgVHT.STBC+1);

% Get the number of occupied subcarriers in VHT fields
ofdmInfo = wlanVHTOFDMInfo('VHT-Data',cfgVHT);
Nst = ofdmInfo.NumTones;

% Generate a random PSDU which will be transmitted
rng(0); % Set random state for repeatability
psdu = randi([0 1],cfgVHT.PSDULength*8,1);
```

Transmission with Spatial Expansion

First a transmission is made using spatial expansion. This type of transmission may be made by a multi-antenna transmitter to a receiver which is not capable of being a beamformee. The `SpatialMapping` property of the format configuration object allows different spatial mapping schemes to be selected. In this example the example spatial expansion matrix provided in Section 2.3.11.1.1.2 of [3] is used. Therefore a `'Custom'` spatial mapping is configured. The custom spatial mapping matrix is used by assigning the `SpatialMappingMatrix` of the format configuration object. This matrix describes the mapping of each subcarrier for each space-time stream to all transmit antennas. Therefore the size of the spatial mapping matrix used is `Nst-by-Nsts-by-Nt`. `Nst` is the number of occupied subcarriers, `Nsts` is the number of space-time streams, and `Nt` is the number of transmit antennas. The spatial mapping matrix duplicates some of the space-time streams to form the desired number of transmit streams.

```
% Configure a spatial expansion transmission
vhtSE = cfgVHT;
vhtSE.SpatialMapping = 'Custom'; % Use custom spatial expansion matrix
vhtSE.SpatialMappingMatrix = helperSpatialExpansionMatrix(vhtSE);

% Generate waveform
tx = wlanWaveformGenerator(psdu,vhtSE);
```

```
% Pass waveform through a fading channel and add noise. Trailing zeros
% are added to allow for channel filter delay.
rx = tgacChannel([tx; zeros(15,NumTxAnts)]);
% Allow same channel realization to be used subsequently
reset(tgacChannel);
rx = awgnChannel(rx);
% Allow same noise realization to be used subsequently
reset(awgnChannel);

% Estimate symbol timing
tOff = wlanSymbolTimingEstimate(rx(ind.LSTF(1):ind.LSIG(2),:),vhtSE.ChannelBandwidth);

% Channel estimation
vhtltf = rx(tOff+(ind.VHTLTF(1):ind.VHTLTF(2)),:);
vhtltfDemod = wlanVHTLTFDemodulate(vhtltf,vhtSE);
chanEstSE = wlanVHTLTFChannelEstimate(vhtltfDemod,vhtSE);
```

The received data field is demodulated and equalized to recover OFDM symbols for each spatial stream.

```
% Demodulate and equalize the data
vhtdata = rx(tOff+(ind.VHTData(1):ind.VHTData(2)),:);
[~,~,symSE] = wlanVHTDataRecover(vhtdata,chanEstSE,noiseVar,vhtSE,...
    'PilotPhaseTracking','None');
```

The constellation of each spatial stream is plotted below.

```
refSym = wlanReferenceSymbols(cfgVHT); % Reference constellation
seConst = vhtBeamformingPlotConstellation(symSE,refSym, ...
    'Spatial Expansion Transmission Equalized Symbols');
```
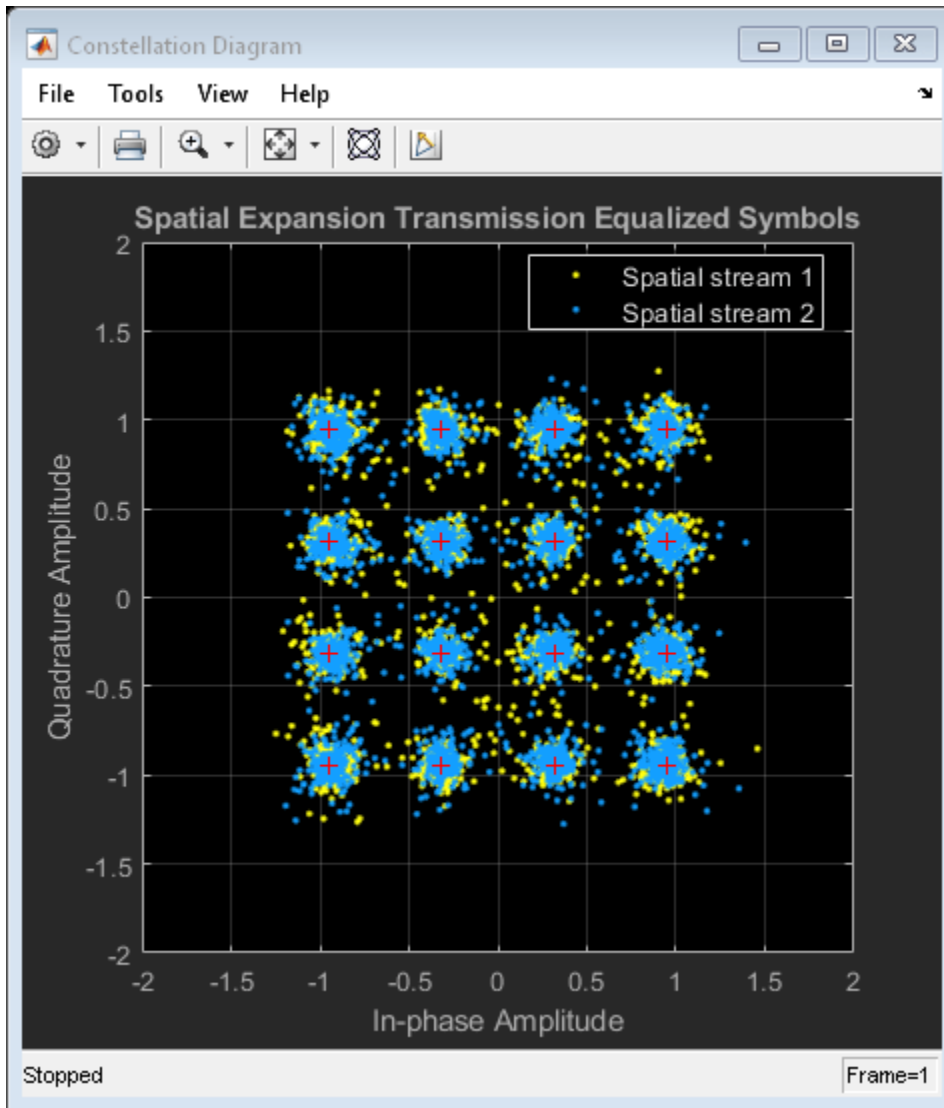
The variance in the constellation is approximately the same for each spatial stream as the SNRs are approximately the same. This is because the average power in the channel is on average approximately the same per space-time stream:

```
disp('Mean received channel power per space-time stream with spatial expansion: ')
for i = 1:NumSTS
    fprintf('  Space-time stream %d: %2.2f W\n',i, ...
        sum(mean(chanEstSE(:,i,:).*conj(chanEstSE(:,i,:)),1),3))
end
```

```
Mean received channel power per space-time stream with spatial expansion:
  Space-time stream 1: 0.73 W
  Space-time stream 2: 0.50 W
```

Transmission with Beamforming

When the receiver is capable of being a beamformee, a beamformed transmission can create a higher SNR compared to spatial expansion. We will now show the advantage of having channel state

information available to create and use a steering matrix. To calculate a beamforming steering matrix, an NDP is passed through the channel. `'Direct'` spatial mapping is used for the NDP transmission and the number of space-time streams is configured to match the number of transmit antennas. This allows the VHT-LTF to be used to sound channels between each of the transmit antennas and receive antennas. The calculated beamforming matrix is then used to beamform a transmission through the channel. The same channel realization is used for sounding and data transmission and there is no feedback compression between beamformee and beamformer, therefore the beamforming can be regarded as perfect in this example.

```
% Configure a sounding packet
vhtSound = cfgVHT;
vhtSound.APEPLength = 0; % NDP so no data
vhtSound.NumSpaceTimeStreams = NumTxAnts;
vhtSound.SpatialMapping = 'Direct'; % Each TxAnt carries a STS

% Generate sounding waveform
soundingPSDU = [];
tx = wlanWaveformGenerator(soundingPSDU,vhtSound);

% Pass sounding waveform through the channel and add noise. Trailing zeros
% are added to allow for channel filter delay.
rx = tgacChannel([tx; zeros(15,NumTxAnts)]);
% Allow same channel realization to be used subsequently
reset(tgacChannel);
rx = awgnChannel(rx);
% Allow same noise realization to be used subsequently
reset(awgnChannel);

% Estimate symbol timing
tOff = wlanSymbolTimingEstimate(rx(ind.LSTF(1):ind.LSIG(2),:),vhtSound.ChannelBandwidth);
```

Channel estimation is performed using the sounding packet to estimate the actual channel response between each transmit and receive antenna.

```
% Channel estimation
vhtLLTFInd = wlanFieldIndices(vhtSound,'VHT-LTF');
vhtltf = rx(tOff+(vhtLLTFInd(1):vhtLLTFInd(2)),:);
vhtltfDemod = wlanVHTLTFDemodulate(vhtltf,vhtSound);
chanEstSound = wlanVHTLTFChannelEstimate(vhtltfDemod,vhtSound);
```

The channel estimated using `wlanVHTLTFChannelEstimate` includes cyclic shifts applied at the transmitter to each space-time stream. To calculate a beamforming steering matrix the cyclic shifts applied at the transmitter are removed from the channel estimate.

```
% Remove impact of cyclic shift from channel estimate
chanEstSound = vhtBeamformingRemoveCSD(chanEstSound, ...
    vhtSound.ChannelBandwidth,vhtSound.NumSpaceTimeStreams);
```

In this example the beamforming steering matrix is calculated using singular value decomposition (SVD) [3]. The SVD of the channel matrix results in two unitary matrices, U and V, and a diagonal matrix of singular values S. The first NumSTS columns of V per subcarrier are used as the beamforming steering matrix. The SVD is computed using the function `svd`.

```
chanEstPerm = permute(chanEstSound,[3 2 1]); % permute to Nr-by-Nt-by-Nst
V = zeros(Nst,NumTxAnts,NumRxAnts);
for i = 1:Nst
    [U,S,V(i,:,:)] = svd(chanEstPerm(:,:,i),'econ');
```

```
end
steeringMatrix = V(:,:,1:NumSTS); % Nst-by-Nt-by-Nsts
```

The beamforming steering matrix calculated above is applied as a custom spatial mapping matrix and is used to send data through the same channel.

```
% Configure a transmission with beamforming
vhtBF = cfgVHT;
vhtBF.SpatialMapping = 'Custom';
% Permute steering matrix to Nst-by-Nsts-by-Nt
vhtBF.SpatialMappingMatrix = permute(steeringMatrix,[1 3 2]);

% Generate beamformed data transmission
tx = wlanWaveformGenerator(psdu,vhtBF);

% Pass through the channel and add noise. Trailing zeros
% are added to allow for channel filter delay.
rx = tgacChannel([tx; zeros(15,NumTxAnts)]);
rx = awgnChannel(rx);

% Estimate symbol timing
tOff = wlanSymbolTimingEstimate(rx(ind.LSTF(1):ind.LSIG(2),:),vhtBF.ChannelBandwidth);

% Channel estimation
vhtltf = rx(tOff+(ind.VHTLTF(1):ind.VHTLTF(2)),:);
vhtltfDemod = wlanVHTLTFDemodulate(vhtltf,vhtBF);
chanEstBF = wlanVHTLTFChannelEstimate(vhtltfDemod,vhtBF);
```

The received data field is demodulated and equalized to recover OFDM symbols for each spatial stream.

```
% Demodulate and equalize the data
vhtdata = rx(tOff+(ind.VHTData(1):ind.VHTData(2)),:);
[~,~,symBF] = wlanVHTDataRecover(vhtdata,chanEstBF,noiseVar,vhtBF,...
    'PilotPhaseTracking','None');
```

The equalized constellation for each spatial stream is plotted below. Note that the higher order spatial stream has a larger variance. This is due to the ordered singular values of the channels used in SVD beamforming.

```
bfConst = vhtBeamformingPlotConstellation(symBF,refSym, ...
    'Beamformed Transmission Equalized Symbols');
```
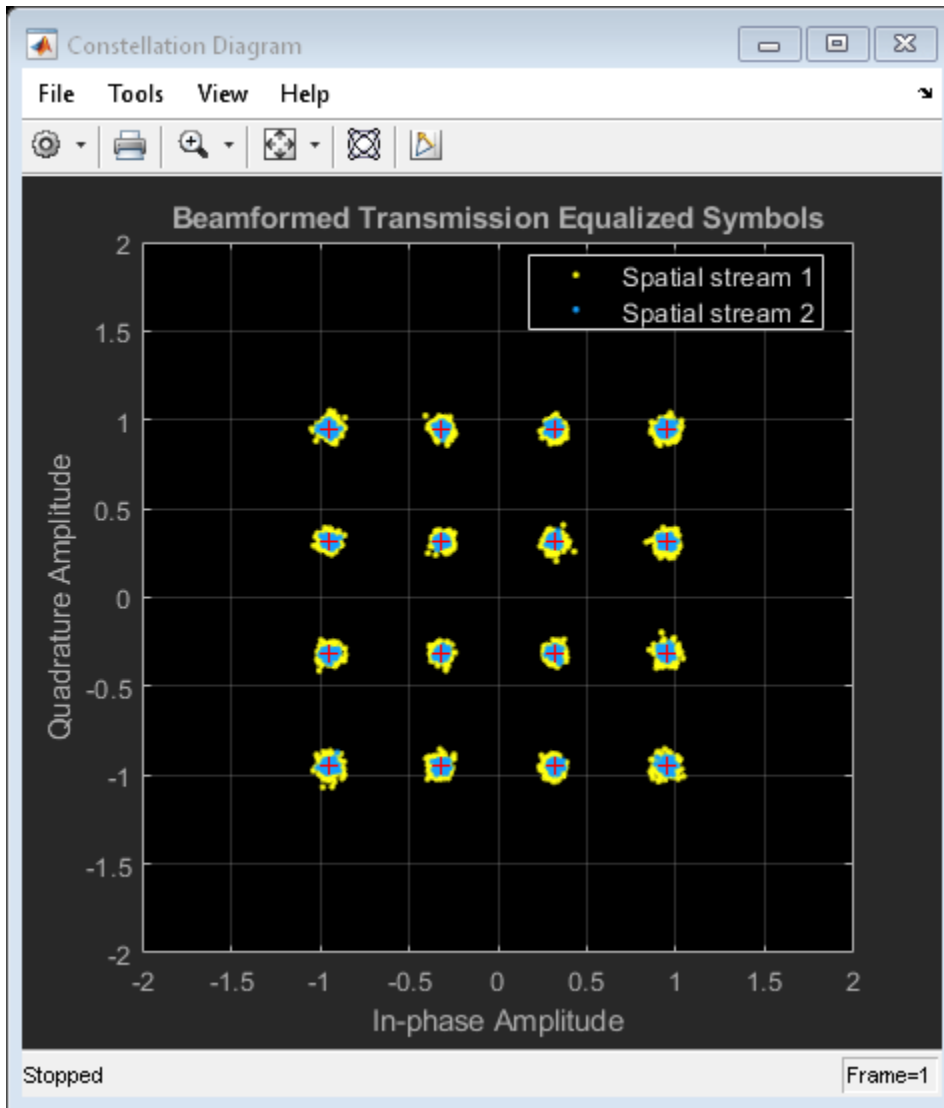
This ordering is also visible in the average power of the received space-time streams. The power of the received first space-time stream is larger than the second space-time stream. This is because the received signal strength is a function of the singular values of the channel which SVD orders in a decreasing fashion.

```
disp('Mean received channel power per space-time stream with SVD transmit beamforming: ')
for i = 1:NumSTS
    fprintf('  Space-time stream %d: %2.2f W\n',i, ...
        sum(mean(chanEstBF(:,i,:).*conj(chanEstBF(:,i,:)),1),3))
end
```

```
Mean received channel power per space-time stream with SVD transmit beamforming:
  Space-time stream 1: 2.08 W
  Space-time stream 2: 0.45 W
```

Comparison and Conclusion

The figure below plots the equalized constellation from the spatial expansion and beamformed transmissions for all spatial streams. Note the improved constellation using SVD-based transmit beamforming.

```
str = sprintf('%dx%d',NumTxAnts,NumRxAnts);
compConst = vhtBeamformingPlotConstellation([symSE(:) symBF(:)],refSym, ...
    'Beamformed Transmission Equalized Symbols', ...
    {[str ' Spatial Expansion'],[str ' Transmit Beamforming']});
```
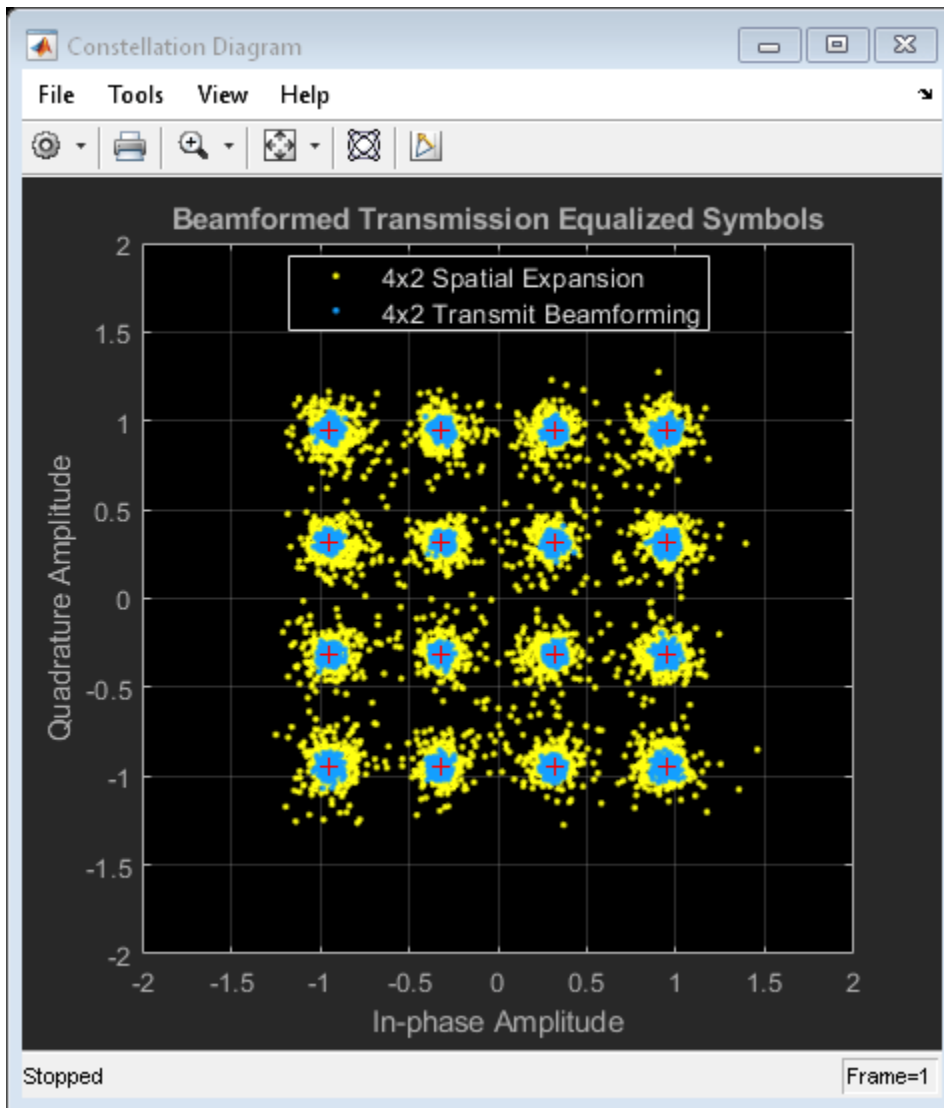

The improvement can also be measured through the RMS and maximum error vector magnitude (EVM). EVM is a measure of demodulated signal quality.

```
EVM = comm.EVM;
EVM.AveragingDimensions = [1 2]; % Average over all subcarriers and symbols
EVM.MaximumEVMOutputPort = true;
EVM.ReferenceSignalSource  = 'Estimated from reference constellation';
```

```
EVM.ReferenceConstellation = refSym;

[rmsEVMSE,maxEVMSE] = EVM(symSE); % EVM using spatial expansion
[rmsEVMBF,maxEVMBF] = EVM(symBF); % EVM using beamforming

for i = 1:Nss
    fprintf(['Spatial stream %d EVM:\n' ...
        '  Spatial expansion:    %2.1f%% RMS, %2.1f%% max\n' ...
        '  Transmit beamforming: %2.1f%% RMS, %2.1f%% max\n'], ...
        i,rmsEVMSE(i),maxEVMSE(i),rmsEVMBF(i),maxEVMBF(i));
end

Spatial stream 1 EVM:
  Spatial expansion:    9.2% RMS, 44.8% max
  Transmit beamforming: 2.0% RMS, 8.6% max
Spatial stream 2 EVM:
  Spatial expansion:    9.2% RMS, 52.3% max
  Transmit beamforming: 4.1% RMS, 12.7% max
```

This example demonstrates that if a receiver is capable of being a beamformee, the SNR can potentially be improved when a transmission is beamformed compared to a spatial expansion transmission. The increase in received power when using beamforming can lead to more reliable demodulation or potentially even a higher order modulation and coding scheme to be used for the transmission.

In a realistic operational simulation the performance of beamforming would be degraded due to the delay between channel state information calculation and feedback by the beamformee and feedback quantization. For more information see [2].

Appendix

This example uses these helper functions.

- helperSpatialExpansionMatrix.m
- vhtBeamformingNoiseVariance.m
- vhtBeamformingPlotConstellation.m
- vhtBeamformingRemoveCSD.m

Selected Bibliography

1 IEEE Std 802.11ac™-2013 IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz.

2 Perahia, Eldad, and Robert Stacey. Next Generation Wireless LANS: 802.11n and 802.11ac. Cambridge University Press, 2013.

3 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.

802.11ah Waveform Generation

This example shows how to generate IEEE® 802.11ah™ S1G waveforms and highlights some of the key features of the standard.

Introduction

802.11ah is intended for extended range and low power applications in the unlicensed sub 1 GHz band, including machine to machine communication and the internet of things. 802.11ah uses narrower contiguous channel bandwidths than 802.11n™ and 802.11ac™ to facilitate long range, low power communication at a lower data rate. Valid channel bandwidths are 1, 2, 4, 8, and 16 MHz.

Since 802.11ah uses the same underlying physical layer technologies as 802.11n and 802.11ac, the processing chains are very similar. With the exception of 1 MHz transmissions, in general data is modulated using the same process as in 802.11ac with a 1/10 clock rate.

In this example a number of 802.11ah S1G [1] waveforms are generated to highlight some of the key modes and features of the 802.11ah standard.

802.11ah Modes and PHY Features

The 802.11ah standard defines three modes:

- The 1 MHz mode (S1G_1M) is intended for low data rate applications. This mode features an extended preamble and a new modulation and coding scheme, MCS10, to improve robustness. MCS10 is BPSK 1/2 rate with 2 times repetition. When MCS10 is used the short training field (STF) is boosted by 3 dB to allow for packet detection [2]. In this mode the whole PPDU is beamformed.
- The >= 2 MHz long preamble mode (S1G_LONG) is used for single or multi-user transmissions with a 2, 4, 8, or 16 MHz channel bandwidth. The PPDU is similar to a 802.11ac VHT PPDU, consisting of an omni-directional portion and beam-changeable portion.
- The >= 2 MHz short preamble mode (S1G_SHORT) is used for single-user transmissions with a 2, 4, 8, or 16 MHz channel bandwidth. In this mode the whole PPDU is beamformed.

The 802.11ah standard is intended to operate outdoors as well as indoors. Traveling Pilots have been introduced to compensate for Doppler spread caused by reflections due to vehicular motion. In previous 802.11 standards the pilot locations are fixed to the same subcarrier for the duration of a packet. Tracking the varying channel conditions due to a high Doppler environment is not effective with fixed pilot locations. Traveling pilots change the subcarriers that carry the pilots over time which improves the ability to track changing channel conditions. In this example a waveform is generated for each of the three modes introduced above with configurations for MCS10 and traveling pilot highlighted.

S1G 1MHz Mode

An S1G 1 MHz PPDU consists of five fields, all of which can be beamformed:

1 STF - The short training field, which is used for coarse synchronization
2 LTF1 - The first long training field, which is used for fine synchronization and initial channel estimation
3 SIG - The signaling field, which the receiver decodes to determine transmission parameters
4 LTF2-N - The subsequent long training fields, which is used for MIMO channel estimation

5 Data - The data field, which carries the user data payload

Examples of waveform generation for MCS0 and MCS10 1 MHz transmissions are shown. When MCS10 is used a 3 dB power boost is applied to the short training field. This power boost will be visualized.

The function `wlanWaveformGenerator` returns an S1G configuration object. Create an S1G configuration object for 1 MHz bandwidth, 1 transmit antenna, 1 space-time stream, BPSK rate 1/2 (MCS0), and a 256 byte APEP length.

```
cfg1MHz = wlanS1GConfig;
cfg1MHz.ChannelBandwidth = 'CBW1';
cfg1MHz.NumTransmitAntennas = 1;
cfg1MHz.NumSpaceTimeStreams = 1;
cfg1MHz.MCS = 0;
cfg1MHz.APEPLength = 256;
```

Create a PSDU of random bits using the required length for the specified format configuration.

```
psdu = randi([0 1],cfg1MHz.PSDULength*8,1);
```

Generate an S1G waveform using the configured S1G format object and PSDU as inputs to the waveform generator, `wlanWaveformGenerator`. The waveform generator modulates PSDU bits according to a format configuration. The waveform generator also performs OFDM windowing. In this example windowing is disabled for clearer visualization.

```
% Generate waveform with windowing disabled
txMCS0 = wlanWaveformGenerator(psdu,cfg1MHz,'WindowTransitionTime',0);
```

Change the MCS of `cfg1MHz` to 10 and generate a second waveform to demonstrate the STF power boost.

```
cfg1MHz.MCS = 10;
txMCS10 = wlanWaveformGenerator(psdu,cfg1MHz,'WindowTransitionTime',0);
```

The power is plotted for the first 320 microseconds of both waveforms to capture the duration of the STF and first LTF in the 1 MHz transmission. Note the power boost of the STF when MCS10 is used. The power boost is required to obtain sufficient packet detection sensitivity to support MCS10 [2].

```
t = 320; % Duration to plot in microseconds
sr = wlanSampleRate(cfg1MHz); % Sample rate Hz
tick = (1/sr)*1e6; % Microseconds per sample
hf = figure;
hp(1) = plot(0:tick:t-tick,20*log10(abs(txMCS10(1:t*sr*1e-6,:))),'bx-');
hold on;
hp(2) = plot(0:tick:t-tick,20*log10(abs(txMCS0(1:t*sr*1e-6,:))),'ro-');
xlim([0 t-1]);
ylim([-20 15]);
s1gWavGenPlotFieldOverlay(cfg1MHz,hf);
grid on;
legend(hp,'1 MHz MCS10','1 MHz MCS0','Location','SouthWest');
title('Power of 1 MHz PPDU');
xlabel('Time (us)');
ylabel('Power (dBW)');
```
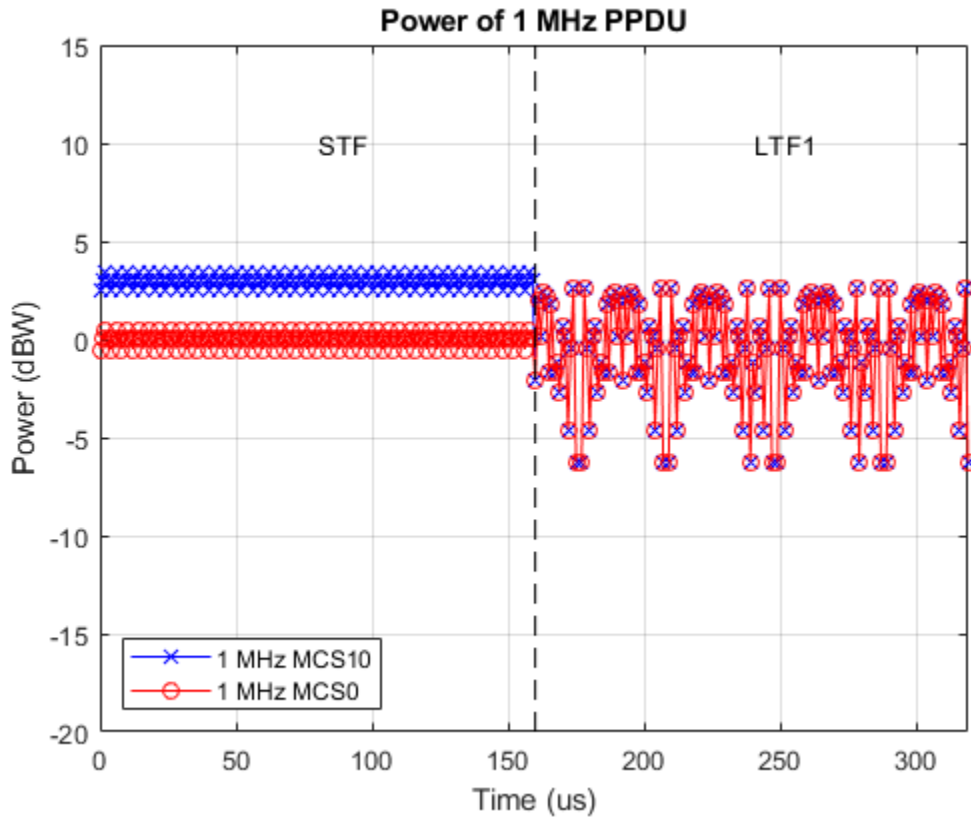
S1G >=2 MHz Long Preamble Mode

The 802.11ah long preamble supports single and multi-user transmissions. The long preamble PPDU consists of two portions; the omni-directional portion and the beam-changeable portion.

The omni-directional portion is transmitted to all users without beamforming. It consists of three fields:

1 STF - The short training field, which is used for coarse synchronization

2 LTF1 - The first long training field, which is used for fine synchronization and initial channel estimation

3 SIG-A - The signaling A field, which the receiver decodes to determine transmission parameters relevant to all users

The beam-changeable portion can be beamformed to each user. It consists of four fields:

1 D-STF - The beamformed short training field, which is used by the receiver for automatic gain control

2 D-LTF - The beamformed long training fields, which is used for MIMO channel estimation

3 SIG-B - The signaling B field. In a multi-user transmission the SIG-B signals the MCS for each user. In a single-user transmission the MCS is signaled in the SIG-A field of the omni-directional portion of the preamble. Therefore in a single-user transmission the SIG-B symbol transmitted is an exact repetition of the first D-LTF. This repetition allows for improved channel estimation.

4 Data - The data field, which carries the user data payload

To visualize repetition of the first D-LTF an S1G 2 MHz long preamble format configuration object is created using the `wlanS1GConfig` function and configured for one space-time stream and one transmit antenna.

```
cfgSU = wlanS1GConfig;
cfgSU.ChannelBandwidth = 'CBW2';
cfgSU.Preamble = 'Long';
cfgSU.NumUsers = 1;
cfgSU.NumSpaceTimeStreams = 1;
cfgSU.NumTransmitAntennas = 1;
cfgSU.MCS = 1;
cfgSU.APEPLength = 150;
```

The >=2 MHz long preamble waveform with a single space-time stream is generated using the `cfgSU` object.

```
% Generate a PSDU containing random bits
psdu = randi([0 1],cfgSU.PSDULength*8,1);

% Generate a PPDU waveform
txSU = wlanWaveformGenerator(psdu,cfgSU);
```

The D-LTF and SIG-B fields are plotted. Note the repetition of the D-LTF in the SIG-B symbol.

```
s1gWavGenPlotSIGB(cfgSU,txSU);
```
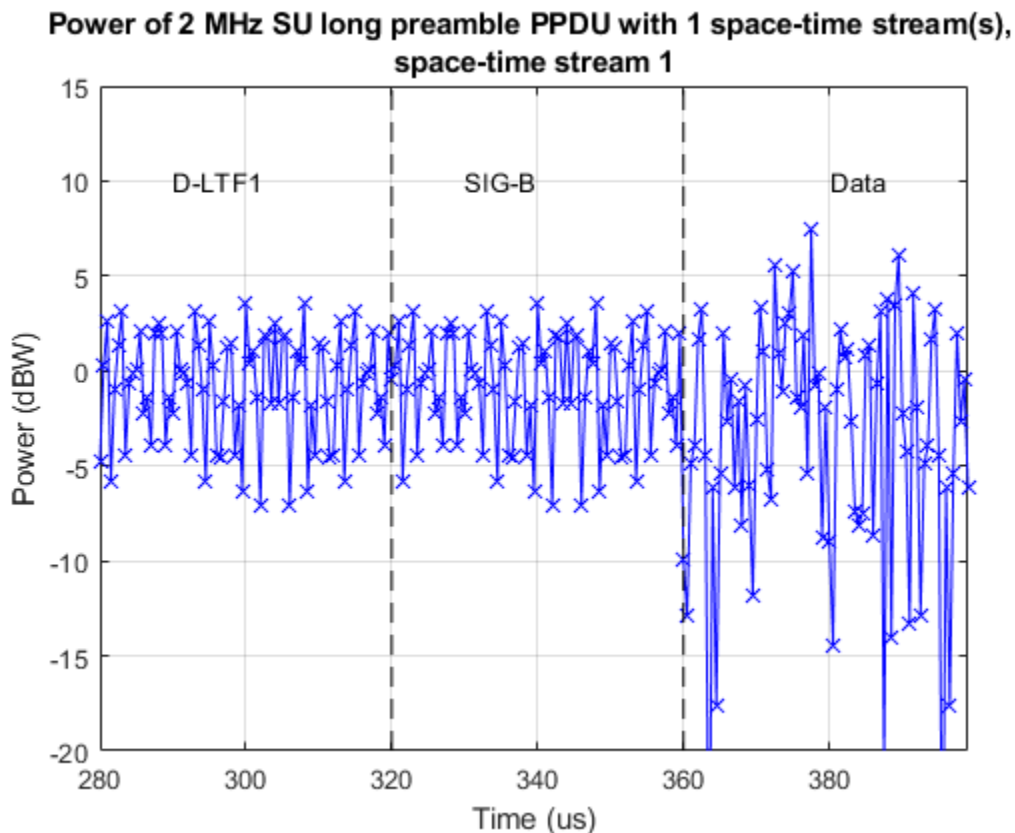

As a comparison a 2 MHz long preamble multi-user waveform will be generated and visualized. First, a format configuration object is created for two users. The user positions, number of space times

streams, MCS and APEP length are configured per user using vectors to parameterize the relevant properties of the `cfgMU` object.

```
cfgMU = wlanS1GConfig;
cfgMU.ChannelBandwidth = 'CBW2';
cfgMU.Preamble = 'Long';
cfgMU.NumUsers = 2;
cfgMU.UserPositions = [0 1];
cfgMU.NumSpaceTimeStreams = [1 1];
cfgMU.NumTransmitAntennas = sum(cfgMU.NumSpaceTimeStreams);
cfgMU.MCS = [1 2];
cfgMU.APEPLength = [150 250];
```

A random PSDU is created for each user and a multi-user waveform is generated. The PSDU length for each user, `cfgMU.PSDULength`, is calculated based on transmission properties by the `cfgMU` object.

```
% Generate cell array containing the PSDUs for all users
psdu = cell(cfgMU.NumUsers,1);
for i = 1:cfgMU.NumUsers
    psdu{i} = randi([0 1],cfgMU.PSDULength(i),1);
end

% Generate waveform
txMU = wlanWaveformGenerator(psdu,cfgMU);
```

The two D-LTF fields and SIG-B fields are plotted for the first space-time stream. Note the SIG-B symbol is no longer a repetition of D-LTF1 as it carries the MCS per user.

```
s1gWavGenPlotSIGB(cfgMU,txMU);
```

Power of 2 MHz MU long preamble PPDU with 2 space-time stream(s), space-time stream 1
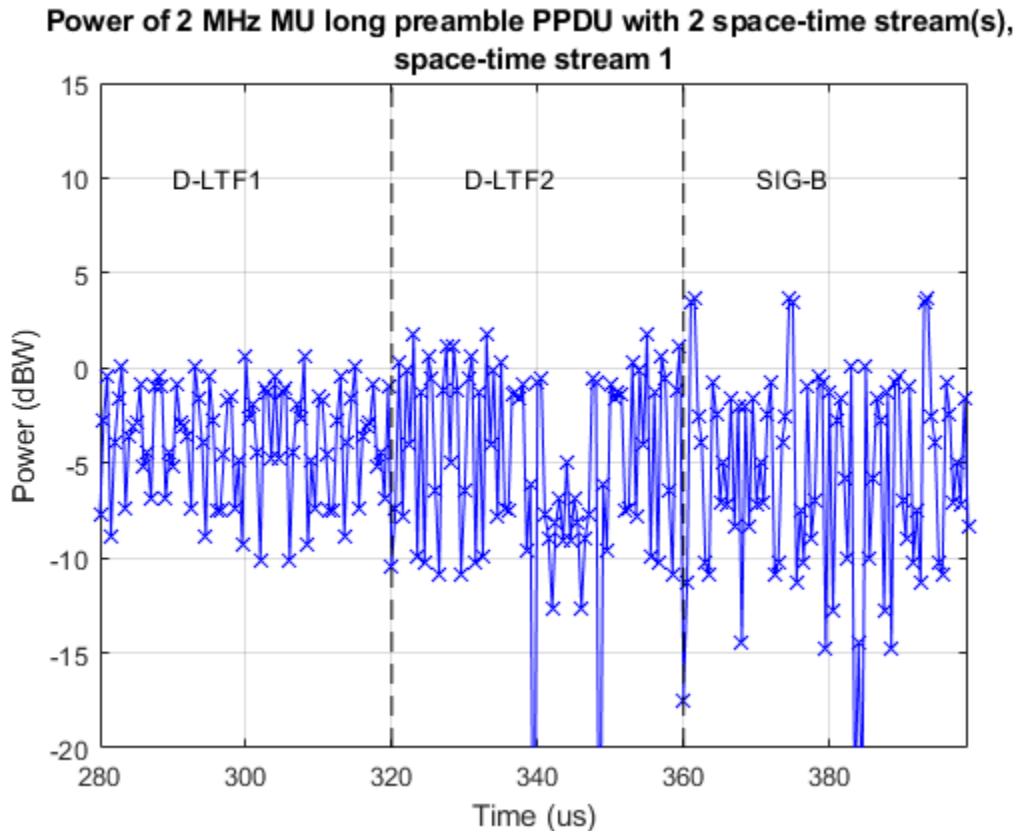
S1G >=2 MHz Short Preamble Mode

An S1G >=2 MHz short preamble waveform consists of five fields, all of which can be beamformed:

1 STF - The short training field, which is used for coarse synchronization
2 LTF1 - The first long training field, which is used for fine synchronization and initial channel estimation
3 SIG - The signaling field, which the receiver decodes to determine transmission parameters
4 LTF2-N - The subsequent long training fields, which is used for MIMO channel estimation
5 Data - The data field, which carries the user data payload

In this example S1G 2 MHz short preamble waveforms with and without traveling pilots will be generated.

Traveling pilots are an optional feature for all three S1G modes to allow for outdoor links where Doppler spread is potentially introduced due to moving vehicles. The traveling pilots are boosted 1.5 times compared to fixed pilots to improve channel estimation performance in this environment [3].

Two >= 2 MHz short preamble waveforms are generated; one with fixed pilots and one with traveling pilots. First a S1G 2 MHz short preamble format configuration with fixed pilot locations is created using the `wlanS1GConfig` object.

```
cfgFix = wlanS1GConfig;
cfgFix.ChannelBandwidth = 'CBW2';
cfgFix.Preamble = 'Short';
```

```
cfgFix.NumTransmitAntennas = 1;
cfgFix.NumSpaceTimeStreams = 1;
cfgFix.MCS = 0; % BPSK so same power on all subcarriers for analysis
cfgFix.APEPLength = 100;
cfgFix.TravelingPilots = false; % Fixed pilot subcarriers
```

Generate a fixed pilot waveform using the `cfgFix` object and random PSDU bits. The PSDU bits are created using the required length for the specified format configuration.

```
% Generate a PSDU containing random bits
psdu = randi([0 1],cfgFix.PSDULength*8,1);

% Generate a PPDU waveform
txFix = wlanWaveformGenerator(psdu,cfgFix);
```

Extract the data field from the time domain waveform using the known duration of the preamble. Plot the magnitude of the OFDM symbols and subcarriers. The location of nulls, data carrying subcarriers, and pilot carrying subcarriers are highlighted. The pilot locations remain unchanged for the duration of the packet.

```
s1gWavGenPlotGrid(txFix,cfgFix, ...
    'Demodulated OFDM symbols with fixed pilots highlighted')
```
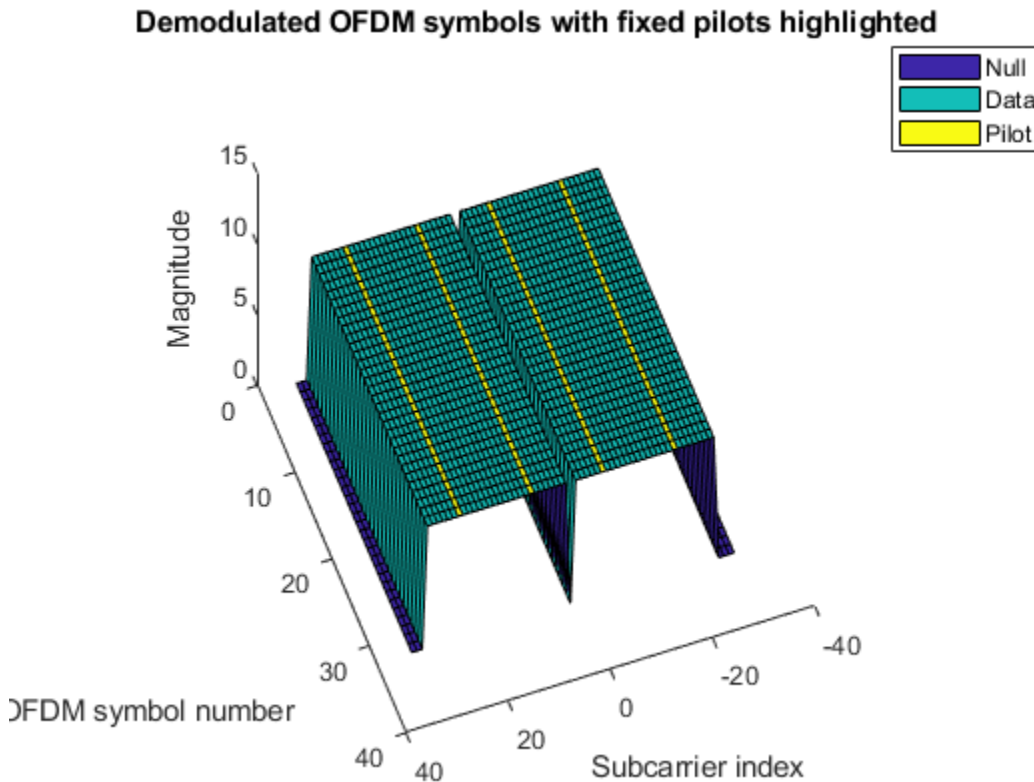

Now a waveform is generated using the same configuration but with traveling pilots. This could be accomplished by changing the `TravelingPilots` property of the existing configuration object and regenerating the waveform, but in this example a separate object is created and used.

```
% Copy the format configuration object and enable traveling pilots
cfgTravel = cfgFix;
cfgTravel.TravelingPilots = true;

% Generate waveform with traveling pilots
txTravel = wlanWaveformGenerator(psdu,cfgTravel);
```

The magnitude of the OFDM symbols and subcarriers is plotted again. The pilot locations now change per OFDM symbol. The magnitude of pilot subcarriers is 1.5 times that of data carrying subcarriers.

```
s1gWavGenPlotGrid(txTravel,cfgTravel, ...
    'Demodulated OFDM symbols with traveling pilots highlighted')
```

Demodulated OFDM symbols with traveling pilots highlighted
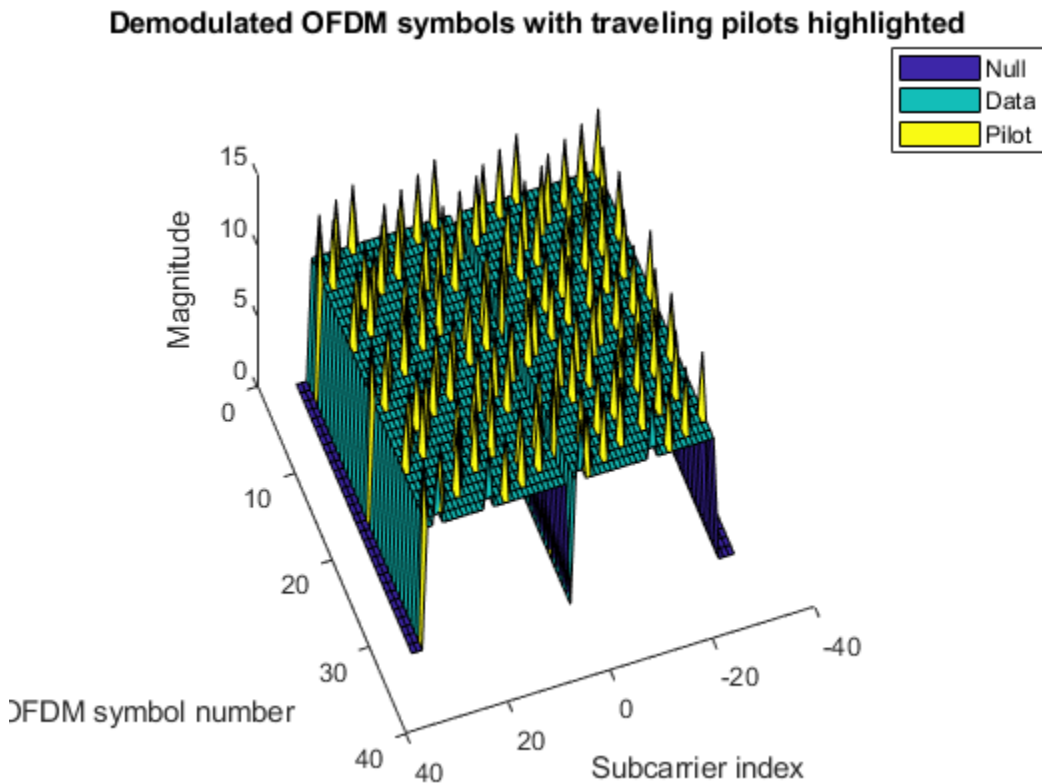
Conclusion

This example has demonstrated how to generate waveforms for different 802.11ah S1G modes and highlighted some of the key features of the standard.

Appendix

This example uses the following helper functions:

- s1gWavGenPlotFieldOverlay.m
- s1gWavGenPlotSIGB.m
- s1gWavGenPlotGrid.m

Selected Bibliography

1 IEEE P802.11ah™/D5.0 Draft 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 2: Sub 1 GHz License Exempt Operation.

2 Sameer Vermani et al. "Preamble Format for 1 MHz", IEEE 802.11-11/1482r4, 2012-01-16.

3 Ron Porat et al. "Traveling Pilots", IEEE 8902.11-12/1322r0, 2012-11-12.

802.11n Link in Simulink

This example shows how to simulate an IEEE® 802.11n™ HT link in Simulink® with WLAN Toolbox™.

Introduction

An 802.11n HT [1] link with a fading channel is simulated in this model. Multiple packets are transmitted through a 2-by-2 TGn MIMO channel, demodulated and the PSDUs recovered. The PSDUs are compared to those transmitted to determine the packet error rate. Packet detection, timing synchronization, carrier frequency offset correction and pilot phase tracking are performed by the receiver.

The MATLAB Function block allows MATLAB® functions to be used in a Simulink model. In this example an 802.11n link is modeled in Simulink by using MATLAB Function blocks to call WLAN Toolbox functions. For an equivalent 802.11n simulation in MATLAB see the example "802.11n Packet Error Rate Simulation for 2x2 TGn Channel" on page 5-17.

Structure of the Example

The model has four main parts:

* Transmitter: Generates a random PSDU and creates an 802.11n HT packet.
* Channel: Models a TGn 2x2 MIMO channel with AWGN.
* Receiver: Recovers the transmitted PSDU by performing packet detection, time and frequency synchronization, MIMO channel estimation and PSDU demodulation and decoding.
* Analysis: Compares the transmitted and recovered PSDUs to determine the packet error rate, and displays the equalized symbols.

The following sections describe the transmitter and receiver in more detail.
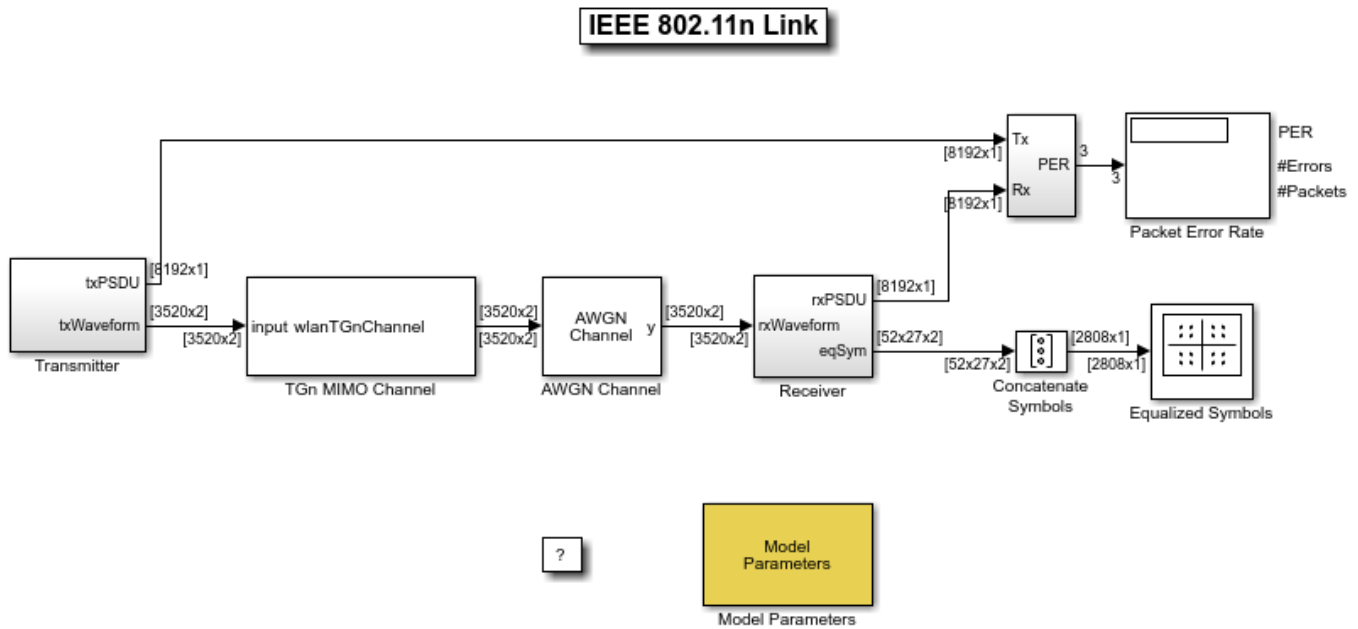
Copyright 2017-2018 The MathWorks, Inc.

Transmitter

The Transmitter block creates a random PSDU and encodes the bits to create a single packet waveform. The `wlanWaveformGenerator` function is called within the Packet Generator block to generate a waveform for a packet. An idle period is added after each packet to create periodic bursts.
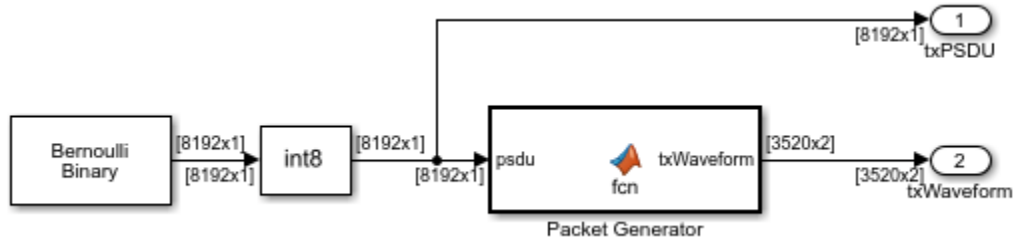
Receiver

The receiver has two components: packet detection and packet recovery.
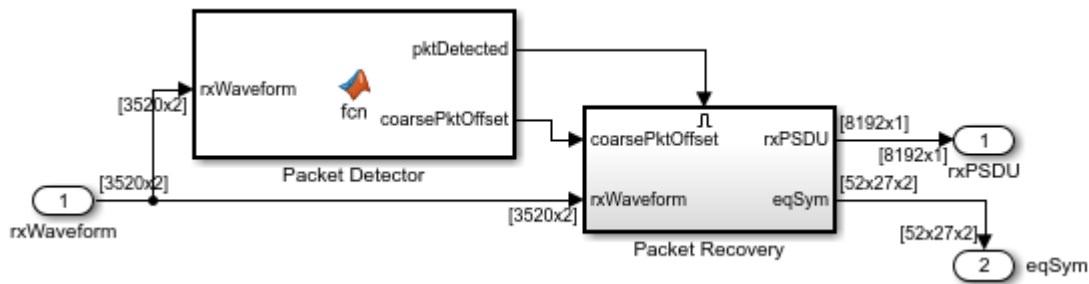
The `wlanPacketDetect` function is called within the Packet Detector block. If a packet is detected the Packet Recovery subsystem is enabled to process the detected packet.

The Packet Recovery subsystem processing consists of the following steps:

1 Coarse carrier frequency offset is estimated and corrected.
2 Fine timing synchronization is established.
3 Fine carrier frequency offset is estimated and corrected.
4 The HT-LTF is extracted from the synchronized received waveform. The HT-LTF is OFDM demodulated and channel estimation is performed.
5 The HT Data field is extracted from the synchronized received waveform.
6 Noise estimation is performed using the demodulated data field pilots.
7 The PSDU is recovered using the extracted field, the channel and noise power estimates.
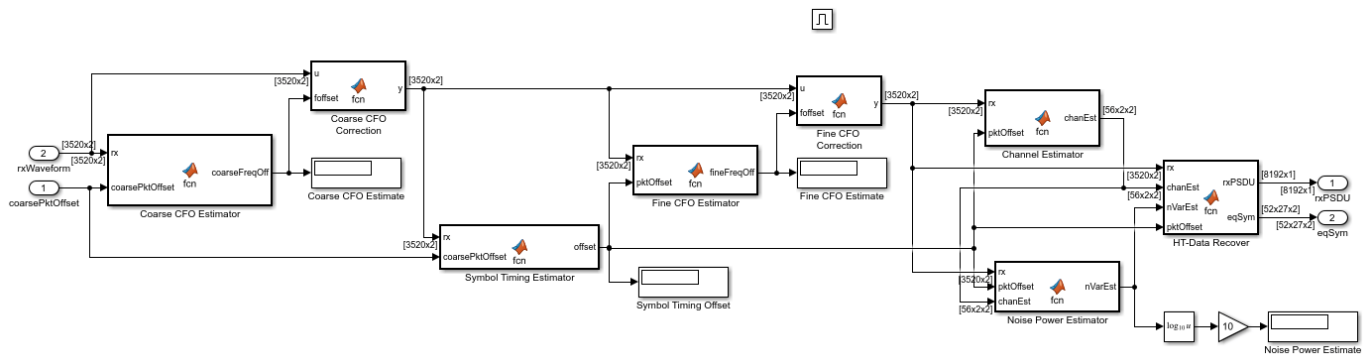
Results and Displays

When the simulation is run, the packet error rate is displayed. This is updated after each packet is processed. The equalized data symbols are also displayed for each packet processed. By default, 200 packets are simulated.
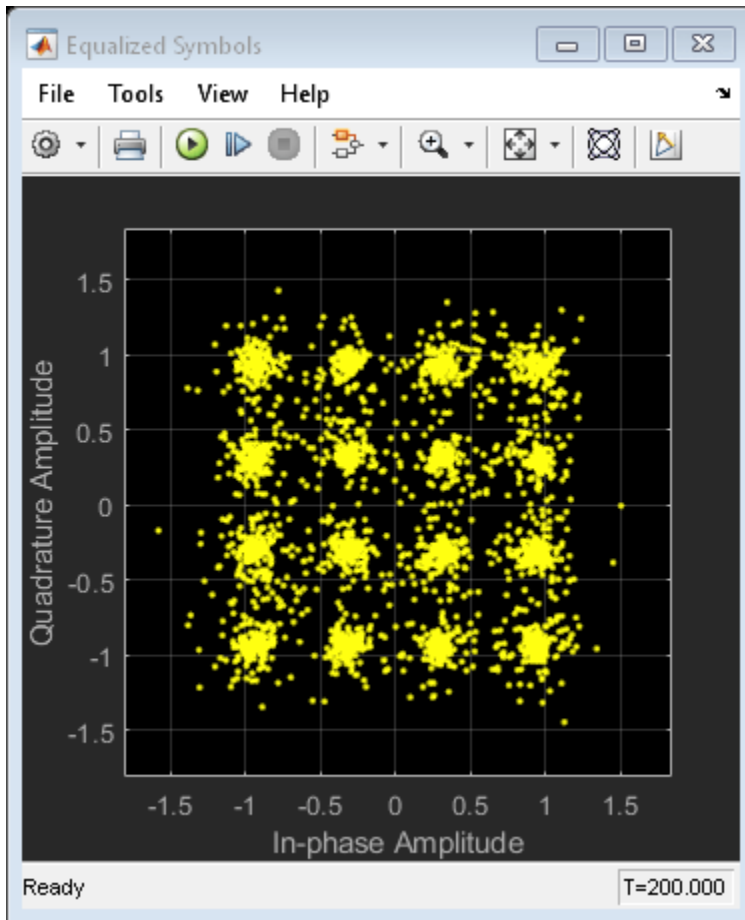
Exploring the Example

Try changing the signal to noise ratio (SNR) in the AWGN channel block. Decreasing the SNR increases the packet error rate and the noise visible in the equalized symbol constellation. Link parameters such as the modulation and coding scheme (MCS), number of transmit and receive antennas and space-time streams can be changed in the Model Parameters block.

Selected Bibliography

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

Signal Reception

Recover and Analyze Packets in 802.11 Waveform

This example blindly detects, decodes, and analyzes multiple IEEE 802.11a™, IEEE 802.11n™, IEEE 802.11ac™, and IEEE 802.11ax™ packets in a waveform. The example provides a summary of the detected packets and displays the MAC contents, error vector magnitude (EVM), power, and signaling information for a selected packet.

Introduction

In this example we detect, decode, and analyze multiple packets within a waveform. This example can decode OFDM non-HT, non-HT duplicate, HT, VHT [1 on page 4-0   ], HE MU, HE SU, and HE ER SU [2 on page 4-0   ] packet formats. The receiver does not know any transmission parameters, except for the channel bandwidth, and retrieves these parameters by decoding the preamble fields of the packet. These measurements are displayed for a detected packet:

- The spectrum and time domain samples.
- The signaling field contents.
- The resource unit and user information for an HE waveform.
- The constellation of the equalized data symbols.
- The EVM of the signaling fields.
- The EVM per data subcarrier averaged over spatial streams and symbols.
- The EVM per data symbol averaged over spatial streams and subcarriers.
- The MAC frame contents: A-MPDU deaggregation status, Address1, Address2, Frame Check Sequence (FCS) and Frame Type.

Setup Waveform Recovery Parameters

This example analyzes I/Q data containing non-HT and HE packets. The `useSDR` variable controls the data source for this example:

- When set to `false`, `comm.BasebandFileReader` reads a synthetic waveform stored in a binary file format.
- When set to `true`, an SDR platform captures an off-the-air-waveform from a commercial device.

`useSDR =` ` False ▼ ` `;`

Reception with an SDR Device

This section demonstrates how to capture an off-the-air-waveform using an SDR device. Communications Toolbox supports radio communication with several SDR devices. For a list of supported devices and for an overview of how to install and setup a support package for your device, refer to the documentation of these support packages.

Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio

- "Installation and Setup" (Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio) and "Supported Hardware" (Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio)
- Use `Pluto` as `deviceName`

Communications Toolbox Support Package for USRP Embedded Series Radio

- "Installation and Setup" (Communications Toolbox Support Package for USRP Embedded Series Radio) and "Hardware Support" (Communications Toolbox Support Package for USRP Embedded Series Radio)

- Use `E3xx` as `deviceName`

Communications Toolbox Support Package for Xilinx Zynq-Based Radio

- "Installation and Setup" (Communications Toolbox Support Package for Xilinx Zynq-Based Radio) and "Hardware Support" (Communications Toolbox Support Package for Xilinx Zynq-Based Radio)

- Use `AD936x` or `FMCOMMS5` as `deviceName`

```matlab
% Configure the analysis source
if useSDR %#ok<*UNRCH>
    deviceName     = Pluto ▼; % Device name
    frequencyBand  = 5 GHz ▼; % Frequency band
    channelNumber  = 102; % WLAN channel number
    captureTime    = 0.1; % Signal capture duration in seconds
    sr             = 20000000; % Captured sampling rate
    chanBW         = CBW20 ▼; % Channel bandwidth of all packets within the wavefor

    % Set the center frequency to the corresponding channel number. The
    % center frequency should be centered on the channel bandwidth.
    fc = helperWLANChannelFrequency(channelNumber,frequencyBand);
    rx = sdrrx(deviceName,'BasebandSampleRate',sr,'CenterFrequency',fc,'OutputDataType','double'
    rxWaveform = capture(rx,captureTime,'Seconds');
    release(rx);
end
```

To view an example on how to recover 802.11a™ OFDM non-HT beacon frames transmitted over the air from the commercial 802.11 hardware see the "802.11 OFDM Beacon Receiver with USRP® Hardware" on page 10-3 example.

Import a Captured Waveform from a File

This section loads I/Q data from an existing binary file using `comm.BasebandFileReader`. The baseband file format includes the sample rate and the number of channels in the captured waveform. Alternately you can load the waveform in a MAT-file format.

```matlab
% Configure the analysis source
if ~useSDR
    BBR = comm.BasebandFileReader('wlanWaveform.bb'); % Create a baseband file reader object
    chanBW = 'CBW20'; % Channel bandwidth of all packets within the waveform
    bbrInfo = info(BBR);
    BBR.SamplesPerFrame = bbrInfo.NumSamplesInData; % Number of sample in the waveform
    rxWaveform = BBR();  % Load the I&Q sample from a binary file
    sr = BBR.SampleRate; % Sampling rate of the input signal
    release(BBR);
end
```

To view an example that recovers beacon packets from a baseband file see the "802.11 OFDM Beacon Receiver with Captured Data" on page 4-41 example.

Signal Recovery and Analysis

This section detects, analyzes, and displays a summary of the detected packets. All packets in the waveform must have the specified channel bandwidth, `chanBW.`

```
% Create a WaveformAnalyzer object to parse and analyze the packet within a waveform
analyzer = WaveformAnalyzer;
process(analyzer,rxWaveform,chanBW,sr);

% Display the summary of the detected packets
detectionSummary(analyzer);
```

Summary of the Detecte

detSummary=*11×9 table*

Number	Format	PHY Status	Power (dBm)	CFO (Hz)	Offset (samples)	MAC Conten
1	"Non-HT"	"Success"	12.7	61431	97	"Beacon"
2	"Non-HT"	"Success"	13.08	-39757	2577	"RTS"
3	"Non-HT"	"Success"	13.01	62250	4017	"CTS"
4	"HE-MU"	"Success"	14.98	-39660	5297	"A-MPDU"
5	"Non-HT"	"Success"	13.04	-39437	18657	"Block Ac
6	"Non-HT"	"Success"	13.07	-29899	20417	"RTS"
7	"Non-HT"	"Success"	13.01	52489	21857	"CTS"
8	"VHT"	"Success"	17.43	62290	23137	"A-MPDU"
9	"Non-HT"	"Success"	14.99	-38861	28337	"RTS"
10	"Non-HT"	"Success"	14.94	42363	29777	"CTS"
11	"HT-MF"	"Success"	15.03	22238	31058	"A-MPDU"

Use the `pktNum` variable to display the MAC and PHY analysis for a selected packet.

pktNum = [4] [Run] ;

```
% Display the MAC information of the selected packet
macSummary(analyzer,pktNum);
```

Recovered MPDU Summary of Packet 4

AMPDU/MPDU Number	STAID	Address1	Address2	AMPDU/MPDU Decode Status
"AMPDU1_MPDU1"	1	"1342ABC2FF1F"	"00123456789B"	"Success"
"AMPDU2_MPDU1"	2	"23FFAB1234AC"	"00123456789B"	"Success"
"AMPDU3_MPDU1"	3	"13EF35781356"	"00123456789B"	"Success"
"AMPDU4_MPDU1"	4	"159A123AFFFF"	"00123456789B"	"Success"

```
% Display the time samples and the spectrum of the detected packet
plotWaveform(analyzer,pktNum)
```

Detection summary (packet 4)

Power spectrum (packet 4)
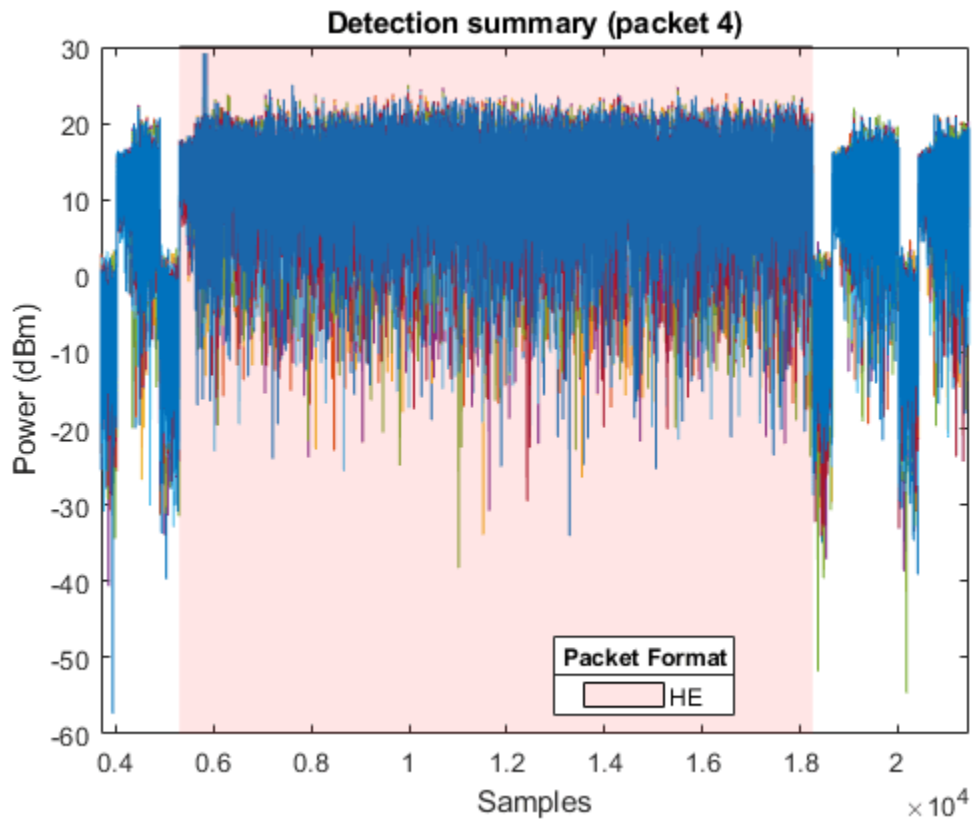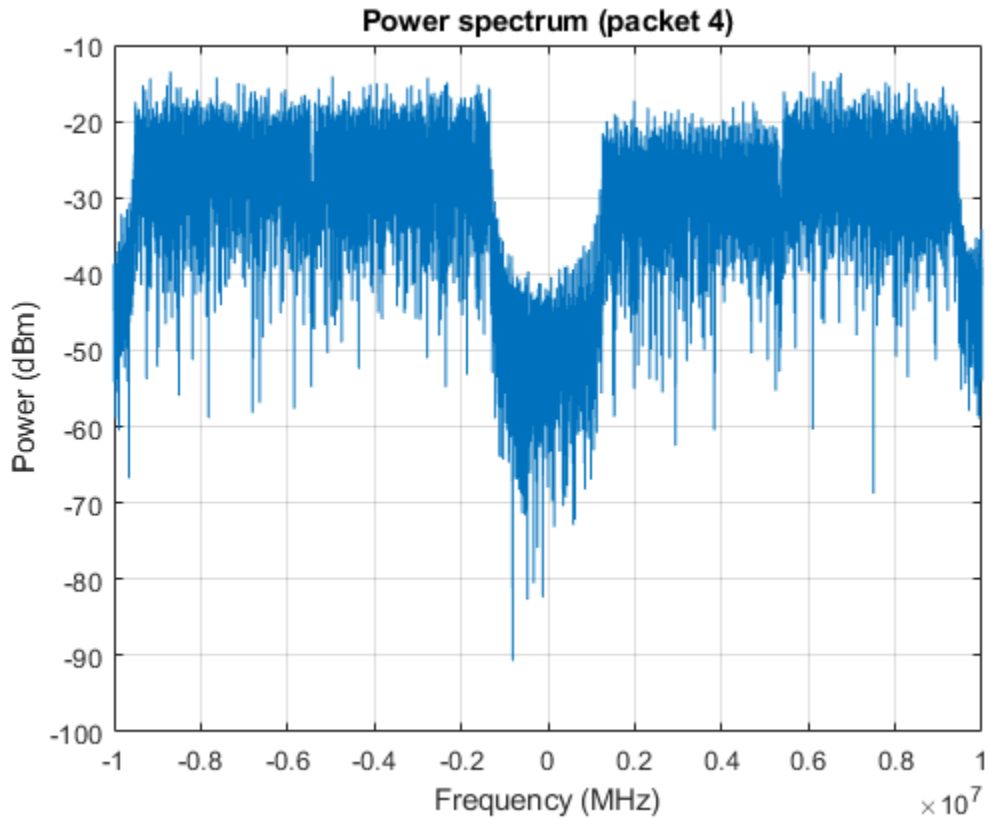

```
% Display the packet field information of the selected packet
fieldSummary(analyzer,pktNum);
```

Field Summary of Packet 4 (HE-MU)

Field Name	Modulation	Num Symbols	Parity Check/CRC	Power (dBm)	RMS EVM (dB)
L-STF	BPSK	2		14.59	
L-LTF	BPSK	2		14.61	
L-SIG	BPSK	1	Pass	14.96	-27.59
RL-SIG	BPSK	1	Pass	14.86	-27.14
HE-SIG-A	BPSK	2	Pass	15.37	-26.14
HE-SIG-B	BPSK	5	Pass	14.98	-27.56
HE-STF	BPSK	1		14.95	
HE-LTF	BPSK	2		15.01	
Data		35		14.99	-25.35

```
% Display signaling field information of the selected packet
signalingSummary(analyzer,pktNum);
```

Signaling Field Summary of Packet 4 (HE-MU)

Property	Value	Property	Value	Property	Value

```
L-SIG Length        467       Bandwidth              CBW20    Num HE-LTF Symbols        2
L-SIG Rate          0xB       Num HE-SIG-B Symbols   5        LDPC Extra Symbol         True
UL/DL Indication    DL        SIGB Compression       False    STBC                      False
SIGB MCS            0         Guard Interval         3.2      Pre-FEC Padding Factor    1
SIGB DCM            False     HE-LTF Type            4        PE Disambiguity           False
BSS Color           0         Doppler                False
Spatial Reuse       0         TXOP                   127
```

```matlab
% Display the RU information
ruSummary(analyzer,pktNum);
```

```
                          Resource Unit (RU) Information of Packet 4 (HE-MU)

    RU Number    RU Size    Subcarrier Index (Start)    Subcarrier Index (End)    Num Users    Nu
    _____    _____    _____    _____    _____    _

     "RU1"         52                -121                        -70                  1
     "RU2"         52                 -68                        -17                  1
     "RU3"         52                  17                         68                  1
     "RU4"         52                  70                        121                  1
```

```matlab
% Display the user information
userSummary(analyzer,pktNum);
```

```
                          User Information of Packet 4 (HE-MU)

    STAID    RU Number    MCS    Modulation    Code Rate    DCM    Channel Coding    Num STS
    _____    _____    ___    _____    _____    ___    _____    _____

      1       "RU1"        0      "BPSK"        "1/2"        0         "LDPC"           1
      2       "RU2"        2      "QPSK"        "3/4"        0         "LDPC"           1
      3       "RU3"        4      "16QAM"       "3/4"        0         "LDPC"           2
      4       "RU4"        6      "64QAM"       "3/4"        0         "LDPC"           1
```

```matlab
% Display EVM per spatial streams for all user
userEVM(analyzer,pktNum);
```

```
          User EVM per Spatial Stream of Packet 4 (HE-MU)

    STAID    Spatial Stream Index    RMS EVM (dB)    Max EVM (dB)
    _____    _____    _____    _____

      1               1                -26.391         -17.295
      2               1                -27.401         -19.682
      3               1                -23.564         -12.353
      3               2                -23.122         -14.444
      4               1                -27.278         -17.793
```

```matlab
% Plot constellation for all users
plotConstellation(analyzer,pktNum);
```

Equalized data symbols (packet 4)
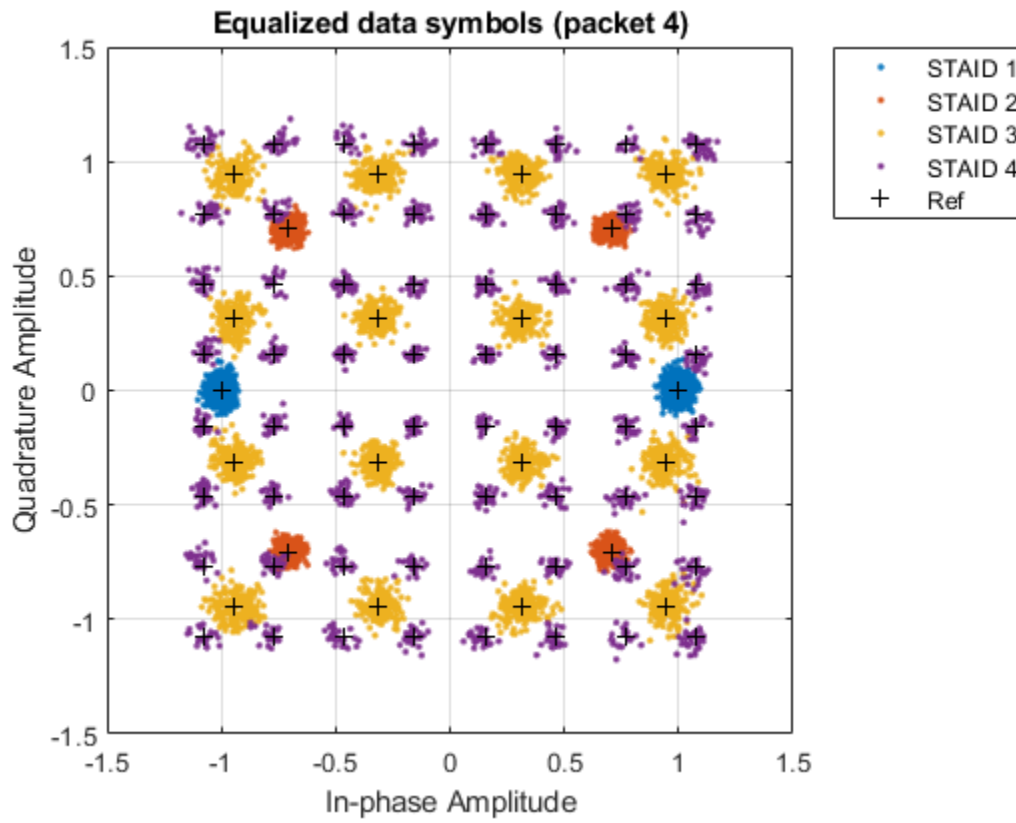
```
% Plot EVM
plotEVM(analyzer,pktNum);
```

Average EVM (RMS) per data subcarrier (packet 4)

Average EVM (RMS) per symbol (packet 4)
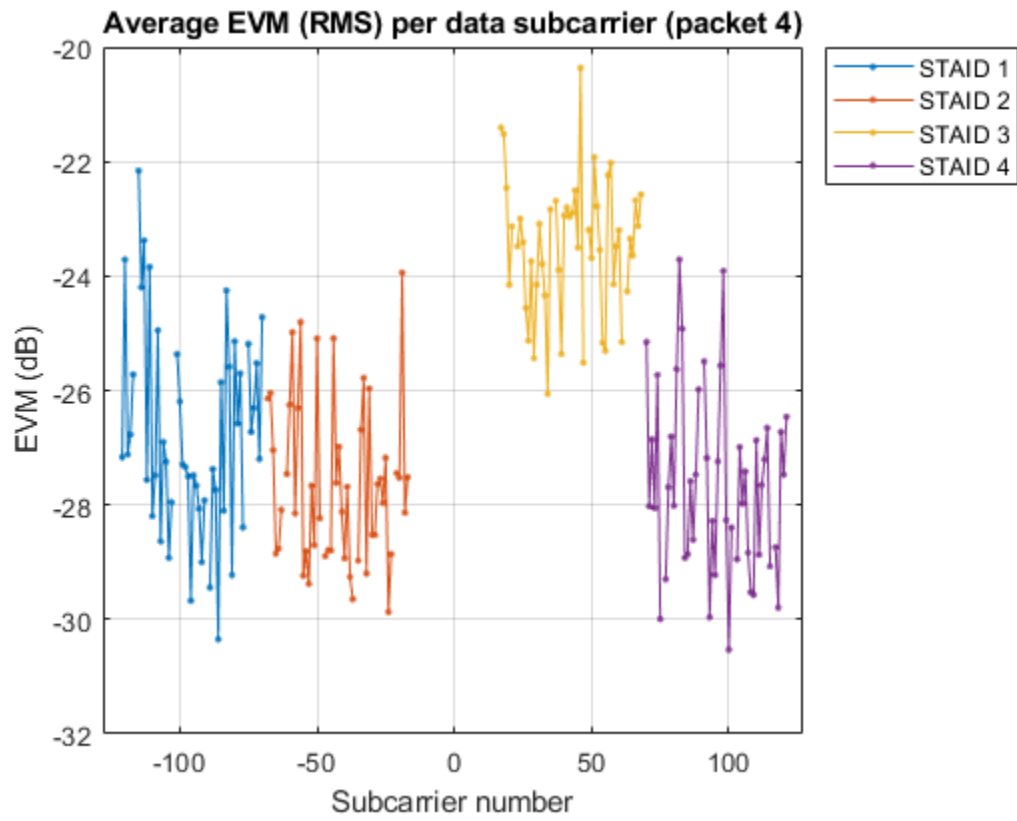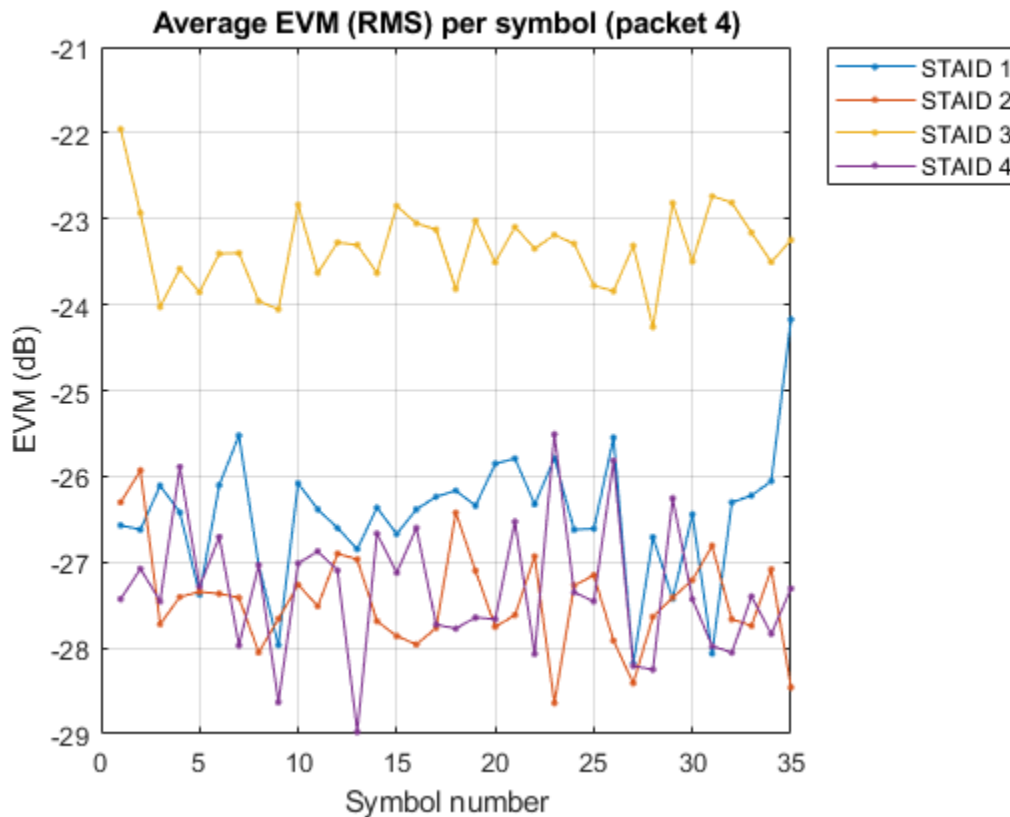
Further Exploration

The `WaveformAnalyzer` provides properties to control the pilot tracking, equalization, DC blocking, and packet detection algorithms that can be tweaked to improve packet detection and analysis performance.

False packet detections

False packet detections are detected packets that you do not believe are actual packets. Evaluating the time domain waveform of the packet is one way to determine if the detected packet is legitimate. If there is a significant number of false detections present these techniques may help reduce them:

- Enable the `EnergyDetection` property and set the `EnergyDetectionThreshold` property to a suitable value given the noise floor of the capture device. When enabled, `EnergyDetection` only detects packets with a power exceeding `EnergyDetectionThreshold` during the preamble.

- Increase the `LLTFSNRDetectionThreshold` and `PacketDetectionThreshold` properties to discard packets with a low measured SNR during detection.

Missed packet detections

Missed packet detections are packets that you believe are in the waveform but have not been detected.

- One possible reason a packet detection may have been missed is if a false detection occurred earlier in the waveform, but the L-SIG check passed, causing samples to be skipped. To search within possible false detections, enable the `SearchWithinUnsupportedPacket` property.

- Alternatively try decreasing the `PacketDetectionThreshold` property to detect packet with low SNR during detection.

For detail on 802.11ax and 802.11ac signal recovery and processing, see the "Recovery Procedure for an 802.11ax Packet" on page 4-12 and "Recovery Procedure for an 802.11ac Packet" on page 4-30 examples.

Selected Bibliography

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

2   IEEE P802.11ax™/D4.1 Draft 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 6: Enhancements for High Efficiency WLAN.

Recovery Procedure for an 802.11ax Packet

This example shows how to detect a packet and decode payload bits in a received IEEE® 802.11ax™ waveform. The receiver recovers the packet format parameters from the preamble fields to decode the data field and the MAC frame.

Introduction

In an 802.11ax packet the transmission parameters are signaled to the receiver using the L-SIG, HE-SIG-A, and HE-SIG-B preamble fields [1]:

- The L-SIG field contains information to allow the receiver to determine the transmission time of the packet.
- The HE-SIG-A field contains common transmission parameters for HE-MU users and all transmission parameters for HE-SU and HE-EXT-SU packets.
- The combination of length information in the L-SIG field, the modulation type, and the number of OFDM symbols in the HE-SIG-A field determines the HE packet format.
- The HE-SIG-B field contains Resource unit (RU) allocation information and the transmission parameters for the users in an HE-MU packet.

In this example we detect and decode an HE-MU packet within a generated waveform. This example can also recover HE-SU and HE-EXT-SU packets. All transmission parameters apart from the channel bandwidth are assumed to be unknown and are therefore retrieved from the decoded L-SIG, HE-SIG-A, and HE-SIG-B preamble fields. The recovered transmission parameters are used to decode the HE-Data field. Additionally, the following analysis is performed:

- The waveform of the detected packet is recovered and displayed.
- The spectrum of the detected packet is recovered and displayed.
- The constellation of the equalized data symbols for all spatial streams is displayed.
- The Error Vector Magnitude (EVM) of each field is measured.
- An A-MPDU is detected and Frame Check Sequence (FCS) is determined for the recovered MAC frame.
- The EVM per data symbol and spatial stream averaged over subcarriers is displayed.
- The EVM per data subcarrier and spatial stream averaged over symbols is displayed.

Waveform Transmission

In this example an 802.11ax HE-MU waveform is synthesized but you can use a captured waveform. You can use MATLAB® to acquire I/Q data from a wide range of instruments with the Instrument Control Toolbox™ and software defined radio platforms.

The synthesized waveform is impaired by a 2x2 TGax indoor fading channel, additive white Gaussian noise, and carrier frequency offset. To generate an HE-MU waveform we configure an HE-MU format configuration object wlanHEMUConfig. Note that the wlanHEMUConfig configuration object is used at the transmitter side only. The receiver will formulate an HE recovery configuration object wlanHERecoveryConfig. The unknown properties of the HE recovery configuration object are set after decoding the information bits in the L-SIG, HE-SIG-A, and HE-SIG-B fields. The helper function heSigRecGenerateWaveform generates the impaired waveform. The following processing steps are performed:

- A random payload of MSDUs is created for the MAC frame, which is encoded into an HE-MU packet.
- The waveform is passed through a TGax indoor fading channel model.
- Carrier frequency offset (CFO) and Additive white Gaussian noise (AWGN) are added to the waveform.

```
% A mixed OFDMA and MU-MIMO configuration is defined for an HE-MU packet.
% The allocation index 17 defines two 52-tone RUs, with one user in each
% RU, and one 106-tone RU. The 106-tone RU has two users in a MU-MIMO
% configuration.
cfgMU = wlanHEMUConfig(17);
cfgMU.NumTransmitAntennas = 2;

% Configure RU 1 and user 1
cfgMU.RU{1}.SpatialMapping = 'Direct';
cfgMU.User{1}.STAID = 1;
cfgMU.User{1}.APEPLength = 1e3;
cfgMU.User{1}.MCS = 5;
cfgMU.User{1}.NumSpaceTimeStreams = 2;
cfgMU.User{1}.ChannelCoding = 'LDPC';

% Configure RU 2 and user 2
cfgMU.RU{2}.SpatialMapping = 'Fourier';
cfgMU.User{2}.STAID = 2;
cfgMU.User{2}.APEPLength = 500;
cfgMU.User{2}.MCS = 4;
cfgMU.User{2}.NumSpaceTimeStreams = 1;
cfgMU.User{2}.ChannelCoding = 'BCC';

% Configure RU 3 and user 1
cfgMU.RU{3}.SpatialMapping = 'Fourier';
cfgMU.User{3}.STAID = 3;
cfgMU.User{3}.APEPLength = 100;
cfgMU.User{3}.MCS = 2;
cfgMU.User{3}.NumSpaceTimeStreams = 1;
cfgMU.User{3}.ChannelCoding = 'BCC';

% Configure RU 3 and user 2
cfgMU.User{4}.STAID = 4;
cfgMU.User{4}.APEPLength = 500;
cfgMU.User{4}.MCS = 3;
cfgMU.User{4}.NumSpaceTimeStreams = 1;
cfgMU.User{4}.ChannelCoding = 'LDPC';

% Specify propagation channel
numRx = 2; % Number of receive antennas
delayProfile = 'Model-D'; % TGax channel delay profile

% Specify impairments
noisePower = -40; % Noise power to apply in dBW
cfo = 62e3; % Carrier frequency offset Hz

% Generate waveform
rx = heSigRecGenerateWaveform(cfgMU,numRx,delayProfile,noisePower,cfo);
```

Packet Recovery

The signal to process is stored in the variable rx. The processing steps to recover a packet are:

- An HE recovery configuration object, wlanHERecoveryConfig is created. The object properties are updated as preamble fields are decoded.
- The packet is detected and synchronized.
- The L-LTF is extracted and demodulated. The demodulated L-LTF symbols do not include tone rotation for each 20 MHz segment as described in [2], section 21.3.7.5.
- The demodulated L-LTF symbols are used for channel and noise estimates.
- The time-domain signal containing samples equivalent to four OFDM symbols immediately following the L-LTF are used to determine the HE packet format. The packet format is updated in the wlanHERecoveryConfig object.
- The L-LTF is demodulated. The demodulated L-LTF symbols include tone rotation for each 20 MHz segment as described in [2], section 21.3.7.5. The L-LTF channel estimates (with tone rotation) are used to decode the pre-HE-LTF.
- The L-SIG and RL-SIG fields are extracted. The channel is estimated on an extra four subcarriers per subchannel in the L-SIG and RL-SIG fields. The L-LTF channel estimates are updated to include the channel estimates on the extra subcarriers.
- The information bits in the L-SIG field are recovered to determine the length of the packet in microseconds.
- After HE-SIG-A decoding, the recovery configuration object is updated with common transmission parameters for an HE-MU packet and all transmission parameters for HE-SU and HE-EXT-SU packets.
- For an HE-MU packet format the HE-SIG-B field is decoded. For a non-compressed SIGB waveform the HE-SIG-B common field is decoded first followed by HE-SIG-B user field. For a compressed SIGB waveform only the HE-SIG-B user field is decoded.
- For an HE-MU format without SIGB compression the RU allocation and user transmission parameters are recovered from the HE-SIG-B field. For a compressed SIGB waveform the RU allocation information is inferred from HE-SIG-A field and the user transmission parameters are determined from the HE-SIG-B user field bits.
- The wlanHERecoveryConfig object is created using the recovered transmission parameters for each user after HE-SIG-B decoding.
- The HE-LTF field is extracted and demodulated. The demodulated symbols are used for channel estimation of the subcarriers allocated to the user of interest. The MIMO channel estimates are used to decode the HE-Data field.
- The HE-Data field is extracted and the PSDU bits are recovered using the wlanHERecoveryConfig object for each user.
- Detect A-MPDU within the recovered PSDU and check the FCS for the recovered MAC frame.

Setup Waveform Recovery Parameters

In this example all transmission parameters apart from the channel bandwidth are assumed to be unknown and will be recovered. A recovery configuration object, wlanHERecoveryConfig, is created to store the recovered information in the L-SIG, HE-SIG-A, and HE-SIG-B preamble fields. The transmission properties in wlanHERecoveryConfig are updated after subsequent decoding of the preamble fields. The following code configures objects and variables for processing.

```
chanBW = cfgMU.ChannelBandwidth; % Assume channel bandwidth is known
sr = wlanSampleRate(cfgMU); % Sample rate
```

```
% Specify pilot tracking method for recovering the data field. This can be:
%  'Joint' - use joint common phase error and sample rate offset tracking
%  'CPE' - use only common phase error tracking
% When recovering 26-tone RUs only CPE tracking is used as the joint
% tracking algorithm is susceptible to noise.
pilotTracking = 'Joint';

% Create an HE recovery configuration object and set the channel bandwidth
cfgRx = wlanHERecoveryConfig;
cfgRx.ChannelBandwidth = chanBW;

% The recovery configuration object is used to get the start and end
% indices of the pre-HE-SIG-B field.
ind = wlanFieldIndices(cfgRx);

% Setup plots for the example
[spectrumAnalyzer,timeScope,ConstellationDiagram,EVMPerSubcarrier,EVMPerSymbol] = heSigRecSetupP

% Minimum packet length is 10 OFDM symbols
lstfLength = double(ind.LSTF(2));
minPktLen = lstfLength*5; % Number of samples in L-STF

rxWaveLen = size(rx,1);
```

Front-End Processing

The front-end processing consists of packet detection, coarse carrier frequency offset correction, timing synchronization, and fine carrier frequency offset correction. A `while` loop is used to detect and synchronize a packet within the received waveform. The sample offset `searchOffset` is used to index into `rx` to detect a packet. The first packet within `rx` is detected and processed. If the synchronization fails for the detected packet, the sample index offset `searchOffset` is incremented to move beyond the processed packet in `rx`. This is repeated until a packet has been successfully detected and synchronized.

```
searchOffset = 0; % Offset from start of waveform in samples
while (searchOffset + minPktLen) <= rxWaveLen
    % Packet detection
    pktOffset = wlanPacketDetect(rx,chanBW,searchOffset);

    % Adjust packet offset
    pktOffset = searchOffset + pktOffset;
    if isempty(pktOffset) || (pktOffset + ind.LSIG(2) > rxWaveLen)
        error('** No packet detected **');
    end

    % Coarse frequency offset estimation and correction using L-STF
    rxLSTF = rx(pktOffset+(ind.LSTF(1):ind.LSTF(2)), :);
    coarseFreqOffset = wlanCoarseCFOEstimate(rxLSTF,chanBW);
    rx = helperFrequencyOffset(rx,sr,-coarseFreqOffset);

    % Symbol timing synchronization
    searchBufferLLTF = rx(pktOffset+(ind.LSTF(1):ind.LSIG(2)),:);
    pktOffset = pktOffset+wlanSymbolTimingEstimate(searchBufferLLTF,chanBW);

    % Fine frequency offset estimation and correction using L-STF
    rxLLTF = rx(pktOffset+(ind.LLTF(1):ind.LLTF(2)),:);
```

```
        fineFreqOffset = wlanFineCFOEstimate(rxLLTF,chanBW);
        rx = helperFrequencyOffset(rx,sr,-fineFreqOffset);

        % Timing synchronization complete: packet detected
        fprintf('Packet detected at index %d\n',pktOffset + 1);

        % Display estimated carrier frequency offset
        cfoCorrection = coarseFreqOffset + fineFreqOffset; % Total CFO
        fprintf('Estimated CFO: %5.1f Hz\n\n',cfoCorrection);

        break; % Front-end processing complete, stop searching for a packet
    end

% Scale the waveform based on L-STF power (AGC)
gain = 1./(sqrt(mean(rxLSTF.*conj(rxLSTF))));
rx = rx.*gain;
```

```
Packet detected at index 404
Estimated CFO: 61942.9 Hz
```

Packet Format Detection

The time-domain samples equivalent to four OFDM symbols immediately following the L-LTF are used to determine the HE packet format [1 Figure. 27-63]. The L-LTF is extracted and demodulated. For format detection, the demodulated L-LTF symbols must not include tone rotation for each 20 MHz segment as described in [2], section 21.3.7.5. The demodulated L-LTF is used for channel and noise estimation. The L-LTF channel (without tone rotation) and noise power estimates are used to detect the packet format.

```
rxLLTF = rx(pktOffset+(ind.LLTF(1):ind.LLTF(2)),:);
lltfDemod = wlanLLTFDemodulate(rxLLTF,chanBW);
lltfChanEst = wlanLLTFChannelEstimate(lltfDemod,chanBW);
noiseVar = helperNoiseEstimate(lltfDemod);

disp('Detect packet format...');
rxSIGA = rx(pktOffset+(ind.LSIG(1):ind.HESIGA(2)),:);
pktFormat = wlanFormatDetect(rxSIGA,lltfChanEst,noiseVar,chanBW);
fprintf('  %s packet detected\n\n',pktFormat);

% Set the packet format in the recovery object and update the field indices
cfgRx.PacketFormat = pktFormat;
ind = wlanFieldIndices(cfgRx);
```

```
Detect packet format...
  HE-MU packet detected
```

L-LTF Channel Estimate

Demodulate the L-LTF and perform channel estimation. The demodulated L-LTF symbols include tone rotation for each 20 MHz segment as described in [2], section 21.3.7.5. The L-LTF channel estimates (with tone rotation) are used to equalize and decode the pre-HE-LTF fields.

```
lltfDemod = wlanHEDemodulate(rxLLTF,'L-LTF',chanBW);
lltfChanEst = wlanLLTFChannelEstimate(lltfDemod,chanBW);
```

L-SIG and RL-SIG Decoding

The L-SIG field is used to determine the receive time, or RXTIME, of the packet. The RXTIME is calculated using the length bits of the L-SIG payload. The L-SIG and RL-SIG fields are recovered to perform the channel estimate on the extra subcarriers in the L-SIG and RL-SIG fields. The `lltfChanEst` channel estimates are updated to include the channel estimates on extra subcarriers in the L-SIG and RL-SIG fields. The L-SIG payload is decoded using an estimate of the channel and noise power obtained from the L-LTF field. The L-SIG length property in wlanHERecoveryConfig is updated after L-SIG decoding.

```matlab
disp('Decoding L-SIG... ');
% Extract L-SIG and RL-SIG fields
rxLSIG = rx(pktOffset+(ind.LSIG(1):ind.RLSIG(2)),:);

% OFDM demodulate
helsigDemod = wlanHEDemodulate(rxLSIG,'L-SIG',chanBW);

% Estimate CPE and phase correct symbols
helsigDemod = preHECommonPhaseErrorTracking(helsigDemod,lltfChanEst,'L-SIG',chanBW);

% Estimate channel on extra 4 subcarriers per subchannel and create full
% channel estimate
preheInfo = wlanHEOFDMInfo('L-SIG',chanBW);
preHEChanEst = preHEChannelEstimate(helsigDemod,lltfChanEst,preheInfo.NumSubchannels);

% Average L-SIG and RL-SIG before equalization
helsigDemod = mean(helsigDemod,2);

% Equalize data carrying subcarriers, merging 20 MHz subchannels
[eqLSIGSym,csi] = preHESymbolEqualize(helsigDemod(preheInfo.DataIndices,:,:), ...
    preHEChanEst(preheInfo.DataIndices,:,:),noiseVar,preheInfo.NumSubchannels);

% Decode L-SIG field
[~,failCheck,lsigInfo] = wlanLSIGBitRecover(eqLSIGSym,noiseVar,csi);

if failCheck
    disp(' ** L-SIG check fail **');
else
    disp(' L-SIG check pass');
end
% Get the length information from the recovered L-SIG bits and update the
% L-SIG length property of the recovery configuration object
lsigLength = lsigInfo.Length;
cfgRx.LSIGLength = lsigLength;

% Measure EVM of L-SIG symbols
EVM = comm.EVM;
EVM.ReferenceSignalSource = 'Estimated from reference constellation';
EVM.Normalization = 'Average constellation power';
EVM.ReferenceConstellation = wlanReferenceSymbols('BPSK');
rmsEVM = EVM(eqLSIGSym);
fprintf(' L-SIG EVM: %2.2fdB\n\n',20*log10(rmsEVM/100));

% Calculate the receive time and corresponding number of samples in the
% packet
RXTime = ceil((lsigLength + 3)/3) * 4 + 20; % In microseconds
numRxSamples = round(RXTime * 1e-6 * sr);   % Number of samples in time
```

```
fprintf(' RXTIME: %dus\n',RXTime);
fprintf(' Number of samples in the packet: %d\n\n',numRxSamples);
```

```
Decoding L-SIG...
 L-SIG check pass
 L-SIG EVM: -36.91dB

 RXTIME: 536us
 Number of samples in the packet: 10720
```

The waveform and spectrum of the detected packet within `rx` is displayed given the calculated RXTIME and corresponding number of samples.

```
sampleOffset = max((-lstfLength + pktOffset),1); % First index to plot
sampleSpan = numRxSamples + 2*lstfLength; % Number samples to plot
% Plot as much of the packet (and extra samples) as we can
plotIdx = sampleOffset:min(sampleOffset + sampleSpan,rxWaveLen);

% Configure timeScope to display the packet
timeScope.TimeSpan = sampleSpan/sr;
timeScope.TimeDisplayOffset = sampleOffset/sr;
timeScope.YLimits = [0 max(abs(rx(:)))];
timeScope(abs(rx(plotIdx,:)));
release(timeScope);

% Display the spectrum of the detected packet
spectrumAnalyzer(rx(pktOffset + (1:numRxSamples),:));
release(spectrumAnalyzer);
```
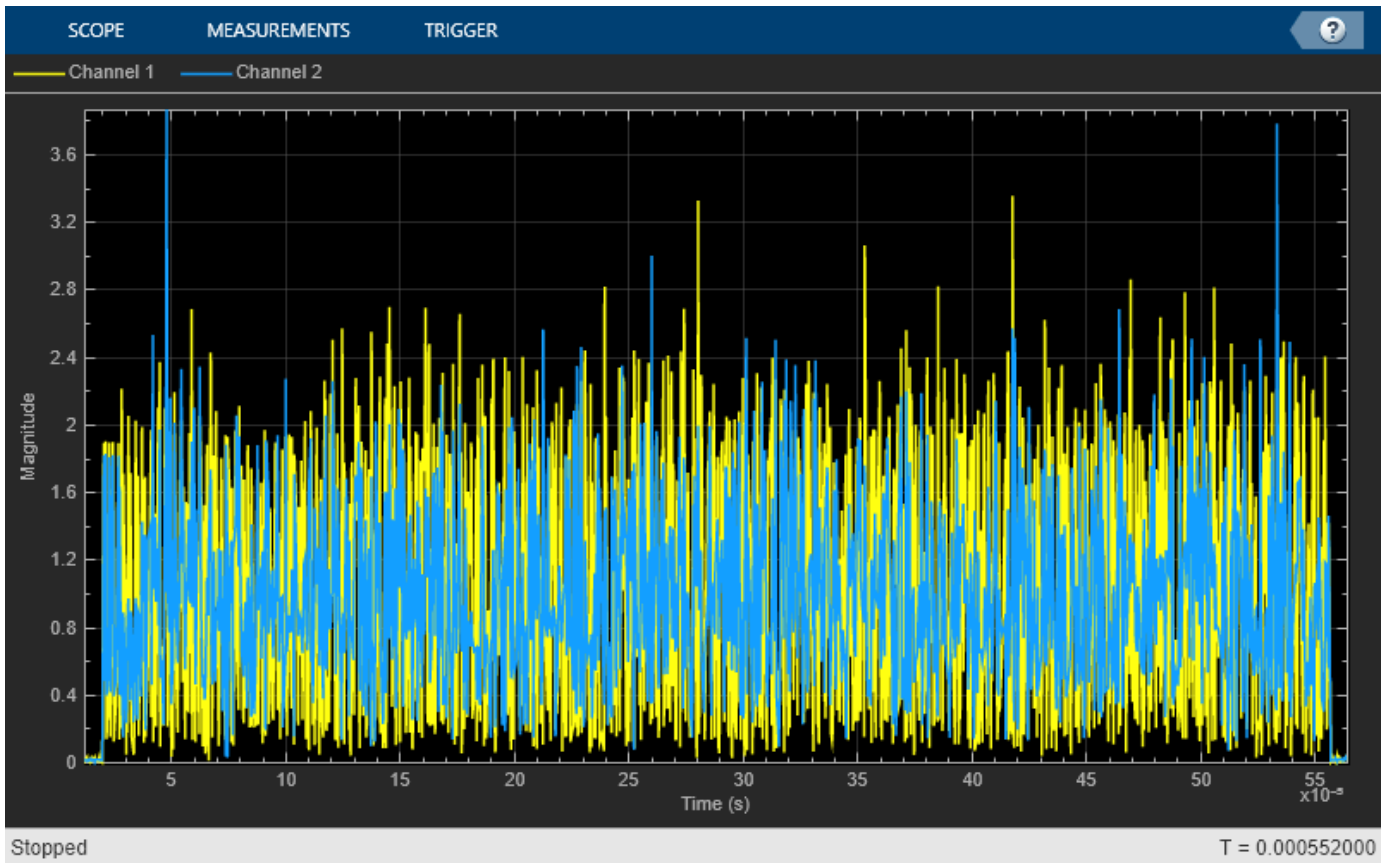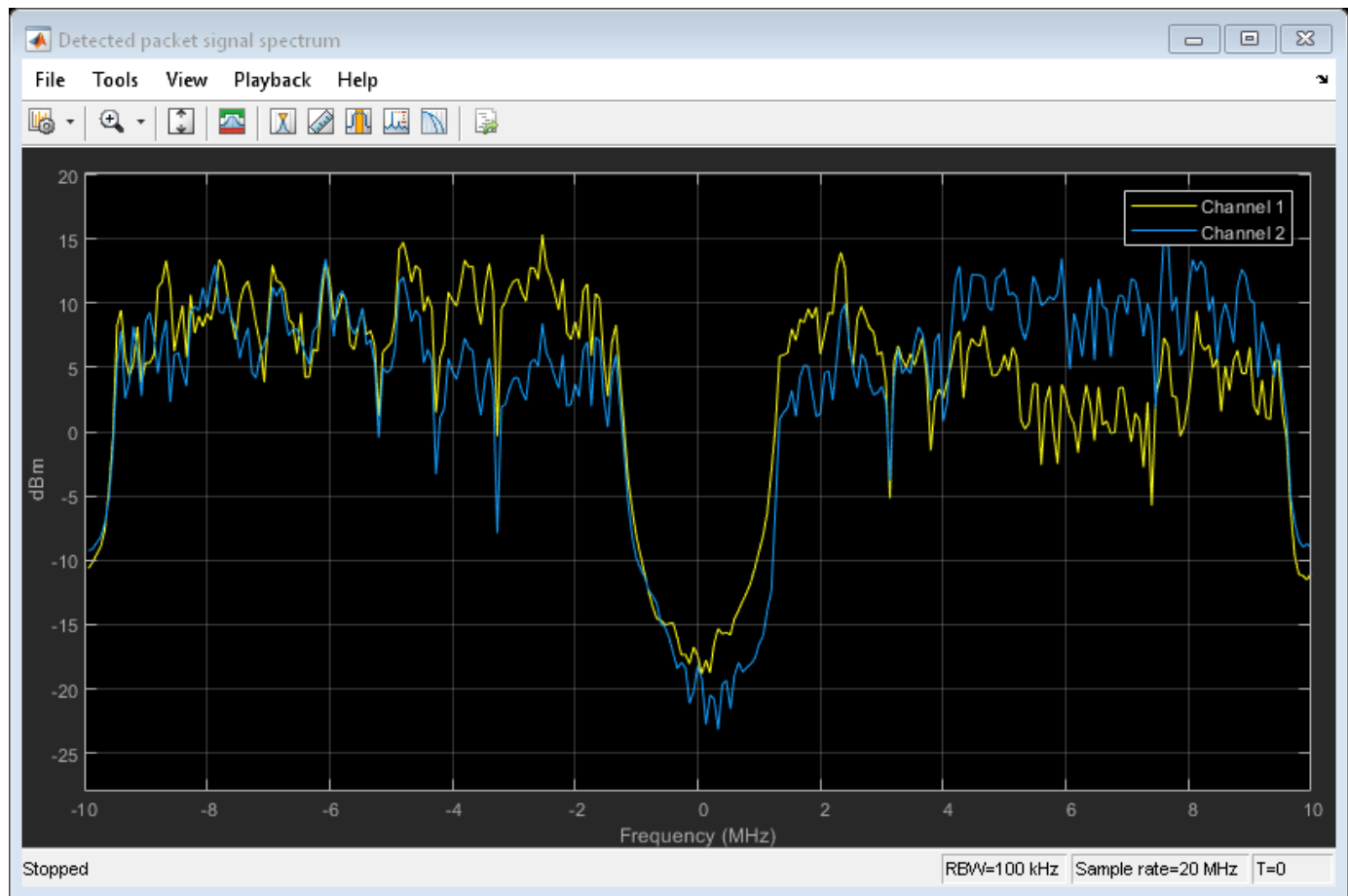
HE-SIG-A Decoding

The HE-SIG-A field contains the transmission configuration of an HE packet. An estimate of the channel and noise power obtained from the L-LTF is required to decode the HE-SIG-A field.

```
disp('Decoding HE-SIG-A...')
rxSIGA = rx(pktOffset+(ind.HESIGA(1):ind.HESIGA(2)),:);
sigaDemod = wlanHEDemodulate(rxSIGA,'HE-SIG-A',chanBW);
hesigaDemod = preHECommonPhaseErrorTracking(sigaDemod,preHEChanEst,'HE-SIG-A',chanBW);

% Equalize data carrying subcarriers, merging 20 MHz subchannels
preheInfo = wlanHEOFDMInfo('HE-SIG-A',chanBW);
[eqSIGASym,csi] = preHESymbolEqualize(hesigaDemod(preheInfo.DataIndices,:,:), ...
                                      preHEChanEst(preheInfo.DataIndices,:,:), ...
                                      noiseVar,preheInfo.NumSubchannels);
% Recover HE-SIG-A bits
[sigaBits,failCRC] = wlanHESIGABitRecover(eqSIGASym,noiseVar,csi);

% Perform the CRC on HE-SIG-A bits
if failCRC
    disp(' ** HE-SIG-A CRC fail **');
else
    disp(' HE-SIG-A CRC pass');
end
```

```
% Measure EVM of HE-SIG-A symbols
release(EVM);
if strcmp(pktFormat,'HE-EXT-SU')
    % The second symbol of an HE-SIG-A field for an HE-EXT-SU packet is
    % QBPSK.
    EVM.ReferenceConstellation = wlanReferenceSymbols('BPSK',[0 pi/2 0 0]);
    % Account for scaling of L-LTF for an HE-EXT-SU packet
    rmsEVM = EVM(eqSIGASym*sqrt(2));
else
    EVM.ReferenceConstellation = wlanReferenceSymbols('BPSK');
    rmsEVM = EVM(eqSIGASym);
end
fprintf(' HE-SIG-A EVM: %2.2fdB\n\n',20*log10(mean(rmsEVM)/100));
```

```
Decoding HE-SIG-A...
 HE-SIG-A CRC pass
 HE-SIG-A EVM: -35.17dB
```

Interpret Recovered HE-SIG-A bits

The wlanHERecoveryConfig object is updated after interpreting the recovered HE-SIG-A bits.

```
cfgRx = interpretHESIGABits(cfgRx,sigaBits);
ind = wlanFieldIndices(cfgRx); % Update field indices
```

Display the common transmission configuration obtained from HE-SIG-A field for an HE-MU packet. The properties indicated by -1 are unknown or undefined. The unknown user-related properties are updated after successful decoding of the HE-SIG-B field.

```
disp(cfgRx)
```

```
  wlanHERecoveryConfig with properties:

                   PacketFormat: 'HE-MU'
                ChannelBandwidth: 'CBW20'
                      LSIGLength: 383
                 SIGBCompression: 0
                         SIGBMCS: 0
                         SIGBDCM: 0
            NumSIGBSymbolsSignaled: 5
                            STBC: 0
                 LDPCExtraSymbol: 0
              PreFECPaddingFactor: 4
                  PEDisambiguity: 0
                   GuardInterval: 3.2000
                       HELTFType: 4
                 NumHELTFSymbols: 2
                UplinkIndication: 0
                        BSSColor: 0
                    SpatialReuse: 0
                   TXOPDuration: 127
                     HighDoppler: 0
                 AllocationIndex: -1
         NumUsersPerContentChannel: -1
            RUTotalSpaceTimeStreams: -1
                          RUSize: -1
                         RUIndex: -1
```

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

HE-SIG-B Decoding

For an HE-MU packet the HE-SIG-B field contains:

- The RU allocation information for a non-compressed SIGB waveform is inferred from HE-SIG-B Common field [1 Table. 27-23]. For a compressed SIGB waveform the RU allocation information is inferred from the recovered HE-SIG-A bits.

- For a non-compressed SIGB waveform the number of HE-SIG-B symbols are updated in the wlanHERecoveryConfig object. The symbols are only updated if the number of HE-SIG-B symbols indicated in the HE-SIG-A field is set to 15 and all content channels pass the CRC. The number of HE-SIG-B symbols indicated in the HE-SIG-A field are not updated if any HE-SIG-B content channel fails the CRC.

- The user transmission parameters for both SIGB compressed and non-compressed waveforms are inferred from the HE-SIG-B user field [1 Table. 27-25, 27-26].

An estimate of the channel and noise power obtained from the L-LTF is required to decode the HE-SIG-B field.

```matlab
if strcmp(pktFormat,'HE-MU')
    fprintf('Decoding HE-SIG-B...\n');
    if ~cfgRx.SIGBCompression
        fprintf(' Decoding HE-SIG-B common field...\n');
        s = getSIGBLength(cfgRx);
        % Get common field symbols. The start of HE-SIG-B field is known
        rxSym = rx(pktOffset+(ind.HESIGA(2)+(1:s.NumSIGBCommonFieldSamples)),:);
        % Decode HE-SIG-B common field
        [status,cfgRx] = heSIGBCommonFieldDecode(rxSym,preHEChanEst,noiseVar,cfgRx);

        % CRC on HE-SIG-B content channels
        if strcmp(status,'Success')
            fprintf('  HE-SIG-B (common field) CRC pass\n');
        elseif strcmp(status,'ContentChannel1CRCFail')
            fprintf('  ** HE-SIG-B CRC fail for content channel-1\n **');
        elseif strcmp(status,'ContentChannel2CRCFail')
            fprintf('  ** HE-SIG-B CRC fail for content channel-2\n **');
        elseif any(strcmp(status,{'UnknownNumUsersContentChannel1','UnknownNumUsersContentChanne
            error('  ** Unknown packet length, discard packet\n **');
        else
            % Discard the packet if all HE-SIG-B content channels fail
            error('  ** HE-SIG-B CRC fail **');
        end
        % Update field indices as the number of HE-SIG-B symbols are
        % updated
        ind = wlanFieldIndices(cfgRx);
    end

    % Get complete HE-SIG-B field samples
```

```
        rxSIGB = rx(pktOffset+(ind.HESIGB(1):ind.HESIGB(2)),:);
        fprintf(' Decoding HE-SIG-B user field... \n');
        % Decode HE-SIG-B user field
        [failCRC,cfgUsers] = heSIGBUserFieldDecode(rxSIGB,preHEChanEst,noiseVar,cfgRx);

        % CRC on HE-SIG-B users
        if ~all(failCRC)
            fprintf('  HE-SIG-B (user field) CRC pass\n\n');
            numUsers = numel(cfgUsers);
        elseif all(failCRC)
            % Discard the packet if all users fail the CRC
            error('  ** HE-SIG-B CRC fail for all users **');
        else
            fprintf('  ** HE-SIG-B CRC fail for at least one user\n **');
            % Only process users with valid CRC
            numUsers = numel(cfgUsers);
        end

else % HE-SU, HE-EXT-SU
    cfgUsers = {cfgRx};
    numUsers = 1;
end

Decoding HE-SIG-B...
 Decoding HE-SIG-B common field...
  HE-SIG-B (common field) CRC pass
 Decoding HE-SIG-B user field...
  HE-SIG-B (user field) CRC pass
```

HE-Data Decoding

The updated wlanHERecoveryConfig object for each user can then be used to recover the PSDU bits for each user in the HE-Data field.

```
cfgDataRec = trackingRecoveryConfig;
cfgDataRec.PilotTracking = pilotTracking;

fprintf('Decoding HE-Data...\n');
for iu = 1:numUsers
    % Get recovery configuration object for each user
    user = cfgUsers{iu};
    if strcmp(pktFormat,'HE-MU')
        fprintf(' Decoding User:%d, STAID:%d, RUSize:%d\n',iu,user.STAID,user.RUSize);
    else
        fprintf(' Decoding RUSize:%d\n',user.RUSize);
    end

    heInfo = wlanHEOFDMInfo('HE-Data',chanBW,user.GuardInterval,[user.RUSize user.RUIndex]);

    % HE-LTF demodulation and channel estimation
    rxHELTF = rx(pktOffset+(ind.HELTF(1):ind.HELTF(2)),:);
    heltfDemod = wlanHEDemodulate(rxHELTF,'HE-LTF',chanBW,user.GuardInterval, ...
        user.HELTFType,[user.RUSize user.RUIndex]);
    [chanEst,pilotEst] = heLTFChannelEstimate(heltfDemod,user);

    % Number of expected data OFDM symbols
    symLen = heInfo.FFTLength+heInfo.CPLength;
```

```matlab
numOFDMSym = (ind.HEData(2)-ind.HEData(1)+1)/symLen;

% HE-Data demodulation with pilot phase and timing tracking
% Account for extra samples when extracting data field from the packet
% for sample rate offset tracking. Extra samples may be required if the
% receiver clock is significantly faster than the transmitter.
maxSRO = 120; % Parts per million
Ne = ceil(numRxSamples*maxSRO*1e-6); % Number of extra samples
Ne = min(Ne,rxWaveLen-numRxSamples); % Limited to length of waveform
numRxSamplesProcess = numRxSamples+Ne;
rxData = rx(pktOffset+(ind.HEData(1):numRxSamplesProcess),:);
if user.RUSize==26
    % Force CPE only tracking for 26-tone RU as algorithm susceptible
    % to noise
    cfgDataRec.PilotTracking = 'CPE';
else
    cfgDataRec.PilotTracking = pilotTracking;
end
    [demodSym,cpe,peg] = heTrackingOFDMDemodulate(rxData,chanEst,numOFDMSym,user,cfgDataRec)

% Estimate noise power in HE fields
demodPilotSym = demodSym(heInfo.PilotIndices,:,:);
nVarEst = heNoiseEstimate(demodPilotSym,pilotEst,user);

% Equalize
[eqSym,csi] = heEqualizeCombine(demodSym,chanEst,nVarEst,user);

% Discard pilot subcarriers
eqSymUser = eqSym(heInfo.DataIndices,:,:);
csiData = csi(heInfo.DataIndices,:);

% Demap and decode bits
rxPSDU = wlanHEDataBitRecover(eqSymUser,nVarEst,csiData,user,'LDPCDecodingMethod','layered-bp

% Deaggregate the A-MPDU
[mpduList,~,status] = wlanAMPDUDeaggregate(rxPSDU,wlanHESUConfig);
if strcmp(status,'Success')
    fprintf('  A-MPDU deaggregation successful \n');
else
    fprintf('  A-MPDU deaggregation unsuccessful \n');
end

% Decode the list of MPDUs and check the FCS for each MPDU
for i = 1:numel(mpduList)
    [~,~,status] = wlanMPDUDecode(mpduList{i},wlanHESUConfig,'DataFormat','octets');
    if strcmp(status,'Success')
        fprintf('  FCS pass for MPDU:%d\n',i);
    else
        fprintf('  FCS fail for MPDU:%d\n',i);
    end
end

% Plot equalized constellation of the recovered HE data symbols for all
% spatial streams per user
hePlotEQConstellation(eqSymUser,user,ConstellationDiagram,iu,numUsers);

% Measure EVM of HE-Data symbols
release(EVM);
```

```
        EVM.ReferenceConstellation = wlanReferenceSymbols(user);
        rmsEVM = EVM(eqSymUser(:));
        fprintf('  HE-Data EVM:%2.2fdB\n\n',20*log10(rmsEVM/100));

        % Plot EVM per symbol of the recovered HE data symbols
        hePlotEVMPerSymbol(eqSymUser,user,EVMPerSymbol,iu,numUsers);

        % Plot EVM per subcarrier of the recovered HE data symbols
        hePlotEVMPerSubcarrier(eqSymUser,user,EVMPerSubcarrier,iu,numUsers);
end

Decoding HE-Data...
 Decoding User:1, STAID:1, RUSize:52
  A-MPDU deaggregation successful
  FCS pass for MPDU:1
  HE-Data EVM:-28.61dB

 Decoding User:2, STAID:2, RUSize:52
  A-MPDU deaggregation successful
  FCS pass for MPDU:1
  HE-Data EVM:-39.94dB

 Decoding User:3, STAID:3, RUSize:106
  A-MPDU deaggregation successful
  FCS pass for MPDU:1
  HE-Data EVM:-28.22dB

 Decoding User:4, STAID:4, RUSize:106
  A-MPDU deaggregation successful
  FCS pass for MPDU:1
  HE-Data EVM:-31.44dB
```
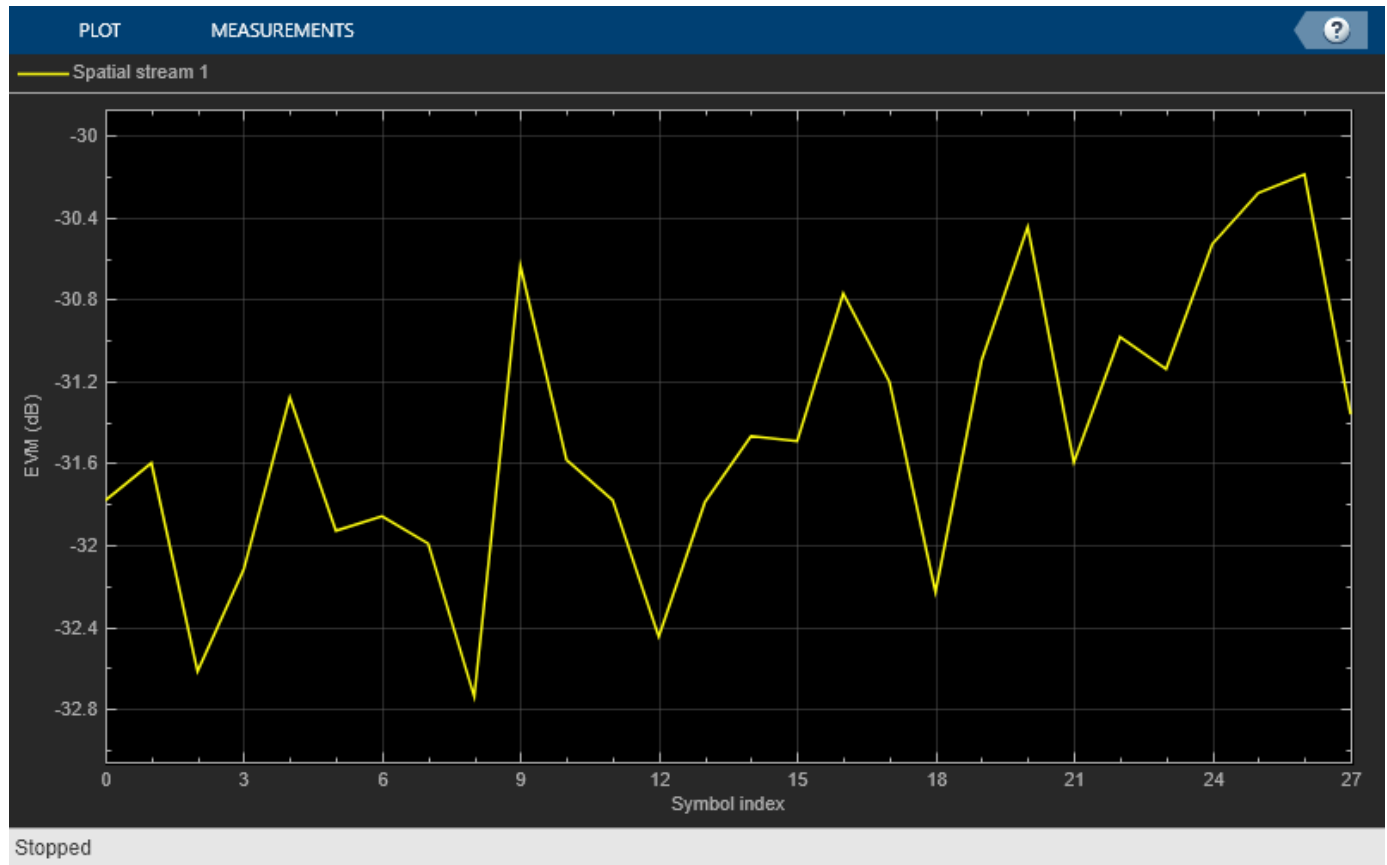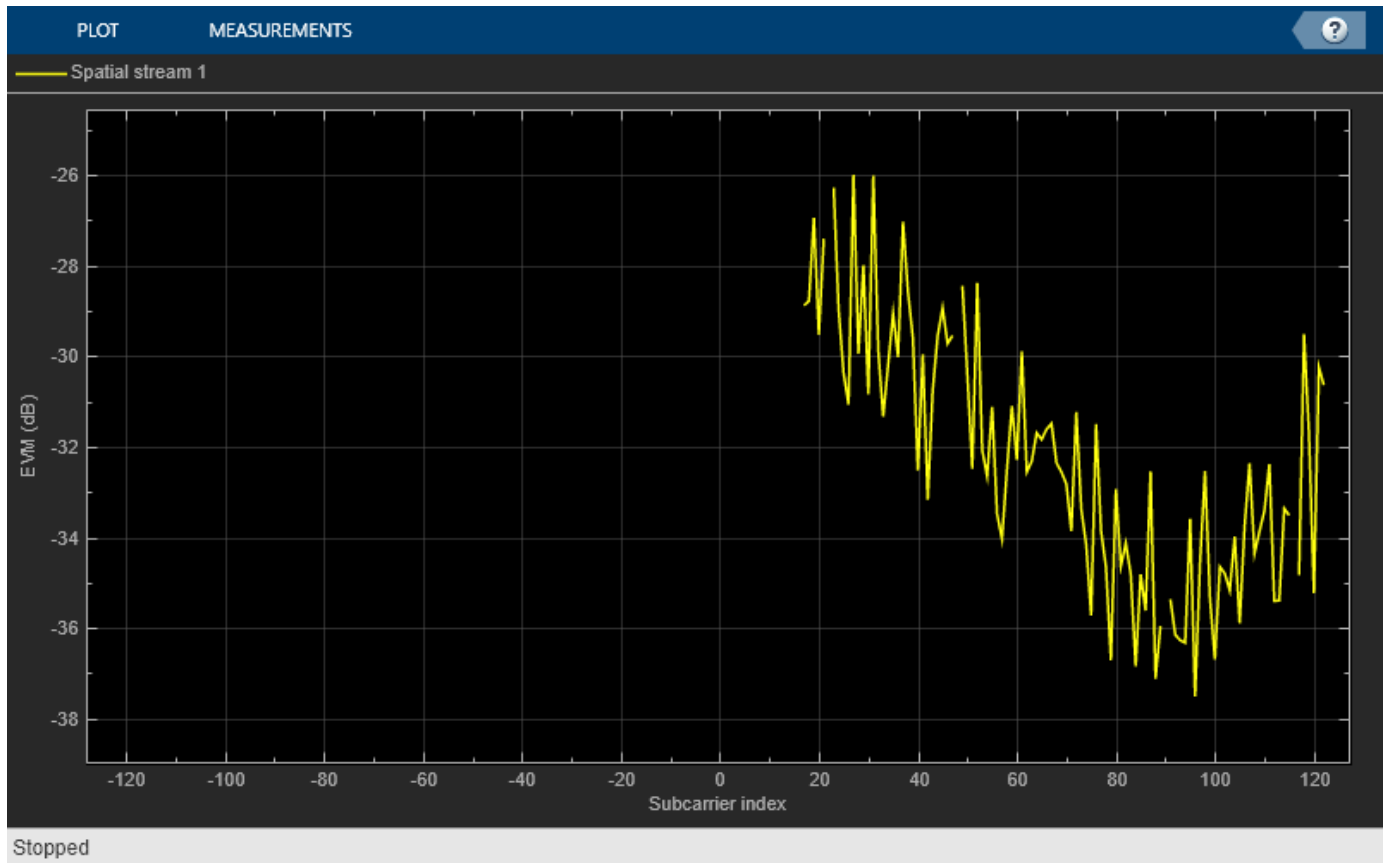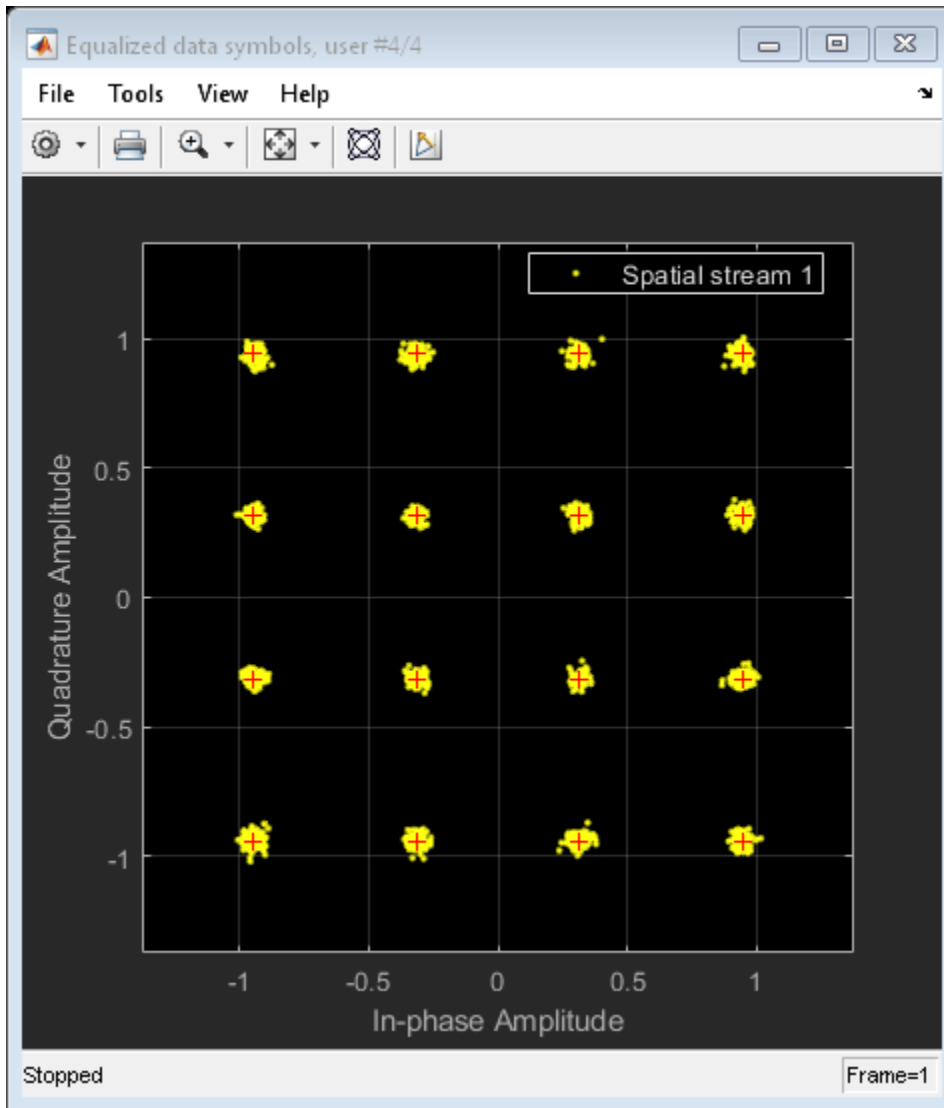
Appendix

This example uses the following helper functions:

- heCommonPhaseErrorTracking.m
- heCPECorrection.m
- heEqualizeCombine.m
- helperFrequencyOffset.m
- helperNoiseEstimate.m
- helperPlotPositions.m
- helperSymbolEqualize.m
- heLTFChannelEstimate.m
- heNoiseEstimate.m

- hePlotEQConstellation.m
- hePlotEVMPerSubcarrier.m
- hePlotEVMPerSymbol.m
- heSigRecSetupPlots.m
- heSigRecGenerateWaveform.m
- heSIGBCommonFieldDecode.m
- heSIGBMergeSubchannels.m
- heSIGBUserFieldDecode.m
- heTrackingOFDMDemodulate.m
- preHEChannelEstimate.m
- preHESymbolEqualize.m
- trackingRecoveryConfig.m

Selected Bibliography

1   IEEE P802.11ax™/D4.1 Draft 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 6: Enhancements for High Efficiency WLAN.

2   IEEE Std 802.11™-2016 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.

Recovery Procedure for an 802.11ac Packet

This example shows how to detect a packet and decode payload bits in a received IEEE® 802.11ac™ VHT waveform. The receiver recovers the packet format parameters from the preamble fields to decode the data.

Introduction

In a single-user 802.11ac packet the transmission parameters are signaled to the receiver using the L-SIG and VHT-SIG-A preamble fields [1]:

- The L-SIG field contains information to allow the receiver to determine the transmission time of a packet.
- The VHT-SIG-A field contains the transmission parameters including the modulation and coding scheme, number of space-time streams and channel coding.

In this example we detect and decode a packet within a generated waveform containing a valid MAC frame with frame check sequence (FCS). All transmission parameters apart from the channel bandwidth are assumed unknown and are therefore retrieved from the decoded L-SIG and VHT-SIG-A preamble fields in each packet. The retrieved transmission configuration is used to decode the VHT-SIG-B and VHT Data fields. Additionally the following analysis is performed:

- The waveform of the detected packet is displayed.
- The spectrum of the detected packet is displayed.
- The constellation of the equalized data symbols per spatial stream is displayed.
- The error vector magnitude (EVM) of each field is measured.

Waveform Transmission

In this example an 802.11ac VHT single-user waveform is generated locally but a captured waveform could be used. MATLAB® can be used to acquire I/Q data from a wide range of instruments using the Instrument Control Toolbox™ and software defined radio platforms.

The locally generated waveform is impaired by a 3x3 TGac fading channel, additive white Gaussian noise, and carrier frequency offset. To generate a waveform locally we configure a VHT packet format configuration object. Note that the VHT packet configuration object is used at the transmitter side only. The receiver will dynamically formulate another VHT configuration object when the packet is decoded. The helper function vhtSigRecGenerateWaveform generates the impaired waveform locally. The processing steps within the helper function are:

- A valid MAC frame is generated and encoded into a VHT waveform.
- The waveform is passed through a TGac fading channel model.
- Carrier frequency offset is added to the waveform.
- Additive white Gaussian noise is added to the waveform.

```
% VHT link parameters
cfgVHTTx = wlanVHTConfig( ...
    'ChannelBandwidth',    'CBW80', ...
    'NumTransmitAntennas', 3, ...
    'NumSpaceTimeStreams', 2, ...
    'SpatialMapping',      'Hadamard', ...
    'STBC',                true, ...
```

```
    'MCS',                  5, ...
    'GuardInterval',        'Long', ...
    'APEPLength',           1052);

% Propagation channel
numRx = 3;                      % Number of receive antennas
delayProfile = 'Model-C';   % TGac channel delay profile

% Impairments
noisePower = -30;   % Noise power to apply in dBW
cfo = 62e3;         % Carrier frequency offset (Hz)

% Generated waveform parameters
numTxPkt = 1;       % Number of transmitted packets
idleTime = 20e-6;   % Idle time before and after each packet

% Generate waveform
rx = vhtSigRecGenerateWaveform(cfgVHTTx, numRx, ...
    delayProfile, noisePower, cfo, numTxPkt, idleTime);
```

Packet Recovery

The signal to process is stored in the variable rx. The processing steps to recover a packet are:

- The packet is detected and synchronized.
- The format of the packet is detected.
- The L-SIG field is extracted and its information bits are recovered to determine the length of the packet in microseconds.
- The VHT-SIG-A field is extracted and its information bits are recovered.
- The packet format parameters are retrieved from the decoded L-SIG and VHT-SIG-A bits.
- The VHT-LTF field is extracted to perform MIMO channel estimation for decoding the VHT-SIG-B and VHT Data fields.
- The VHT-SIG-B field is extracted and its information bits recovered.
- The VHT-Data field is extracted and the PSDU and VHT-SIG-B CRC bits recovered using the retrieved packet parameters.

The start and end indices for some preamble fields depend on the channel bandwidth, but are independent of all other transmission parameters. These indices are calculated using a default transmission configuration object with the known bandwidth.

```
cfgVHTRx = wlanVHTConfig('ChannelBandwidth', cfgVHTTx.ChannelBandwidth);
idxLSTF = wlanFieldIndices(cfgVHTRx, 'L-STF');
idxLLTF = wlanFieldIndices(cfgVHTRx, 'L-LTF');
idxLSIG = wlanFieldIndices(cfgVHTRx, 'L-SIG');
idxSIGA = wlanFieldIndices(cfgVHTRx, 'VHT-SIG-A');
```

The following code configures objects and variables for processing.

```
chanBW = cfgVHTTx.ChannelBandwidth;
sr = wlanSampleRate(cfgVHTTx);

% Setup plots for example
[spectrumAnalyzer, timeScope, constellationDiagram] = vhtSigRecSetupPlots(sr);
```

4-31

```
% Minimum packet length is 10 OFDM symbols
lstfLen = double(idxLSTF(2)); % Number of samples in L-STF
minPktLen = lstfLen*5;

rxWaveLen = size(rx, 1);
```

Front-End Processing

The front-end processing consists of packet detection, coarse carrier frequency offset correction, timing synchronization and fine carrier frequency offset correction. A `while` loop is used to detect and synchronize a packet within the received waveform. The sample offset `searchOffset` is used to index elements within the array `rx` to detect a packet. The first packet within `rx` is detected and processed. If the synchronization fails for the detected packet, the sample index offset `searchOffset` is incremented to move beyond the processed packet in `rx`. This is repeated until a packet has been successfully detected and synchronized.

```
searchOffset = 0; % Offset from start of waveform in samples
while (searchOffset + minPktLen) <= rxWaveLen
    % Packet detection
    pktOffset = wlanPacketDetect(rx, chanBW, searchOffset);

    % Adjust packet offset
    pktOffset = searchOffset + pktOffset;
    if isempty(pktOffset) || (pktOffset + idxLSIG(2) > rxWaveLen)
        error('** No packet detected **');
    end

    % Coarse frequency offset estimation using L-STF
    LSTF = rx(pktOffset + (idxLSTF(1):idxLSTF(2)), :);
    coarseFreqOffset = wlanCoarseCFOEstimate(LSTF, chanBW);

    % Coarse frequency offset compensation
    rx = helperFrequencyOffset(rx,sr,-coarseFreqOffset);

    % Symbol timing synchronization
    LLTFSearchBuffer = rx(pktOffset+(idxLSTF(1):idxLSIG(2)),:);
    pktOffset = pktOffset+wlanSymbolTimingEstimate(LLTFSearchBuffer,chanBW);
    if (pktOffset + minPktLen) > rxWaveLen
        fprintf('** Not enough samples to recover packet **\n\n');
        break;
    end

    % Timing synchronization complete: packet detected
    fprintf('Packet detected at index %d\n\n', pktOffset + 1);

    % Fine frequency offset estimation using L-LTF
    LLTF = rx(pktOffset + (idxLLTF(1):idxLLTF(2)), :);
    fineFreqOffset = wlanFineCFOEstimate(LLTF, chanBW);

    % Fine frequency offset compensation
    rx = helperFrequencyOffset(rx, sr, -fineFreqOffset);

    % Display estimated carrier frequency offset
    cfoCorrection = coarseFreqOffset + fineFreqOffset; % Total CFO
    fprintf('Estimated CFO: %5.1f Hz\n\n', cfoCorrection);
```

```
        break; % Front-end processing complete, stop searching for a packet
    end
```

```
Packet detected at index 1600
```

```
Estimated CFO: 61954.3 Hz
```

Format Detection

The format of the packet is detected using the three OFDM symbols immediately following the L-LTF. An estimate of the channel and noise power obtained from the L-LTF is required.

```
% Channel estimation using L-LTF
LLTF = rx(pktOffset + (idxLLTF(1):idxLLTF(2)), :);
demodLLTF = wlanLLTFDemodulate(LLTF, chanBW);
chanEstLLTF = wlanLLTFChannelEstimate(demodLLTF, chanBW);

% Estimate noise power in non-HT fields
noiseVarNonHT = helperNoiseEstimate(demodLLTF);

% Detect the format of the packet
fmt = wlanFormatDetect(rx(pktOffset + (idxLSIG(1):idxSIGA(2)), :), ...
    chanEstLLTF, noiseVarNonHT, chanBW);
disp([fmt ' format detected']);
if ~strcmp(fmt,'VHT')
    error('** A format other than VHT has been detected **');
end
```

```
VHT format detected
```

L-SIG Decoding

In a VHT transmission the L-SIG field is used to determine the receive time, or RXTIME, of the packet. RXTIME is calculated using the field bits of the L-SIG payload [1 Eq. 22-105]. The number of samples which contain the packet within `rx` can then be calculated. The L-SIG payload is decoded using an estimate of the channel and noise power obtained from the L-LTF.

```
% Recover L-SIG field bits
disp('Decoding L-SIG... ');
[rxLSIGBits, failCheck, eqLSIGSym] = wlanLSIGRecover(rx(pktOffset + (idxLSIG(1):idxLSIG(2)), :),
    chanEstLLTF, noiseVarNonHT, chanBW);

if failCheck % Skip L-STF length of samples and continue searching
    disp('** L-SIG check fail **');
else
    disp('L-SIG check pass');
end

% Measure EVM of L-SIG symbol
EVM = comm.EVM;
EVM.ReferenceSignalSource = 'Estimated from reference constellation';
EVM.ReferenceConstellation = wlanReferenceSymbols('BPSK');
rmsEVM = EVM(eqLSIGSym);
fprintf('L-SIG EVM: %2.2f%% RMS\n', rmsEVM);

% Calculate the receive time and corresponding number of samples in the
% packet
```

```
lengthBits = rxLSIGBits(6:17).';
RXTime = ceil((bi2de(double(lengthBits)) + 3)/3) * 4 + 20; % us
numRxSamples = RXTime * 1e-6 * sr; % Number of samples in receive time

fprintf('RXTIME: %dus\n', RXTime);
fprintf('Number of samples in packet: %d\n\n', numRxSamples);

Decoding L-SIG...
L-SIG check pass
L-SIG EVM: 1.83% RMS
RXTIME: 84us
Number of samples in packet: 6720
```

The waveform and spectrum of the detected packet within `rx` are displayed for the calculated RXTIME and corresponding number of samples.

```
sampleOffset = max((-lstfLen + pktOffset), 1); % First index to plot
sampleSpan = numRxSamples + 2*lstfLen;         % Number of samples to plot
% Plot as much of the packet (and extra samples) as we can
plotIdx = sampleOffset:min(sampleOffset + sampleSpan, rxWaveLen);

% Configure timeScope to display the packet
timeScope.TimeSpan = sampleSpan/sr;
timeScope.TimeDisplayOffset = sampleOffset/sr;
timeScope.YLimits = [0 max(abs(rx(:)))];
timeScope(abs(rx(plotIdx ,:)));

% Display the spectrum of the detected packet
spectrumAnalyzer(rx(pktOffset + (1:numRxSamples), :));
```
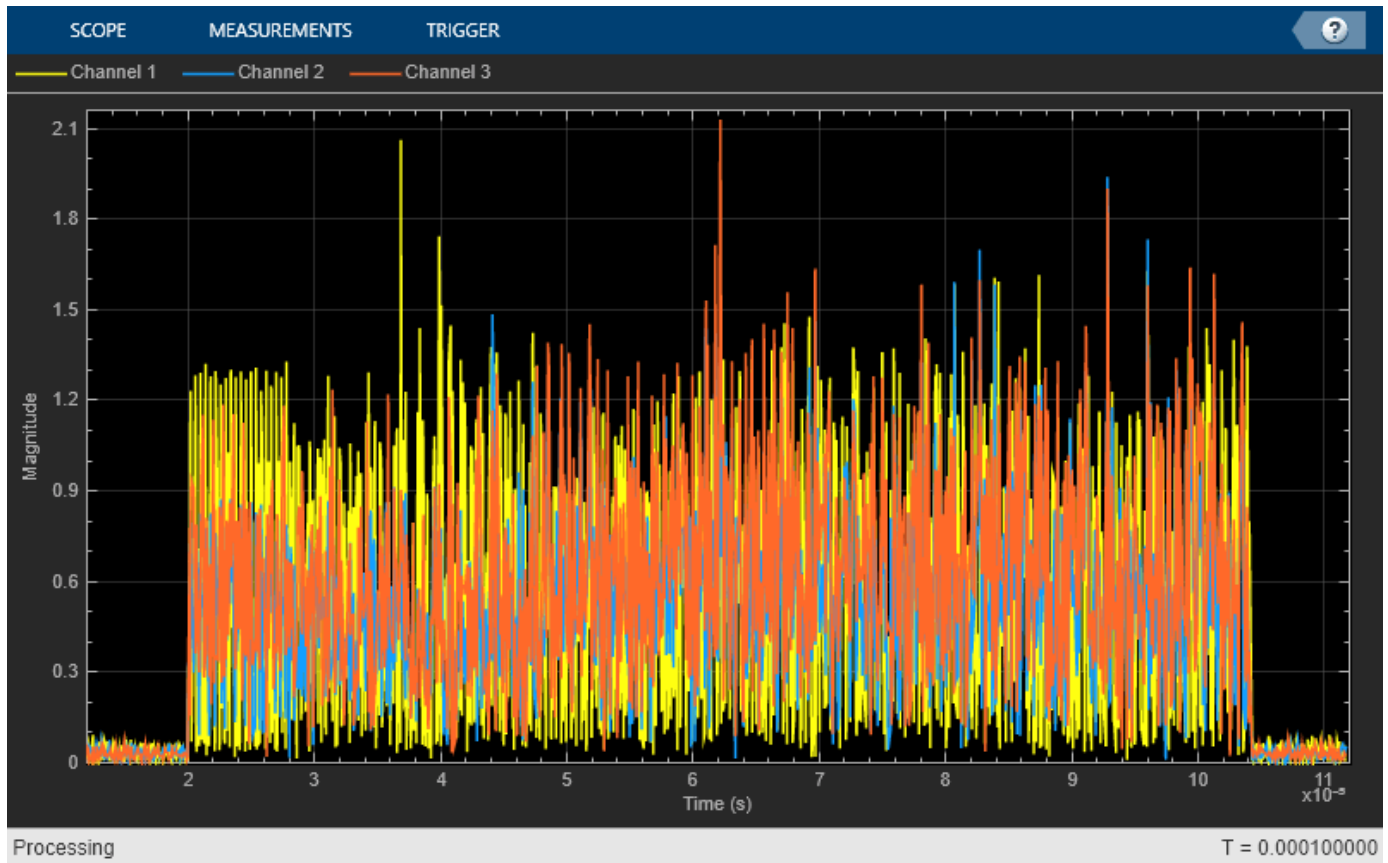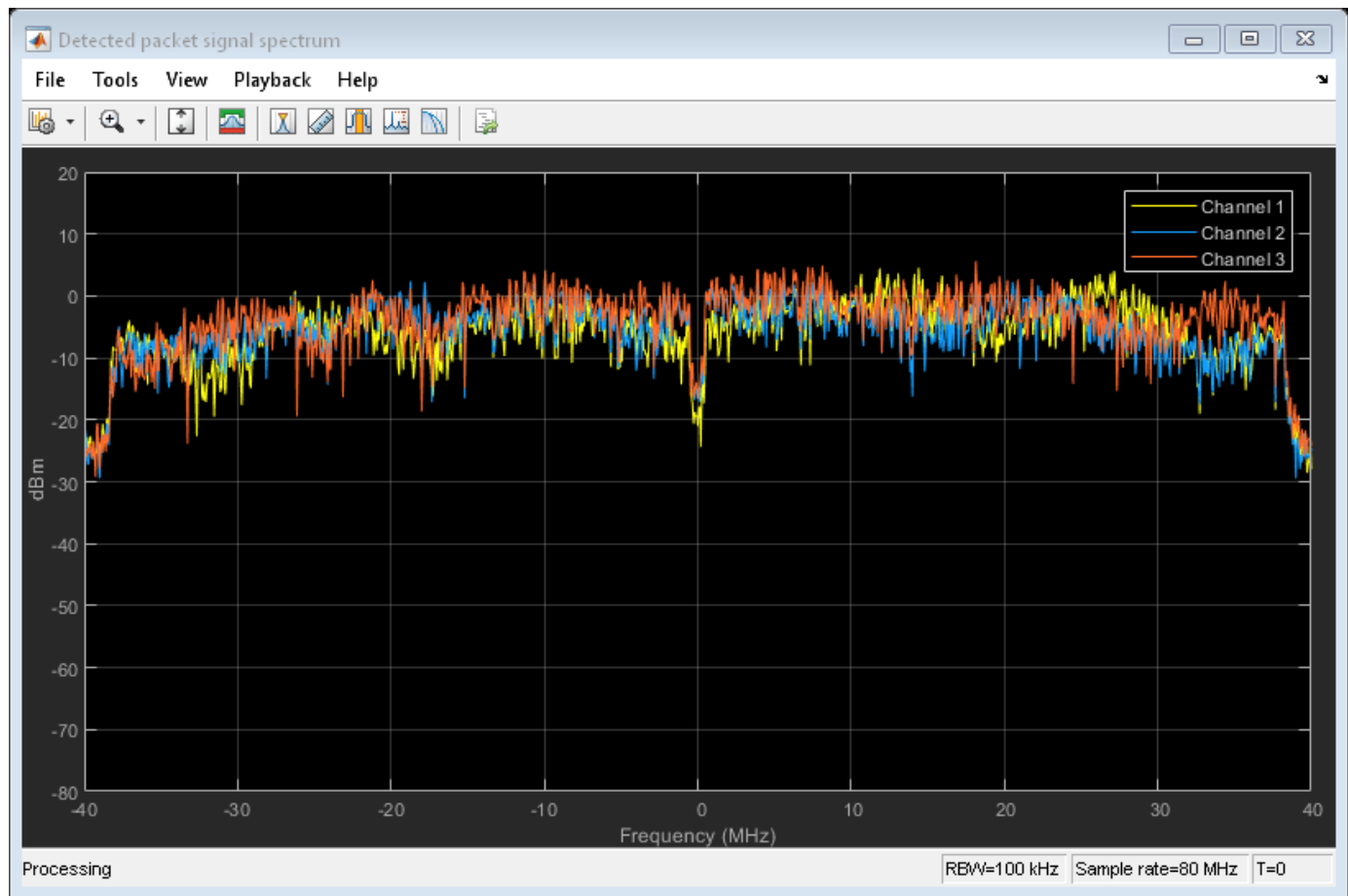
VHT-SIG-A Decoding

The VHT-SIG-A field contains the transmission configuration of the packet. The VHT-SIG-A bits are recovered using the channel and noise power estimates obtained from the L-LTF.

```
% Recover VHT-SIG-A field bits
disp('Decoding VHT-SIG-A... ');
[rxSIGABits, failCRC, eqSIGASym] = wlanVHTSIGARecover(rx(pktOffset + (idxSIGA(1):idxSIGA(2)), :)
    chanEstLLTF, noiseVarNonHT, chanBW);

if failCRC
    disp('** VHT-SIG-A CRC fail **');
else
    disp('VHT-SIG-A CRC pass');
end

% Measure EVM of VHT-SIG-A symbols for BPSK and QBPSK modulation schemes
release(EVM);
EVM.ReferenceConstellation = wlanReferenceSymbols('BPSK');
rmsEVMSym1 = EVM(eqSIGASym(:,1));
release(EVM);
EVM.ReferenceConstellation = wlanReferenceSymbols('QBPSK');
rmsEVMSym2 = EVM(eqSIGASym(:,2));
fprintf('VHT-SIG-A EVM: %2.2f%% RMS\n', mean([rmsEVMSym1 rmsEVMSym2]));
```

```
Decoding VHT-SIG-A...
VHT-SIG-A CRC pass
VHT-SIG-A EVM: 2.06% RMS
```

The helper function helperVHTConfigRecover returns a VHT format configuration object, `cfgVHTRx`, based on recovered VHT-SIG-A and L-SIG bits. Properties that are not required to decode the waveform are set to default values for a `wlanVHTConfig` object and therefore may differ from the value in `cfgVHTTx`. Examples of such properties include `NumTransmitAntennas` and `SpatialMapping`.

```
% Create a VHT format configuration object by retrieving packet parameters
% from the decoded L-SIG and VHT-SIG-A bits
cfgVHTRx = helperVHTConfigRecover(rxLSIGBits, rxSIGABits);

% Display the transmission configuration obtained from VHT-SIG-A
vhtSigRecDisplaySIGAInfo(cfgVHTRx);

  Decoded VHT-SIG-A contents:
       ChannelBandwidth: 'CBW80'
    NumSpaceTimeStreams: 2
                   STBC: 1
                    MCS: 5
          ChannelCoding: {'BCC'}
          GuardInterval: 'Long'
                GroupID: 63
              PartialAID: 275
             Beamforming: 0
              PSDULength: 1167
```

The information provided by VHT-SIG-A allows the location of subsequent fields within the received waveform to be calculated.

```
% Obtain starting and ending indices for VHT-LTF and VHT-Data fields
% using retrieved packet parameters
idxVHTLTF  = wlanFieldIndices(cfgVHTRx, 'VHT-LTF');
idxVHTSIGB = wlanFieldIndices(cfgVHTRx, 'VHT-SIG-B');
idxVHTData = wlanFieldIndices(cfgVHTRx, 'VHT-Data');

% Warn if waveform does not contain whole packet
if (pktOffset + double(idxVHTData(2))) > rxWaveLen
    fprintf('** Not enough samples to recover entire packet **\n\n');
end
```

VHT-SIG-B Decoding

The primary use of VHT-SIG-B is for signaling user information in a multi-user packet. In a single-user packet the VHT-SIG-B carries the length of the packet which can also be calculated using the L-SIG and VHT-SIG-A (which is demonstrated in the sections above). Despite not being required to decode a single-user packet, the VHT-SIG-B is recovered below and the bits interpreted. The VHT-SIG-B symbols are demodulated using a MIMO channel estimate obtained from the VHT-LTF. Note the CRC for VHT-SIG-B is carried in the VHT Data field.

```
% Estimate MIMO channel using VHT-LTF and retrieved packet parameters
demodVHTLTF = wlanVHTLTFDemodulate(rx(pktOffset + (idxVHTLTF(1):idxVHTLTF(2)), :), cfgVHTRx);
chanEstVHTLTF = wlanVHTLTFChannelEstimate(demodVHTLTF, cfgVHTRx);

% Estimate noise power in VHT-SIG-B fields
```

```
noiseVarVHT = helperNoiseEstimate(demodLLTF, chanBW, cfgVHTRx.NumSpaceTimeStreams);

% VHT-SIG-B Recover
disp('Decoding VHT-SIG-B...');
[rxSIGBBits, eqSIGBSym] = wlanVHTSIGBRecover(rx(pktOffset + (idxVHTSIGB(1):idxVHTSIGB(2)),:), ..
    chanEstVHTLTF, noiseVarVHT, chanBW);

% Measure EVM of VHT-SIG-B symbol
release(EVM);
EVM.ReferenceConstellation = wlanReferenceSymbols('BPSK');
rmsEVM = EVM(eqSIGBSym);
fprintf('VHT-SIG-B EVM: %2.2f%% RMS\n', rmsEVM);

% Interpret VHT-SIG-B bits to recover the APEP length (rounded up to a
% multiple of four bytes) and generate reference CRC bits
[refSIGBCRC, sigbAPEPLength] = helperInterpretSIGB(rxSIGBBits, chanBW, true);
disp('Decoded VHT-SIG-B contents: ');
fprintf('  APEP Length (rounded up to 4 byte multiple): %d bytes\n\n', sigbAPEPLength);

Decoding VHT-SIG-B...
VHT-SIG-B EVM: 5.21% RMS
Decoded VHT-SIG-B contents:
  APEP Length (rounded up to 4 byte multiple): 1052 bytes
```

VHT Data Decoding

The reconstructed VHT configuration object can then be used to recover the VHT Data field. This includes the VHT-SIG-B CRC bits and PSDU.

The recovered VHT data symbols can then be analyzed as required. In this example the equalized constellation of the recovered VHT data symbols per spatial stream are displayed.

```
% Get single stream channel estimate
chanEstSSPilots = vhtSingleStreamChannelEstimate(demodVHTLTF, cfgVHTRx);

% Extract VHT Data samples from the waveform
vhtdata = rx(pktOffset + (idxVHTData(1):idxVHTData(2)), :);

% Estimate the noise power in VHT data field
noiseVarVHT = vhtNoiseEstimate(vhtdata, chanEstSSPilots, cfgVHTRx);

% Recover PSDU bits using retrieved packet parameters and channel
% estimates from VHT-LTF
disp('Decoding VHT Data field...');
[rxPSDU, rxSIGBCRC, eqDataSym] = wlanVHTDataRecover(vhtdata, chanEstVHTLTF, noiseVarVHT, cfgVHTRx
% Plot equalized constellation for each spatial stream
refConst = wlanReferenceSymbols(cfgVHTRx);
[Nsd, Nsym, Nss] = size(eqDataSym);
eqDataSymPerSS = reshape(eqDataSym, Nsd*Nsym, Nss);
for iss = 1:Nss
    constellationDiagram{iss}.ReferenceConstellation = refConst;
    constellationDiagram{iss}(eqDataSymPerSS(:, iss));
end

% Measure EVM of VHT-Data symbols
release(EVM);
```
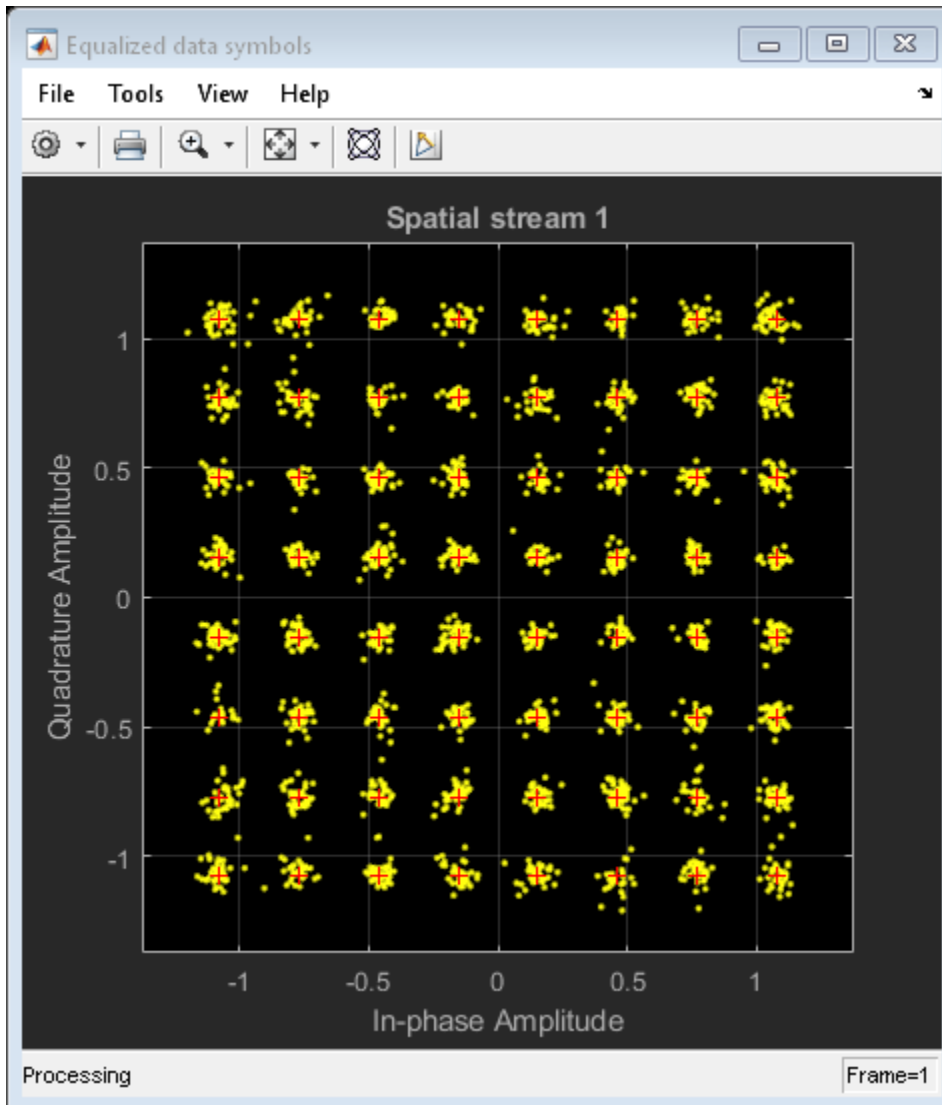
```
EVM.ReferenceConstellation = refConst;
rmsEVM = EVM(eqDataSym(:));
fprintf('VHT-Data EVM: %2.2f%% RMS\n', rmsEVM);

Decoding VHT Data field...
VHT-Data EVM: 4.68% RMS
```


The CRC bits for VHT-SIG-B recovered in VHT Data are then compared to the locally generated reference to determine whether the VHT-SIG-B and VHT data service bits have been recovered successfully.

```
% Test VHT-SIG-B CRC from service bits within VHT Data against
% reference calculated with VHT-SIG-B bits
if ~isequal(refSIGBCRC, rxSIGBCRC)
    disp('** VHT-SIG-B CRC fail **');
else
    disp('VHT-SIG-B CRC pass');
end
```

```
VHT-SIG-B CRC pass
```

The FCS in the MAC frames can be validated using `wlanMPDUDecode`. As a VHT format frame is recovered, the PSDU contains an A-MPDU. The MPDUs are extracted from the A-MPDU using `wlanAMPDUDeaggregate`.

```
mpduList = wlanAMPDUDeaggregate(rxPSDU, cfgVHTRx);
fprintf('Number of MPDUs present in the A-MPDU: %d\n', numel(mpduList));
```

```
Number of MPDUs present in the A-MPDU: 1
```

The `mpduList` contains the de-aggregated list of MPDUs. Each MPDU in the list is passed to `wlanMPDUDecode` which validates the FCS and decodes the MPDU.

```
for i = 1:numel(mpduList)
    [macCfg, payload, decodeStatus] = wlanMPDUDecode(mpduList{i}, cfgVHTRx, ...
                                                  'DataFormat', 'octets');
    if strcmp(decodeStatus, 'FCSFailed')
        fprintf('** FCS failed for MPDU-%d **\n', i);
    else
        fprintf('FCS passed for MPDU-%d\n', i);
    end
end
```

```
FCS passed for MPDU-1
```

Appendix

This example uses the following helper functions:

- helperFrequencyOffset.m
- helperInterpretSIGB.m
- helperNoiseEstimate.m
- helperVHTConfigRecover.m
- vhtNoiseEstimate.m
- vhtSigRecDisplaySIGAInfo.m
- vhtSigRecGenerateWaveform.m
- vhtSigRecSetupPlots.m
- vhtSingleStreamChannelEstimate.m

Selected Bibliography

1   IEEE Std 802.11ac™-2013 IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz.

802.11 OFDM Beacon Receiver with Captured Data

This example shows a receiver design that is able to recover 802.11™ OFDM beacon packets in non-HT format transmitted over the air from commercial 802.11 hardware. Beacon packets are typically transmitted in non-HT format, even for HT [1], VHT [1], and/or HE [2] capable hardware. Packet information such as SSID is printed to the command-line during recovery.

Introduction

This example illustrates the use of WLAN Toolbox™ to recover real-world signals. It demonstrates a receiver design including synchronization, transmission configuration recovery, and payload decoding for non-HT packets. The example recovers beacon packets from a file containing a captured baseband waveform.

Beacon Packet Recovery

The following steps happen sequentially to recover one non-HT packet:

- Packet Detection: First a packet must be detected before any processing begins. This is accomplished by auto-correlating input symbols. Since the front of each 802.11 OFDM packet contains a repetitive structure called the L-STF, peaks will occur in the correlation when this packet is present. The L-STF field is then extracted and used for coarse frequency estimation.

- Symbol Timing: Once a packet has been detected, future symbols will be collected and cross-correlated to locate the L-LTF. The resulting correlation peaks provide an accurate timing estimate. Once the full L-LTF is located, it is extracted and used for channel estimation and fine frequency estimation.

- L-SIG Decoding: The first OFDM symbol after the L-LTF is the L-SIG field. This field must be recovered and decoded to determine the modulation, code rate, and length of the following payload. The information is used to capture the correct amount of data after the L-SIG for the complete payload and to decode that information.

- Payload Decoding: All OFDM symbols after the L-SIG are buffered to a length determined by the L-SIG field. After all the symbols have been captured they are demodulated and decoded into their source bits. The source bits are then evaluated. This evaluation includes frame check sequence (FCS) validation and extraction of the header and body. If the packet is of subtype beacon, summary information such as SSID will be printed for the recovered packet.

Once a full packet is received or any failures occur during the processing chain, the receiver will return to packet detection to search for more packets. This process is repeated for the duration of the signal.

Streaming Process on Captured Data

In this example an off-the-air capture is processed to recover beacon frames. A Wi-Fi® signal was captured using an RF interface with one receive antenna at a sampling rate of 20 Msps. The captured waveform is stored in a binary baseband file. The file was created using `comm.BasebandFileWriter`.

The captured waveform is processed in a streaming fashion. A block of samples is pulled in for processing in each iteration. As many valid packets are retrieved as possible. `comm.BasebandFileReader` is used to read blocks of samples from the binary baseband file.

```
% Create an object to stream the data from the file
basebandReader = comm.BasebandFileReader( ...
```

```
    'Filename',        'nonHTBeaconRxData.bb', ...
    'SamplesPerFrame', 80); % Number of samples in 1 OFDM symbol at 20 MHz
```

The center frequency, sample rate and number of channels in the captured waveform are provided by the comm.BasebandFileReader object.

```
disp(['Center frequency: ' num2str(basebandReader.CenterFrequency/1e6) ' MHz'])
disp(['Sample rate: ' num2str(basebandReader.SampleRate/1e6) ' Msps'])
disp(['Number of receive antennas: ' num2str(basebandReader.NumChannels) newline])
```

```
Center frequency: 5785 MHz
Sample rate: 20 Msps
Number of receive antennas: 1
```

A nonHTFrontEnd object performs front-end processing and L-SIG decoding. The object is configured with a channel bandwidth of 20 MHz to process non-HT packets. Only one receive antenna is supported.

```
rxFrontEnd = nonHTFrontEnd('ChannelBandwidth', 'CBW20');
```

A while loop is used to process blocks of samples and recover beacon packets until no more data is available in the baseband file. In each iteration of the loop a block of samples is read from the baseband file and is processed by `rxFrontEnd`. `rxFrontEnd` performs front-end processing and buffers samples until a packet has been detected and the payload received. When `payloadFull` is true, the full payload has been buffered and `rxFrontEnd` returns variables to allow the data within the packet to be recovered:

- `cfgNonHT` contains the recovered packet parameters from L-SIG.
- `rxNonHTData` is the time-domain non-HT data field signal.
- `chanEst` contains the channel estimates obtained from the L-LTF.
- `noiseVar` is the fixed noise variance value.

The packet payload bits are recovered from the non-HT data field samples using `wlanNonHTDataRecover`. The bits are then validated and decoded by `wlanMPDUDecode` to recover the MAC frame parameters. `wlanMPDUDecode` returns the following outputs that determine whether the received packet passed FCS check and whether the received packet is a beacon frame.

- `mpduCfg` is an object of type `wlanMACFrameConfig` containing the recovered MAC frame parameters from the beacon frame.
- `status` is an enumeration of type "status" which is returned as 'Success' when MPDU passes FCS check and returned as 'FCSFailed' when the MPDU fails FCS check.

If a valid beacon is detected, the decoded SSID is displayed.

```
% Symbol-by-symbol streaming process
numValidPackets = 0;
while ~isDone(basebandReader)
    % Pull in one OFDM symbol, i.e. 80 samples
    data = basebandReader();

    % Perform front-end processing and payload buffering
    [payloadFull, cfgNonHT, rxNonHTData, chanEst, noiseVar] = ...
        rxFrontEnd(data);
```

```matlab
    if payloadFull
        % Recover payload bits with zero-forcing equalization
        recBits = wlanNonHTDataRecover(rxNonHTData, chanEst, ...
            noiseVar, cfgNonHT, 'EqualizationMethod', 'ZF');

        % Decode and evaluate recovered bits
        [mpduCfg, ~, status] = wlanMPDUDecode(recBits, cfgNonHT);
        if strcmp(status, 'Success') && strcmp(mpduCfg.FrameType, 'Beacon')
            frameBody = mpduCfg.ManagementConfig; % Display SSID
            disp(['SSID: ', frameBody.SSID])
            numValidPackets = numValidPackets + 1;
        end
    end
end

disp([num2str(numValidPackets), ' Valid Beacon Packets Found'])

release(basebandReader);
release(rxFrontEnd);
```

```
SSID: MathWorks-SDR
SSID: MathWorks-SDR
SSID: MathWorks-SDR
3 Valid Beacon Packets Found
```

Further Exploration

See "802.11 OFDM Beacon Receiver with USRP® Hardware" on page 10-3 for an example of processing live signals with USRP.

Appendix

This example uses the following helper functions and objects:

- nonHTFrontEnd.m

Selected Bibliography

1   IEEE® Std 802.11™-2016 IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

2   IEEE P802.11ax™/D4.1 Draft 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 6: Enhancements for High Efficiency WLAN.

Joint Sampling Rate and Carrier Frequency Offset Tracking

This example demonstrates joint sampling rate and carrier frequency offset tracking in a WLAN receiver.

Introduction

In a WLAN radio a single oscillator is typically used to derive clocks for sampling and modulation. The oscillators in the transmitter and receiver radios do not run at the exact same frequency. Due to this mismatch, the sampling instants at the receiver shift relative to the transmitter. Therefore, a sample rate offset (SRO) exists between receiver and transmitter. Similarly, a carrier frequency offset (CFO) exists between the receiver and transmitter due to mismatched carrier frequencies. The inclusion of pilot subcarriers in the IEEE® 802.11™ standard allows for tracking and correction of SRO and CFO impairments.

In OFDM systems SRO manifests itself as a subcarrier and symbol dependent phase rotation and inter-carrier interference (ICI) [1]. When the SRO is large, and a packet is long, subcarriers far away from DC will experience a substantial impairment. CFO manifests itself as ICI and a symbol dependent phase rotation common to all subcarriers. The phase rotation on subcarriers from one OFDM symbol to the next due to these impairments is illustrated below. Φ_k is the phase error, k is the subcarrier index, K is the number of subcarriers. ζ is the SRO, Δf is the carrier frequency offset, T_u is the period of the symbol, δ is the phase error gradient (PEG), and ω is the common phase error (CPE). The PEG and CPE can be used to estimate SRO and residual CFO.
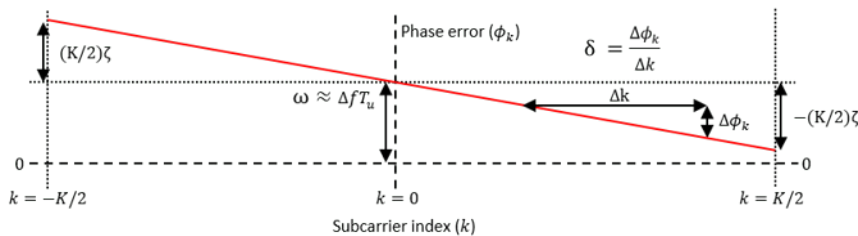
In this example, an IEEE 802.11ac™ VHT waveform is generated with fixed SRO and CFO impairments [2]. The impaired waveform is synchronized and the data portion demodulated and decoded with and without joint timing and phase tracking to correct for SRO and CFO. The equalized constellation with and without tracking is shown to demonstrate the effectiveness of tracking.

Generate a Baseband Waveform

A VHT configuration object is created to parameterize the transmission. Because the impairments are easier to visualize with fewer OFDM symbols, a data payload with only 500 bytes is transmitted using 16-QAM modulation.

```
% Create a VHT configuration
cfgVHT = wlanVHTConfig;
cfgVHT.ChannelBandwidth = 'CBW20';
cfgVHT.NumTransmitAntennas = 1;
cfgVHT.NumSpaceTimeStreams = 1;
cfgVHT.MCS = 4;              % 16-QAM and 3/4 coding rate
cfgVHT.APEPLength = 500; % Bytes

% Create a random PSDU
```

```
s = rng(10); % Seed the random number generator
psdu = randi([0 1],cfgVHT.PSDULength*8,1,'int8');

% Generate a VHT packet
tx = wlanWaveformGenerator(psdu,cfgVHT);
```

Model Impairments

Sample rate offset between the transmitter and receiver is modeled by resampling the transmitted waveform. The `resample` function can be used to model a limited range of large sample rate offsets. In this example a -100 parts per million (PPM) sample rate offset is modeled.

```
% Model sample rate offset
p = 1e4;    % Interpolation factor
q = 1e4-1; % Decimation factor
sro = (1-p/q)*1e6;
disp('Impairments:');
disp(['  Sample rate offset (SRO): ' num2str(sro,'%.1f') ' PPM']);

% Resample the waveform, appending zeros to allow for filter delay
N = 100; % Size of filter used for resampling
rx = resample([tx; zeros(N,cfgVHT.NumTransmitAntennas)],p,q,N);

Impairments:
  Sample rate offset (SRO): -100.0 PPM
```

Residual carrier frequency offset is added to the waveform using the helperFrequencyOffset function. In this example we assume the same oscillator is used for sampling and modulation, therefore the CFO will be a function of the SRO and carrier frequency.

```
fc = 5.25e9;          % Carrier frequency, Hertz
cfo = (sro*1e-6)*fc; % Carrier frequency offset, Hertz
disp(['  Carrier frequency offset (CFO): ' num2str(cfo,'%.1f') ' Hz']);

fs = wlanSampleRate(cfgVHT);           % Baseband sample rate
rx = helperFrequencyOffset(rx,fs,cfo); % Add frequency offset

  Carrier frequency offset (CFO): -525052.5 Hz
```

Noise is added to the waveform with 30 dBW variance.

```
awgnChannel = comm.AWGNChannel('NoiseMethod','Variance','Variance',10^(-30/10));
rx = awgnChannel(rx);
```

Front-End Synchronization and Receiver Processing

The following processing steps occur to synchronize the packet, in preparation for recovering the data field.

1. The packet is detected
2. Coarse carrier frequency offset is estimated and corrected
3. Symbol timing synchronization is established
4. Fine carrier frequency offset is estimated and corrected
5. The L-LTF is OFDM demodulated and noise estimation is performed
6. The VHT-LTF is OFDM demodulated and channel estimation is performed

```
% Generate field indices
ind = wlanFieldIndices(cfgVHT);

% Packet detection
tOff = wlanPacketDetect(rx,cfgVHT.ChannelBandwidth);

% Coarse frequency offset correction
lstf = rx(tOff+(ind.LSTF(1):ind.LSTF(2)),:);
coarseCFOEst = wlanCoarseCFOEstimate(lstf,cfgVHT.ChannelBandwidth);
rx = helperFrequencyOffset(rx,fs,-coarseCFOEst);

% Symbol timing synchronization
nonhtPreamble = rx(tOff+(ind.LSTF(1):ind.LSIG(2)),:);
symOff = wlanSymbolTimingEstimate(nonhtPreamble,cfgVHT.ChannelBandwidth);
tOff = tOff+symOff;

% Fine frequency offset correction
lltf = rx(tOff+(ind.LLTF(1):ind.LLTF(2)),:);
fineCFOEst = wlanFineCFOEstimate(lltf,cfgVHT.ChannelBandwidth);
rx = helperFrequencyOffset(rx,fs,-fineCFOEst);

% Channel estimation
vhtltf = rx(tOff+(ind.VHTLTF(1):ind.VHTLTF(2)),:);
vhtltfDemod = wlanVHTLTFDemodulate(vhtltf,cfgVHT);
chanEst = wlanVHTLTFChannelEstimate(vhtltfDemod,cfgVHT);

% Get single stream channel estimate
chanEstSSPilots = vhtSingleStreamChannelEstimate(vhtltfDemod,cfgVHT);
```

Recovery Without Sample Rate Offset or Residual CFO Tracking

The coarse and fine frequency offset estimation and correction removes the majority of CFO, but residual CFO remains due to the presence of impairments in the waveform. This must be tracked and corrected by the receiver.

```
disp('Front-end impairment correction:');
frontEndCFOEst = coarseCFOEst+fineCFOEst;
disp(['  Estimated CFO: ' num2str(frontEndCFOEst,'%.1f') ' Hz']);
residualCFO = cfo-frontEndCFOEst;
disp(['  Residual CFO after initial correction: ' num2str(residualCFO,'%.1f') ' Hz']);
```

```
Front-end impairment correction:
  Estimated CFO: -524764.3 Hz
  Residual CFO after initial correction: -288.2 Hz
```

The trackingVHTDataRecover function recovers the VHT data field with optional pilot tracking to correct for timing and phase errors due to SRO and CFO. Pilot tracking is controlled using the trackingRecoveryConfig object .

The data field is first recovered without pilot tracking. The data field is extracted from the waveform using the start and end sample indices of the field at the baseband rate. If the receiver sampling rate is higher than the transmitter rate, the receiver requires more samples than the transmitter produces. To allow for this, Ne additional samples are extracted from the waveform and passed to the recovery function. The maximum number of additional samples required is a function of the expected SRO, the baseband sampling rate, and the maximum packet duration.

```
% Recovery configuration with pilot tracking disabled
cfgRec = trackingRecoveryConfig;
```

```
cfgRec.PilotTracking = 'None';

% Extract data field with Ne additional samples to allow for negative SRO
maxDuration = 5.484e-3; % Maximum packet duration in seconds
maxSRO = 120;           % PPM
Ne = ceil(fs*maxDuration*maxSRO*1e-6); % Number of extra samples
dataInd = tOff+(ind.VHTData(1):ind.VHTData(2)+Ne);
dataInd = dataInd(dataInd<=length(rx)); % Only use indices within waveform
data = rx(dataInd,:);

% Perform demodulation and decoding
[rxPSDUNoTrack,~,eqSymNoTrack] = trackingVHTDataRecover(data,chanEst,chanEstSSPilots,cfgVHT,cfgRe
```

The equalized constellation is plotted which shows a rotation of all constellation points caused by residual CFO, and a spreading of constellation points due to SRO. Despite the modest AWGN added to the waveform, the impairments cause bit errors within the decoded PSDU.

```
ConstNoTrack = comm.ConstellationDiagram;
ConstNoTrack.Title = 'Equalized symbols with no pilot tracking';
ConstNoTrack.ReferenceConstellation = wlanReferenceSymbols(cfgVHT);
ConstNoTrack(eqSymNoTrack(:));

[~,berNoTrack] = biterr(rxPSDUNoTrack,psdu);
disp('Bit error rate:');
disp(['  Without tracking: ' num2str(berNoTrack)]);

Bit error rate:
  Without tracking: 0.091022
```
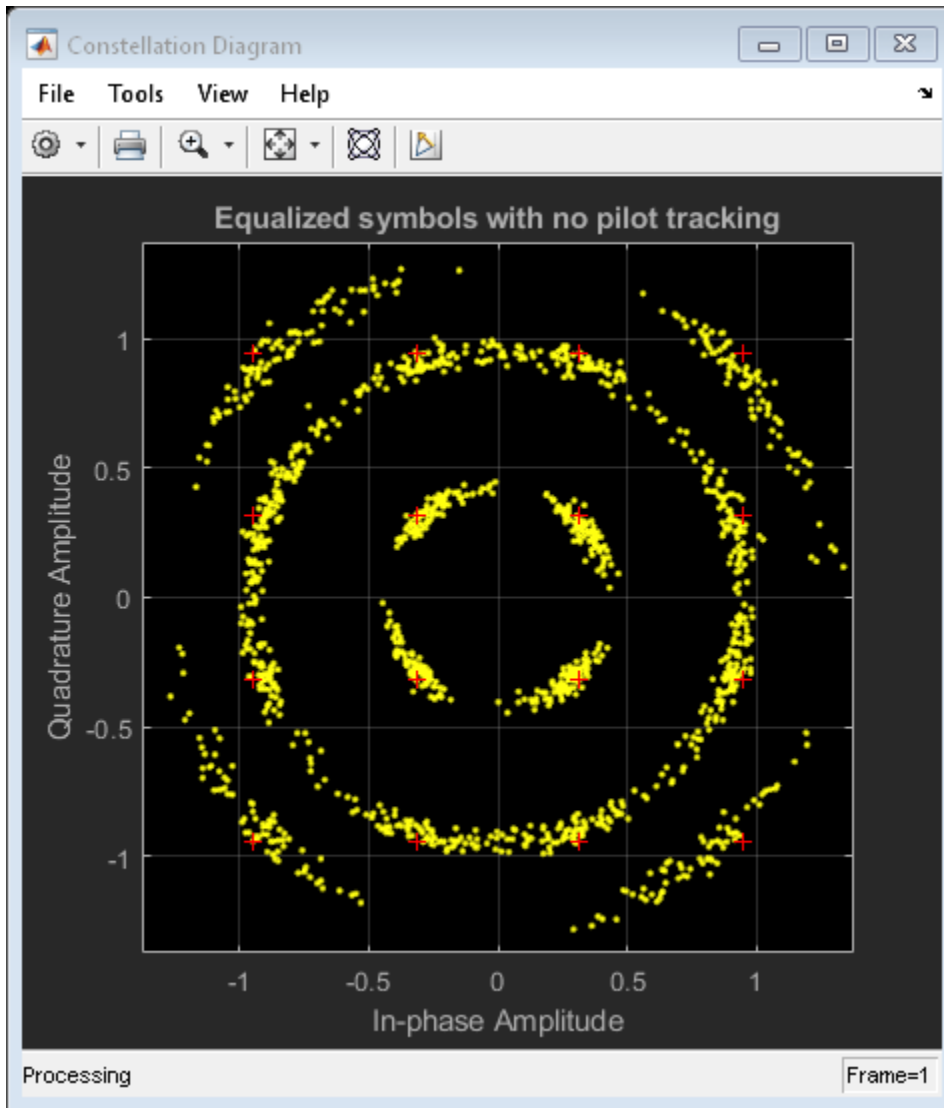
Recovery With Sample Rate Offset Tracking and Residual CFO Tracking

Now the data field is recovered with joint timing and phase pilot tracking to correct for SRO and residual CFO.

The tracking algorithm used in this example estimates absolute values of δ and ω per OFDM symbol, and applies a per subcarrier and symbol phase correction to the demodulated symbols to reverse the phase errors caused by SRO and CFO. The phase error between each received pilot subcarrier and the expected value is calculated per symbol and averaged over `PilotTrackingWindow` OFDM symbols. From this, least-square estimates of δ and ω are calculated per symbol. These estimates are used to apply a phase correction to each symbol and subcarrier [3, 4].

```
% Recovery configuration with pilot tracking enabled
cfgRec = trackingRecoveryConfig;
cfgRec.PilotTracking = 'Joint'; % Joint timing and phase tracking
cfgRec.PilotTrackingWindow = 9; % Averaging window in OFDM symbols
```

```
% Perform demodulation and decoding
[rxPSDU,~,eqSymTrack,cpe,peg] = trackingVHTDataRecover(data,chanEst,chanEstSSPilots,cfgVHT,cfgRec
```

The equalized constellation is plotted which shows a clear 16-QAM constellation with no spreading or rotation. There are no bit errors.

```
ConstTrack = comm.ConstellationDiagram;
ConstTrack.Title = 'Equalized symbols with joint pilot tracking';
ConstTrack.ReferenceConstellation = wlanReferenceSymbols(cfgVHT);
ConstTrack(eqSymTrack(:));

[~,berTrack] = biterr(rxPSDU,psdu);
disp(['  With tracking: ' num2str(berTrack)]);

  With tracking: 0
```
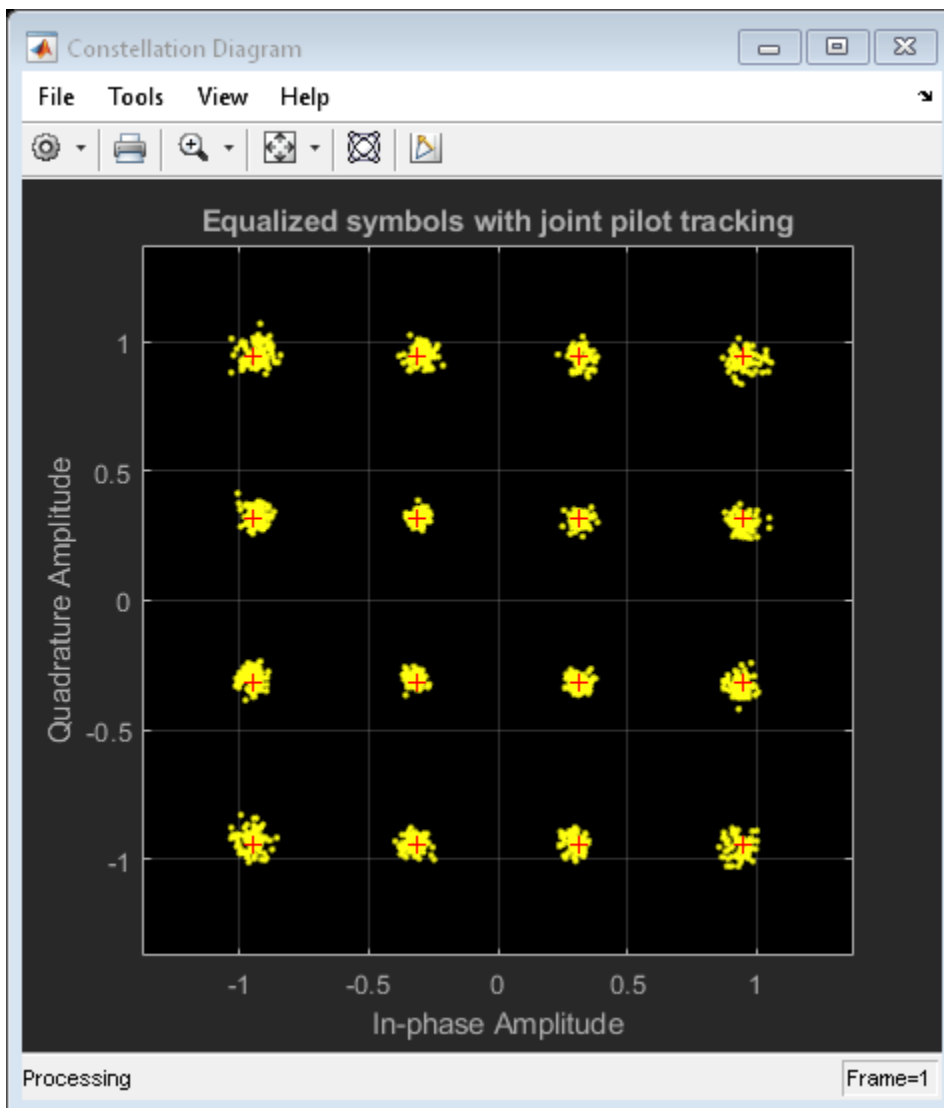

The trackingVHTDataRecover function returns measurements from which the residual CFO, and SRO can be estimated:

- `cpe` - The common phase error (radians) per symbol
- `peg` - The phase error gradient (radians per subcarrier) per symbol

The SRO and residual CFO are estimated from these measurements using a linear least-square fit of the rate of change. The measurements are plotted using the trackingPlotSROCFOEstimates function.

```
[residualCFOEst,sroEst] = trackingPlotSROCFOEstimates(cpe,peg,cfgVHT);

% Display estimated SRO, residual CFO and total CFO
fprintf('Tracked impairments:\n');
fprintf('  Estimated residual CFO: %3.1f Hz (%.1f Hz error)\n', ...
    residualCFOEst,residualCFOEst-residualCFO);
fprintf('  Estimated SRO: %3.1f PPM (%.1f PPM error)\n',sroEst,sroEst-sro);
cfoEst = frontEndCFOEst+residualCFOEst; % Initial + tracked CFO estimate
fprintf('Estimated CFO (initial + tracked): %.1f Hz (%.1f Hz error)\n',cfoEst,cfoEst-cfo);

rng(s); % Restore the state of the random number generator
```
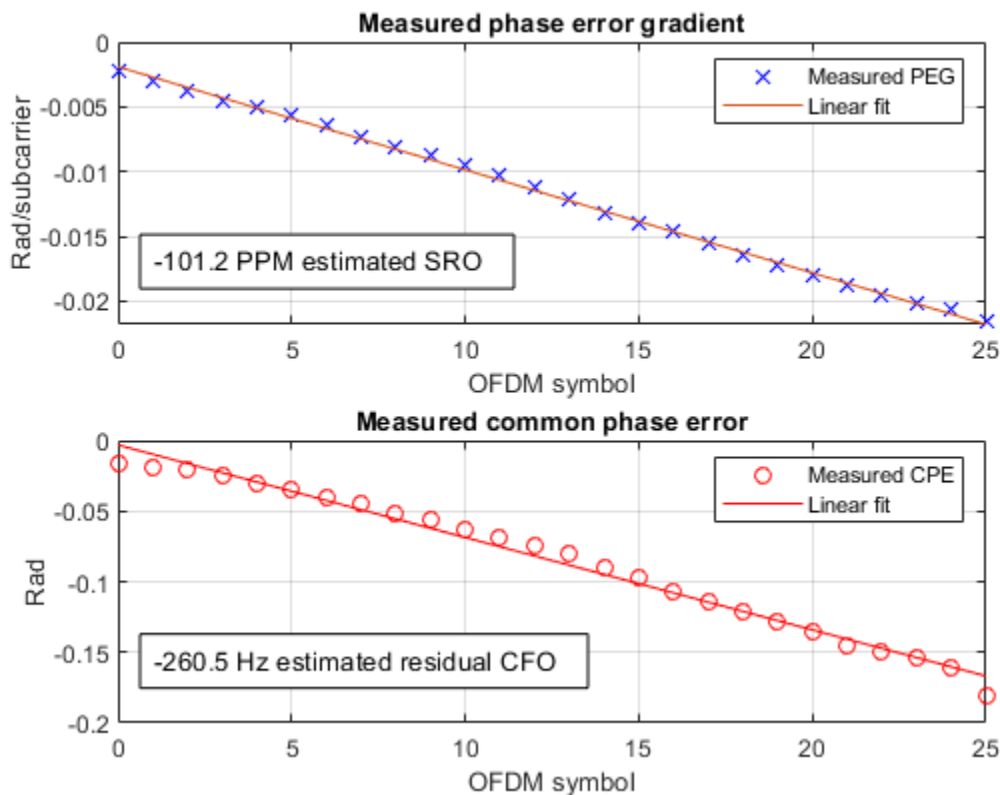
```
Tracked impairments:
  Estimated residual CFO: -260.5 Hz (27.8 Hz error)
  Estimated SRO: -101.2 PPM (-1.2 PPM error)
Estimated CFO (initial + tracked): -525024.8 Hz (27.8 Hz error)
```


Conclusion

This example shows how you can track and correct sample rate and carrier frequency offsets when recovering the data field of a WLAN waveform.

Data field recovery functions with joint pilot tracking for VHT, HT-MF and non-HT formats are provided in this example, along with an object to configure the recovery algorithms:

- trackingVHTDataRecover.m
- trackingHTDataRecover.m
- trackingNonHTDataRecover.m
- trackingRecoveryConfig.m

To see an example of pilot tracking for HE format packets see "Recovery Procedure for an 802.11ax Packet" on page 4-12.

Appendix

This example uses the following helper functions:

- helperFrequencyOffset.m
- trackingPlotSROCFOEstimates.m
- trackingRecoveryConfig.m
- trackingVHTDataRecover.m
- vhtSingleStreamChannelEstimate.m

Selected Bibliography

1   Speth, Michael, et al. "Optimum receiver design for wireless broad-band systems using OFDM." IEEE Transactions on communications 47.11 (1999): 1668-1677.

2   IEEE Std 802.11™-2016 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   Chiueh, Tzi-Dar, Pei-Yun Tsai, and I-Wei Lai. Baseband receiver design for wireless MIMO-OFDM communications. John Wiley & Sons, 2012.

4   Horlin, P.F. and Bourdoux, A. Digital Compensation for Analog Front-Ends: A New Approach to Wireless Transceiver Design. Wiley, 2008.

Transmit and Recover L-SIG, VHT-SIG-A, VHT-SIG-B in Fading Channel

Transmit a VHT waveform through a noisy MIMO channel. Extract the L-SIG, VHT-SIG-A, and VHT-SIG-B fields and verify that they were correctly recovered.

Set the parameters used throughout the example.

```
cbw = 'CBW40';                      % Channel bandwidth
fs = 40e6;                          % Sample rate (Hz)
ntx = 2;                            % Number of transmit antennas
nsts = 2;                           % Number of space-time streams
nrx = 3;                            % Number of receive antennas
```

Create a VHT configuration object that supports a 2x2 MIMO transmission and has an APEP length of 2000.

```
vht = wlanVHTConfig('ChannelBandwidth',cbw,'APEPLength',2000, ...
    'NumTransmitAntennas',ntx,'NumSpaceTimeStreams',nsts, ...
    'SpatialMapping','Direct','STBC',false);
```

Generate a VHT waveform containing a random PSDU.

```
txPSDU = randi([0 1],vht.PSDULength*8,1);
txPPDU = wlanWaveformGenerator(txPSDU,vht);
```

Create a 2x2 TGac channel and an AWGN channel with an SNR=10 dB.

```
tgacChan = wlanTGacChannel('SampleRate',fs,'ChannelBandwidth',cbw, ...
    'NumTransmitAntennas',ntx,'NumReceiveAntennas',nrx, ...
    'LargeScaleFadingEffect','Pathloss and shadowing', ...
    'DelayProfile','Model-C');

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

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

```
rxPPDU = chNoise(tgacChan(txPPDU));
```

Add additional white noise corresponding to a receiver with a 9 dB noise figure. The noise variance is equal to $k*T*B*F$, where k is Boltzmann's constant, T is the ambient temperature, B is the channel bandwidth (sample rate), and F is the receiver noise figure.

```
nVar = 10^((-228.6+10*log10(290) + 10*log10(fs) + 9 )/10);
rxNoise = comm.AWGNChannel('NoiseMethod','Variance','Variance',nVar);

rxPPDU = rxNoise(rxPPDU);
```

Find the start and stop indices for all component fields of the PPDU.
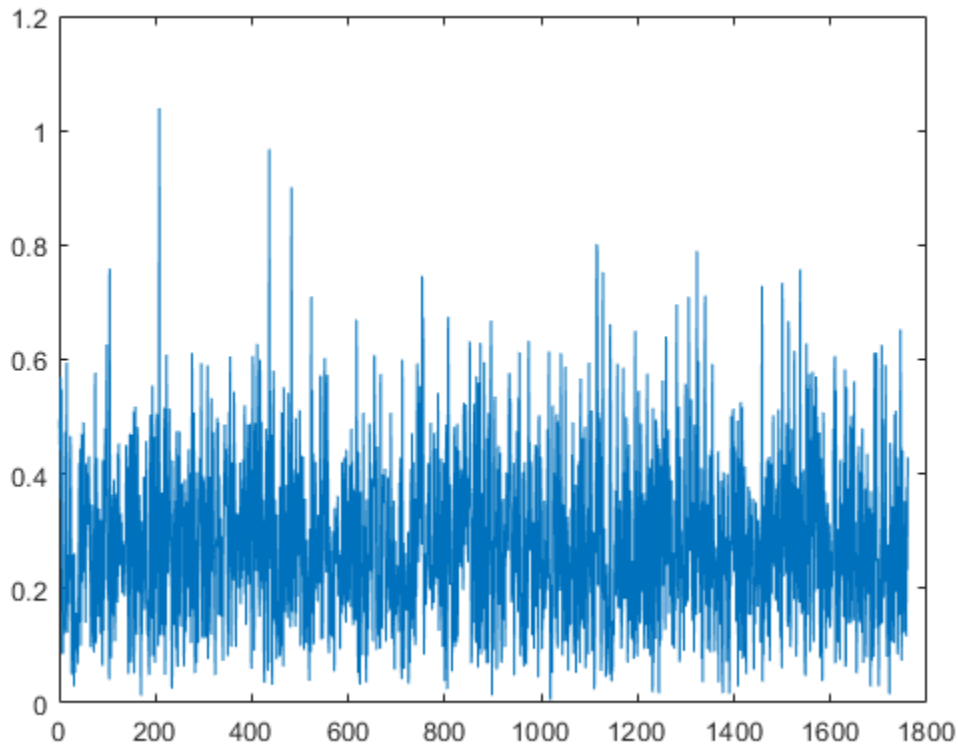
```
ind = wlanFieldIndices(vht)

ind = struct with fields:
     LSTF: [1 320]
     LLTF: [321 640]
     LSIG: [641 800]
```

```
VHTSIGA: [801 1120]
 VHTSTF: [1121 1280]
 VHTLTF: [1281 1600]
VHTSIGB: [1601 1760]
VHTData: [1761 25600]
```

The preamble is contained in the first 1760 symbols. Plot the preamble.

```
plot(abs(rxPPDU(1:1760)))
```


Extract the L-LTF from the received PPDU using the start and stop indices determined by the `wlanFieldIndices` function. Demodulate the L-LTF and estimate the channel coefficients.

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

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

```
rxLSIG = rxPPDU(ind.LSIG(1):ind.LSIG(2),:);
infoLSIG = wlanLSIGRecover(rxLSIG,chEstLLTF,nVar,cbw);
```

Inspect the L-SIG rate information and confirm that the sequence [1 1 0 1] is received. This sequence corresponds to a 6 MHz data rate, which is used for all VHT transmissions.

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

```
rate = 1x4 int8 row vector

   0   1   1   1
```

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

```
rxVHTSIGA = rxPPDU(ind.VHTSIGA(1):ind.VHTSIGA(2),:);
[infoVHTSIGA,failCRC] = wlanVHTSIGARecover(rxVHTSIGA, ...
    chEstLLTF,nVar,cbw);
failCRC

failCRC = logical
   1
```

Extract and demodulate the VHT-LTF. Use the demodulated signal to estimate the channel coefficients needed to recover the VHT-SIG-B field.

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

Extract and recover the VHT-SIG-B.

```
rxVHTSIGB = rxPPDU(ind.VHTSIGB(1):ind.VHTSIGB(2),:);
infoVHTSIGB = wlanVHTSIGBRecover(rxVHTSIGB,chEstVHTLTF,nVar,cbw);
```

Verify that the APEP length, contained in the first 19 bits of the VHT-SIG-B, corresponds to the specified length of 2000 bits.

```
pktLbits = infoVHTSIGB(1:19)';
pktLen = bi2de(double(pktLbits))*4

pktLen = 1676920
```

End-to-End VHT Simulation with Frequency Correction

This example shows how to generate, transmit, recover and view a VHT MIMO waveform.

Steps in the example:

- Transmit a VHT waveform through a MIMO channel with AWGN
- Perform a two-stage process to estimate and correct for a frequency offset
- Estimate the channel response
- Recover the VHT data field
- Compare the transmitted and received PSDUs to determine if bit errors occurred

Set the parameters used throughout the example.

```
cbw = 'CBW160';                    % Channel bandwidth
fs = 160e6;                        % Sample rate (Hz)
ntx = 2;                           % Number of transmit antennas
nsts = 2;                          % Number of space-time streams
nrx = 2;                           % Number of receive antennas
```

Create a VHT configuration object that supports a 2x2 MIMO transmission and has an APEP length of 2000.

```
vht = wlanVHTConfig('ChannelBandwidth',cbw,'APEPLength',2000, ...
    'NumTransmitAntennas',ntx,'NumSpaceTimeStreams',nsts, ...
    'SpatialMapping','Direct','STBC',false);
```

Generate a VHT waveform containing a random PSDU.

```
txPSDU = randi([0 1],vht.PSDULength*8,1);
txPPDU = wlanWaveformGenerator(txPSDU,vht);
```

Create a 2x2 TGac channel and an AWGN channel.

```
tgacChan = wlanTGacChannel('SampleRate',fs,'ChannelBandwidth',cbw, ...
    'NumTransmitAntennas',ntx,'NumReceiveAntennas',nrx, ...
    'LargeScaleFadingEffect','Pathloss and shadowing', ...
    'DelayProfile','Model-C');
awgnChan = comm.AWGNChannel('NoiseMethod','Variance', ...
    'VarianceSource','Input port');
```

Create a phase/frequency offset object.

```
pfOffset = comm.PhaseFrequencyOffset('SampleRate',fs,'FrequencyOffsetSource','Input port');
```

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

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

Introduce a frequency offset of 500 Hz.

```
rxPPDUcfo = pfOffset(rxPPDU,500);
```

Find the start and stop indices for all component fields of the PPDU.

```
ind = wlanFieldIndices(vht);
```

Extract the L-STF. Estimate and correct for the carrier frequency offset.

```
rxLSTF = rxPPDUcfo(ind.LSTF(1):ind.LSTF(2),:);
```

```
foffset1 = wlanCoarseCFOEstimate(rxLSTF,cbw);
rxPPDUcorr = pfOffset(rxPPDUcfo,-foffset1);
```

Extract the L-LTF from the corrected signal. Estimate and correct for the residual frequency offset.

```
rxLLTF = rxPPDUcorr(ind.LLTF(1):ind.LLTF(2),:);
```

```
foffset2 = wlanFineCFOEstimate(rxLLTF,cbw);
rxPPDU2 = pfOffset(rxPPDUcorr,-foffset2);
```

Extract and demodulate the VHT-LTF. Estimate the channel coefficients.

```
rxVHTLTF = rxPPDU2(ind.VHTLTF(1):ind.VHTLTF(2),:);
dLTF = wlanVHTLTFDemodulate(rxVHTLTF,vht);
chEst = wlanVHTLTFChannelEstimate(dLTF,vht);
```

Extract the VHT data field from the received and frequency-corrected PPDU. Recover the data field.

```
rxVHTData = rxPPDU2(ind.VHTData(1):ind.VHTData(2),:);
rxPSDU = wlanVHTDataRecover(rxVHTData,chEst,nVar,vht);
```

Calculate the number of bit errors in the received packet.

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

```
numErr = 0
```

Propagation Channel Models

802.11ad Packet Error Rate Single Carrier PHY Simulation with TGay Channel

This example shows how to measure the packet error rate of an IEEE® 802.11ad™ DMG single carrier (SC) PHY link using an end-to-end simulation.

Introduction

In this example an end-to-end simulation is used to determine the packet error rate for an 802.11ad SC [1] link with a TGay millimeter-wave channel [2] at a selection of SNR points. At each SNR point multiple packets are transmitted through the fading channel, synchronized, demodulated and the PSDUs recovered. Carrier frequency offset and a time delay are also modeled. The PSDUs are compared to those transmitted to determine the number of packet errors and hence the packet error rate. The processing for each packet is summarized in the following diagram.
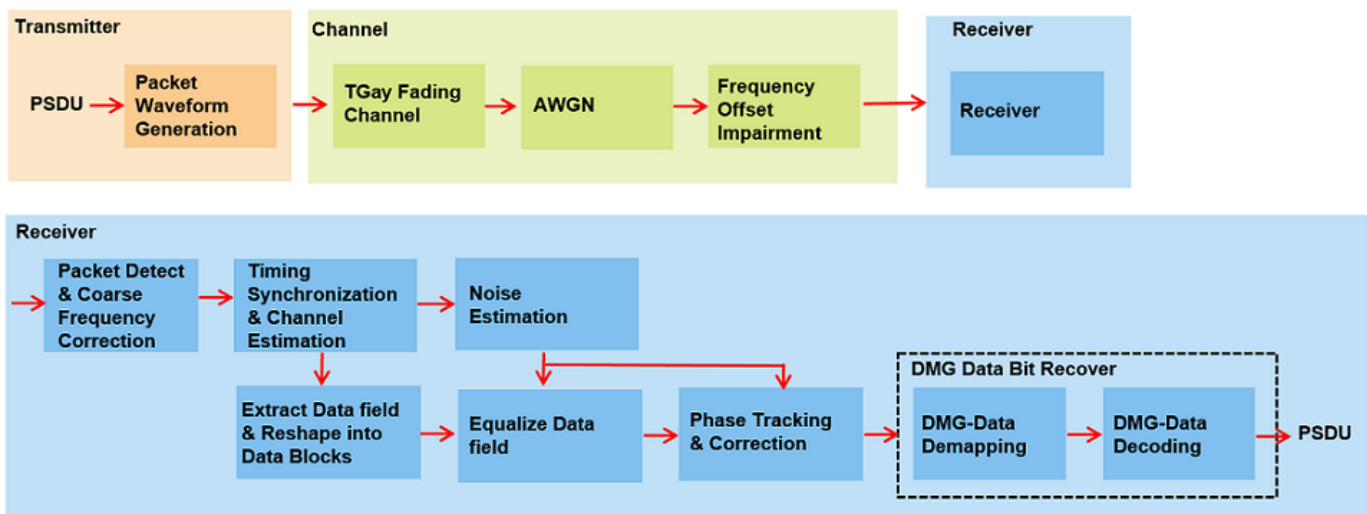
This example also demonstrates how a `parfor` loop can be used instead of the `for` loop when simulating each SNR point to speed up a simulation. `parfor`, as part of the Parallel Computing Toolbox™, executes processing for each SNR in parallel to reduce the total simulation time.

Waveform Configuration

An 802.11ad DMG SC PHY transmission is simulated in this example. The DMG format configuration object contains the format specific configuration of the transmission. The object is created using the `wlanDMGConfig` function. The properties of the object contain the configuration of the transmitted packet. In this example the object is configured to generate a single carrier waveform of MCS "9". The MCS determines the PHY type used and in this example it must be a string within the range 1-12, or one of {9.1 12.1 12.2 12.3 12.4 12.5 12.6} to simulate the SC PHY.

```
% Create a format configuration object
cfgDMG = wlanDMGConfig;
mcs = "9"; % MCS specified as a string scalar or string vector
cfgDMG.PSDULength = 4096; % PSDULength in bytes
```

Fading Channel Configuration

In this example we simulate a TGay channel model for the open area hotspot scenario, using the `wlanTGayChannel` object. Both the transmit and receive arrays are a 4x4 uniform rectangular array

(URA). Ray-tracing is performed from the transmit to the receive array to derive two deterministic rays: one LOS ray and another NLOS ray with one-order reflection from the ground. Random rays and intra-cluster rays are subsequently generated according to the quasi-deterministic (Q-D) modeling approach and parameters in [2]. Beamforming is performed to set the transmit and receive antenna steering direction along the ray with the maximum power. The fading channel impulse responses are normalized so no power gain or loss is introduced by the channel. Both the channel input and output signals are unpolarized.

```matlab
% Get sampling rate and specify carrier frequency
fs = wlanSampleRate(cfgDMG);
fc = 60e9;

% Create a TGay channel object
tgayChan = wlanTGayChannel;
tgayChan.SampleRate                = fs;
tgayChan.CarrierFrequency          = fc;
tgayChan.Environment               = 'Open area hotspot';
tgayChan.TransmitArray.Size        = [4 4];
tgayChan.TransmitArrayPosition     = [0; 0; 6];      % Meters
tgayChan.TransmitArrayOrientation  = [0; 270; 0];    % Degrees
tgayChan.ReceiveArray.Size         = [4 4];
tgayChan.ReceiveArrayPosition      = [6; 6; 1.5];    % Meters
tgayChan.ReceiveArrayOrientation   = [90; 0; 0];     % Degrees
tgayChan.BeamformingMethod         = 'Maximum power ray';
tgayChan.NormalizeImpulseResponses = true;
```

Given the channel object configuration, we display a 3D map to show the environment and antenna array settings. The two deterministic rays from ray-tracing are also shown in the figure.

```matlab
showEnvironment(tgayChan);
```

Open Area Hotspot With Antenna Arrays and D-Rays
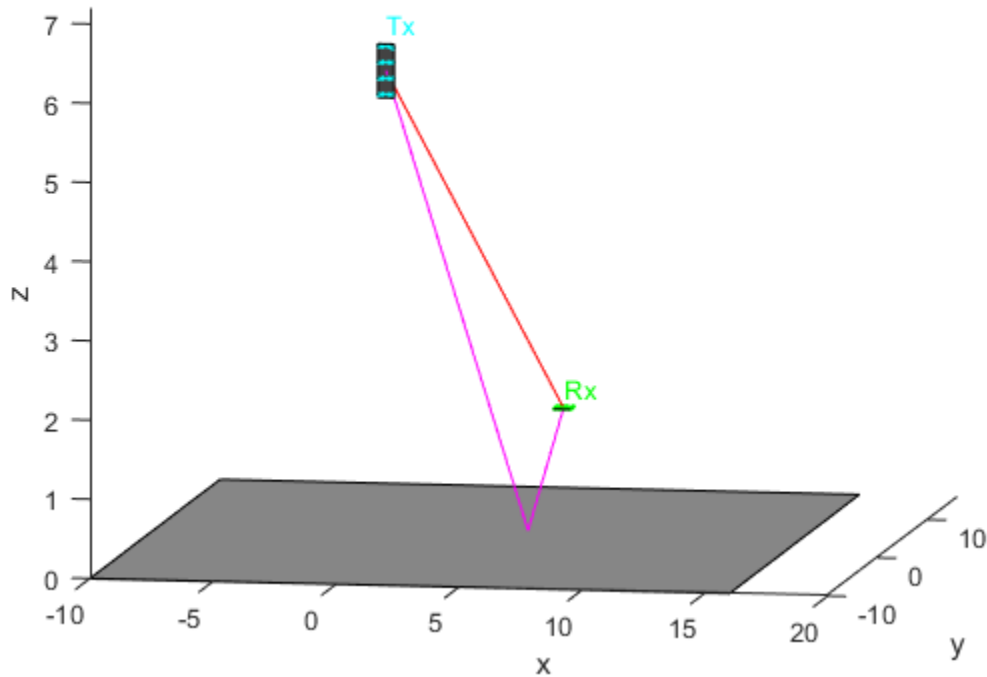
Channel Impairment

The maximum tolerance for the transmitter center frequency must be within [-20, +20] ppm [1]. In this example, a clock accuracy of 20ppm is considered to derive the CFO. The transmitted signal is delayed by 500 samples and also appended by 100 zero samples at the end to account for delays from TGay channel filtering.

```
ppm = 20;                   % Clock accuracy to drive the CFO (ppm)
freqOffset = ppm*1e-6*fc;   % Carrier frequency offset (Hz)
delay = 500;                % Sample to delay the waveform
zeroPadding = 100;          % Add trailing zeros to allow for channel delay
```

Simulation Parameters

For each SNR point (dB) in the cell `snrRanges` a number of packets are generated, passed through a channel and demodulated to determine the packet error rate. The SNR points to test are selected from `snrRanges` based on the MCS simulated.

```
snrRanges = {-2.0:0.5:0.5, ...    % MCS 1
             0.0:1.0:5.0, ...     % MCS 2
             1.0:1.2:7.0, ...     % MCS 3
             2.0:1.2:8.0, ...     % MCS 4
             2.0:1.3:8.5, ...     % MCS 5
             2.5:1.3:9.0, ...     % MCS 6
             4.0:1.3:10.5, ...    % MCS 7
             5.0:1.5:12.5,...     % MCS 8
             5.5:1.5:13.0, ...    % MCS 9
```

```
    7.0:1.5:14.5, ...  % MCS 9.1
    8.0:1.8:17.0, ...  % MCS 10
    10.0:2.0:20.0, ... % MCS 11
    12.0:2.0:22.0, ... % MCS 12
    12.0:2.0:22.0, ... % MCS 12.1
    14.0:2.0:24.0, ... % MCS 12.2
    16.0:2.5:28.5, ... % MCS 12.3
    17.0:2.5:29.5, ... % MCS 12.4
    17.0:2.5:29.5, ... % MCS 12.5
    20.0:2.5:32.5}; % MCS 12.6
```

The number of packets tested at each SNR point is controlled by two parameters:

1. `maxNumErrors` is the maximum number of packet errors simulated at each SNR point. When the number of packet errors reaches this limit, the simulation at this SNR point is complete.

2. `maxNumPackets` is the maximum number of packets simulated at each SNR point and limits the length of the simulation if the packet error limit is not reached.

The numbers chosen in this example will lead to a very short simulation. For meaningful results we recommend increasing the numbers.

```
maxNumErrors = 10;   % The maximum number of packet errors at an SNR point
maxNumPackets = 100; % Maximum number of packets at an SNR point
```

Processing SNR Points

For each SNR point a number of packets are tested and the packet error rate calculated.

For each packet the following processing steps occur:

1. A PSDU is created and encoded to create a single packet waveform.
2. The waveform is passed through a TGay channel model. Different channel realizations are modeled for different packets.
3. AWGN is added to the received waveform. `comm.AWGNChannel` is configured to provide the correct SNR.
4. The frequency offset impairment is added to each packet.
5. The packet is detected.
6. Carrier frequency offset is estimated and corrected.
7. Fine timing synchronization is established. The CE field samples are provided for fine timing to allow for packet detection at the start of the STF.
8. The STF and CE fields are extracted from the synchronized received waveform. The noise and channel estimation is performed on the recovered fields respectively.
9. The data field, excluding the first guard interval is extracted and reshaped into blocks. The received symbols in the data field are equalized.
10. The received symbols are tracked and corrected for phase errors caused by any residual carrier frequency offset.
11. The data field is decoded to recover the PSDU bits.

A `parfor` loop can be used to parallelize processing of the SNR points, therefore for each SNR point an AWGN channel is created and configured with `comm.AWGNChannel`. To enable the use of parallel computing for increased speed comment out the `for` statement and uncomment the `parfor` statement below.

```matlab
numSNR = numel(snrRanges{1}); % Number of SNR points
if ~isstring(mcs)
    error('MCS must be specified as a string scalar or string vector');
end
numMCS = numel(mcs);            % Number of MCS
packetErrorRate = zeros(numMCS,numSNR);
Ngi = 64; % Fixed GI length defined in the standard (20.6.3.2.5)
validMCS = string(sort([1:12 9.1 12.1:0.1:12.6]));

for imcs = 1:numMCS
    cfgDMG.MCS = mcs(imcs);
    if ~strcmp(phyType(cfgDMG),'SC')
        error('This example only supports DMG SC PHY simulation');
    end
    ind = wlanFieldIndices(cfgDMG);
    snr = snrRanges{mcs(imcs)==validMCS}; % SNR points to simulate from MCS

    % parfor isnr = 1:numSNR % Use 'parfor' to speed up the simulation
    for isnr = 1:numSNR     % Use 'for' to debug the simulation
        % Set random substream index per iteration to ensure that each
        % iteration uses a repeatable set of random numbers
        stream = RandStream('combRecursive','Seed',10);
        stream.Substream = isnr;
        RandStream.setGlobalStream(stream);

        % Create an instance of the AWGN channel per SNR point simulated
        awgnChannel = comm.AWGNChannel;
        awgnChannel.NoiseMethod = 'Signal to noise ratio (SNR)';
        awgnChannel.SNR = snr(isnr);

        % Set simulation parameters
        numPacketErrors = 0;
        numPkt = 1; % Index of the transmitted packet

        while numPacketErrors<=maxNumErrors && numPkt<=maxNumPackets
            % Generate a packet waveform
            psdu = randi([0 1],cfgDMG.PSDULength*8,1);
            txWaveform = wlanWaveformGenerator(psdu,cfgDMG);

            % Add delay and trailing zeros
            tx = [zeros(delay,1); txWaveform; zeros(zeroPadding,1)];

            % Transmit through a TGay channel. Reset the channel for a
            % different realization per packet.
            reset(tgayChan);
            chanOut = tgayChan(tx);

            % Add noise
            rx = awgnChannel(chanOut);

            % Add CFO
            rx = helperFrequencyOffset(rx,fs,freqOffset);

            % Packet detection
            threshold = 0.03; % Good for low SNRs
            pktStartOffset = dmgPacketDetect(rx,0,threshold);
            if isempty(pktStartOffset) % If empty no STF detected; packet error
                numPacketErrors = numPacketErrors+1;
```

```
            numPkt = numPkt+1;
            continue; % Go to next loop iteration
        end

        % Frequency offset estimation and correction
        stf = rx(pktStartOffset+(ind.DMGSTF(1):ind.DMGSTF(2)));
        fOffsetEst = dmgCFOEstimate(stf);
        rx = helperFrequencyOffset(rx,fs,-fOffsetEst);

        % Symbol timing and channel estimate
        preamblefield = rx(pktStartOffset+1:pktStartOffset+ind.DMGHeader(2),:);
        [symbolTimingOffset,chanEst] = dmgTimingAndChannelEstimate(preamblefield);
        startOffset = pktStartOffset+symbolTimingOffset;

        % If not enough samples to decode detected data field start,
        % then assume synchronization error and packet error
        if (startOffset+ind.DMGData(2))>size(rx,1)
            numPacketErrors = numPacketErrors+1;
            numPkt = numPkt+1;
            continue; % Go to next loop iteration
        end

        % Noise estimation using the STF as repeating sequence
        stf = rx(pktStartOffset+(ind.DMGSTF(1):ind.DMGSTF(2)));
        nVarEst = dmgSTFNoiseEstimate(stf);

        % Extract data field (ignore first GI)
        rxData = rx(startOffset+((ind.DMGData(1)+Ngi):ind.DMGData(2)));

        % Linear frequency domain equalization
        rxEqDataBlks = dmgSingleCarrierFDE(rxData,chanEst,nVarEst);

        % Unique word phase tracking
        rxEqDataBlks = dmgUniqueWordPhaseTracking(rxEqDataBlks);

        % Discard GI from all blocks
        rxDataSym = rxEqDataBlks(1:end-Ngi,:);

        % Recover the transmitted PSDU from DMG Data field
        dataDecode = wlanDMGDataBitRecover(rxDataSym,nVarEst,cfgDMG);

        % Determine if any bits are in error, i.e. a packet error
        packetError = any(biterr(psdu,dataDecode));
        numPacketErrors = numPacketErrors+packetError;
        numPkt = numPkt+1;
    end

    % Calculate packet error rate (PER) at SNR point
    packetErrorRate(imcs,isnr) = numPacketErrors/(numPkt-1);
    disp(join(["      MCS:" cfgDMG.MCS ", SNR " ...
        num2str(snr(isnr)) " completed after " ...
        num2str(numPkt-1) " packets, PER: " ...
        num2str(packetErrorRate(imcs,isnr))],""));
    end
end

MCS:9, SNR 5.5 completed after 11 packets, PER: 1
MCS:9, SNR 7 completed after 14 packets, PER: 0.78571
```

```
MCS:9, SNR 8.5 completed after 20 packets, PER: 0.55
MCS:9, SNR 10 completed after 35 packets, PER: 0.31429
MCS:9, SNR 11.5 completed after 70 packets, PER: 0.15714
MCS:9, SNR 13 completed after 100 packets, PER: 0.08
```
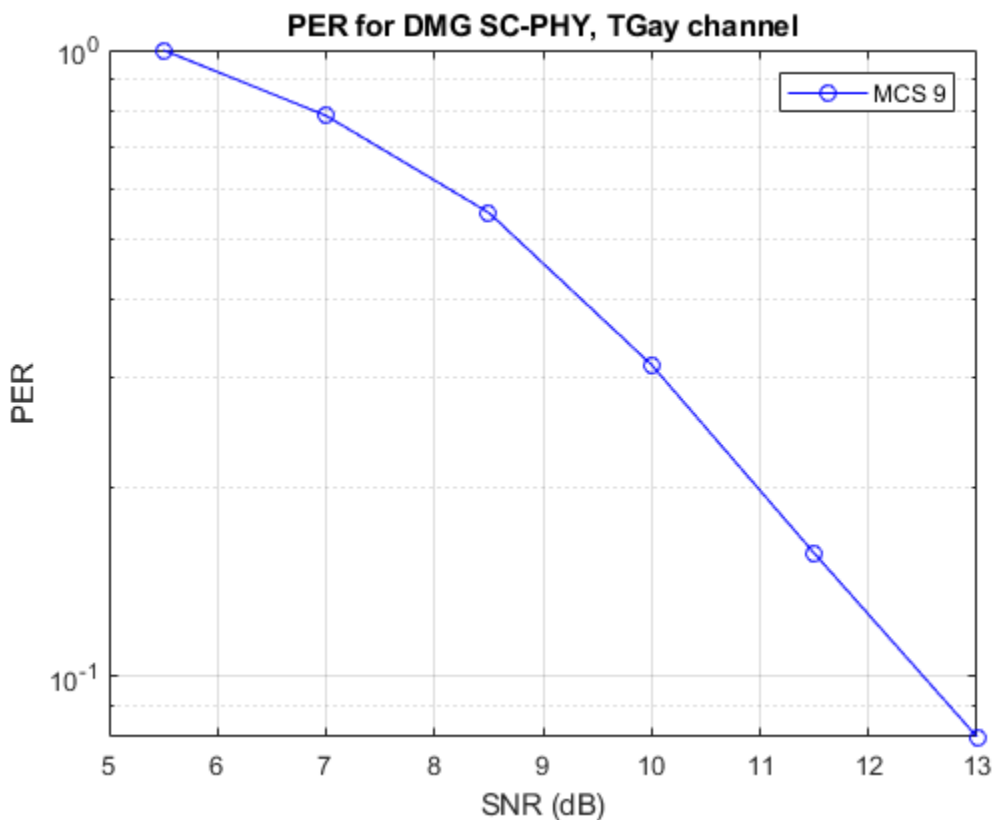
Plot Packet Error Rate vs SNR Results

```
markers = 'ox*sd^v><ph+ox*sd^v';
color = 'bmcrgbrkymcrgbrkymc';
figure;
for imcs = 1:numMCS
    semilogy(snrRanges{mcs(imcs)==validMCS},packetErrorRate(imcs,:).',['-' markers(imcs) color(in
    hold on;
end
grid on;
xlabel('SNR (dB)');
ylabel('PER');
dataStr = arrayfun(@(x)sprintf('MCS %s',x),mcs,'UniformOutput',false);
legend(dataStr);
title('PER for DMG SC-PHY, TGay channel');
```


Further Exploration

The number of packets tested at each SNR point is controlled by two parameters; `maxNumErrors` and `maxNumPackets`. For meaningful results, it is recommended that these values should be larger than those presented in this example. Increasing the number of packets simulated allows the PER under different scenarios to be compared. Try changing the MCS value and compare the packet error

rate. As an example, the figure below was created by running the example for all single carrier MCS with `PSDULength`: 8192 bytes, `maxNumErrors`: 1000 and `maxNumPackets`: 10000.

PER for DMG SC-PHY, TGay channel
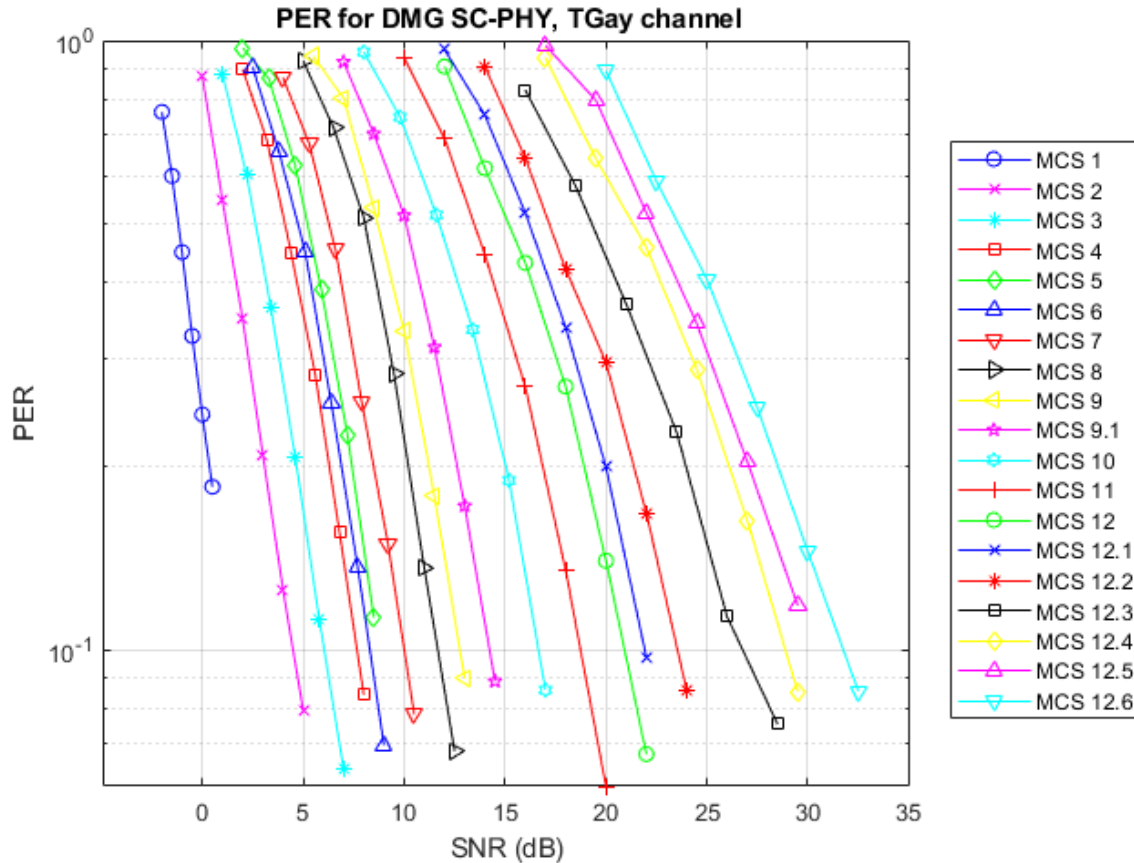
Explore the TGay channel settings by changing the environment ('Street canyon hotspot' or 'Large hotel lobby'), user configurations, polarization type, array configurations, beamforming method and so on. In the street canyon scenario, the object calculates deterministic rays up to 1-order reflection from the ground and walls. In the hotel lobby scenario, the deterministic rays are up to 2-order reflection from the ground, ceiling and/or walls. Due to the additional reflection of rays, the PER results are typically better than the above obtained from the open area scenario.

Appendix

This example uses the following helper functions and objects:

- helperFrequencyOffset.m
- dmgCFOEstimate.m
- dmgPacketDetect.m
- dmgTimingAndChannelEstimate.m

- dmgSingleCarrierFDE.m
- dmgSTFNoiseEstimate.m
- dmgUniqueWordPhaseTracking.m
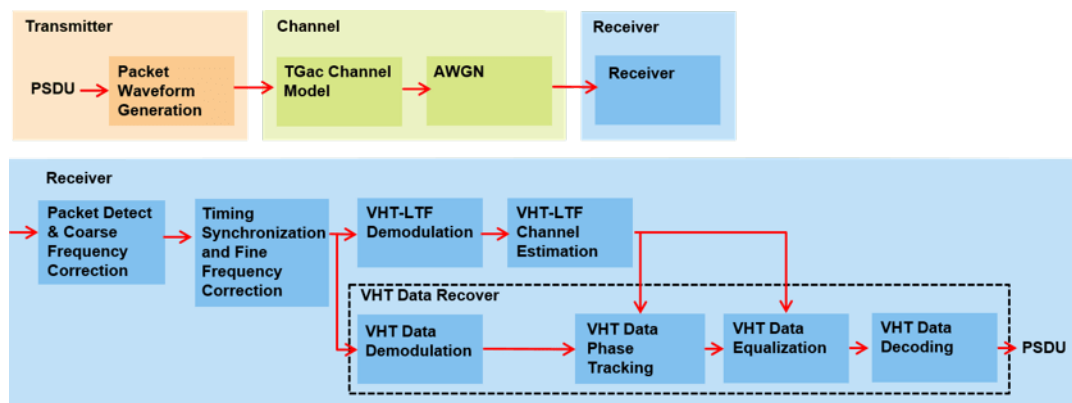
Selected Bibliography

1   IEEE Std 802.11ad™-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. Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band.

2   A. Maltsev and et. al, Channel Models for IEEE 802.11ay, IEEE 802.11-15/1150r9, Mar. 2017.

802.11ac Packet Error Rate Simulation for 8x8 TGac Channel

This example shows how to measure the packet error rate of an IEEE® 802.11ac™ VHT link using an end-to-end simulation with a fading TGac channel model and additive white Gaussian noise.

Introduction

In this example an end-to-end simulation is used to determine the packet error rate for an 802.11ac [1] VHT link with a fading channel at a selection of SNR points. At each SNR point multiple packets are transmitted through a channel, demodulated and the PSDUs recovered. The PSDUs are compared to those transmitted to determine the number of packet errors and hence the packet error rate. Packet detection, timing synchronization, carrier frequency offset correction and phase tracking are performed by the receiver. The processing for each packet is summarized in the following

This example also demonstrates how a `parfor` loop can be used instead of the `for` loop when simulating each SNR point to speed up a simulation. The `parfor` function, as part of the Parallel Computing Toolbox™, executes processing for each SNR in parallel to reduce the total simulation time.

Waveform Configuration

An 802.11ac VHT transmission is simulated in this example. The VHT format configuration object, `wlanVHTConfig`, contains the format-specific configuration of the transmission. The properties of the object contain the configuration. In this example the object is configured for a 80 MHz channel bandwidth, 8 transmit antennas, 8 space-time streams, no space time block coding and 256-QAM rate-5/6 (MCS 9).

```
% Create a format configuration object for a 8-by-8 VHT transmission
cfgVHT = wlanVHTConfig;
cfgVHT.ChannelBandwidth = 'CBW80'; % 80 MHz channel bandwidth
cfgVHT.NumTransmitAntennas = 8;    % 8 transmit antennas
cfgVHT.NumSpaceTimeStreams = 8;    % 8 space-time streams
cfgVHT.APEPLength = 3000;          % APEP length in bytes
cfgVHT.MCS = 9;                    % 256-QAM rate-5/6
```

Channel Configuration

In this example a TGac N-LOS channel model is used with delay profile Model-D. For Model-D when the distance between transmitter and receiver is greater than or equal to 10 meters, the model is NLOS. This is described further in `wlanTGacChannel`. An 8x8 MIMO channel is simulated in this example therefore 8 receive antennas are specified.

```
% Create and configure the channel
tgacChannel = wlanTGacChannel;
tgacChannel.DelayProfile = 'Model-D';
tgacChannel.NumReceiveAntennas = 8;
tgacChannel.TransmitReceiveDistance = 10; % Distance in meters for NLOS
tgacChannel.ChannelBandwidth = cfgVHT.ChannelBandwidth;
tgacChannel.NumTransmitAntennas = cfgVHT.NumTransmitAntennas;
tgacChannel.LargeScaleFadingEffect = 'None';
```

Simulation Parameters

For each SNR point in the vector `snr` a number of packets are generated, passed through a channel and demodulated to determine the packet error rate.

```
snr = 40:5:50;
```

The number of packets tested at each SNR point is controlled by two parameters:

1 `maxNumErrors` is the maximum number of packet errors simulated at each SNR point. When the number of packet errors reaches this limit, the simulation at this SNR point is complete.

2 `maxNumPackets` is the maximum number of packets simulated at each SNR point and limits the length of the simulation if the packet error limit is not reached.

The numbers chosen in this example will lead to a very short simulation. For meaningful results we recommend increasing the numbers.

```
maxNumErrors = 10;   % The maximum number of packet errors at an SNR point
maxNumPackets = 100; % Maximum number of packets at an SNR point
```

Set the remaining variables for the simulation.

```
% Get the baseband sampling rate
fs = wlanSampleRate(cfgVHT);

% Get the OFDM info
ofdmInfo = wlanVHTOFDMInfo('VHT-Data',cfgVHT);

% Set the sampling rate of the channel
tgacChannel.SampleRate = fs;

% Indices for accessing each field within the time-domain packet
ind = wlanFieldIndices(cfgVHT);
```

Processing SNR Points

For each SNR point a number of packets are tested and the packet error rate calculated.

For each packet the following processing steps occur:

1 A PSDU is created and encoded to create a single packet waveform.

2 The waveform is passed through a different realization of the TGac channel model.

3 AWGN is added to the received waveform to create the desired average SNR per subcarrier after OFDM demodulation. The `comm.AWGNChannel` object is configured to provide the correct SNR. The configuration accounts for normalization within the channel by the number of receive antennas, and the noise energy in unused subcarriers which are removed during OFDM demodulation.

4　The packet is detected.

5　Coarse carrier frequency offset is estimated and corrected.

6　Fine timing synchronization is established. The L-STF, L-LTF and L-SIG samples are provided for fine timing to allow for packet detection at the start or end of the L-STF.

7　Fine carrier frequency offset is estimated and corrected.

8　The VHT-LTF is extracted from the synchronized received waveform. The VHT-LTF is OFDM demodulated and channel estimation is performed.

9　The VHT Data field is extracted from the synchronized received waveform. The PSDU is recovered using the extracted field and the channel estimate.

A `parfor` loop can be used to parallelize processing of the SNR points, therefore for each SNR point an AWGN channel is created and configured with the `comm.AWGNChannel` object. To enable the use of parallel computing for increased speed comment out the `for` statement and uncomment the `parfor` statement below.

```matlab
S = numel(snr);
packetErrorRate = zeros(S,1);
%parfor i = 1:S % Use 'parfor' to speed up the simulation
for i = 1:S      % Use 'for' to debug the simulation
    % Set random substream index per iteration to ensure that each
    % iteration uses a repeatable set of random numbers
    stream = RandStream('combRecursive','Seed',0);
    stream.Substream = i;
    RandStream.setGlobalStream(stream);

    % Create an instance of the AWGN channel per SNR point simulated
    awgnChannel = comm.AWGNChannel;
    awgnChannel.NoiseMethod = 'Signal to noise ratio (SNR)';
    % Normalization
    awgnChannel.SignalPower = 1/tgacChannel.NumReceiveAntennas;
    % Account for energy in nulls
    awgnChannel.SNR = snr(i)-10*log10(ofdmInfo.FFTLength/ofdmInfo.NumTones);

    % Loop to simulate multiple packets
    numPacketErrors = 0;
    numPkt = 1; % Index of packet transmitted
    while numPacketErrors<=maxNumErrors && numPkt<=maxNumPackets
        % Generate a packet waveform
        txPSDU = randi([0 1],cfgVHT.PSDULength*8,1); % PSDULength in bytes
        tx = wlanWaveformGenerator(txPSDU,cfgVHT);

        % Add trailing zeros to allow for channel delay
        tx = [tx; zeros(50,cfgVHT.NumTransmitAntennas)]; %#ok<AGROW>

        % Pass the waveform through the fading channel model
        reset(tgacChannel); % Reset channel for different realization
        rx = tgacChannel(tx);

        % Add noise
        rx = awgnChannel(rx);

        % Packet detect and determine coarse packet offset
        coarsePktOffset = wlanPacketDetect(rx,cfgVHT.ChannelBandwidth);
        if isempty(coarsePktOffset) % If empty no L-STF detected; packet error
            numPacketErrors = numPacketErrors+1;
```

```
        numPkt = numPkt+1;
        continue; % Go to next loop iteration
    end

    % Extract L-STF and perform coarse frequency offset correction
    lstf = rx(coarsePktOffset+(ind.LSTF(1):ind.LSTF(2)),:);
    coarseFreqOff = wlanCoarseCFOEstimate(lstf,cfgVHT.ChannelBandwidth);
    rx = helperFrequencyOffset(rx,fs,-coarseFreqOff);

    % Extract the non-HT fields and determine fine packet offset
    nonhtfields = rx(coarsePktOffset+(ind.LSTF(1):ind.LSIG(2)),:);
    finePktOffset = wlanSymbolTimingEstimate(nonhtfields,...
                cfgVHT.ChannelBandwidth);

    % Determine final packet offset
    pktOffset = coarsePktOffset+finePktOffset;

    % If packet detected outwith the range of expected delays from the
    % channel modeling; packet error
    if pktOffset>50
        numPacketErrors = numPacketErrors+1;
        numPkt = numPkt+1;
        continue; % Go to next loop iteration
    end

    % Extract L-LTF and perform fine frequency offset correction
    lltf = rx(pktOffset+(ind.LLTF(1):ind.LLTF(2)),:);
    fineFreqOff = wlanFineCFOEstimate(lltf,cfgVHT.ChannelBandwidth);
    rx = helperFrequencyOffset(rx,fs,-fineFreqOff);

    % Extract VHT-LTF samples from the waveform, demodulate and perform
    % channel estimation
    vhtltf = rx(pktOffset+(ind.VHTLTF(1):ind.VHTLTF(2)),:);
    vhtltfDemod = wlanVHTLTFDemodulate(vhtltf,cfgVHT);

    % Get single stream channel estimate
    chanEstSSPilots = vhtSingleStreamChannelEstimate(vhtltfDemod,cfgVHT);

    % Channel estimate
    chanEst = wlanVHTLTFChannelEstimate(vhtltfDemod,cfgVHT);

    % Extract VHT Data samples from the waveform
    vhtdata = rx(pktOffset+(ind.VHTData(1):ind.VHTData(2)),:);

    % Estimate the noise power in VHT data field
    nVarVHT = vhtNoiseEstimate(vhtdata,chanEstSSPilots,cfgVHT);

    % Recover the transmitted PSDU in VHT Data
    rxPSDU = wlanVHTDataRecover(vhtdata,chanEst,nVarVHT,cfgVHT);

    % Determine if any bits are in error, i.e. a packet error
    packetError = any(biterr(txPSDU,rxPSDU));
    numPacketErrors = numPacketErrors+packetError;
    numPkt = numPkt+1;
end

% Calculate packet error rate (PER) at SNR point
packetErrorRate(i) = numPacketErrors/(numPkt-1);
```

```
    disp(['SNR ' num2str(snr(i)) ' completed after ' ...
        num2str(numPkt-1) ' packets, PER: ' ...
        num2str(packetErrorRate(i))]);
end
```

```
SNR 40 completed after 11 packets, PER: 1
SNR 45 completed after 15 packets, PER: 0.73333
SNR 50 completed after 100 packets, PER: 0.04
```
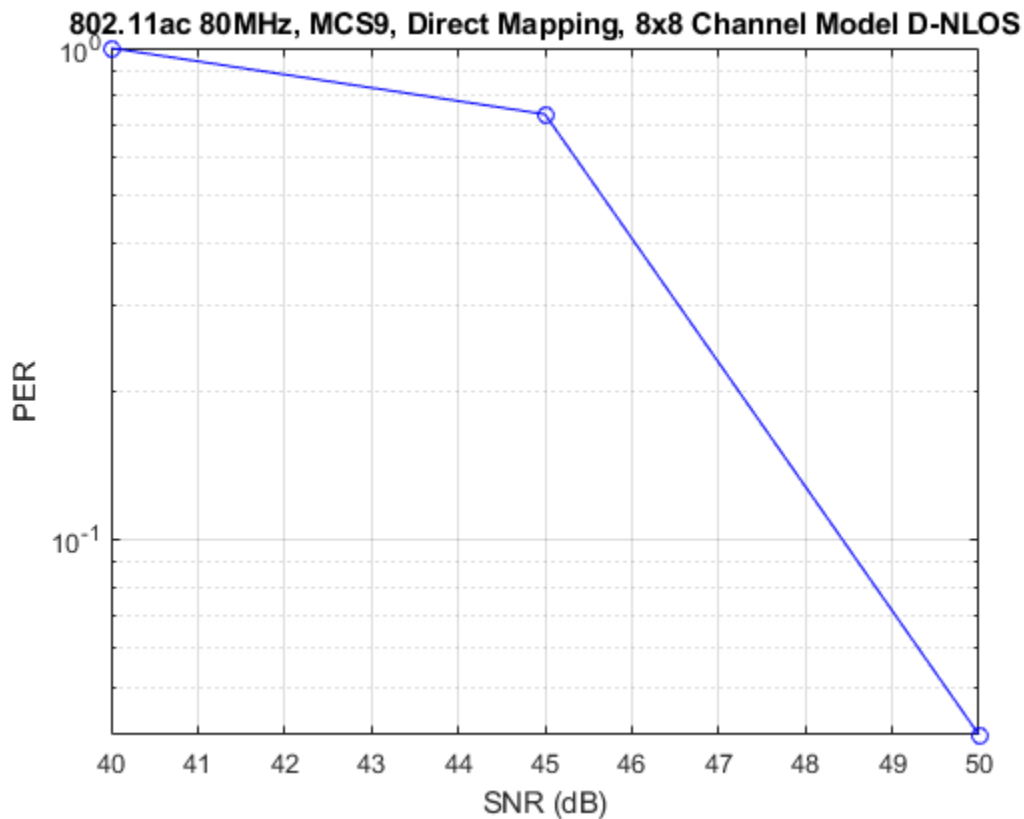
**Plot Packet Error Rate vs SNR Results**

```
figure
semilogy(snr,packetErrorRate,'-ob');
grid on;
xlabel('SNR (dB)');
ylabel('PER');
title('802.11ac 80MHz, MCS9, Direct Mapping, 8x8 Channel Model D-NLOS');
```


Further Exploration

The number of packets tested at each SNR point is controlled by two parameters; `maxNumErrors` and `maxNumPackets`. For meaningful results it is recommend that these values should be larger than those presented in this example. Increasing the number of packets simulated allows the PER under different scenarios to be compared. Try changing the transmission and reception configurations and compare the packet error rate. As an example, the figure below was created by running the example for `maxNumErrors`: 1000 and `maxNumPackets`: 10000.
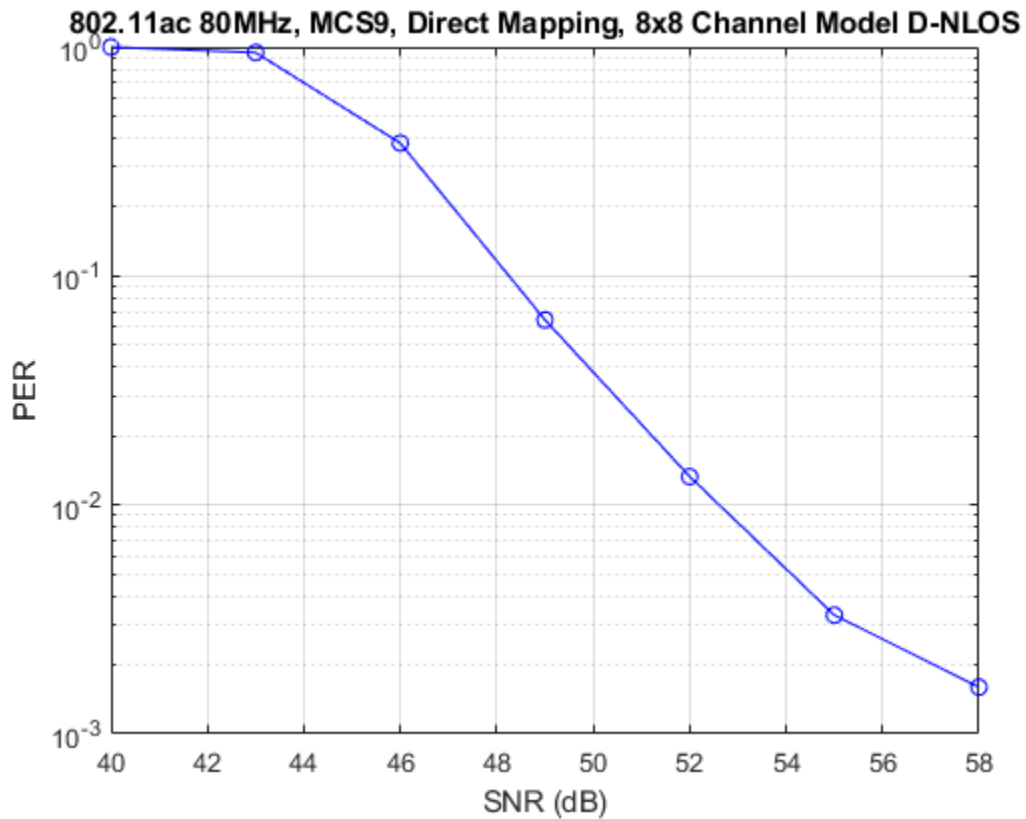
Appendix

This example uses the following helper functions:

- helperFrequencyOffset.m
- vhtNoiseEstimate.m
- vhtSingleStreamChannelEstimate.m

Selected Bibliography

1   IEEE Std 802.11ac™-2013 IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz.

802.11n Packet Error Rate Simulation for 2x2 TGn Channel

This example shows how to measure the packet error rate of an IEEE® 802.11n™ HT link using an end-to-end simulation with a fading TGn channel model and additive white Gaussian noise.

Introduction

In this example an end-to-end simulation is used to determine the packet error rate for an 802.11n HT [1] link with a fading channel at a selection of SNR points. At each SNR point multiple packets are transmitted through a channel, demodulated and the PSDUs recovered. The PSDUs are compared to those transmitted to determine the number of packet errors and hence the packet error rate. Packet detection, timing synchronization, carrier frequency offset correction and phase tracking are performed by the receiver. The processing for each packet is summarized in the following diagram.
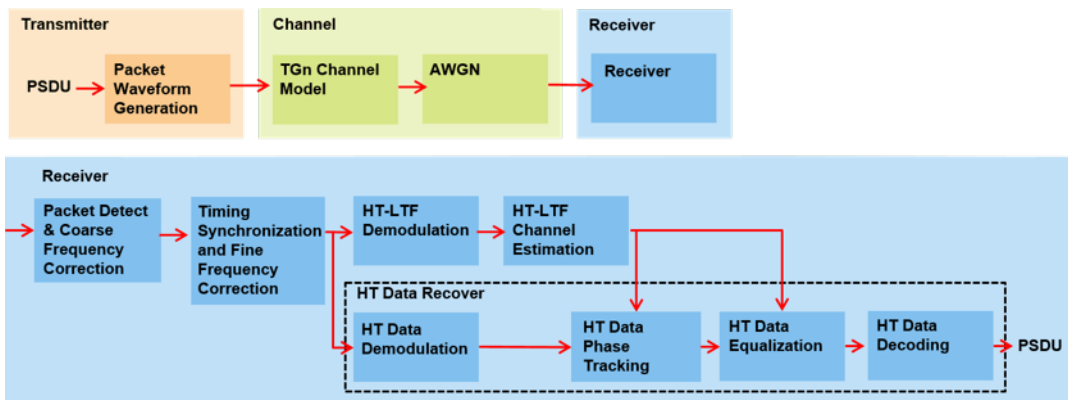
This example also demonstrates how a `parfor` loop can be used instead of a `for` loop when simulating each SNR point to speed up a simulation. The `parfor` function, as part of the Parallel Computing Toolbox™, executes processing for each SNR in parallel to reduce the total simulation time.

Waveform Configuration

An 802.11n HT transmission is simulated in this example. The HT format configuration object, `wlanHTConfig`, contains the format specific configuration of the transmission. The properties of the object contain the configuration. In this example the object is configured for a 20 MHz channel bandwidth, 2 transmit antennas, 2 space time streams and no space time block coding.

```
% Create a format configuration object for a 2-by-2 HT transmission
cfgHT = wlanHTConfig;
cfgHT.ChannelBandwidth = 'CBW20'; % 20 MHz channel bandwidth
cfgHT.NumTransmitAntennas = 2;    % 2 transmit antennas
cfgHT.NumSpaceTimeStreams = 2;    % 2 space-time streams
cfgHT.PSDULength = 1000;          % PSDU length in bytes
cfgHT.MCS = 15;                   % 2 spatial streams, 64-QAM rate-5/6
cfgHT.ChannelCoding = 'BCC';      % BCC channel coding
```

Channel Configuration

In this example a TGn N-LOS channel model is used with delay profile Model-B. For Model-B when the distance between transmitter and receiver is greater than or equal to five meters, the model is NLOS. This is described further in `wlanTGnChannel`.

```
% Create and configure the channel
tgnChannel = wlanTGnChannel;
tgnChannel.DelayProfile = 'Model-B';
tgnChannel.NumTransmitAntennas = cfgHT.NumTransmitAntennas;
tgnChannel.NumReceiveAntennas = 2;
tgnChannel.TransmitReceiveDistance = 10; % Distance in meters for NLOS
tgnChannel.LargeScaleFadingEffect = 'None';
```

Simulation Parameters

For each SNR point in the vector `snr` a number of packets are generated, passed through a channel and demodulated to determine the packet error rate.

```
snr = 25:10:45;
```

The number of packets tested at each SNR point is controlled by two parameters:

1 `maxNumPEs` is the maximum number of packet errors simulated at each SNR point. When the number of packet errors reaches this limit, the simulation at this SNR point is complete.

2 `maxNumPackets` is the maximum number of packets simulated at each SNR point and limits the length of the simulation if the packet error limit is not reached.

The numbers chosen in this example will lead to a very short simulation. For meaningful results we recommend increasing the numbers.

```
maxNumPEs = 10; % The maximum number of packet errors at an SNR point
maxNumPackets = 100; % Maximum number of packets at an SNR point
```

Set the remaining variables for the simulation.

```
% Get the baseband sampling rate
fs = wlanSampleRate(cfgHT);

% Get the OFDM info
ofdmInfo = wlanHTOFDMInfo('HT-Data',cfgHT);

% Set the sampling rate of the channel
tgnChannel.SampleRate = fs;

% Indices for accessing each field within the time-domain packet
ind = wlanFieldIndices(cfgHT);
```

Processing SNR Points

For each SNR point a number of packets are tested and the packet error rate calculated.

For each packet the following processing steps occur:

1 A PSDU is created and encoded to create a single packet waveform.

2 The waveform is passed through a different realization of the TGn channel model.

3 AWGN is added to the received waveform to create the desired average SNR per subcarrier after OFDM demodulation. The `comm.AWGNChannel` object is configured to provide the correct SNR. The configuration accounts for normalization within the channel by the number of receive antennas, and the noise energy in unused subcarriers which are removed during OFDM demodulation.

4   The packet is detected.

5   Coarse carrier frequency offset is estimated and corrected.

6   Fine timing synchronization is established. The L-STF, L-LTF and L-SIG samples are provided for fine timing to allow for packet detection at the start or end of the L-STF.

7   Fine carrier frequency offset is estimated and corrected.

8   The HT-LTF is extracted from the synchronized received waveform. The HT-LTF is OFDM demodulated and channel estimation is performed.

9   The HT Data field is extracted from the synchronized received waveform. The PSDU is recovered using the extracted field and the channel estimate.

A `parfor` loop can be used to parallelize processing of the SNR points, therefore for each SNR point an AWGN channel is created and configured with the `comm.AWGNChannel` object. To enable the use of parallel computing for increased speed comment out the 'for' statement and uncomment the 'parfor' statement below.

```matlab
S = numel(snr);
packetErrorRate = zeros(S,1);
%parfor i = 1:S % Use 'parfor' to speed up the simulation
for i = 1:S % Use 'for' to debug the simulation
    % Set random substream index per iteration to ensure that each
    % iteration uses a repeatable set of random numbers
    stream = RandStream('combRecursive','Seed',0);
    stream.Substream = i;
    RandStream.setGlobalStream(stream);

    % Create an instance of the AWGN channel per SNR point simulated
    awgnChannel = comm.AWGNChannel;
    awgnChannel.NoiseMethod = 'Signal to noise ratio (SNR)';
    % Normalization
    awgnChannel.SignalPower = 1/tgnChannel.NumReceiveAntennas;
    % Account for energy in nulls
    awgnChannel.SNR = snr(i)-10*log10(ofdmInfo.FFTLength/ofdmInfo.NumTones);

    % Loop to simulate multiple packets
    numPacketErrors = 0;
    n = 1; % Index of packet transmitted
    while numPacketErrors<=maxNumPEs && n<=maxNumPackets
        % Generate a packet waveform
        txPSDU = randi([0 1],cfgHT.PSDULength*8,1); % PSDULength in bytes
        tx = wlanWaveformGenerator(txPSDU,cfgHT);

        % Add trailing zeros to allow for channel filter delay
        tx = [tx; zeros(15,cfgHT.NumTransmitAntennas)]; %#ok<AGROW>

        % Pass the waveform through the TGn channel model
        reset(tgnChannel); % Reset channel for different realization
        rx = tgnChannel(tx);

        % Add noise
        rx = awgnChannel(rx);

        % Packet detect and determine coarse packet offset
        coarsePktOffset = wlanPacketDetect(rx,cfgHT.ChannelBandwidth);
        if isempty(coarsePktOffset) % If empty no L-STF detected; packet error
            numPacketErrors = numPacketErrors+1;
```

```matlab
            n = n+1;
            continue; % Go to next loop iteration
        end

        % Extract L-STF and perform coarse frequency offset correction
        lstf = rx(coarsePktOffset+(ind.LSTF(1):ind.LSTF(2)),:);
        coarseFreqOff = wlanCoarseCFOEstimate(lstf,cfgHT.ChannelBandwidth);
        rx = helperFrequencyOffset(rx,fs,-coarseFreqOff);

        % Extract the non-HT fields and determine fine packet offset
        nonhtfields = rx(coarsePktOffset+(ind.LSTF(1):ind.LSIG(2)),:);
        finePktOffset = wlanSymbolTimingEstimate(nonhtfields,...
            cfgHT.ChannelBandwidth);

        % Determine final packet offset
        pktOffset = coarsePktOffset+finePktOffset;

        % If packet detected outwith the range of expected delays from the
        % channel modeling; packet error
        if pktOffset>15
            numPacketErrors = numPacketErrors+1;
            n = n+1;
            continue; % Go to next loop iteration
        end

        % Extract L-LTF and perform fine frequency offset correction
        lltf = rx(pktOffset+(ind.LLTF(1):ind.LLTF(2)),:);
        fineFreqOff = wlanFineCFOEstimate(lltf,cfgHT.ChannelBandwidth);
        rx = helperFrequencyOffset(rx,fs,-fineFreqOff);

        % Extract HT-LTF samples from the waveform, demodulate and perform
        % channel estimation
        htltf = rx(pktOffset+(ind.HTLTF(1):ind.HTLTF(2)),:);
        htltfDemod = wlanHTLTFDemodulate(htltf,cfgHT);
        chanEst = wlanHTLTFChannelEstimate(htltfDemod,cfgHT);

        % Extract HT Data samples from the waveform
        htdata = rx(pktOffset+(ind.HTData(1):ind.HTData(2)),:);

        % Estimate the noise power in HT data field
        nVarHT = htNoiseEstimate(htdata,chanEst,cfgHT);

        % Recover the transmitted PSDU in HT Data
        rxPSDU = wlanHTDataRecover(htdata,chanEst,nVarHT,cfgHT);

        % Determine if any bits are in error, i.e. a packet error
        packetError = any(biterr(txPSDU,rxPSDU));
        numPacketErrors = numPacketErrors+packetError;
        n = n+1;
    end

    % Calculate packet error rate (PER) at SNR point
    packetErrorRate(i) = numPacketErrors/(n-1);
    disp(['SNR ' num2str(snr(i))...
        ' completed after '  num2str(n-1) ' packets,'...
        ' PER: ' num2str(packetErrorRate(i))]);
end
```
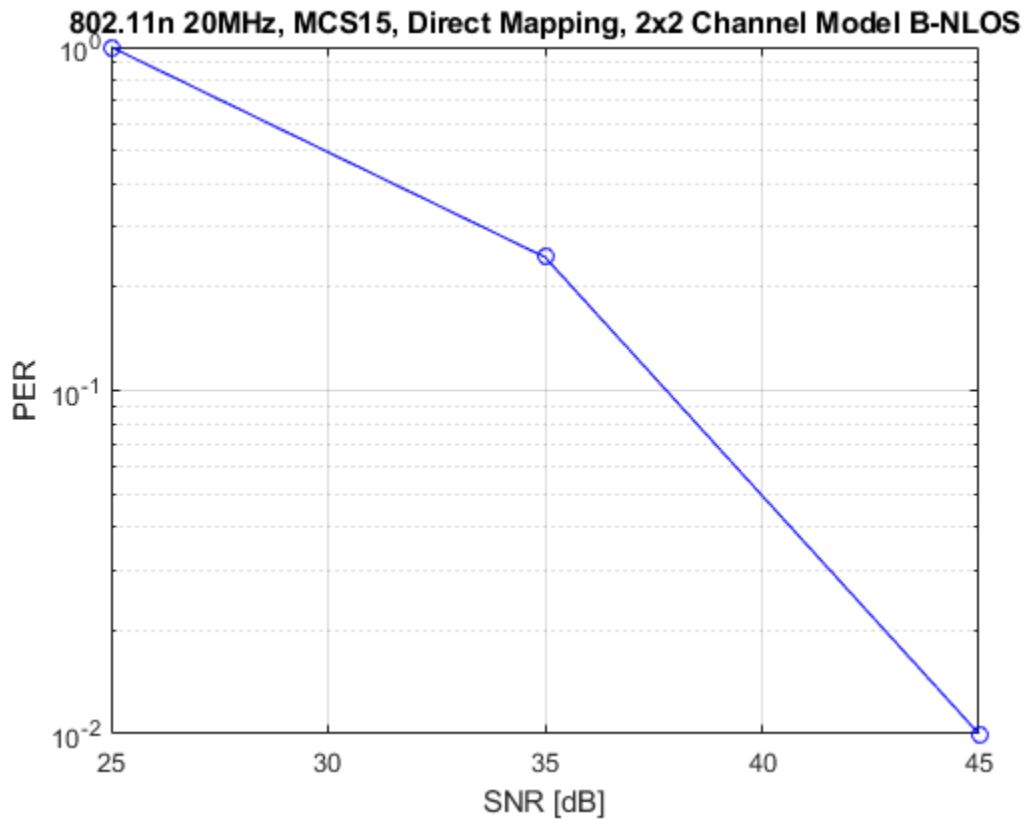
```
SNR 25 completed after 11 packets, PER: 1
SNR 35 completed after 45 packets, PER: 0.24444
SNR 45 completed after 100 packets, PER: 0.01
```
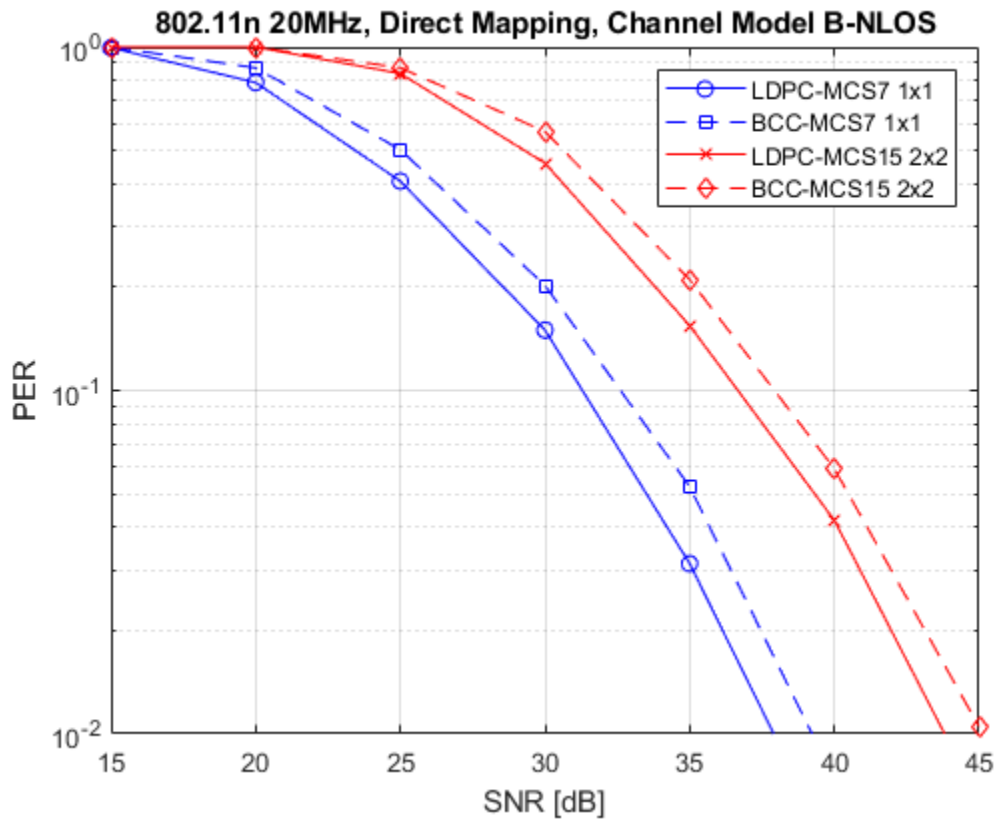
Plot Packet Error Rate vs SNR Results

```
figure;
semilogy(snr,packetErrorRate,'-ob');
grid on;
xlabel('SNR [dB]');
ylabel('PER');
title('802.11n 20MHz, MCS15, Direct Mapping, 2x2 Channel Model B-NLOS');
```


Further Exploration

The number of packets tested at each SNR point is controlled by two parameters; `maxNumPEs` and `maxNumPackets`. For meaningful results it is recommended that these values should be larger than those presented in this example. Increasing the number of packets simulated allows the PER under different scenarios to be compared. Try changing the transmit encoding scheme to LDPC and compare the packet error rate. As an example, the figure below was created by running the example for `maxNumPEs`: 200 and `maxNumPackets`: 10000, with four different configurations; 1x1 and 2x2 with BCC and LDPC encoding.

802.11n 20MHz, Direct Mapping, Channel Model B-NLOS

Legend:
- LDPC-MCS7 1x1
- BCC-MCS7 1x1
- LDPC-MCS15 2x2
- BCC-MCS15 2x2

X-axis: SNR [dB]
Y-axis: PER

Appendix

This example uses the following helper functions:

- helperFrequencyOffset.m
- htNoiseEstimate.m

Selected Bibliography

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

802.11ah Packet Error Rate Simulation for 2x2 TGah Channel

This example shows how to measure the packet error rate of an IEEE® 802.11ah™ S1G short preamble link using an end-to-end simulation with a fading TGah indoor channel model and additive white Gaussian noise.

Introduction

In this example an end-to-end simulation is used to determine the packet error rate for an 802.11ah [1] S1G short preamble link with a fading channel at a selection of SNR points. For each SNR point multiple packets are transmitted through a channel, demodulated and the PSDUs recovered. The PSDUs are compared to those transmitted to determine the number of packet errors and hence the packet error rate. The processing for each packet is summarized in the following diagram.
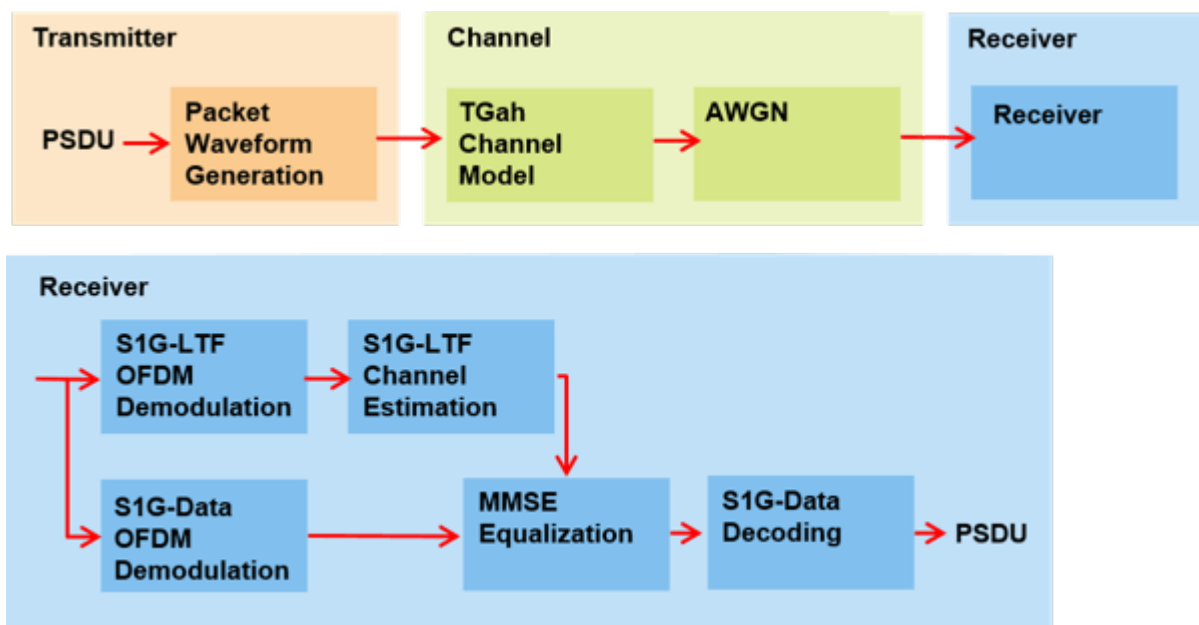
This example simulates the S1G-Short format with no impairment correction apart from channel estimation and equalization. The received signal is synchronized to the start of the packet by compensating for a known delay and the default OFDM demodulation symbol sampling offset. No frequency synchronization is performed. For information on how to automatically detect and synchronize to the received signal, see these examples for 802.11n™ and 802.11ac™.

- "802.11n Packet Error Rate Simulation for 2x2 TGn Channel" on page 5-17
- "802.11ac Packet Error Rate Simulation for 8x8 TGac Channel" on page 5-11

This example also demonstrates how a `parfor` loop can be used instead of the `for` loop when simulating each SNR point to speed up a simulation. `parfor` as part of the Parallel Computing Toolbox™, executes processing for each SNR in parallel to reduce the total simulation time.

Waveform Configuration

A single-user 802.11ah S1G short preamble transmission is simulated in this example. An S1G format configuration object contains the format specific configuration of the transmission. The object is created using the `wlanS1GConfig` function. The properties of the object contain the configuration.

In this example the object is configured for a 2 MHz channel bandwidth, short preamble, 2 transmit antennas, 2 space-time streams, 256 bytes payload, and 64-QAM rate-5/6 (MCS 7).

```matlab
% Create S1G configuration object for single user S1G short preamble
% transmission with 2 transmit antennas and 2 space-time streams
cfgS1G = wlanS1GConfig;
cfgS1G.ChannelBandwidth = 'CBW2'; % 2 MHz channel bandwidth
cfgS1G.Preamble = 'Short';        % Short preamble
cfgS1G.NumTransmitAntennas = 2;   % 2 transmit antennas
cfgS1G.NumSpaceTimeStreams = 2;   % 2 space-time streams
cfgS1G.APEPLength = 256;          % APEP length in bytes
cfgS1G.MCS = 7;                   % 64-QAM rate-5/6
```

Channel Configuration

In this example a TGah N-LOS indoor channel model is used with delay profile Model-B. For Model-B, when the distance between transmitter and receiver is greater than or equal to 5 meters, the model is NLOS. This is described further in `wlanTGahChannel`. A 2x2 MIMO channel is simulated in this example therefore 2 receive antennas are specified.

```matlab
% Create and configure the TGah channel
tgahChannel = wlanTGahChannel;
tgahChannel.DelayProfile = 'Model-B';
tgahChannel.NumTransmitAntennas = cfgS1G.NumTransmitAntennas;
tgahChannel.NumReceiveAntennas = 2;
tgahChannel.TransmitReceiveDistance = 5; % Distance in meters for NLOS
tgahChannel.ChannelBandwidth = cfgS1G.ChannelBandwidth;
tgahChannel.LargeScaleFadingEffect = 'None';
```

Simulation Parameters

For each SNR point in the vector `snr` a number of packets are generated, passed through a channel and demodulated to determine the packet error rate.

```matlab
snr = 25:10:45;
```

The number of packets tested at each SNR point is controlled by two parameters:

1 `maxNumErrors` is the maximum number of packet errors simulated at each SNR point. When the number of packet errors reaches this limit, the simulation at this SNR point is complete.

2 `maxNumPackets` is the maximum number of packets simulated at each SNR point and limits the length of the simulation if the packet error limit is not reached.

The numbers chosen in this example will lead to a very short simulation. For meaningful results we recommend increasing the numbers.

```matlab
maxNumErrors = 1e2;  % The maximum number of packet errors at an SNR point
maxNumPackets = 1e3; % Maximum number of packets at an SNR point
```

Set the remaining variables for the simulation.

```matlab
% Indices for accessing each field within the time-domain packet
fieldInd = wlanFieldIndices(cfgS1G);

% OFDM information
ofdmInfo = wlanS1GOFDMInfo('S1G-Data',cfgS1G);
```

```matlab
% Set the sampling rate of the channel
tgahChannel.SampleRate = wlanSampleRate(cfgS1G);

if ~strcmp(packetFormat(cfgS1G),'S1G-Short')
    error('This example only supports the S1G-Short packet format');
end
```

Processing SNR Points

For each SNR point a number of packets are tested and the packet error rate calculated. At each SNR point:

1 Multiple data packets are transmitted through a 2x2 TGah channel with AWGN.

2 Each packet is time synchronized given a known delay.

3 The S1G-LTF1 and S1G-LTF2N fields are demodulated and channel estimation is performed.

4 The S1G-Data field is extracted from the synchronized received waveform and OFDM demodulated.

5 The data carrying subcarriers are equalized using the channel estimates.

6 The PSDU is recovered using the equalized data subcarriers, noise variance estimate, and channel state information (CSI).

7 The recovered PSDU of each packet is compared to those transmitted to determine the number of packet errors and hence the packet error rate.

A `parfor` loop can be used to parallelize processing of the SNR points, therefore for each SNR point an AWGN channel is created and configured with `comm.AWGNChannel`.

```matlab
packetErrorRate = zeros(numel(snr),1);
%parfor i = 1:numel(snr) % Use 'parfor' to speed up the simulation
for i = 1:numel(snr) % Use 'for' to debug the simulation
    % Set random substream index per iteration to ensure that each
    % iteration uses a repeatable set of random numbers
    stream = RandStream('combRecursive','Seed',0);
    stream.Substream = i;
    RandStream.setGlobalStream(stream);

    % Create an instance of the AWGN channel per SNR point simulated
    awgnChannel = comm.AWGNChannel;
    awgnChannel.NoiseMethod = 'Signal to noise ratio (SNR)';
    % Normalization
    awgnChannel.SignalPower = 1/tgahChannel.NumReceiveAntennas;
    % Account for energy in nulls
    awgnChannel.SNR = snr(i)-10*log10(ofdmInfo.FFTLength/ofdmInfo.NumTones);

    % Loop to simulate multiple packets
    numPacketErrors = 0;
    numPkt = 1; % Index of packet transmitted
    while numPacketErrors<=maxNumErrors && numPkt<=maxNumPackets
        % Generate a packet for 802.11ah short preamble
        txPSDU = randi([0 1],cfgS1G.PSDULength*8,1);
        txWaveform = wlanWaveformGenerator(txPSDU,cfgS1G);

        % Add trailing zeros to allow for channel delay
        tx = [txWaveform; zeros(50,cfgS1G.NumTransmitAntennas)];

        % Pass through fading indoor TGah channel
```

```
        reset(tgahChannel); % Reset channel for different realization
        rx = tgahChannel(tx);

        % Add noise
        rx = awgnChannel(rx);

        % Synchronize
        % The received signal is synchronized to the start of the packet by
        % compensating for a known delay and the default OFDM demodulation
        % symbol sampling offset.
        delay = 4;
        rxSync = rx(delay+1:end,:);

        % LTF demodulation and channel estimation
        % Demodulate S1G-LTF1
        rxLTF1 = rxSync(fieldInd.S1GLTF1(1):fieldInd.S1GLTF1(2),:);
        demodLTF1 = wlanS1GDemodulate(rxLTF1,'S1G-LTF1',cfgS1G);

        % If required, demodulate S1G-LTF2N, and perform channel estimation
        if cfgS1G.NumSpaceTimeStreams>1
            % Use S1G-LTF1 and S1G-LTF2N for channel estimation
            rxLTF2N = rxSync(fieldInd.S1GLTF2N(1):fieldInd.S1GLTF2N(2),:);
            demodLTF2N = wlanS1GDemodulate(rxLTF2N,'S1G-LTF2N',cfgS1G);
            chanEst = s1gLTFChannelEstimate([demodLTF1 demodLTF2N],cfgS1G);
        else
            % Use only S1G-LTF1 for channel estimation
            chanEst = s1gLTFChannelEstimate(demodLTF1,cfgS1G);
        end

        % Noise variance estimate from S1G-LTF1 demodulated symbols
        noiseVarEst = helperNoiseEstimate(demodLTF1);

        % Extract S1G-Data field
        rxData = rxSync(fieldInd.S1GData(1):fieldInd.S1GData(2),:);

        % OFDM demodulation
        demodSym = wlanS1GDemodulate(rxData,'S1G-Data',cfgS1G);

        % Extract data subcarriers from demodulated symbols and channel
        % estimate
        demodDataSym = demodSym(ofdmInfo.DataIndices,:,:);
        chanEstData = chanEst(ofdmInfo.DataIndices,:,:);

        % MMSE frequency domain equalization
        [eqDataSym,csi] = helperSymbolEqualize(demodDataSym,chanEstData,noiseVarEst);

        % Recover PSDU bits
        rxPSDU = s1gDataBitRecover(eqDataSym,noiseVarEst,csi,cfgS1G);

        % Determine if any bits are in error, i.e. a packet error
        packetError = any(biterr(txPSDU,rxPSDU));
        numPacketErrors = numPacketErrors+packetError;
        numPkt = numPkt+1;
    end

    % Compute PER for this SNR point
    packetErrorRate(i) = numPacketErrors/(numPkt-1);
    disp(['SNR ' num2str(snr(i))...
```

```
                ' completed after ' num2str(numPkt-1) ' packets,'...
                ' PER: ' num2str(packetErrorRate(i))]);
    end
```

```
SNR 25 completed after 123 packets, PER: 0.82114
SNR 35 completed after 922 packets, PER: 0.10954
SNR 45 completed after 1000 packets, PER: 0.013
```
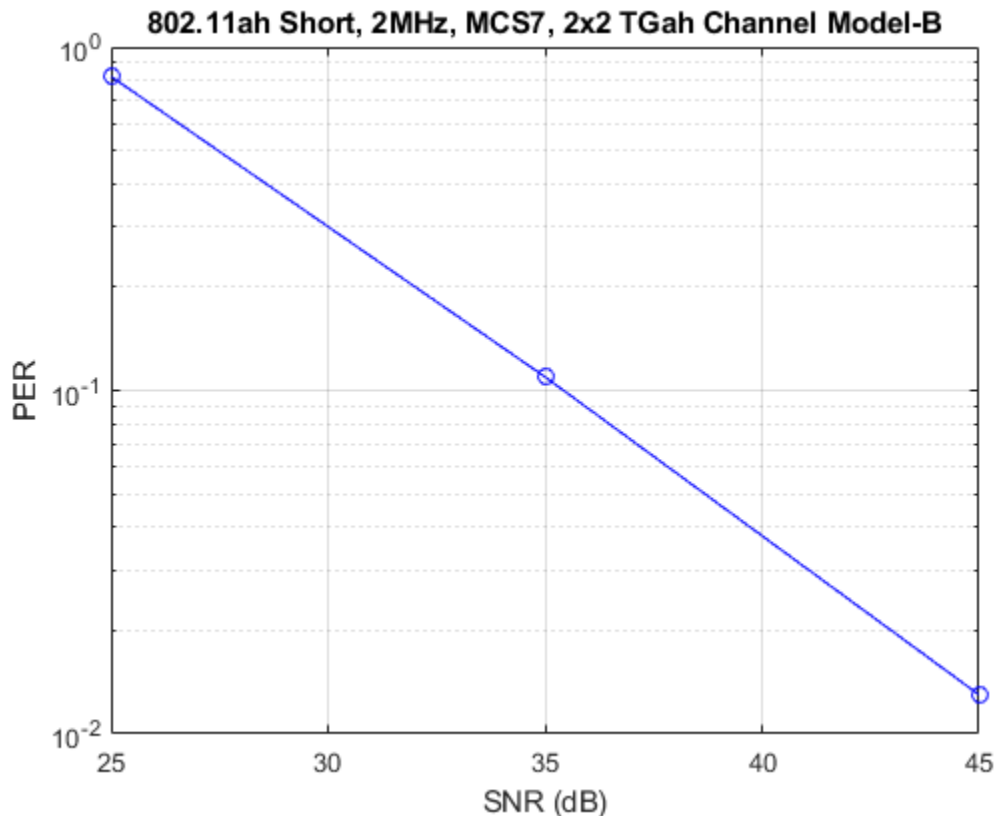
Plot Packet Error Rate vs SNR Results

```
figure;
semilogy(snr,packetErrorRate,'-ob');
grid on;
xlabel('SNR (dB)');
ylabel('PER');
title(['802.11ah ' cfgS1G.Preamble ', ' num2str(cfgS1G.ChannelBandwidth(4:end)) ...
    'MHz, MCS' num2str(cfgS1G.MCS) ', ' ...
    num2str(tgahChannel.NumTransmitAntennas) 'x' num2str(tgahChannel.NumReceiveAntennas) ...
    ' TGah Channel ' num2str(tgahChannel.DelayProfile)]);
```


Further Exploration

The number of packets tested at each SNR point is controlled by two parameters: `maxNumErrors` and `maxNumPackets`. For meaningful results, these values should be larger than those presented in this example. As an example, the figure below was created by running a longer simulation with `maxNumErrors = 1e3` and `maxNumPackets = 1e4`, and the SNR range `snr = 20:8:60`.
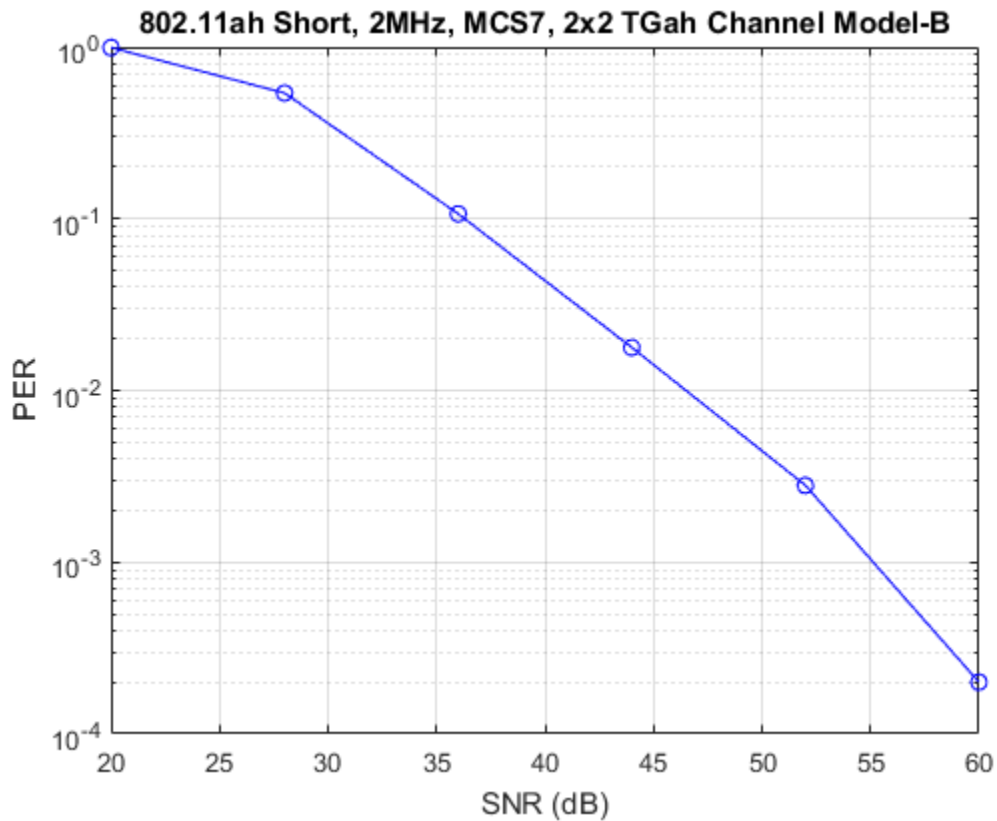
Appendix

This example uses the following helper functions:

- helperNoiseEstimate.m
- helperSymbolEqualize.m
- s1gDataBitRecover.m
- s1gLTFChannelEstimate.m

Selected Bibliography

1 IEEE P802.11ah™ D5.0 IEEE Draft 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 2: Sub 1 GHz License Exempt Operation.

Delay Profile and Fluorescent Lighting Effects

This example demonstrates the impact of changing the TGac delay profile, and it shows how fluorescent lighting affects the time response of the channel.

Delay Profile Effects

Create VHT configuration object. Set the sample rate to 80 MHz.

```
cfgVHT = wlanVHTConfig;
fs = 80e6;
```

Generate random binary data and create a transmit waveform parameterized by the VHT configuration object.

```
d = randi([0 1],8*cfgVHT.PSDULength,1);
testWaveform = wlanWaveformGenerator(d,cfgVHT);
```

Create a TGac channel object. Set the delay profile to `'Model-A'`, which corresponds to flat fading. Disable large-scale fading effects.

```
tgacChan = wlanTGacChannel('SampleRate',fs, ...
    'ChannelBandwidth',cfgVHT.ChannelBandwidth, ...
    'DelayProfile','Model-A', ...
    'LargeScaleFadingEffect','None');
```

Pass the transmitted waveform through the TGac channel.

```
rxModelA = tgacChan(testWaveform);
```

Set the delay profile to Model-C, which corresponds to a multipath channel with 14 distinct paths and a 30 ns RMS delay spread. The maximum delay spread is 200 ns, which corresponds to a coherence bandwidth of 2.5 MHz.

```
release(tgacChan)
tgacChan.DelayProfile = 'Model-C';
```

Pass the waveform through the model-C channel.

```
rxModelC = tgacChan(testWaveform);
```

Create a spectrum analyzer and use it to visualize the spectrum of the received signals.

```
saScope = dsp.SpectrumAnalyzer('SampleRate',fs, ...
    'ShowLegend',true,'ChannelNames',{'Model-A','Model-C'}, ...
    'AveragingMethod','Exponential','ForgettingFactor',0.99);
saScope([rxModelA rxModelC])
```
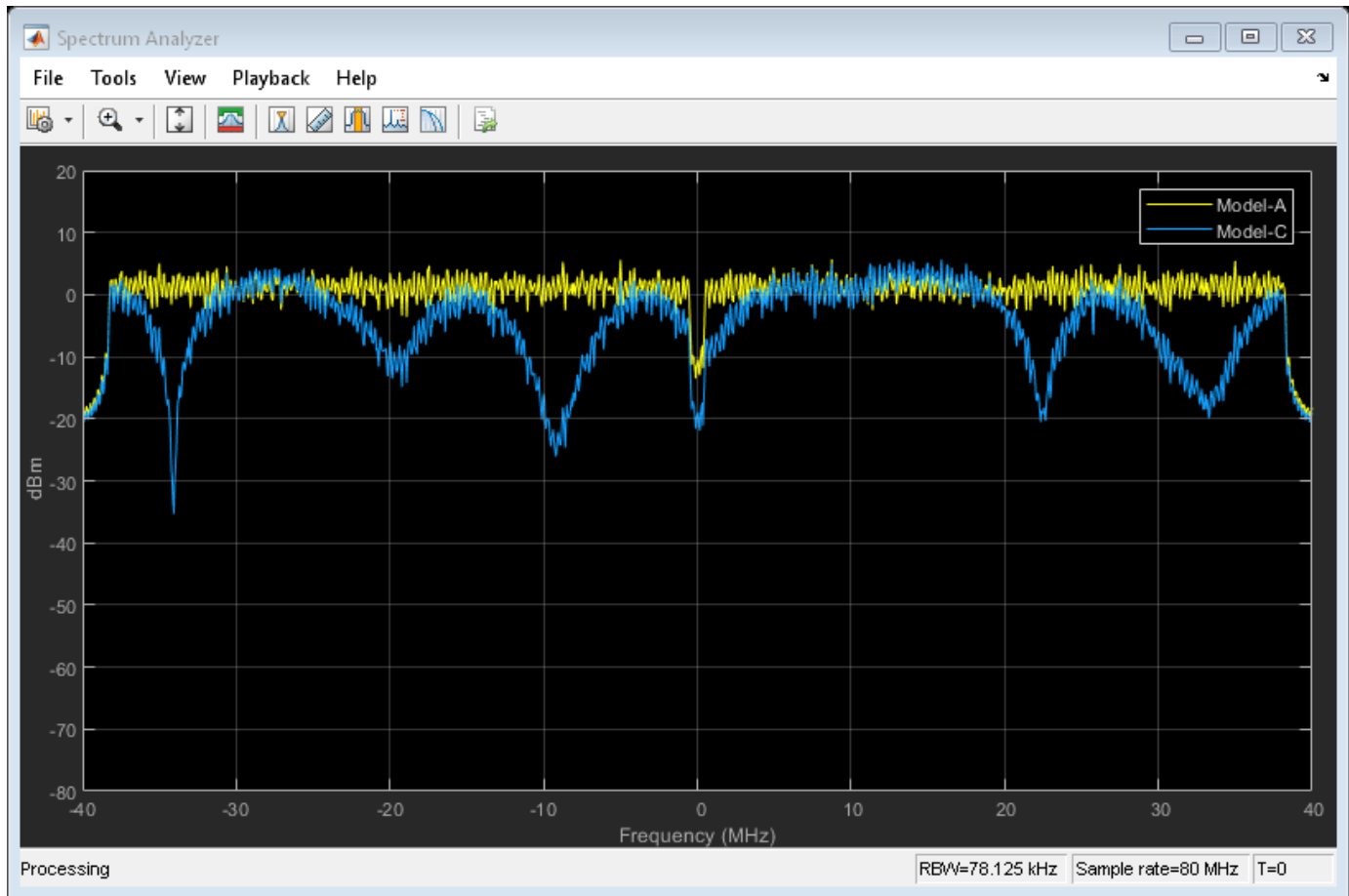
As expected, the frequency response of the model-A signal is flat across the 80 MHz bandwidth. Conversely, the model-C frequency response varies because its coherence bandwidth is much smaller than the channel bandwidth.

Fluorescent Effects

Release the TGac channel, and set its delay profile to `'Model-D'`. Disable the fluorescent lighting effect.

```
release(tgacChan)
tgacChan.DelayProfile = 'Model-D';
tgacChan.FluorescentEffect = false;
```

To better illustrate the Doppler effects of fluorescent lighting, change the bandwidth and sample rate of the channel. Generate a test waveform of ones.

```
tgacChan.ChannelBandwidth = 'CBW20';
fs = 20e6;
tgacChan.SampleRate = fs;
testWaveform = ones(5e5,1);
```

To ensure repeatability, set the global random number generator to a fixed value.

```
rng(37)
```

Pass the waveform through the TGac channel.

```
rxSig0 = tgacChan(testWaveform);
```

Enable the fluorescent lighting effect. Reset the random number generator, and pass the waveform through the channel.

```
release(tgacChan)
tgacChan.FluorescentEffect = true;
rng(37)
rxSig1 = tgacChan(testWaveform);
```

Determine the time axis and channel filter delay.

```
t = ((1:size(rxSig0,1))'-1)/fs;
fDelay = tgacChan.info.ChannelFilterDelay;
```

Plot the magnitude of the received signals while accounting for the channel filter delay.

```
plot(t(fDelay+1:end),[abs(rxSig0(fDelay+1:end)) abs(rxSig1(fDelay+1:end))])
xlabel('Time (s)')
ylabel('Magnitude (V)')
legend('Fluorescent Off','Fluorescent On','location','best')
```
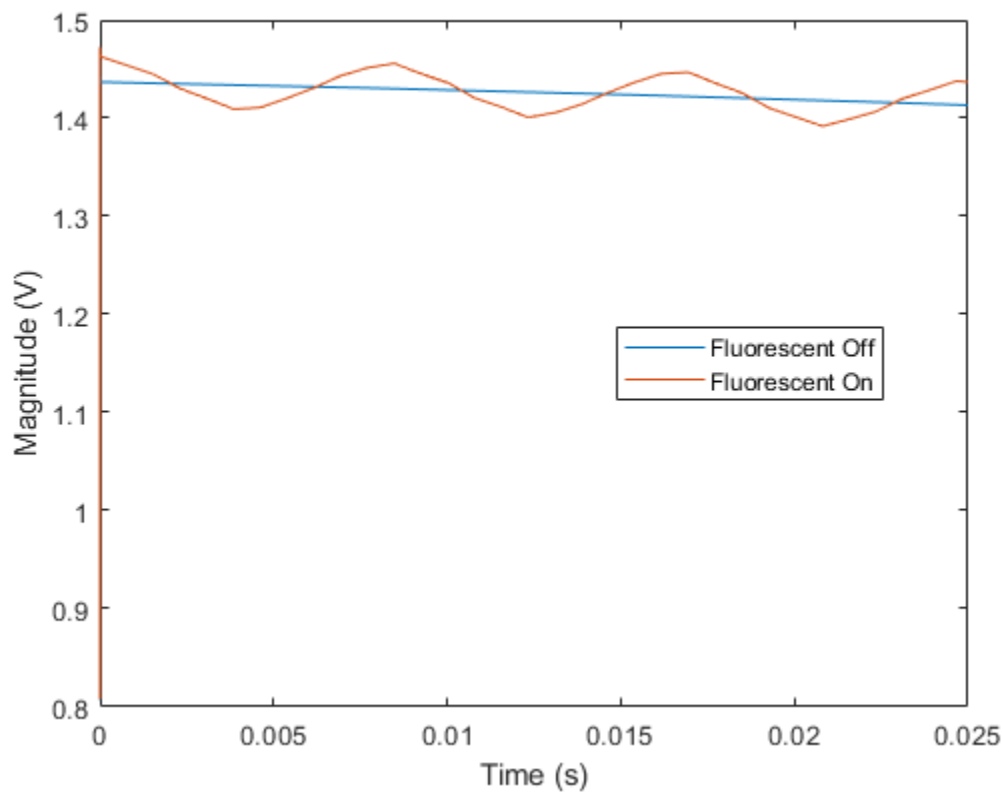
Fluorescent lighting introduces a Doppler component at twice the power line frequency (120 Hz in the U.S.).

Confirm that the peaks are separated by approximately 0.0083 s (inverse of 120 Hz) by measuring distance between the second and third peaks.

```
[~,loc] = findpeaks(abs(rxSig1(1e5:4e5)));
peakTimes = loc/fs;
peakSeparation = diff(peakTimes)
```

peakSeparation = 0.0085

End-to-End Simulation

802.11ax Packet Error Rate Simulation for Single-User Format

This example shows how to measure the packet error rate of an IEEE® 802.11ax™ high efficiency (HE) single user format link.

Introduction

In this example, an end-to-end simulation is used to determine the packet error rate for an 802.11ax [1] single user format link for a selection of SNR points. At each SNR point, multiple packets are transmitted through a noisy TGax indoor channel, demodulated and the PSDUs recovered. The PSDUs are compared to those transmitted to determine the packet error rate. The processing for each packet is summarized in the following diagram.
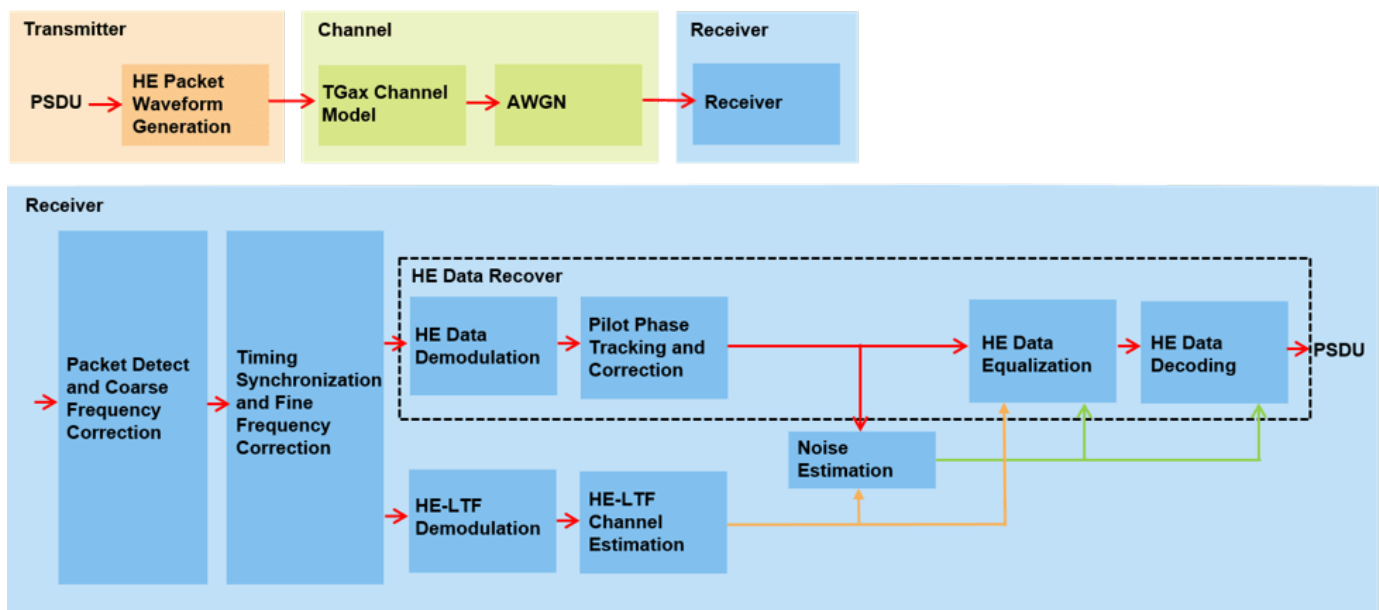
Waveform Configuration

An HE single user (SU) packet is a full-band transmission to a single user. The transmit parameters for the HE SU format are configured using an `wlanHESUConfig` object. The properties of the object contain the configuration. In this example, the object is configured for a 20 MHz channel bandwidth, 2 transmit antennas, 2 space-time streams, no space time block coding and 16-QAM rate-1/2 (MCS 3).

```
cfgHE = wlanHESUConfig;
cfgHE.ChannelBandwidth = 'CBW20';      % Channel bandwidth
cfgHE.NumSpaceTimeStreams = 2;         % Number of space-time streams
cfgHE.NumTransmitAntennas = 2;         % Number of transmit antennas
cfgHE.APEPLength = 1e3;                % Payload length in bytes
cfgHE.ExtendedRange = false;           % Do not use extended range format
cfgHE.Upper106ToneRU = false;          % Do not use upper 106 tone RU
cfgHE.PreHESpatialMapping = false;     % Spatial mapping of pre-HE fields
cfgHE.GuardInterval = 0.8;             % Guard interval duration
cfgHE.HELTFType = 4;                   % HE-LTF compression mode
cfgHE.ChannelCoding = 'LDPC';          % Channel coding
cfgHE.MCS = 3;                         % Modulation and coding scheme
```

Channel Configuration

In this example, a TGax NLOS indoor channel model is used with delay profile Model-B. Model-B is considered NLOS when the distance between transmitter and receiver is greater than or equal to 5 meters. This is described further in `wlanTGaxChannel`. A 2x2 MIMO channel is simulated in this example.

```
% Create and configure the TGax channel
chanBW = cfgHE.ChannelBandwidth;
tgaxChannel = wlanTGaxChannel;
tgaxChannel.DelayProfile = 'Model-B';
tgaxChannel.NumTransmitAntennas = cfgHE.NumTransmitAntennas;
tgaxChannel.NumReceiveAntennas = 2;
tgaxChannel.TransmitReceiveDistance = 5; % Distance in meters for NLOS
tgaxChannel.ChannelBandwidth = chanBW;
tgaxChannel.LargeScaleFadingEffect = 'None';
fs = wlanSampleRate(cfgHE);
tgaxChannel.SampleRate = fs;
```

Simulation Parameters

For each SNR point (dB) in the `snr` vector a number of packets are generated, passed through a channel and demodulated to determine the packet error rate.

```
snr = 10:5:35;
```

The number of packets tested at each SNR point is controlled by two parameters:

1 `maxNumErrors` is the maximum number of packet errors simulated at each SNR point. When the number of packet errors reaches this limit, the simulation at this SNR point is complete.

2 `maxNumPackets` is the maximum number of packets simulated at each SNR point and limits the length of the simulation if the packet error limit is not reached.

The numbers chosen in this example will lead to a very short simulation. For statistically meaningful results we recommend increasing these numbers.

```
maxNumErrors = 10;   % The maximum number of packet errors at an SNR point
maxNumPackets = 100; % The Maximum number of packets at an SNR point
```

Processing SNR Points

For each SNR point a number of packets are tested and the packet error rate calculated. The pre-HE preamble of 802.11ax is backwards compatible with 802.11ac™, therefore in this example the front-end synchronization components for a VHT waveform are used to synchronize the HE waveform at the receiver. For each packet the following processing steps occur:

1 A PSDU is created and encoded to create a single packet waveform.

2 The waveform is passed through an indoor TGax channel model. Different channel realizations are modeled for different packets.

3 AWGN is added to the received waveform to create the desired average SNR per subcarrier after OFDM demodulation. The `comm.AWGNChannel` object is configured to provide the correct SNR. The configuration accounts for normalization within the channel by the number of receive antennas, and the noise energy in unused subcarriers which are removed during OFDM demodulation.

4 The packet is detected.

5 Coarse carrier frequency offset is estimated and corrected.

6 Fine timing synchronization is established. The L-STF, L-LTF and L-SIG samples are provided for fine timing to allow for packet detection at the start or end of the L-STF.

7 Fine carrier frequency offset is estimated and corrected.

8 The HE-LTF is extracted from the synchronized received waveform. The HE-LTF is OFDM demodulated and channel estimation is performed.

9 The data field is extracted from the synchronized received waveform and OFDM demodulated.

10 Common phase error pilot tracking is performed to track any residual carrier frequency offset.

11 Noise estimation is performed using the demodulated data field pilots and single-stream channel estimate at pilot subcarriers.

12 The phase corrected OFDM symbols are equalized with the channel estimate.

13 The equalized symbols are demodulated and decoded to recover the PSDU.

A `parfor` loop can be used to parallelize processing of the SNR points, therefore for each SNR point an AWGN channel is created and configured by using the `comm.AWGNChannel` object. To enable the use of parallel computing for increased speed comment out the 'for' statement and uncomment the 'parfor' statement below.

```
numSNR = numel(snr); % Number of SNR points
packetErrorRate = zeros(1,numSNR);

% Get occupied subcarrier indices and OFDM parameters
ofdmInfo = wlanHEOFDMInfo('HE-Data',cfgHE);

% Indices to extract fields from the PPDU
ind = wlanFieldIndices(cfgHE);

%parfor isnr = 1:numSNR % Use 'parfor' to speed up the simulation
for isnr = 1:numSNR
    % Set random substream index per iteration to ensure that each
    % iteration uses a repeatable set of random numbers
    stream = RandStream('combRecursive','Seed',99);
    stream.Substream = isnr;
    RandStream.setGlobalStream(stream);

    % Create an instance of the AWGN channel per SNR point simulated
    awgnChannel = comm.AWGNChannel;
    awgnChannel.NoiseMethod = 'Signal to noise ratio (SNR)';
    awgnChannel.SignalPower = 1/tgaxChannel.NumReceiveAntennas;
    % Account for noise energy in nulls so the SNR is defined per
    % active subcarrier
    awgnChannel.SNR = snr(isnr)-10*log10(ofdmInfo.FFTLength/ofdmInfo.NumTones);

    % Loop to simulate multiple packets
    numPacketErrors = 0;
    numPkt = 1; % Index of packet transmitted
    while numPacketErrors<=maxNumErrors && numPkt<=maxNumPackets
        % Generate a packet with random PSDU
        psduLength = getPSDULength(cfgHE); % PSDU length in bytes
        txPSDU = randi([0 1],psduLength*8,1);
        tx = wlanWaveformGenerator(txPSDU,cfgHE);
```

```matlab
% Add trailing zeros to allow for channel delay
txPad = [tx; zeros(50,cfgHE.NumTransmitAntennas)];

% Pass through a fading indoor TGax channel
reset(tgaxChannel); % Reset channel for different realization
rx = tgaxChannel(txPad);

% Pass the waveform through AWGN channel
rx = awgnChannel(rx);

% Packet detect and determine coarse packet offset
coarsePktOffset = wlanPacketDetect(rx,chanBW);
if isempty(coarsePktOffset) % If empty no L-STF detected; packet error
    numPacketErrors = numPacketErrors+1;
    numPkt = numPkt+1;
    continue; % Go to next loop iteration
end

% Extract L-STF and perform coarse frequency offset correction
lstf = rx(coarsePktOffset+(ind.LSTF(1):ind.LSTF(2)),:);
coarseFreqOff = wlanCoarseCFOEstimate(lstf,chanBW);
rx = helperFrequencyOffset(rx,fs,-coarseFreqOff);

% Extract the non-HT fields and determine fine packet offset
nonhtfields = rx(coarsePktOffset+(ind.LSTF(1):ind.LSIG(2)),:);
finePktOffset = wlanSymbolTimingEstimate(nonhtfields,chanBW);

% Determine final packet offset
pktOffset = coarsePktOffset+finePktOffset;

% If packet detected outwith the range of expected delays from
% the channel modeling; packet error
if pktOffset>50
    numPacketErrors = numPacketErrors+1;
    numPkt = numPkt+1;
    continue; % Go to next loop iteration
end

% Extract L-LTF and perform fine frequency offset correction
rxLLTF = rx(pktOffset+(ind.LLTF(1):ind.LLTF(2)),:);
fineFreqOff = wlanFineCFOEstimate(rxLLTF,chanBW);
rx = helperFrequencyOffset(rx,fs,-fineFreqOff);

% HE-LTF demodulation and channel estimation
rxHELTF = rx(pktOffset+(ind.HELTF(1):ind.HELTF(2)),:);
heltfDemod = wlanHEDemodulate(rxHELTF,'HE-LTF',cfgHE);
[chanEst,pilotEst] = heLTFChannelEstimate(heltfDemod,cfgHE);

% Data demodulate
rxData = rx(pktOffset+(ind.HEData(1):ind.HEData(2)),:);
demodSym = wlanHEDemodulate(rxData,'HE-Data',cfgHE);

% Pilot phase tracking
demodSym = heCommonPhaseErrorTracking(demodSym,chanEst,cfgHE);

% Estimate noise power in HE fields
nVarEst = heNoiseEstimate(demodSym(ofdmInfo.PilotIndices,:,:),pilotEst,cfgHE);
```

```
        % Extract data subcarriers from demodulated symbols and channel
        % estimate
        demodDataSym = demodSym(ofdmInfo.DataIndices,:,:);
        chanEstData = chanEst(ofdmInfo.DataIndices,:,:);

        % Equalization and STBC combining
        [eqDataSym,csi] = heEqualizeCombine(demodDataSym,chanEstData,nVarEst,cfgHE);

        % Recover data
        rxPSDU = wlanHEDataBitRecover(eqDataSym,nVarEst,csi,cfgHE,'LDPCDecodingMethod','layered-
        % Determine if any bits are in error, i.e. a packet error
        packetError = ~isequal(txPSDU,rxPSDU);
        numPacketErrors = numPacketErrors+packetError;
        numPkt = numPkt+1;
    end

    % Calculate packet error rate (PER) at SNR point
    packetErrorRate(isnr) = numPacketErrors/(numPkt-1);
    disp(['MCS ' num2str(cfgHE.MCS) ','...
          ' SNR ' num2str(snr(isnr)) ...
          ' completed after ' num2str(numPkt-1) ' packets,'...
          ' PER:' num2str(packetErrorRate(isnr))]);
end

MCS 3, SNR 10 completed after 11 packets, PER:1
MCS 3, SNR 15 completed after 17 packets, PER:0.64706
MCS 3, SNR 20 completed after 52 packets, PER:0.21154
MCS 3, SNR 25 completed after 100 packets, PER:0.02
MCS 3, SNR 30 completed after 100 packets, PER:0
MCS 3, SNR 35 completed after 100 packets, PER:0
```

Plot Packet Error Rate vs SNR

```
figure;
semilogy(snr,packetErrorRate,'-*');
hold on;
grid on;
xlabel('SNR (dB)');
ylabel('PER');
dataStr = arrayfun(@(x)sprintf('MCS %d',x),cfgHE.MCS,'UniformOutput',false);
legend(dataStr);
title(sprintf('PER for HE Channel %s, %s, %s, PSDULength: %d',tgaxChannel.DelayProfile,cfgHE.Chan
```

PER for HE Channel Model-B, CBW20, LDPC, PSDULength: 1000

The number of packets tested at each SNR point is controlled by two parameters: `maxNumErrors` and `maxNumPackets`. For meaningful results, these values should be larger than those presented in this example. As an example, the figure below was created by running a longer simulation with `maxNumErrors`:1e3 and `maxNumPackets`:1e4.

PER for HE Channel Model-B, CBW20, LDPC, PSDULength: 1000

Appendix

This example uses the following helper functions:

- heCommonPhaseErrorTracking.m
- heEqualizeCombine.m
- helperFrequencyOffset.m
- heLTFChannelEstimate.m
- heNoiseEstimate.m

Selected Bibliography

1 IEEE P802.11ax™/D4.1 Draft 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 6: Enhancements for High Efficiency WLAN.

802.11ax Downlink OFDMA and Multi-User MIMO Throughput Simulation

This example shows the transmit and receive processing for an IEEE® 802.11ax™ multi-user downlink transmission over a TGax indoor fading channel. Three transmission modes are simulated: OFDMA, MU-MIMO, and a combination of OFDMA and MU-MIMO.

Introduction

This example simulates a scenario of an access point (AP) transmitting to four stations (STAs) simultaneously using high efficiency (HE) multi-user (MU) format packets as specified in IEEE P802.11ax™/D4.1 [1].
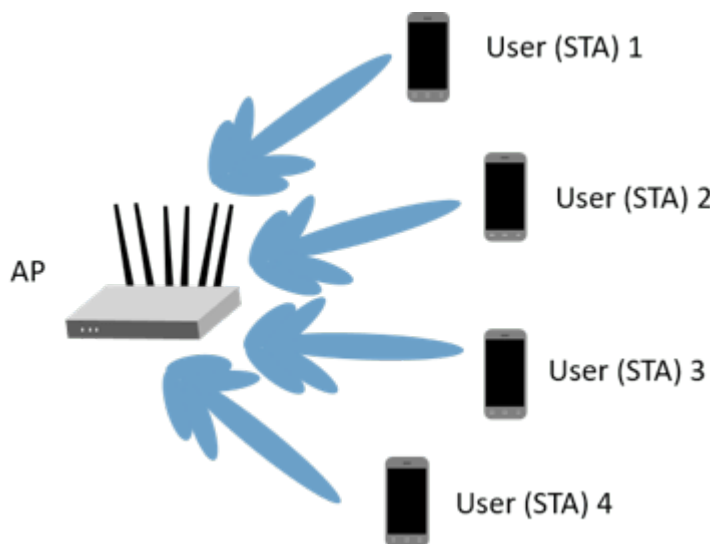
The HE multi-user format can be configured for an OFDMA transmission, a MU-MIMO transmission, or a combination of the two. This flexibility allows an HE-MU packet to transmit to a single user over the whole band, multiple users over different parts of the band (OFDMA), or multiple users over the same part of the band (MU-MIMO).

Three transmission modes are compared for the multi-user downlink scenario:

1 OFDMA - each of the four users is assigned a separate resource unit (RU) and the transmission is beamformed.

2 MU-MIMO - all four users share the full band.

3 Mixed MU-MIMO and OFDMA - Two users share a single RU in a MU-MIMO configuration, and the remaining two users are assigned a single RU each.

For a detailed overview of 802.11ax formats, see the "802.11ax Parameterization for Waveform Generation and Simulation" on page 1-24 example.

For each transmission mode, the AP transmits a burst of 10 packets, and each STA demodulates and decodes the data intended for it. An evolving TGax indoor MIMO channel with AWGN is modeled between the AP and each STA. The raw AP throughput is provided as a metric to compare the transmission modes and is calculated by counting the number of packets transmitted successfully to all STAs. The simulation is repeated for different path losses. In this example, all the transmissions

are beamformed. Therefore, before simulating the data transmission, the channel between the AP and each station is sounded under perfect conditions to obtain channel state information.

Transmission Configuration

An `wlanHEMUConfig` object is used to configure the transmission of an HE-MU packet. Three transmission configuration objects are specified to define the three different AP transmissions:

1   `cfgMUMIMO` is a MU-MIMO configuration which consists of a single 242-tone RU with 4 users. Each user has one space-time stream.

2   `cfgOFDMA` is an OFDMA configuration which consists of four 52-tone RUs, each with one user. Each user has two space-time streams.

3   `cfgMixed` is a mixed OFDMA and MU-MIMO configuration which consists of one 106-tone RU shared by two users, and two 52-tone RUs, each with one user. The MU-MIMO users each have one space-time stream, and the OFDMA users each have two space-time streams.

A 20 MHz channel bandwidth is used for all transmissions. Other transmission parameters such as the APEPLength and MCS are the same for all users in all configurations.

First, the MU-MIMO configuration is defined. The allocation index 195 defines a single 242-tone RU, with four users in MU-MIMO. For a description of selecting an allocation index, see the "802.11ax Parameterization for Waveform Generation and Simulation" on page 1-24 example.

```
% MU-MIMO configuration - 4 users on one 242-tone RU
cfgMUMIMO = wlanHEMUConfig(195);
```

The allocation plot shows a single RU is assigned to all four users.

```
showAllocation(cfgMUMIMO);
```
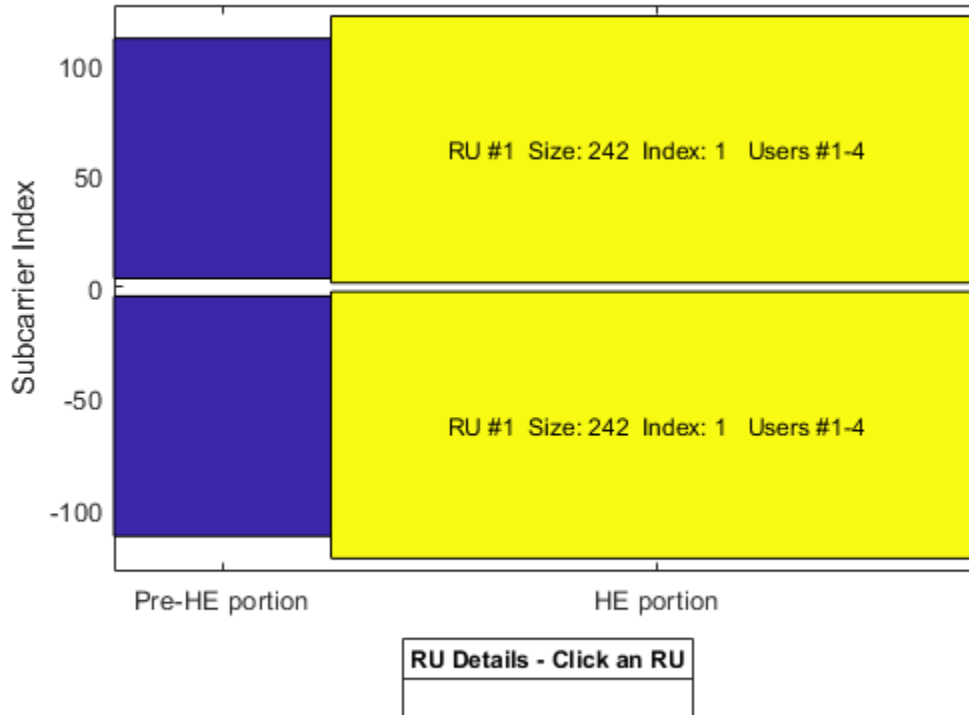
The transmission parameters for each user are now configured.

```
numTx = 6; % Number of transmit antennas
guardInterval = 0.8; % Guard interval in Microseconds

% Configure common parameters for all users
cfgMUMIMO.NumTransmitAntennas = numTx;
cfgMUMIMO.GuardInterval = guardInterval;

% Configure per user parameters
% STA #1
cfgMUMIMO.User{1}.NumSpaceTimeStreams = 1;
cfgMUMIMO.User{1}.MCS = 4;
cfgMUMIMO.User{1}.APEPLength = 1000;
% STA #2
cfgMUMIMO.User{2}.NumSpaceTimeStreams = 1;
cfgMUMIMO.User{2}.MCS = 4;
cfgMUMIMO.User{2}.APEPLength = 1000;
% STA #3
cfgMUMIMO.User{3}.NumSpaceTimeStreams = 1;
cfgMUMIMO.User{3}.MCS = 4;
cfgMUMIMO.User{3}.APEPLength = 1000;
% STA #4
cfgMUMIMO.User{4}.NumSpaceTimeStreams = 1;
cfgMUMIMO.User{4}.MCS = 4;
cfgMUMIMO.User{4}.APEPLength = 1000;
```

Next the OFDMA configuration is defined. The allocation index 112 defines four 52-tone RUs, each serving a single user.

```
% OFDMA configuration - 4 users, each on a 52-tone RU
cfgOFDMA = wlanHEMUConfig(112);
```

The allocation plot shows the four RUs, each with a single user. When comparing this allocation plot to the full band MU-MIMO plot, it is apparent that the total number of subcarriers used (4x52 = 208 subcarriers) is less than the MU-MIMO allocation (242 subcarriers). The fewer number of subcarriers allow guards between each OFDMA user.

```
showAllocation(cfgOFDMA);
```
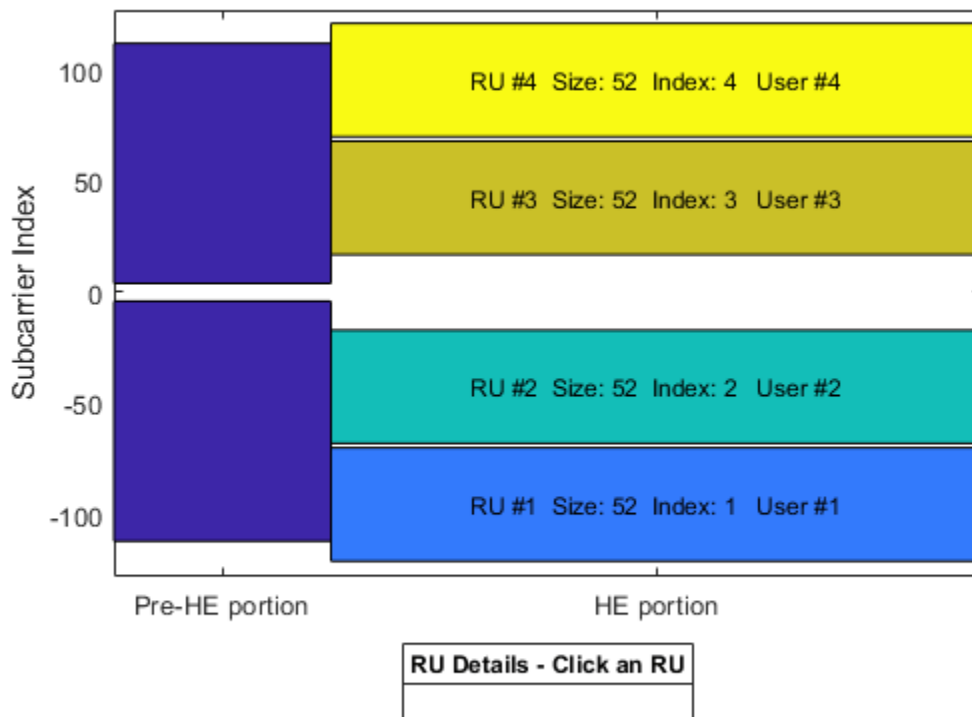

The transmission parameters for each user are now configured.

```
% Configure common parameters for all users
cfgOFDMA.NumTransmitAntennas = numTx;
cfgOFDMA.GuardInterval = guardInterval;

% Configure per user parameters
% STA #1 (RU #1)
cfgOFDMA.User{1}.NumSpaceTimeStreams = 2;
cfgOFDMA.User{1}.MCS = 4;
cfgOFDMA.User{1}.APEPLength = 1000;
% STA #2 (RU #2)
cfgOFDMA.User{2}.NumSpaceTimeStreams = 2;
cfgOFDMA.User{2}.MCS = 4;
```

```
cfgOFDMA.User{2}.APEPLength = 1000;
% STA #3 (RU #3)
cfgOFDMA.User{3}.NumSpaceTimeStreams = 2;
cfgOFDMA.User{3}.MCS = 4;
cfgOFDMA.User{3}.APEPLength = 1000;
% STA #4 (RU #4)
cfgOFDMA.User{4}.NumSpaceTimeStreams = 2;
cfgOFDMA.User{4}.MCS = 4;
cfgOFDMA.User{4}.APEPLength = 1000;
```

Finally, the mixed MU-MIMO and OFDMA configuration is defined. The allocation index 25 defines a 106-tone RU with two users, and two 52-tone RUs, each with one user.

```
% Mixed OFDMA and MU-MIMO configuration
cfgMixed = wlanHEMUConfig(25);
```

The allocation plot shows the three RUs, one with 2 users (MU-MIMO), and the others with one user each (OFDMA).

```
showAllocation(cfgMixed);
```
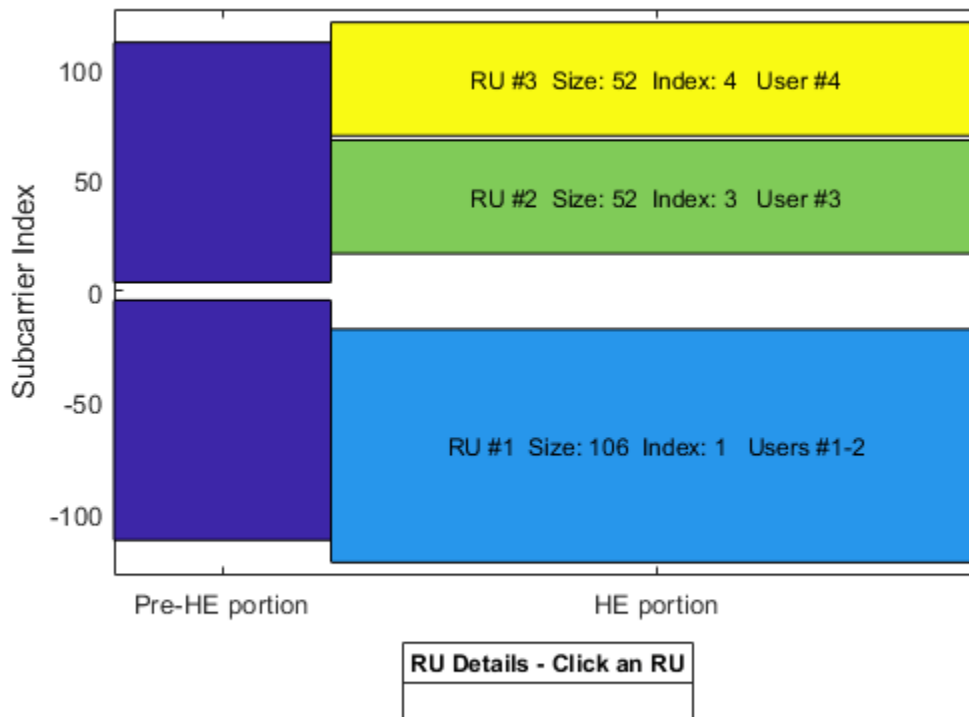

The transmission parameters for each user are now configured.

```
% Configure common parameters for all users
cfgMixed.NumTransmitAntennas = numTx;
cfgMixed.GuardInterval = guardInterval;

% Configure per user parameters
```

```
% RU #1 has two users (MU-MIMO) and a total of 2 STS (1 per user)
% STA #1 (RU #1)
cfgMixed.User{1}.NumSpaceTimeStreams = 1;
cfgMixed.User{1}.MCS = 4;
cfgMixed.User{1}.APEPLength = 1000;
% STA #2 (RU #1)
cfgMixed.User{2}.NumSpaceTimeStreams = 1;
cfgMixed.User{2}.MCS = 4;
cfgMixed.User{2}.APEPLength = 1000;

% The remaining two users are OFDMA
% STA #3 (RU #2)
cfgMixed.User{3}.NumSpaceTimeStreams = 2;
cfgMixed.User{3}.MCS = 4;
cfgMixed.User{3}.APEPLength = 1000;
% STA #4 (RU #3)
cfgMixed.User{4}.NumSpaceTimeStreams = 2;
cfgMixed.User{4}.MCS = 4;
cfgMixed.User{4}.APEPLength = 1000;
```

Channel Model Configuration

A TGax indoor channel model is used in this example. An individual channel is used to simulate the link between the AP and each user. A TGax channel object, `tgaxBase` is created with properties relevant for all users. In this example, the delay profile (Model-D) and number of receive antennas are common for all users. Model-D is considered non-line of sight when the distance between transmitter and receiver is greater than or equal to 10 meters. This is described further in `wlanTGaxChannel`. A fixed seed is used for the channel to allow repeatability.

```
% Create channel configuration common for all users
tgaxBase = wlanTGaxChannel;
tgaxBase.DelayProfile = 'Model-D';     % Delay profile
tgaxBase.NumTransmitAntennas = numTx;  % Number of transmit antennas
tgaxBase.NumReceiveAntennas = 2;       % Each user has two receive antennas
tgaxBase.TransmitReceiveDistance = 10; % Non-line of sight distance
tgaxBase.ChannelBandwidth = cfgMUMIMO.ChannelBandwidth;
tgaxBase.SampleRate = wlanSampleRate(cfgMUMIMO);
% Set a fixed seed for the channel
tgaxBase.RandomStream = 'mt19937ar with seed';
tgaxBase.Seed = 5;
```

Next a channel is created for each user. The channel for each user is a clone of the `tgaxBase`, but with a unique `UserIndex` property, and is stored in a cell array `tgax`. The `UserIndex` property of each individual channel is set to provide a unique channel for each user. The resultant channels are used in the simulation as shown below.
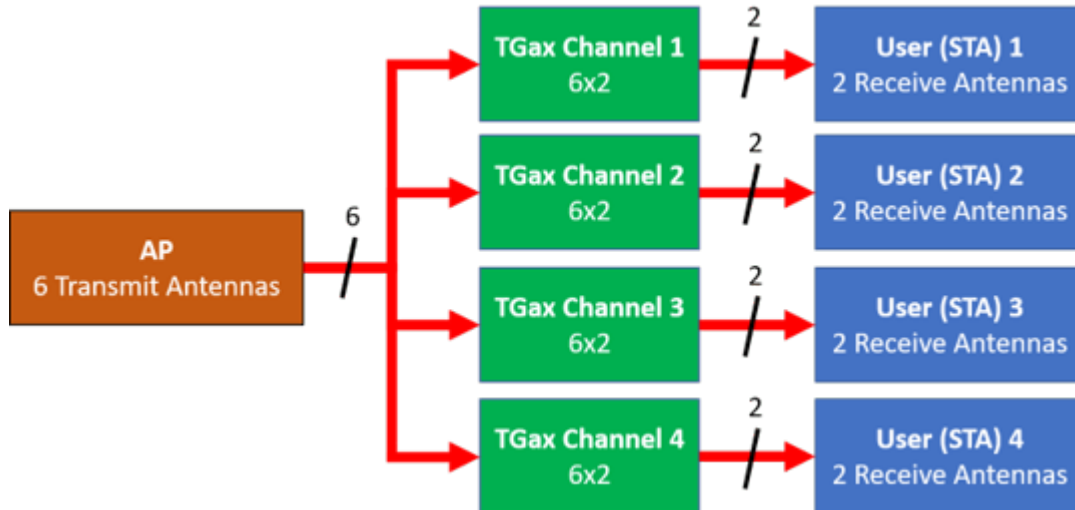
```matlab
% A cell array stores the channel objects, one per user
numUsers = numel(cfgMixed.User); % Number of users simulated in this example
tgax = cell(1,numUsers);

% Generate per-user channels
for userIdx = 1:numUsers
    tgax{userIdx} = clone(tgaxBase);
    tgax{userIdx}.UserIndex = userIdx; % Set unique user index
end
```

Beamforming Feedback

Transmit beamforming for both OFDMA and MU-MIMO relies on knowledge of the channel state between transmitter and receiver at the beamformer. Feedback of the per-subcarrier channel state is provided by each STA by channel sounding. A null data packet (NDP) is transmitted by the AP, and each STA uses this packet to determine the channel state. The channel state is then fed-back to the AP. The same process is used for 802.11ac™ in the "802.11ac Transmit Beamforming" on page 3-21 and "802.11ac Multi-User MIMO Precoding" on page 1-53 examples, but an HE single user NDP packet is used instead of a VHT packet. In this example, the feedback is considered perfect; there is no noise present for channel sounding and the feedback is uncompressed. The heUserBeamformingFeedback helper function detects the NDP and uses channel estimation to determine the channel state information. Singular value decomposition (SVD) is then used to calculate the beamforming feedback.

```matlab
% Create an NDP with the correct number of space-time streams to generate
% enough LTF symbols
cfgNDP = wlanHESUConfig('APEPLength',0,'GuardInterval',0.8); % No data in an NDP
cfgNDP.ChannelBandwidth = tgaxBase.ChannelBandwidth;
cfgNDP.NumTransmitAntennas = cfgMUMIMO.NumTransmitAntennas;
cfgNDP.NumSpaceTimeStreams = cfgMUMIMO.NumTransmitAntennas;

% Generate NDP packet - with an empty PSDU as no data
txNDP = wlanWaveformGenerator([],cfgNDP);

% For each user STA, pass the NDP packet through the channel and calculate
% the feedback channel state matrix by SVD.
staFeedback = cell(1,numUsers);
for userIdx = 1:numel(tgax)
```

```
        % Received waveform at user STA with 50 sample padding. No noise.
        rx = tgax{userIdx}([txNDP; zeros(50,size(txNDP,2))]);

        % Get the full-band beamforming feedback for a user
        staFeedback{userIdx} = heUserBeamformingFeedback(rx,cfgNDP);
    end
```

Simulation Parameters

Different path losses are simulated in this example. The same path loss and noise floor is applied to all users. For each path loss simulated, 10 packets are passed through the channel. Packets are separated by 20 microseconds.

```
cfgSim = struct;
cfgSim.NumPackets = 10;        % Number of packets to simulate for each path loss
cfgSim.Pathloss = (96:3:105); % Path losses to simulate in dB
cfgSim.TransmitPower = 30;     % AP transmit power in dBm
cfgSim.NoiseFloor = -89.9;     % STA noise floor in dBm
cfgSim.IdleTime = 20;          % Idle time between packets in us
```

Simulation with OFDMA

The scenario is first simulated with the OFDMA configuration and transmit beamforming.

The steering matrix for each RU is calculated using the feedback from the STAs. The `heMUCalculateSteeringMatrix` helper function calculates the beamforming matrix for an RU given the CSI feedback.

```
% For each RU, calculate the steering matrix to apply
for ruIdx = 1:numel(cfgOFDMA.RU)
    % Calculate the steering matrix to apply to the RU given the feedback
    steeringMatrix = heMUCalculateSteeringMatrix(staFeedback,cfgOFDMA,cfgNDP,ruIdx);

    % Apply the steering matrix to each RU
    cfgOFDMA.RU{ruIdx}.SpatialMapping = 'Custom';
    cfgOFDMA.RU{ruIdx}.SpatialMappingMatrix = steeringMatrix;
end
```

The `heMUSimulateScenario` helper function performs the simulation. The pre-HE preamble of 802.11ax is backwards compatible with 802.11ac, therefore in this example the front-end synchronization components for a VHT waveform are used to synchronize the HE waveform at each STA. For each packet and path loss simulated the following processing steps occur:

1 A PSDU is created and encoded to create a single packet waveform.

2 The waveform is passed through an evolving TGax channel model and AWGN is added to the received waveform. The channel state is maintained between packets.

3 The packet is detected.

4 Coarse carrier frequency offset is estimated and corrected.

5 Fine timing synchronization is established.

6 Fine carrier frequency offset is estimated and corrected.

7 The HE-LTF is extracted from the synchronized received waveform. The HE-LTF is OFDM demodulated and channel estimation is performed.

8 The HE Data field is extracted from the synchronized received waveform and OFDM demodulated.

9   Common pilot phase tracking is performed to track any residual carrier frequency offset.

10   The phase corrected OFDM symbols are equalized with the channel estimate.

11   Noise estimation is performed using the demodulated data field pilots and single-stream channel estimate at pilot subcarriers.

12   The equalized symbols are demodulated and decoded to recover the PSDU.

13   The recovered PSDU is compared to the transmitted PSDU to determine if the packet has been recovered successfully.

The simulation is run for the OFDMA configuration.

```
disp('Simulating OFDMA...');
throughputOFDMA = heMUSimulateScenario(cfgOFDMA,tgax,cfgSim);

Simulating OFDMA...
 Pathloss 96.0 dB, AP throughput 66.1 Mbps
 Pathloss 99.0 dB, AP throughput 66.1 Mbps
 Pathloss 102.0 dB, AP throughput 49.6 Mbps
 Pathloss 105.0 dB, AP throughput 16.5 Mbps
```

Simulation with MU-MIMO

Now the scenario is simulated with the MU-MIMO configuration. The `heMUCalculateSteeringMatrix` helper function calculates the beamforming matrix for an RU given the CSI feedback for all users in the MU-MIMO allocation. A zero forcing solution is used to calculate the steering matrix within the helper function.

```
% Calculate the steering matrix to apply to the RU given the feedback
ruIdx = 1; % Index of the one and only RU
steeringMatrix = heMUCalculateSteeringMatrix(staFeedback,cfgMUMIMO,cfgNDP,ruIdx);

% Apply the steering matrix to the RU
cfgMUMIMO.RU{1}.SpatialMapping = 'Custom';
cfgMUMIMO.RU{1}.SpatialMappingMatrix = steeringMatrix;
```

Run the simulation for the MU-MIMO configuration.

```
disp('Simulating MU-MIMO...');
throughputMUMIMO = heMUSimulateScenario(cfgMUMIMO,tgax,cfgSim);

Simulating MU-MIMO...
 Pathloss 96.0 dB, AP throughput 110.5 Mbps
 Pathloss 99.0 dB, AP throughput 110.5 Mbps
 Pathloss 102.0 dB, AP throughput 71.8 Mbps
 Pathloss 105.0 dB, AP throughput 0.0 Mbps
```

Simulation with Combined MU-MIMO and OFDMA

Finally, the scenario is simulated with the combined MU-MIMO and OFDMA configuration.

The steering matrix for each RU is calculated using the feedback from the STAs, including the MU-MIMO RU. The `heMUCalculateSteeringMatrix` helper function calculates the beamforming matrix for an RU given the CSI feedback.

```
% For each RU calculate the steering matrix to apply
for ruIdx = 1:numel(cfgMixed.RU)
    % Calculate the steering matrix to apply to the RU given the feedback
```

```
        steeringMatrix = heMUCalculateSteeringMatrix(staFeedback,cfgMixed,cfgNDP,ruIdx);

        % Apply the steering matrix to each RU
        cfgMixed.RU{ruIdx}.SpatialMapping = 'Custom';
        cfgMixed.RU{ruIdx}.SpatialMappingMatrix = steeringMatrix;
    end
```

Run the simulation for the combined MU-MIMO and OFDMA configuration.

```
disp('Simulating Mixed MU-MIMO and OFDMA...');
throughputMixed = heMUSimulateScenario(cfgMixed,tgax,cfgSim);
```

```
Simulating Mixed MU-MIMO and OFDMA...
 Pathloss 96.0 dB, AP throughput 66.1 Mbps
 Pathloss 99.0 dB, AP throughput 66.1 Mbps
 Pathloss 102.0 dB, AP throughput 66.1 Mbps
 Pathloss 105.0 dB, AP throughput 47.9 Mbps
```

Plot Results

The raw AP throughput for each transmission mode is plotted. The results show for this channel realization at high SNRs (low pathloss) the throughput provided by the MU-MIMO configuration exceeds OFDMA configuration. The packet duration of the MU-MIMO configuration is roughly half that of the OFDMA configuration which provides the throughput gain. As the SNR decreases, the noise dominates and transmit beamforming with OFDMA becomes more effective. The performance of the combined MU-MIMO and OFDMA configuration follow a similar trend to the OFDMA configuration as the packet duration is the same. The performance differs due to different RU sizes and number of space-time streams.

```
% Sum throughput for all STAs and plot for all configurations
figure;
plot(cfgSim.Pathloss,sum(throughputOFDMA,2),'-x');
hold on;
plot(cfgSim.Pathloss,sum(throughputMUMIMO,2),'-o');
plot(cfgSim.Pathloss,sum(throughputMixed,2),'-s');
grid on;
xlabel('Pathloss (dB)');
ylabel('Throughput (Mbps)');
legend('OFDMA','MU-MIMO','MU-MIMO & OFDMA');
title('Raw AP Throughput');
```
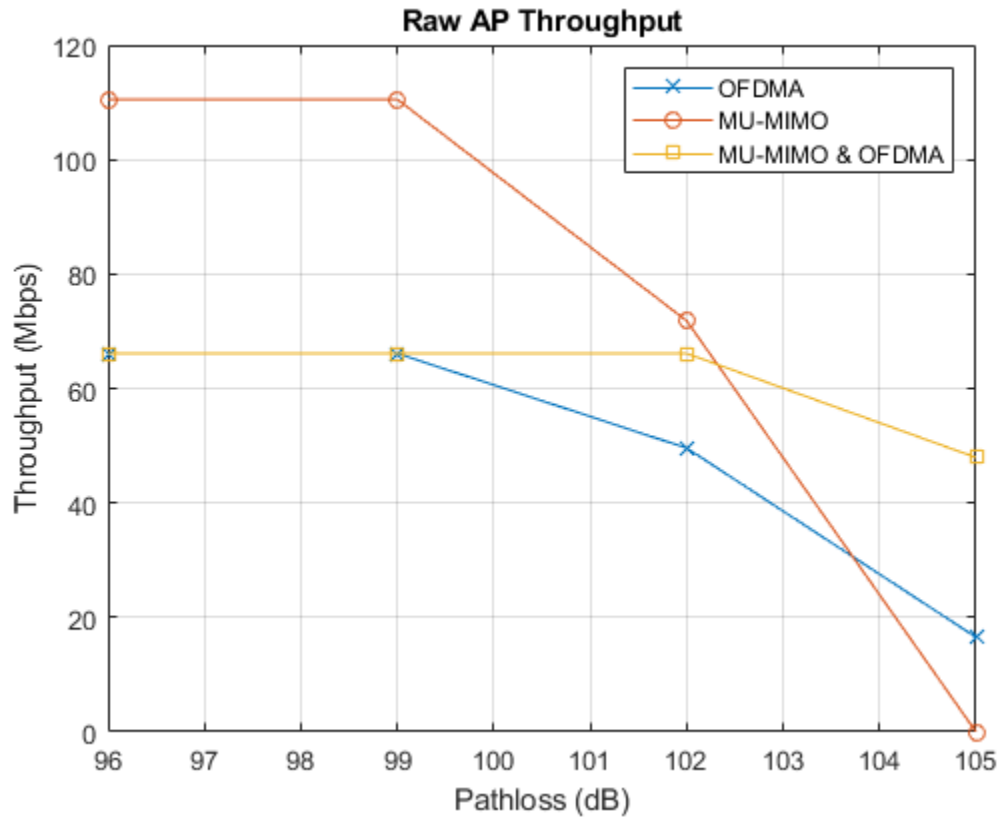
Raw AP Throughput

Appendix

This example uses these helper functions.

- heCommonPhaseErrorTracking.m
- heEqualizeCombine.m
- helperFrequencyOffset.m
- heLTFChannelEstimate.m
- heMUCalculateSteeringMatrix.m
- heMUSimulateScenario.m
- heNoiseEstimate.m
- heUserBeamformingFeedback.m

Selected Bibliography

1   IEEE P802.11ax™/D4.1 Draft 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 6: Enhancements for High Efficiency WLAN.

802.11ax Packet Error Rate Simulation for Uplink Trigger-Based Format

This example shows how to measure the packet error rate of an IEEE® 802.11ax™ high efficiency (HE) uplink, trigger-based (TB) format.

Introduction

The 802.11ax [1] HE trigger-based (HE TB) format allows for OFDMA or MU-MIMO transmission in the uplink. An HE TB transmission is controlled entirely by an access point (AP). All the parameters required for the transmission are provided in a trigger frame to all STAs participating in the HE TB transmission. Each station (STA) transmits an HE TB packet simultaneously, when triggered by the AP as shown in the following diagram.
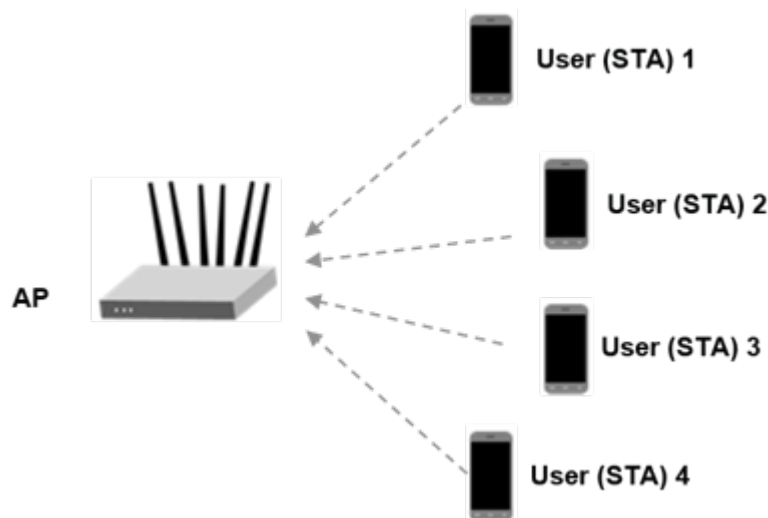
In this example an end-to-end simulation is used to determine the packet error rate of an HE TB link for four STAs in a MU-MIMO configuration. At each SNR point multiple packets are transmitted with no impairments apart from channel and noise. The received packets are demodulated and the PSDUs recovered for each STA. The PSDUs are compared to those transmitted to determine the number of packet errors and hence the packet error rate for all users. Packet detection, timing synchronization and symbol equalization is performed by the receiver. No frequency offset correction is performed in this example. The processing of HE TB processing chain is shown in the following diagram.
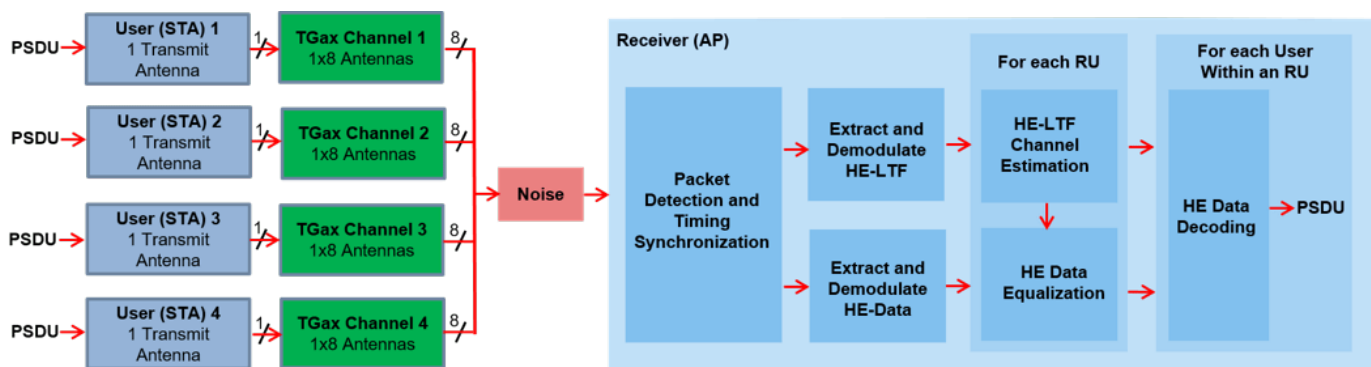
The receiver performs a minimum-mean-square-error-based ordered successive interference cancellation (MMSE-SIC) process for data equalization [2]. To avoid error propagation in the cancellation stage, data streams for all STAs are sorted in descending order based on the channel state information and equalized sequentially. This diagram shows the procedure of MMSE-SIC equalization.
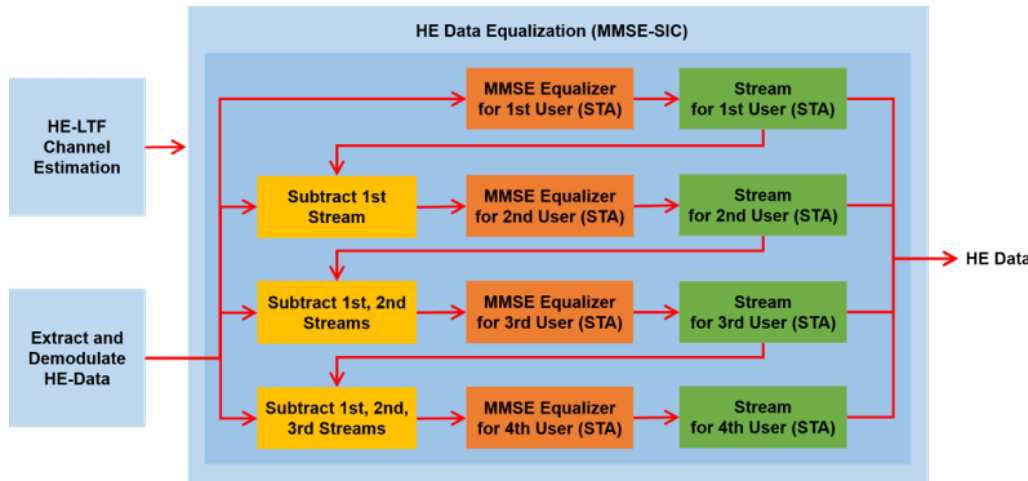
Equalization Method

In this example, you can specify the equalization method as 'mmse' or 'mmse-sic'. The default equalizer is 'mmse-sic'.

```
equalizationMethod = 'mmse-sic';
```

User Configuration

In this example the allocation information and transmit parameters for multiple uplink STAs are configured using an `heTBSystemConfig` object.

```
allocationIndex = 195; % Four uplink users in a MU-MIMO configuration
cfgSys = heTBSystemConfig(allocationIndex);
```

In a trigger-based transmission some parameters are the same for all uplink users, while some can differ. The `User` property of `cfgSys` contains a cell array of user configurations. Each element of the cell array is an object which can be configured to set the parameters of individual users. In this example, all users have the same transmission parameters.

```
% These parameters are same for all users in the MU-MIMO system
cfgSys.HELTFType = 4;          % HE-LTF compression mode
cfgSys.GuardInterval = 3.2;    % Guard interval type
cfgSys.SingleStreamPilots = 1; % Single stream pilot transmission of HE-LTF
numRx = 8;                     % Number of receive(AP) antennas

% The individual parameters for each user are specified below
allocInfo = ruInfo(cfgSys);
numUsers = allocInfo.NumUsers; % Number of uplink users

for userIdx = 1:numUsers
    cfgSys.User{userIdx}.NumTransmitAntennas = 1;
    cfgSys.User{userIdx}.NumSpaceTimeStreams = 1;
```

```
    cfgSys.User{userIdx}.SpatialMapping = 'Direct';
    cfgSys.User{userIdx}.MCS = 7;
    cfgSys.User{userIdx}.APEPLength = 1e3;
    cfgSys.User{userIdx}.ChannelCoding = 'LDPC';
end
```

A trigger-based transmission for a single user within the system is configured with an `wlanHETBConfig` object. The transmission configurations for all users are generated using the method `getUserConfig`. A cell array of four HE TB objects is created to describe the transmission of four users.

```
cfgTB = getUserConfig(cfgSys);
```

Simulation Parameters

For each SNR point (dB) in the `snr` vector a number of packets are generated, passed through a channel and demodulated to determine the packet error rate.

```
snr = 20:2:24;
```

```
% The sample rate and field indices for the HE TB packet is same for all
% users. Here the trigger configuration of the first user is used to get
% the sample rate and field indices of the HE TB PPDU.
fs = wlanSampleRate(cfgTB{1});    % Same for all users
ind = wlanFieldIndices(cfgTB{1}); % Same for all users
```

Channel Configuration

In this example, a TGax NLOS indoor channel model is used with delay profile Model-B. Model-B is considered NLOS when the distance between the transmitter and receiver is greater than or equal to 5 meters. This is described further in `wlanTGaxChannel`. In this example all the STAs are assumed to be at the same distance from the AP.

```
tgaxBase = wlanTGaxChannel;
tgaxBase.SampleRate = fs;
tgaxBase.TransmissionDirection = 'Uplink';
tgaxBase.TransmitReceiveDistance = 10;
chanBW = cfgSys.ChannelBandwidth;
tgaxBase.ChannelBandwidth = chanBW;
tgaxBase.NumReceiveAntennas = numRx;
```

An individual channel is created for each of the four users. Each channel is a clone of `tgaxBase`, but with a different `UserIndex` property, and is stored in a cell array `tgax`. The `UserIndex` property of each individual channel is set to provide a unique channel for each user. In this example a random channel realization is used for each packet by randomly varying the `UserIndex` property for each transmitted packet.

```
% A cell array stores the channel objects, one per user
tgax = cell(1,numUsers);
for userIdx = 1:numUsers
    tgax{userIdx} = clone(tgaxBase);
    tgax{userIdx}.NumTransmitAntennas = cfgSys.User{userIdx}.NumTransmitAntennas;
    tgax{userIdx}.UserIndex = userIdx;
end
```

Processing SNR Points

For each SNR point a number of packets are tested and the packet error rate is calculated. The pre-HE preamble of 802.11ax is backwards compatible with 802.11ac™, therefore in this example the

timing synchronization components for a VHT waveform are used to synchronize the HE waveform at the receiver. For each user, the following processing steps occur to create a waveform at the receiver containing all four users:

1 To create an HE TB waveform, a PSDU is created and encoded for each user based on predefined user parameters.

2 The waveform for each user is passed through an indoor TGax channel model. Different channel realizations are modeled for different users and packets, by randomly varying the `UserIndex` property of the channel. This results in same spatial correlation properties for all users.

3 The waveforms for all HE TB users are scaled and combined to ensure same SNR for each user after the addition of noise.

4 AWGN is added to the received waveform to create the desired average SNR per subcarrier after OFDM demodulation for each user. `comm.AWGNChannel` is configured to provide the correct SNR. The configuration accounts for normalization within the channel by the number of receive antennas, and the noise energy in unused subcarriers which is removed during OFDM demodulation.

At the receiver (AP) the following processing steps occur:

1 The packet is detected.

2 Fine timing synchronization is established. The L-STF, L-LTF and L-SIG samples are provided for fine timing to allow for packet detection at the start or end of the L-STF.

3 The HE-LTF and HE-Data fields for all users are extracted from the synchronized received waveform. The HE-LTF and HE-Data fields are OFDM demodulated.

4 The demodulated HE-LTF is extracted for each RU and channel estimation is performed.

5 Noise estimation is performed using the demodulated data field pilots for each RU.

6 The data field is extracted and equalized for all users within an RU, from the demodulated data field.

7 For each RU, and user within the RU, the spatial streams for a user are demodulated and decoded to recover the transmitted PSDU.

A parfor loop can be used to parallelize processing of the SNR points, therefore for each SNR point an AWGN channel is created and configured with comm.AWGNChannel. To enable the use of parallel computing for increased speed comment out the 'for' statement and uncomment the 'parfor' statement below.

```
ofdmInfo = wlanHEOFDMInfo('HE-Data',cfgSys.ChannelBandwidth,cfgSys.GuardInterval);
numSNR = numel(snr); % Number of SNR points
numPackets = 50; % Number of packets to simulate
packetErrorRate = zeros(numUsers,numSNR);
txPSDU = cell(numUsers);

% parfor isnr = 1:numSNR % Use 'parfor' to speed up the simulation
for isnr = 1:numSNR
    % Set random substream index per iteration to ensure that each
    % iteration uses a repeatable set of random numbers
    stream = RandStream('combRecursive','Seed',0);
    stream.Substream = isnr;
    RandStream.setGlobalStream(stream);

    % Create an instance of the AWGN channel per SNR point simulated
    awgn = comm.AWGNChannel;
```

```matlab
awgn.NoiseMethod = 'Signal to noise ratio (SNR)';
awgn.SignalPower = 1/numRx;
sysInfo = ruInfo(cfgSys);

% Simulate multiple packets
numPacketErrors = zeros(numUsers,1);
for pktIdx = 1:numPackets

    % Transmit processing
    rxWaveform = 0;
    packetError = zeros(numUsers,1);
    txPSDU = cell(1,numUsers);

    % Generate random channel realization for each packet by varying
    % the UserIndex property of the channel. This assumes all users
    % have the same number of transmit antennas.
    chPermutations = randperm(numUsers);
    for userIdx = 1:numUsers
        % HE TB config object for each user
        cfgUser = cfgTB{userIdx};

        % Generate a packet with random PSDU
        txPSDU{userIdx} = randi([0 1],getPSDULength(cfgUser)*8,1,'int8');

        % Generate HE TB waveform, containing payload for single user
        txTrig = wlanWaveformGenerator(txPSDU{userIdx},cfgUser);

        % Pass waveform through a random TGax Channel
        channelIdx = chPermutations(userIdx);
        reset(tgax{channelIdx}); % New channel realization
        rxTrig = tgax{channelIdx}([txTrig; zeros(15,size(txTrig,2))]);

        % Scale the transmit power of the user within an RU. This is to
        % ensure same SNR for each user after the addition of noise.
        ruNum = cfgSys.User{userIdx}.RUNumber;
        SF = sqrt(1/sysInfo.NumUsersPerRU(ruNum))*sqrt(cfgUser.RUSize/(sum(sysInfo.RUSizes))

        % Combine uplink users into one waveform
        rxWaveform = rxWaveform+SF*rxTrig;
    end

    % Pass the waveform through AWGN channel. Account for noise
    % energy in nulls so the SNR is defined per active subcarriers.
    awgn.SNR = snr(isnr)-10*log10(ofdmInfo.FFTLength/sum(sysInfo.RUSizes));
    rxWaveform = awgn(rxWaveform);

    % Receive processing
    % Packet detect and determine coarse packet offset
    coarsePktOffset = wlanPacketDetect(rxWaveform,chanBW,0,0.05);
    if isempty(coarsePktOffset) % If empty no L-STF detected; packet error
        numPacketErrors = numPacketErrors+1;
        continue; % Go to next loop iteration
    end

    % Extract the non-HT fields and determine fine packet offset
    nonhtfields = rxWaveform(coarsePktOffset+(ind.LSTF(1):ind.LSIG(2)),:);
    finePktOffset = wlanSymbolTimingEstimate(nonhtfields,chanBW);
```

```matlab
        % Determine final packet offset
        pktOffset = coarsePktOffset+finePktOffset;

        % If packet detected out with the range of expected delays from
        % the channel modeling; packet error
        if pktOffset>50
            numPacketErrors = numPacketErrors+1;
            continue; % Go to next loop iteration
        end

        % Extract HE-LTF and HE-Data fields for all RUs
        rxLTF = rxWaveform(pktOffset+(ind.HELTF(1):ind.HELTF(2)),:);
        rxData = rxWaveform(pktOffset+(ind.HEData(1):ind.HEData(2)),:);

        for ruIdx = 1:allocInfo.NumRUs
            % Demodulate HE-LTF and HE-Data field for the RU of interest
            ru = [allocInfo.RUSizes(ruIdx) allocInfo.RUIndices(ruIdx)];
            demodHELTFRU = wlanHEDemodulate(rxLTF,'HE-LTF',chanBW,cfgSys.GuardInterval,cfgSys.HEL
            demodHEDataRU = wlanHEDemodulate(rxData,'HE-Data',chanBW,cfgSys.GuardInterval,ru);

            % Channel estimate
            [chanEst,ssPilotEst] = heLTFChannelEstimate(demodHELTFRU,cfgSys,ruIdx);

            % Get indices of data and pilots within RU (without nulls)
            ruOFDMInfo = wlanHEOFDMInfo('HE-Data',cfgSys.ChannelBandwidth,cfgSys.GuardInterval,
                [allocInfo.RUSizes(ruIdx) allocInfo.RUIndices(ruIdx)]);

            % Estimate noise power in HE fields of each user
            nVarEst = heNoiseEstimate(demodHEDataRU(ruOFDMInfo.PilotIndices,:,:),ssPilotEst,cfgSy

            % Discard pilot subcarriers
            demodDataSym = demodHEDataRU(ruOFDMInfo.DataIndices,:,:);
            chanEstData = chanEst(ruOFDMInfo.DataIndices,:,:);

            % Equalize
            if strcmpi(equalizationMethod,'mmse-sic')
                [eqSym,csi] = heSuccessiveEqualize(demodDataSym,chanEstData,nVarEst,cfgSys,ruIdx
            else
                [eqSym,csi] = heEqualizeCombine(demodDataSym,chanEstData,nVarEst,cfgSys);
            end

            for userIdx = 1:allocInfo.NumUsersPerRU(ruIdx)
                % Get TB config object for each user
                userNum = cfgSys.RU{ruIdx}.UserNumbers(userIdx);
                cfgUser = cfgTB{userNum};

                % Get space-time stream indices for the current user
                stsIdx = cfgUser.StartingSpaceTimeStream-1+(1:cfgUser.NumSpaceTimeStreams);

                % Demap and decode bits
                rxPSDU = wlanHEDataBitRecover(eqSym(:,:,stsIdx),nVarEst,csi(:,stsIdx),cfgUser,'L

                % PER calculation
                packetError(userNum) = any(biterr(txPSDU{userNum},rxPSDU));
            end
        end
        numPacketErrors = numPacketErrors+packetError;
    end
```

```matlab
        % Calculate packet error rate (PER) at SNR point
        packetErrorRate(:,isnr)= numPacketErrors/numPackets;
        disp(['SNR ' num2str(snr(isnr)) ...
                ' completed for ' num2str(numUsers) ' users']);

end
```

```
SNR 20 completed for 4 users
SNR 22 completed for 4 users
SNR 24 completed for 4 users
```

Plot Packet Error Rate vs SNR

```matlab
markers = 'ox*sd^v><ph+ox*sd^v';
color = 'bmcrgbrkymcrgbrkymc';
figure;

for nSTA = 1:numUsers
    semilogy(snr,packetErrorRate(nSTA,:).',['-' markers(nSTA) color(nSTA)]);
    hold on;
end

grid on;
xlabel('SNR (dB)');
ylabel('PER');
dataStr = arrayfun(@(x)sprintf('STA- %d',x),1:numUsers,'UniformOutput',false);
legend(dataStr);
title('PER for uplink 802.11ax link');
```

PER for uplink 802.11ax link
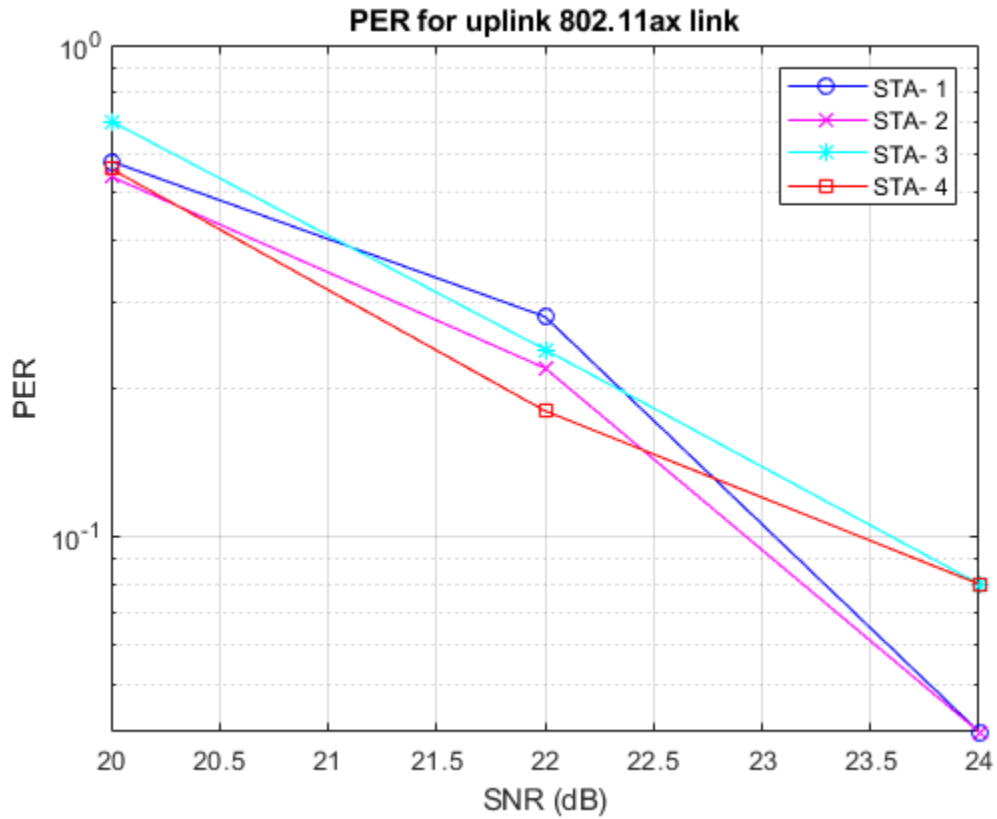
The number of packets tested at each SNR point is controlled by `numPackets`. For meaningful results, this value should be larger than those presented in this example. The figure below was created by running a longer simulation with `numPackets`:1e4 and `snr`:20:2:28, which shows the packet error rate of both MMSE equalizer and MMSE-SIC equalizer.

PER for uplink 802.11ax link
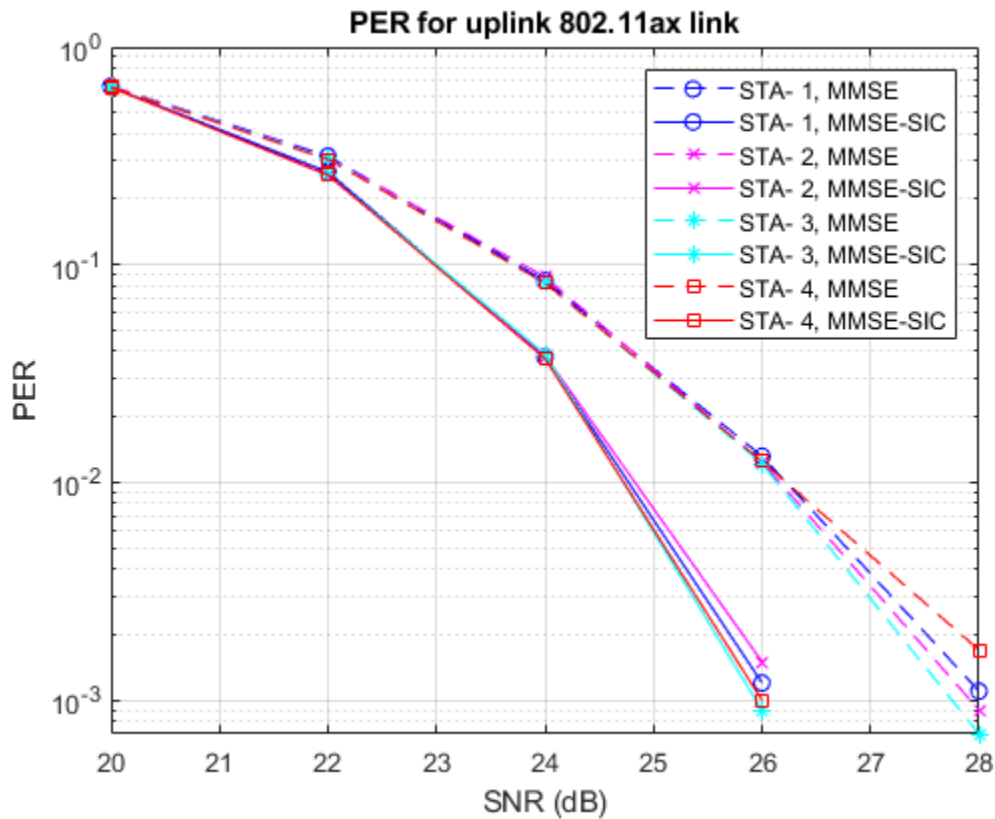
Appendix

This example uses the following helper functions and objects:

- heEqualizeCombine.m
- heLTFChannelEstimate.m
- heNoiseEstimate.m
- heSuccessiveEqualize.m
- heTBRU.m
- heTBSystemConfig.m
- heTBUser.m

Selected Bibliography

1 IEEE P802.11ax™/D4.1 Draft 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 6: Enhancements for High Efficiency WLAN.

2 M. Debbah, B. Muquet, M. de Courville, M. Muck, S. Simoens, and P. Loubaton. A MMSE successive interference cancellation scheme for a new adjustable hybrid spread OFDM system. IEEE 51st Vehicular Technology Conference Proceedings, pp. 745-749, vol. 2, 2000.

802.11ax Compressed Beamforming Packet Error Rate Simulation

This example shows how to measure the packet error rate of a beamformed IEEE® 802.11ax™ high efficiency single user (HE SU) format link with different beamforming feedback quantization levels.

Introduction

Transmit beamforming focuses energy towards a receiver to improve the SNR of a link. In this scheme, the transmitter is called a beamformer and the receiver is called a beamformee. A steering matrix is used by the beamformer to direct the energy to the beamformee. The steering matrix is calculated using channel state information obtained through channel measurements. These measurements are obtained by sounding the channel between beamformer and beamformee. To sound the channel, the beamformer sends a null data packet (NDP) to the beamformee. The beamformee measures the channel information during sounding to calculate a feedback matrix. This matrix is compressed in the form of quantized angles (phi and psi) and fed back to the beamformer. The beamformer can then calculate the feedback matrix from the quantized angles to create a steering matrix and beamform transmissions to the beamformee. The process of forming steering matrix is shown in "802.11ac Transmit Beamforming" on page 3-21.

In this example, a 4x2 MIMO configuration is considered between a transmitter and receiver, with two space-time streams used for each data packet transmission. An end-to-end simulation is used to determine the packet error rate (PER) for an 802.11ax [1] single user (SU) format link with compressed beamforming feedback quantization for different quantization levels and a selection of SNR points. A plot is generated showing the PER vs SNR curve for each quantization resolution. This example does not consider grouping of sub-carriers (see Section 9.4.1.65 in [1]).
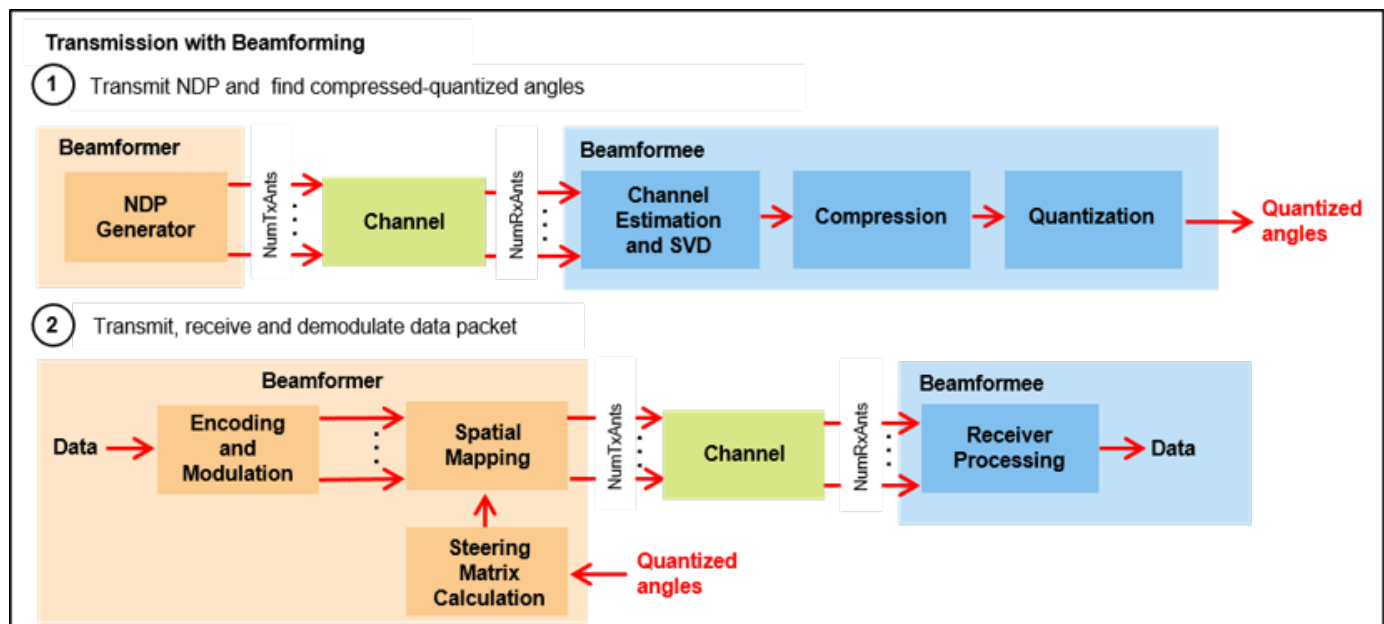
Waveform Configuration

An HE-SU packet is a full-band transmission to a single user. The transmit parameters for the HE-SU format are configured using a `wlanHESUConfig` object. The properties of the object contain the

configuration. In this example, the object is configured for a 20 MHz channel bandwidth, 4 transmit antennas, 2 space-time streams and 16-QAM rate-1/2 (MCS 3).

```
NumTxAnts = 4; % Number of transmit antennas
NumSTS = 2;    % Number of space-time streams
NumRxAnts = 2; % Number of receive antennas
cfgHEBase = wlanHESUConfig;
cfgHEBase.ChannelBandwidth = 'CBW20';       % Channel bandwidth
cfgHEBase.NumSpaceTimeStreams = NumSTS;     % Number of space-time streams
cfgHEBase.NumTransmitAntennas = NumTxAnts;  % Number of transmit antennas
cfgHEBase.APEPLength = 1e3;                 % Payload length in bytes
cfgHEBase.ExtendedRange = false;            % Do not use extended range format
cfgHEBase.Upper106ToneRU = false;           % Do not use upper 106 tone RU
cfgHEBase.PreHESpatialMapping = false;      % Spatial mapping of pre-HE fields
cfgHEBase.GuardInterval = 0.8;              % Guard interval duration
cfgHEBase.HELTFType = 4;                    % HE-LTF compression mode
cfgHEBase.ChannelCoding = 'LDPC';           % Channel coding
cfgHEBase.MCS = 3;                          % Modulation and coding scheme
cfgHEBase.SpatialMapping = 'Custom';        % Custom for beamforming
```

Null Data Packet (NDP) Configuration

The NDP transmission is configured to have data length of zero. Since the NDP is used to obtain the channel state information, the number of space-time streams is equal to number of transmit antennas. This results in a direct mapping of each space-time stream to a transmit antenna.

```
cfgNDP = cfgHEBase;
cfgNDP.APEPLength = 0;                      % NDP has no data
cfgNDP.NumSpaceTimeStreams = NumTxAnts;     % For feedback matrix calculation
cfgNDP.SpatialMapping = 'Direct';           % Each TxAnt carries a STS
```

Channel Configuration

In this example, a TGax NLOS indoor channel model is used with delay profile Model-B. The Model-B profile is considered NLOS when the distance between transmitter and receiver is greater than or equal to 5 meters. This is described further in `wlanTGaxChannel`. A 4x2 MIMO channel is simulated in this example.

```
% Create and configure the TGax channel
chanBW = cfgHEBase.ChannelBandwidth;
tgaxChannel = wlanTGaxChannel;
tgaxChannel.DelayProfile = 'Model-B';
tgaxChannel.NumTransmitAntennas = NumTxAnts;
tgaxChannel.NumReceiveAntennas = NumRxAnts;
tgaxChannel.TransmitReceiveDistance = 5; % Distance in meters for NLOS
tgaxChannel.ChannelBandwidth = chanBW;
tgaxChannel.LargeScaleFadingEffect = 'None';
fs = wlanSampleRate(cfgHEBase);
tgaxChannel.SampleRate = fs;
```

Simulation Parameters

This example compares the performance of beamforming with two different resolutions of compression quantization, and without compression. For each quantization resolution, an end to end simulation with various SNR values is run to determine the packet error rate. 802.11ax Draft 4.1 specifies only two sets of quantization resolution for single user beamforming (Table 9-31a in [1]). The value of `codeBookSize` determines the number of bits used to quantize the beamforming

feedback angles (phi and psi) in this simulation. When `codeBookSize` is Inf, no compression is performed. The quantization levels selected by `codeBookSize` are shown in the table below:

```
codeBookSize            Compression Configuration
---------------------------------------------------------
    0                   NumBitsphi = 4; NumBitspsi = 2
    1                   NumBitsphi = 6; NumBitspsi = 4
  Inf                   No compression
---------------------------------------------------------
```

```
codeBookSize = [0 1 Inf];
```

A number of packets are generated, passed through a channel and demodulated to determine the packet error rate for each compression configuration at each SNR (dB) value in the `snr` vector.

```
snr = 10:2:18;
```

The number of packets tested at each SNR point is limited to a `maxNumErrors` or `maxNumPackets`:

1 `maxNumErrors` is the maximum number of packet errors simulated at each SNR point. When the number of packet errors reaches this limit, the simulation at this SNR point is complete.

2 `maxNumPackets` is the maximum number of packets simulated at each SNR point and limits the length of the simulation if the packet error limit is not reached.

The numbers chosen in this example will lead to a very short simulation. For statistically meaningful results we recommend increasing these numbers.

```
maxNumErrors = 10;   % The maximum number of packet errors at an SNR point
maxNumPackets = 100; % The maximum number of packets at an SNR point
```

Processing SNR Points

For each SNR point, a number of packets are tested and the packet error rate calculated. The pre-HE preamble of 802.11ax is backwards compatible with 802.11ac™, therefore in this example the front-end synchronization components for a VHT waveform are used to synchronize the HE waveform at the receiver. For each packet, the following processing steps occur.

The beamformer obtains the steering matrix by transmitting an NDP which is processed by the beamformee to create a feedback matrix:

1 An NDP waveform is transmitted through an indoor TGax channel model. Different channel realizations are modeled for different packets.

2 AWGN is added to the received waveform to create the desired average SNR per subcarrier after OFDM demodulation. The `comm.AWGNChannel` is configured to provide the correct SNR. The configuration accounts for normalization within the channel by the number of receive antennas, and the noise energy in unused subcarriers which are removed during OFDM demodulation.

3 The packet is detected at the beamformee.

4 Coarse carrier frequency offset is estimated and corrected.

5 Fine timing synchronization is established. The L-STF, L-LTF and L-SIG samples are provided for fine timing to allow for packet detection at the start or end of the L-STF.

6 Fine carrier frequency offset is estimated and corrected.

7 The HE-LTF is extracted from the synchronized received waveform. The HE-LTF is OFDM demodulated and channel estimation is performed.

8 Singular value decomposition is performed on the estimated channel and the beamforming feedback matrix, V is calculated.

9 If there is no compression, this feedback matrix, V will be used as the steering matrix by the beamformer.

10 If compression is used, the feedback matrix, V will be compressed and quantized to create a set of angles as specified in the standard.

The beamformer transmits a data packet using the recovered steering matrix and the beamformee decodes the beamformed data transmission to recover the PSDU:

1 Since the current example assumes zero delay in getting the beamforming feedback from the beamformee, the quantized angles are converted back to the beamforming feedback matrix, V.

2 A PSDU is created and encoded to create a single packet waveform with the steering matrix set to the beamforming feedback matrix, V.

3 The waveform is passed through the same indoor TGax channel realization as the NDP transmission.

4 AWGN is added to the received waveform.

5 As with NDP, synchronization and HE channel estimation are performed.

6 The data field is extracted from the synchronized received waveform and OFDM demodulated.

7 Common phase error pilot tracking is performed to track any residual carrier frequency offset.

8 Noise estimation is performed using the demodulated data field pilots and single-stream channel estimate at pilot subcarriers.

9 The phase corrected OFDM symbols are equalized with the channel estimate.

10 The equalized symbols are demodulated and decoded to recover the PSDU.

A `parfor` loop can be used to parallelize processing of the SNR points. To enable the use of parallel computing for increased speed comment out the 'for' statement and uncomment the 'parfor' statement below.

```
numQuant = numel(codeBookSize);
numSNR = numel(snr); % Number of SNR points
packetErrorRate = zeros(numQuant,numSNR);

% Get occupied subcarrier indices and OFDM parameters
ofdmInfo = wlanHEOFDMInfo('HE-Data',cfgHEBase);

% Indices to extract fields from the PPDU
ind = wlanFieldIndices(cfgHEBase);
indSound = wlanFieldIndices(cfgNDP);

for ibf = 1:numQuant
    switch codeBookSize(ibf) % See P802.11ax/D4.1 Section 9.4.1.64
        case 0
            NumBitsPsi = 2; % Number of bits for psi
            NumBitsPhi = 4; % Number of bits for phi
            disp('End-to-End simulation with compressed beamforming quantization with');
            disp(['Number of Bits for phi = ' num2str(NumBitsPhi) ...
                ' and Number of Bits for psi = ' num2str(NumBitsPsi)]);
        case 1
            NumBitsPsi = 4; % Number of bits for psi
            NumBitsPhi = 6; % Number of bits for phi
            disp('End-to-End simulation with compressed beamforming quantization with');
```

```
            disp(['Number of Bits for phi = ' num2str(NumBitsPhi) ...
                ' and Number of Bits for psi = ' num2str(NumBitsPsi)]);
        otherwise
            disp('End-to-End simulation with non-compressed beamforming');
    end

%parfor isnr = 1:numSNR % Use 'parfor' to speed up the simulation
for isnr = 1:numSNR
    % Set random substream index per iteration to ensure that each
    % iteration uses a repeatable set of random numbers
    stream = RandStream('combRecursive','Seed',100);
    stream.Substream = isnr;
    RandStream.setGlobalStream(stream);

    % Create an instance of the AWGN channel per SNR point simulated
    awgnChannel = comm.AWGNChannel;
    awgnChannel.NoiseMethod = 'Signal to noise ratio (SNR)';
    awgnChannel.SignalPower = 1/tgaxChannel.NumReceiveAntennas;
    % Account for noise energy in nulls so the SNR is defined per
    % active subcarrier
    awgnChannel.SNR = snr(isnr)-10*log10(ofdmInfo.FFTLength/ofdmInfo.NumTones);

    % Create an instance of the HE configuration object per SNR point
    % simulated. This will enable to use parfor
    cfgHE = cfgHEBase;

    % Loop to simulate multiple packets
    numPacketErrors = 0;
    numPkt = 1; % Index of packet transmitted
    while numPacketErrors<=maxNumErrors && numPkt<=maxNumPackets
        % Null data packet transmission
        tx = wlanWaveformGenerator([],cfgNDP);

        % Add trailing zeros to allow for channel delay
        txPad = [tx; zeros(50,cfgNDP.NumTransmitAntennas)];

        % Pass through a fading indoor TGax channel
        reset(tgaxChannel); % Reset channel for different realization
        rx = tgaxChannel(txPad);

        % Pass the waveform through AWGN channel
        rx = awgnChannel(rx);

        % Calculate the steering matrix at the beamformee
        V = heUserBeamformingFeedback(rx,cfgNDP,true);

        if isempty(V)
            % User feedback failed, packet error
            numPacketErrors = numPacketErrors+1;
            numPkt = numPkt+1;
            continue; % Go to next loop iteration
        end

        if ~isinf(codeBookSize(ibf))
            % Find quantized angles of the beamforming feedback matrix
            angidx = bfCompressQuantize(V(:,1:NumSTS,:),NumBitsPhi,NumBitsPsi);

            % Calculate steering matrix from the quantized angles at
```

```matlab
        % beamformer:
        % Assuming zero delay in transmitting the quantized angles
        % from beamformee to beamformer, the steering matrix is
        % calculated from the quantized angles and is used in the
        % data transmission of beamformer.

        [~,Nc,Nr] = size(V(1,1:NumSTS,:));
        V = bfDecompress(angidx,Nr,Nc,NumBitsPhi,NumBitsPsi);
    end

    steeringMat = V(:,1:NumSTS,:);

    % Beamformed data transmission
    psduLength = getPSDULength(cfgHE); % PSDU length in bytes
    txPSDU = randi([0 1],psduLength*8,1); % Generate random PSDU
    cfgHE.SpatialMappingMatrix = steeringMat;
    tx = wlanWaveformGenerator(txPSDU,cfgHE);

    % Add trailing zeros to allow for channel delay
    txPad = [tx; zeros(50,cfgHE.NumTransmitAntennas)];

    % Pass through a fading indoor TGax channel
    rx = tgaxChannel(txPad);

    % Pass the waveform through AWGN channel
    rx = awgnChannel(rx);

    % Packet detect and determine coarse packet offset
    coarsePktOffset = wlanPacketDetect(rx,chanBW);
    if isempty(coarsePktOffset) % If empty no L-STF detected; packet error
        numPacketErrors = numPacketErrors+1;
        numPkt = numPkt+1;
        continue; % Go to next loop iteration
    end

    % Extract L-STF and perform coarse frequency offset correction
    lstf = rx(coarsePktOffset+(ind.LSTF(1):ind.LSTF(2)),:);
    coarseFreqOff = wlanCoarseCFOEstimate(lstf,chanBW);
    rx = helperFrequencyOffset(rx,fs,-coarseFreqOff);

    % Extract the non-HT fields and determine fine packet offset
    nonhtfields = rx(coarsePktOffset+(ind.LSTF(1):ind.LSIG(2)),:);
    finePktOffset = wlanSymbolTimingEstimate(nonhtfields,chanBW);

    % Determine final packet offset
    pktOffset = coarsePktOffset+finePktOffset;

    % If packet detected outwith the range of expected delays from
    % the channel modeling; packet error
    if pktOffset>50
        numPacketErrors = numPacketErrors+1;
        numPkt = numPkt+1;
        continue; % Go to next loop iteration
    end

    % Extract L-LTF and perform fine frequency offset correction
    rxLLTF = rx(pktOffset+(ind.LLTF(1):ind.LLTF(2)),:);
    fineFreqOff = wlanFineCFOEstimate(rxLLTF,chanBW);
```

```matlab
            rx = helperFrequencyOffset(rx,fs,-fineFreqOff);

            % HE-LTF demodulation and channel estimation
            rxHELTF = rx(pktOffset+(ind.HELTF(1):ind.HELTF(2)),:);
            heltfDemod = wlanHEDemodulate(rxHELTF,'HE-LTF',cfgHE);
            [chanEst,pilotEst] = heLTFChannelEstimate(heltfDemod,cfgHE);

            % Data demodulate
            rxData = rx(pktOffset+(ind.HEData(1):ind.HEData(2)),:);
            demodSym = wlanHEDemodulate(rxData,'HE-Data',cfgHE);

            % Pilot phase tracking
            % Average single-stream pilot estimates over symbols (2nd dimension)
            pilotEstTrack = mean(pilotEst,2);
            demodSym = heCommonPhaseErrorTracking(demodSym,pilotEstTrack,cfgHE);

            % Estimate noise power in HE fields
            nVarEst = heNoiseEstimate(demodSym(ofdmInfo.PilotIndices,:,:),pilotEstTrack,cfgHE);

            % Extract data subcarriers from demodulated symbols and channel
            % estimate
            demodDataSym = demodSym(ofdmInfo.DataIndices,:,:);
            chanEstData = chanEst(ofdmInfo.DataIndices,:,:);

            % Equalization and STBC combining
            [eqDataSym,csi] = heEqualizeCombine(demodDataSym,chanEstData,nVarEst,cfgHE);

            % Recover data
            rxPSDU = wlanHEDataBitRecover(eqDataSym,nVarEst,csi,cfgHE,'LDPCDecodingMethod','layer

            % Determine if any bits are in error, i.e. a packet error
            packetError = ~isequal(txPSDU,rxPSDU);
            numPacketErrors = numPacketErrors+packetError;
            numPkt = numPkt+1;
        end

        % Calculate packet error rate (PER) at SNR point
        packetErrorRate(ibf,isnr) = numPacketErrors/(numPkt-1);
        disp(['MCS ' num2str(cfgHE.MCS) ','...
            ' SNR ' num2str(snr(isnr)) ...
            ' completed after ' num2str(numPkt-1) ' packets,'...
            ' PER:' num2str(packetErrorRate(ibf,isnr))]);
    end
    disp(newline);
end
```

```
End-to-End simulation with compressed beamforming quantization with
Number of Bits for phi = 4 and Number of Bits for psi = 2
MCS 3, SNR 10 completed after 13 packets, PER:0.84615
MCS 3, SNR 12 completed after 54 packets, PER:0.2037
MCS 3, SNR 14 completed after 100 packets, PER:0.07
MCS 3, SNR 16 completed after 100 packets, PER:0
MCS 3, SNR 18 completed after 100 packets, PER:0


End-to-End simulation with compressed beamforming quantization with
Number of Bits for phi = 6 and Number of Bits for psi = 4
MCS 3, SNR 10 completed after 13 packets, PER:0.84615
```

6-35

```
MCS 3, SNR 12 completed after 54 packets, PER:0.2037
MCS 3, SNR 14 completed after 100 packets, PER:0.06
MCS 3, SNR 16 completed after 100 packets, PER:0
MCS 3, SNR 18 completed after 100 packets, PER:0


End-to-End simulation with non-compressed beamforming
MCS 3, SNR 10 completed after 13 packets, PER:0.84615
MCS 3, SNR 12 completed after 59 packets, PER:0.18644
MCS 3, SNR 14 completed after 100 packets, PER:0.06
MCS 3, SNR 16 completed after 100 packets, PER:0
MCS 3, SNR 18 completed after 100 packets, PER:0
```

Plot Packet Error Rate vs Signal to Noise Ratio

```matlab
figure;
lineTypes = ["k-o" "b-s" "r-*"];
semilogy(snr,packetErrorRate(1,:),lineTypes(1));
hold on;
grid on;
xlabel('SNR (dB)');
ylabel('PER');
for ibf = 2:numQuant
    semilogy(snr,packetErrorRate(ibf,:),lineTypes(ibf));
end
dataStr = [string(['Compressed Beamforming, ' newline ...
                   'NumBitsPhi = 4, NumBitsPsi = 2' newline])...
            string(['Compressed Beamforming, ' newline ...
                   'NumBitsPhi = 6, NumBitsPsi = 4' newline]) ...
                   "Non-Compressed Beamforming"];
legend(dataStr);
title(sprintf('802.11ax Beamforming PER for Channel %s, %s, %s',tgaxChannel.DelayProfile,cfgHEBas
```
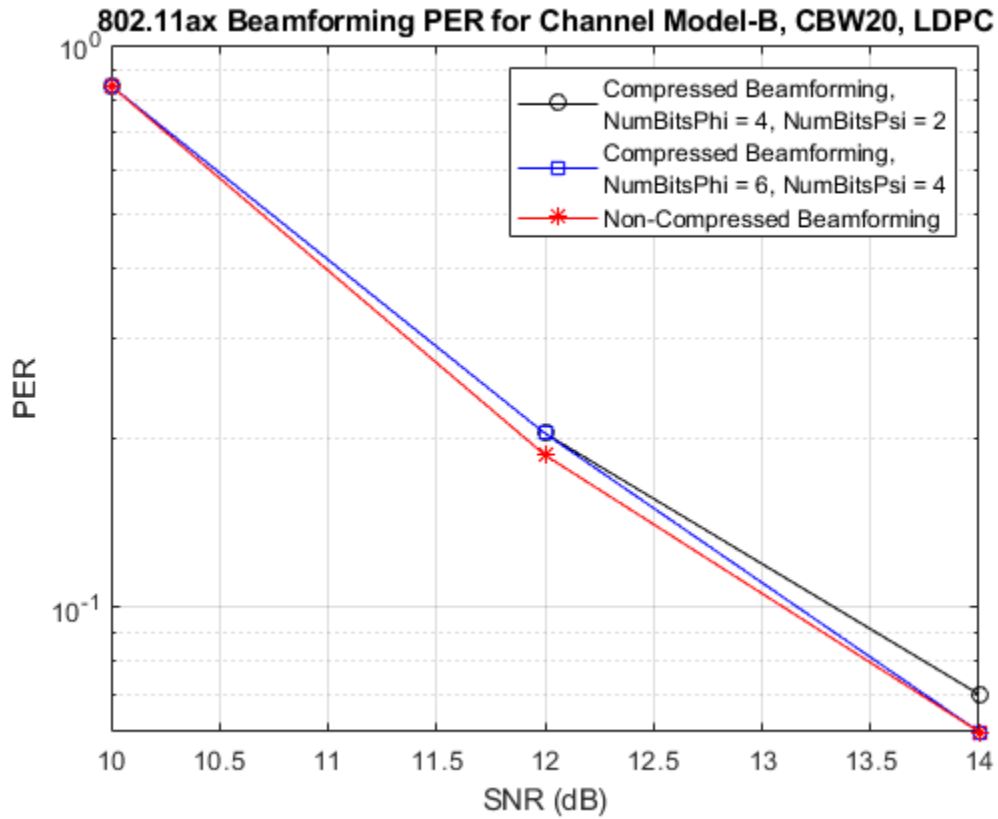
802.11ax Beamforming PER for Channel Model-B, CBW20, LDPC

The number of packets tested at each SNR point is controlled by two parameters: `maxNumErrors` and `maxNumPackets`. For meaningful results, these values should be larger than those presented in this example. As an example, the figure below was created by running a longer simulation with `maxNumErrors`:1e3 and `maxNumPackets`:1e4.

802.11ax Beamforming PER for Channel Model-B, CBW20, LDPC
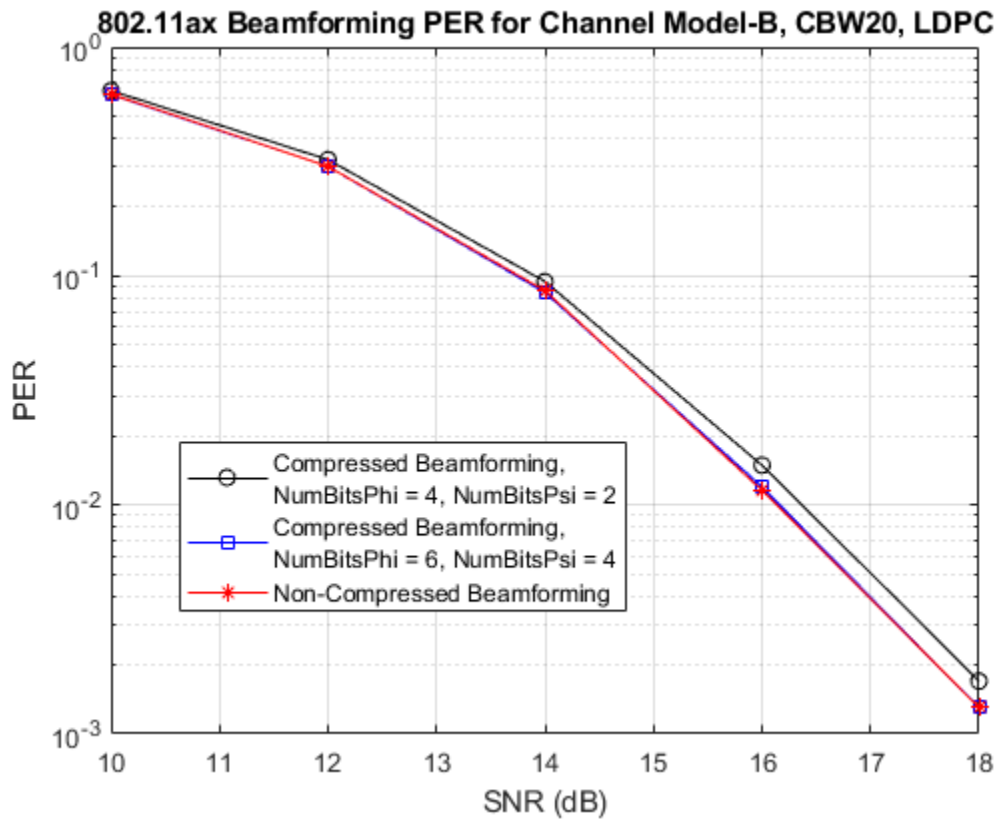
Appendix

This example uses the following helper functions:

- bfCompressQuantize.m
- bfDecompress.m
- heCommonPhaseErrorTracking.m
- heEqualizeCombine.m
- helperFrequencyOffset.m
- heLTFChannelEstimate.m
- heNoiseEstimate.m
- heUserBeamformingFeedback.m

Selected Bibliography

1   IEEE P802.11ax™/D4.1 Draft 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 6: Enhancements for High Efficiency WLAN.

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

This example shows how to measure the probability of misdetecting feedback status information in an uplink high efficiency (HE) trigger-based (TB) feedback null data packet (NDP) transmission from multiple uplink stations (STAs).

Introduction

The 802.11ax [1 on page 6-0   ] HE TB feedback NDP is a variant of the HE TB physical layer protocol data unit (PPDU). The HE TB feedback NDP transmission is controlled entirely by an access point (AP). All the parameters required for the transmission are provided in a trigger frame of type NDP feedback report poll (NFRP) sent to all STAs participating in the HE TB feedback transmission. Following the transmission of an NFRP trigger frame from the AP, multiple STAs may simultaneously transmit an HE TB feedback NDP, which carries the resource request information (feedback status) as shown in this diagram. For more information on the NDP feedback report procedure, see the NDP Feedback Report Procedure section of the `wlanHETBConfig` reference page.
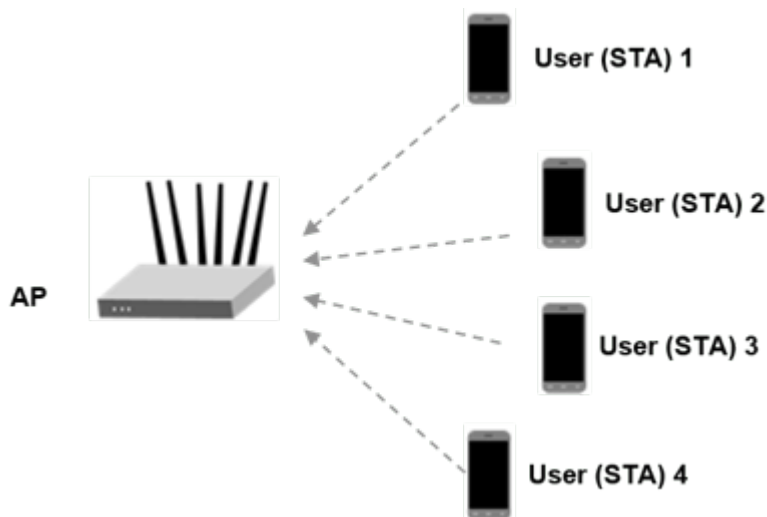
This example measures the probability of misdetecting feedback status information for an HE TB feedback NDP by comparing the transmitted and received feedback statuses. A misdetection is recorded when the recovered feedback status is incorrect or undetermined. The example performs this measurement for a transmission between four STAs and an AP by using an end-to-end simulation. The STA infers the transmission parameters from the User Info field of the soliciting NFRP trigger frame. At each signal-to-noise ratio (SNR) point the STA transmits multiple packets with no impairments apart from multipath fading and noise. The STA demodulates the packet and recovers the feedback status. The AP determines the probability of misdetecting feedback status by comparing the recovered and transmitted feedback status.

Each STA transmits a waveform by performing these processing steps.

1 Determine if the STA is scheduled to transmit an HE TB feedback NDP.

2 Determine the RU tone set index and starting space time stream number for all transmitting STAs.

3 Generate an HE TB feedback NDP for the STAs.

4 Pass the waveform for each STA through an indoor channel by using the `wlanTGaxChannel` System object(TM) . Model different channel realizations for each user and each packet by randomly varying the UserIndex property of `wlanTGaxChannel`. This process results in the same spatial correlation properties for all STAs.

5 Combine waveforms from all STAs.

6 Add additive white Gaussian noise (AWGN) to the received waveform. The AWGN creates the desired average SNR per subcarrier after orthogonal frequency-division multiplexing (OFDM) demodulation for each STA.

The receiver (AP) performs these processing steps on the received waveform.

1 Use perfect channel delay estimate to synchronize.

2 Extract the HE-LTF from the synchronized waveform and demodulate the HE-LTF.

3 Recover feedback status information from the demodulated HE-LTF symbols [2 on page 6-0   ].

This figure shows the processing for each link between the STA and AP.
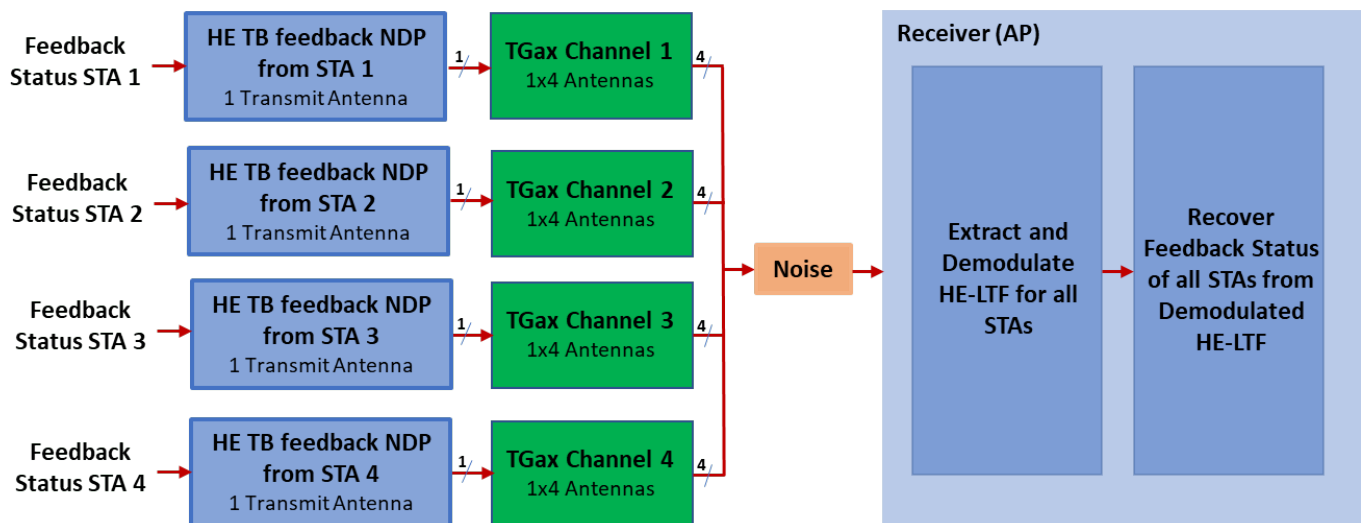
Simulate Uplink Transmission

This section simulates an end-to-end uplink scenario for multiple STAs and SNR points. Specify the number of STAs and the SNR range. This section estimates the probability of misdetecting feedback status for all STAs. The feedback status represents the resource request information from the STA and is defined in Table 26-3 of [1 on page 6-0   ].

```
% Set transmission parameters
numSTAs          =  4                 ; % Number of uplink STAs
STAID            =  [1 19 2 4]        ; % STA association ID, assigned to each associated STA
multiplexingFlag =  [0 1 0 0]        ; % Signaled in the trigger frame of type NFRP for each
feedbackStatus   =  [1 0 1 0]        ; % Resource request information signaled by each STA
startingAID      =  0                 ; % Signaled in the trigger frame of type NFRP
```

```
% Set simulation parameters
numPackets =  50                     ; % Number of packets to simulate

snrRange   =  -2:2:6                 ; % SNR points (dB)

chanBW     =  CBW20          ▼       ; % Channel bandwidth

numTx      =  1                      ; % Assume same number of transmit antennas for all STAs

numRx      =  4                      ; % Number of receive (AP) antennas
```

HE TB Feedback NDP Waveform Configuration

Configure the waveform generator for each STA. The STA performs these steps.

- Check if the STA is scheduled to transmit.
- Calculate the RUToneSetIndex for each STA from STAID, startingAID, and chanBW.
- Calculate starting space time stream number for all STAs from STAID, startingAID, and chanBW.
- Generate configuration object for all STAs.

```
% Return the index of the transmitting STAs. Calculate RUToneSetIndex and starting space time str
[txSTAIndex,ruToneSetIndexPerSTA,startingSTSNumPerSTA] = heTBNDPMappingParams(chanBW,numSTAs,mult
numTxSTAs = numel(txSTAIndex); % STAs scheduled to transmit
cfgSTA = cell(1,numTxSTAs);

% Generate the configuration object and set the feedback status property for all STAs
cfgBase = wlanHETBConfig('ChannelBandwidth',chanBW,'NumTransmitAntennas',numTx,'SpatialMapping',
cfgNDP = getNDPFeedbackConfiguration(cfgBase);
if numSTAs~=numel(feedbackStatus)
    error('The number of elements in FeedbackStatus must be equal to the number of STAs');
end

for u=1:numTxSTAs
    cfgNDP.RUToneSetIndex = ruToneSetIndexPerSTA(u);
    cfgNDP.StartingSpaceTimeStream = startingSTSNumPerSTA(u);
    cfgNDP.FeedbackStatus = feedbackStatus(txSTAIndex(u));
    cfgSTA{u} = cfgNDP;
end
```

Channel Configuration

This example uses a TGax non-line-of-sight (NLOS) indoor channel model with delay profile Model-D. Model-D is considered NLOS when the distance between the transmitter and receiver is greater than or equal to ten meters. For more information, see wlanTGaxChannel. This example assumes that all STAs are at the same distance from the AP.

```
delayProfile =  Model-D          ▼    ; % TGax channel multipath delay profile
tgaxBase = wlanTGaxChannel;
tgaxBase.DelayProfile = delayProfile;
tgaxBase.SampleRate = wlanSampleRate(cfgSTA{1});
tgaxBase.TransmissionDirection = 'Uplink';
tgaxBase.TransmitReceiveDistance = 10;
tgaxBase.ChannelBandwidth = chanBW;
tgaxBase.NumReceiveAntennas = numRx;
```

```
tgaxBase.NormalizeChannelOutputs = false;
tgaxBase.PathGainsOutputPort = true;
```

Create an individual channel for each STA. Each channel is a clone of `tgaxBase`, but with a different `UserIndex` property, and is stored in cell array `tgax`. The `UserIndex` property of each individual channel creates a unique channel for each user. This example uses a random channel realization for each packet by randomly varying the `UserIndex` property of each transmitted packet.

```
% A cell array stores the channel objects, one per STA.
tgax = cell(1,numTxSTAs);
for u=1:numTxSTAs
    tgax{u} = clone(tgaxBase);
    tgax{u}.NumTransmitAntennas = numTx;
    tgax{u}.UserIndex = u;
end
chInfo = info(tgaxBase);
chFilterCoefficients = chInfo.ChannelFilterCoefficients; % Channel filter coefficients
```

Processing SNR Points

This section tests a number of packets at each SNR point and calculates the probability of misdetecting recovered feedback status.

To parallelize processing of the SNR points, you can use a parfor loop. To enable the use of parallel computing for increased speed comment out the 'for' statement and uncomment the 'parfor' statement below.

```
% Processing SNR Points
ofdmInfo = wlanHEOFDMInfo('HE-LTF',cfgNDP);
numSNR = numel(snrRange); % Number of SNR points
misdetectionProbability = zeros(numTxSTAs,numSNR);
ind = wlanFieldIndices(cfgNDP); % Same for all STAs

%parfor isnr=1:numSNR % Use 'parfor' to speed up the simulation
for isnr=1:numSNR
    % Set random substream index per iteration to ensure that each
    % iteration uses a repeatable set of random numbers
    stream = RandStream('combRecursive','Seed',0);
    stream.Substream = isnr;
    RandStream.setGlobalStream(stream);
    rxFeedbackStatus = zeros(numPackets,numTxSTAs);
    chDelay = zeros(1,numTxSTAs);

    for pktIdx=1:numPackets
        rxWaveform = 0;

        % Generate random channel realization for each packet by varying
        % the UserIndex property of the channel. This assumes all STAs
        % have the same number of transmit antennas.
        chPermutations = randperm(numTxSTAs);
        for u=1:numTxSTAs
            % Generate HE TB feedback NDP waveform for each STA
            txSTA = wlanWaveformGenerator([],cfgSTA{u});

            % Pass waveform through a random TGax Channel
            channelIdx = chPermutations(u);
            reset(tgax{channelIdx}); % New channel realization
            [rxSTA,h] = tgax{channelIdx}([txSTA; zeros(50,size(txSTA,2))]);
```

```matlab
            % Perform perfect channel delay estimate to find the start of
            % the packet
            chDelay(u) = channelDelay(h,chFilterCoefficients);

            % Combine uplink waveform from all STAs into one waveform
            rxWaveform = rxWaveform+rxSTA;
        end

        % Synchronize later in time by using the maximum channel delay
        % between all channels as the start of the packet
        pktOffset = max(chDelay(u)); % Packet start index

        % Pass the waveform through AWGN channel. Account for noise
        % energy in unused subcarriers.
        snrVal = snrRange(isnr)-10*log10(ofdmInfo.FFTLength/ofdmInfo.NumTones);
        rxWaveform = awgn(rxWaveform,snrVal);

        % Uplink processing (at the AP)
        rxHELTF = rxWaveform(pktOffset+(ind.HELTF(1):ind.HELTF(2)),:);
        heltfDemod = wlanHEDemodulate(rxHELTF,'HE-LTF',chanBW,cfgNDP.GuardInterval,cfgNDP.HELTFTy

        % Recover feedback status for all STAs
        for u=1:numTxSTAs
            rxFeedbackStatus(pktIdx,u) = wlanHETBNDPFeedbackStatus(heltfDemod,cfgSTA{u});
        end
    end
    % Probability of misdetection per STA
    misdetectionProbability(:,isnr) = 1-sum(rxFeedbackStatus==feedbackStatus(txSTAIndex))/numPack
    disp(['SNR ' num2str(snrRange(isnr)) ' completed']);
end
```

```
SNR -2 completed
SNR 0 completed
SNR 2 completed
SNR 4 completed
SNR 6 completed
```

Plot Probability of Misdetection Against SNR

```matlab
markers = 'ox*sd^v><ph+';
numMarkers = numel(markers);
for u=1:numTxSTAs
    semilogy(snrRange,misdetectionProbability(u,:),['-' markers(mod(u-1,numMarkers)+1)]); hold on
end
xlabel('SNR(dB)');
ylabel('Probability of misdetection');
dataStr = arrayfun(@(x)sprintf('STAID %d',x),STAID(txSTAIndex),'UniformOutput',false);
title(sprintf('%s, Channel %s, %d-by-%d',chanBW,delayProfile,numTx,numRx));
legend(dataStr);
grid on;
```
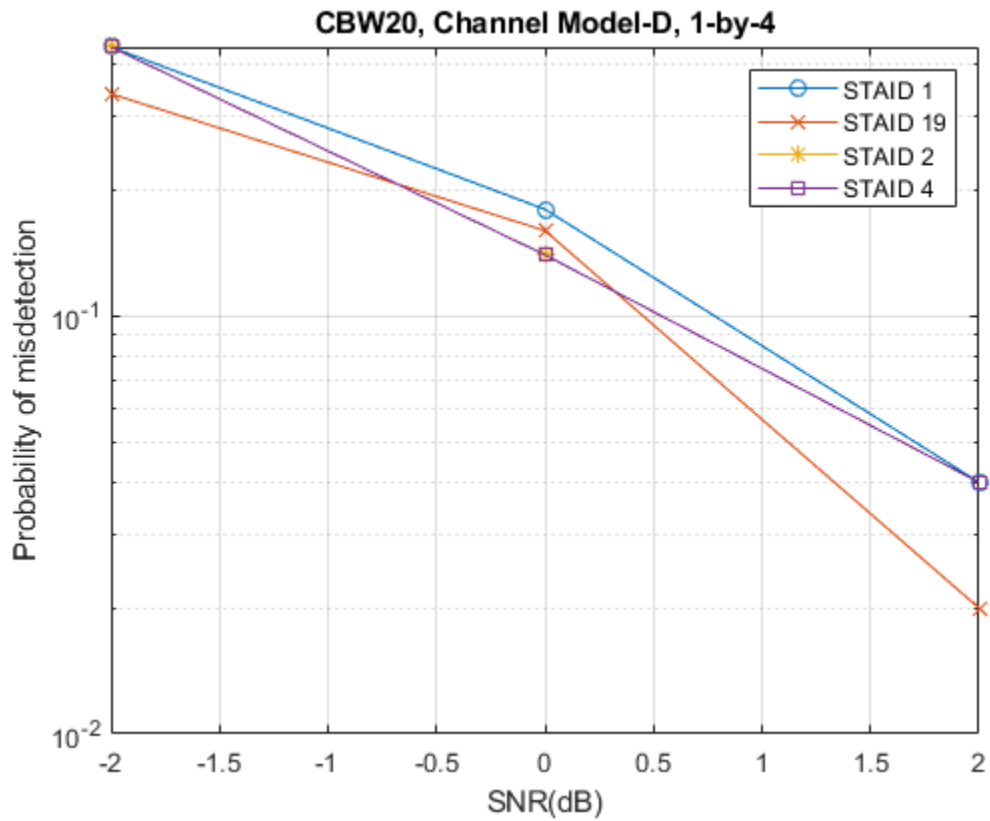
The number of packets tested at each SNR point depends on `numPackets`. For meaningful results, increase the value of `numPackets`. This figure was created by running a longer simulation with `numPackets`:1e4 and `snrRange`:-2:2:6.

CBW20, Channel Model-D, 1-by-4
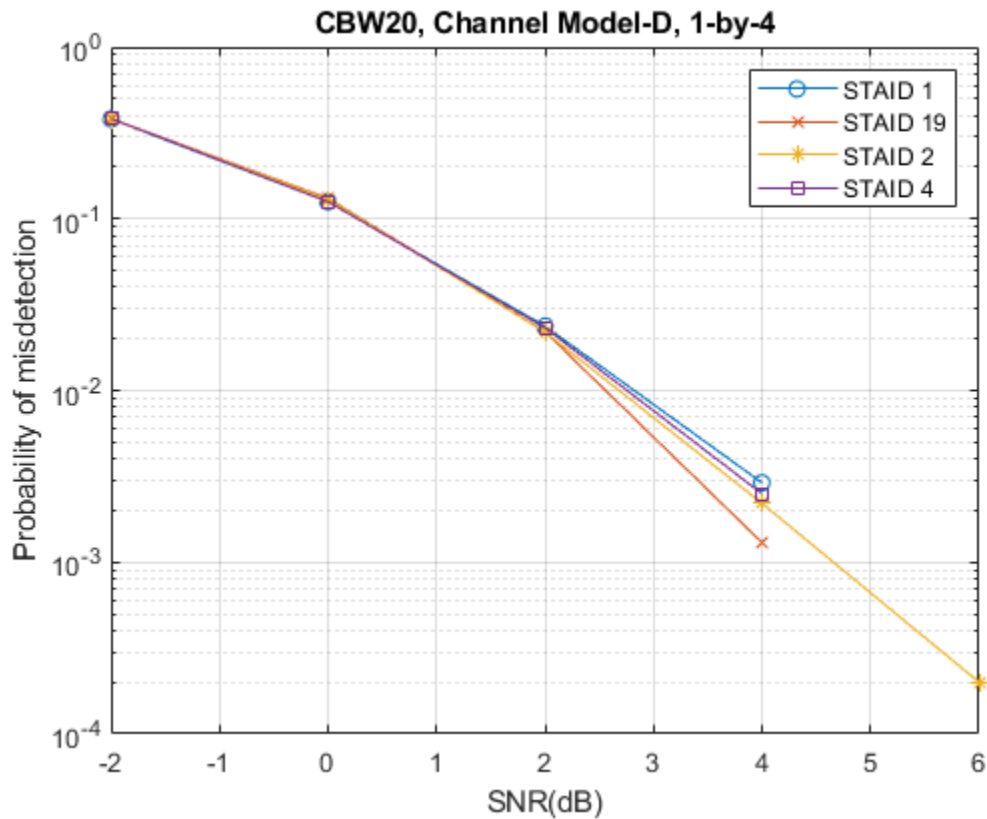
Selected Bibliography

1 IEEE P802.11ax™/D7.0 Draft 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 1: Enhancements for High Efficiency WLAN.

2 Montreuil, L. *et al. NDP Short Feedback Design.* IEEE 802.11-17/0044r4, May 2017.

Three-Dimensional Indoor Positioning with 802.11az Fingerprinting and Deep Learning

This example shows how to train a convolutional neural network (CNN) for localization and positioning by using generated IEEE® 802.11az™ data. Using the trained CNN, you can predict the location or position of multiple stations (STAs) based on a fingerprint consisting of multiple channel impulse responses (CIRs) and evaluate the performance of the CNN.

Introduction

The IEEE 802.11az Wi-Fi™ standard [1 on page 6-0   ], commonly referred to as next generation positioning (NGP), provides physical layer features that enable enhanced ranging and positioning using classical techniques.

Classical techniques rely on line-of-sight (LOS) conditions to effectively extract temporal information, such as time of arrival (ToA), or spatial information, such as angle of arrival (AoA), from a multipath signal to compute a distance or range between devices. When the range between a minimum of three devices can be measured, trilateration can be used to compute a position estimate. For more information about how to use classical ranging and positioning techniques, see the"802.11az Positioning Using Super-Resolution Time of Arrival Estimation" on page 6-63 example.

Fingerprinting and deep learning techniques can be used in Wi-Fi positioning systems to achieve sub-meter accuracies even in non-line-of-sight (NLOS) multipath environments [2 on page 6-0   ]. A *fingerprint* typically contains channel state information (CSI), such as a received signal strength indicator (RSSI) or a channel estimate from a received signal, measured at a specific location in an environment [3 on page 6-0   ].

During the training phase of the network, the example creates a database by sampling the channel fingerprints at multiple known locations in an environment. The network estimates the user location based on a signal received at an unknown location by using the database as a reference.

This example creates a data set of CIR fingerprints by using 802.11az signals in an indoor environment, labeling each fingerprint with its location information. The example trains a CNN to predict STA locations by using a subset of the fingerprints, then evaluates the performance of the trained model by generating predictions of STA locations based on their CIR fingerprint remainder of the data set.

For simplicity, this example uses a small data set, which results in a short simulation time. For more accurate results, use a larger data set. The example provides pretrained models on page 6-0   to show the high levels of performance that can be achieved with more training data.

Simulation Parameters

Indoor Propagation Environment

Generate training data for an indoor office environment specified by the `office.stl` file.

```
mapFileName = "office.stl";
helperVisualizeScenario(mapFileName)
```
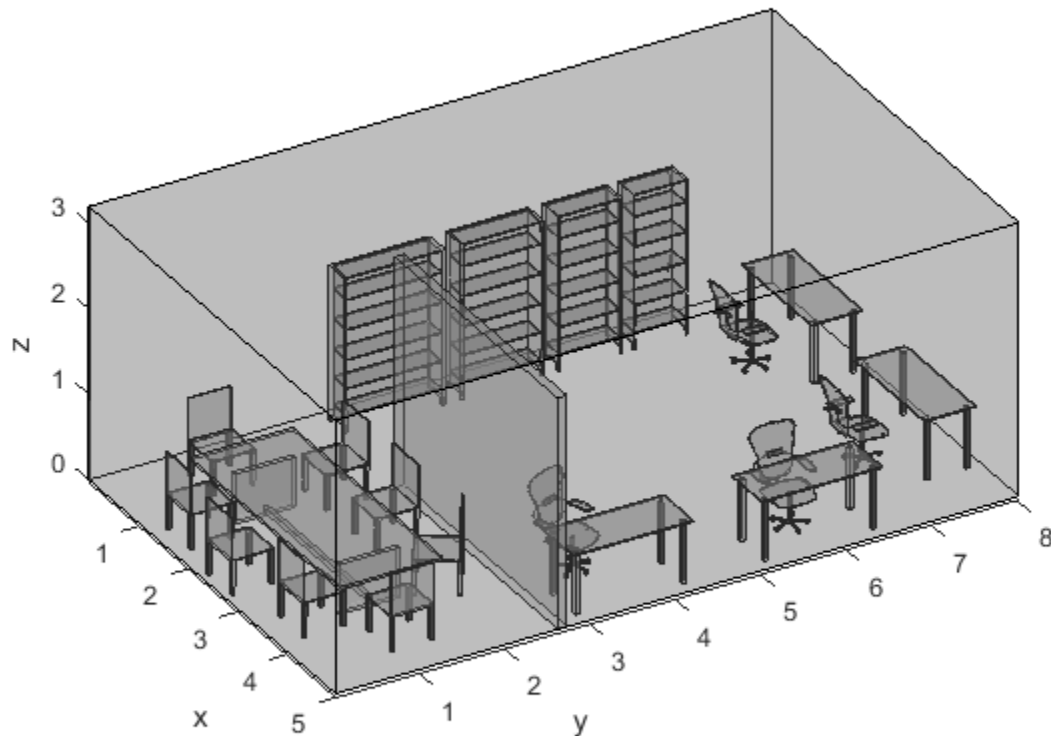
The example places four access points (APs) and a number of STAs that you specify in this environment. The example generates the fingerprints based on the propagation channel defined by the environment, then generates the associated channel impulse response by using ray-tracing techniques. For more information on ray tracing for indoor environments, see the "Indoor MIMO-OFDM Communication Link using Ray Tracing" example.

AP and STA Parameters

Select the size of the transmit and receive antenna arrays, and the channel bandwidth. These parameters control the quantity of data and its resolution within each fingerprint. Larger antenna arrays produce more channel realizations and more CIRs per fingerprint. A higher bandwidth increases the sample rate of the CIR, capturing it in more detail. Changing these parameters makes the data set incompatible with the pretrained models, as the dimensions of each fingerprint must match the shape of the model input layer.

```
S = RandStream("mt19937ar","Seed",5489); % Set the RNG for reproducibility.
RandStream.setGlobalStream(S);
```

txArraySize = [4x1 ▼] ; % Linear transmit array

rxArraySize = [4x1 ▼] ; % Linear receive array

chanBW = [CBW40 ▼] ;

Specify the number of STAs used to map the environment and the distribution of STAs. To distribute STAs uniformly in each dimension as an evenly spaced grid, set `distribution` to `uniform`. To distribute STAs randomly within the environment, set `distribution` to `random`.

When you distribute the STAs uniformly, the example determines the number of STAs by using the value of `staSeparation`, which measures the distance, in meters, between STAs in all dimensions.

When you randomly distribute STAs, specify the number of STAs by setting the value of `numSTAs`.

distribution = [uniform ▼] ;

staSeparation = [0.5] ; % STA separation, in meters, used only when the distribu

numSTAs = [500] ;   % Number of STAs, used only when distribution is random

Localization and Positioning

Determine whether the example performs localization or positioning [4 on page 6-0   ]. Localization is a classification task in which the output of the model is a predicted label related to a discrete area of the map at which a STA is located. Localization is useful for tasks where precise positioning is less important than being able to estimate the general location of a user with a high degree of accuracy, for example, the room on the floor of a building or an aisle in a store. Positioning is a regression task in which the output of the model is the predicted position of a STA. Positioning can estimate the exact location of the user but can have a higher error for positions across an area when compared to localization.

task = [localization ▼] ;

A localization task determines the general location of a STA, rather than its precise location. This diagram shows the layout of the small office with discrete areas used as classes for localization. The red square markers indicate the locations of the APs. The blue dashed box represents the area where the example distributes the STAs during the training process. The example restricts the height of the STAs to a range between 0.8 and 1.8 meters. This range represents a realistic set of values for portable consumer devices. This constraint also minimizes the chance of STAs being placed in unreachable positions.

Office Enivronment
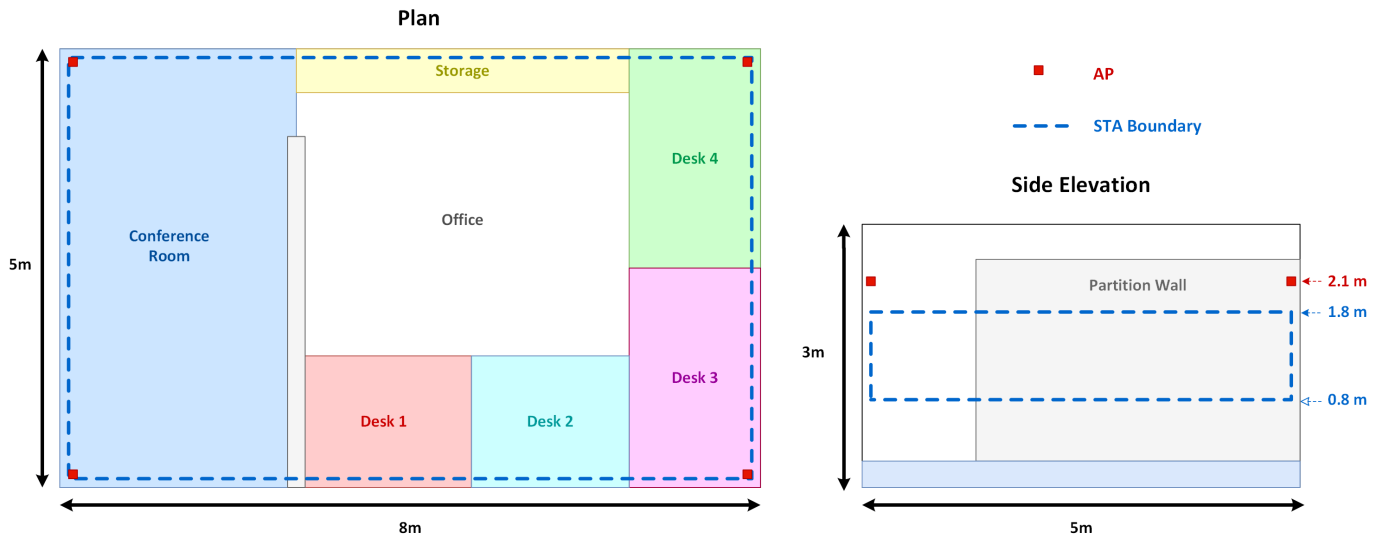
Training Data Synthesis

This section shows how to synthesize the data set for training the CNN.

Generate AP and STA Positions

Generate the AP and STA objects and visualize them in the indoor scenario. If you use a file other than `office.stl`, you must adjust the AP and STA locations within the `dlPositioningCreateEnvironment` function to fit the new environment.

```
if distribution == "uniform"
    [APs,STAs] = dlPositioningCreateEnvironment(txArraySize,rxArraySize,staSeparation,"uniform");
else
    [APs,STAs] = dlPositioningCreateEnvironment(txArraySize,rxArraySize,numSTAs,"random");
end
helperVisualizeScenario(mapFileName,APs,STAs)
view([65.99 32.49])
```
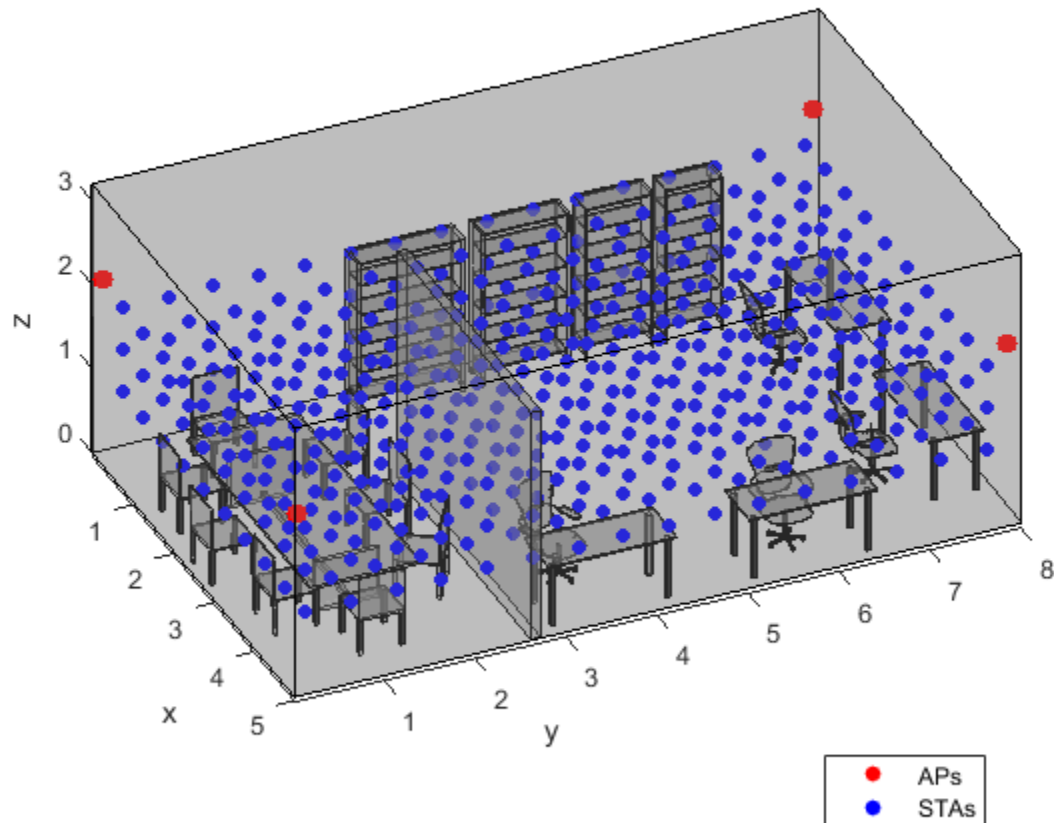
Generate Channel Characteristics by Using Ray-Tracing Techniques

Set the parameters of the ray propagation model. This example considers only LOS and first-order reflections by setting the MaxNumReflections parameter to 1. For more accurate propagation models, set the MaxNumReflections property to 2. Increasing the maximum number of reflections increases the simulation time. To consider only LOS propagation, set the MaxNumReflections property to 0.

```
pm = propagationModel("raytracing-image-method", ...
    "CoordinateSystem","cartesian", ...
    "SurfaceMaterial","wood", ...
    "MaxNumReflections",1);
```

Perform ray-tracing analysis for all AP-STA pairs. The raytrace function returns the generated rays in a cell array of size N_{AP}-by-N_{STA}, where N_{AP} is the number of APs and N_{STA} is the number of STAs.

```matlab
% Perform ray tracing for all transmitters and receivers in parallel
rays = raytrace(APs,STAs,pm,"Map",mapFileName);
size(rays)
```

```
ans = 1×2

     4    480
```

Visualize the calculated rays between all APs and a single STA. The color represents the associated path loss in dB.

```matlab
helperVisualizeScenario(mapFileName,APs,STAs(30))
helperVisualizeRays([rays{:, 30}]);
view([84.75 56.38])
```
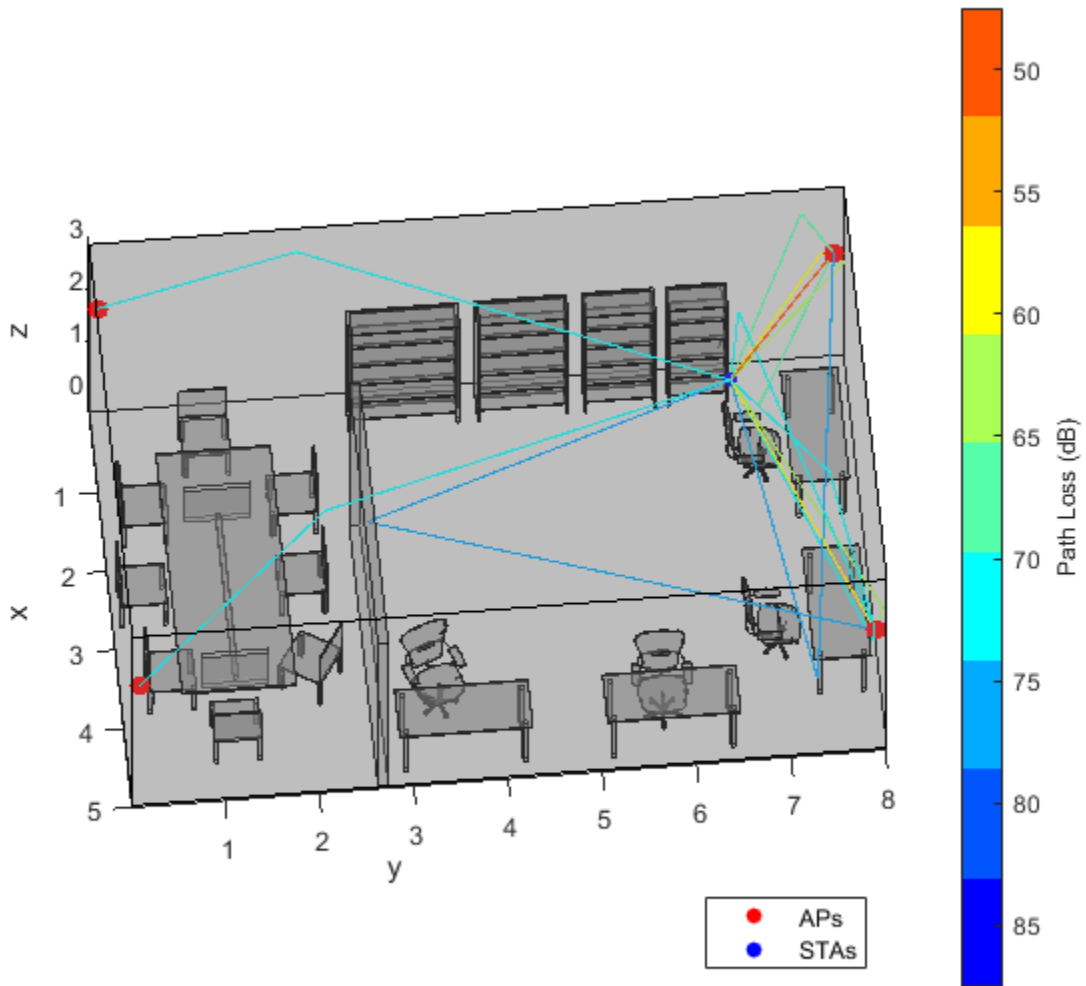
Generate 802.11az CIR Fingerprint Features and Labels

This section shows how to compute the CIRs for each AP-STA pair by using the calculated rays. This diagram shows the processing chain that the example uses to generate the CIRs.
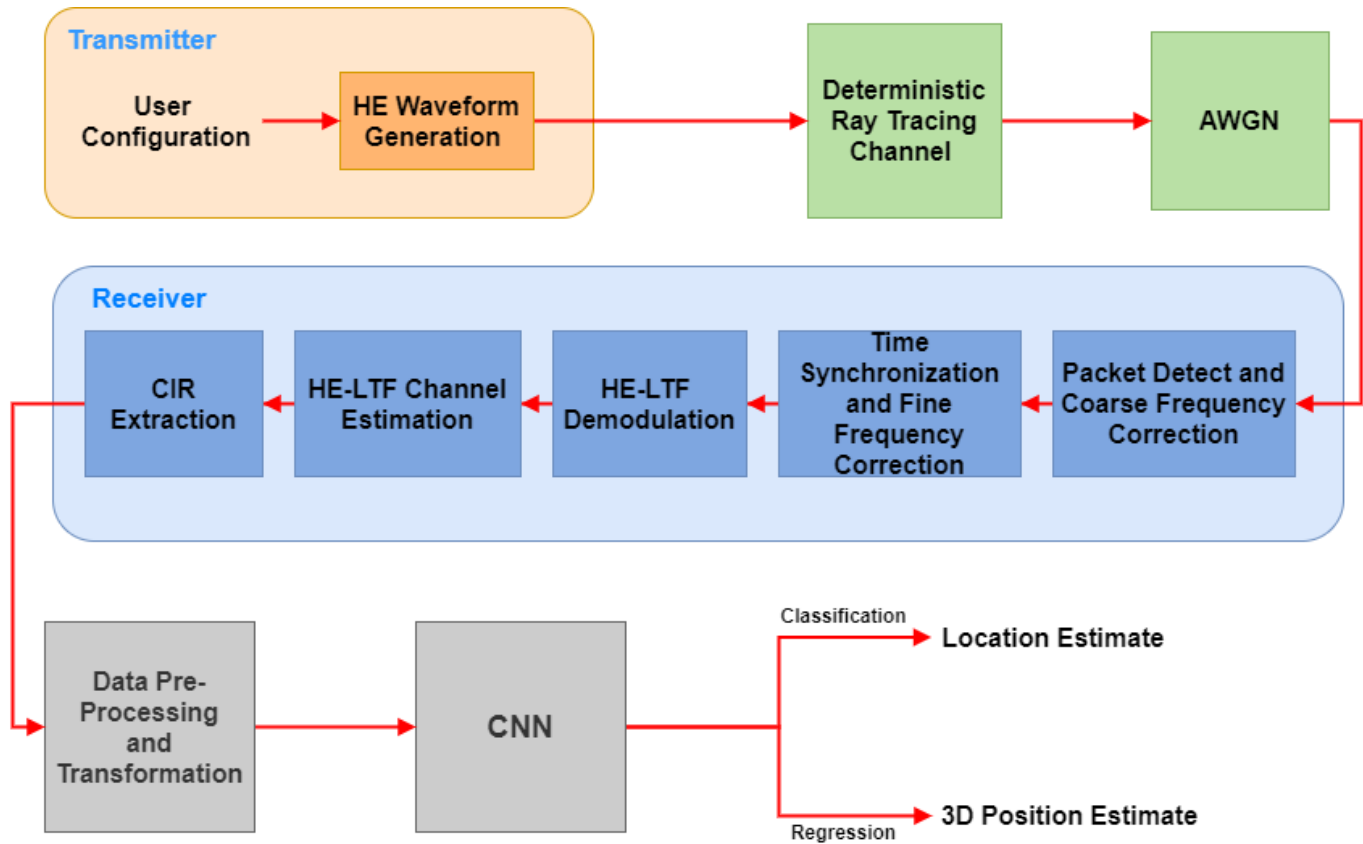
Each AP transmits an 802.11az packet through a noisy channel, and each STA receives the packet. The example assumes that each STA can differentiate between APs, and that no interference occurs between APs.

Packet reception at a location fails if the path between the location and an AP, or if synchronization fails due to low SNR. In this case, the generated CIR is a vector of zeros.

This example uses the magnitude of each multipath component in the CIR as training data. Therefore, the generated CIRs are real-valued. The example stores the CIRs in a four-dimensional array of size N_s-by-$N_{tx\text{-}rx}$-by-N_{AP}-by-N_r.

- N_s is the number of time-domain samples in the CIR.

- $N_{tx\text{-}rx}$ is the total number of transmit-receive antenna pairs.

- N_{AP} is the number of APs.

- N_r is the number of channel realizations for all SNR points.

The example trains the CNN by combining these features with labels of the STA position or location names.

To simulate variations in the environment, repeat the fingerprinting process under different noise conditions by specifying a range of SNR values.

```
snrs = [10 15 20];
```

Configure the waveform parameters. In particular, set the number of space-time streams (STSs) to the size of the transmit antenna array to ensure that the signal from each transmit antenna contributes to the fingerprint of a STA during channel estimation.

```
cfg = heRangingConfig('ChannelBandwidth',chanBW, ...
    "NumTransmitAntennas",prod(txArraySize), ...
    "SecureHELTF",true, ...
    "GuardInterval",1.6);
cfg.User{1}.NumSpaceTimeStreams = prod(txArraySize);
```

Generate the data set.

```
[features,labels] = dlPositioningGenerateDataSet(rays,STAs,APs,cfg,snrs);
```

```
Generating Dataset: 10% complete.
Generating Dataset: 20% complete.
Generating Dataset: 30% complete.
Generating Dataset: 40% complete.
Generating Dataset: 50% complete.
Generating Dataset: 60% complete.
Generating Dataset: 70% complete.
Generating Dataset: 80% complete.
Generating Dataset: 90% complete.
Generating Dataset: 100% complete.
```

Deep Learning

Deep learning uses neural networks to approximate functions across a diverse range of domains. A CNN is a neural network architecture that uses two-dimensional multichannel images. CNNs preserve and learn features from spatial aspects of the data. The deep learning workflow comprises these steps.

1   Define the training data.
2   Define the neural network model.
3   Configure the learning process.
4   Train the model.
5   Evaluate model performance.

Define Training Data

Neural networks are powerful models that can fit a variety of data. To validate results, split the data set into 80% training data and 20% validation data. Training data is the data that the model learns to fit by adjusting its weighted parameters based on the error of its predictions. Validation data is the data that the example uses to ensure that the model is performing as expected on unseen data and not overfitting the training data.

```
[training,validation] = dlPositioningSplitDataSet(features,labels,0.2);
```

Define Neural Network Model

A typical CNN consists of seven main layers.

1   Input layer, which defines the size and type of the input data
2   Convolutional layer, which performs convolution operations on the layer's input by using a set of filters
3   Batch normalization layer, which prevents unstable gradients by normalizing the activations of a layer
4   Activation (ReLU) layer, which a nonlinear activation function that thresholds the output of the previous functional layer
5   Pooling layer, which extracts and pools feature information
6   Dropout layer, which randomly deactivates a percentage of the parameters of the previous layer during training to prevent overfitting
7   Output layer, which defines the size and type of output data

You can form a deep network by arranging layers from these classes and repeating them in blocks. The CNN in this example consists of four blocks, each with a convolution, batch normalization, ReLU, and average pooling layer. The example adds dropout regularization (20%) before the final layers. This architecture is similar to that of the CNN used in the "Train Convolutional Neural Network for Regression" (Deep Learning Toolbox) example, which predicts the rotated positions of handwritten digit images and demonstrates the diverse applications of deep learning models with minimal changes to their parameters.

Construct the CNN.

```
layers = [

    imageInputLayer(size(training.X, 1:3))

    convolution2dLayer(3,256,"Padding","same")
    batchNormalizationLayer
    reluLayer

    averagePooling2dLayer(2,"Stride",2,"Padding","same")

    convolution2dLayer(3,256,"Padding","same")
    batchNormalizationLayer
    reluLayer

    averagePooling2dLayer(2,"Stride",2,"Padding","same")

    convolution2dLayer(3,256,"Padding","same")
    batchNormalizationLayer
    reluLayer

    averagePooling2dLayer(2,"Stride",2,"Padding","same")

    convolution2dLayer(3,256,"Padding","same")
    batchNormalizationLayer
    reluLayer

    averagePooling2dLayer(2,"Stride",2,"Padding","same")

    dropoutLayer(0.2)];
```

Specify the size of the output layer and the activation function for the output, depending on the selected task.

```matlab
if task == "localization"
    layers = [
        layers
        fullyConnectedLayer(7)
        softmaxLayer
        classificationLayer];
else % positioning
    layers = [
        layers
        fullyConnectedLayer(3)
        regressionLayer];
end
```

Match the labels given to the network with the expected model output for each task. The features are an image of CIR fingerprints for both tasks.

```matlab
if task == "localization"
    valY = validation.Y.classification;
    trainY = training.Y.classification;
else % positioning
    valY = validation.Y.regression;
    trainY = training.Y.regression;
end
```

Configure Learning Process and Train Model

Set the number of training data samples that the model evaluates during each training iteration. Increase the number of samples when using a larger data set.

```matlab
miniBatchSize = 32;
```

Set the validation frequency so that the network is validated about once per epoch. The network evaluates the unseen samples to test generalization at the end of each validation period.

```matlab
validationFrequency = floor(size(training.X,4)/miniBatchSize);
```

Specify the options that control the training process. The number of epochs controls how many times the example trains the model consecutively on the full training data set. By default, the example trains the model on a GPU if one is available. Using a GPU required Parallel Computing Toolbox™ and a supported GPU device. For information on supported devices, see .

```matlab
options = trainingOptions("adam", ...
    "MiniBatchSize",miniBatchSize, ...
    "MaxEpochs",3, ...
    "InitialLearnRate", 1e-4,...
    "Shuffle","every-epoch", ...
    "ValidationData",{validation.X,valY'}, ...
    "ValidationFrequency",validationFrequency, ...
    "Verbose",true, ...
    "ResetInputNormalization",true, ...
    "ExecutionEnvironment","auto");
```

Train the model.

```matlab
net = trainNetwork(training.X,trainY',layers,options);

Training on single CPU.
Initializing input data normalization.
```

```
|================================================================================================
| Epoch   |   Iteration  |  Time Elapsed  |  Mini-batch  |  Validation  |  Mini-batch  |  Validati
|         |              |  (hh:mm:ss)    |   Accuracy   |   Accuracy   |    Loss      |    Loss
|================================================================================================
|       1 |            1 |    00:00:03    |    21.88%    |    51.04%    |    1.9794    |     1.0
|       1 |           36 |    00:00:30    |    68.75%    |    64.93%    |    0.9004    |     0.9
|       2 |           50 |    00:00:39    |    59.38%    |              |    1.0421    |
|       2 |           72 |    00:00:56    |    68.75%    |    68.75%    |    0.9528    |     0.8
|       3 |          100 |    00:01:14    |    62.50%    |              |    0.9362    |
|       3 |          108 |    00:01:21    |    56.25%    |    67.01%    |    1.0847    |     0.8
|================================================================================================
|
```

Evaluate Model Performance

Investigate the model performance by examining predicted values for the validation data. Predict labels by passing the validation set features through the network, then evaluate the performance of the network by comparing the predicted labels with the validation set labels.

```
if task == "localization"
    YPred = net.classify(validation.X);
else % positioning
    YPred = net.predict(validation.X);
end
```

Generate a visual and statistical view of the results.

For localization, the dlPositioningPlotResults function generates a three-dimensional map, which displays the true locations of STAs. The color assigned to each STA denotes its predicted location. The function also generates a confusion matrix in which the rows correspond to the predicted class and the columns correspond to the true class. The diagonal cells correspond to observations that are correctly classified. The off-diagonal cells correspond to incorrectly classified observations. If the network performs well, elements in the main diagonal are significantly larger than the other matrix elements.

For positioning, the color assigned to each STA denotes the distance error of the predicted position. The function also generates a cumulative distribution function (CDF). The y-axis measures the proportion of the data for which the measured distance error is less than or equal to the corresponding value on the x-axis.

```
metric = dlPositioningPlotResults(mapFileName,validation.Y,YPred,task);
```

CNN Location Prediction

True STA positions coloured by Predicted Class

Accuracy: 69.7917%

Pretrained Model

This example uses a small data set and limited training period, which produces modest results. You can obtain more accurate results by using a CNN pretrained with a large data set. Download the resulting CNN for positioning and localization from https://www.mathworks.com/supportfiles/spc/wlan/Positioning80211azExample/pretrained_networks.zip. This zip file contains two MAT files:

- localization_cbw40_4x4.mat - CNN trained for localization
- positioning_cbw40_4x4.mat - CNN trained for positioning

All models were trained for 100 epochs with a `miniBatchSize` value of 256 on a data set of fingerprints from a uniform distribution of STAs with a `staSeparation` value of `0.1`. The bandwidth was 40 MHz bandwidth with four-antenna linear arrays in the transmitter and receiver (4×4 MIMO).

To explore the performance of these pretrained models, download and extract the zip file, load the appropriate MAT file into the workspace, and run the Evaluate Model Performance on page 6-0 section again.

Effect of Antenna Array Size and Bandwidth

The pretrained models show that an increased number of STAs and training epochs improve performance. This section shows the effect of channel bandwidth and size of antenna arrays on performance.

These figures show the impact of channel bandwidth on positioning and localization performance. These results were generated by training a model for 100 epochs with a `miniBatchSize` value of

256 with a data set compiled from a uniform distribution of STAs sampled at intervals of 0.1 meters, in which each STA and AP contained a four-element linear antenna array. Using a larger bandwidth results in more accurate estimation as a higher sampling rate increases the resolution of CIRs.
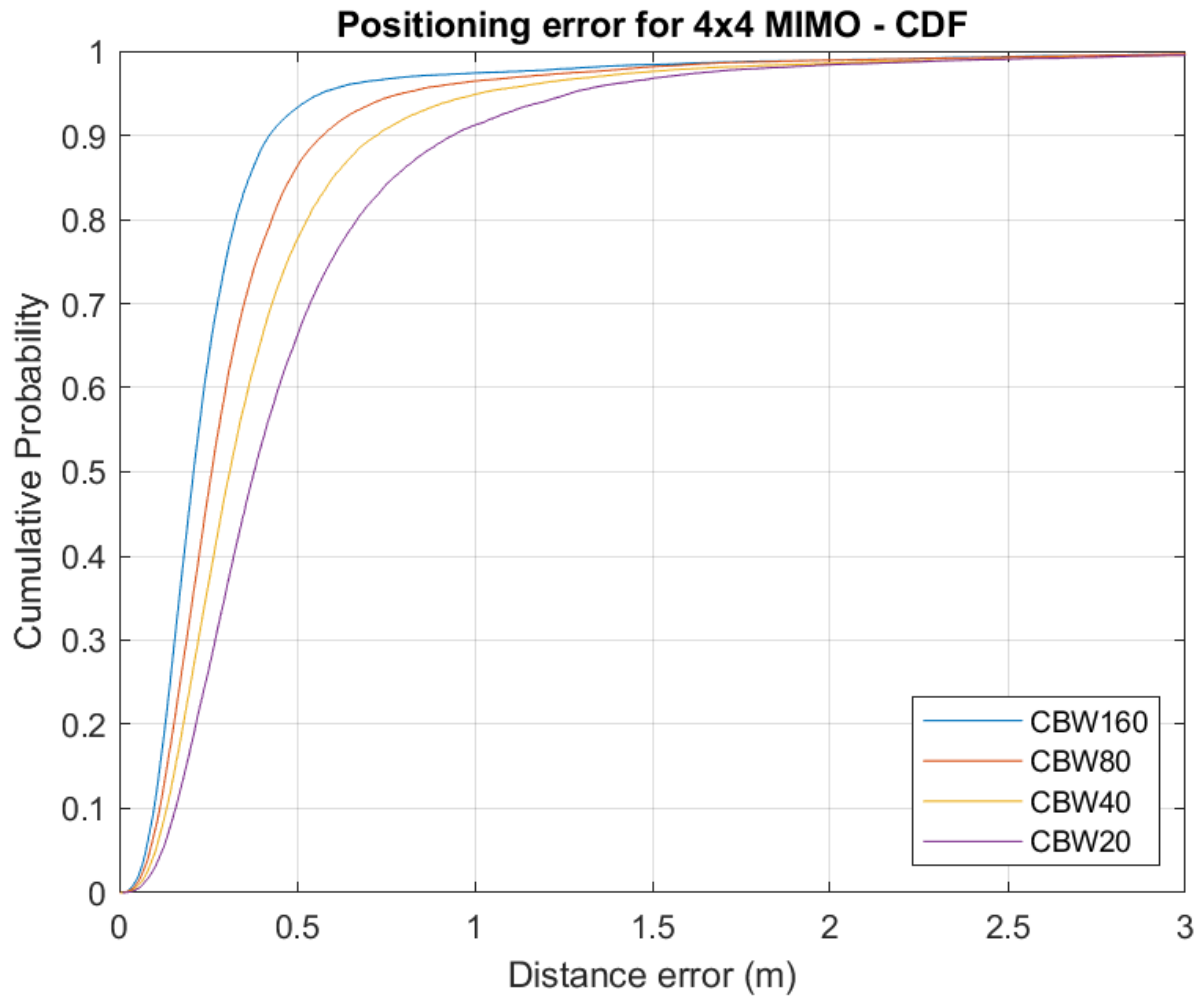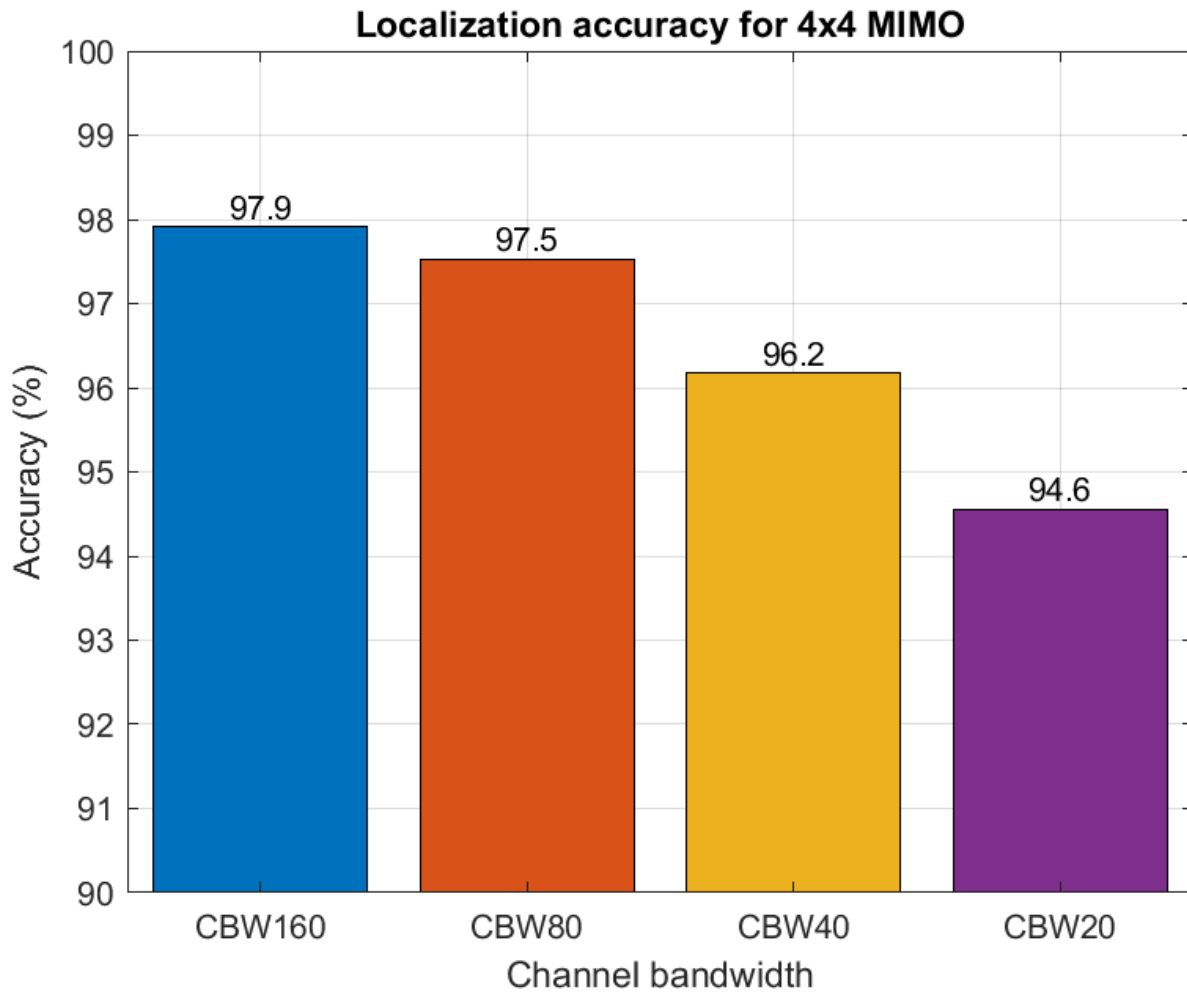
These figures show the impact of the number of transmit and receive antennas. These results were generated using a bandwidth of 20 MHz. The accuracy increases for larger antenna arrays due to the presence of more channel information. For example, a 4×4 channel contains 16 channel realizations, whereas a 2×2 channel contains only four. As expected, the CNN performs better when trained with more data.

Positioning error for 20 MHz channel bandwidth - CDF

Localization accuracy for 20 MHz channel bandwidth
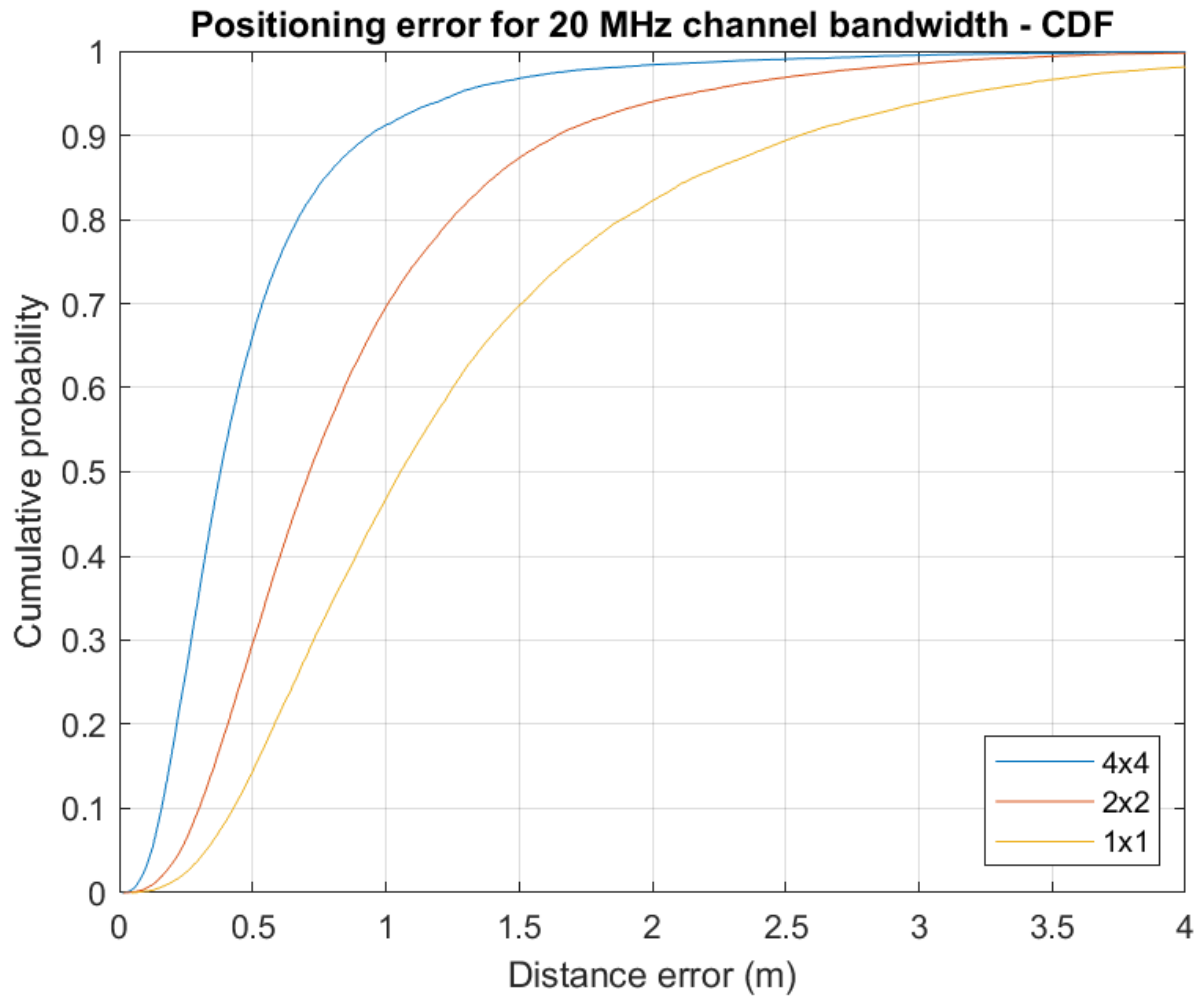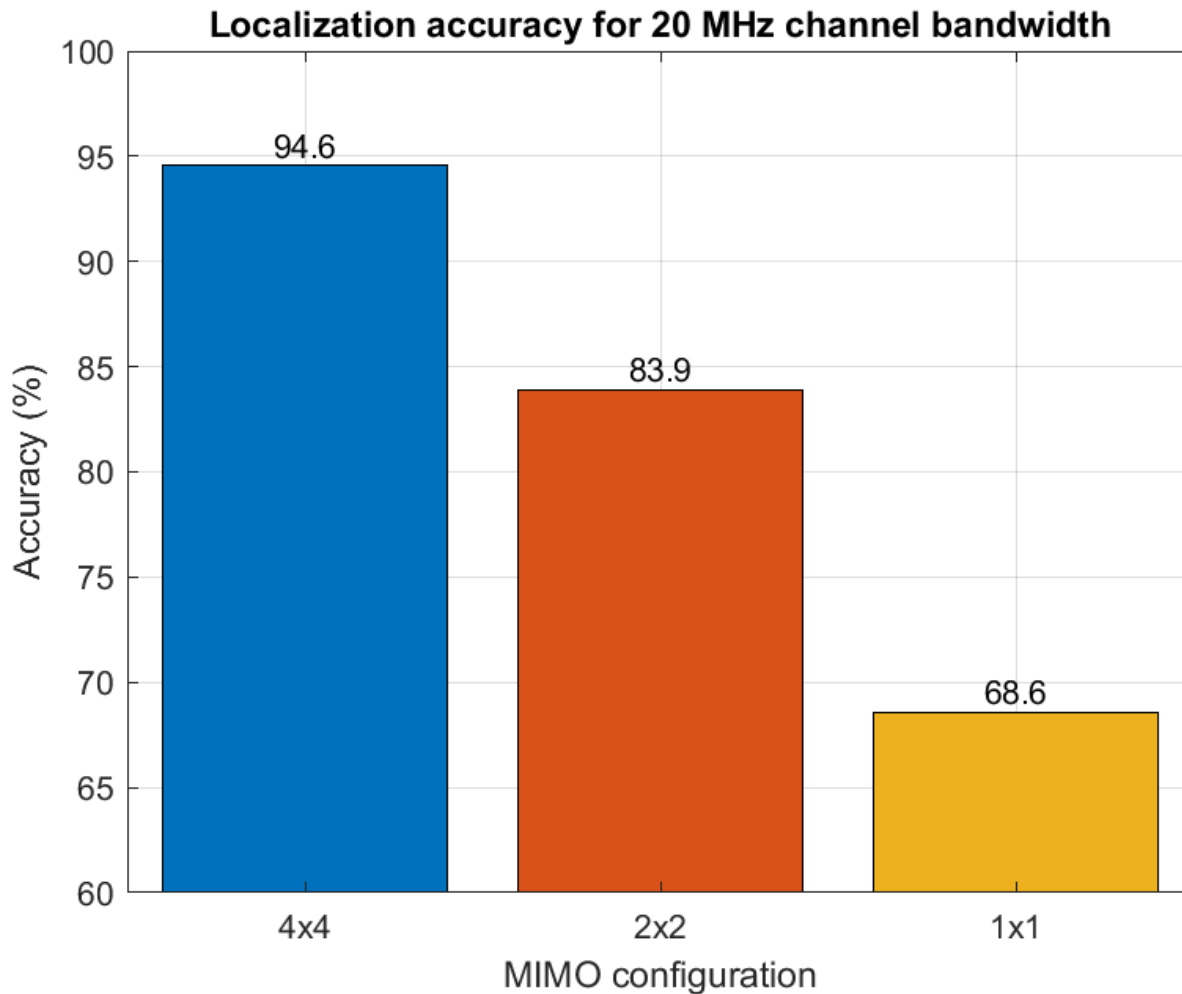
These results were generated with data samples which were not used to train the CNNs. If data samples used to train the CNN are also used to evaluate performance the results may differ.

References

1 IEEE P802.11az™/D2.6 Draft Standard for Information technology— Telecommunications and information exchange between systems Local and metropolitan area networks— Specific requirements - Amendment 3: Enhancements for positioning.

2 Kokkinis, Akis, Loizos Kanaris, Antonio Liotta, and Stavros Stavrou. "RSS Indoor Localization Based on a Single Access Point." Sensors 19, no. 17 (August 27, 2019): 3711. https://doi.org/10.3390/s19173711.

3 Wang, Xuyu, Lingjun Gao, Shiwen Mao, and Santosh Pandey. "CSI-Based Fingerprinting for Indoor Localization: A Deep Learning Approach." IEEE Transactions on Vehicular Technology, 2016, 763–776. https://doi.org/10.1109/TVT.2016.2545523.

4 Groves, Paul D. Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems. Boston: Artech House, 2008.

802.11az Positioning Using Super-Resolution Time of Arrival Estimation

This example shows how to estimate the position of a station (STA) in a multipath environment by using a time-of-arrival-based (ToA-based) positioning algorithm defined in the IEEE® 802.11az™ Wi-Fi™ standard. The example estimates the ToA by using a multiple signal classification (MUSIC) super-resolution approach, then estimates the two-dimensional position of a STA by using trilateration. The example then evaluates and compares the performance of the positioning algorithm at multiple signal-to-noise ratio (SNR) points.

Introduction

The 802.11az standard [1 on page 6-0   ], commonly referred to as next generation positioning (NGP), enables a STA to identify its position relative to multiple access points (APs). This standard supports two high-efficiency (HE) ranging physical layer (PHY) protocol data unit (PPDU) formats:

- HE ranging null data packet (NDP)
- HE trigger-based (TB) ranging NDP

The HE ranging NDP and HE TB ranging NDP are the respective analogues of the HE sounding NDP and HE TB feedback NDP PPDU formats, as defined in the 802.11ax™ standard. For more information on these HE PPDU formats, see [2] on page 6-0   .

The HE ranging NDP supports the positioning of one or more users with an optional secure HE long training field (HE-LTF) sequence. The single-user HE ranging waveform contains HE-LTF symbols for a single user, which also support an optional secure HE-LTF sequence. The multi-user HE ranging waveform permits only secure HE-LTF symbols for multiple users. To improve position estimation accuracy, single-user and multi-user waveforms can contain multiple repetitions of the HE-LTF symbols. To parameterize and generate HE ranging NDPs, see the "802.11az Waveform Generation" on page 1-2 example.

This example simulates an 802.11az network consisting of a STA and multiple APs. To estimate the position of a STA, the network requires a minimum of three APs. The example simulates a ranging measurement exchange for each STA-AP pair, then trilaterates the position of the STA by using these measurements. The example repeats this simulation for multiple iterations and SNR points. This diagram shows the positioning process in a network with one STA and three APs.
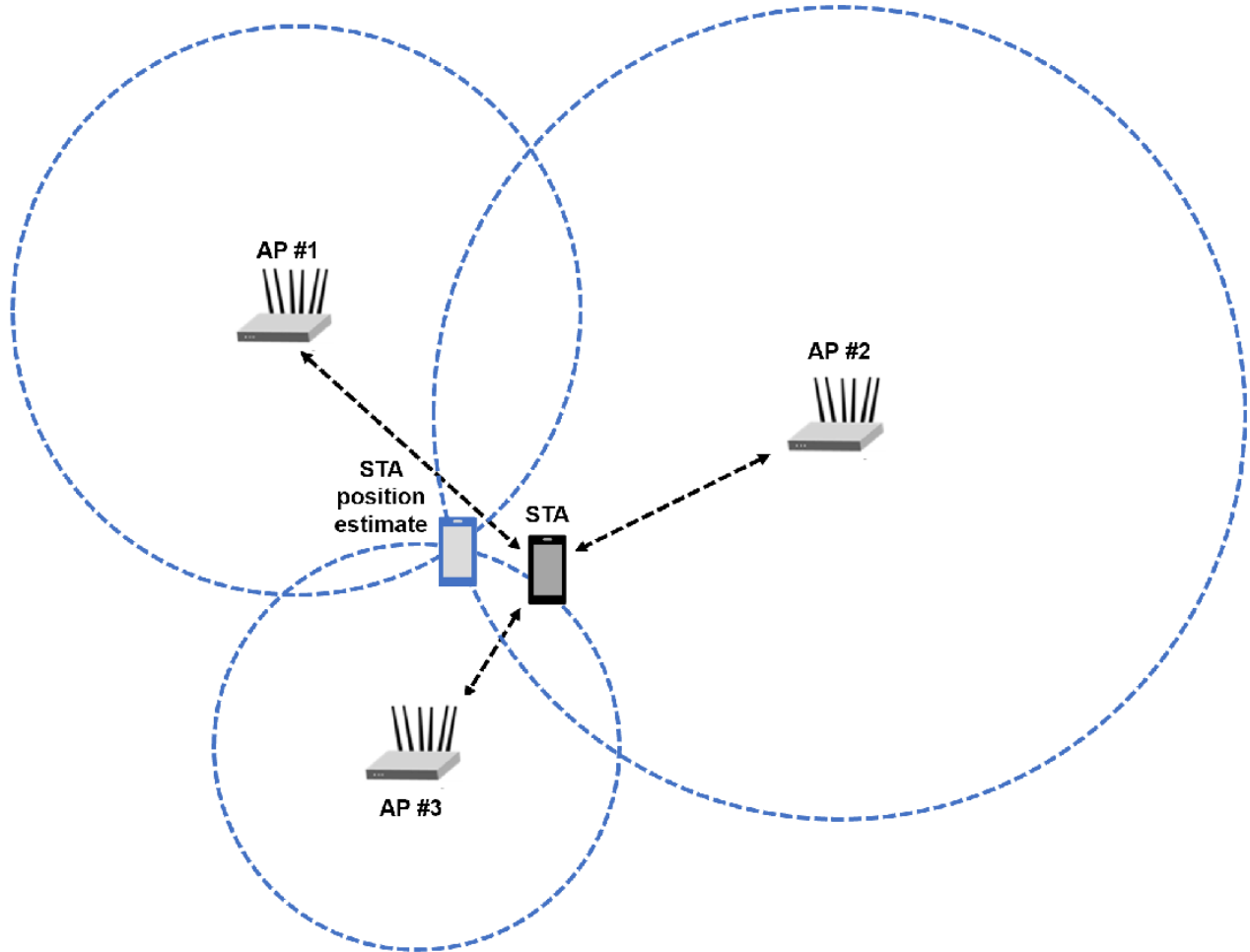
Packet Transmission and Reception

This example models the measurement exchange between the STA and APs by performing these steps.

1   Generate a ranging NDP

2   Delay the NDP according to a randomly generated distance between the STA and AP, adding fractional and integer sample delay

3   Pass the waveform through an indoor TGax channel. The example models different channel realizations for different packets

4   Add additive white Gaussian noise (AWGN) to the received waveform. The example uses the same SNR value for all the links between STA and APs

5   Perform synchronization and frequency correction on the received waveform

6   Demodulate the HE-LTF

7   Estimate the channel frequency response from the HE-LTF

8   Estimate the distance by using the MUSIC super-resolution algorithm

9   Combine distance estimates from other STA-AP pairs and trilaterate the position of the STA

This figure illustrates the processing for each STA-AP link.
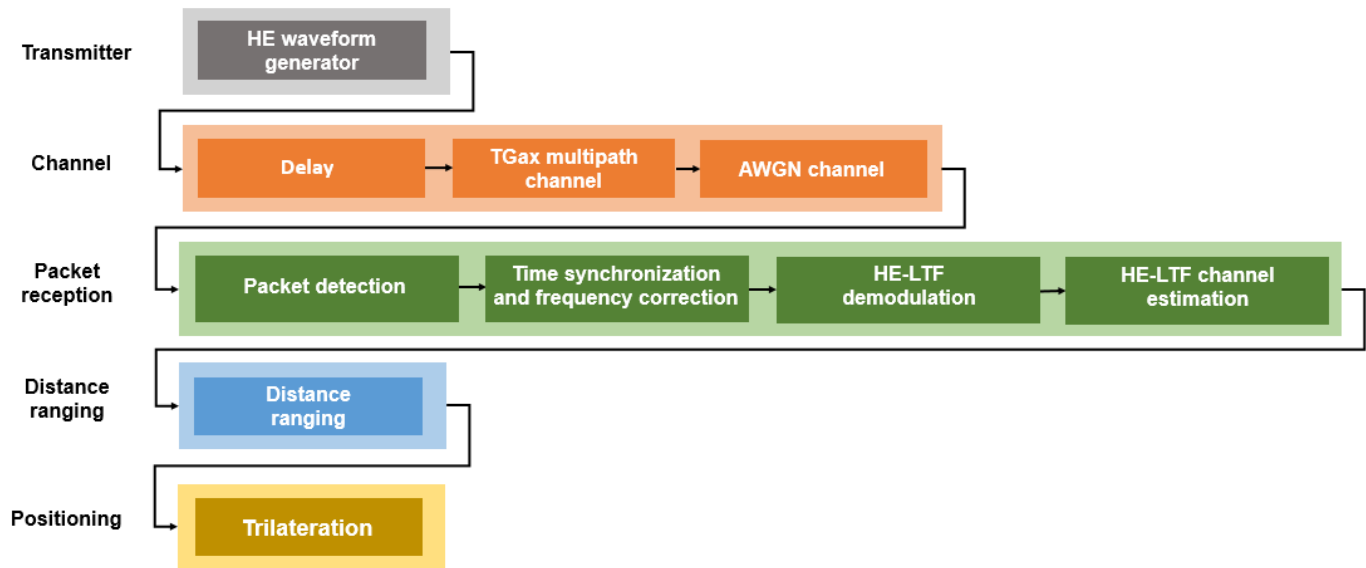
Distance Ranging

The example performs a distance ranging measurement between the STA and each AP by capturing the timestamps of the NDP. The STA records the time t_1 (UL ToD) at which it transmits the uplink NDP (UL NDP). The AP then captures the time t_2 (UL ToA) at which it receives the UL NDP and records the time t_3 (DL ToD) at which it transmits the downlink NDP (DL NDP). The STA then captures the time t_4 (DL ToA) at which it receives the DL NDP. This diagram illustrates the measurement sounding phase between a STA and a single AP [1 on page 6-0].
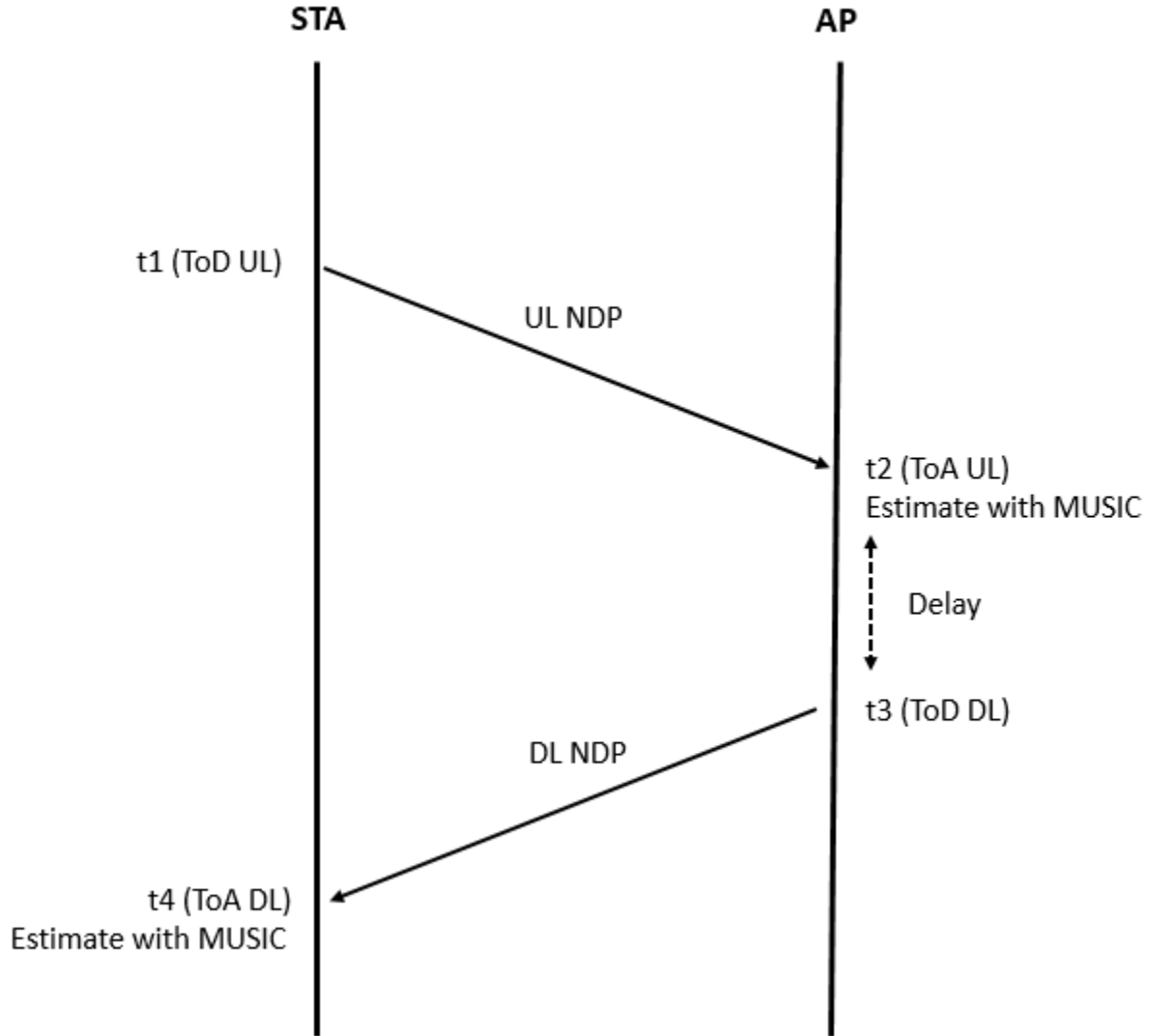
The example estimates T_{RTT} the round trip time (RTT), by combining these timestamps.

$$T_{RTT} = (t_4 - t_1) - (t_3 - t_2)$$

The example then computes the distance d between the STA and AP by using this equation.

$d = \dfrac{T_{RTT}}{2}c$, where c is speed of light

The example estimates t_2 and t_4 by using MUSIC super-resolution. To compute these estimates, the example performs these steps [3 on page 6-0   ].

1   Interpolate across missing subcarriers from the channel frequency response (CFR), assuming uniform subcarrier spacing
2   Estimate the CFR correlation matrix
3   Decorrelate the multipaths by using spatial smoothing

4 Improve the correlation matrix estimate by performing forward-backward averaging. The example assumes that CFR estimates from multiple spatial streams are different CFR snapshots, and uses all the snapshots for correlation matrix estimation.

5 Run the MUSIC algorithm. Perform eigendecomposition on the correlation matrix to separate it into signal and noise subspaces. Estimate the time-domain delay profile by finding all instances where signal and noise subspaces are orthogonal. The example assumes that the precise signal subspace dimension, equal to the number of multipaths, is known.

6 Determine the ToA by finding the first peak of the recovered multipaths in the estimated delay profile, assumed to be the direct-line-of-sight (DLOS) path

This diagram illustrates the distance ranging process.
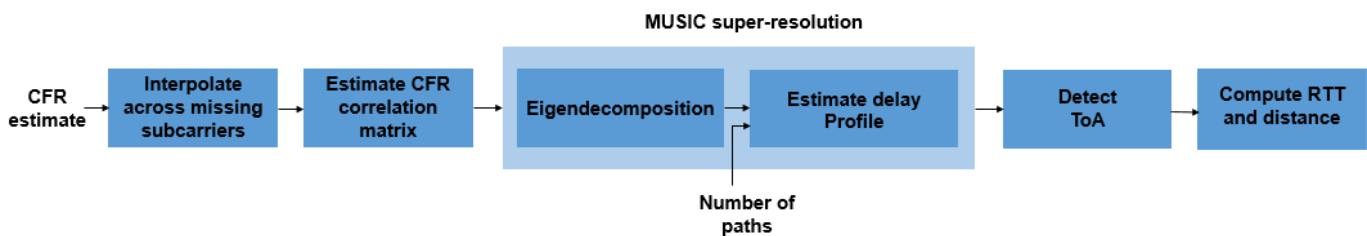
This plot compares the true multipath delay profile and the MUSIC estimated delay profile for a single 802.11az link simulation.

```
heRangingPlotDelayProfile()
```
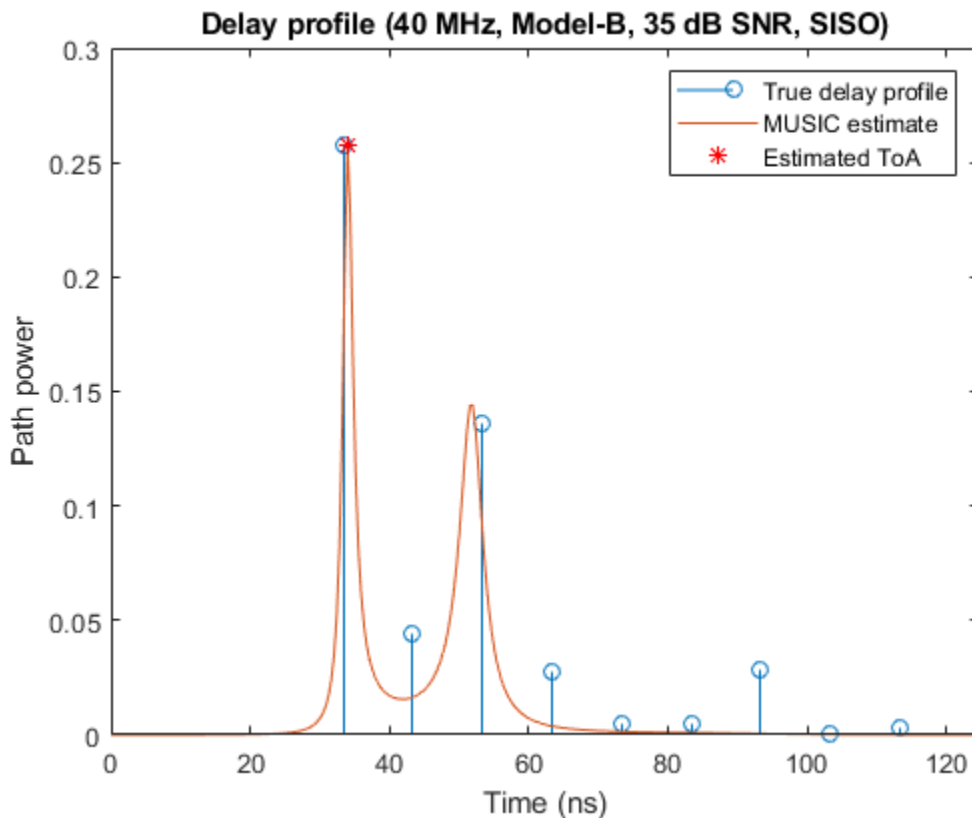
Simulation Parameters

This example performs a ranging and positioning simulation for multiple iterations and SNR points. At each iteration, the AP and the STA exchange multiple uplink and downlink packets. The example estimates the ranging error between the AP and the STA for each iteration by comparing the estimated distance between the AP and the STA with the known distance.

Specify the number of iterations, SNR points, and APs in the networks. To estimate the position of a STA, the network requires a minimum of three APs. For each iteration, use a different random set of AP positions, a different channel realization, and a different AWGN profile. The example generates a cumulative distribution function (CDF) for the absolute ranging error by using the ranging measurements from all iterations and all STA-AP pairs.

```
numIterations = 50                    ; % Number of iterations
snrRange = 15:10:35                   ; % SNR points, in dB
numAPs = 3                            ; % Number of APs
```

802.11az Waveform Configuration

Configure waveform generators for each AP and the STA.

```
chanBW = CBW20                        ; % Channel bandwidth
numTx  = 2                            ; % Number of transmit antennas
numRx  = 2                            ; % Number of receive antennas
numSTS = 2                            ; % Number of space-time streams
numLTFRepetitions = 3                 ; % Number of HE-LTF repetitions
```

Configure the HE ranging NDP parameters of the STA.

```
cfgSTABase = heRangingConfig;
cfgSTABase.ChannelBandwidth = chanBW;
cfgSTABase.NumTransmitAntennas = numTx;
cfgSTABase.SecureHELTF = true;
cfgSTABase.User{1}.NumSpaceTimeStreams = numSTS;
cfgSTABase.User{1}.NumHELTFRepetition = numLTFRepetitions;
cfgSTABase.GuardInterval = 1.6;
```

Configure the HE ranging NDP parameters of the APs.

```
cfgAPBase = cell(1,numAPs);
for iAP = 1:numAPs
    cfgAPBase{iAP} = heRangingConfig;
    cfgAPBase{iAP}.ChannelBandwidth = chanBW;
    cfgAPBase{iAP}.NumTransmitAntennas = numTx;
    cfgAPBase{iAP}.SecureHELTF = true;
    cfgAPBase{iAP}.User{1}.NumSpaceTimeStreams = numSTS;
    cfgAPBase{iAP}.User{1}.NumHELTFRepetition = numLTFRepetitions;
    cfgAPBase{iAP}.GuardInterval = 1.6;
end
```

```
ofdmInfo = wlanHEOFDMInfo('HE-LTF',chanBW,cfgSTABase.GuardInterval);
sampleRate = heRangingSampleRate(cfgSTABase);
```

Channel Configuration

Configure the WLAN TGax multipath channel by using the `wlanTGaxChannel` System object™. This System object can generate a channel with a dominant direct path, in which the DLOS path is the strongest path, or a channel with a non-dominant direct path, for which the DLOS path is present, but not the strongest path.

```
delayProfile = Model-B ▼ ; % TGax channel multipath delay profile

carrierFrequency = 5e9; % Carrier frequency, in Hz
speedOfLight = physconst('lightspeed');

chanBase = wlanTGaxChannel;
chanBase.DelayProfile = delayProfile;
chanBase.NumTransmitAntennas = numTx;
chanBase.NumReceiveAntennas = numRx;
chanBase.SampleRate = sampleRate;
chanBase.CarrierFrequency = carrierFrequency;
chanBase.ChannelBandwidth = chanBW;
chanBase.PathGainsOutputPort = true;
chanBase.NormalizeChannelOutputs = false;
```

Get channel filter delay and the number of paths

```
chBaseInfo = info(chanBase);
chDelay = chBaseInfo.ChannelFilterDelay;
numPaths = size(chBaseInfo.PathDelays,2);
```

Ranging Measurement

Run a ranging simulation with multiple iterations for all STA-AP pairs. Display the ranging mean absolute error (MAE) and the ranging error CDF for each SNR point.

```
delayULDL = 16e-6; % Time delay between UL NDP ToA and DL NDP ToD, in seconds

numSNR = numel(snrRange);
distEst = zeros(numAPs,numIterations,numSNR);  % Estimated distance
distance = zeros(numAPs,numIterations,numSNR); % True distance
positionSTA = zeros(2,numIterations,numSNR);    % Two-dimensional position of the STA
positionAP= zeros(2,numAPs,numIterations,numSNR); % Two-dimensional positions of the APs
per = zeros(numSNR,1); % Packet error rate (PER)

%parfor isnr = 1:numSNR % Use 'parfor' to speed up the simulation
for isnr = 1:numSNR

    % Use a separate channel and waveform configuration object for each parfor stream
    chan = chanBase;
    cfgAP = cfgAPBase;
    cfgSTA = cfgSTABase;

    % Initialize ranging error and total failed packet count variables
    rangingError = 0;
    failedPackets = 0;
```

```matlab
% Set random substream index per iteration to ensure that each
% iteration uses a repeatable set of random numbers
stream = RandStream('combRecursive','Seed',123456);
stream.Substream = isnr;
RandStream.setGlobalStream(stream);

% Define the SNR per active subcarrier to account for noise energy in nulls
snrVal = snrRange(isnr) - 10*log10(ofdmInfo.FFTLength/ofdmInfo.NumTones);

for iter = 1:numIterations

    % Gennerate random AP positions
    [positionSTA(:,iter,isnr),positionAP(:,:,iter,isnr),distanceAllAPs] = heGeneratePositions
    distance(:,iter,isnr) = distanceAllAPs;

    % Range-based delay
    delay = distance(:,iter,isnr)/speedOfLight;
    sampleDelay = delay*sampleRate;

    % Loop over the number of APs
    for ap = 1:numAPs

        linkType = ["Uplink","Downlink"];

        % ToD of UL NDP (t1)
        todUL = randsrc(1,1,0:1e-9:1e-6);

        % Loop for both UL and DL transmission
        numLinks = numel(linkType);
        txTime = zeros(1,numLinks);

        for l = 1:numLinks
            if linkType(l) == "Uplink" % STA to AP
                cfgSTA.UplinkIndication = 1; % For UL
                % Generate a random secure HE-LTF sequence for the exchange
                cfgSTA.User{1}.SecureHELTFSequence = dec2hex(randsrc(1,10,(0:15)))';
                cfg = cfgSTA;
            else % AP to STA
                % Generate a random secure HE-LTF sequence for the exchange
                cfgAP{ap}.User{1}.SecureHELTFSequence = dec2hex(randsrc(1,10,(0:15)))';
                cfg = cfgAP{ap}; % For DL
            end

            % Set different channel for UL and DL, assuming that the channel is not reciproca
            reset(chan)

            % Generate HE Ranging NDP transmission
            tx = heRangingWaveformGenerator(cfg);

            % Introduce time delay (fractional and integer) in the transmit waveform
            txDelay = heDelaySignal(tx,sampleDelay(ap));

            % Pad signal and pass through multipath channel
            txMultipath = chan([txDelay;zeros(50,cfg.NumTransmitAntennas)]);

            % Pass waveform through AWGN channel
            rx = awgn(txMultipath,snrVal);
```

```matlab
                % Perform synchronization and channel estimation
                [chanEstActiveSC,integerOffset] = heRangingSynchronize(rx,cfg);

                % Estimate the transmission time between UL and DL
                if ~isempty(chanEstActiveSC) % If packet detection is successful

                    % Estimate fractional delay with MUSIC super-resolution
                    fracDelay = heRangingTOAEstimate(chanEstActiveSC,ofdmInfo.ActiveFFTIndices,
                                            ofdmInfo.FFTLength,sampleRate,numPaths);

                    integerOffset = integerOffset - chDelay; % Account for channel filter delay
                    intDelay = integerOffset/sampleRate; % Estimate integer time delay
                    txTime(l) = intDelay + fracDelay; % Transmission time

                else % If packet detection fails
                    txTime(l) = NaN;
                end

            end

            if ~any(isnan(txTime)) % If packet detection succeeds

                % TOA of UL waveform (t2)
                toaUL = todUL + txTime(1);

                % Time of departure of DL waveform (t3)
                todDL = toaUL + delayULDL;

                % TOA DL waveform (t4)
                toaDL = todDL + txTime(2);

                % Compute the RTT
                rtt = (toaDL-todUL) - (todDL-toaUL);

                % Estimate the distance between the STA and AP
                distEst(ap,iter,isnr) = (rtt/2)*speedOfLight;
                % Accumulate error to MAE
                rangingError = rangingError + abs(distanceAllAPs(ap) - distEst(ap,iter,isnr));

            else % If packet detection fails
                distEst(ap,iter,isnr) = NaN;
                failedPackets = failedPackets + 1;
            end

        end
    end
    mae = rangingError/((numAPs*numIterations) - failedPackets); % MAE for successful packets
    per(isnr) = failedPackets/(numAPs*numIterations); % PER
    if(per(isnr) > 0.01) % Use only successful packets for ranging and positioning
        warning('wlan:discardPacket','At SNR = %d dB, %d%% of packets were discarded',snrRange(is
    end
    disp(['At SNR = ',num2str(snrRange(isnr)),' dB, ','Ranging mean absolute error = ',num2str(ma
end
```

```
At SNR = 15 dB, Ranging mean absolute error = 0.57082 meters.
At SNR = 25 dB, Ranging mean absolute error = 0.40224 meters.
At SNR = 35 dB, Ranging mean absolute error = 0.25439 meters.
```

```
% Reshape to consider all packets within one SNR point as one dataset
rangingError = reshape(abs(distance - distEst),[numAPs*numIterations,numSNR]);
hePlotErrorCDF(rangingError,snrRange)
xlabel('Absolute ranging error (meters)')
title('Ranging Error CDF')
```
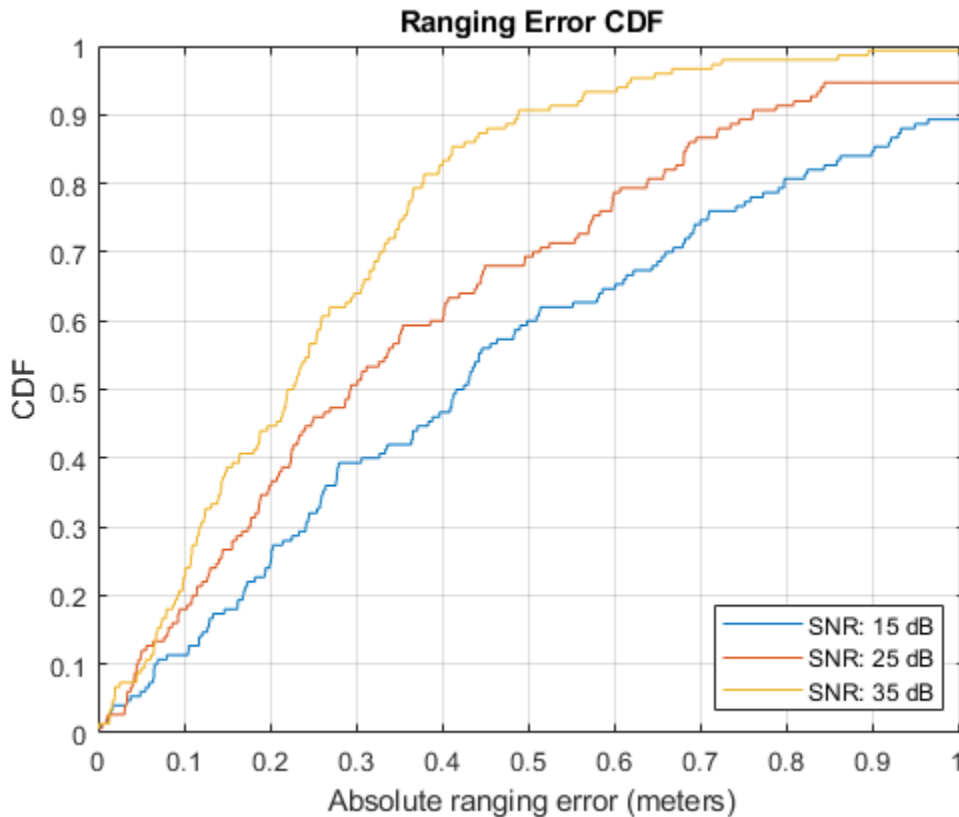

Trilateration

Trilaterate the location of the STA in two dimensions by using the distance estimates, then calculate the positioning root-mean-square error (RMSE) for each iteration by using the STA position estimate. Display the average RMSE and its CDF for each SNR point.

```
positionSTAEst = zeros(2,numIterations,numSNR);
RMSE = zeros(numIterations,numSNR);
for isnr = 1:numSNR
    for i = 1:numIterations
        positionSTAEst(:,i,isnr) = hePositionEstimate(squeeze(positionAP(:,:,i,isnr)),squeeze(dis
    end
    % Find the RMSE for each iteration, then take the mean of all RMSEs
    RMSE = reshape(sqrt(mean(((positionSTAEst-positionSTA).^2),1)),[numIterations numSNR]);
    posEr = mean(RMSE(:,isnr),'all','omitnan');
    disp(['At SNR = ',num2str(snrRange(isnr)),' dB, ', 'Average RMS Positioning error = ', num2st
end
```

```
At SNR = 15 dB, Average RMS Positioning error = 0.61713 meters.
At SNR = 25 dB, Average RMS Positioning error = 0.40481 meters.
At SNR = 35 dB, Average RMS Positioning error = 0.26968 meters.
```

```
hePlotErrorCDF(RMSE,snrRange)
xlabel('RMS positioning error (meters)')
title('Positioning Error CDF')
```
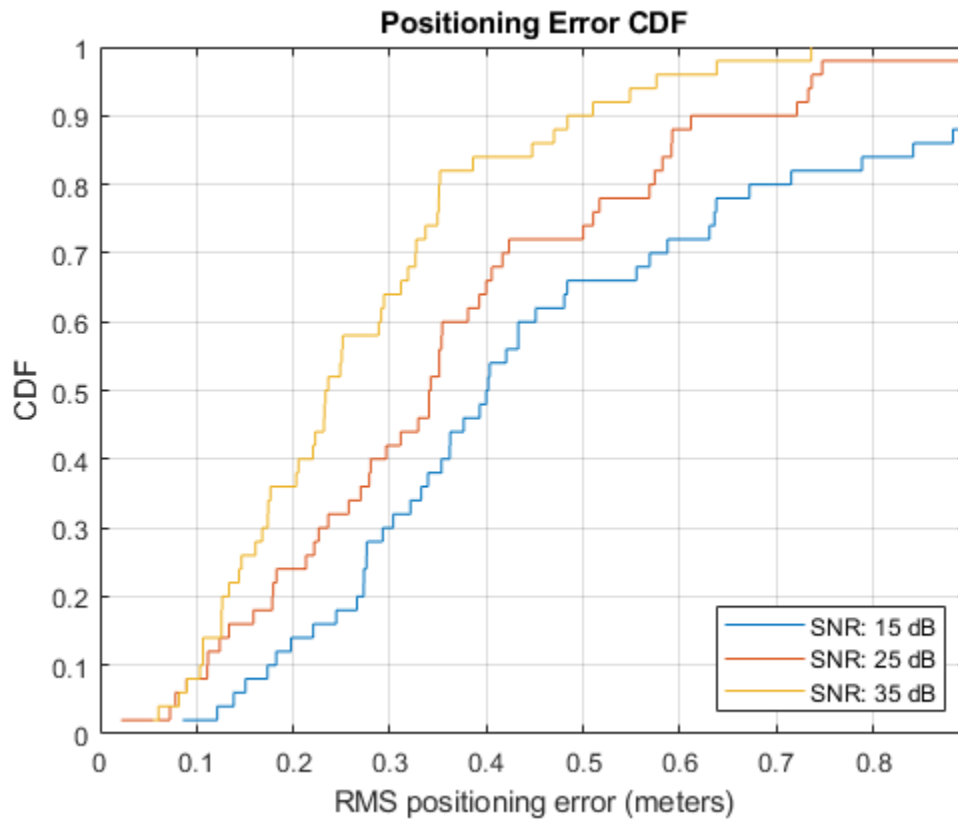

Plot the location estimate and the trilateration circles of the last iteration.

```
hePlotTrilaterationCircles(positionAP(:,:,numIterations,numSNR),positionSTAEst(:,numIterations,nu
```

Node positions at SNR 35 dB for iteration #50
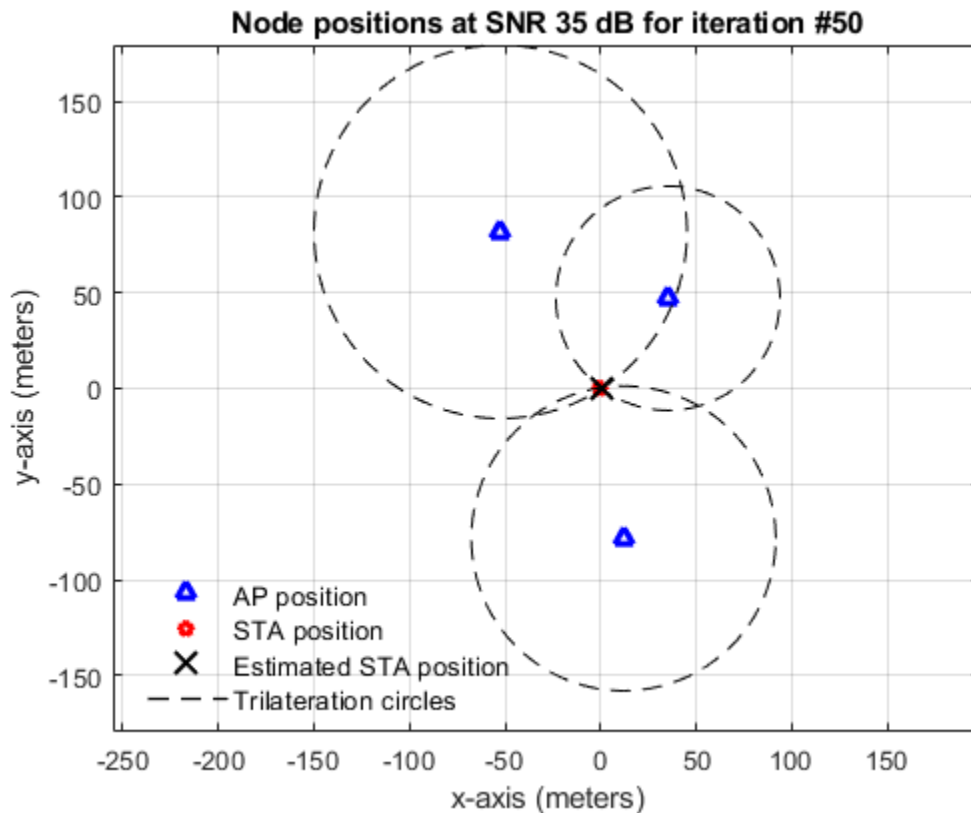
Conclusion

This example shows how to use a positioning algorithm with the IEEE® 802.11az™ standard. In particular, the example shows how to estimate the transmit-receive distance between a STA and AP by using MUSIC super-resolution, and how to locate a STA in two dimensions by using the ranging measurements from multiple STA-AP pairs. The example demonstrates the increase in performance of the positioning system at higher SNRs by computing the positioning estimate at multiple SNR points.

Further Exploration

Besides SNR, several important parameters impact positioning performance, such as HE-LTF repetitions, number of spatial streams, higher bandwidths, and channel delay profiles. This figure shows the impact of HE-LTF repetitions on ranging performance. To generate this figure, run a longer simulation with three randomly placed APs and 4000 iterations for a Model-B 2×2 MIMO channel of 20 MHz at 20 dB SNR. The figure shows that the ranging error decreases with the increase in HE-LTF repetitions. This decrease occurs because HE-LTF repetitions effectively reduce the noise in the CFR by averaging out multiple CFR estimates.
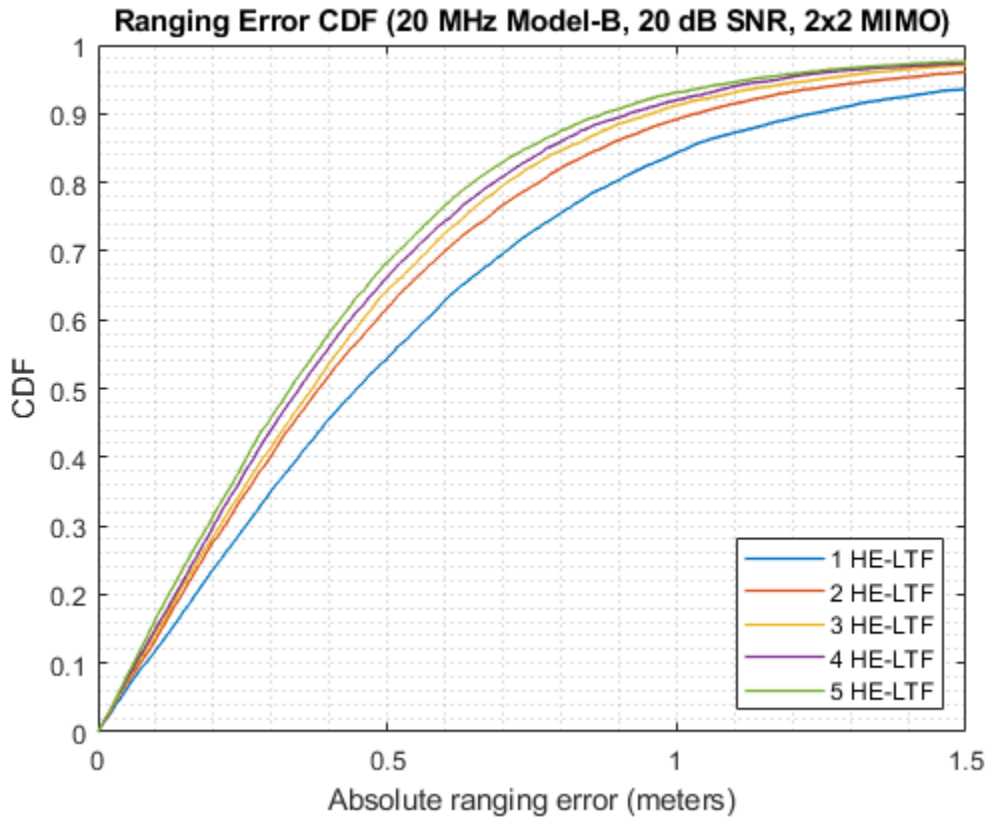
This figure shows the impact of different MIMO configurations on ranging performance. To generate this figure, run a longer simulation with three randomly placed APs and 4000 iterations, generating ranging packets with three HE-LTF repetitions at 20 MHz and specifying a Model-B channel at 20 dB SNR. The figure shows that the ranging error decreases with higher-order MIMO configurations. This decrease occurs because higher-order MIMO configurations produce more CFR snapshots from the different spatial streams available. More CFR snapshots give a better correlation matrix estimate which yields better ToA and distance estimates.

Ranging Error CDF (20 MHz Model-B, 20 dB SNR, 3 LTF repetitions)
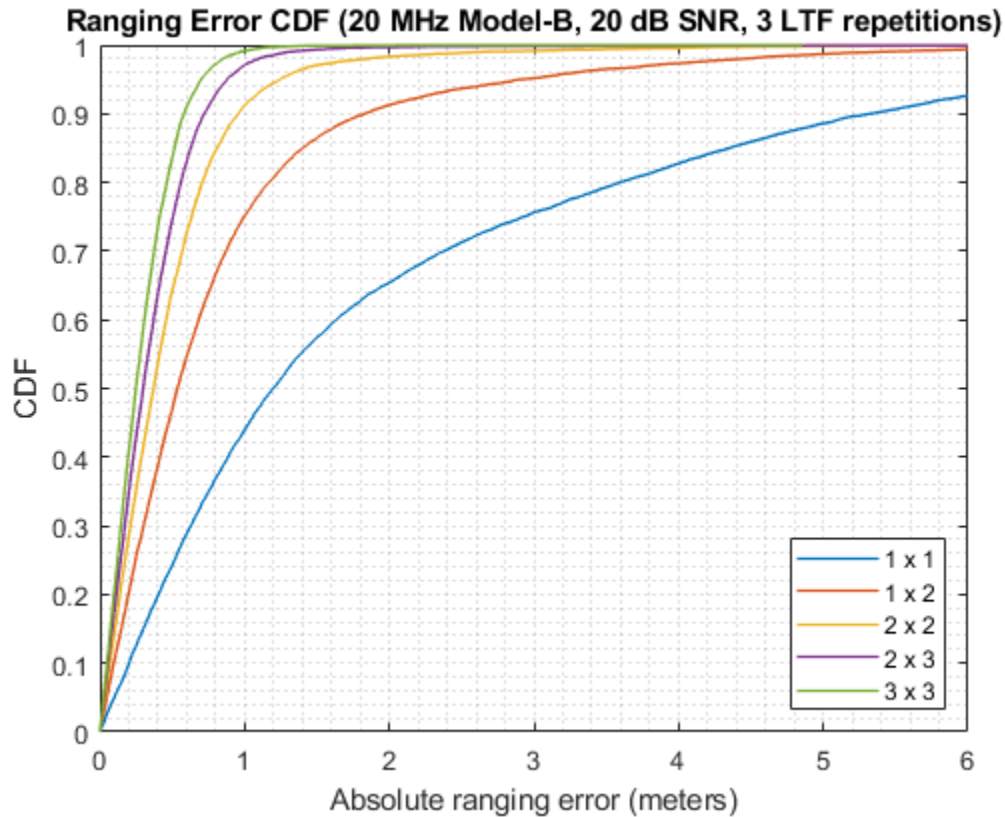
Related Examples

- "Three-Dimensional Indoor Positioning with 802.11az Fingerprinting and Deep Learning" on page 6-46. Trains a convolutional neural network (CNN) for localization and positioning by using Deep Learning Toolbox and IEEE 802.11az data generated with WLAN Toolbox.
- "802.11az Waveform Generation" on page 1-2. Parameterizes and generates IEEE 802.11az high-efficiency (HE) ranging null data packet (NDP) waveforms and highlights some of the key features of the standard.

References

1 IEEE P802.11az™/D2.0 Draft Standard for Information technology— Telecommunications and information exchange between systems Local and metropolitan area networks— Specific requirements - Amendment 3: Enhancements for positioning.

2 IEEE P802.11ax™/D4.1 Draft 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 6: Enhancements for High Efficiency WLAN.

3 Xinrong Li and K. Pahlavan, "Super-resolution TOA estimation with diversity for indoor geolocation," in IEEE Transactions on Wireless Communications, vol. 3, no. 1, pp. 224-234, Jan. 2004, doi: 10.1109/TWC.2003.819035.

802.11ad Packet Error Rate Simulation for Control PHY

This example shows how to measure the packet error rate of an IEEE® 802.11ad™ DMG control PHY AWGN link using an end-to-end simulation.

Introduction

In this example an end-to-end simulation is used to determine the packet error rate for an 802.11ad [1] control PHY link with an AWGN channel at a selection of SNR points. At each SNR point multiple packets are transmitted through a noisy channel, de-spread and the PSDUs recovered. The PSDUs are compared to those transmitted to determine the number of packet errors and hence the packet error rate. The receiver assumes perfect synchronization when recovering data bits from the received signal. The following diagram summarizes the processing for each packet.

This example also demonstrates how a `parfor` loop can be used instead of the `for` loop when simulating each SNR point to speed up a simulation. The `parfor` function, as part of the Parallel Computing Toolbox™, executes processing for each SNR in parallel to reduce the total simulation time.

Waveform Configuration

An 802.11ad DMG control PHY transmission is simulated in this example. The DMG format configuration object, `wlanDMGConfig`, contains the format-specific configuration of the transmission. The properties of the object contain the configuration of the transmitted packet. In this example the object is configured to generate a control PHY waveform. The MCS determines the PHY type used, therefore the MCS must be set to 0 to use the control PHY.

```
% Create a format configuration object
cfgDMG = wlanDMGConfig;
cfgDMG.MCS = 0; % MCS 0 represents Control PHY
cfgDMG.PSDULength = 256; % PSDULength in bytes
```

Spectral Filtering

Spectral filtering is used to reduce the out-of-band spectral emissions due to the spread spectrum characteristics of the transmitted waveform. In this example, the waveform is filtered through a raised cosine filter both at the transmitter and receiver using the `comm.RaisedCosineTransmitFilter` and `comm.RaisedCosineReceiveFilter` objects, respectively. To meet the spectral mask requirements, the raised cosine filter is truncated to the duration of eight symbols and the roll-off factor is set to 0.5.

```
% Define the pulse shaping filter characteristics
pulseShaping = true; % Enable pulse shaping
Nsym = 8;            % Filter span in symbol durations
alpha = 0.5;         % Roll-off factor
osps = 4;            % Output samples per symbol
```

```
% Transmit pulse shaping filter
txFilter = comm.RaisedCosineTransmitFilter;
txFilter.RolloffFactor = alpha;
txFilter.FilterSpanInSymbols = Nsym;
txFilter.OutputSamplesPerSymbol = osps;
txFilter.Shape = 'Normal';

% Receive pulse shaping filter
rxFilter = comm.RaisedCosineReceiveFilter;
rxFilter.RolloffFactor = alpha;
rxFilter.DecimationFactor = osps;
rxFilter.InputSamplesPerSymbol = osps;
rxFilter.FilterSpanInSymbols = Nsym;
rxFilter.Shape = 'Normal';
```

Simulation Parameters

For each SNR point in the vector `snrVec` a number of packets are generated, passed through an AWGN channel and demodulated to determine the packet error rate.

```
snrVec = -13.5:0.5:-10.5;
```

The number of packets tested at each SNR point is controlled by two parameters:

1   `maxNumErrors` is the maximum number of packet errors simulated at each SNR point. When the number of packet errors reaches this limit, the simulation at this SNR point is complete.

2   `maxNumPackets` is the maximum number of packets simulated at each SNR point and limits the length of the simulation if the packet error limit is not reached.

The numbers chosen in this example will lead to a very short simulation. For meaningful results we recommend increasing these numbers.

```
maxNumErrors = 10;   % The maximum number of packet errors at an SNR point
maxNumPackets = 100; % Maximum number of packets at an SNR point
```

Processing SNR Points

For each SNR point a number of packets are tested and the packet error rate calculated.

For each packet the following processing steps occur:

1   A PSDU is created and encoded to create a single packet waveform.

2   AWGN is added to the waveform. The `comm.AWGNChannel` object is configured to provide the correct SNR before despreading.

3   The packet is received with perfect synchronization.

4   The header and data fields are extracted from the received waveform and are processed together.

5   The recovered field is de-rotated by pi/2 and is de-spread.

6   The PSDU is recovered from the extracted field.

A `parfor` loop can be used to parallelize processing of the SNR points, therefore for each SNR point an AWGN channel is created and configured with the `comm.AWGNChannel` object. To enable the use of parallel computing for increased speed comment out the `for` statement and uncomment the `parfor` statement below.

```matlab
numSNR = numel(snrVec); % Number of SNR points
packetErrorRate = zeros(numSNR,1);
indices = wlanFieldIndices(cfgDMG);

if ~strcmp(phyType(cfgDMG),'Control')
    error('This example only supports DMG Control PHY simulation');
end

%parfor isnr = 1:numSNR % Use 'parfor' to speed up the simulation
for isnr = 1:numSNR      % Use 'for' to debug the simulation
    % Set random substream index per iteration to ensure that each
    % iteration uses a repeatable set of random numbers
    stream = RandStream('combRecursive','Seed',0);
    stream.Substream = isnr;
    RandStream.setGlobalStream(stream);

    % Create an instance of the AWGN channel per SNR point simulated
    awgnChannel = comm.AWGNChannel;
    awgnChannel.NoiseMethod = 'Signal to noise ratio (SNR)';
    awgnChannel.SNR = snrVec(isnr);

    % Noise power
    nVar = 10^(-snrVec(isnr)/10);
    numPacketErrors = 0;
    numPkt = 1; % Index of packet transmitted

    while numPacketErrors<=maxNumErrors && numPkt<=maxNumPackets
        % Generate a packet waveform
        psdu = randi([0 1],cfgDMG.PSDULength*8,1); % PSDULength in bytes
        tx = wlanWaveformGenerator(psdu,cfgDMG);

        % Transmitter filtering
        if pulseShaping
            % Append zero to compensate for filter group delay
            tx = txFilter([tx; zeros(Nsym,1)]);
            reset(txFilter);
        end

        % Add noise
        rx = awgnChannel(tx);

        % Receiver filtering
        if pulseShaping
            rx = rxFilter(rx);
            reset(rxFilter);
        end

        % Synchronize
        % The received signal is synchronized to the start of the packet by
        % compensating for a known delay due the spectral shaping filters
        if pulseShaping
            offset = Nsym;
        else
            offset = 0; %#ok<UNRCH>
        end

        % Process header and data field together
        rxHeaderDataField = rx(offset+(indices.DMGHeader(1):indices.DMGData(2)));
```

```
        % Apply pi/2 de-rotation and de-spread the received signal
        [rxSym,SF] = dmgControlDespread(rxHeaderDataField);

        % Recover the transmitted PSDU from DMG Data field. Scale the noise
        % power by the spreading factor
        dataDecode = wlanDMGDataBitRecover(rxSym,nVar/SF,cfgDMG);

        % Determine if any bits are in error, i.e. a packet error
        packetError = any(biterr(psdu,dataDecode));
        numPacketErrors = numPacketErrors + packetError;
        numPkt = numPkt+1;
    end

    % Calculate packet error rate (PER) at SNR point
    packetErrorRate(isnr) = numPacketErrors/(numPkt-1);
    disp(['SNR ' num2str(snrVec(isnr))...
          ' completed after ' num2str(numPkt-1) ' packets,'...
          ' PER: ' num2str(packetErrorRate(isnr))]);
end
```

Plot Packet Error Rate vs SNR Results

```
figure;
semilogy(snrVec,packetErrorRate,'-o');
grid on;
xlabel('SNR (dB)');
ylabel('PER');
legend('MCS 0');
title('PER for DMG Control-PHY, AWGN channel');
```
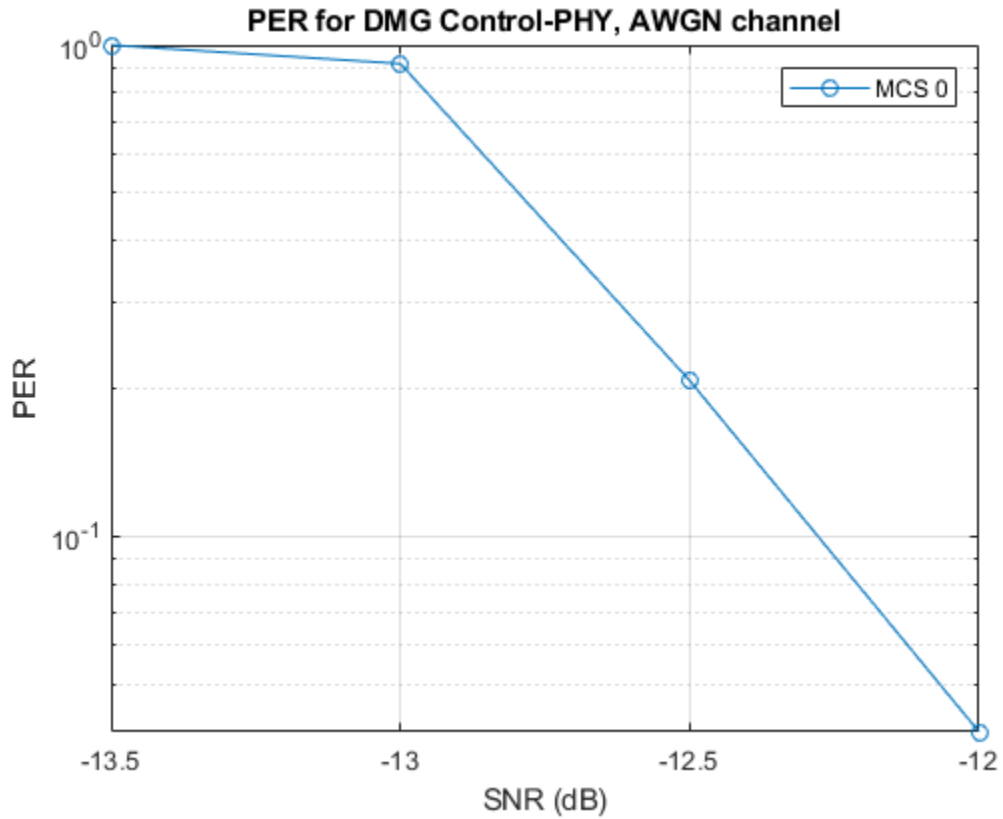
PER for DMG Control-PHY, AWGN channel

Further Exploration

The number of packets tested at each SNR point is controlled by two parameters: `maxNumErrors` and `maxNumPackets`. For meaningful results, it is recommended that these values should be larger than those presented in this example. Increasing the number of packets simulated allows the PER under different scenarios to be compared. As an example, the figure below was created by running the example for a `PSDULength` of 256 bytes, `maxNumErrors`:1000 and `maxNumPackets`: 100000.
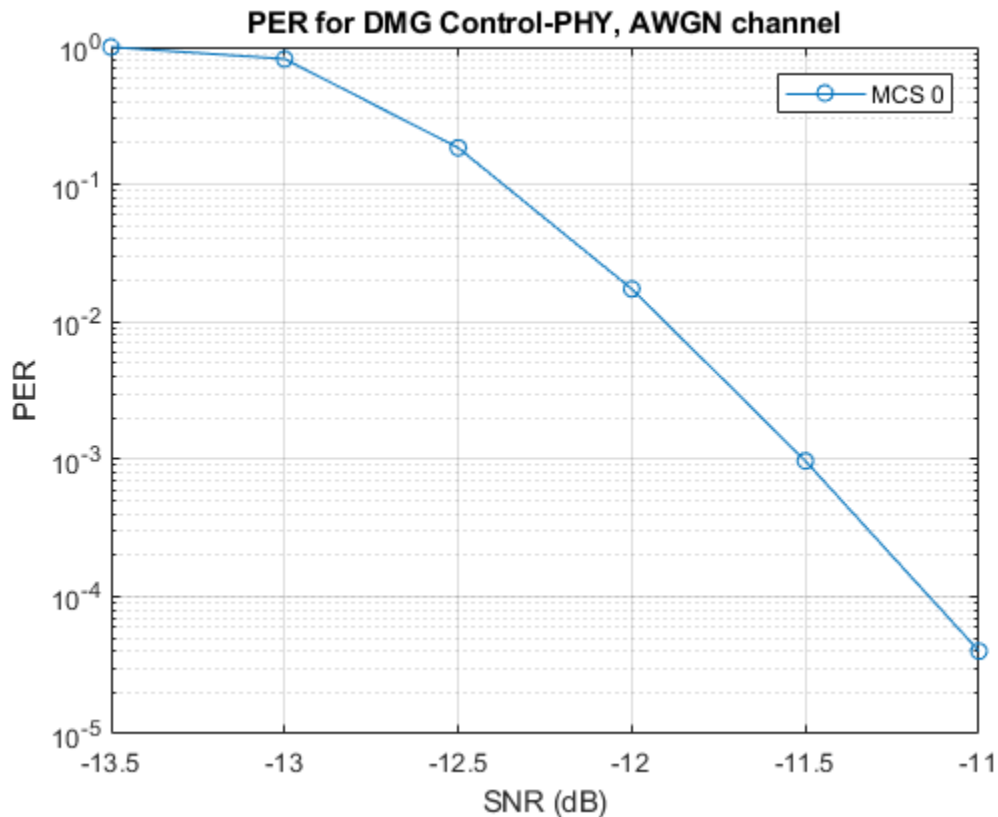
PER for DMG Control-PHY, AWGN channel

Selected Bibliography

1   IEEE Std 802.11ad™-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. Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band.

Local Functions

The following local function is used in this example:

* `dmgControlDespread`: De-spread the receive signal

```
function [y,SF] = dmgControlDespread(rx)
    SF = 32; % Spreading factor
    dataField = rx.*exp(-1i*pi/2*(0:size(rx,1)-1).'); % De-rotate symbols
    Ga = wlanGolaySequence(SF); % Generate Golay sequence
    y = (reshape(dataField,SF,length(dataField)/SF)'*Ga)/SF;
end
```

```
SNR -13.5 completed after 11 packets, PER: 1
SNR -13 completed after 12 packets, PER: 0.91667
SNR -12.5 completed after 53 packets, PER: 0.20755
SNR -12 completed after 100 packets, PER: 0.04
SNR -11.5 completed after 100 packets, PER: 0
```

```
SNR -11 completed after 100 packets, PER: 0
SNR -10.5 completed after 100 packets, PER: 0
```

802.11ad Packet Error Rate Simulation for OFDM PHY

This example shows how to measure the packet error rate of an IEEE® 802.11ad™ DMG OFDM PHY link using an end-to-end simulation with an AWGN channel.

Introduction

In this example an end-to-end simulation is used to determine the packet error rate for an 802.11ad DMG [1] OFDM link with an AWGN channel at a selection of SNR points for a defined modulation and coding scheme (MCS). For each SNR point, multiple packets are transmitted through a channel, demodulated and the PSDUs recovered. The PSDUs are compared to those transmitted to determine the number of packet errors and hence the packet error rate. Perfect time and frequency synchronization is assumed in this example. The following diagram summarizes the processing for each packet.
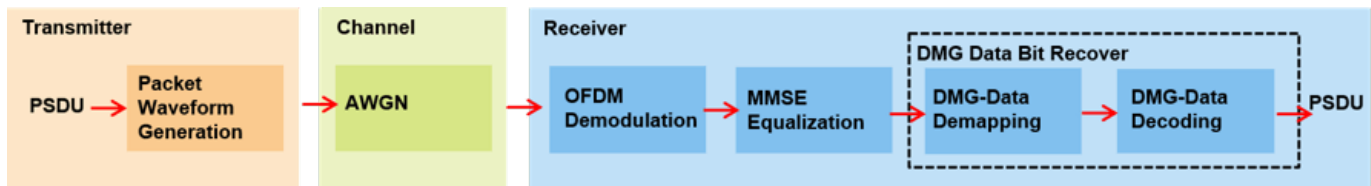
This example also demonstrates how speed up simulations by using a `parfor` loop instead of a `for` loop when simulating each SNR point. The `parfor` function, as part of the Parallel Computing Toolbox™, executes processing for each SNR in parallel to reduce the total simulation time.

Waveform Configuration

An 802.11ad DMG OFDM transmission is simulated in this example. The DMG format configuration object, `wlanDMGConfig`, contains the format specific configuration of the transmission. The properties of the object contain the configuration. In this example the object is configured for an OFDM transmission with MCS 21 and an 8192 byte PSDU. If `mcs` is specified as a vector, the simulation is performed for each MCS element. The MCS determines the PHY type used, therefore the MCS must be within the range 13-24 to use the OFDM PHY.

```
% Create a format configuration object for a DMG OFDM transmission
cfgDMG = wlanDMGConfig;
cfgDMG.PSDULength = 8192; % bytes

% For DMG OFDM PHY, the valid range of MCS is 13-24(inclusive)
mcs = 21; % OFDM PHY, 16QAM, rate 13/16
```

Simulation Parameters

For each SNR point a number of packets are generated, passed through a channel and demodulated to determine the packet error rate. The SNR points to simulate are selected from `snrRanges` based on the MCS to simulate. The SNR range for each MCS is selected in order to simulate the transition from all packets being decoded in error to all packets being decoded successfully as the SNR increases.

```
% SNR ranges to use for AWGN
snrRanges = {...
    -1:0.5:1.5, ...   % MCS 13
    0:0.5:2.5, ...    % MCS 14
    1.5:0.5:4, ...    % MCS 15
```

```
3:0.5:5.5, ...    % MCS 16
4.5:0.5:7, ...    % MCS 17
7.5:0.5:10, ...   % MCS 18
9:0.5:11.5, ...   % MCS 19
10.5:0.5:13, ...  % MCS 20
12:0.5:14.5, ...  % MCS 21
14.5:0.5:17, ...  % MCS 22
16.5:0.5:19, ...  % MCS 23
17.5:0.5:20, ...  % MCS 24
};
```

The number of packets tested at each SNR point is controlled by two parameters:

1 `maxNumErrors` is the maximum number of packet errors simulated at each SNR point. When the number of packet errors reaches this limit, the simulation at this SNR point is complete.

2 `maxNumPackets` is the maximum number of packets simulated at each SNR point and limits the length of the simulation if the packet error limit is not reached.

The numbers chosen in this example will lead to a very short simulation. For meaningful results we recommend increasing these numbers.

```
maxNumErrors = 10;    % The maximum number of packet errors at an SNR point
maxNumPackets = 100;  % Maximum number of packets at an SNR point
```

Set the remaining variables for the simulation.

```
% OFDM information
ofdmInfo = wlanDMGOFDMInfo();

% Indices of data and pilot occupied subcarriers
cfgDMG.MCS = mcs(1); % Set OFDM MCS to get subcarrier indices
Nsd = numel(ofdmInfo.DataIndices); % Number of data carrying subcarriers
```

Processing SNR Points

For each SNR point a number of packets are tested and the packet error rate calculated.

For each packet the following processing steps occur:

1 A PSDU is created and encoded to create a single packet waveform.

2 AWGN is added to the waveform to create the desired average SNR per subcarrier after OFDM demodulation. The `comm.AWGNChannel` object is configured to provide the correct SNR. The configuration accounts for the noise energy in unused subcarriers which are removed during OFDM demodulation.

3 The DMG-Data field is extracted from the perfectly synchronized received waveform and OFDM demodulated.

4 The pilots are discarded and the remaining OFDM demodulated symbols are equalized using the known channel response. As an AWGN link is used in this example, the complex channel gain is assumed to be one for each subcarrier.

5 The PSDU is recovered from the equalized data symbols.

A `parfor` loop can be used to parallelize processing of the SNR points, therefore for each SNR point an AWGN channel is created and configured with the `comm.AWGNChannel` object. To use parallel computing for increased speed, comment out the `for` statement and uncomment the `parfor` statement in this code.

```matlab
numSNR = numel(snrRanges{1}); % Number of SNR points
numMCS = numel(mcs);          % Number of MCS
packetErrorRate = zeros(numMCS,numSNR);

for imcs = 1:numMCS
    cfgDMG.MCS = mcs(imcs);
    if ~strcmp(phyType(cfgDMG),'OFDM')
        error('This example only supports DMG OFDM PHY simulation');
    end

    % Indices of fields within the packet
    fieldIndices = wlanFieldIndices(cfgDMG);

    % SNR points to simulate from MCS
    snr = snrRanges{cfgDMG.MCS-12};

    %parfor isnr = 1:numSNR % Use 'parfor' to speed up the simulation
    for isnr = 1:numSNR % Use 'for' to debug the simulation
        % Set random substream index per iteration to ensure that each
        % iteration uses a repeatable set of random numbers
        stream = RandStream('combRecursive','Seed',0);
        stream.Substream = isnr;
        RandStream.setGlobalStream(stream);

        % Create an instance of the AWGN channel per SNR point simulated
        awgnChannel = comm.AWGNChannel;
        awgnChannel.NoiseMethod = 'Signal to noise ratio (SNR)';
        awgnChannel.SignalPower = 1;
        % Account for noise energy in nulls so the SNR is defined per
        % active subcarrier
        awgnChannel.SNR = snr(isnr)-10*log10(ofdmInfo.FFTLength/ofdmInfo.NumTones);

        % Loop to simulate multiple packets
        numPacketErrors = 0;
        numPkt = 1; % Index of packet transmitted
        while numPacketErrors<=maxNumErrors && numPkt<=maxNumPackets
            % Generate a packet waveform
            txPSDU = randi([0 1],cfgDMG.PSDULength*8,1); % PSDULength in bytes
            tx = wlanWaveformGenerator(txPSDU,cfgDMG);

            % Pass the waveform through AWGN channel model
            rx = awgnChannel(tx);

            % Extract data field
            rxData = rx(fieldIndices.DMGData(1):fieldIndices.DMGData(2));

            % OFDM demodulate
            demodSym = wlanDMGOFDMDemodulate(rxData);
            dataSym = demodSym(ofdmInfo.DataIndices,:); % Discard pilots

            % Equalize
            chanSym = ones(Nsd,1); % Set channel gains to 1 as AWGN channel
            nVar = 10^(-snr(isnr)/10); % Noise variance
            [eqSym,csi] = helperSymbolEqualize(dataSym,chanSym,nVar);

            % Recover data
            rxPSDU = wlanDMGDataBitRecover(eqSym,nVar,csi,cfgDMG);
```

```
            % Determine if any bits are in error, i.e. a packet error
            packetError = any(biterr(txPSDU,rxPSDU));
            numPacketErrors = numPacketErrors+packetError;
            numPkt = numPkt+1;
        end

        % Calculate packet error rate (PER) at SNR point
        packetErrorRate(imcs,isnr) = numPacketErrors/(numPkt-1);
        disp(['MCS ' num2str(mcs(imcs)) ','...
              ' SNR ' num2str(snr(isnr)) ...
              ' completed after ' num2str(numPkt-1) ' packets,'...
              ' PER:' num2str(packetErrorRate(imcs,isnr))]);
    end
end

MCS 21, SNR 12 completed after 11 packets, PER:1
MCS 21, SNR 12.5 completed after 12 packets, PER:0.91667
MCS 21, SNR 13 completed after 71 packets, PER:0.15493
MCS 21, SNR 13.5 completed after 100 packets, PER:0.02
MCS 21, SNR 14 completed after 100 packets, PER:0
MCS 21, SNR 14.5 completed after 100 packets, PER:0
```

Plot Packet Error Rate vs SNR Results

```
markers = 'ox*sd^v><ph+';
color = 'bmcrgbrkymcr';
figure;
for imcs = 1:numMCS
    semilogy(snrRanges{mcs(imcs)-12},packetErrorRate(imcs,:).',['-' markers(imcs) color(imcs)]);
    hold on;
end
grid on;
xlabel('SNR (dB)');
ylabel('PER');
dataStr = arrayfun(@(x)sprintf('MCS %d',x),mcs,'UniformOutput',false);
legend(dataStr);
title('PER for DMG OFDM-PHY with AWGN channel');
```
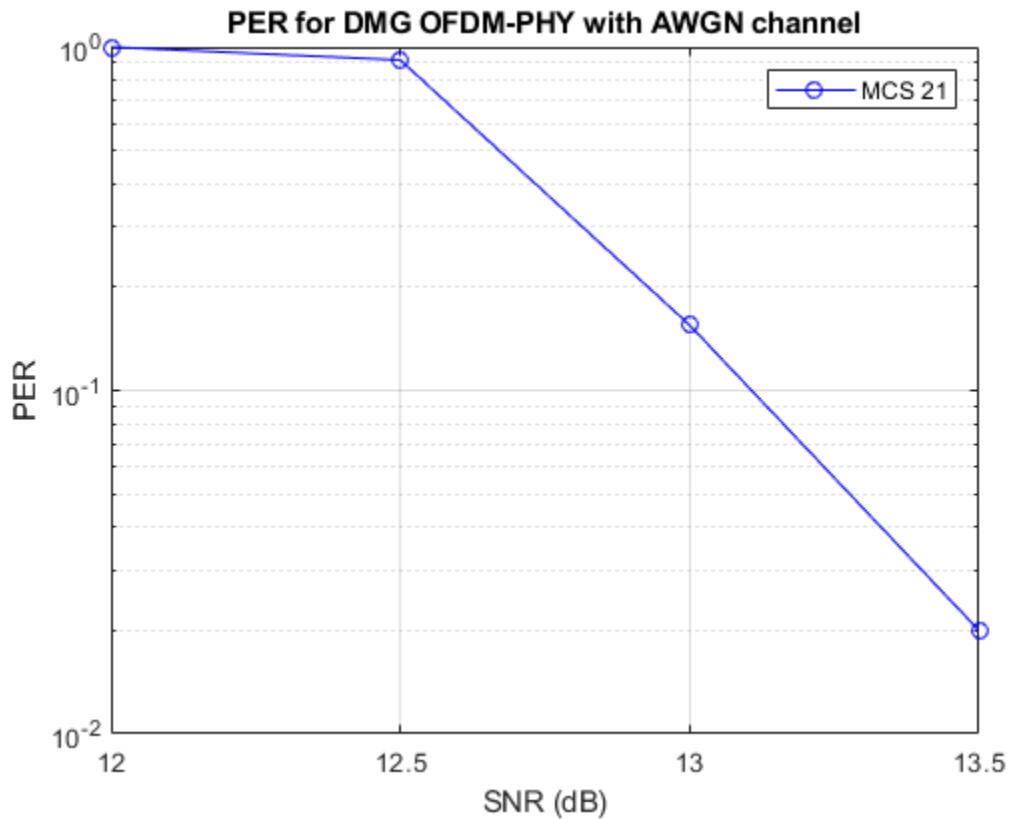
Further Exploration

The number of packets tested at each SNR point is controlled by two parameters: `maxNumErrors` and `maxNumPackets`. For meaningful results these values should be larger than those presented in this example. Increasing the number of packets simulated allows the PER under different scenarios to be compared. As an example, the figure below was created by running the example for longer with `maxNumErrors = 1e3` and `maxNumPackets = 1e4`, for `mcs = 13:24`.

PER for DMG OFDM-PHY with AWGN channel
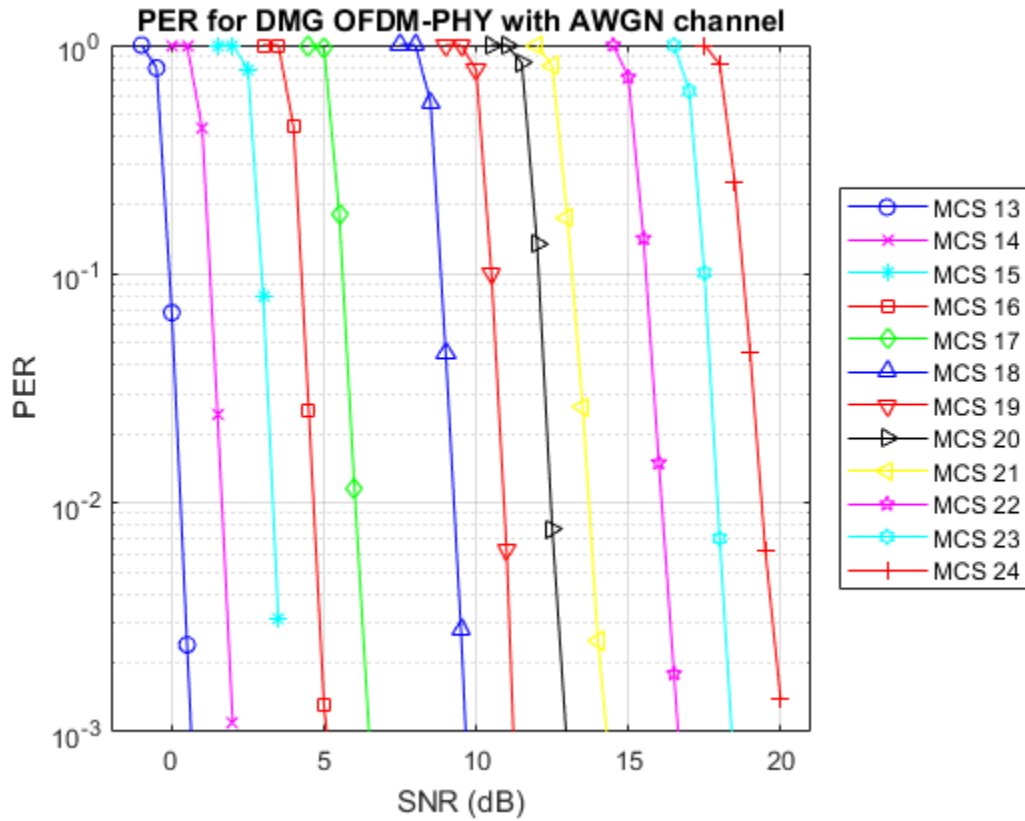
Appendix

This example uses the following helper function:

- helperSymbolEqualize.m

Selected Bibliography

1 IEEE Std 802.11ad™-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. Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band.

802.11ad Single Carrier Link with RF Beamforming in Simulink

This example shows how to model an IEEE® 802.11ad™ single carrier link in Simulink® which includes a phased array antenna with RF beamforming. This example requires the following products:

- WLAN Toolbox™ for baseband transmitter and receiver
- Phased Array System Toolbox™ for receive antenna array
- RF Blockset™ for RF receiver

Introduction

This model simulates an 802.11ad single carrier (SC) [1] link with RF beamforming. Multiple packets are transmitted through free space, then RF beamformed, demodulated and the PLCP service data units (PSDU) are recovered. The PSDUs are compared with those transmitted to determine the packet error rate. The receiver performs packet detection, timing synchronization, carrier frequency offset correction and unique word based phase tracking.

The MATLAB function block allows Simulink models to use MATLAB® functions. In this example, an 802.11ad SC link modeled in Simulink uses WLAN Toolbox functions called using MATLAB function blocks. For an 802.11ad baseband simulation in MATLAB, see the example "802.11ad Packet Error Rate Single Carrier PHY Simulation with TGay Channel" on page 5-2.
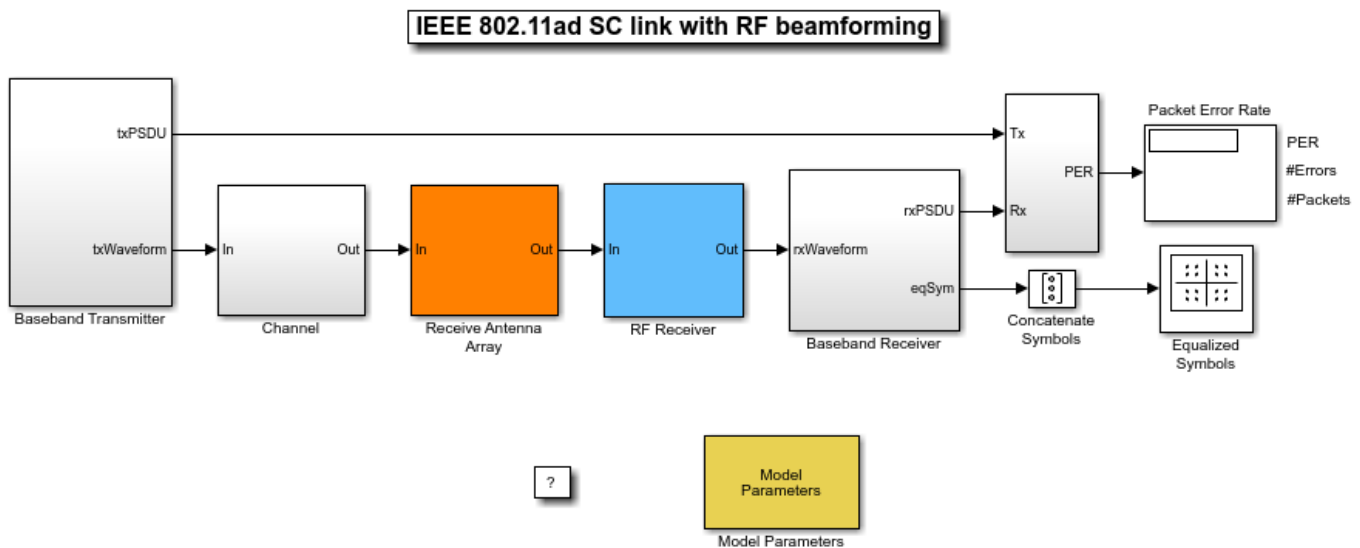
Copyright 2018-2020 The MathWorks, Inc.
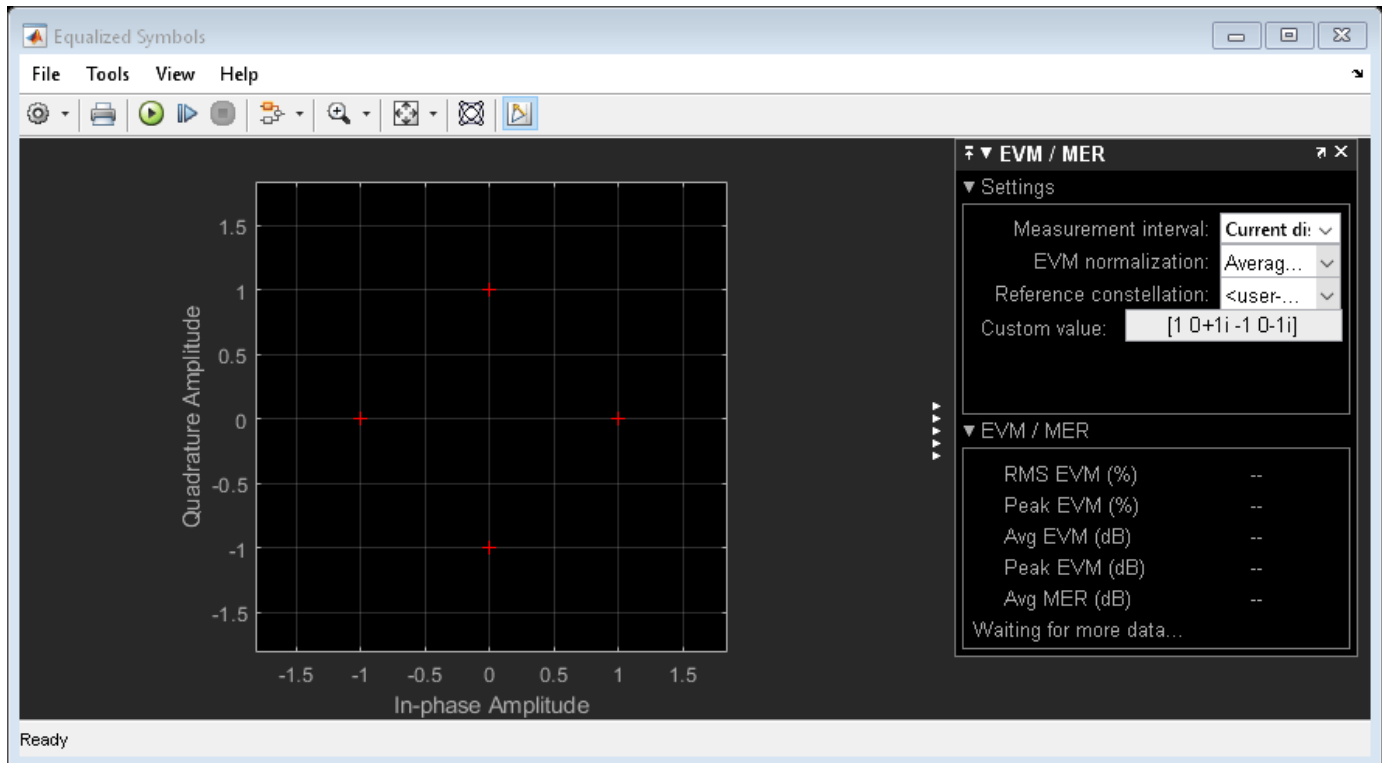
System Architecture

The system consists of:

- A baseband transmitter which generates a random PSDU and an 802.11ad SC packet.
- A free space channel.
- A receive antenna array which supports up to 16 elements. This module allows control of the array geometry, number of elements in an array, operating frequency, and receiver direction.
- A 16 channel RF receiver module to process the RF signals. This receiver module includes low noise amplifiers, phase shifters, Wilkinson 16:1 combiner, and a down-converter. This module allows control of the beamforming direction used to calculate the corresponding phase shifts.
- A baseband receiver which recovers the transmitted PSDU by performing packet detection, time and frequency synchronization, channel estimation, PSDU demodulation, and decoding.

The system diagnostics includes the display of equalized constellation and the obtained packet error rate.

The following sections describe the transmitter and receiver in more detail.

Baseband Transmitter

The baseband transmitter block creates a random PSDU and encodes the bits to create a single packet waveform based on the MCS and PSDU length values in the Model Parameters block. The packet generator block uses the function `wlanWaveformGenerator` to encode a packet.
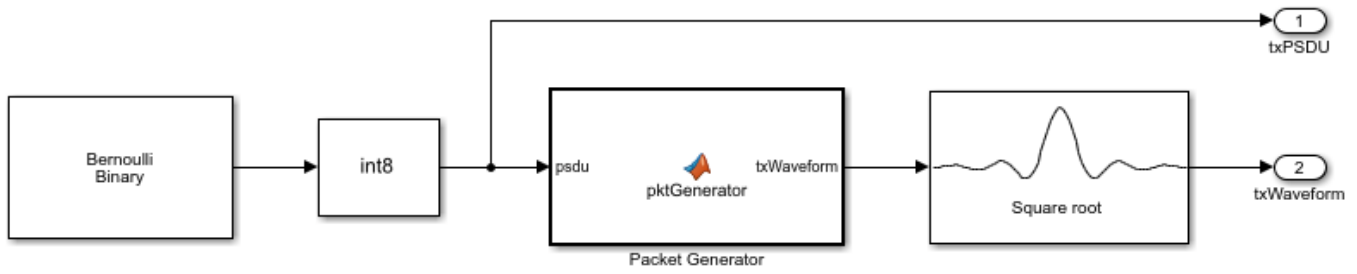
RF Receiver

The RF receiver consists of amplifiers, phase shifters, Wilkinson 16:1 combiner and is implemented in superheterodyne fashion.

RF RECEIVE MODULE FOR A 16 CHANNEL PHASED ARRAY with Ideal 16:1 Combiner
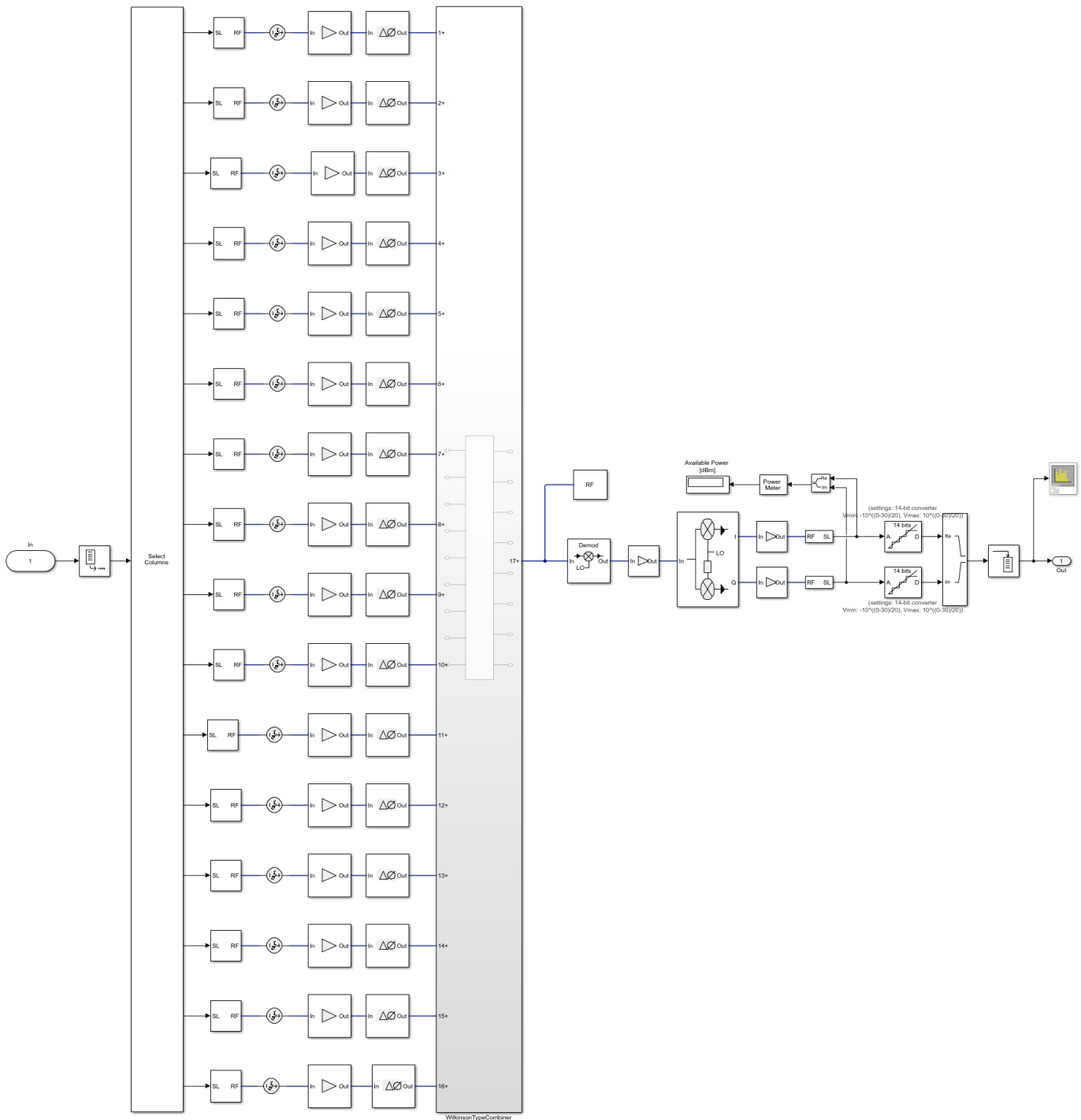
The phase shift applied to each element is calculated based on the beamforming direction. This is provided by the user and indicates the direction of the main beam. The receiver maximizes the SNR when the receiver's main beam points to the transmitter. Transmitter is omnidirectional and the

receiver direction (az,el) indicates the direction of incident signal. The scenario where the receiver direction and the beamforming direction are different is shown. In this case, there will be a reduction in the received signal power leading to high packet error rate (PER) and error vector magnitude (EVM). The results section shows these values.

RX Phased Array
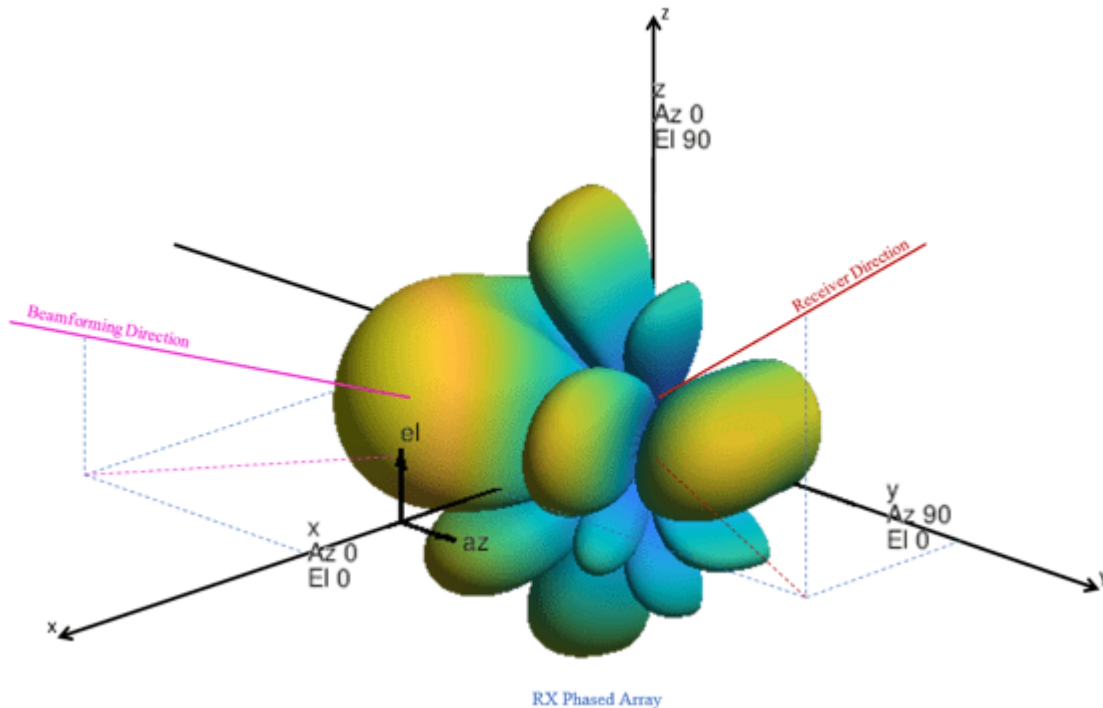
Baseband Receiver

The baseband receiver has two components: packet detection and packet recovery.
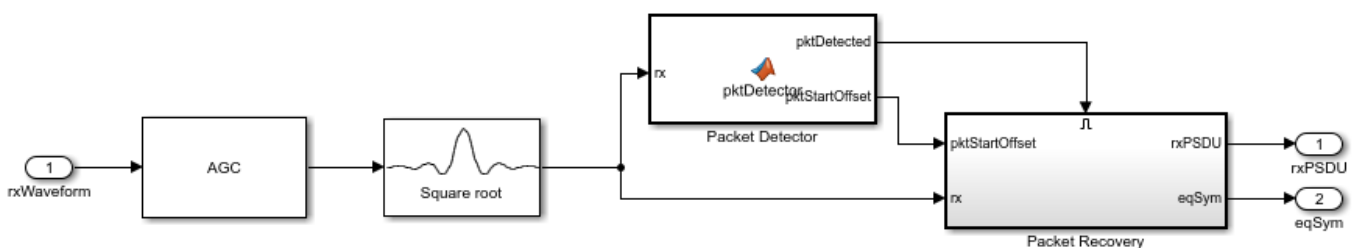
If a packet is detected, the packet recovery subsystem is enabled to process the detected packet.

The packet recovery subsystem processing consists of the following steps:

1   Frequency offset estimation and correction.
2   Symbol timing and channel frequency response estimation.
3   Noise power estimation.
4   Synchronization error checking. This determines whether the packet can be decoded or not.
5   Packet decoding.
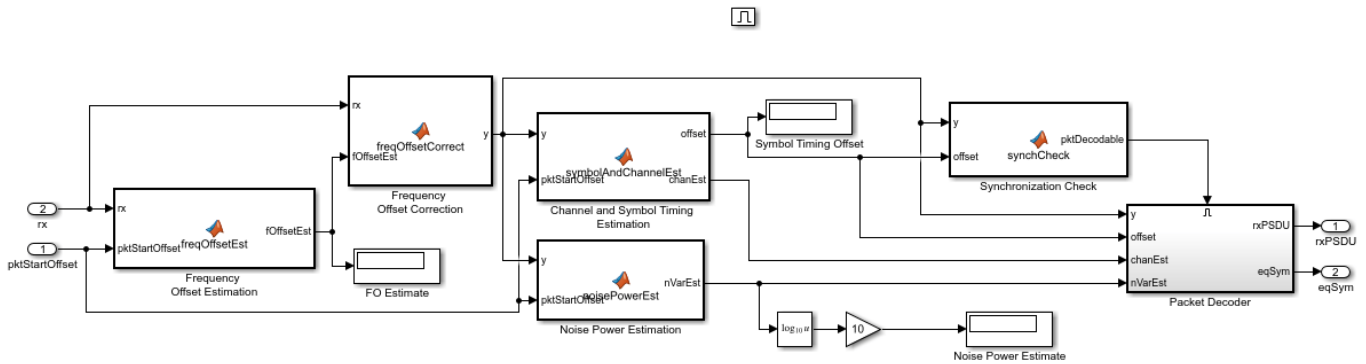
In the packet decoder subsystem, the SC data field is extracted from the synchronized received waveform. Then, the PSDU is recovered using the extracted field, channel, and noise power estimates.
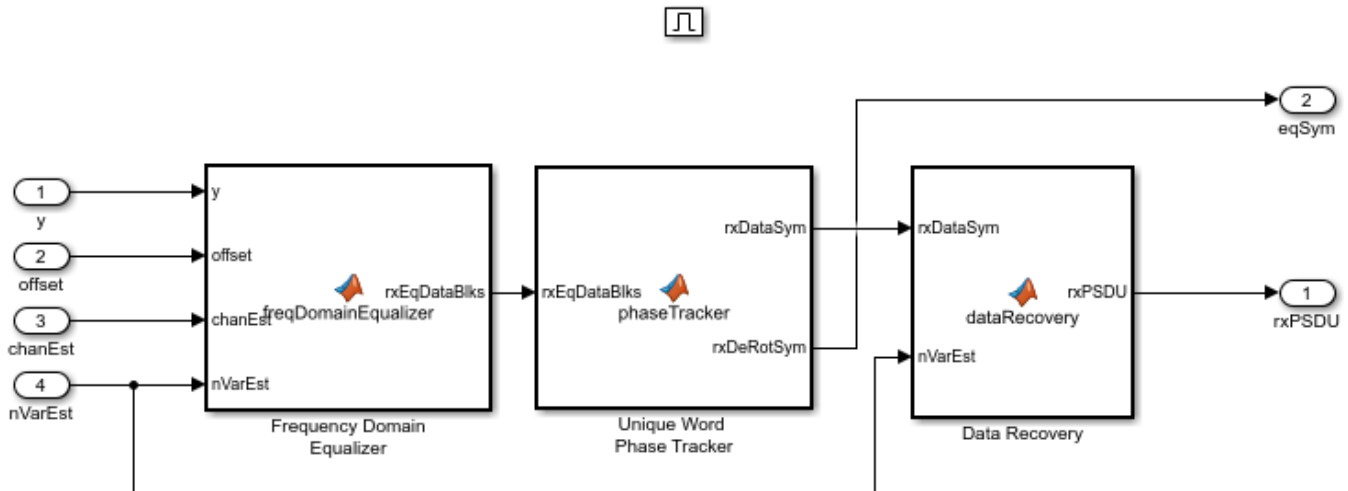
Results

Running the simulation displays the packet error rate. The model updates the PER after processing each packet. The model also displays the equalized symbol constellation along with the EVM measurement. Note that for statistically valid results, long simulation times are required.

By default, the main beam of the receive antenna array points towards the direction: azimuth = 0 deg. and elevation = 0 deg.
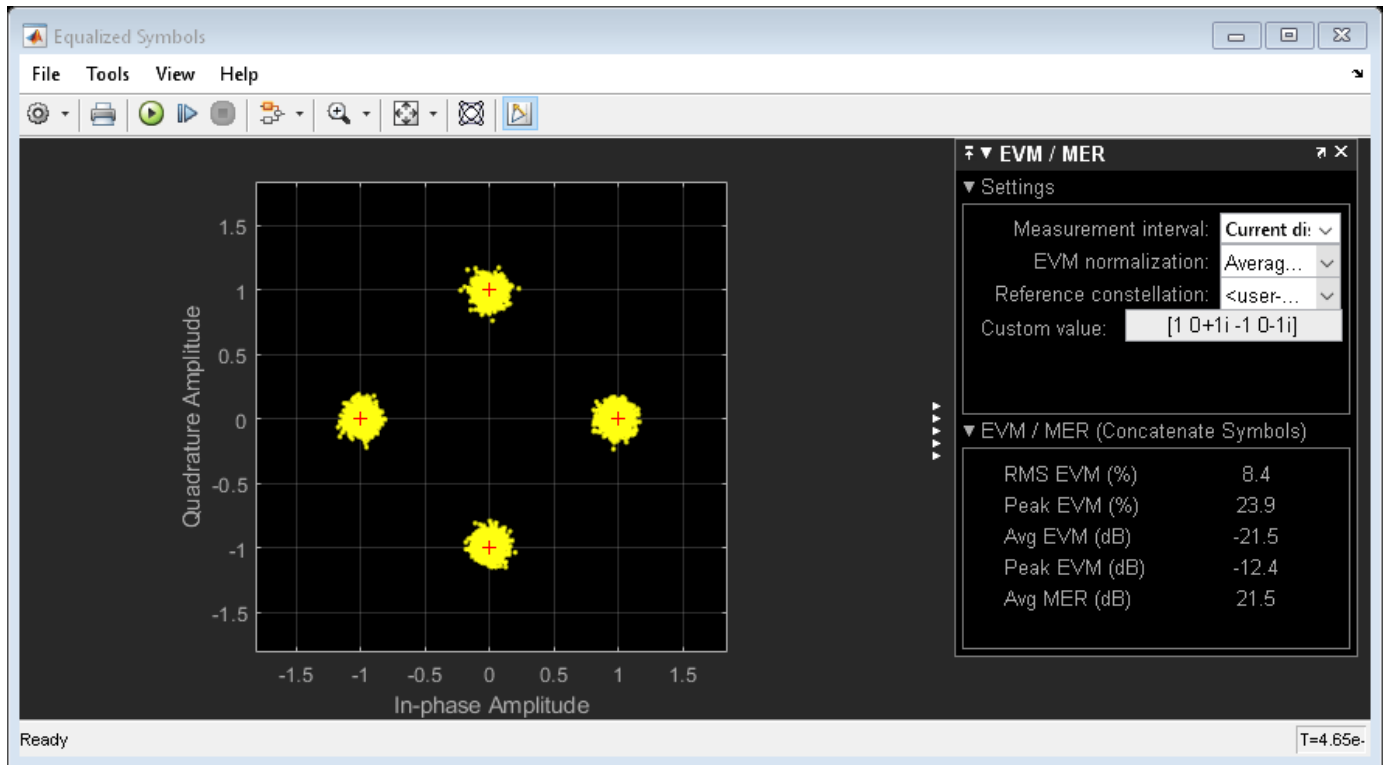
If you change the `Receiver direction` value in the receive antenna array towards a proximity null in the array radiation, the EVM increases and the packets cannot be successfully decoded.
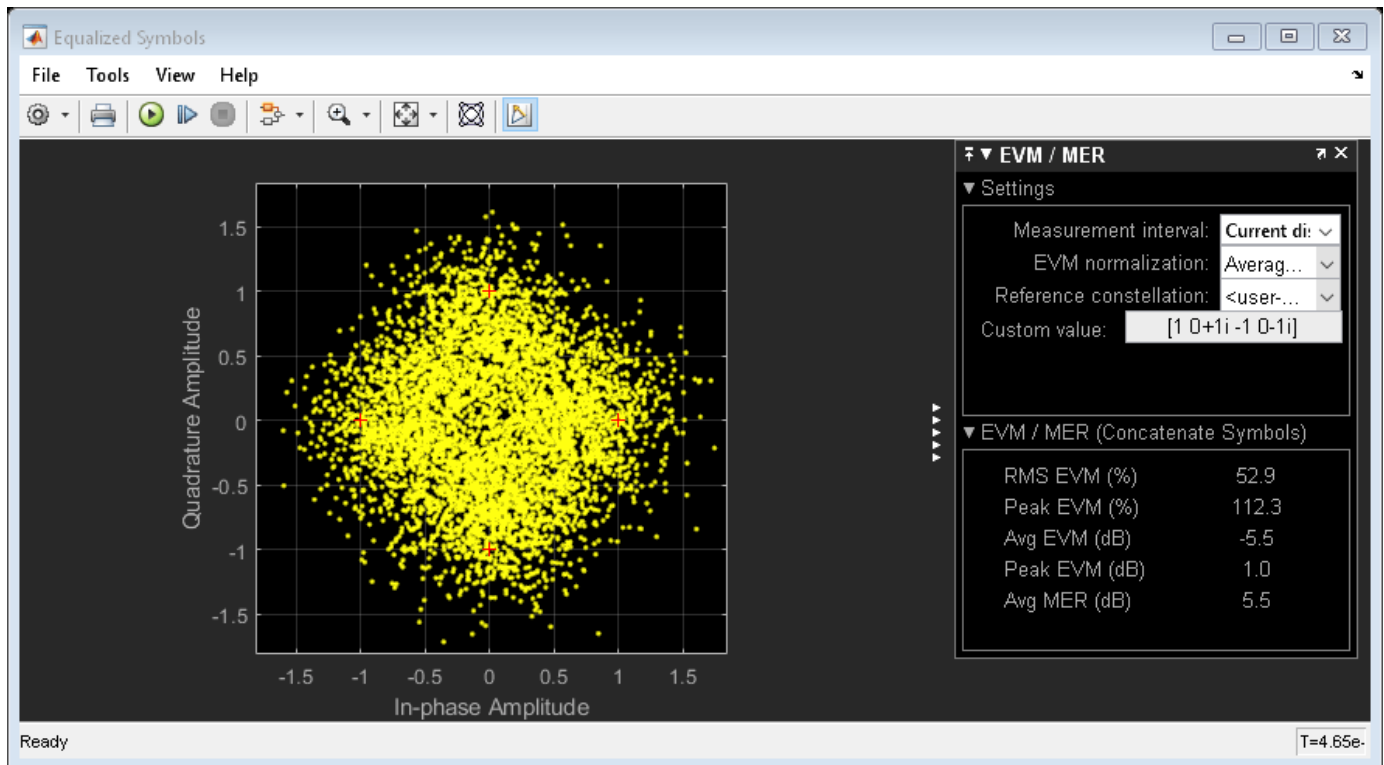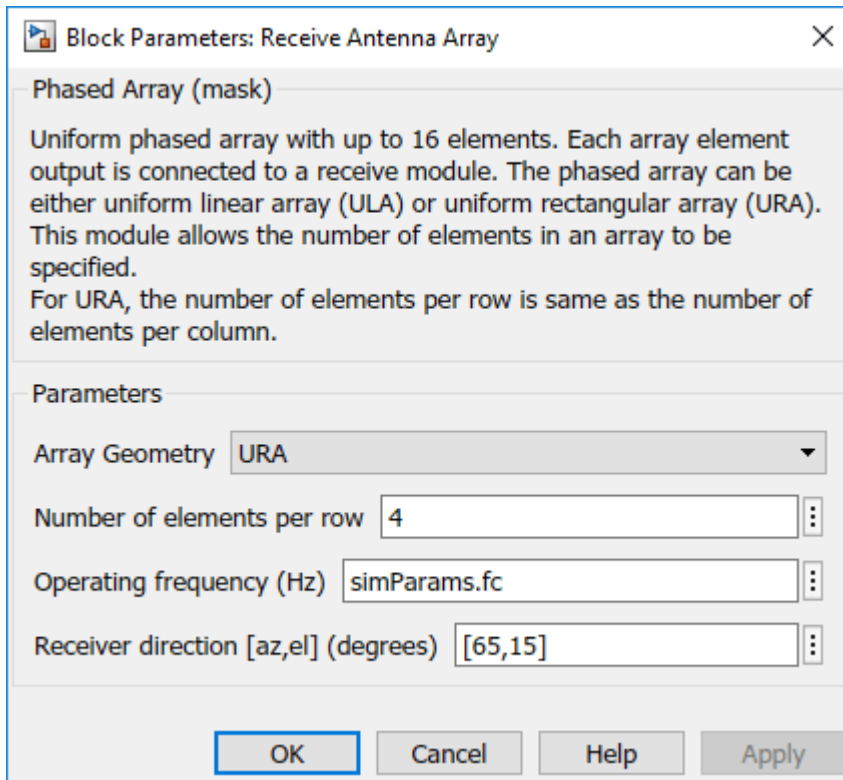
If you change the `Beamforming direction` value in the RF receiver such that the main beam points towards the transmitter, the EVM improves and packets are successfully decoded.
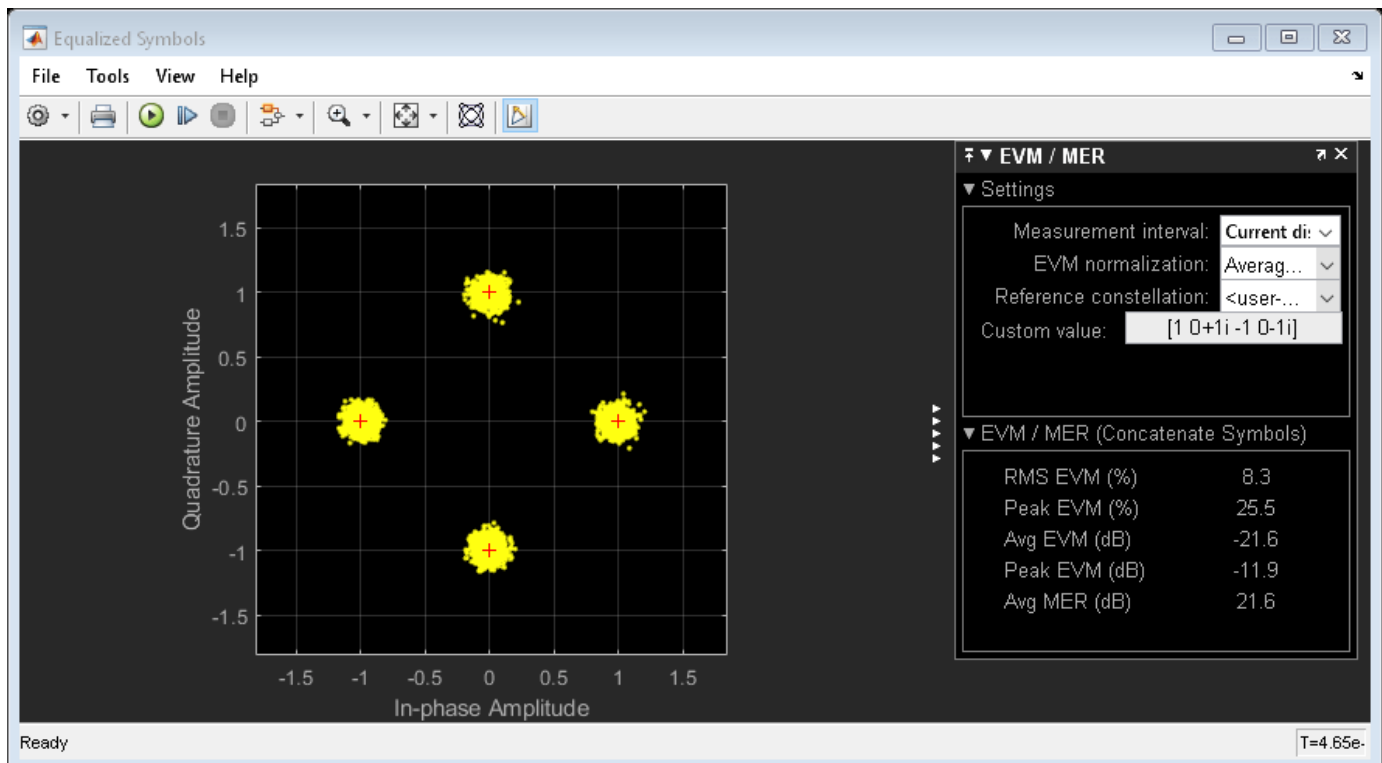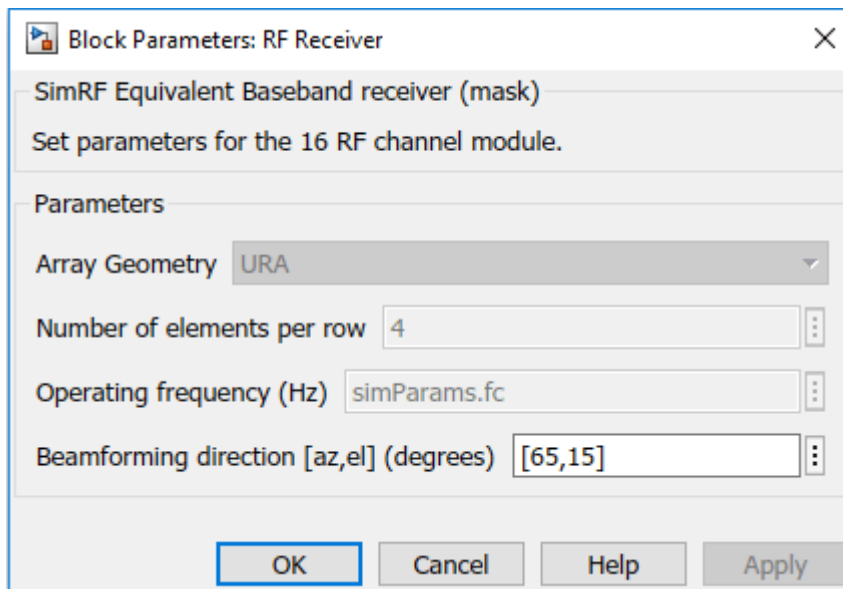
Exploring the Example

- Try changing the signal to noise ratio (SNR) value in the Model Parameters block. Increasing SNR leads to lower packet error rates and improved EVM of equalized symbols constellation. The SNR

specified is the signal to noise ratio at the input to the ADC, if a single receive chain is used. The SNR accounts for free space path loss, thermal noise and the noise figure of RF components.

- You can change the array geometry and the number of elements in an array present in the receive antenna array block. Increasing the number of antenna elements improves the EVM. The diversity gain due to receiver antenna array can be observed in the equalized symbols constellation.

Appendix

This example uses the following helper functions:

- dmgCFOEstimate.m
- dmgPacketDetect.m
- dmgSingleCarrierFDE.m
- dmgSTFNoiseEstimate.m
- dmgTimingAndChannelEstimate.m
- dmgUniqueWordPhaseTracking.m
- helperFrequencyOffset.m

Selected Bibliography

1 IEEE Std 802.11ad™-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. Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band.

802.11p Packet Error Rate Simulation for a Vehicular Channel

This example shows how to measure the packet error rate (PER) of an IEEE® 802.11p™ link using an end-to-end simulation with a Vehicle-to-Vehicle (V2V) fading channel and additive white Gaussian noise. The PER performance of a receiver with and without channel tracking is compared. In a vehicular environment (high Doppler), a receiver with channel tracking performs better.

Introduction

IEEE 802.11p [1] is an approved amendment to the IEEE 802.11™ standard to enable support for wireless access in vehicular environments (WAVE). Using the half-clocked mode with a 10 MHz channel bandwidth, it operates in 5.85-5.925 GHz bands to support applications for Intelligent Transportation Systems (ITS) [2].

In this example, an end-to-end simulation is used to determine the packet error rate for an 802.11p [1] link with a fading channel at a selection of SNR points with and without channel tracking. For each SNR point, multiple packets are transmitted through a V2V channel, demodulated and the PSDUs are recovered. The PSDUs are compared to those transmitted to determine the number of packet errors. For each packet, packet detection, timing synchronization, carrier frequency offset correction and phase tracking are performed at the receiver. For channel tracking, decision directed channel estimation [3] is used to compensate for the high Doppler spread. The figure below shows the processing chain with channel tracking.
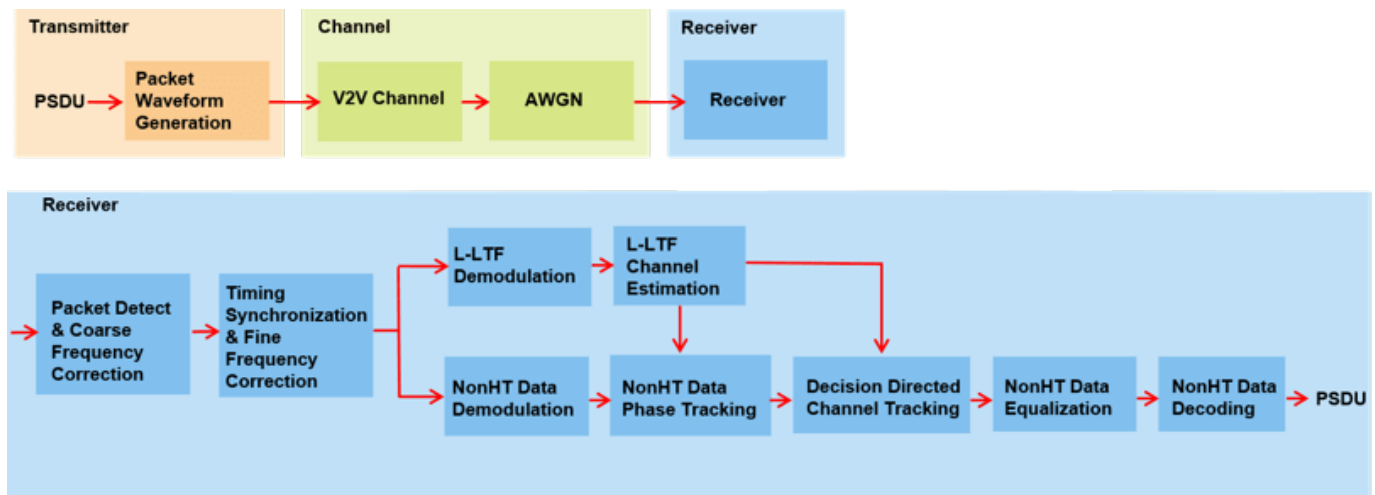
Waveform Configuration

An 802.11p non-HT format transmission is simulated in this example. A non-HT format configuration object contains the format specific configuration of the transmission. This object is created using the `wlanNonHTConfig` function. In this example, the object is configured for a 10 MHz channel bandwidth and QPSK rate 1/2 (MCS 2) operation.

```
% Link parameters
mcs = 2;        % QPSK rate 1/2
psduLen = 500; % PSDU length in bytes

% Create a format configuration object for an 802.11p transmission
cfgNHT = wlanNonHTConfig;
cfgNHT.ChannelBandwidth = 'CBW10';
```

```
cfgNHT.PSDULength = psduLen;
cfgNHT.MCS = mcs;
```

Channel Configuration

The V2V radio channel model defines five scenarios to represent fading conditions within a vehicular environment. In this example, 'Urban NLOS' [4] scenario is used. This corresponds to a scenario with two vehicles crossing each other at an urban blind intersection with building and fences present on the corners.

```
% Create and configure the channel
fs = wlanSampleRate(cfgNHT); % Baseband sampling rate for 10 MHz

chan = V2VChannel;
chan.SampleRate = fs;
chan.DelayProfile = 'Urban NLOS';
```

Simulation Parameters

For each SNR (dB) point in the vector `snr` a number of packets are generated, passed through a channel and demodulated to determine the packet error rate.

```
snr = 15:5:30;
```

The number of packets tested at each SNR point is controlled by two parameters:

1   `maxNumErrors` is the maximum number of packet errors simulated at each SNR point. When the number of packet errors reaches this limit, the simulation at this SNR point is complete.

2   `maxNumPackets` is the maximum number of packets simulated at each SNR point. It limits the length of the simulation if the packet error limit is not reached.

The numbers chosen in this example lead to a short simulation. For statistical meaningful results these numbers should be increased.

```
maxNumErrors = 20;   % The maximum number of packet errors at an SNR point
maxNumPackets = 200; % Maximum number of packets at an SNR point

% Set random stream for repeatability of results
s = rng(98);
```

Processing SNR Points

For each SNR point, a number of packets are tested and the packet error rate is calculated. For each packet the following processing steps occur:

1   A PSDU is created and encoded to create a single packet waveform.

2   The waveform is passed through the channel. Different channel realizations are used for each transmitted packet.

3   AWGN is added to the received waveform to create the desired average SNR per subcarrier after OFDM demodulation. comm.AWGNChannel is configured to provide the correct SNR. The configuration accounts for normalization within the channel by the number of receive antennas, and the noise energy in unused subcarriers which are removed during OFDM demodulation.

4   The per-packet processing includes packet detection, coarse carrier frequency offset estimation and correction, symbol timing and fine carrier frequency offset estimation and correction.

5 The L-LTF is extracted from the synchronized received waveform. The L-LTF is OFDM demodulated and initial channel estimates are obtained.

6 Channel tracking can be enabled using the switch `enableChanTracking`. If enabled, the channel estimates obtained from L-LTF are updated per symbol using decision directed channel tracking as presented in J. A. Fernandez et al in [3]. If disabled, the initial channel estimates from L-LTF are used for the entire packet duration.

7 The non-HT Data field is extracted from the synchronized received waveform. The PSDU is recovered using the extracted data field and the channel estimates and noise power estimate.

```matlab
% Set up a figure for visualizing PER results
h = figure;
grid on;
hold on;
ax = gca;
ax.YScale = 'log';
xlim([snr(1), snr(end)]);
ylim([1e-3 1]);
xlabel('SNR (dB)');
ylabel('PER');
h.NumberTitle = 'off';
h.Name = '802.11p ';
title(['MCS ' num2str(mcs) ', V2V channel - ' chan.DelayProfile ' profile']);

% Simulation loop for 802.11p link
S = numel(snr);
per_LS = zeros(S,1);
per_STA = per_LS;
for i = 1:S
    enableChanTracking = true;
    % 802.11p link with channel tracking
    per_STA(i) = v2vPERSimulator(cfgNHT, chan, snr(i), ...
        maxNumErrors, maxNumPackets, enableChanTracking);

    enableChanTracking = false;
    % 802.11p link without channel tracking
    per_LS(i) = v2vPERSimulator(cfgNHT, chan, snr(i), ...
        maxNumErrors, maxNumPackets, enableChanTracking);

    semilogy(snr, per_STA, 'bd-');
    semilogy(snr, per_LS, 'ro--');
    legend('with Channel Tracking','without Channel Tracking')
    drawnow;
end

axis([10 35 1e-3 1])
hold off;

% Restore default stream
rng(s);

SNR 15 dB with channel tracking completed after 51 packets, PER: 0.41176
SNR 15 dB without channel tracking completed after 59 packets, PER: 0.35593
SNR 20 dB with channel tracking completed after 201 packets, PER: 0.069652
SNR 20 dB without channel tracking completed after 109 packets, PER: 0.19266
SNR 25 dB with channel tracking completed after 201 packets, PER: 0.0199
SNR 25 dB without channel tracking completed after 182 packets, PER: 0.11538
```

```
SNR 30 dB with channel tracking completed after 201 packets, PER: 0.0099502
SNR 30 dB without channel tracking completed after 201 packets, PER: 0.094527
```

MCS 2, V2V channel - Urban NLOS profile
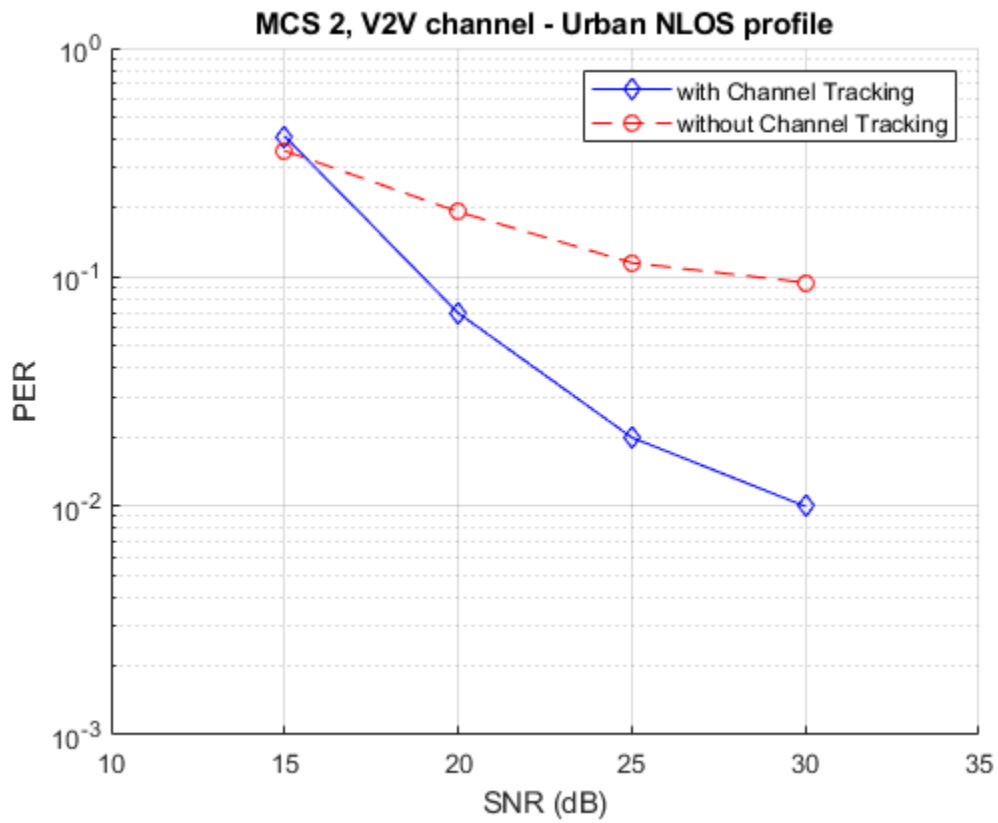
For meaningful results `maxNumErrors`, `maxNumPackets` should be increased. The below plot provides results for `maxNumErrors: 1000` and `maxNumPackets: 10000`.
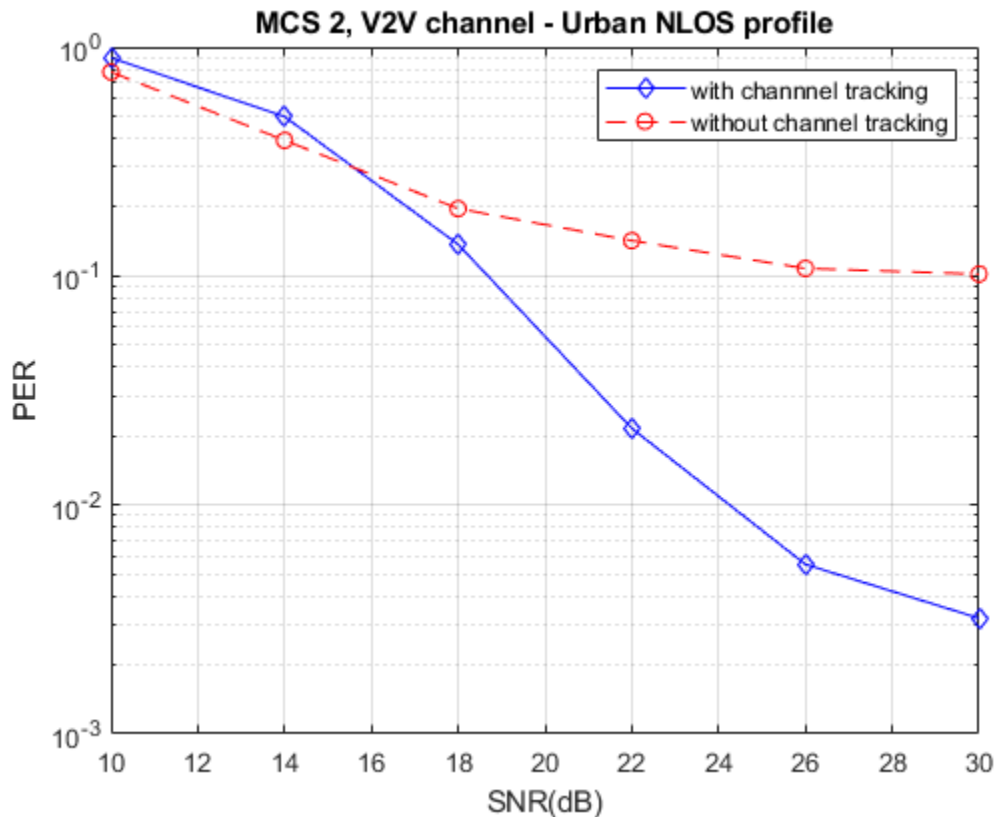
Further Exploration

Try changing the channel delay profile, the length of the packet or the data rate (`mcs` values) and observe the performance of channel tracking. For some configurations channel tracking provides little performance improvement. For a small number of OFDM symbols (small PSDU length or high MCS), temporal averaging performed during decision directed channel tracking may not be effective. The characteristics of the channel may also limit the performance for higher order modulation schemes (`mcs > 5`).

Appendix

This example uses the following helper functions and objects:

*   v2vPERSimulator.m
*   V2VChannel.m

Selected Bibliography

1 IEEE Std 802.11p-2010: 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 6: Wireless Access in Vehicular Environments, IEEE, New York, NY, USA, 2010.

2 ETSI, https://www.etsi.org/technologies-clusters/technologies/automotive-intelligent-transport.

3 J. A. Fernandez, D. D. Stancil and F. Bai, "Dynamic channel equalization for IEEE 802.11p waveforms in the vehicle-to-vehicle channel," 2010 48th Annual Allerton Conference on

Communication, Control, and Computing (Allerton), Allerton, IL, 2010, pp. 542-551. doi: 10.1109/ALLERTON.2010.5706954

4 P. Alexander, D. Haley and A. Grant, "Cooperative Intelligent Transport Systems: 5.9-GHz Field Trials," in Proceedings of the IEEE, vol. 99, no. 7, pp. 1213-1235, July 2011.

802.11 Dynamic Rate Control Simulation

This example shows dynamic rate control by varying the Modulation and Coding scheme (MCS) of successive packets transmitted over a frequency selective multipath fading channel.

Introduction

The IEEE® 802.11™ standard supports dynamic rate control by adjusting the MCS value of each transmitted packet based on the underlying radio propagation channel. Maximizing link throughput, in a propagation channel that is time varying due to multipath fading or movement of the surrounding objects, requires dynamic variation of MCS. The IEEE 802.11 standard does not define any standardized rate control algorithm (RCA) for dynamically varying the modulation rate. The implementation of RCA is left open to the WLAN device manufacturers. This example uses a closed-loop rate control scheme. A recommended MCS for transmitting a packet is calculated at the receiver and is available at the transmitter without any feedback latency. In a real system this information would be conveyed through a control frame exchange. The MCS is adjusted for each subsequent packet in response to an evolving channel condition with noise power varying over time.

In this example, an IEEE 802.11ac™ [1] waveform consisting of a single VHT format packet is generated using the `wlanWaveformGenerator` function. The waveform is passed through a TGac channel and noise is added. The packet is synchronized and decoded to recover the PSDU. The SNR is estimated and compared against thresholds to determine which MCS is suitable for transmission of the next packet. This figure shows the processing for each packet.
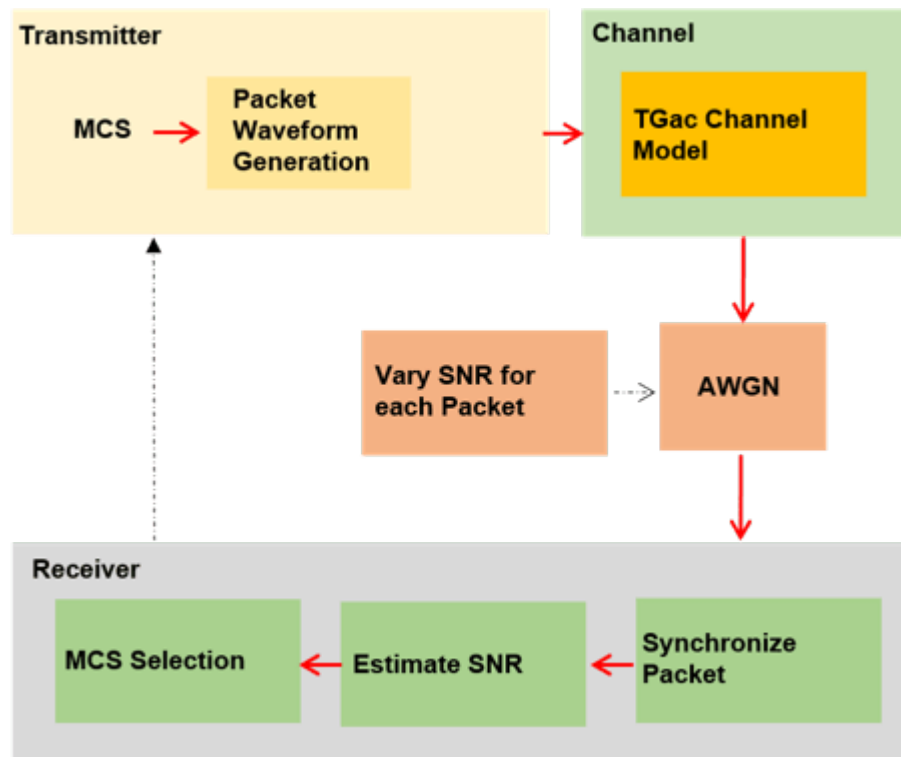
Waveform Configuration

An IEEE 802.11ac VHT transmission is simulated in this example. The VHT waveform properties are specified in a `wlanVHTConfig` configuration object. In this example the object is initially configured

for a 40 MHz channel bandwidth, single transmit antenna and QPSK rate-1/2 (MCS 1). The MCS for the subsequent packets is changed by the algorithm throughout the simulation.

```
cfgVHT = wlanVHTConfig;
cfgVHT.ChannelBandwidth = 'CBW40'; % 40 MHz channel bandwidth
cfgVHT.MCS = 1;                    % QPSK rate-1/2
cfgVHT.APEPLength = 4096;          % APEP length in bytes

% Set random stream for repeatability of results
s = rng(21);
```

Channel Configuration

In this example a TGac N-LOS channel model is used with delay profile Model-D. For Model-D when the distance between the transmitter and receiver is greater than or equal to 10 meters, the model is NLOS. This is described further in wlanTGacChannel.

```
tgacChannel = wlanTGacChannel;
tgacChannel.DelayProfile = 'Model-D';
tgacChannel.ChannelBandwidth = cfgVHT.ChannelBandwidth;
tgacChannel.NumTransmitAntennas = 1;
tgacChannel.NumReceiveAntennas = 1;
tgacChannel.TransmitReceiveDistance = 20; % Distance in meters for NLOS
tgacChannel.RandomStream = 'mt19937ar with seed';
tgacChannel.Seed = 0;

% Set the sampling rate for the channel
sr = wlanSampleRate(cfgVHT);
tgacChannel.SampleRate = sr;
```

Rate Control Algorithm Parameters

Typically RCAs use channel quality or link performance metrics, such as SNR or packet error rate, for rate selection. The RCA presented in this example estimates the SNR of a received packet. On reception, the estimated SNR is compared against a predefined threshold. If the SNR exceeds the predefined threshold then a new MCS is selected for transmitting the next packet. The rcaAttack and rcaRelease controls smooth rate changes to avoid changing rates prematurely. The SNR must exceed the threshold + rcaAttack value to increase the MCS and must be under the threshold - rcaRelease value to decrease the MCS. In this simulation rcaAttack and rcaRelease are set to conservatively increase the MCS and aggressively reduce it. For the threshold values selected for the scenario simulated in this example, a small number of packet errors are expected. These settings may not be suitable for other scenarios.

```
rcaAttack = 1;  % Control the sensitivity when MCS is increasing
rcaRelease = 0; % Control the sensitivity when MCS is decreasing
threshold = [11 14 19 20 25 28 30 31 35];
snrUp = [threshold inf]+rcaAttack;
snrDown = [-inf threshold]-rcaRelease;
snrInd = cfgVHT.MCS; % Store the start MCS value
```

Simulation Parameters

In this simulation numPackets packets are transmitted through a TGac channel, separated by a fixed idle time. The channel state is maintained throughout the simulation, therefore the channel evolves slowly over time. This evolution slowly changes the resulting SNR measured at the receiver. Since the TGac channel changes very slowly over time, here an SNR variation at the receiver visible over a short simulation can be forced using the walkSNR parameter to modify the noise power:

1   Setting `walkSNR` to true generates a varying SNR by randomly setting the noise power per packet during transmission. The SNR walks between 14-33 dB (using the `amplitude` and `meanSNR` variables).

2   Setting `walkSNR` to false fixes the noise power applied to the received waveform, so that channel variations are the main source of SNR changes at the receiver.

```
numPackets = 100; % Number of packets transmitted during the simulation
walkSNR = true;

% Select SNR for the simulation
if walkSNR
    meanSNR = 22;    % Mean SNR
    amplitude = 14; % Variation in SNR around the average mean SNR value
    % Generate varying SNR values for each transmitted packet
    baseSNR = sin(linspace(1,10,numPackets))*amplitude+meanSNR;
    snrWalk = baseSNR(1); % Set the initial SNR value
    % The maxJump controls the maximum SNR difference between one
    % packet and the next
    maxJump = 0.5;
else
    % Fixed mean SNR value for each transmitted packet. All the variability
    % in SNR comes from a time varying radio channel
    snrWalk = 22; %#ok<UNRCH>
end

% To plot the equalized constellation for each spatial stream set
% displayConstellation to true
displayConstellation = false;
if displayConstellation
    ConstellationDiagram = comm.ConstellationDiagram; %#ok<UNRCH>
    ConstellationDiagram.ShowGrid = true;
    ConstellationDiagram.Name = 'Equalized data symbols';
end

% Define simulation variables
snrMeasured = zeros(1,numPackets);
MCS = zeros(1,numPackets);
ber = zeros(1,numPackets);
packetLength = zeros(1,numPackets);
```

Processing Chain

The following processing steps occur for each packet:

1   A PSDU is created and encoded to create a single packet waveform.

2   A fixed idle time is added between successive packets.

3   The waveform is passed through an evolving TGac channel.

4   AWGN is added to the transmitted waveform to create the desired average SNR per subcarrier.

5   This local function `processPacket` passes the transmitted waveform through the TGac channel, performs receiver processing, and SNR estimation.

6   The VHT-LTF is extracted from the received waveform. The VHT-LTF is OFDM demodulated and channel estimation is performed.

7   The VHT Data field is extracted from the synchronized received waveform.

8 Noise estimation is performed using the demodulated data field pilots and single-stream channel estimate at pilot subcarriers.

9 The estimated SNR for each packet is compared against the threshold, the comparison is used to adjust the MCS for the next packet.

10 The PSDU is recovered using the extracted VHT-Data field.

For simplicity, this example assumes:

1 Fixed bandwidth and antenna configuration for each transmitted packet.

2 There is no explicit feedback packet to inform the transmitter about the suggested MCS setting for the next packet. The example assumes that this information is known to the transmitter before transmitting the subsequent packet.

3 Fixed idle time of 0.5 milliseconds between packets.

```matlab
for numPkt = 1:numPackets
    if walkSNR
        % Generate SNR value per packet using random walk algorithm biased
        % towards the mean SNR
        snrWalk = 0.9*snrWalk+0.1*baseSNR(numPkt)+rand(1)*maxJump*2-maxJump;
    end

    % Generate a single packet waveform
    txPSDU = randi([0,1],8*cfgVHT.PSDULength,1,'int8');
    txWave = wlanWaveformGenerator(txPSDU,cfgVHT,'IdleTime',5e-4);

    % Receive processing, including SNR estimation
    y = processPacket(txWave,snrWalk,tgacChannel,cfgVHT);

    % Plot equalized symbols of data carrying subcarriers
    if displayConstellation && ~isempty(y.EstimatedSNR)
        release(ConstellationDiagram);
        ConstellationDiagram.ReferenceConstellation = wlanReferenceSymbols(cfgVHT);
        ConstellationDiagram.Title = ['Packet ' int2str(numPkt)];
        ConstellationDiagram(y.EqDataSym(:));
        drawnow
    end

    % Store estimated SNR value for each packet
    if isempty(y.EstimatedSNR)
        snrMeasured(1,numPkt) = NaN;
    else
        snrMeasured(1,numPkt) = y.EstimatedSNR;
    end

    % Determine the length of the packet in seconds including idle time
    packetLength(numPkt) = y.RxWaveformLength/sr;

    % Calculate packet error rate (PER)
    if isempty(y.RxPSDU)
        % Set the PER of an undetected packet to NaN
        ber(numPkt) = NaN;
    else
        [~,ber(numPkt)] = biterr(y.RxPSDU,txPSDU);
    end

    % Compare the estimated SNR to the threshold, and adjust the MCS value
```

```
    % used for the next packet
    MCS(numPkt) = cfgVHT.MCS; % Store current MCS value
    increaseMCS = (mean(y.EstimatedSNR) > snrUp((snrInd==0)+snrInd));
    decreaseMCS = (mean(y.EstimatedSNR) <= snrDown((snrInd==0)+snrInd));
    snrInd = snrInd+increaseMCS-decreaseMCS;
    cfgVHT.MCS = snrInd-1;
end
```

Display and Plot Simulation Results

This example plots the variation of MCS, SNR, BER, and data throughput over the duration of the simulation.

1 The MCS used to transmit each packet is plotted. When compared to the estimated SNR, you can see the MCS selection is dependent on the estimated SNR.

2 The bit error rate per packet depends on the channel conditions, SNR, and MCS used for transmission.

3 The throughput is maximized by varying the MCS according to the channel conditions. The throughput is calculated using a sliding window of three packets. For each point plotted, the throughput is the number of data bits, successfully recovered over the duration of three packets. The length of the sliding window can be increased to further smooth the throughput. You can see drops in the throughput either when the MCS decreases or when a packet error occurs.

```
% Display and plot simulation results
disp(['Overall data rate: ' num2str(8*cfgVHT.APEPLength*(numPackets-numel(find(ber)))/sum(packetl
disp(['Overall packet error rate: ' num2str(numel(find(ber))/numPackets)]);

plotResults(ber,packetLength,snrMeasured,MCS,cfgVHT);

% Restore default stream
rng(s);

Overall data rate: 20.631 Mbps
Overall packet error rate: 0.03
```
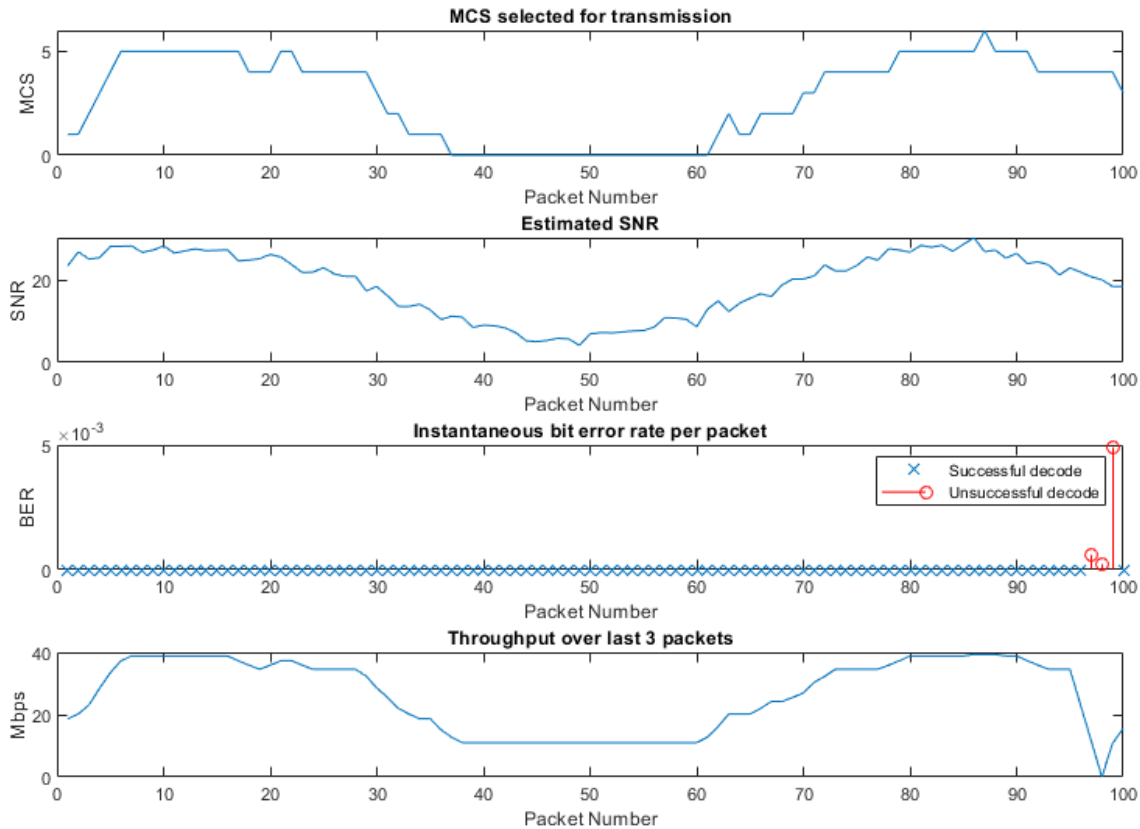
Conclusion and Further Exploration

This example uses a closed-loop rate control scheme where knowledge of the MCS used for subsequent packet transmission is assumed to be available to the transmitter.

In this example the variation in MCS over time due to the received SNR is controlled by the `threshold`, `rcaAttack` and `rcaRelease` parameters. The `rcaAttack` and `rcaRelease` are used as controls to smooth the rate changes, this is to avoid changing rates prematurely. Try changing the `rcaRelease` control to two. In this case, the decrease in MCS is slower to react when channel conditions are not good, resulting in higher BER.

Try setting the `displayConstellation` to true in order to plot the equalized symbols per received packet, you can see the modulation scheme changing over time. Also try setting `walkSNR` to false in order to visualize the MCS change per packet. Here the variability in SNR is only caused by the radio channel, rather than the combination of channel and random walk.

Further exploration includes using an alternate RCA scheme, more realistic MCS variation including changing number of space time streams, packet size and enabling STBC for subsequent transmitted packets.

Appendix

This example uses the following helper functions:

- vhtNoiseEstimate.m
- vhtSingleStreamChannelEstimate.m

Selected Bibliography

1 IEEE Std 802.11ac™-2013 IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz.

Local Functions

The following local functions are used in this example:

- `processPacket`: Add channel impairments and decode receive packet
- `plotResults`: Plot the simulation results

```matlab
function Y = processPacket(txWave,snrWalk,tgacChannel,cfgVHT)
    % Pass the transmitted waveform through the channel, perform
    % receiver processing, and SNR estimation.

    chanBW = cfgVHT.ChannelBandwidth; % Channel bandwidth
    % Set the following parameters to empty for an undetected packet
    estimatedSNR = [];
    eqDataSym = [];
    noiseVarVHT = [];
    rxPSDU = [];

    % Get the OFDM info
    ofdmInfo = wlanVHTOFDMInfo('VHT-Data',cfgVHT);

    % Pass the waveform through the fading channel model
    rxWave = tgacChannel(txWave);

    % Create an instance of the AWGN channel for each transmitted packet
    awgnChannel = comm.AWGNChannel;
    awgnChannel.NoiseMethod = 'Signal to noise ratio (SNR)';
    % Normalization
    awgnChannel.SignalPower = 1/tgacChannel.NumReceiveAntennas;
    % Account for energy in nulls
    awgnChannel.SNR = snrWalk-10*log10(ofdmInfo.FFTLength/ofdmInfo.NumTones);

    % Add noise
    rxWave = awgnChannel(rxWave);
    rxWaveformLength = size(rxWave,1); % Length of the received waveform

    % Recover packet
    ind = wlanFieldIndices(cfgVHT); % Get field indices
    pktOffset = wlanPacketDetect(rxWave,chanBW); % Detect packet

    if ~isempty(pktOffset) % If packet detected
        % Extract the L-LTF field for fine timing synchronization
        LLTFSearchBuffer = rxWave(pktOffset+(ind.LSTF(1):ind.LSIG(2)),:);

        % Start index of L-LTF field
        finePktOffset = wlanSymbolTimingEstimate(LLTFSearchBuffer,chanBW);
```

```matlab
        % Determine final packet offset
        pktOffset = pktOffset+finePktOffset;

        if pktOffset<15 % If synchronization successful
            % Extract VHT-LTF samples from the waveform, demodulate and
            % perform channel estimation
            VHTLTF = rxWave(pktOffset+(ind.VHTLTF(1):ind.VHTLTF(2)),:);
            demodVHTLTF = wlanVHTLTFDemodulate(VHTLTF,cfgVHT);
            chanEstVHTLTF = wlanVHTLTFChannelEstimate(demodVHTLTF,cfgVHT);

            % Get single stream channel estimate
            chanEstSSPilots = vhtSingleStreamChannelEstimate(demodVHTLTF,cfgVHT);

            % Extract VHT data field
            vhtdata = rxWave(pktOffset+(ind.VHTData(1):ind.VHTData(2)),:);

            % Estimate the noise power in VHT data field
            noiseVarVHT = vhtNoiseEstimate(vhtdata,chanEstSSPilots,cfgVHT);

            % Recover equalized symbols at data carrying subcarriers using
            % channel estimates from VHT-LTF
            [rxPSDU,~,eqDataSym] = wlanVHTDataRecover(vhtdata,chanEstVHTLTF,noiseVarVHT,cfgVHT);

            % SNR estimation per receive antenna
            powVHTLTF = mean(VHTLTF.*conj(VHTLTF));
            estSigPower = powVHTLTF-noiseVarVHT;
            estimatedSNR = 10*log10(mean(estSigPower./noiseVarVHT));
        end
    end

    % Set output
    Y = struct( ...
        'RxPSDU',           rxPSDU, ...
        'EqDataSym',        eqDataSym, ...
        'RxWaveformLength', rxWaveformLength, ...
        'NoiseVar',         noiseVarVHT, ...
        'EstimatedSNR',     estimatedSNR);

end

function plotResults(ber,packetLength,snrMeasured,MCS,cfgVHT)
    % Visualize simulation results

    figure('Outerposition',[50 50 900 700])
    subplot(4,1,1);
    plot(MCS);
    xlabel('Packet Number')
    ylabel('MCS')
    title('MCS selected for transmission')

    subplot(4,1,2);
    plot(snrMeasured);
    xlabel('Packet Number')
    ylabel('SNR')
    title('Estimated SNR')

    subplot(4,1,3);
```

```matlab
        plot(find(ber==0),ber(ber==0),'x')
        hold on; stem(find(ber>0),ber(ber>0),'or')
        if any(ber)
            legend('Successful decode','Unsuccessful decode')
        else
            legend('Successful decode')
        end
        xlabel('Packet Number')
        ylabel('BER')
        title('Instantaneous bit error rate per packet')

        subplot(4,1,4);
        windowLength = 3; % Length of the averaging window
        movDataRate = movsum(8*cfgVHT.APEPLength.*(ber==0),windowLength)./movsum(packetLength,windowL
        plot(movDataRate)
        xlabel('Packet Number')
        ylabel('Mbps')
        title(sprintf('Throughput over last %d packets',windowLength))

    end
```

System-Level Simulation

802.11ax Multinode System-Level Simulation of Residential Scenario Using MATLAB

This example shows how to model performance of an IEEE® 802.11ax™ [1 on page 7-0   ] network in a residential scenario by using WLAN Toolbox™.

Using this example, you can -

- Model a residential scenario by configuring the network and channel parameters.
- Simulate a multinode WLAN system and visualize the network-related statistics.

The Results section plots performance metrics such as throughput, latency, and packet loss.

Residential Scenario Description

This example demonstrates a system-level simulation to evaluate the performance of an 802.11ax network in a residential scenario. The residential scenario consists of a building with 3 floors. The spacing between the floors is 1.5 meters. Each floor consists of four rooms, each having dimensions 10m x 10m x 3m. Each room has an access point (AP) and two stations (STAs) placed in random *x*- and *y*- locations at a height of 1.5 meters from the floor. Each AP has data for STAs present in the same room. The simulation scenario specifies a path loss model based on the distance between the nodes, and the number of walls and floors traversed by the WLAN signal. This figure shows the residential scenario simulated in this example.
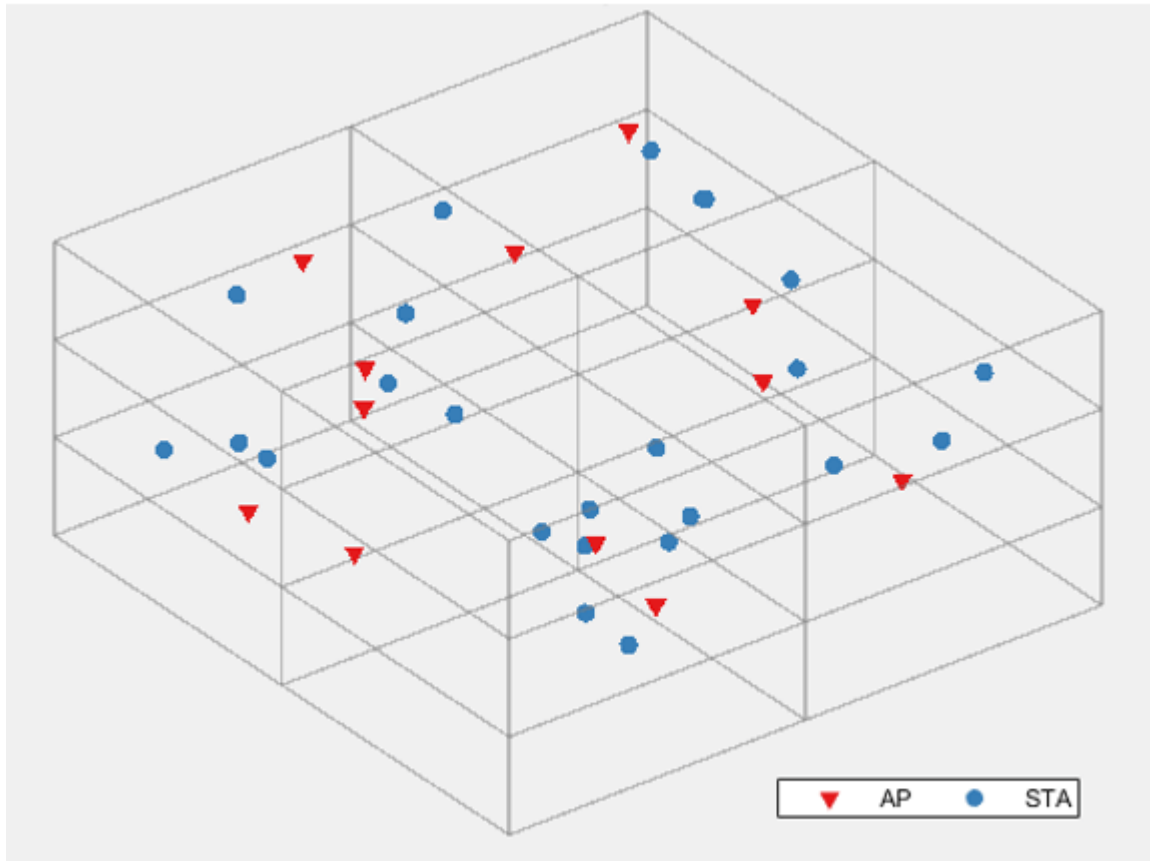
Residential Scenario

This example models the medium access control (MAC) layer and physical layer (PHY) of all the nodes (APs and STAs) using abstractions. The MAC layer implements enhanced distributed channel access (EDCA) functionality. The MAC layer uses abstraction for frame generation and decoding. Abstraction refers to the fact that MAC layer sends and receives frame metadata instead of sending or receiving encoded MAC frame bits. Similarly, the PHY uses abstraction for WLAN signal generation and decoding. For more information on PHY abstraction, see Physical Layer Abstraction for System-Level Simulation example.

This example is calibrated against Box-3 and Box-5 scenarios specified in TGax evaluation methodology [2 on page 7-0   ]. The network throughput calculated for the scenarios mentioned in TGax simulation scenarios document [3 on page 7-0   ] are validated against the published calibration results from the TGax Task Group, to confirm compliance with IEEE 802.11.

Configuration Parameters

Simulation Parameters

Set the seed for the random number generator to 1. For greater accuracy in the simulation results, change the seed and average the results over multiple simulations. Specify simulation time in microseconds using `simulationTime` variable. To visualize a live state transition plot for all nodes, set `showLiveStateTransitionPlot` variable to true. To visualize the table containing network statistics at the end of simulation, set `displayStatsInUITable` variable to true.

```
rng(1,'twister');                      % Seed for random number generator
simulationTime = 0.1*1e6;              % Simulation time in microseconds
showLiveStateTransitionPlot = true;    % Show live state transition plot for all nodes
displayStatsInUITable = true;          % Display table of statistics

% Add the folder to the path for access to all helper files
addpath(genpath(fullfile(pwd, 'mlWLANSystemSimulation')));
```

Residential Scenario Parameters

The ScenarioParameters structure defines the size and layout of the residential building using these parameters.

- BuildingLayout: Specifies the building layout in terms of number of rooms in each of the three directions

- RoomSize: Specifies the size of each room in meters

- NumRxPerRoom: Specifies the number of stations per room

The example assumes one transmitting AP and two receiving STAs in each room. The hDropNodes function randomly generates the positions of the AP and STAs within each room.

```
ScenarioParameters = struct;
% Number of rooms in [x,y,z] directions
ScenarioParameters.BuildingLayout = [2 2 3];
% Size of each room in meters [x,y,z]
ScenarioParameters.RoomSize = [10 10 3];
% Number of STAs per room
ScenarioParameters.NumRxPerRoom = 2;

% Obtain random positions for placing nodes
[apPositions, staPositions] = hDropNodes(ScenarioParameters);
```

Node Parameters

The hLoadConfiguration function loads the MAC and PHY configurations for the nodes specified by nodeConfigs and loads the application traffic configuration for the transmitting nodes specified by trafficConfigs. This function assigns identifiers (IDs) and positions to all the nodes in the network.

```
% Get the IDs and positions of each node
[nodeConfigs, trafficConfigs] = hLoadConfiguration(ScenarioParameters, apPositions, staPositions);
```

The wlanNodeConfig.mat file defines the structure for specifying the MAC and PHY configuration of a node. For more information about the detailed configuration parameters in this MAT file, use the command hConfigurationHelp('wlanNodeConfig'). The nodeConfigs output of the hLoadConfiguration function is an array of these structures. You can modify the MAC configuration parameters such as format, channel bandwidth, modulation and coding scheme (MCS) index, for the transmitted packets. You can also modify the physical layer parameters such as transmit power, transmit gain, receive gain, noise power. For example, this code configures node-1 to transmit packets with a fixed MCS-6.

```
nodeConfigs(1).TxMCS = 6

nodeConfigs=1×36 struct array with fields:
    NodePosition
    TxFormat
```

```
          Bandwidth
          TxMCS
          NumTxChains
          MPDUAggregation
          DisableAck
          MaxSubframes
          RTSThreshold
          DisableRTS
          MaxShortRetries
          MaxLongRetries
          BasicRates
          Use6MbpsForControlFrames
          BandAndChannel
          CWMin
          CWMax
          AIFSSlots
          RateControl
          PowerControl
          TxPower
          TxGain
          RxGain
          EDThreshold
          RxNoiseFigure
          ReceiverRange
          FreeSpacePathloss
          PHYAbstractionType
            ⋮
```

The `wlanTrafficConfig.mat` file defines the structure for specifying the application traffic configuration. For more information about the detailed configuration parameters in this MAT file, use the command `hConfigurationHelp('wlanTrafficConfig')`. The `trafficConfigs` output of the `hLoadConfiguration` function is an array of these structures. Each structure corresponds to a specific destination STA node. You can modify parameters like packet size, data rate, or access category for each application in the array. The simulation scenario in this example configures the Best Effort (AC0) traffic from APs to STAs. For example, the first structure in the array specifies application traffic for node-1 (AP) to node-13 (STA). For example, this code configures the transmission of 1000 byte-sized application packets from node-1 to node-13. All other transmitters use a default packet size of 1500 bytes.

`trafficConfigs(1).PacketSize = 1000`

```
trafficConfigs=1×24 struct array with fields:
    SourceNode
    DestinationNode
    PacketSize
    DataRateKbps
    AccessCategory
```

Create Network

Create transmitter and receiver sites from the node configurations. Create the building geometry from the scenario parameters.

```
% Create transmitter and receiver sites
[txs,rxs] = hCreateSitesFromNodes(nodeConfigs);
```

```
% Create triangulation object and visualize the scenario
tri = hTGaxResidentialTriangulation(ScenarioParameters);
hVisualizeScenario(tri,txs,rxs,apPositions);
```
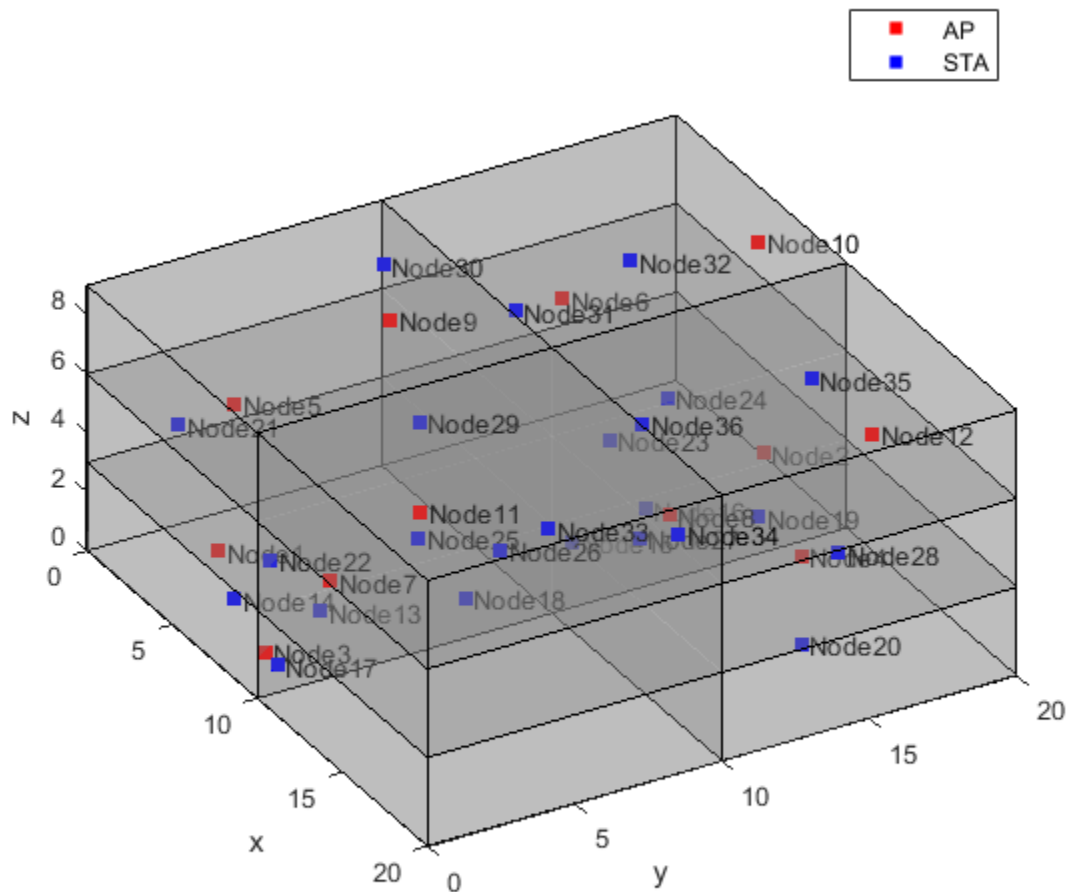

This example uses the TGax residential propagation model to determine pathloss between nodes. Path loss is a function of the number of walls, floors, and the distance between nodes. Create a path loss model using `hTGaxResidentialPathLoss` function. Create a function handle which returns the path loss between each pair of nodes in the network using the propagation model and transmitter and receiver sites. The `hCreateWLANNodes` function creates the configured WLAN nodes.

```
% Generate propagation model and lookup table
propModel = hTGaxResidentialPathLoss('Triangulation',tri,'ShadowSigma',0,'FacesPerWall',1);
[pl,tgaxIndoorPLFn] = hCreatePathlossTable(txs,rxs,propModel);
```

```
% Create WLAN nodes
wlanNodes = hCreateWLANNodes(nodeConfigs, trafficConfigs, simulationTime, tgaxIndoorPLFn);
```

Simulation

Initialize the visualization and simulation parameters.

```
% Initialize visualization parameters and create an object for
% hStatsLogger which is a helper for retrieving, and displaying
% the statistics.
visualizationInfo = struct;
visualizationInfo.DisablePlot = ~showLiveStateTransitionPlot;
visualizationInfo.Nodes = wlanNodes;
statsLogger = hStatsLogger(visualizationInfo);  % Object that handles retrieving and visualizing
networkSimulator = hWirelessNetworkSimulator;   % Object that handles network simulation
```

Run all the nodes in the network for the specified `simulationTime` time.

```
% Run the simulation
run(networkSimulator, wlanNodes, simulationTime, statsLogger);
```
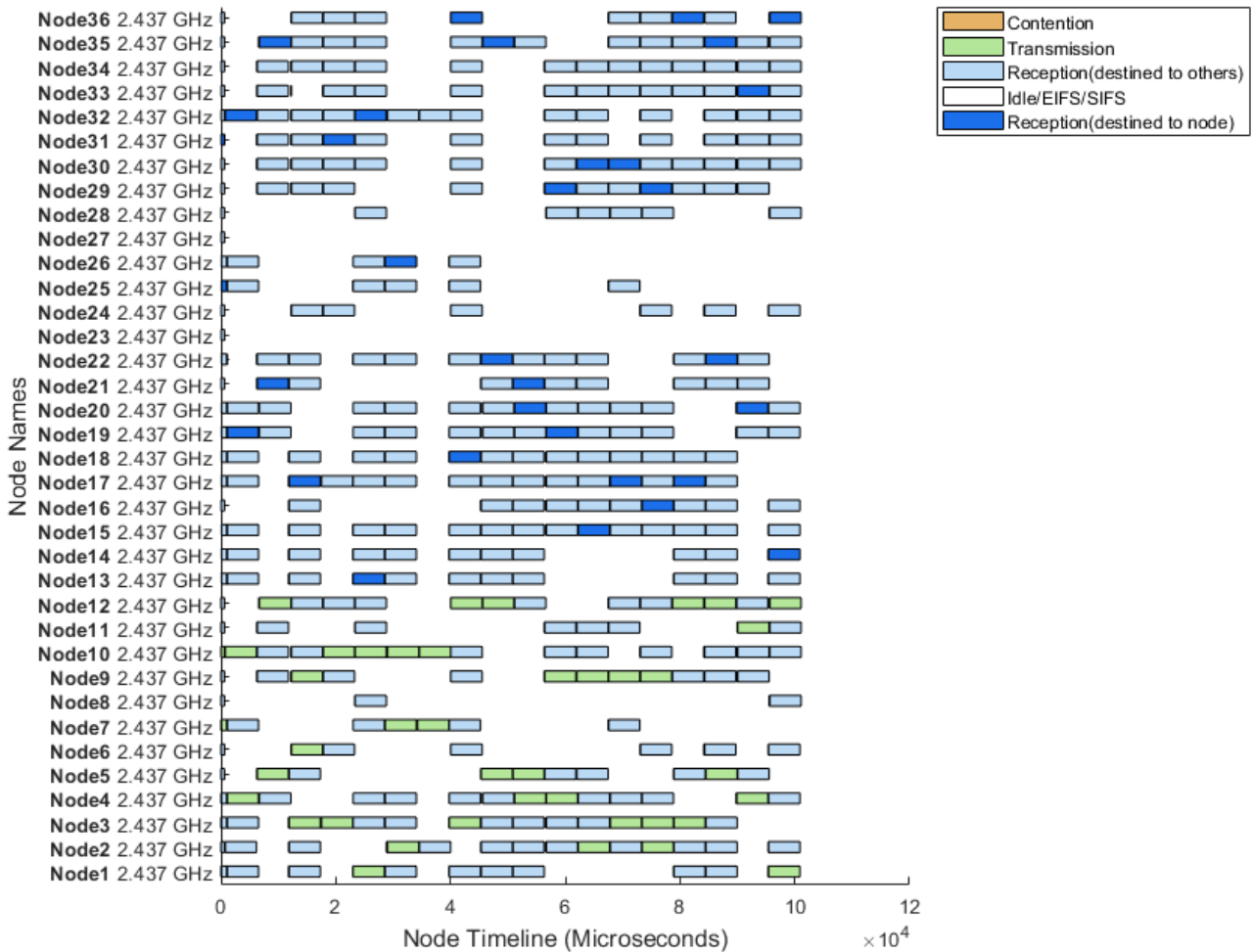
```
% Cleanup the persistent variables used in functions
clear edcaPlotStats;
```

Results

Retrieve the statistics and store them in a mat file. The UI table shows all the statistics collected during the simulation.

```
% Retrieve the statistics and store them in a mat file
statistics = getStatistics(statsLogger, ~displayStatsInUITable);
```

statisticsTable=*101×36 table*

	Node1	Node2	Node3	Node4	Node5	Node6	Node7	Nc
	-----	-----	-----	-----	-----	-----	-----	--
ActiveOperationInFreq	1	1	1	1	1	1	1	
AppTx	12500	10000	10000	10000	10000	10000	10000	1(
AppRx	0	0	0	0	0	0	0	
AppRxBytes	0	0	0	0	0	0	0	

```
AppAvgPacketLatency            0       0       0       0       0       0       0
MACInternalCollisionsAC1       0       0       0       0       0       0       0
MACInternalCollisionsAC2       0       0       0       0       0       0       0
MACInternalCollisionsAC3       0       0       0       0       0       0       0
MACInternalCollisionsAC4       0       0       0       0       0       0       0
MACBackoffAC1                261     378     486     396     261     297     459
MACBackoffAC2                  0       9     135      90       0       0      99
MACBackoffAC3                 18      72     162     135     126      27     117
MACBackoffAC4                 27      45      63      45      27      18      27
MACDataTx                     72      96     192     128     128      32      69
MACTxAC1                      72      96     192     128     128      32      69
MACTxAC2                       0       0       0       0       0       0       0
   ⋮
```

You can access all the statistics from the above table by exploring the `statistics.mat` file.

```
% Save the statistics to a mat file
save('statistics.mat', 'statistics');
```

The `hPlotNetworkStats` helper function parses the collected statistics and plots the throughput, packet loss ratio, and average packet latency values at each node. The plot shows throughput and packet loss ratio at each transmitter (APs). The plot also shows the average packet latency experienced at each receiver node (STAs). The throughput shows the achieved data rate at each AP in units of megabits per second (Mbps). The packet loss ratio shows the ratio of unsuccessful data transmissions to the total data transmissions. The average packet latency shows the average latency experienced at each STA to receive its downlink traffic from the AP.

```
% Plot the throughput, packet loss ratio, and average packet latency at each node
hPlotNetworkStats(statistics, wlanNodes);
```
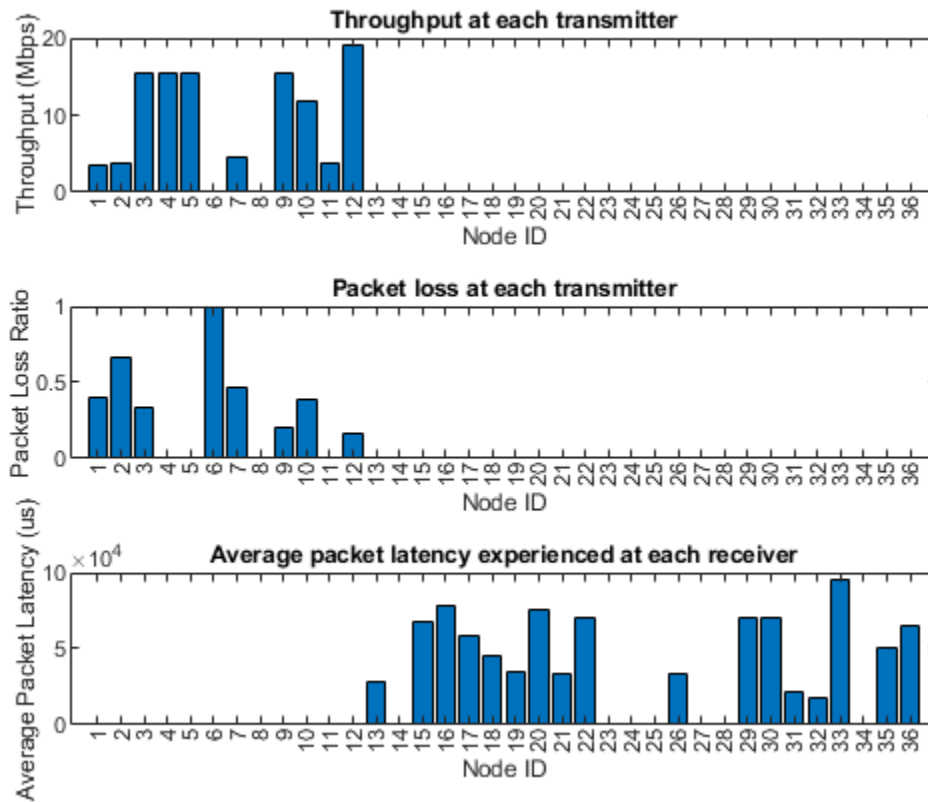
```
% Remove the folder from the path
rmpath(genpath(fullfile(pwd, 'mlWLANSystemSimulation')));
```

Further Exploration

To observe the variation of throughput when a STA moves within a room, you can run the simulation for different positions of a STA with a fixed AP. You can observe the variation of throughput with respect to distance from its AP. The set of throughputs captured at all different positions can be used to plot a heatmap like the one shown here.
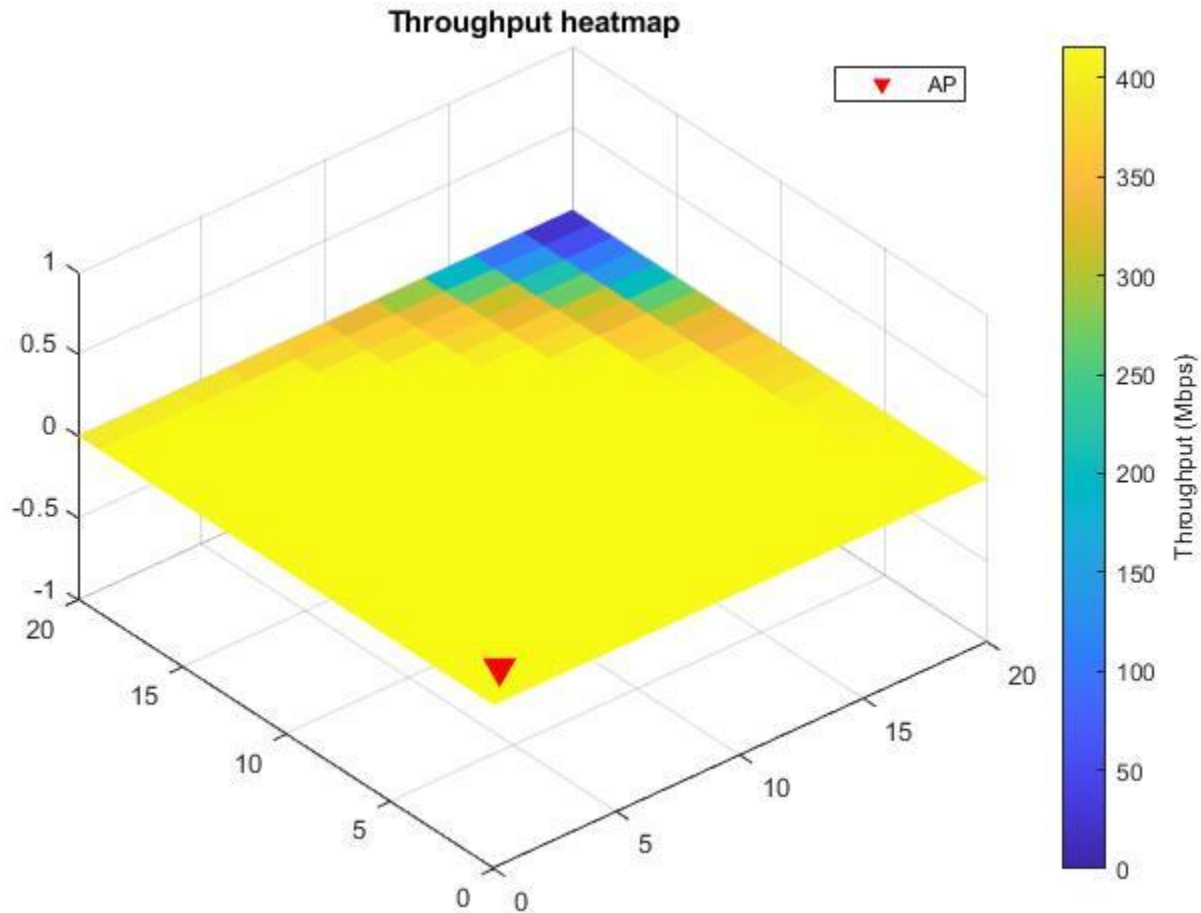
Throughput heatmap

References

1 IEEE P802.11ax™/D4.1. "Amendment 6: Enhancements for High Efficiency WLAN.." Draft Standard for Information technology - Telecommunications and information exchange between systems Local and metropolitan area networks - Specific requirements -Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

2 "TGax simulation scenarios", doc. IEEE 802.11-14/0980r16.

3 "11ax evaluation methodology", doc. IEEE 802.11-14/0571r12.

Spatial Reuse with BSS Coloring in 802.11ax Residential Scenario

This example demonstrates how to simulate the impact of spatial reuse (SR) with basic service set (BSS) coloring on the throughput of an IEEE® 802.11ax™ residential scenario network using SimEvents®, Stateflow®, and WLAN Toolbox™. This example simulates non-spatial reuse groups overlapping BSS packet detect (OBSS PD) defined in IEEE® P802.11ax™/D4.1 Amendment [1]. The fundamental objective of this feature is to increase the network performance in a dense network scenario by maximizing the frequency reuse between BSSs. This is achieved by adjusting OBSS PD threshold in the range [-62, -82] dBm. Each BSS is assigned a unique color code called BSS color. The BSS color enables 802.11ax station (STA) to decide whether the simultaneous use of frequency spectrum along with the OBSS is allowed or not. This results in ignoring the frames from OBSS and enables concurrent transmission of data to multiple devices in a dense network scenario mitigating the exposed node problem. To demonstrate the SR with BSS coloring feature, the example models a network topology consisting of four BSSs in a residential apartment of two floors. Each floor contains two BSSs and each BSS consists of an access point (AP) and a STA. The model enables you to configure the OBSS PD threshold values in the range [-62, -82] dBm and study its impact on the throughput of each BSS. You can also write your own algorithm to control OBSS PD threshold levels dynamically during the simulation.

IEEE 802.11ax OBSS PD-based Spatial Reuse Operation

In dense network scenarios consisting of legacy IEEE® 802.11 devices, multiple APs operate in the same frequency due to limited spectrum. This resulted in an inefficient paradigm causing network congestion and slowdown. Also, the legacy devices were unable to efficiently communicate with each other to maximize channel resources. To address this issue, the IEEE® P802.11ax™/D4.1 Amendment [1] introduced the OBSS PD-based SR operation to optimize the efficient reutilization of frequency spectrum in dense network scenarios. To achieve this, the number of parallel transmissions that could potentially improve network performance of BSSs operating in the same frequency is increased. To increase the number of parallel transmissions, the clear channel assessment/carrier sense (CCA/CS) threshold for the detected OBSS transmissions is adjusted to a new value called OBSS PD threshold. The OBSS PD threshold is higher then the default CCA/CS threshold. This figure illustrates the SR operation in an OBSS. The network topology consists of two BSSs, each containing an AP and a STA.
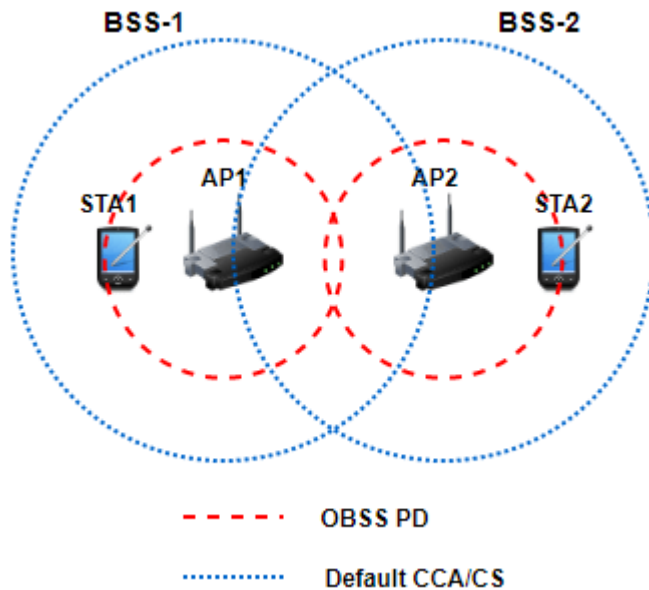
The default CCA/CS threshold (denoted by blue dashed lines) does not enable simultaneous transmissions between AP1 and AP2. In this case, transmission in BSS-2 must be deferred if devices in BSS-1 occupy the channel for transmission. However, by optimally selecting the value of OBSS PD threshold (denoted by red dashed lines), both APs can simultaneously transmit to their corresponding STAs. This SR operation when used with BSS coloring technique improves channel utilization and gives better throughput.

BSS Coloring

The SR operation uses BSS color identifier to enable 802.11ax devices to distinguish between different BSSs when other devices transmit on same channel. The BSS color is a numerical identifier of the BSS having a value in the range [1, 63]. BSS color information is present in a 6-bit BSS color field in the HE SIG-A which is present in the preamble of 802.11ax physical layer (PHY) header. If the color of the frame matches with the color of the BSS, it is an intra-BSS transmission as the transmitting device belongs to the same BSS as the receiver. If the detected frame has a different BSS color from its own, the frame is an inter-BSS frame. An 802.11ax AP can change its BSS color if it detects an OBSS using the same color. This flowchart shows the BSS coloring procedure.

```
┌─────────────────────┐
│ 11ax frame detected │
└─────────────────────┘
            │
            ▼
         ◇ Is RSSI > CCA/CS ? ◇
    No  /                  \  Yes
       ▼                    ▼
┌──────────────┐    ┌──────────────┐
│ Signal is weak.│   │ Check the BSS │
│ Ignore the     │   │ color field   │
│ received frame │   └──────────────┘
│ and go for     │          │
│ parallel       │          ▼
│ transmission   │    ◇ Is frame color same ◇
└────────────────┘  No /  as BSS color ?  \ Yes
                      ▼                     ▼
               ◇ Is RSSI > OBSS/PD ? ◇  ┌──────────────┐
          No  /                    \ Yes│ Wireless     │
             ▼                      ▼    │ medium is    │
   ┌──────────────┐      ┌──────────────┐│ busy. Intra- │
   │ Signal is weak│     │ Wireless      ││ BSS frame    │
   │ Ignore the    │     │ medium is     ││ transmissions│
   │ received frame│     │ busy. Inter-  │└──────────────┘
   │ and go for    │     │ BSS frame     │
   │ parallel      │     │ transmissions │
   │ transmission  │     └───────────────┘
   └───────────────┘
```

When detecting a transmission, a 802.11ax device differentiates between intra and inter-BSS frames by inspecting the BSS color field. For intra-BSS frames, the 802.11ax devices use the default PD threshold of -82 dBm. For inter-BSS frames, more aggressive PD threshold values are applied to increase the number of parallel transmissions.

This is the possible use case scenario of SR with BSS coloring.

Residential scenario Consider a two-floored residential apartment.
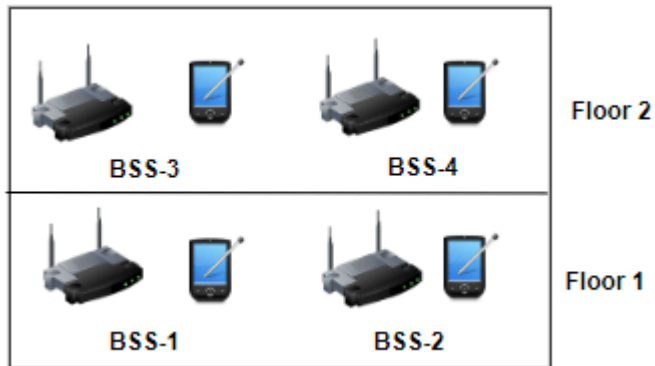
Each floor contains two BSSs and each BSS consists of an AP and an STA. In this scenario, SR with BSS coloring would improve the aggregate throughput of the network and throughput per BSS. This example models and simulates this use case scenario.

Model 802.11ax Residential Scenario with BSS Coloring

This example demonstrates the communication in an 802.11ax network with four BSSs containing one AP and one station per BSS. These stations implement carrier-sense multiple access with collision avoidance (CSMA/CA) with physical carrier sense and virtual carrier sense. The physical carrier sensing uses the clear CCA mechanism to determine whether the medium is busy before transmitting. Whereas, the virtual carrier sensing uses the RTS/CTS handshake to prevent the hidden node problem. All the nodes implement virtual carrier sensing using dual network allocation vectors (NAVs) (Intra NAV and Basic NAV) defined in IEEE® P802.11ax™/D4.1 Amendment [1].
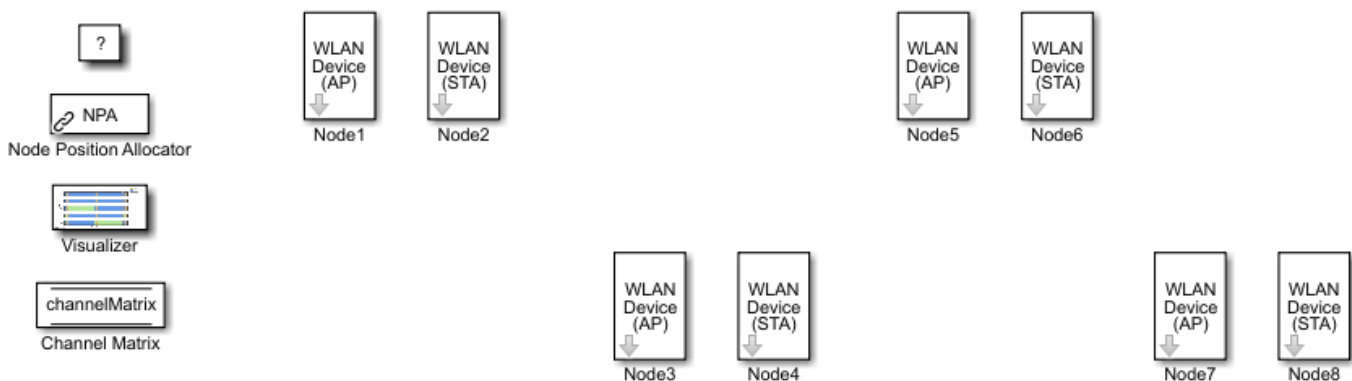
Node Configuration

Specify a node as an AP or an STA through the `NodeType` mask parameter. To configure the node as an STA, you must specify the name of the AP node associated with it through the `AP Name` parameter. All the nodes configured as STA with the same `AP Name` belong to one BSS.

In the Application Traffic Generator block, the `Destination Name` parameter of an AP is always configured to the STA nodes of the same BSS. Whereas, the `Destination Name` parameter of the STA node is always configured to the same value of the `AP Name` parameter.
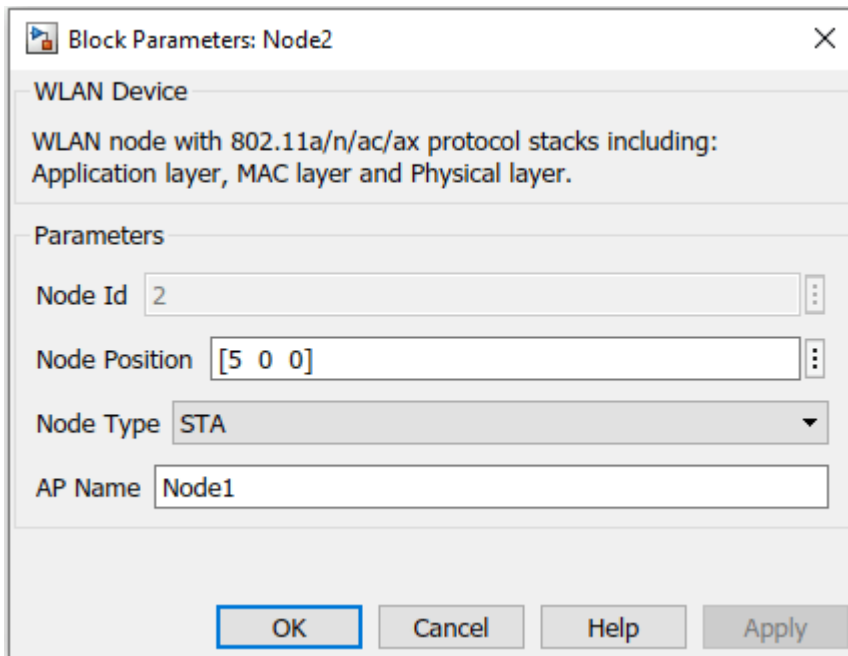
To view the components of the WLAN node, see the "802.11ax System-Level Simulation with Physical Layer Abstraction" on page 7-52 example.

MAC Configuration

Implement these steps to configure nodes to perform SR operation.

1 Set `PHY Tx Format` to HE format (`HE-SU` or `HE-EXT-SU` or `HE-MU-OFDMA`) in MAC configuration parameters.

2 To enable SR operation, select `Enable Spatial Reuse With BSS Color` option.

3 Specify the BSS color by setting the BSS Color field to a value in the range [1, 63]. All the nodes in a BSS must be configured to the same value of `BSS Color` field. Each BSS must have a unique BSS color.

4 Specify OBSS PD threshold by setting `OBSS PD Threshold` field to a value in the range [-62, -82].

Simulations Results

Run the simulation to visualize the parallel transmissions and get the aggregated throughput for BSS.

- **Runtime visualization plot to show the time spent by each node on channel contention, transmission, and reception.**

In this plot, the mapping of nodes to BSS is as shown:

- BSS1 - Node1 and Node2
- BSS2 - Node3 and Node4
- BSS3 - Node5 and Node6
- BSS4 - Node7 and Node8
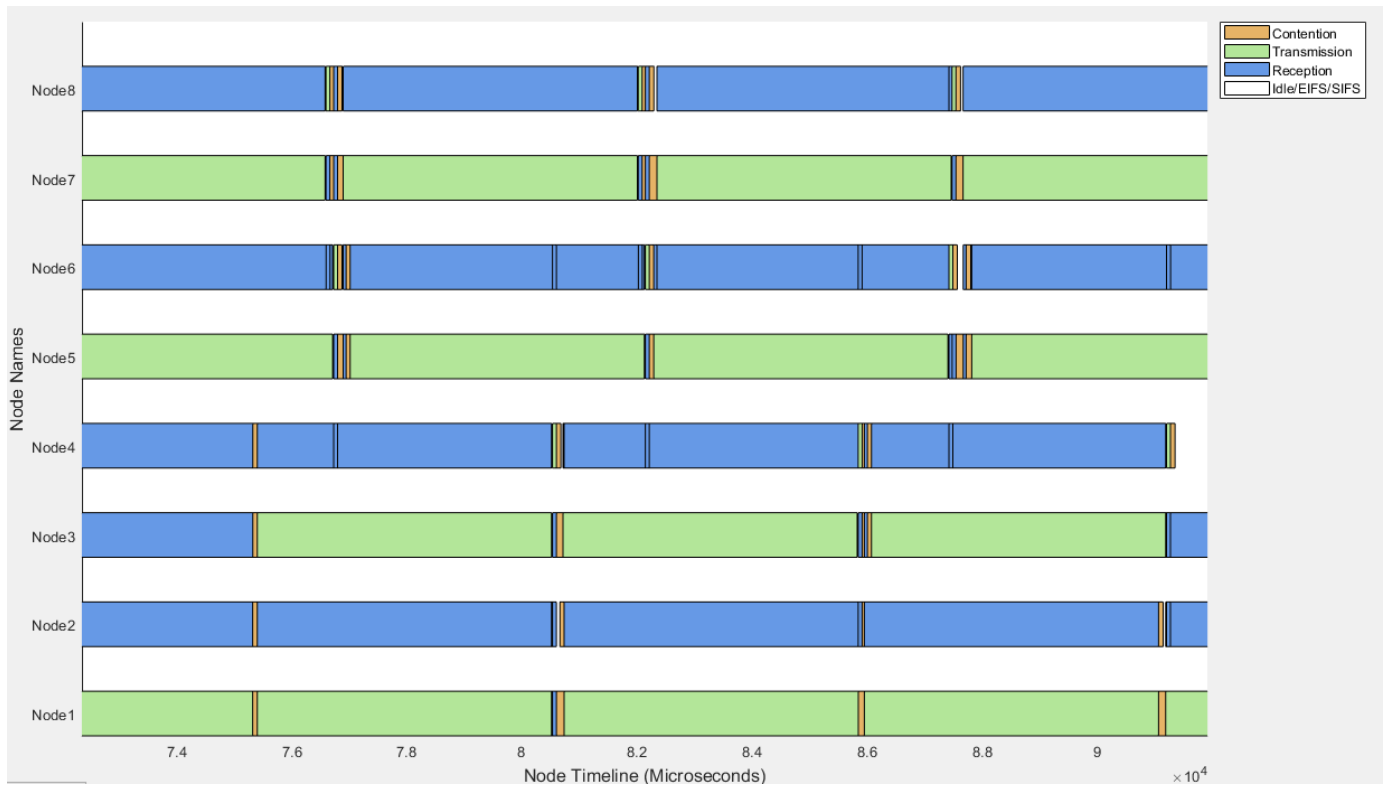
Observe that the nodes transmit the frames concurrently. Thus, the SR operation enables simultaneous transmission of data, resulting in efficient utilization of the channel.

- **Per BSS aggregated throughput plot**
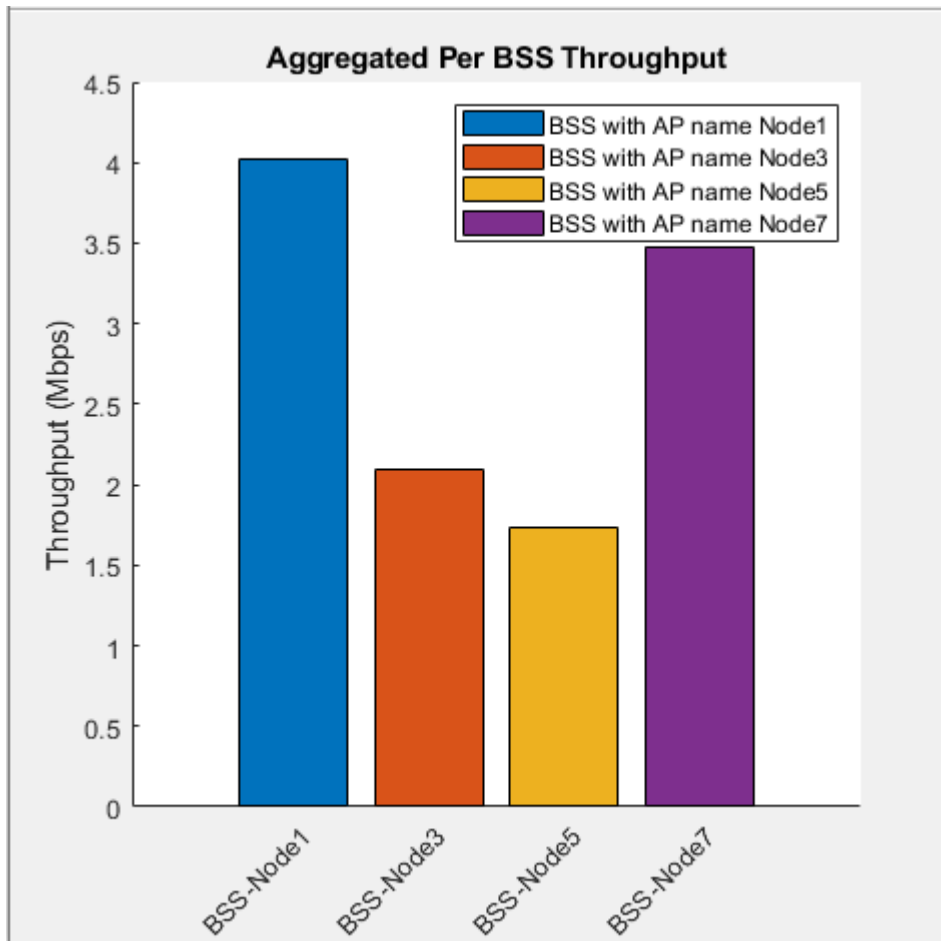
This plot shows the per BSS aggregated throughput of the network.

You can analyze the impact of varying the OBSS PD threshold values on the aggregated network throughput by running the simulations detailed in the following sections.

- **Impact of the offered load and OBSS PD threshold on the network throughput**

In each BSS, specify the STA and AP, with the AP serving to its corresponding STA. To enable STA as transmitters, set `App State` of an Application Traffic Generator block to 'On'. For each BSS, configure the AP and STA with the values shown in this table.

Parameter	Value
Packet size (bytes)	1700
Packet Interval (seconds)	0.0001
Access Category	Best Effort
Max A-MPDU Subframes	64
MCS	0
PHY Tx Format	HE-SU
Ack Policy	No Ack

RTS Threshold (bytes)	65535
Max Short Retries	10
Max Long Retries	10
Number of Transmit Chains	1
Tx Queue Size (Per Destination and Per AC)	64
Simulation time (sec)	0.5

Vary the `OBSS PD Threshold` as -82, -72, -68, -65, -62, for each simulation. At the end of each simulation run, the obtained throughput value is retrieved from the `statistics.mat` file. Plot the impact of offered load and OBSS PD threshold on the network throughput.

This code plots the impact of offered load and OBSS PD threshold on the average network throughput, when AP is transmitting to its corresponding STA.

```
% Offered load (Mbps)
offeredLoad = [40 64 88 112 126 160];

% Throughput results for OBSS PD threshold -82 dBm at varying offered loads (Mbps)
throughput1 = [1.7640 1.7152 1.8524 1.8536 1.8020 1.8320];

% Throughput results for OBSS PD threshold -72 dBm at varying offered loads (Mbps)
throughput2 = [2.5900 2.3392 2.4860 2.4416 2.4820 2.5360];

% Throughput results for OBSS PD threshold -68 dBm at varying offered loads (Mbps)
throughput3 = [3.0040 2.9632 2.9964 2.8112 2.9036 3.0480];

% Throughput results for OBSS PD threshold -65 dBm at varying offered loads (Mbps)
throughput4 = [3.0400 3.2672 3.4012 3.2984 3.3388 3.4320];

% Throughput results for OBSS PD threshold -62 dBm at varying offered loads (Mbps)
throughput5 = [2.8780 2.9952 3.0140 3.3320 3.2708 3.2720];

% Get screen resolution
resolution = get(0, 'screensize');
screenWidth = resolution(3);
screenHeight = resolution(4);
figureWidth = screenWidth*0.7;
figureHeight = screenHeight*0.7;

% Create figure
figure('Position', [screenWidth*0.2, screenHeight*0.1, figureWidth, figureHeight])

% Retain plot
hold on

% Plot throughput obtained with OBSS PD threshold set to -82 dBm at varying
% offered loads (Mbps) and retain plot
plot(offeredLoad, throughput1,'-o')
hold on

% Plot throughput obtained with OBSS PD threshold set to -72 dBm at varying
% offered loads (Mbps) and retain plot
plot(offeredLoad, throughput2,'-x')
hold on
```

```matlab
% Plot throughput obtained with OBSS PD threshold set to -68 dBm at varying
% offered loads (Mbps) and retain plot
plot(offeredLoad, throughput3,'-+')
hold on

% Plot throughput obtained with OBSS PD threshold set to -65 dBm at varying
% offered loads (Mbps) and retain plot
plot(offeredLoad, throughput4,'-s')
hold on

% Plot throughput obtained with OBSS PD threshold set to -62 dBm at varying
% offered loads (Mbps)
plot(offeredLoad, throughput5,'-d')
grid on

% Add X-label, Y-label, and legend
xlabel('Offered load (Mbps)')
ylabel('Throughput (Mbps)')
legend('No Spatial Reuse', 'Spatial Reuse OBSS PD -72 dBm', ...
    'Spatial Reuse OBSS PD -68 dBm', 'Spatial Reuse OBSS PD -65 dBm', ...
    'Spatial Reuse OBSS PD -62 dBm', 'Location', 'northeastoutside');

% Add title
title('Impact of offered load and OBSS PD threshold on the network throughput')
```

Impact of offered load and OBSS PD threshold on the network throughput
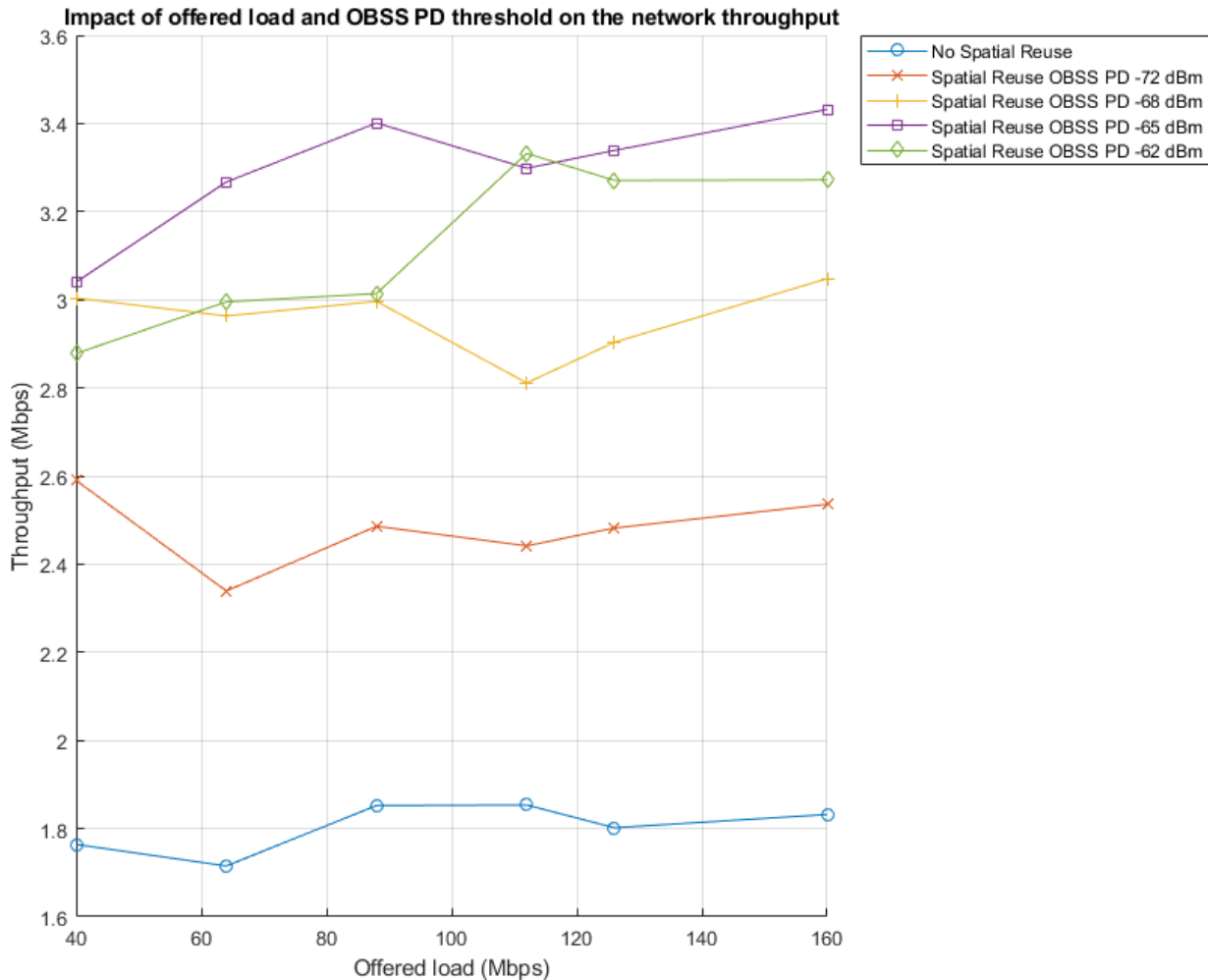
Observe that the network throughput is higher when SR operation is enabled. Also, throughput increases with an increase in the OBSS PD threshold.

- **Impact of OBSS PD threshold on the throughput of each BSS**

Vary the OBSS PD threshold as -82, -77, -72, -67, and -62 dBm and plot the impact on the throughput of each BSS.

```
numOBSSPDValues = 5;

OBSSPDValues = {'-82', '-77', '-72', '-67', '-62'};

% Initial SR statistics
throughput = zeros(numOBSSPDValues, 1);

% Throughput results for BSS1 (Mbps)
throughputWithOutSpatialBSS1 = [2.1216 4.5696 4.5696 4.7328 3.318];
```

```matlab
% Throughput results for BSS2 (Mbps)
throughputWithSpatialReuseBSS2 = [1.9040 1.0336 1.7952 1.1686 2.448];

% Throughput results for BSS3 (Mbps)
throughputWithSpatialReuseBSS3 = [2.0672 2.0128 1.1088 2.23 3.590];

% Throughput results for BSS4 (Mbps)
throughputWithSpatialReuseBSS4 = [1.1152 2.3120 4.1616 4.706 3.726];

% Create a plot matrix with different statistics.
PlotMatrix = [throughputWithOutSpatialBSS1' throughputWithSpatialReuseBSS2'...
    throughputWithSpatialReuseBSS3' throughputWithSpatialReuseBSS4'];

nextBarOffset = 1;

% Set colors for the bars
colors = [ 0.9294    0.6941    0.1255;
    0.4941    0.1843    0.5569;
    0.4667    0.6745    0.1882;
    0.3020    0.7451    0.9333;
    0.6353    0.0784    0.1843;];

% Get screen resolution
resolution = get(0, 'screensize');
screenWidth = resolution(3);
screenHeight = resolution(4);
figureWidth = screenWidth*0.7;
figureHeight = screenHeight*0.7;

% Create figure
figure('Name', 'MultiNode 802.11 Network (PHY and MAC) Statistics', ...
    'Position', [screenWidth*0.2, screenHeight*0.1, figureWidth, figureHeight]);

% Set figure axis.
figAxis = gca;
hold on

% Plot using the plot matrix created
for i = 1: 5
    bar(nextBarOffset, PlotMatrix(i,1), 'FaceColor', colors(1, :))
    nextBarOffset = nextBarOffset+1;
    bar(nextBarOffset, PlotMatrix(i,2), 'FaceColor', colors(2, :))
    nextBarOffset = nextBarOffset+1;
    bar(nextBarOffset, PlotMatrix(i,3), 'FaceColor', colors(3, :))
    nextBarOffset = nextBarOffset+1;
    bar(nextBarOffset, PlotMatrix(i,4), 'FaceColor', colors(4, :))
    nextBarOffset = nextBarOffset+6;
end
lastBarOffset = nextBarOffset;

% Set legend to the plot
legend('BSS1', 'BSS2', ...
    'BSS3', 'BSS4', 'Location', 'northeastoutside');

% Name X-axis of the plot
xlabel('OBSS PD Threshold (dBm)')

% Name Y-axis of the plot
```

```
ylabel('Throughput (Mbps)')

% Gap between each tick on X-axis
tickInterval = (lastBarOffset-1)/numOBSSPDValues;

% Create ticks for X-axis
xticks(1:tickInterval:numOBSSPDValues*tickInterval)

% Add created tick labels to axis
figAxis.XTickLabel = OBSSPDValues;

% Add title
title('Impact of OBSS PD threshold on the throughput of each BSS')
```
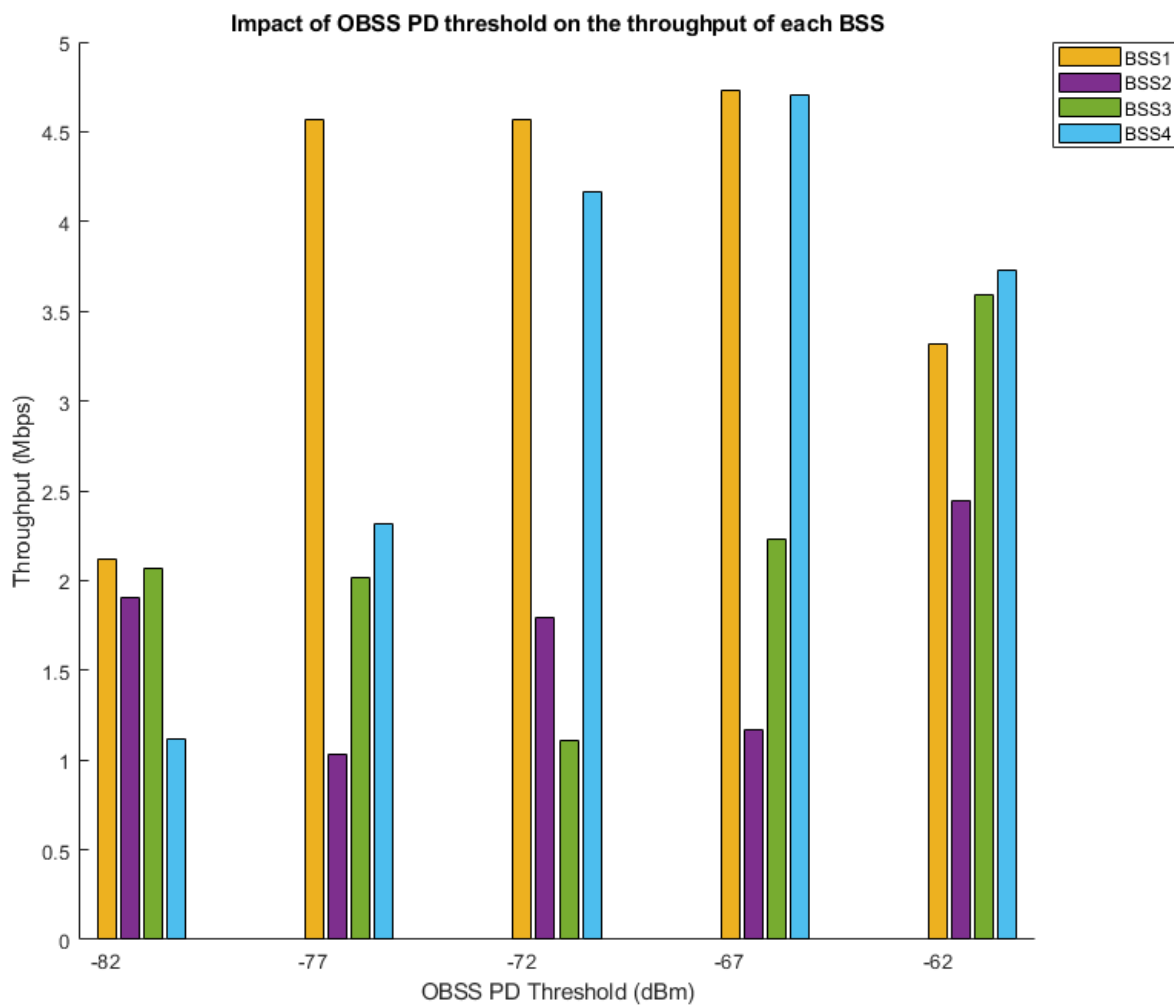

Observe the increase in the throughput with SR as compared to no SR (OBSS PD Threshold -82 dBm). As OBSS PD threshold increases, PHY discards more frames that are detected as inter-BSS, enabling the STAs to transmit the frames concurrently. Consequently, you can also see that the throughput increases with OBSS PD threshold.

Running the simulation using the configuration parameters specified in the preceding table:

- For BSS1 & BSS4, maximum throughput is achieved when OBSS PD threshold is set to -67 dBm.
- For BSS2 & BSS3, maximum throughput is achieved when OBSS PD threshold is set to -62 dBm.

You can simulate the model for multiple runs to visualize the true impact of OBSS PD threshold values on the network throughput.

This example enables you to model a residential scenario in a multinode IEEE 802.11ax network to study SR with BSS coloring. The spatial reuse with BSS coloring support is added to the PHY and MAC library blocks.The model uses the constant OBSS PD algorithm to select an OBSS PD threshold. The simulation results validate that the throughput performance and channel efficiency increases by using SR with BSS coloring.

Further Exploration

This example uses the constant OBSS PD algorithm. This algorithm decides whether the incoming OBSS frame must be allowed for further processing or discarded based on a configured OBSS PD value. You can modify the OBSS PD algorithm by updating the constantOBSSPDAlgorithm helper function used by the `EDCA MAC` block present in the MAC layer.

In this example, multi-BSS configuration is possible only when SR is enabled. To compare the results of a multi-BSS scenario with and without SR operation, configure the same value for `OBSS PD Threshold` and `ED Threshold` parameters to simulate non-SR operation without actually disabling the `Enable Spatial Reuse with BSS Color` parameter.

Appendix

The example uses these helpers:

1. edcaAssignBSSIDs.m: Assign BSS IDs
2. edcagetBSSInfo.m: Return BSS information
3. edcaValidateBSS.m: Validate the BSS
4. hDisplayNetworkStats.m: Display per BSS aggregated throughput
5. constantOBSSPDAlgorithm.m: Create a constant OBSS PD algorithm object.
6. obssPDAlgorithm.m: Create a OBSS PD algorithm object.
7. edcaFrameFormats.m: Create an enumeration for PHY frame formats.
8. edcaNodeInfo.m: Return MAC address of a node.
9. edcaPlotQueueLengths.m: Plot MAC queue lengths in the simulation.
10. edcaPlotStats.m: Plot MAC state transitions with respect to simulation times.
11. edcaStats.m: Create an enumeration for simulation statistics.
12. edcaUpdateStats.m: Update simulation statistics.
13. helperSubframeBoundaries.m : Return subframe boundaries of an A-MPDU.
14. phyTxAbstracted: Model PHY operations related to packet transmission
15. phyRxAbstracted: Model PHY operations related to packet reception
16. channelBlock: Model the channel for a node
17. addMUPadding.m: Add or remove the padding difference between an HE-SU and HE-MU PSDU
18. macQueueManagement.m: Create a WLAN MAC queue management object

19 roundRobinScheduler.m: Create round-robin scheduler object

20 calculateSubframesCount.m: Calculate the number of subframes required to form MU-PSDU

21 hCreateWLANNetworkModel: Create a WLAN network with given number of nodes

22 hSetupAbstractChannel: TGax channel setup

23 rateAdaptationARF.m: Create an auto rate fallback (ARF) algorithm object.

24 rateAdaptationMinstrelNonHT.m: Create a minstrel algorithm object.

References

1 IEEE P802.11ax™/D4.1. "Amendment 6: Enhancements for High Efficiency WLAN" Draft Standard for Information technology - Telecommunications and information exchange between systems Local and metropolitan area networks - Specific requirements -Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

2 IEEE Std 802.11ac™ -2016. "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 Wilhelmi, Francesc, Sergio Barrachina Munoz, Cristina Cano, Ioannis Selinis, and Boris Bellalta. "Spatial Reuse in IEEE 802.11ax WLANs." ArXiv:1907.04141 [Cs], November 29, 2019.

4 Wilhelmi, Francesc, Sergio Barrachina-Munoz, and Boris Bellalta. "On the Performance of the Spatial Reuse Operation in IEEE 802.11ax WLANs." In 2019 *IEEE Conference on Standards for Communications and Networking (CSCN)* , 1-6, 2019.

802.11 MAC and Application Throughput Measurement

This example shows how to measure the MAC and application layer throughput in a multi-node 802.11a/n/ac/ax network using SimEvents®, Stateflow®, and WLAN Toolbox™. The system-level model presented in this example includes functionalities such as configuring the priority of the traffic at the application layer, capability to generate and decode waveforms of Non-HT, HT-MF, VHT, HE-SU and HE-EXT-SU formats, MPDU aggregation and enabling block acknowledgment of MPDUs. The application layer throughput calculated using this model is validated against published calibration results from the TGax Task Group [4] for Box 3 scenarios (Tests 1a, 1b, and 2a) specified in TGax evaluation methodology [3]. The obtained application layer throughput is within the range of minimum and maximum throughput specified in published calibration results [4].

Throughput in 802.11 Networks

The IEEE® 802.11™ working group is continually adding features to 802.11 specification [1] to improve the throughput and reliability in WLAN networks. Throughput is the amount of data transmitted over a period of time. Medium Access Control (MAC) layer throughput refers to the amount of data successfully transmitted by the MAC layer over a period of time. MAC protocol data unit (MPDU) is the unit of transmission at MAC layer. In 802.11n, MPDU aggregation was introduced to increase the throughput. When MPDU aggregation is supported, MAC layer aggregates multiple MPDUs into an aggregated MPDU (A-MPDU) for transmission. This reduces the overhead of channel contention for transmitting multiple frames, resulting in enhanced throughput. In 802.11ac [1] and 802.11ax [2], the maximum limits for an A-MPDU length were increased resulting in even better throughput in WLAN networks.

Model 802.11 Network

This example models a WLAN network with five nodes as shown in this figure. These nodes implement carrier-sense multiple access with collision avoidance (CSMA/CA) with physical carrier sense and virtual carrier sense. The physical carrier sensing uses the clear channel assessment (CCA) mechanism to determine whether the medium is busy before transmitting. Whereas, the virtual carrier sensing uses the RTS/CTS handshake to prevent the hidden node problem.

The model in the example displays various statistics such as the number of transmitted, received, and dropped packets at PHY and MAC layers. Moreover, the runtime figures that help in analyzing/ estimating the node-level and network-level performance are also displayed in this model. This model is validated against the published calibration results from the TGax Task Group [4] for Box 3 scenarios (Tests 1a, 1b, and 2a) specified in TGax evaluation methodology [3].

WLAN Network

Copyright 2019 The MathWorks, Inc.
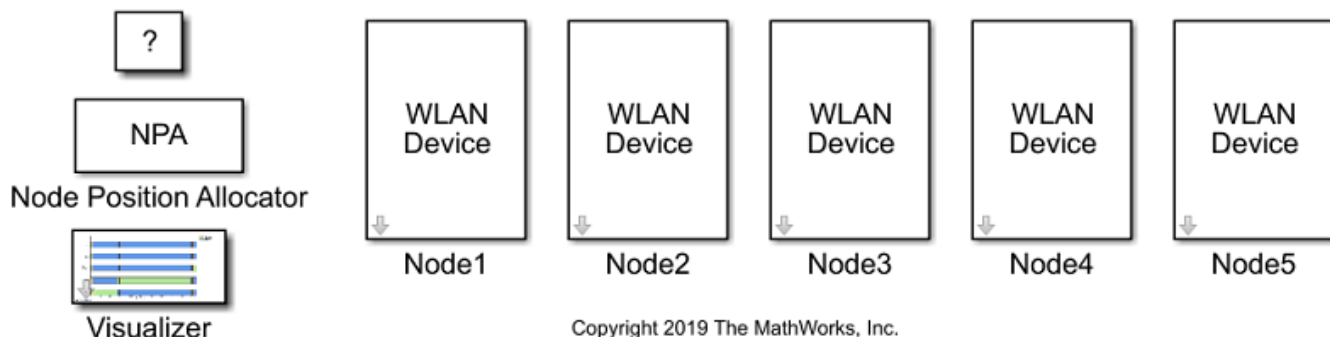
Components of a WLAN Node

The components of a WLAN node are shown in this figure. The information is retrieved by pressing the arrow button for each node in the above figure.
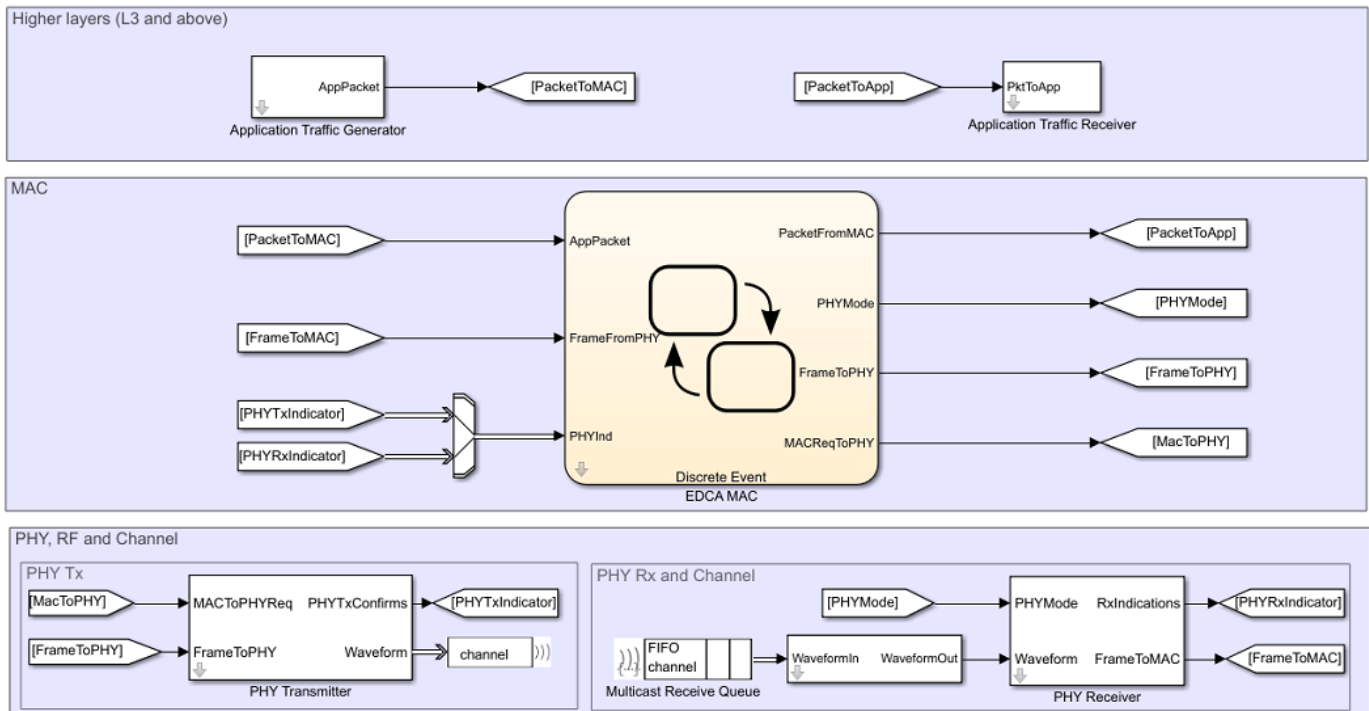
Application, EDCA MAC and PHY Block Enhancements

This example is an enhancement over "Multi-Node 802.11a Network Modeling with PHY and MAC" on page 7-63 example. Refer to the example documentation page for more information about each layer in a WLAN node. The application, the EDCA MAC and the PHY blocks used in this example has these enhancements over "Multi-Node 802.11a Network Modeling with PHY and MAC" on page 7-63.

Application:

The application layer has the capability to generate data with different priority levels as shown in this figure. These priority levels are configured using `Access Category` property in the mask parameters of the Application Traffic Generator block inside a WLAN node.
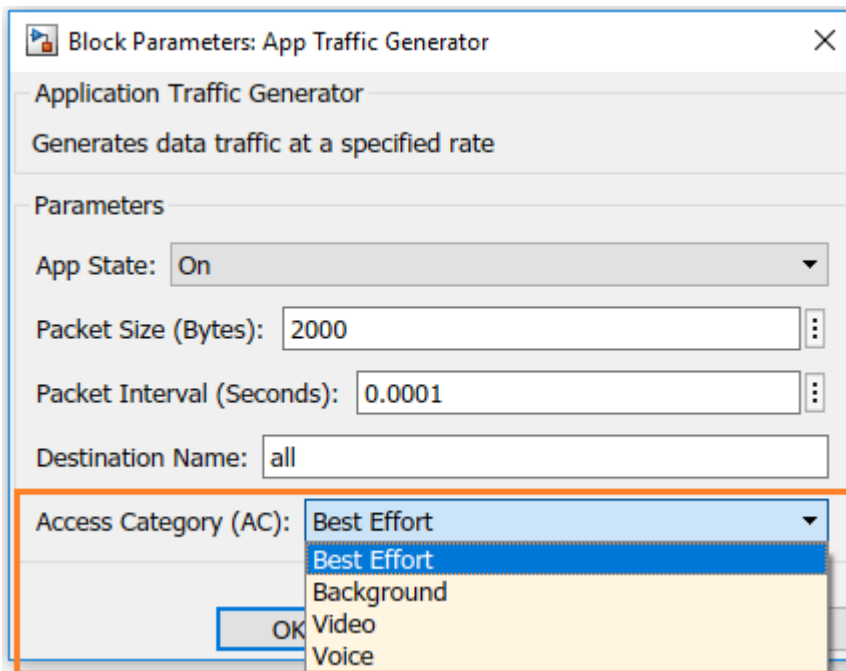
EDCA MAC:

The EDCA MAC block used in this example has these enhancements over MAC block used in "Multi-Node 802.11a Network Modeling with PHY and MAC" on page 7-63 example

- Generate and decode MAC frames of high efficiency single user (HE-SU), high efficiency extended range single user (HE-EXT-SU), very high throughput (VHT), high throughput mixed format (HT-MF) and Non-HT formats. These formats are configured using the `PHY Tx Format` property in the mask parameters of the MAC EDCA block inside a WLAN node as shown in this figure.

- Aggregate MPDUs to form an A-MPDU. This can be configured by setting `PHY Tx Format` to one of `HT-MF`, `VHT`, `HE-SU`, or `HE-EXT-SU`. In case of `HT-MF`, `MPDU Aggregation` property must also be enabled for A-MPDU generation.

- Acknowledge multiple MPDUs in an A-MPDU with a single block acknowledgment (BA) frame. MAC assumes a pre-configured BA session between the transmitter and the receiver of an A-MPDU.

- Enable/disable acknowledgments. This can be configured using the `Ack Policy` property.

- Maintain separate retry limits for shorter frames (less than RTS threshold) and longer frames (greater than or equal to RTS threshold). These limits can be configured using the `Max Short Retries` and `Max Long Retries` properties.

- Transmit multiple streams of data using the multiple-input multiple-output (MIMO) capability. You can configure this capability using the `Number of Transmit Chains` property. This property is applicable only when the value of `PHY Tx Format` property is set to VHT, `HE-SU`, or `HE-EXT-SU`. The MIMO capability can also be used for `HT` format through the `MCS` property. The range of values [0, 7], [8, 15], [16, 23], and [24, 31] correspond to one, two, three, and four streams of data respectively.

- Adapt the data rate according to the channel conditions through the `Rate Adaptation Algorithm` property. This is applicable only when the value of `PHY Tx Format` property is set to `Non-HT`. You can choose between `Auto Rate Fallback (ARF)` and `Minstrel` algorithms. To maintain a constant data rate throughout the simulation, `Fixed-Rate` option is available.

- Enable parallel transmissions between the basic service sets (BSSs) through the `Enable Spatial Reuse with BSS Color` property. This property is applicable only when `PHY Tx Format` property is set to `HE-SU`, `HE-EXT-SU`, or `HE-MU-OFDMA`. This model does not support the spatial reuse (SR) functionality. To study the impact of SR with BSS coloring on the network throughput, refer "Spatial Reuse with BSS Coloring in 802.11ax Residential Scenario" on page 7-12 example.

PHY:

Capability to generate and decode waveforms of Non-HT, HT-MF, VHT, HE-SU and HE-EXT-SU formats

Throughput Measurement

Throughput varies for different configuration parameters pertaining to the application, MAC & PHY layers. Any change in the configuration may either increase or decrease the throughput. You can vary the combination of these parameters to measure and analyze the throughput.

- MCS: PHY data rate
- PHY Tx Format: PHY transmission format
- Packet Size: Application packet size
- Max A-MPDU Subframes: Maximum number of subframes in an A-MPDU
- Max Tx Queue Size: MAC transmission queue size

Along with above parameters, you can also vary the node positions, Tx & Rx gains, channel loss, number of nodes in the network, MAC contention parameters, number of transmit chains and rate adaptation algorithms to analyze MAC throughput. This example demonstrates the measurement and analysis of the MAC throughput by varying packet size in the Application Traffic Generator block.

Application Packet Size

Throughput is directly proportional to the application packet size. Smaller packet size results in greater number of packets to be transmitted. At the MAC layer, there is an overhead of contention time for each transmitted packet. This is because the MAC layer makes sure that the channel is idle for a specific amount of time (Refer section 10.3.2.3 of [1]) before transmitting any packet. Therefore, as the packet size decreases, the contention overhead increases resulting in lower throughput.

Model Configuration

You can configure the application packet size using these steps:

1 Open model WLANMACThroughputMeasurementModel.slx
2 To go inside a node subsystem, click on the downward arrow at the bottom left of the node
3 To open mask parameters of the application, double click on Application Traffic Generator
4 To enable application, set App State to 'On'
5 Configure the value of Packet Size

Run the simulation and observe the throughput. The TGax calibration results for test-1a in [4] are shown below:
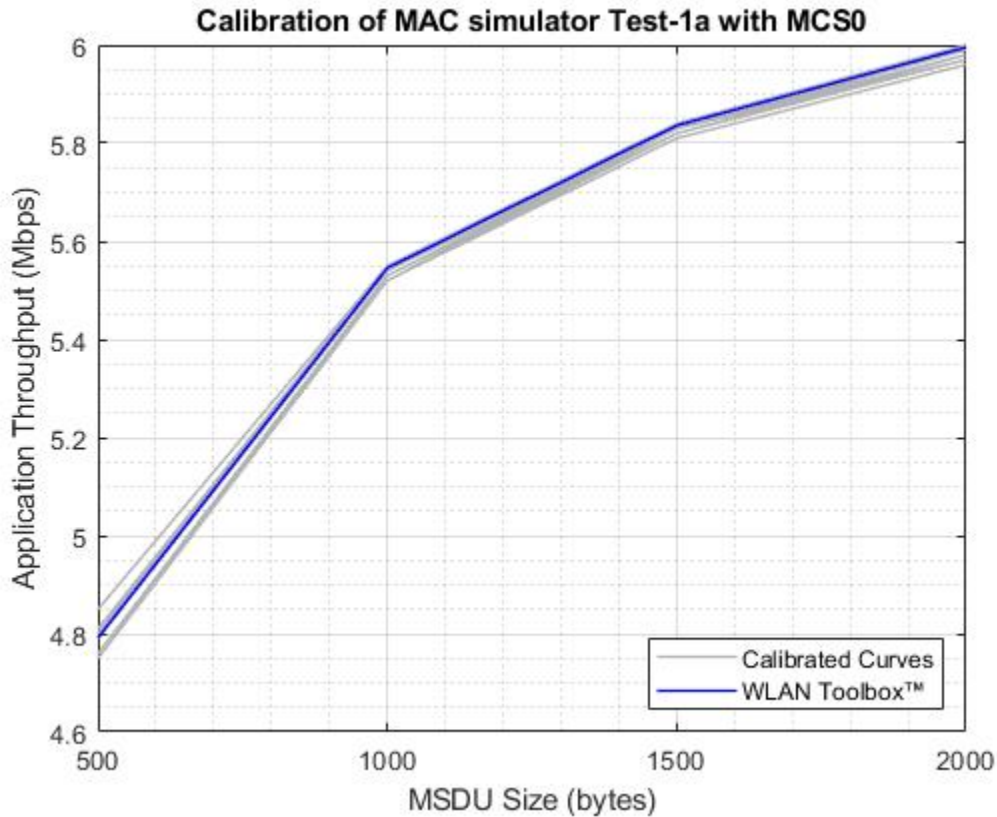
The above plot compares the calibration results for WLAN Toolbox against the published results of other companies listed in [4]. The blue colored curve represents the results of WLAN Toolbox, while the grey colored curves represent the results of other companies.

Simulation Results

The simulation of the model generates:

1 A run-time visualization showing the time spent on channel contention, transmission, and reception for each node

2 An optional run-time visualization (during the simulation) showing the number of frames queued in MAC transmission queues for a selected node.

3 A bar graph showing metrics for each node such as number of transmitted, received, and dropped packets at PHY and MAC layers

4 A MAT file `statistics.mat` with detailed statistics obtained at each layer for each node

This figure shows MAC state transitions with respect to simulation time.
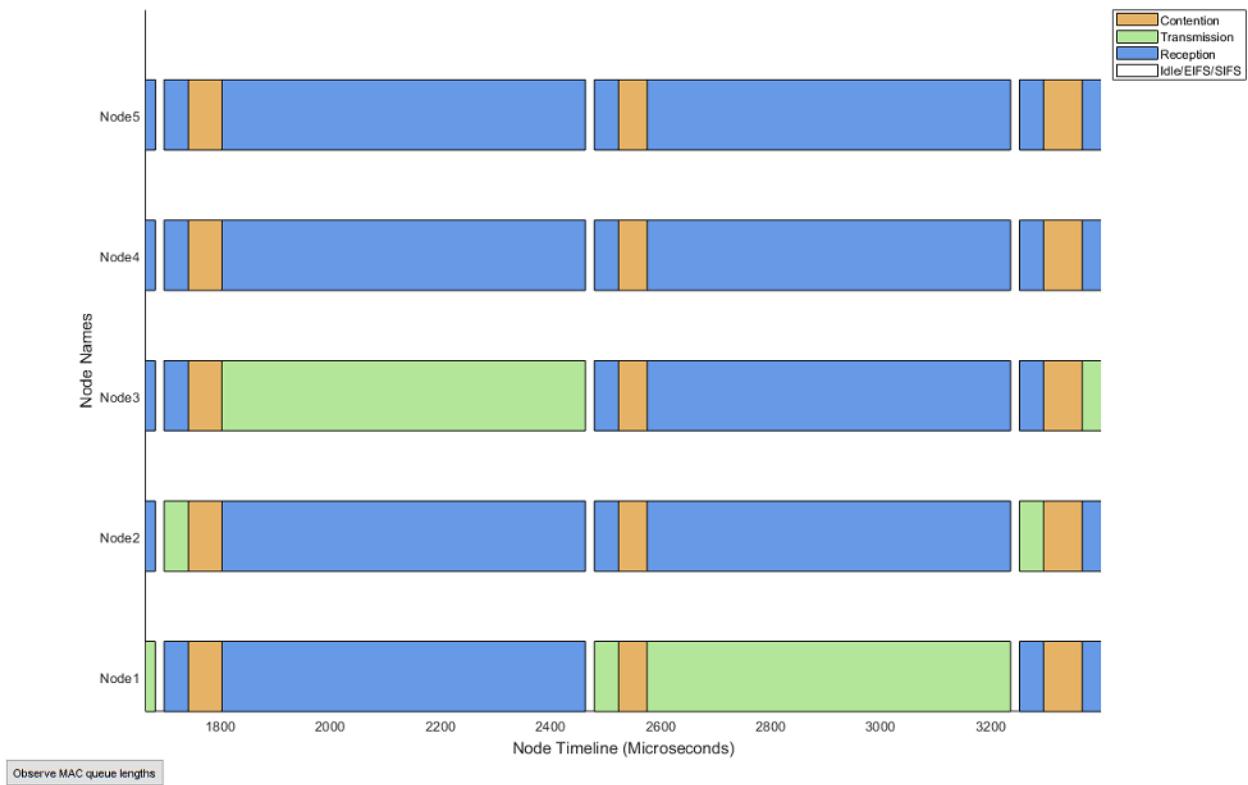
You can also observe the live state of the MAC layer transmission buffers using the 'Observe MAC queue lengths' button in the above visualization.
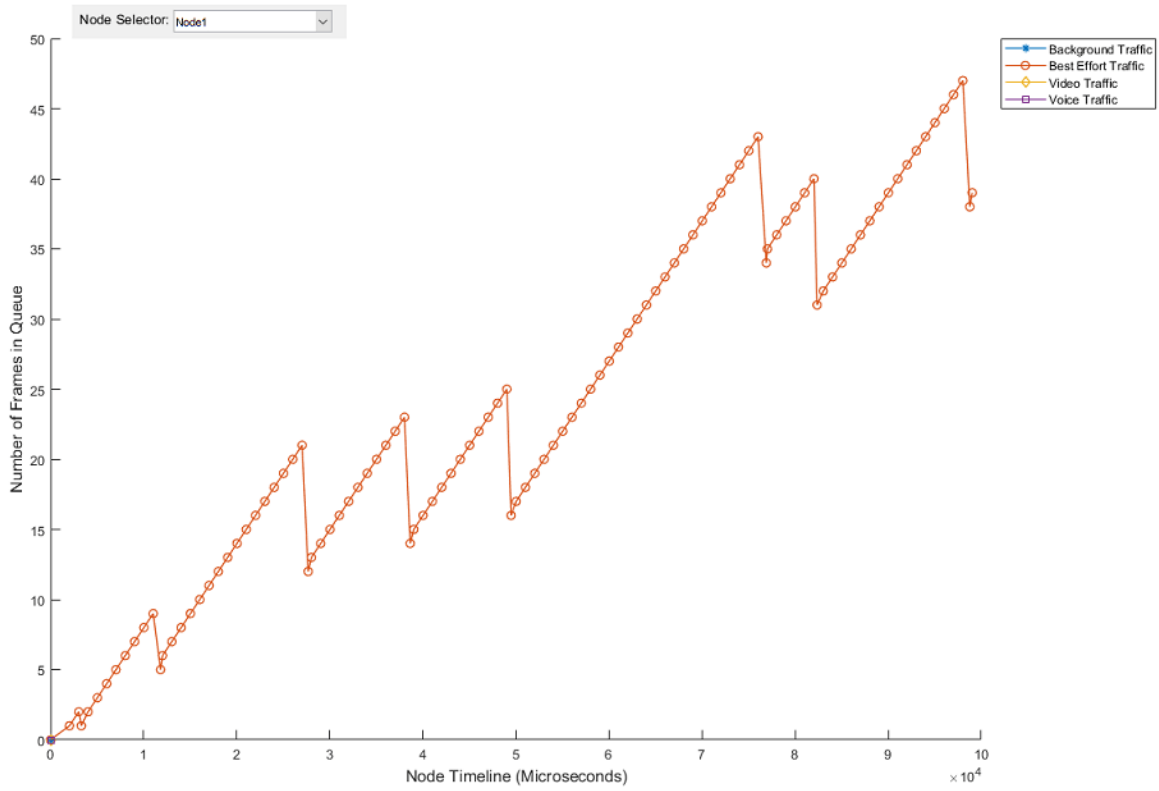
This figure shows the network statistics at the end of simulation.
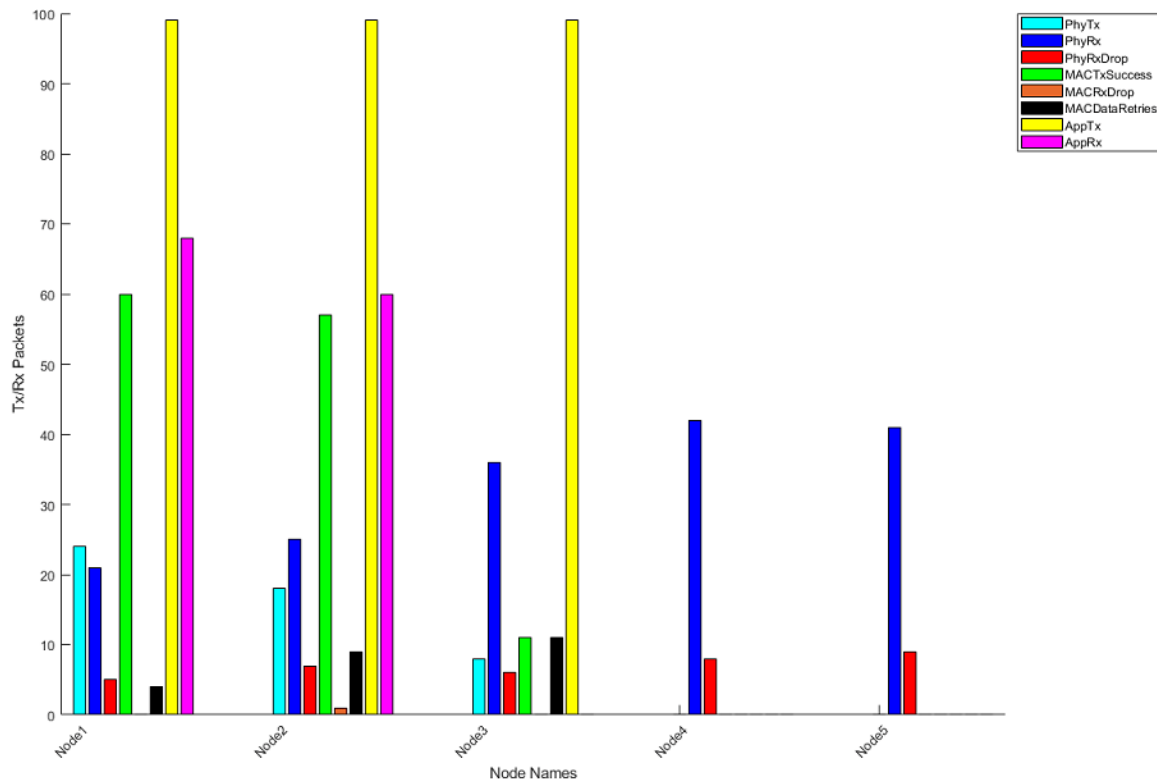
Validating Application Layer Throughput with TGax Calibration Results

The TGax Task Group [4] published application throughput results for different scenarios. You can observe the Layer 3 (above MAC layer) throughput of each node in the network in 'Throughput' column in 'statisticsTable' stored in 'statistics.mat'. The TGax calibration scenarios for MAC simulator published results of application throughput for a User Datagram Protocol (UDP) with Logical Link Control (LLC) layers overhead.

To calculate application throughput from simulation results use the code below:

```
% Load statistics.mat (Output of the simulation) file
simulationResults = load('statistics', 'statisticsTable');
% Statistics
stats = simulationResults.statisticsTable;

% Successfully transmitted MAC layer bytes in the network
totalMACTxBytes = sum(stats.MACTxBytes);

% UDP & LLC overheads (bytes)
udpOverhead = 36;
llcOverhead = 8;

% UDP & LLC overhead (bytes) in the network
udpAndLLCOverhead = sum(stats.MACTxSuccess)*(udpOverhead + llcOverhead);

% Successfully transmitted application bytes
```

```
totalAppTxBytes = totalMACTxBytes - udpAndLLCOverhead;

% Time at which last transmission is completed in the network (Microseconds)
simulationTime = max(stats.MACRecentFrameStatusTimestamp);

% Application throughput (Mbps)
applicationThroughput = (totalAppTxBytes*8)/simulationTime;
disp(['Application Throughput = ' num2str(applicationThroughput) ' Mbps']);

Application Throughput = 4.7276 Mbps
```

The application throughput for different TGax calibration scenarios is plotted against different MAC service data unit (MSDU) sizes for a simulation time of 30 seconds as shown below:

Calibration of MAC simulator Test-1b with MCS0

Calibration of MAC simulator Test-1b with MCS8

Calibration of MAC simulator Test-2a Without RTS

Calibration of MAC simulator Test-2a With RTS
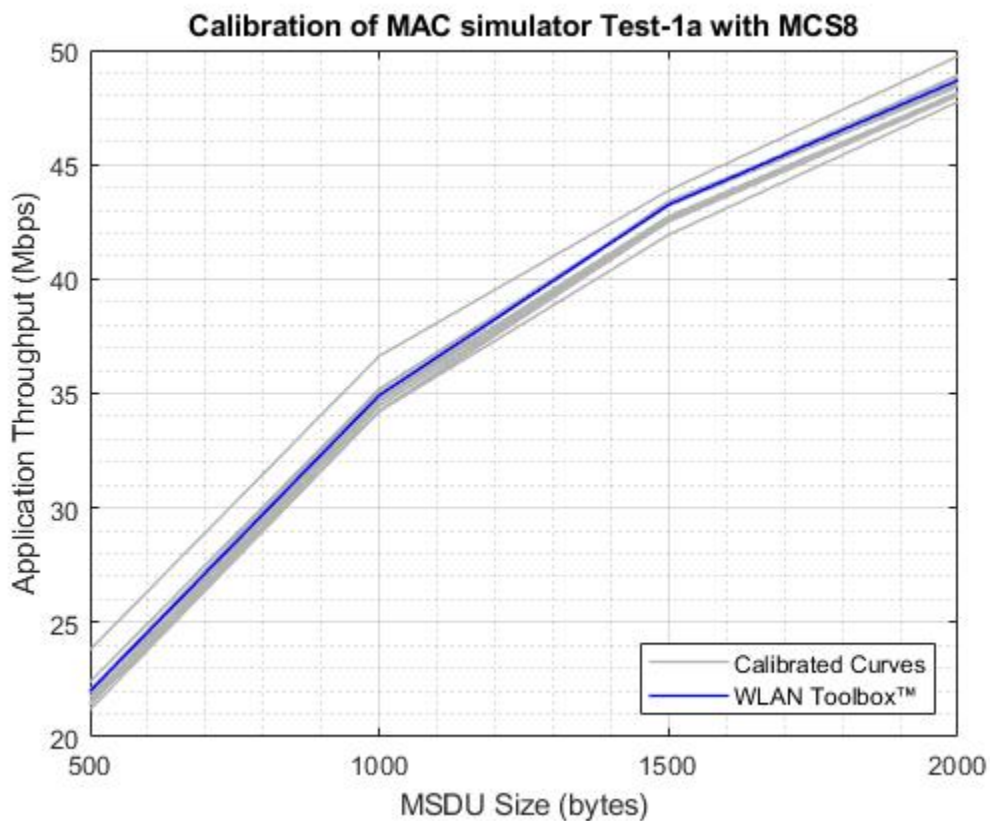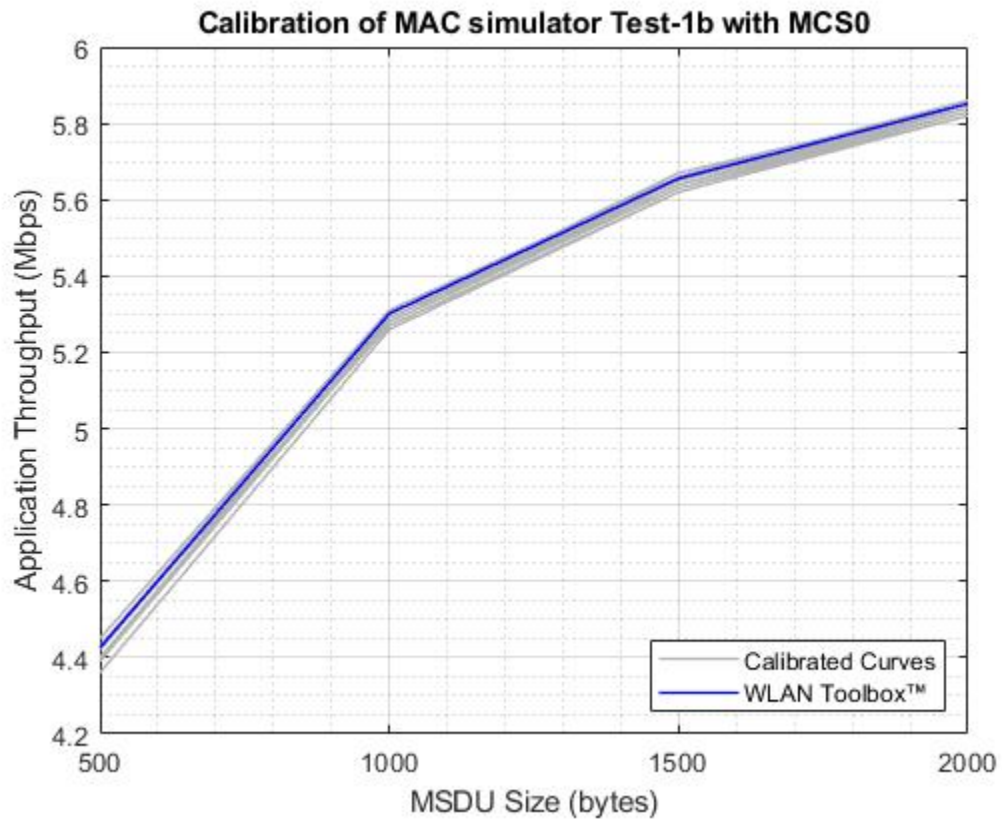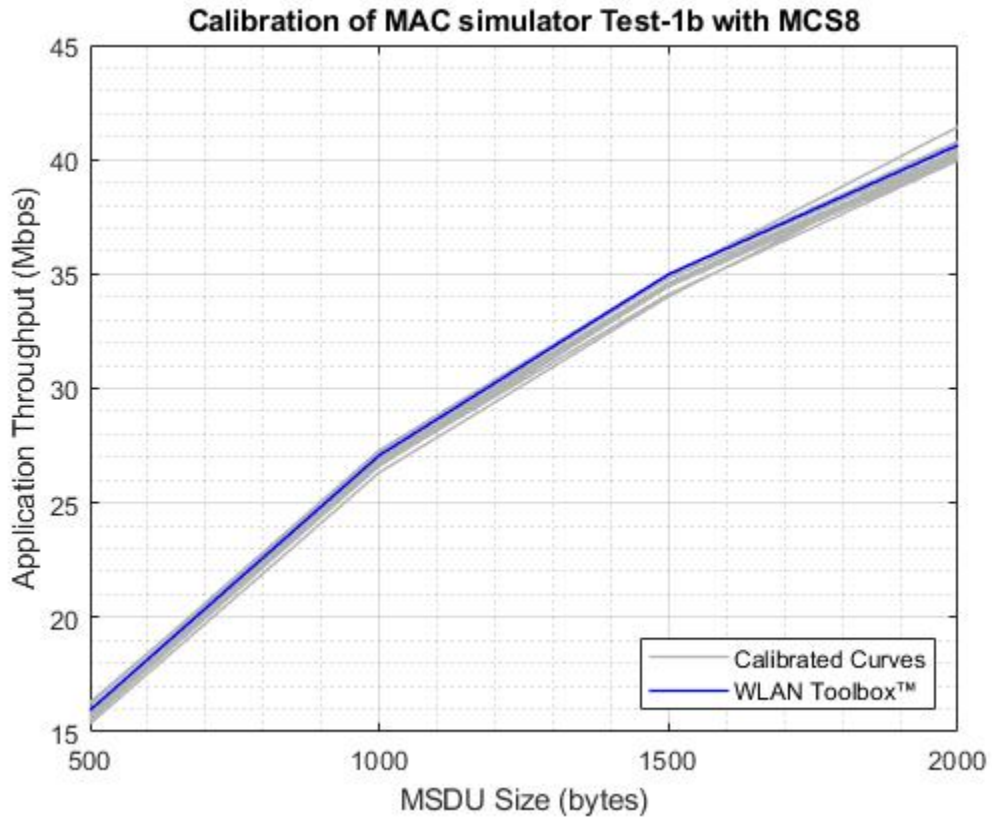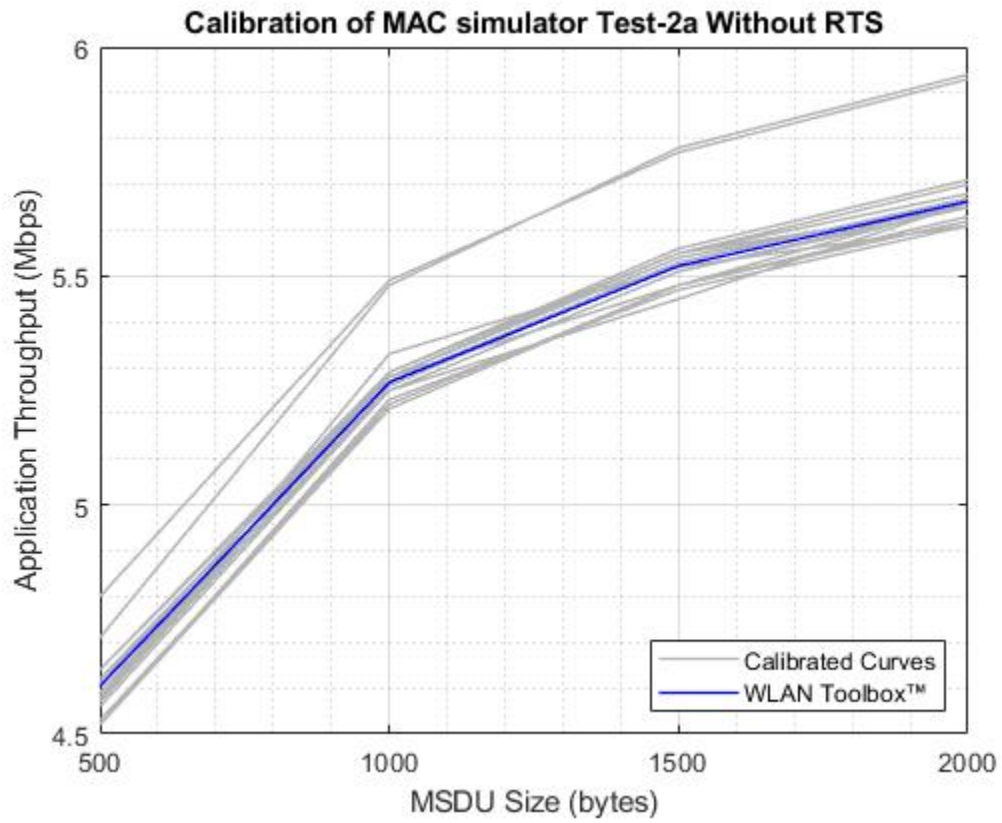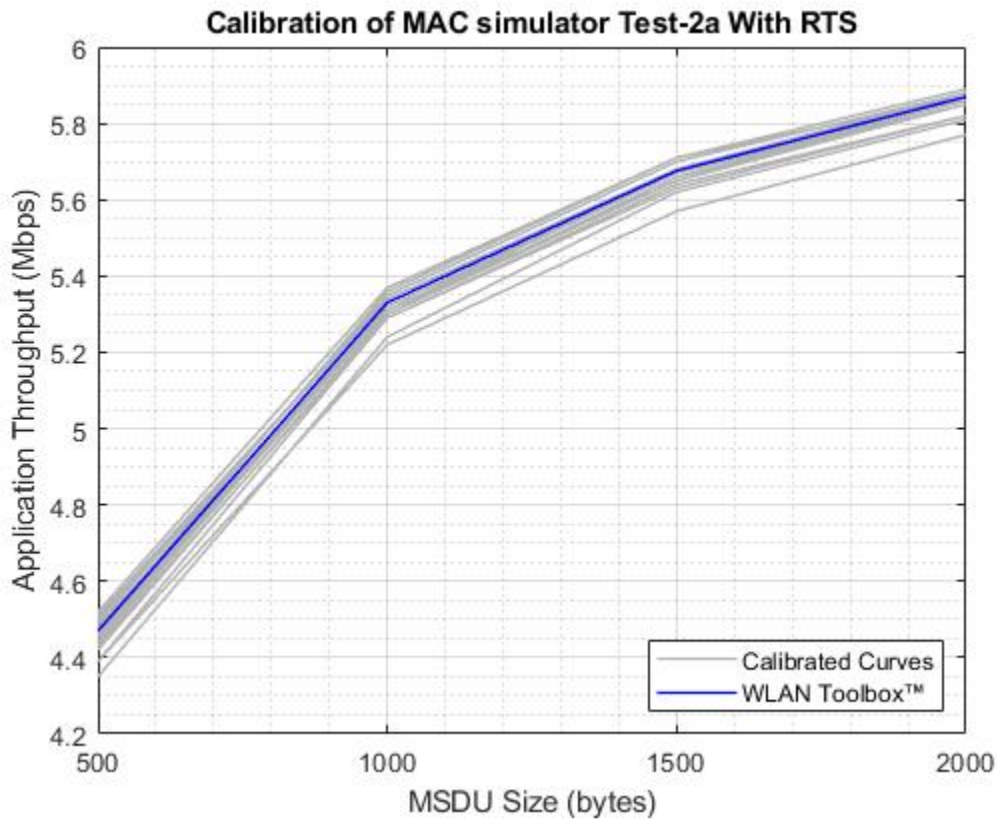
Further Exploration

Configuration options

You can change these configuration parameters to further explore this example:

- Application layer: Access category and packet interval
- MAC layer: RTS threshold, Tx queue size, data rate, short retry limit, long retry limit, transmitting frame format, MPDU aggregation, ack policy, number of transmit chains and the rate adaptation algorithms
- PHY: PHY Tx gain, PHY Rx gain, and Rx noise figure
- Channel modeling: Rayleigh fading, free space pathloss, range propagation loss and packet receive range
- Node positions using node position allocator
- The state of each node can be visualized during the run-time through the configuration available in the Visualizer block
- By default, the PHY transmitter and the receiver blocks run in the `Interpreted execution` mode. For longer simulation time, configure all the blocks to `Code generation` mode for better performance.

Related examples

Refer these examples for further exploration:

- To simulate the MAC Quality of Service (QoS) traffic scheduling in 802.11a/n/ac/ax networks, refer "802.11 MAC QoS Traffic Scheduling" on page 7-43 example.

- To model a multi-node IEEE 802.11ax network with abstracted PHY, refer "802.11ax System-Level Simulation with Physical Layer Abstraction" on page 7-52 example.

- To model a multi-node network using Distributed Coordination Function (DCF) MAC and 802.11a PHY, refer "Multi-Node 802.11a Network Modeling with PHY and MAC" on page 7-63 example.

This example enables you to create and configure a multi-node 802.11 network using a Simulink model for analyzing the MAC and application layer throughput. In this model, the MAC throughput obtained through the simulation results is used to calculate the application layer throughput. This model is validated using the Box 3 scenarios (Tests 1a, 1b, and 2a) specified in TGax evaluation methodology [3] to confirm that it complies with the IEEE 802.11 [1]. This example concludes that the calculated application layer throughput is within the range of minimum and maximum throughput specified in published calibration results [4].

Appendix

The helper functions and objects used in this example are:

1 edcaFrameFormats.m: Create an enumeration for PHY frame formats.
2 edcaNodeInfo.m: Return MAC address of a node.
3 edcaPlotQueueLengths.m: Plot MAC queue lengths in the simulation.
4 edcaPlotStats.m: Plot MAC state transitions with respect to simulation times.
5 edcaStats.m: Create an enumeration for simulation statistics.
6 edcaUpdateStats.m: Update statistics of the simulation.
7 helperAggregateMPDUs.m: Generate an A-MPDU, by creating and appending the MPDUs containing the MSDUs in the MSDULIST.
8 helperSubframeBoundaries.m: Return subframes information of an A-MPDU.
9 phyRx.m: Model PHY operations related to packet reception.
10 phyTx.m: Model PHY operations related to packet transmission.
11 edcaApplyFading.m: Apply Rayleigh fading effect on the waveform.
12 heSIGBUserFieldDecode.m: Decode HE-SIG-B user field.
13 heCPECorrection.m: Estimate and correct common phase error.
14 heSIGBCommonFieldDecode.m: Decode HE-SIG-B common field.
15 heSIGBMergeSubchannels.m: Merge 20MHz HE-SIG-B subchannels.
16 addMUPadding.m: Add multiuser PSDU padding.
17 macQueueManagement.m: Create a WLAN MAC queue management object.
18 roundRobinScheduler.m: Create a round-robin scheduler object.
19 calculateSubframesCount.m: Return number of subframes to be aggregated.
20 interpretVHTSIGABitsFailCheck.m: Interprets the bits in VHT-SIG-A field
21 rateAdaptationARF.m: Create an auto rate fallback (ARF) algorithm object.
22 rateAdaptationMinstrelNonHT.m: Create a minstrel algorithm object.

References

1   IEEE Std 802.11™. "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. "Amendment 6: Enhancements for High Efficiency WLAN.." Draft 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 802.11-14/0571r12. "11ax Evaluation Methodology." IEEE P802.11P: Wireless LANs.

4   Baron, Stephane., Nezou, Patrice., Guignard, Romain., and Viger, Pascal. "MAC Calibration Results." Presentation at the IEEE P802.11 - Task Group AX, September 2015.

802.11 MAC QoS Traffic Scheduling

This example shows how to simulate medium access control (MAC) quality of service (QoS) traffic scheduling in 802.11a/n/ac/ax network using SimEvents®, Stateflow®, and WLAN Toolbox™. Using QoS, the MAC layer prioritizes the application data and allocates more resources to high priority traffic compared to low priority traffic. The MAC layer used in this example implements the enhanced distributed channel access (EDCA) algorithm to get access to the channel for transmission. This example models a 802.11 network with five nodes that can be simultaneously configured with four types of application traffic namely: Best-effort, Background, Video and Voice. The type and the priority of application traffic is identified by the access category (AC) at the MAC layer. The derived results show that the highest priority voice traffic incurs the lowest latency, whereas the lowest priority background traffic results in highest latency.

QoS Scheduling in 802.11 Network

The IEEE 802.11 WLAN [1] standard supports low rate applications and provides best-effort service at the MAC layer. The widespread use of real-time multimedia content along with deployment of time-sensitive critical applications has created a need to provide QoS support in 802.11 networks. To achieve this, the IEEE 802.11 TGe introduced QoS as a part of the IEEE 802.11e amendments to the IEEE 802.11 specification. By providing QoS support, the IEEE 802.11 MAC layer prioritizes the application data and allocates more resources to high priority traffic as compared to low priority traffic.

Simulate QoS Network

This example models a WLAN network with five nodes as shown in this figure. These nodes implement carrier-sense multiple access with collision avoidance (CSMA/CA) with physical carrier sense and virtual carrier sense. The physical carrier sensing uses the clear channel assessment (CCA) mechanism to determine whether the medium is busy before transmitting. Whereas, the virtual carrier sensing uses the RTS/CTS handshake to prevent the hidden node problem.

Copyright 2019 The MathWorks, Inc.
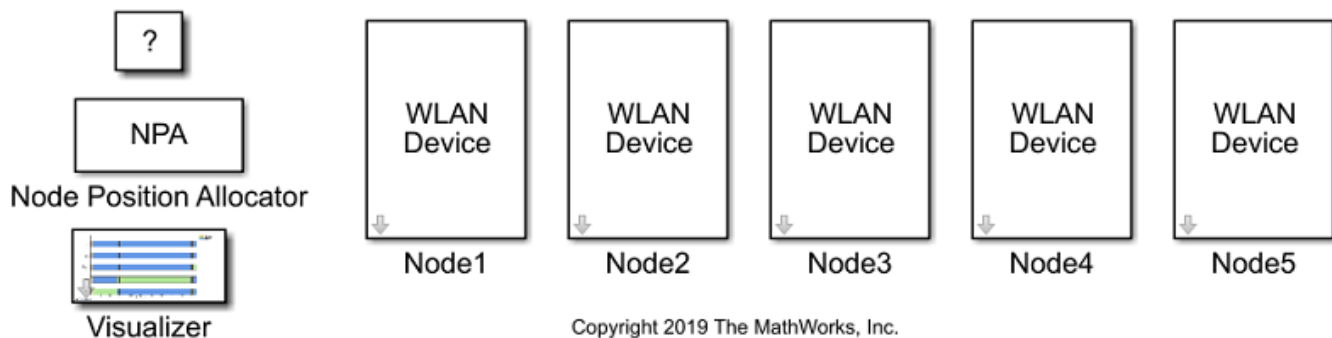
Each node can be simultaneously configured with these four types of application traffic:

- Best-effort (AC0)
- Background (AC1)
- Video (AC2)
- Voice (AC3)

The AC present at the MAC layer identifies the type and the priority of the application traffic.
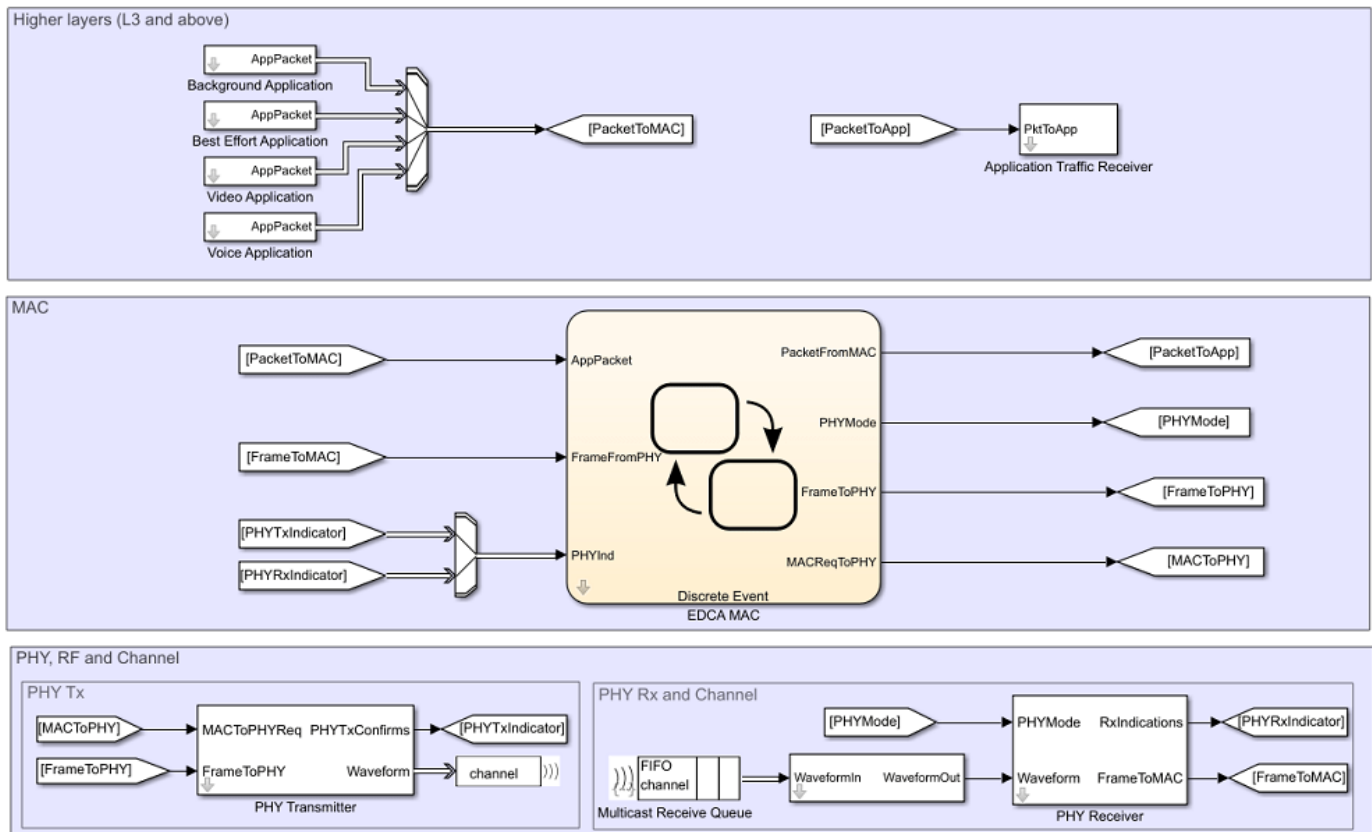
The MAC layer implements the EDCA algorithm to access the channel for transmission. The EDCA contention parameters are designed such that the high priority traffic gets more transmission opportunities than the low priority traffic. The default configuration assigns priorities to the application traffic in this order: Background, Best-effort, Video and Voice. However, even after using these contention parameters, there is a chance that more than one AC obtains transmission opportunity at the same time resulting in internal collision. If an internal collision occurs:

- The high priority AC gets the transmission opportunity
- The low priority AC is considered as a transmission failure and invokes a retransmission procedure

This example differs from the "802.11 MAC and Application Throughput Measurement" on page 7-27 example as multiple applications at the application layer to generate different types of traffic.

Model Configuration

You can configure the application packet size using these steps:

1. Open model `WLANMACQosSchedulingModel.slx`
2. To go inside a node subsystem, click on the downward arrow at the bottom left of the node
3. To open mask parameters of the application, double click on `Best Effort Application`
4. To enable application, set `App State` to 'On'
5. Configure the values of `Packet Size`, `Packet Interval`, `Destination Name` and `Access category`

6   Use steps 3, 4 and 5 to configure other applications
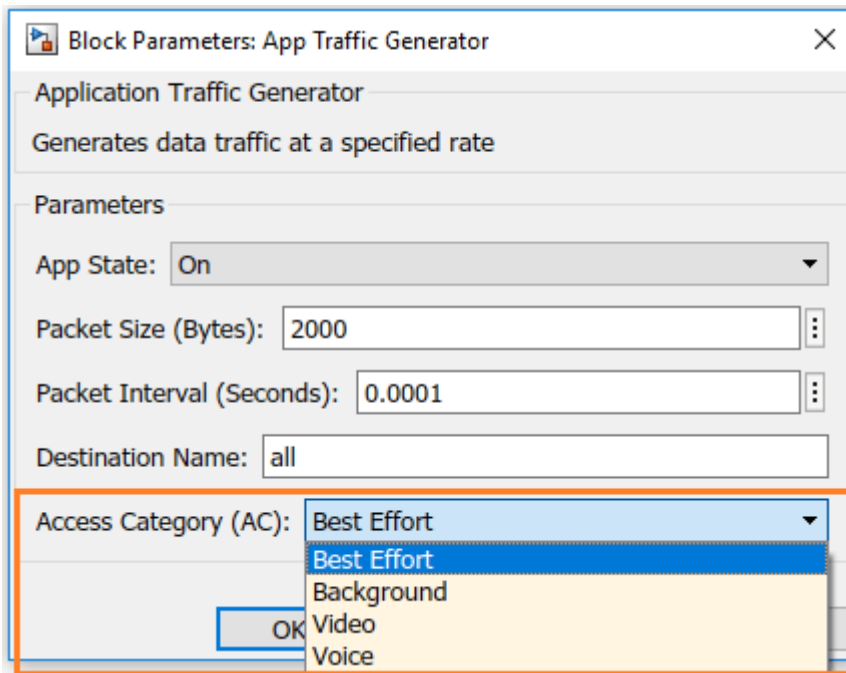
For more configuration options, refer "802.11 MAC and Application Throughput Measurement" on page 7-27. To add additional nodes in the network, copy and paste any existing node in the network.

Simulation Results

This example generates similar results that are obtained in the "802.11 MAC and Application Throughput Measurement" on page 7-27. These results include:

A run-time visualization showing the time spent on channel contention, transmission, and reception for each node. This plot shows MAC state transitions with respect to simulation time.
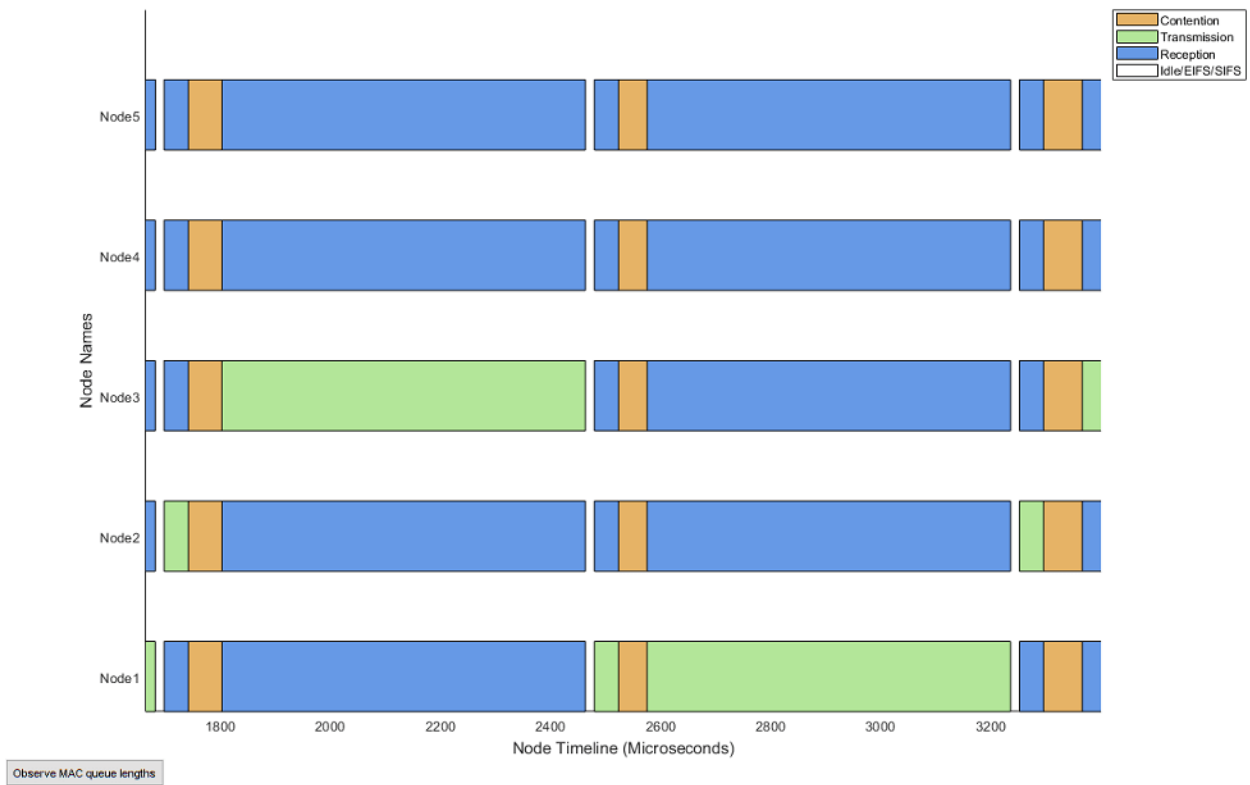
During the simulation, an optional run-time visualization plot displays the number of frames queued in MAC transmission queues for a selected node. You can observe the run-time state of the MAC layer transmission buffers using the 'Observe MAC queue lengths' button in the above visualization.
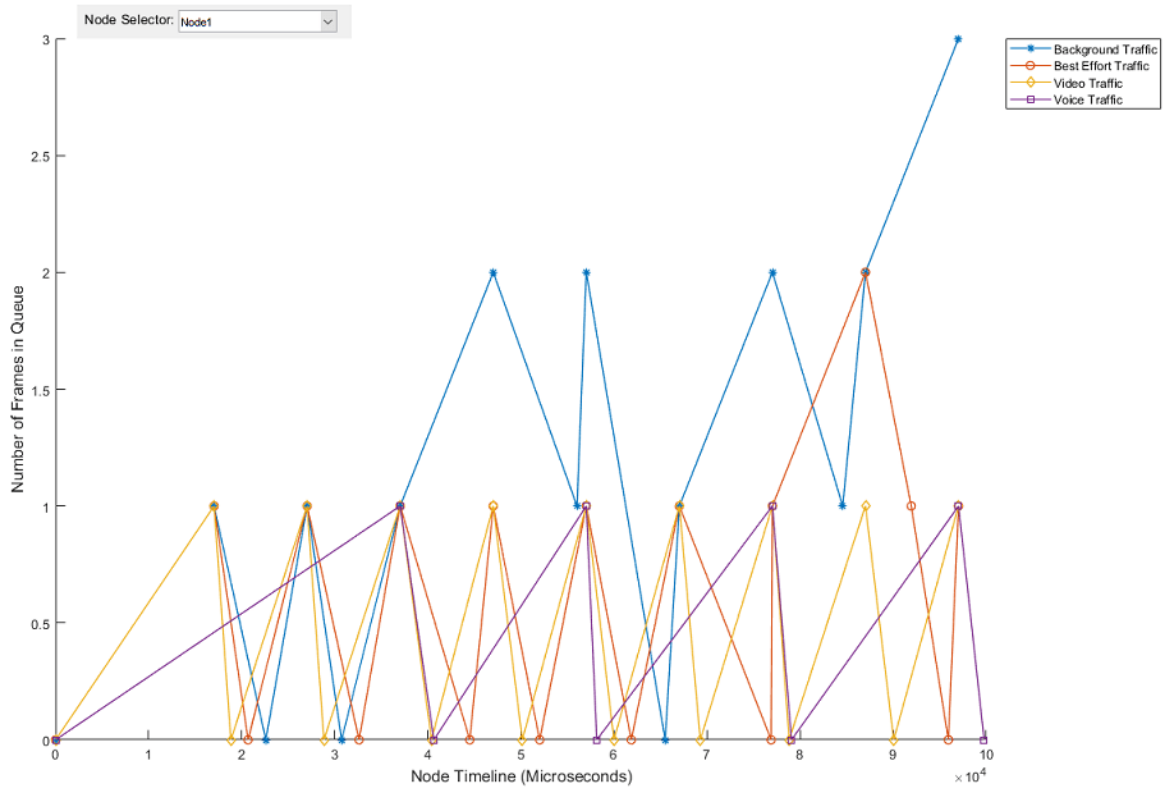
A bar graph showing metrics for each node such as number of transmitted, received, and dropped packets at PHY and MAC layers. This plot shows the network statistics at the end of simulation.
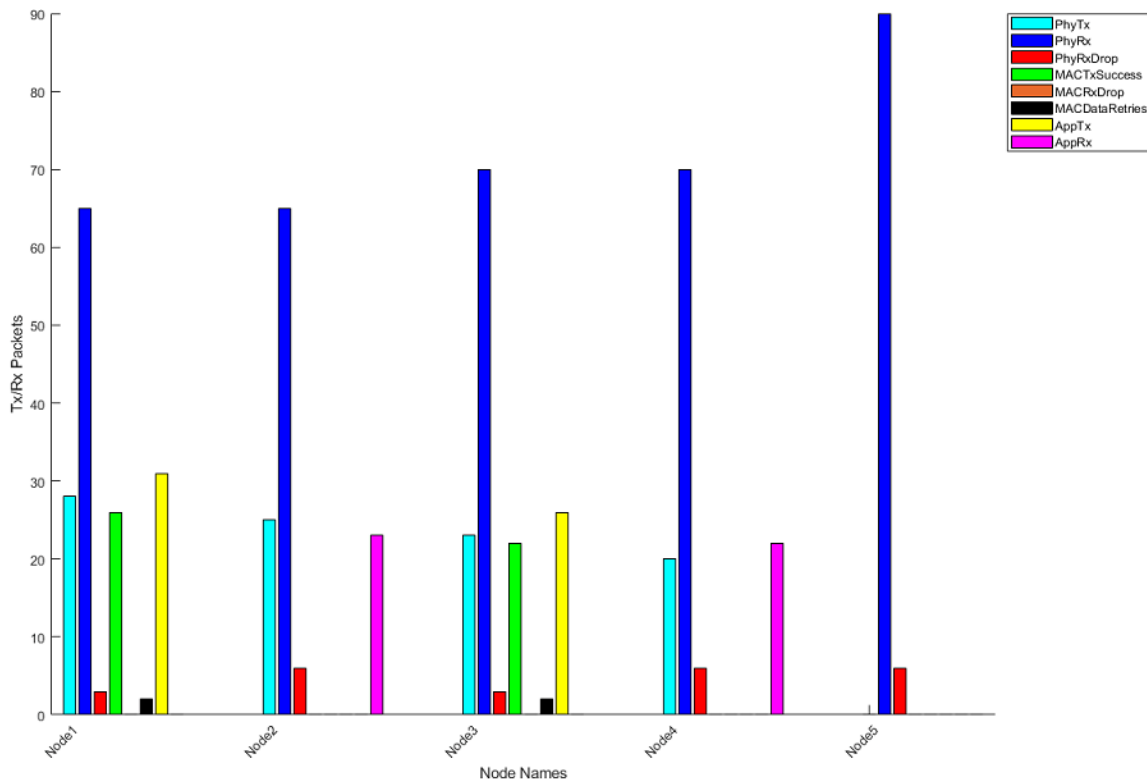
A MAT file 'statistics.mat' stores the detailed statistics for each layer and node in the network. Since this example has multiple applications, the MAC layer transmits data packets from different ACs. You can observe the number of transmissions from each AC in 'MACAC0DataTx', 'MACAC1DataTx', 'MACAC2DataTx' and 'MACAC3DataTx' columns of 'statisticsTable' stored in 'statistics.mat'. In a given simulation, the number of transmissions per AC is directly proportional to the priority of the AC.

In addition to the above results, this example shows MAC latencies of all ACs for the selected node in a plot and also creates a MAT file 'macLatenciesLog.mat'. This file contains information about each MAC frame such as:

- **Entry timestamp:** The time at which application packet has arrives at the MAC layer
- Node ID: The node ID of the transmitter
- AC: Access category of the packet
- Clear timestamp: Time at which packet is cleared from the MAC. A packet is cleared from the MAC after successful reception of an acknowledgment or after completing the maximum number of retransmission attempts.

This information is used to calculate MAC latencies per packet. Here, the MAC latency is the time delay between a packet arriving at the MAC from the application layer, and it being cleared from the MAC. A packet is cleared from MAC in these scenarios:

- Successful transmission: A receiver sends an acknowledgment to the sender after receiving a packet intended for it. If an acknowledgment is received by the sender, it is considered as a successful transmission.
- Packet discard: If an acknowledgment is not received by the sender, it retransmits the packet until the retransmission limit is reached. If an acknowledgment is not received even after completing the maximum number of retransmissions, the packet is discarded at the MAC layer.
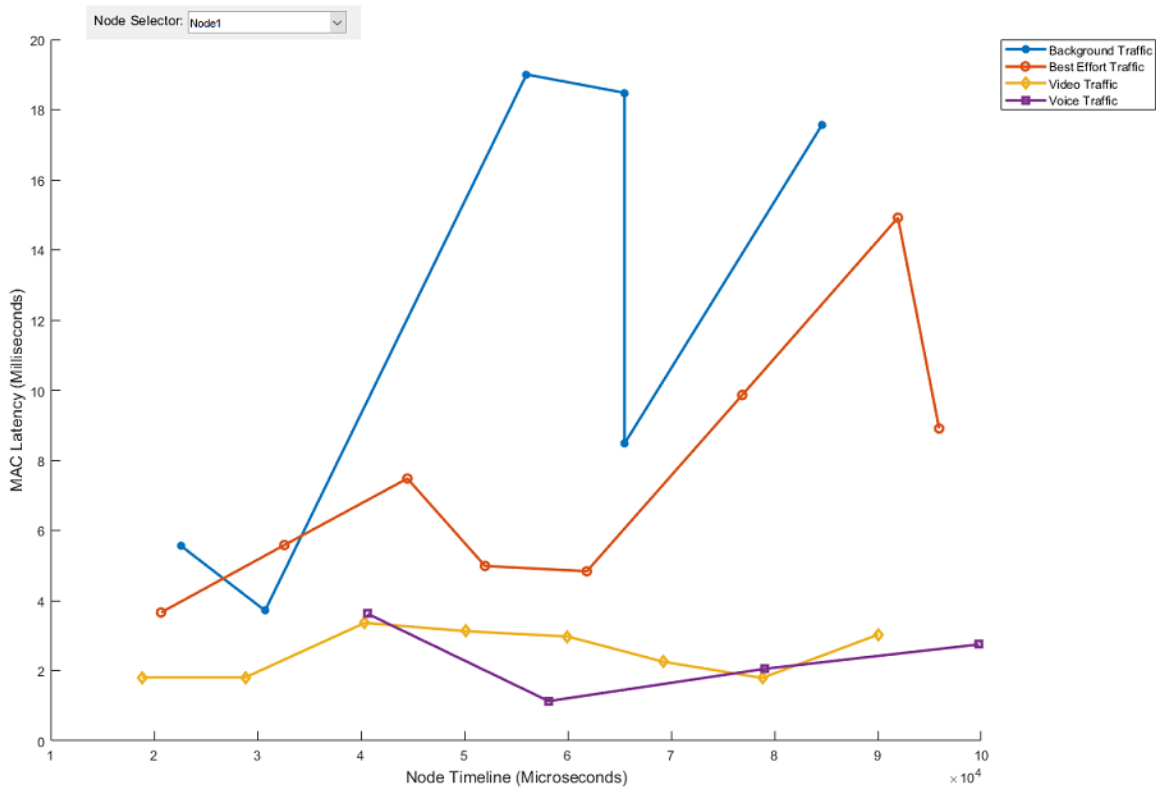
The voice traffic has the highest priority and hence incurs the lowest latency. Whereas, the lowest priority of background traffic results in highest latency.

Further Exploration

Configuration options:

In this model, you can configure the MAC contention parameters for each AC. The contention parameters for each AC are stored in 'macConfiguration.mat' as a structure containing three fields namely aifsn, cwMin and cwMax. You can change the priority of the AC (Best-effort/Background/Video/Voice) by changing these contention parameters. You can observe this by enabling multiple applications in a node and changing the contention parameters of the traffic. To change the contention parameters, use this code:

```
% Change the contention parameters for Background traffic
backgroundContentionParams.cwMin = 15; % CW minimum
backgroundContentionParams.cwMax = 1023; % CW maximum
backgroundContentionParams.aifsn = 7; % AIFSN slots
```

```
% Change the contention parameters for Best-effort traffic
bestEffortContentionParams.cwMin = 15; % CW minimum
bestEffortContentionParams.cwMax = 1023; % CW maximum
bestEffortContentionParams.aifsn = 3; % AIFSN slots

% Change the contention parameters for Video traffic
videoContentionParams.cwMin = 7; % CW minimum
videoContentionParams.cwMax = 15; % CW maximum
videoContentionParams.aifsn = 2; % AIFSN slots

% Change the contention parameters for Voice traffic
voiceContentionParams.cwMin = 3; % CW minimum
voiceContentionParams.cwMax = 7; % CW maximum
voiceContentionParams.aifsn = 2; % AIFSN slots

% Save updated contention parameters to 'macConfiguration.mat'
save('macConfiguration.mat', 'backgroundContentionParams', 'bestEffortContentionParams', 'videoCo
```

Related examples

Refer these examples for further exploration:

- To measure MAC and Application layer throughput in a multi-node 802.11a/n/ac/ax network, refer "802.11 MAC and Application Throughput Measurement" on page 7-27 example.

- To model a multi-node IEEE 802.11ax network with abstracted PHY, refer "802.11ax System-Level Simulation with Physical Layer Abstraction" on page 7-52 example.

- To model a multi-node network using Distributed Coordination Function (DCF) MAC and 802.11a PHY, refer "Multi-Node 802.11a Network Modeling with PHY and MAC" on page 7-63 example.

This example enables you to configure and simulate MAC QoS traffic scheduling in 802.11 network using a Simulink model for analyzing MAC latencies in each AC. In this model, MAC contention parameters for each AC can be configured to change the application traffic priority at the MAC layer. This example concludes that the voice traffic which has the highest priority, incurs the lowest latency. Whereas, the background traffic that has the lowest priority results in highest latency.

Appendix

The helper functions and objects used in this example are:

1 edcaFrameFormats.m: Create an enumeration for PHY frame formats.
2 edcaNodeInfo.m: Return MAC address of a node.
3 edcaPlotQueueLengths.m: Plot MAC queue lengths in the simulation.
4 edcaPlotStats.m: Plot MAC state transitions with respect to simulation times.
5 edcaStats.m: Create an enumeration for simulation statistics.
6 edcaUpdateStats.m: Update statistics of the simulation.
7 edcaLogLatencies.m: Logs the given information of all frames exchanged in the network.
8 edcaPlotLatencies.m: Plots the MAC frame latencies in all access category.
9 helperAggregateMPDUs.m: Aggregate MPDUs to form an A-MPDU.
10 helperSubframeBoundaries.m : Return subframe boundaries of an A-MPDU.

11 phyRx.m: Model PHY operations related to packet reception.

12 phyTx.m: Model PHY operations related to packet transmission.

13 edcaApplyFading.m: Apply Rayleigh fading effect on the waveform.

14 heSIGBUserFieldDecode.m: Decode HE-SIG-B user field.

15 heCPECorrection.m: Estimate and correct common phase error.

16 heSIGBCommonFieldDecode.m: Decode HE-SIG-B common field.

17 heSIGBMergeSubchannels.m: Merge 20MHz HE-SIG-B subchannels.

18 addMUPadding.m: Add or remove the padding difference between an HE-SU and HE-MU PSDU.

19 macQueueManagement.m: Create a WLAN MAC queue management object.

20 roundRobinScheduler.m: Create round-robin scheduler object.

21 calculateSubframesCount.m: Calculate the number of subframes required to form MU-PSDU.

22 interpretVHTSIGABitsFailCheck.m: Interprets the bits in VHT-SIG-A field

23 rateAdaptationARF.m: Create an auto rate fallback (ARF) algorithm object.

24 rateAdaptationMinstrelNonHT.m: Create a minstrel algorithm object.

References

1 IEEE Std 802.11™. "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; LAN/MAN Standards Committee of the IEEE Computer Society

802.11ax System-Level Simulation with Physical Layer Abstraction

This example demonstrates how to model a multi-node IEEE® 802.11ax™ [1] network with abstracted physical layer (PHY) using SimEvents®, Stateflow®, and WLAN Toolbox™. A PHY abstraction model largely reduces the complexity and the duration of system-level simulations by replacing the actual physical layer computations. This makes it possible to evaluate systems consisting of large number of nodes, resulting in increased scalability. Abstracted PHY models signal-power, gain, delay, loss and interference on each packet without generating physical layer packets, as specified by the TGax Evaluation Methodology [3].

Physical Layer Abstraction

This example shows how to model an 802.11ax network with abstracted PHY. The example presents a variation of the system model used in the example "802.11 MAC and Application Throughput Measurement" on page 7-27. In "802.11 MAC and Application Throughput Measurement" on page 7-27 example, full PHY processing is modeled where waveforms are generated and decoded at the physical layer. However, this example models an abstracted PHY where no waveforms are generated or decoded. Abstracting the physical layer reduces the time taken for simulation at the cost of fidelity. Fidelity refers to the degree of exactness with which the PHY is modeled in the simulation. Simulations that tolerate low fidelity at the physical layer can use the abstracted PHY model.

The abstracted PHY operates on pre-computed packet error rate (PER) tables and equations. These tables and equations are used to estimate the corrupted packet without any actual modulation or demodulation of packets, resulting in a low fidelity model. Refer the example "Physical Layer Abstraction for System-Level Simulation" on page 7-85 for more details related to the PHY abstraction.

Abstracted Physical Layer Blocks

This section explains the blocks used for modeling the abstracted PHY and how it fits into the 802.11 [2] network model. Full PHY modeling involves operations related to waveform transmission and reception through a fading channel. Abstracted PHY models signal-power, gain, delay, loss and interference on each packet without generating physical layer packets. This example provides a `PHY Transmitter`, a `Statistical Channel`, and a `PHY Receiver` for modeling an abstracted PHY. These blocks are available in the library wlanAbstractedPHYLib.

Abstracted PHY Transmitter:

The `Abstracted PHY Transmitter` block models the transmit chain of the physical layer. This block consumes the frame and corresponding transmission parameters from the MAC layer. Parameters like transmit power, preamble duration, header duration and payload duration are calculated in the block. This information is passed along with the MAC frame as the metadata to simulate the transmission of a waveform.
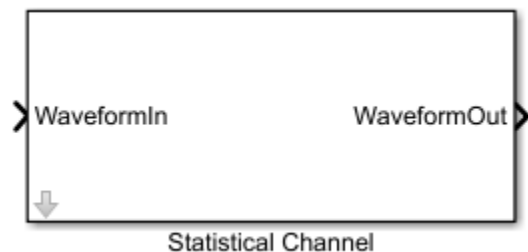
Abstracted PHY Transmitter
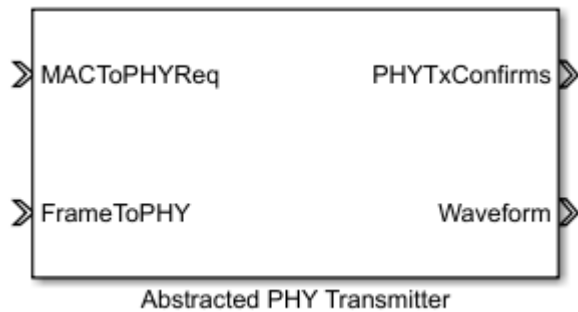
Interfaces to the `Abstracted PHY Transmitter` block are:

- MACToPHYReq: Triggers for indicating transmission start/end requests from MAC layer
- FrameToPHY: MAC frame to be transmitted
- PhyTxConfirms: Confirmation triggers to MAC layer for indicating completion of MAC layer requests
- Waveform: Abstract waveform transmitted into the channel (MAC frame and the metadata)

Statistical Channel:

The `Statistical Channel` block models pathloss, propagation delay, and reception range of the packet. To enable the estimation of loss, delay, and range at each receiver, the `Statistical Channel` block must be modeled inside every node coupled with the `Abstracted PHY Receiver`. Propagation delay is applied on each received packet, and the signal strength of each packet is degraded with optional pathloss. If the receiving node is within the range, the packet is forwarded to the `Abstracted PHY Receiver` with the effective signal strength. The packet is dropped if the receiving node is outside the range of the transmitter.

Statistical Channel

Interfaces to the `Statistical Channel` are:

- WaveformIn: Input packet received from a PHY transmitter
- WaveformOut: Output packet intended for PHY receivers after applying channel loss

Abstracted PHY Receiver:

The `Abstracted PHY Receiver` block models the receive chain of the physical layer. This block receives and processes the packet based on the received metadata. The `Abstracted PHY Receiver` block models interference based on the packets received at overlapping timelines. The received packets are processed only at these checkpoints: (a) End of the preamble duration (b) End of

each subframe duration in the payload for aggregated frames (or) end of the payload duration for non-aggregated frames.

This block also provides an option for configuring the level of abstraction through the PHY `Abstraction` mask parameter. You can configure it to `'TGax Evaluation Methodology Appendix 1'` [3] to predict the performance of a link with a TGax channel model using effective SINR mapping. Details of this procedure can be found in the example "Physical Layer Abstraction for System-Level Simulation" on page 7-85. Alternatively, you can configure it to `'TGax Simulation Scenarios MAC Calibration'` [4] to assume a packet failure on interference, without actually calculating the link performance. Note that the option `'TGax Evaluation Methodology Appendix 1'` works for only MCS values in the range [0-9], as the TGax Evaluation Methodology [3] is defined only for these values.

Abstracted PHY Receiver
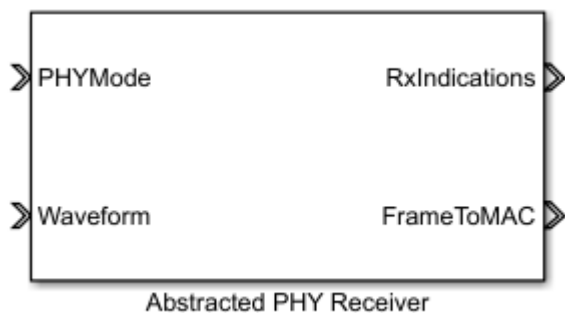
Interfaces to the `Abstracted PHY Receiver` block are:

- PHYMode: Trigger for switching off the receiver function when transmission is in progress
- Waveform: Abstract waveform received from the channel (MAC frame and the metadata)
- RxIndications: Triggers to MAC for indicating channel state shift (busy/idle) events or receive (start/end) events
- FrameToMAC: Received MAC frame

System-Level Simulation

This example simulates a network with 10 nodes in the model, WLANMultiNodeAbstractedPHYModel, as shown in this figure. These nodes implement carrier-sense multiple access with collision avoidance (CSMA/CA) with physical carrier sense and virtual carrier sense. The physical carrier sensing uses the clear channel assessment (CCA) mechanism to determine whether the medium is busy before transmitting. The virtual carrier sensing uses the RTS/CTS handshake to prevent the hidden node problem.
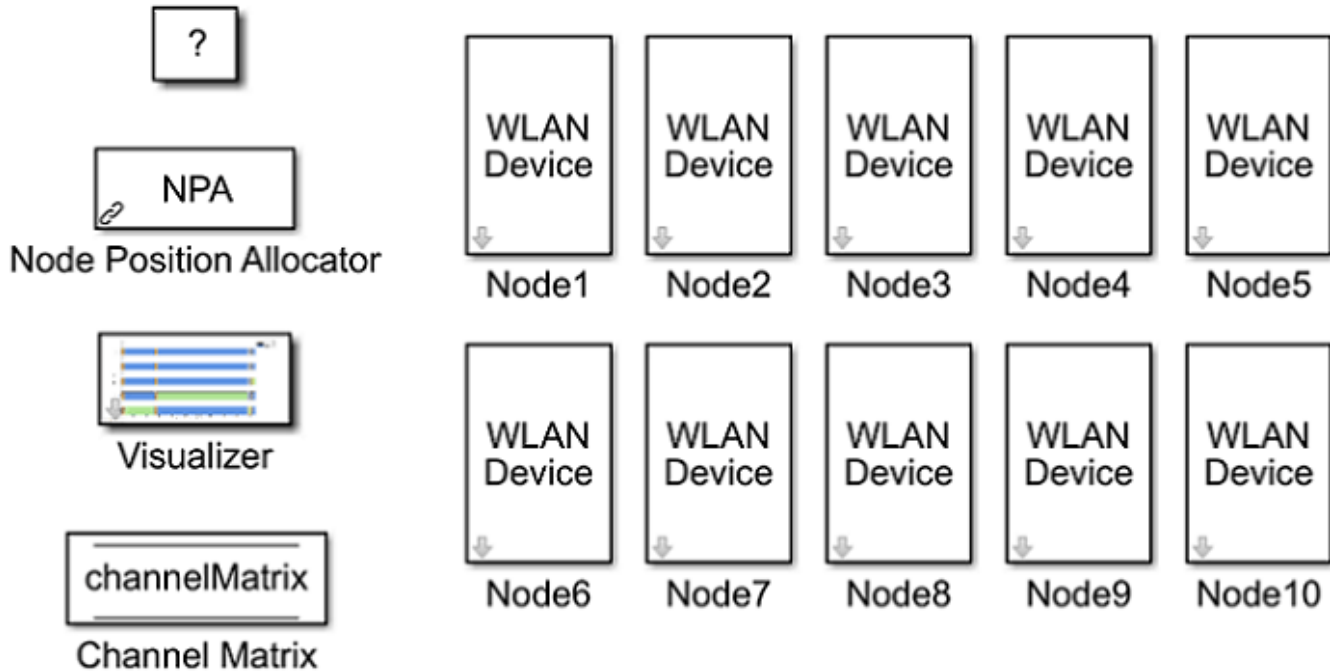
The positions for all the nodes in the network are configured through the node position allocator (NPA) block in the model. The state of each node can be visualized during run-time through the configuration available in the `Visualizer` block. The `Channel Matrix` block is a Data Store Memory. On initialization, a TGax channel realization is generated between each pair of nodes in the network and the resulting channel matrix per subcarrier is stored in the block. During the simulation, each receiver node accesses the memory to obtain the channel matrix between itself and a transmitting node to determine the link quality. In this model, nodes 1, 2, 3, 6, 7, and 8 act as both the transmitters and receivers, while nodes 4, 5, 9, and 10 are just passive receivers.

Node Subsystem

Each node in the above model is a subsystem representing a WLAN device. Each node contains an application layer, a MAC layer and a physical layer. The physical layer is modeled using the abstracted PHY blocks described in the previous section. You can configure a node to transmit and receive packets on a specific channel (frequency) by changing the `Multicast tag` parameter of the `Entity Multicast` and the `Multicast Receive Queue` blocks. By default, all nodes operate on the same channel. You can also configure the receive range for a specific node using the `Packet Receive Range` parameter of the `Statistical Channel` block.

You can easily switch between abstracted PHY blocks available in the wlanAbstractedPHYLib and full PHY processing blocks available in the `wlanFullPHYLib.slx` library of the example "802.11 MAC and Application Throughput Measurement" on page 7-27. The interfaces to the transmitter, receiver and channel blocks remain the same. By default, the abstracted PHY blocks run in the `Interpreted execution` mode. For longer simulation time, configure all the blocks to `Code generation` mode for better performance.
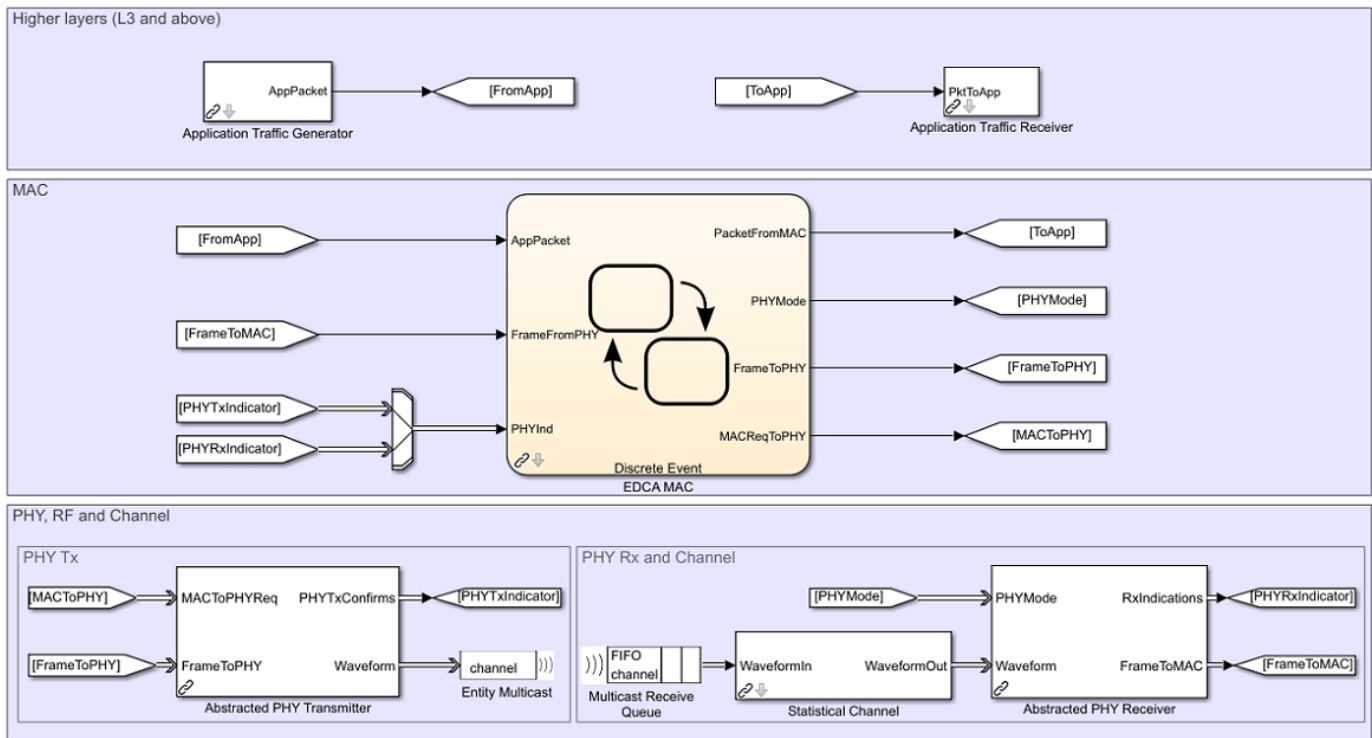
Simulation results

Running the model simulates the WLAN network for the specified simulation time. A plot with network-level statistics (corresponding to MAC layer) is generated at the end of simulation. Detailed node-level statistics (corresponding to application, MAC, and physical layers) are collected during the simulation and saved to a base workspace file `statistics.mat`. You can also enable an optional live visualization, to see the state of each node during run-time, through the mask configuration of the `Visualizer` block.

Network-Level Statistics
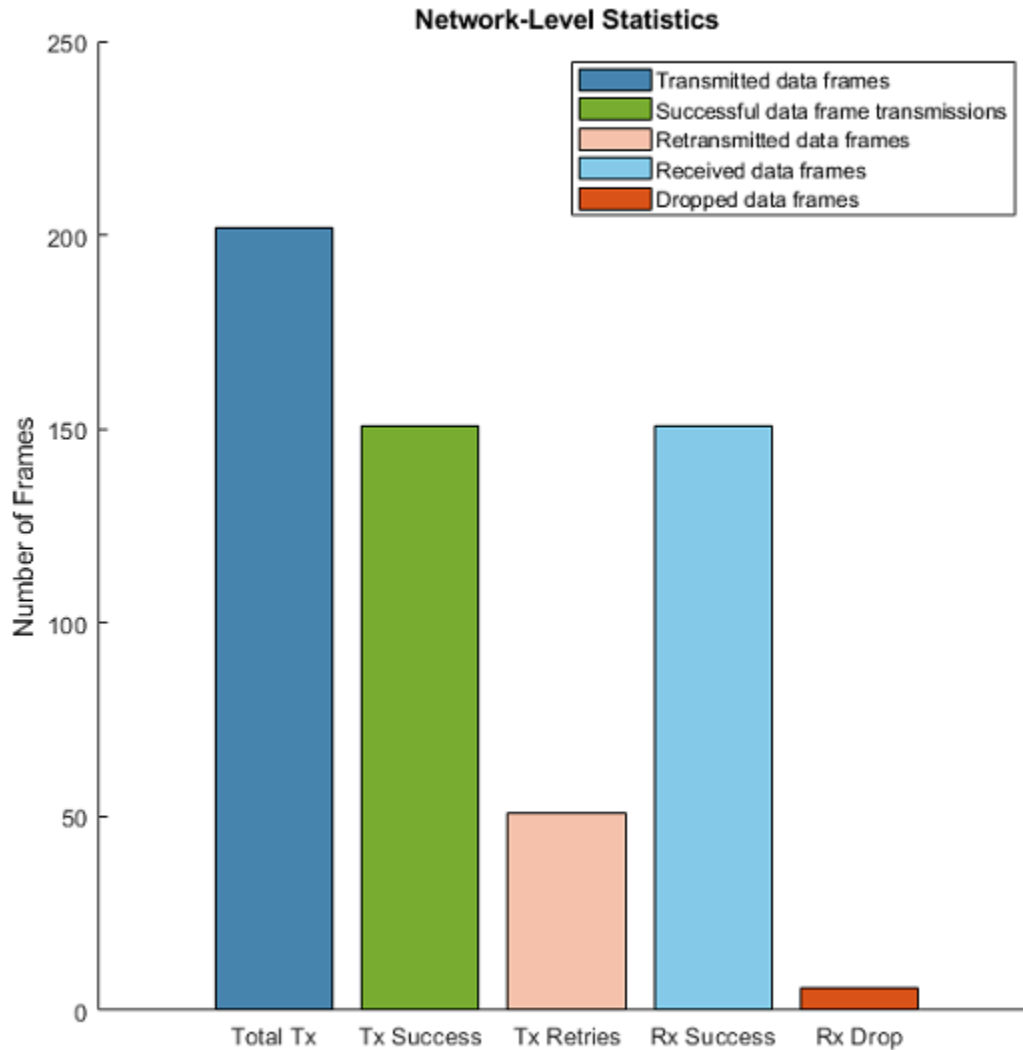
Scalability

The above model shows a network of 10 nodes. You can create a network with a large number of nodes by using the hCreateWLANNetworkModel function. This helper function uses the node subsystem from this example and creates a network of WLAN nodes positioned linearly 10 meters apart from each other. You can create different simulation scenarios and analyze the node-level or network-level statistics with varying number of nodes. For example, the plot below shows the retransmissions and successful transmissions relative to the total transmissions, as the number of nodes in the network increase. The configuration parameters used for collecting the results are:

- Format: HE-SU
- Modulation and coding scheme (MCS) index: 0
- Number of subframes in A-MPDU: 1

- Distance between nodes: 10 meters
- Path loss: Not applied
- PHY abstraction type: "TGax Evaluation Methodology Appendix 1"
- Range propagation: All the nodes are within range of each other
- Operating frequency: All the nodes operate in the same frequency
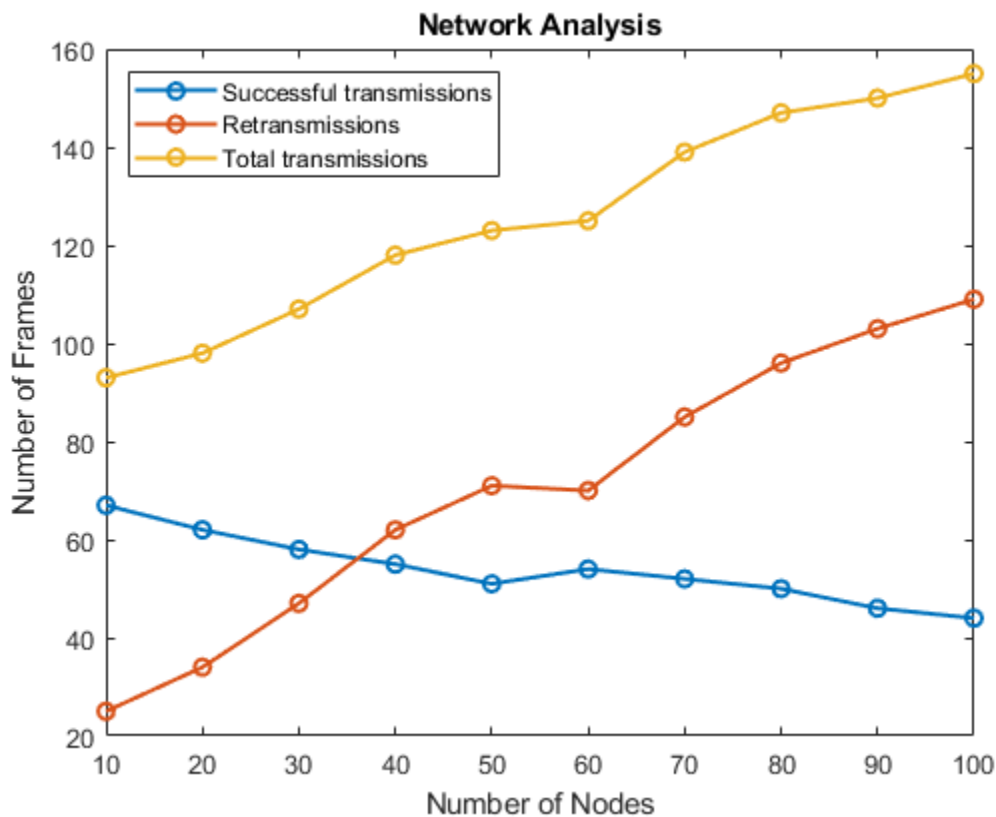
The plot below shows that the simulation runs faster with abstracted PHY as compared to full PHY processing, thus making it more scalable. The configuration parameters used for collecting the performance results are:

- Format: HE-SU
- Modulation and coding scheme (MCS) index: 0
- Number of subframes in A-MPDU: 2
- Distance between nodes: 1 meter
- Path loss: Not applied
- PHY abstraction type: "TGax Evaluation Methodology Appendix 1"
- Range propagation: All nodes are within range of each other
- Operating frequency: All the nodes operate in the same frequency
- Simulation mode: `Code generation` mode for all the blocks
- Simulation time: 5 seconds

- Packet generation interval: 0.001 seconds
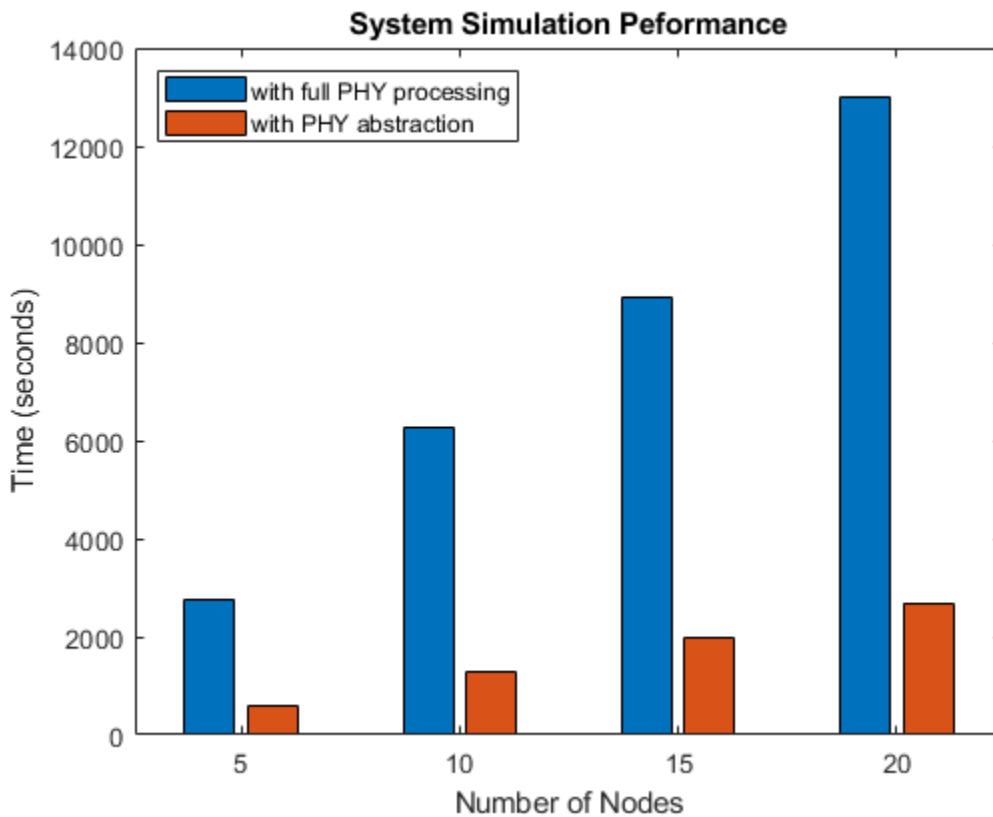
This example explained the physical layer abstraction and demonstrated a 10-node WLAN network with abstracted PHY. This example shows that a network simulation with abstracted PHY is faster and more scalable compared to using full PHY processing.

Further Exploration

In this example, the A-MPDUs exchanged between the nodes are deaggregated to MPDUs at the receiving node. These MPDUs are exported to packet capture (PCAP) and packet capture next generation (PCAPNG) format file using the `pcapDump` DES block. To use the `pcapDump` DES block, go to wlanSystemLevelComponentsLib

Export to PCAP/PCAPNG Format File

The PCAP/PCAPNG format files contain the packet data of the network. These files are mainly associated with network analyzers like Wireshark [5], a third party tool used to visualize and analyze PCAP/PCAPNG files. The main advantages of using PCAP/PCAPNG files during system level simulations are:

- Monitor the network traffic.

- Visualize and analyze the network characteristics of the data.

To duplicate the MAC layer input entities (received A-MPDUs, `FrameToMAC`, and `PhyRxIndicator` vector) and output entities (transmitted A-MPDUs, `FrameToPHY`, and `MACReqToPHY` vector), use the

`Entity Replicator` blocks. The MAC layer provides `RxFrameToPCAP`, `PhyIndToPCAP`, `TxFrameToPCAP`, and `MACReqToPCAP` as inputs to the `pcapDump` DES block.
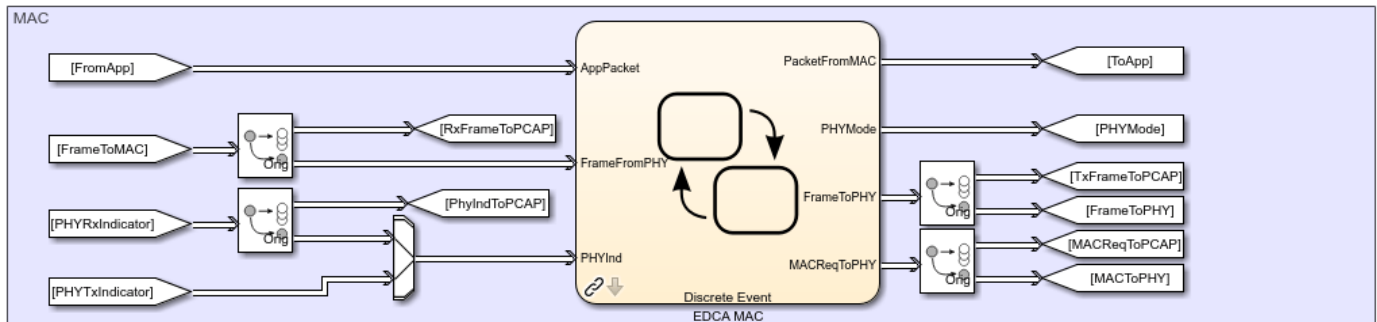
The `pcapDump` DES block contains two input ports, one for Tx/Rx A-MPDUs and other for Tx/Rx information.
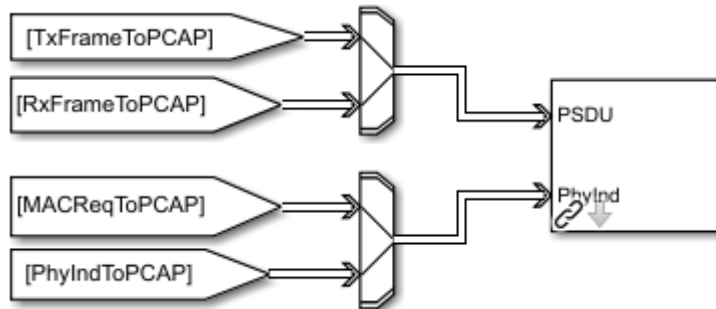
Select the capture format as pcap or pcapng. As the simulation starts, the packets exchanged between the nodes are logged into the selected capture format file.

To capture the packet, double click the `pcapDump` DES block and select the parameter Capture as Enable.
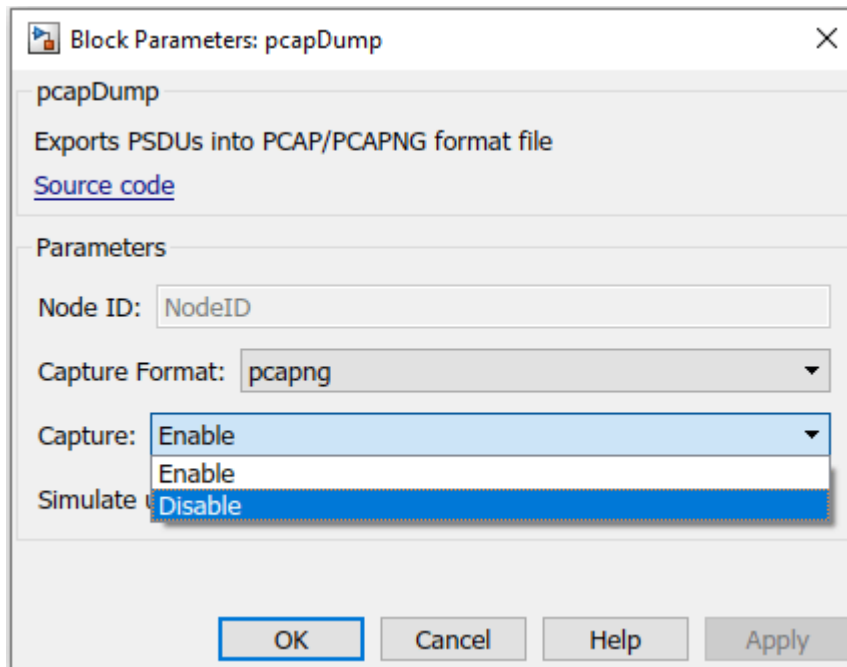
A new capture file (PCAP/PCAPNG format) is created for every node. The file name corresponds to name of the node. If name of the node is Node1, the captured file name is Node1.pcap or Node1.pcapng.

Appendix

The example uses these helpers:

1. edcaFrameFormats.m: Create an enumeration for PHY frame formats.
2. edcaNodeInfo.m: Return MAC address of a node.
3. edcaPlotQueueLengths.m: Plot MAC queue lengths in the simulation.
4. edcaPlotStats.m: Plot MAC state transitions with respect to simulation times.
5. edcaStats.m: Create an enumeration for simulation statistics.
6. edcaUpdateStats.m: Update statistics of the simulation.
7. helperSubframeBoundaries.m : Return subframe boundaries of an A-MPDU. * phyTxAbstracted: Model PHY operations related to packet transmission * phyRxAbstracted: Model PHY operations related to packet reception * channelBlock: Model the channel for a node
8. addMUPadding.m: Add or remove the padding difference between an HE-SU and HE-MU PSDU
9. macQueueManagement.m: Create a WLAN MAC queue management object
10. roundRobinScheduler.m: Create round-robin scheduler object
11. calculateSubframesCount.m: Calculate the number of subframes required to form MU-PSDU * hCreateWLANNetworkModel: Create a WLAN network with given number of nodes * hDisplayNetworkStats: Display network level statistics * hSetupAbstractChannel: TGax channel setup * HelperPCAPNGWriter: Create a PCAPNG file writer handle object * HelperPCAPWriter: Create a PCAP file writer handle object * HelperWLANPacketWriter: Create a file writer handle object that writes WLAN packets into PCAP/PCAPNG format file * HelperPCAPUtils: Provide methods that are commonly used in PCAP helpers * createRadiotapHeader: Create a radiotap header

12 rateAdaptationARF.m: Create an auto rate fallback (ARF) algorithm object.

13 rateAdaptationMinstrelNonHT.m: Create a minstrel algorithm object.

References

1 IEEE P802.11ax™/D4.1 Draft 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 6: Enhancements for High Efficiency WLAN.

2 IEEE Std 802.11™ - 2016 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 802.11-14/0571r12 - 11ax Evaluation Methodology.

4 IEEE 802.11-14/0980r16 - TGax Simulation Scenarios.

5 Wireshark - Go Deep. https://www.wireshark.org/. Accessed 9 Dec. 2019.

Multi-Node 802.11a Network Modeling with PHY and MAC

This example shows how to model communication between multiple WLAN nodes containing medium access control (MAC) layer and PHY using SimEvents®, Stateflow®, and WLAN Toolbox™. This example models a 802.11 network consisting of five WLAN nodes. The distributed coordination function (DCF) in MAC is modeled as a Discrete Event Chart, whereas the PHY is modeled using MATLAB Discrete Event System block. The derived simulation results include a run-time plot displaying the time spent on channel contention, transmission, and reception for each node. A plot representing the per-node metrics such as number of transmitted, received, and dropped packets at PHY and MAC layers is also shown.

Background

The IEEE® 802.11™ is a set of MAC and PHY specifications for WLAN implementation. A typical WLAN network will have multiple devices (nodes) sharing the same channel resources. Each node can have different types of applications transmitting or receiving packets from other nodes. As the nodes share the channel, their transmissions may collide or interfere with each other.

Modeling a WLAN network with multiple nodes typically involves modeling the MAC layer, PHY, shared communication channel, data traffic pattern, and interference. This example shows how to model a multi-node WLAN network with 802.11a PHY and MAC using discrete event simulation.

Multi-Node 802.11a Network Model

This example models a WLAN network with five nodes. These nodes implement carrier-sense multiple access with collision avoidance (CSMA/CA) with physical carrier sense and virtual carrier sense. The physical carrier sensing uses the clear channel assessment (CCA) mechanism to determine whether the medium is busy before transmitting. Whereas, the virtual carrier sensing uses the RTS/CTS handshake to prevent the hidden node problem.

The model in the example outputs various statistics such as the number of transmitted, received, and dropped packets at PHY and MAC layers, and plots that help in analyzing/estimating the node-level and network-level performance.

The modeling includes:

- Multiple nodes, where each node contains an application, a MAC layer, and a PHY.
- Packets transmitted over the channel, which is shared across multiple nodes.
- A shared channel, which is simulated with these channel impairment options: free-space pathloss, range propagation loss and multi-path fading (using Rayleigh channel model).
- A node position allocator, which is used to configure the position of nodes in the network.

In the example, nodes 1, 2, and 3 are active participants in the communication, while nodes 4 and 5 passively receive the packets over the channel.
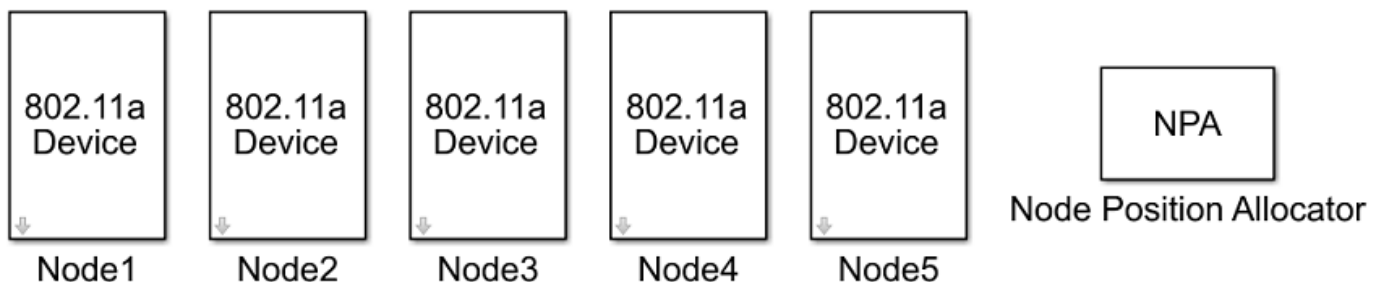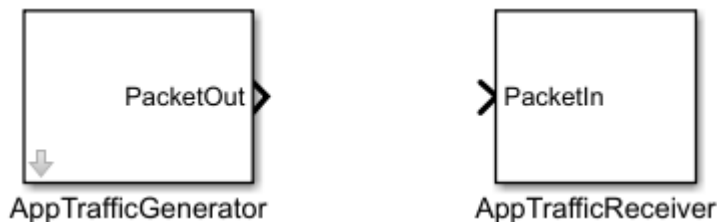
Node

Each node is modeled as a subsystem with a network stack, which includes application, MAC, and physical layers.
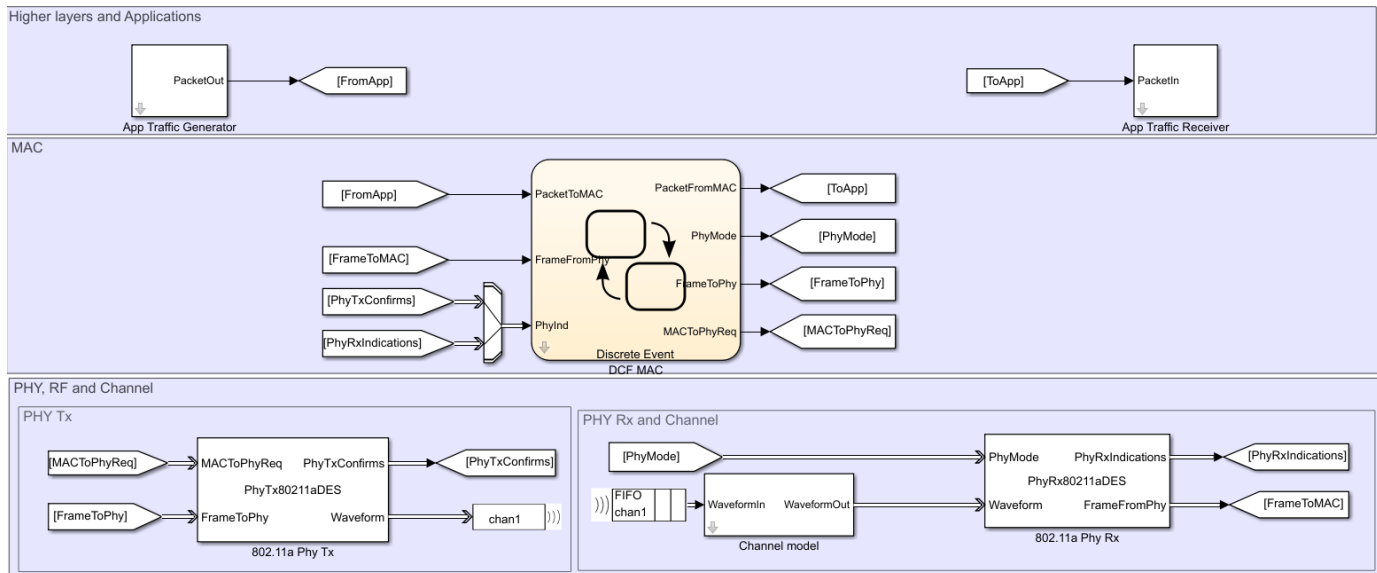
- The application layer generates packets using SimEvents Entity Generator (SimEvents) block.
- The DCF in MAC is modeled as a Discrete Event Chart.
- The PHY is modeled using SimEvents Discrete Event System block.
- The shared channel is modeled inside each node in the receive path.

Application Layer

The application layer is implemented to generate and receive application traffic. It is divided into two sub-blocks:

- AppTrafficGenerator – Uses SimEvents Entity Generator (SimEvents) block to generate data packets with configured packet size, inter-packet interval, and destination node. These data packets are passed to the MAC layer.
- AppTrafficReceiver – Uses SimEvents Entity Terminator (SimEvents) block to receive the data packets from MAC layer.

MAC Layer

The MAC layer implements the DCF algorithm specified in section 10.3 of [1]. Before transmitting a packet, the MAC layer senses the shared channel to determine the state of the channel. If the channel

is idle, packet transmission is initiated. If the channel is busy, packet transmission is deferred. The packets waiting for transmission are queued.

In the model, the MAC layer has two components: queue management and contention algorithm.

Queue Management

Packets received from the application layer are queued until the channel is available. Packets are dequeued for transmission once the channel becomes available. The size of the queue is configurable. The default queue size is 10.
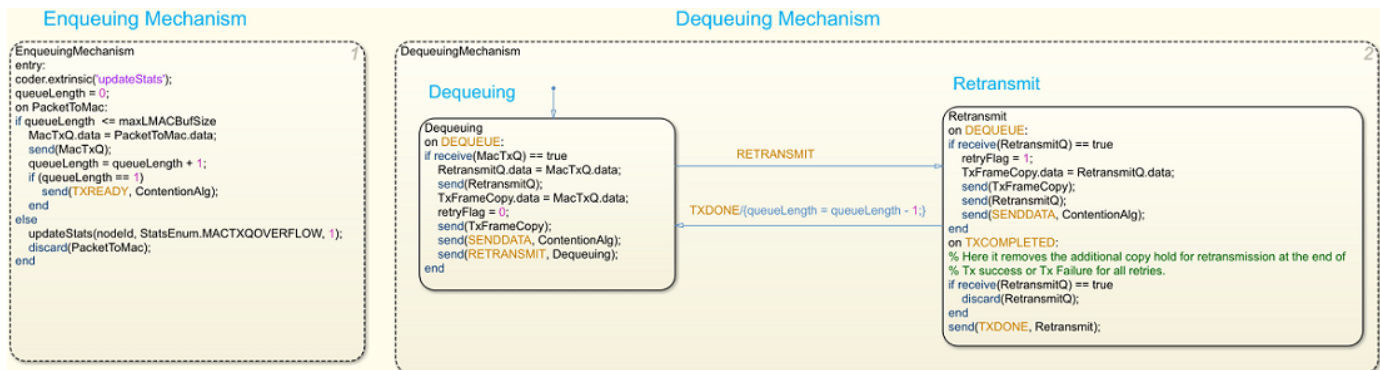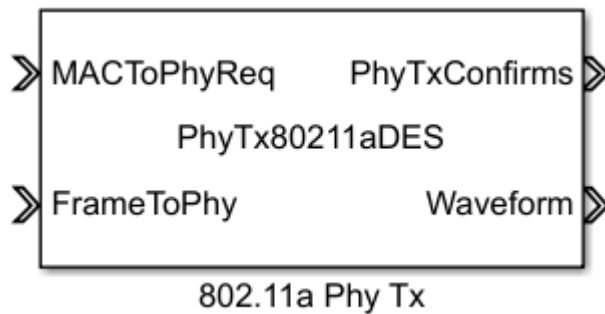

Contention Algorithm

The contention algorithm implemented in the MAC layer is the DCF functionality specified in section 10.3 of [1]. It is implemented as a finite state machine with these six different states:

- Idle: No active operations
- Contend: Contends for the channel
- Rx: Receives and processes the frame
- WaitForRx: Waits for a response frame
- SendingData: Transmits a frame
- Eifs: Defers transmission for error recovery

State Level Processing in the MAC Layer

- Initially the MAC layer is in *Idle* state. On receiving a packet from the application layer, the MAC layer moves to the *Contend* state.

- In the *Contend* state, if the channel is sensed as idle for a period of DCF inter frame space (DIFS) time followed by a random back-off time, the MAC layer moves to the *SendingData* state.

- In the *SendingData* state, a frame is transmitted, and the MAC layer moves to the *WaitForRx* state.

- In the *WaitForRx* state, the MAC layer waits for a period of ACK/CTS timeout. On receiving a response frame, MAC initiates next frame transmission. If a response frame is not received within the timeout period, the frame is re-transmitted.

- On receiving a clear channel assessment (CCA) as busy from the PHY layer, the MAC layer moves to the *Rx* state. Frames are received and processed in the *Rx* state. If an errored frame is received in the *Rx* state, the MAC layer waits for extended inter frame space (EIFS) time in the *Eifs* state. If the frame is intended for other node, network allocation vector (NAV) is updated and transmission is deferred until NAV becomes zero. If the frame is intended to this node, a response frame is sent if needed.

MAC parameters such as RTS threshold, retry limit and data rates are configurable for each node.

PHY and Channel

Transmit Chain:

The MAC layer initiates the transmission by sending a transmission start request containing Tx vector information. On receiving the start request, the PhyTx80211aDES block configures the PHY transmission parameters with the given Tx vector, and sends the transmission start confirm to the MAC layer. The PHY parameters are configured in a non-HT format configuration object of type `wlanNonHTConfig`. On receiving the start confirm, the MAC layer sends the frame to the PhyTx80211aDES block.

The PhyTx80211aDES block generates a waveform for the MAC frame using `wlanWaveformGenerator` function. It also scales the samples of waveform with the configured Tx gain. The generated waveform is transmitted through the shared channel.

You can configure the mask properties of Tx power (dBm) and the Tx gain (dB) for the PhyTx80211aDES block.

Channel Impairments Modeling:

Channel impairments determined by free-space path-loss model and Rayleigh multipath fading are added to the transmitted PHY waveform. You can choose to enable or disable these impairment models. In addition to the impairment models, the signal reception range can also be limited by an optional range propagation loss model. To model any of these losses, the channel model must contain both the sender and receiver positions along with the transmitted signal strength. The channel is modeled inside each receiving node, before passing the waveform to the PhyRx80211aDES block.

Receiver Chain:

When the PhyRx80211aDES block receives a waveform, it scales the waveform with the configured Rx gain. The PhyRx80211aDES block then applies thermal noise and interference to the received waveform. This is done by calculating the expected signal to interference-plus-noise Ratio (SINR) at the end of preamble, header, and payload. The calculated SINR is added to the preamble, header, and payload of the received waveform as an Additive White Gaussian Noise (AWGN). The PhyRx80211aDES block then compares the waveform power with the energy detection (ED) threshold. If the waveform power is greater than the ED threshold, the PHY sends a CCA busy indication to the MAC layer and starts decoding the waveform. Otherwise, the PHY considers the waveform as noise and adds it as interference to the upcoming waveforms for the duration of the current waveform. If an error is found while decoding, the PHY stops further processing of the waveform and sends an error indication to the MAC layer. If the preamble and header are decoded successfully, the PhyRx80211aDES block sends a start indication to the MAC layer. If the payload is also decoded successfully, the payload is passed to the MAC layer along with a success indication.
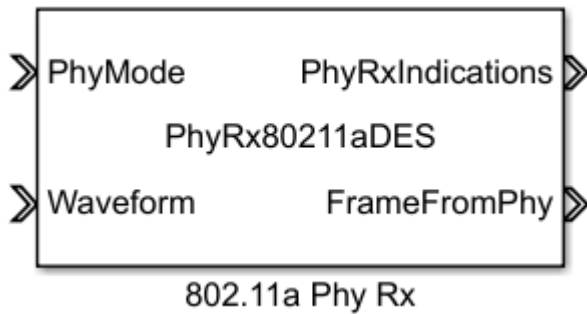
802.11a Phy Rx

The Rx gain (dB) and the noise figure (dB) can be configured for the PhyRx80211aDES block.

Node Position Allocator

Node position allocator is used to assign initial position of nodes. It supports linear and list position allocation strategies.

Linear Position Allocation Strategy – Places nodes uniformly in a straight line, on a 2D grid.

List Position Allocation Strategy – Assigns node positions from a list [[x1 y1 z1] [x2 y2 z2] ... [xn yn zn]] such that (xk, yk, zk) is the position of the kth node for all k in (1, 2, ..., n).

Simulation Results

The simulation of this model generates:

1   A run-time plot depicting the time spent on channel contention, transmission, and reception for each node.
2   A plot depicting metrics for each node such as – number of transmitted, received, and dropped packets at PHY and MAC layers.
3   A mat file `statistics.mat` with detailed statistics obtained at each layer for each node.
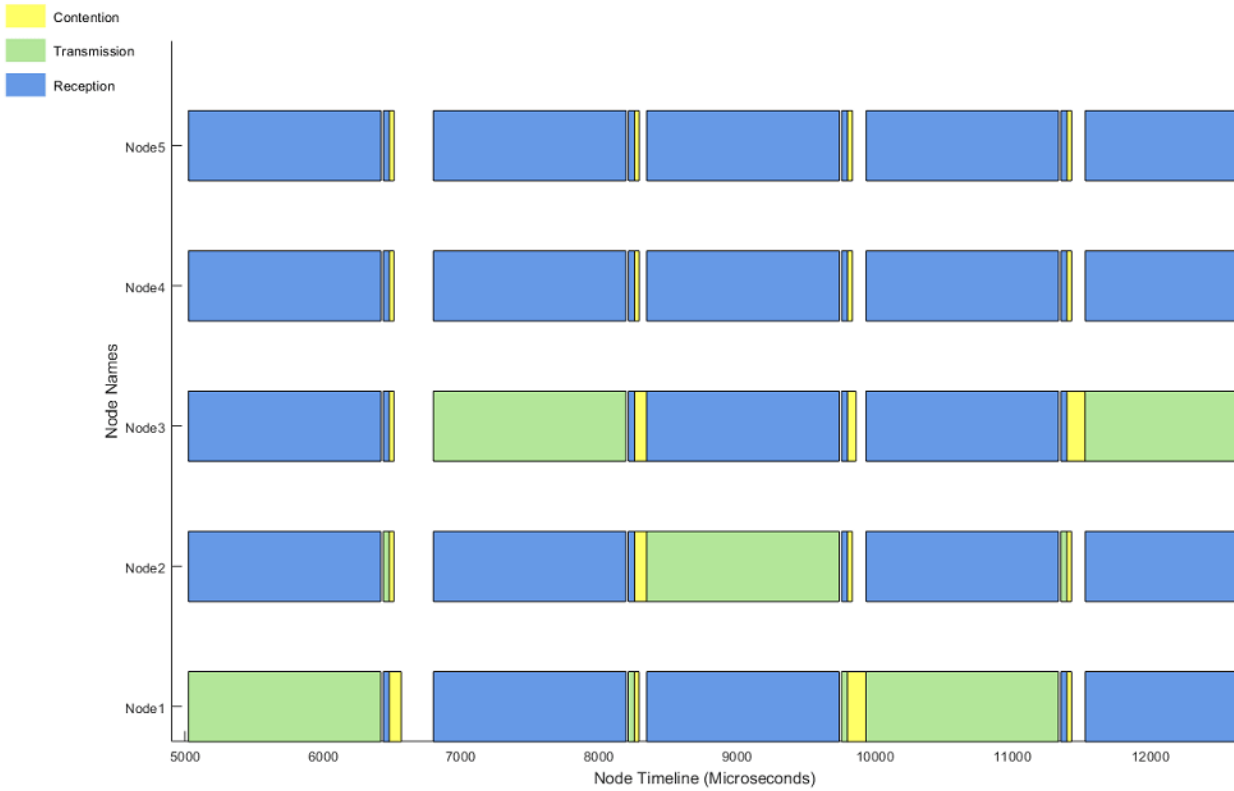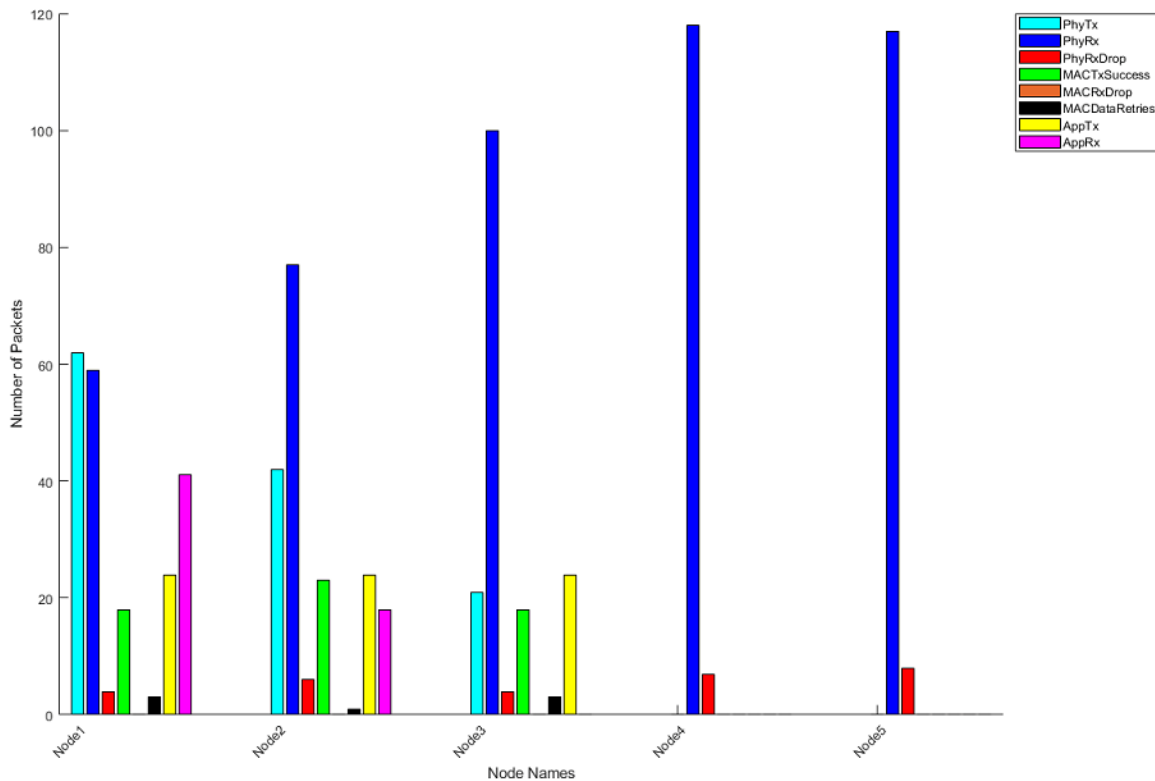
Further Exploration

1   Increase the number of nodes in the network either by copying the existing nodes or by creating a custom node using library blocks.

2   Vary the application parameters: packet size and packet interval.

3   Change the MAC parameters: RTS threshold, Tx queue size, data rate, and retry limit.

4   Change the PHY parameters: PHY Tx gain, PHY Rx gain, and Rx noise figure.

5   Vary the channel modeling parameters.

6   Change the node positions using node position allocator.

7   Compare the throughput by changing different parameters.

8   Enable sequence viewer and see the messages exchanged between the nodes and inside components of the node.

This example presents a multi-node 802.11a network and show you how to model a communication between multiple WLAN nodes using discrete event simulation. DCF in MAC is modeled as a Discrete Event Chart, whereas the PHY is modeled using MATLAB Discrete Event System block. The run-time visualization display showing the time spent on channel contention, transmission, and reception by each node helps you to study and analyze the DCF functionality. The network communication performance of each node is analyzed based on the plot exhibiting the node-related metrics such as number of transmitted, received, and dropped packets at PHY and MAC layers.

Limitations

1 The MAC layer retransmissions are based on a common retry counter instead of short retry count (SRC) and long retry count (LRC) specified in section 10.3.4.4 of [1].

2 No management frames are exchanged.

3 The MAC layer in each node maintains a single sequence-number counter for all the frame transmissions, instead of a per-destination counter.

4 Data rate adaptation at the MAC layer is not implemented.

5 MAC service data unit (MSDU) fragmentation is not implemented.

6 The interference is modeled as an AWGN with SINR derived from the signal strength of interfering packets and thermal noise. The interference can also be modeled by combining the IQ samples of interfering signals.

Appendix

The example uses these helpers:

1 dcfApplyFading.m: Apply Rayleigh fading effect on the waveform.

2 dcfAssignNodeIDs.m: Assign node IDs.

3 dcfDisplayStats.m: Display the statistics.

4 dcfGetNodeInfo.m: Perform get or set operation for MAC address of the given node.

5 dcfGetNodeNamesList.m: Get the list of nodes in the model.

6 dcfGenerateMACFrame.m: Generate MAC frames.

7 dcfPlotQueueLengths.m: Plot the queue lengths in the simulation.

8 dcfPlotStats.m: Plot the simulation time-line statistics.

9 dcfStats.m: Enumeration to indicate model statistics.

10 dcfUpdateStats.m: Update the statistics for the given node.

11 PhyPrimitives.m:Enumeration for indications between the PHY and the MAC layer

12 PhyRx80211aDES.m: Model PHY operations related to packet reception.

13 PhyTx80211aDES.m: Model PHY operations related to packet transmission.

References

1 IEEE Std 802.11™ – 2016 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 PHY Specifications

802.11ax PHY-Focused System-Level Simulation

This example shows how to perform a PHY-focused system-level simulation for IEEE® 802.11ax™. Part (A) validates the simulation scenario, radio characteristics, and large-scale fading model by comparing against published calibration results. Part (B) estimates the packet error rate of the 802.11ax network by simulating individual links between active nodes under a basic clear channel assessment scheme.

Introduction

In this example the performance of an 802.11ax [1] network in a residential apartment block is evaluated using a PHY-focused system-level simulation.

The residential apartment block simulation scenario is specified in [2]. This consists of a building with five floors, and twenty 10m x 10m x 3m apartments per floor. Each apartment has an access point (AP) and one or more stations (STAs) placed in random xy-locations, a process referred to as 'dropping' nodes. This creates a basic service set (BSS) per apartment which is randomly assigned one of three channels. The simulation scenario specifies a large-scale path loss model based on the distance between nodes, and the number of walls and floors traversed.
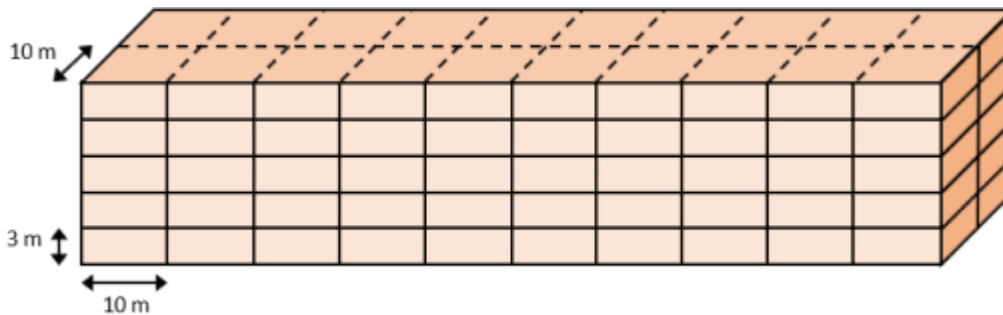
The TGax evaluation methodology [3] for 'PHY System Simulation' is followed for this example:

1   APs and STAs are randomly 'dropped' within the scenario.
2   For each pair of nodes, the large-scale path loss is calculated.
3   One or more 'transmission events' is performed. Each transmission event consists of selecting active APs and STAs based on channel access rules and determining the performance of each link.

This example consists of two parts:

In part (A), the 'calibration' stage, the signal-to-interference-plus-noise ratio (SINR) is calculated for multiple 'drops', assuming downlink interfering transmissions. SINR captures long-term radio characteristics. The cumulative distribution function (CDF) of the SINR is compared with published results from the TGax Task Group [4].

In part (B), the 'PHY system-level simulation' stage, for each transmission event the PHY layer is modeled for individual links. A basic clear channel assessment (CCA) scheme is used to control which APs are active. Waveforms for the signal of interest and interference, impaired by fading channel models, are generated and combined. The resultant packets are processed by a receiver to recover the packet of interest. The average packet error rate for the network is calculated.

The two parts of this example can be disabled using the parameters `calibrate` and `systemLevelSimulation`. A figure displaying the simulation scenarios, nodes, active links, and interference is displayed when `showScenarioPlot` is true.

```
calibrate = true;              % To execute Part A calibration test
systemLevelSimulation = true;  % To execute Part B system-level simulation
showScenarioPlot = true;       % To show dynamic simulation plotting updates
```

Simulation Parameters

The major simulation parameters are defined as either belonging to Physical Layer (PHY), Medium Access Control Layer (MAC), scenario, or simulation. In this example the PHY and MAC parameters are assumed to be the same for all nodes.

```
PHYParameters = struct;
PHYParameters.TxPower = 20;       % Transmitter power in dBm
PHYParameters.TxGain = 0;         % Transmitter antenna gain in dBi
PHYParameters.RxGain = -2;        % Receiver antenna gain in dBi
PHYParameters.NoiseFigure = 7;    % Receiver noise figure in dB
PHYParameters.NumTxAntennas = 1;  % Number of transmitter antennas
PHYParameters.NumRxAntennas = 1;  % Number of receiver antennas
PHYParameters.ChannelBandwidth = 'CBW80'; % Bandwidth of system
PHYParameters.TransmitterFrequency = 5e9; % Transmitter frequency in Hz

MACParameters = struct;
MACParameters.NumChannels = 3; % Number of non-overlapping channels
MACParameters.CCALevel = -70;  % Transmission threshold in CCA algorithm (dBm)
```

The scenario parameters define the size and layout of the residential building as per [3].

```
% Number of Rooms in [x,y,z] directions
ScenarioParameters = struct;
ScenarioParameters.BuildingLayout = [10 2 5];

% Size of each room in meters [x,y,z]
ScenarioParameters.RoomSize = [10 10 3];

% Number of receivers per room. Note that only one receiver (STA) can be
% active at any given time.
ScenarioParameters.NumRxPerRoom = 1;
```

The `NumDrops` and `NumTxEventsPerDrop` parameters control the length of the simulation. For this example, these parameters are configured for a short simulation, but for meaningful results these should be increased.

A 'drop' randomly places transmitters and receivers within the scenario and selects the channel for a BSS. A 'transmission' event randomly selects transmitters and receivers for transmissions according to basic channel access rules.

```
SimParameters = struct;
SimParameters.NumDrops = 3;
SimParameters.NumTxEventsPerDrop = 2;
```

Generate Transmitter Sites

Before the main body of the simulation, the transmitter site objects `txsite` are generated and assigned room names of the form 'Room#' for ease of reference. One transmitter (AP) per room is assumed. Each transmitter is assumed to be isotropic.

```
% Total number of transmitters, assuming one transmitter (tx) per room
numTx = prod(ScenarioParameters.BuildingLayout);

% Create transmitter sites with and isotropic antenna element
roomNames = strings(1,numTx);
for siteInd = 1:numTx
    roomNames(siteInd) = "Room " + siteInd;
end
txs = txsite('cartesian','Name',roomNames,...
    'TransmitterFrequency',PHYParameters.TransmitterFrequency, ...
    'TransmitterPower',10.^((PHYParameters.TxPower+PHYParameters.TxGain-30)/10),...
    'Antenna','isotropic');
```

Generate Receiver Sites

The receive site objects `rxsite` are generated and assigned names of the form 'Room#-STA#' for ease of reference. The Scenario parameter `NumRxPerRoom` is used to define how many receivers (STAs) are present in each room. Each receiver is assumed to be isotropic.

```
% Total number of receivers, assuming one transmitter (tx) per room
numRx = numTx*ScenarioParameters.NumRxPerRoom;

% Create receiver sites
roomNames = strings(1,numRx);
for siteInd = 1:numRx
    roomNames(siteInd) = "Room " + (mod(siteInd-1,numTx)+1) + "-" + ceil(siteInd/numTx);
end
rxs = rxsite('cartesian','Name',roomNames,'Antenna','isotropic');

% Receiver noise power in dBm
T = 290; % Temperature (Kelvin)
k = physconst('Boltzmann'); % Boltzmann constant
% Sample rate (Hz)
fs = wlanSampleRate(wlanHESUConfig('ChannelBandwidth',PHYParameters.ChannelBandwidth));
rxNoisePower = 10*log10(k*T*fs)+30+PHYParameters.NoiseFigure;
```

Part A - Align Long-Term Radio Characteristics

In this section, the simulation scenario, radio characteristics, and large-scale fading model are verified by performing the TGax Evaluation Methodology Box 1 Test 2 Downlink Only calibration test [3]. This test calculates the SINR at all receivers (STAs) assuming all transmitters (APs) are active. Multiple drops of transmitters and receivers are performed as part of the simulation. One active receiver is selected per drop.

The SINR for each receiver is calculated and aggregated over all drops simulated to generate a CDF curve. This curve is compared with the calibration results provided in [4].

A plot showing the node positions, active links, and interfering links is generated per drop. Individual channels can be hidden and shown in the plot by clicking the corresponding legend entry.

```
seed = rng(6); % Seed random number generator and store state

if showScenarioPlot
    hGrid = tgaxBuildResidentialGrid(ScenarioParameters.RoomSize,ScenarioParameters.BuildingLayou
        numTx,numRx,MACParameters.NumChannels);
end
```

```matlab
if calibrate

    fprintf('Running calibration ...\n');

    % Pre-allocate output
    output = struct;
    output.sinr = zeros(SimParameters.NumDrops,numTx); % for storing SINR values
    for drop = 1:SimParameters.NumDrops
        % Drop receivers in each room
        [association,txChannels,rxChannels,txPositions,rxPositions] = tgaxDropNodes( ...
            txs,rxs,ScenarioParameters,MACParameters.NumChannels);

        % All transmitters active
        activeTx = true(numTx,1);

        % Only pick one receiver per Room
        rxAlloc = randi([1 ScenarioParameters.NumRxPerRoom],numTx,1);
        activeRx = reshape(rxAlloc==1:ScenarioParameters.NumRxPerRoom,[],1);

        % Generate propagation model
        propModel = TGaxResidential('roomSize',ScenarioParameters.RoomSize);

        % Get the index of the transmitter for each receiver
        tnum = repmat((1:numTx),1,numRx/numTx);

        % SINR calculation - loop over each non-overlapping channel
        numChannels = numel(unique(txChannels));
        for k = 1:numChannels
            % Use kth non-overlapping channel
            tind = txChannels == k;
            rind = false(size(activeRx));
            rind(activeRx) = rxChannels(activeRx) == k;
            % Get the index of the transmitter of interest for each active receiver
            tsigind = tnum(rind);

            % Calculate SNR
            output.sinr(drop,tind) = sinr(rxs(rind),txs(tind),...
                'ReceiverGain',PHYParameters.RxGain,...
                'ReceiverNoisePower',rxNoisePower,...
                'PropagationModel',propModel,...
                'SignalSource',txs(tsigind));
        end

        % Plot nodes and links
        if showScenarioPlot % update plot data
            mask = txChannels==rxChannels';
            tgaxUpdatePlot(hGrid,txPositions,rxPositions,activeTx,activeRx,mask,txChannels,rxChannels
                sprintf('Box 1 Test 2 "downlink only" calibration, drop #%d/%d',drop,SimParameters.Nu
        end
    end

    % Plot the CDF of SINR and compare with calibration curves
    tgaxCalibrationCDF(output.sinr,'SS1Box1Test2','Long-term Radio Characteristics');

    fprintf('Calibration complete \n')

end
```

```
Running calibration ...
Calibration complete
```

Box 1 Test 2 "downlink only" calibration, drop #3/3

Link, channel 1	Active Tx
Link, channel 2	Inactive Tx
Link, channel 3	Active Rx
Interference, channel 1	Inactive Rx
Interference, channel 2	Room edge
Interference, channel 3	

Calibration: Long-term Radio Characteristics
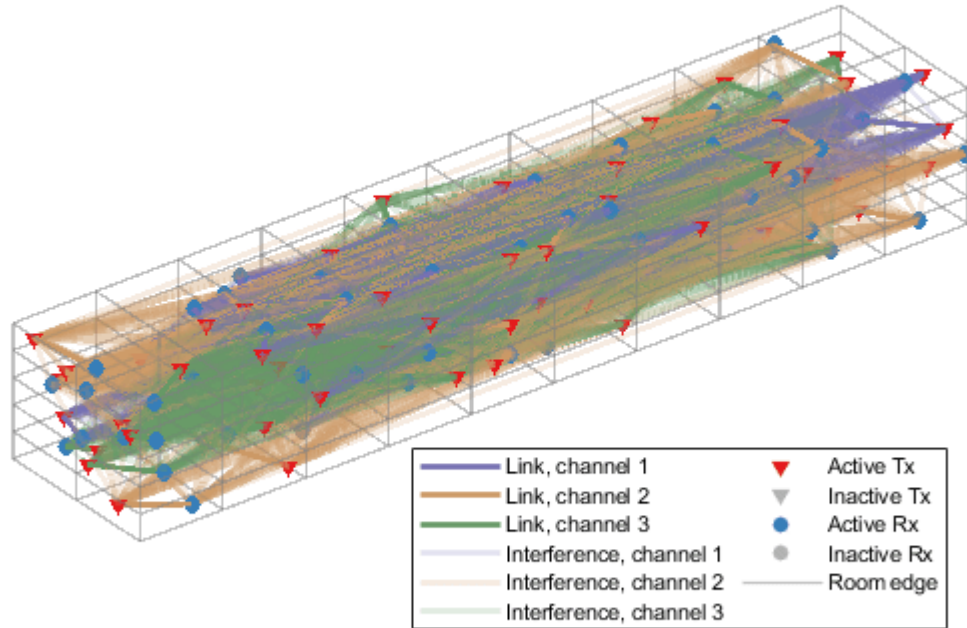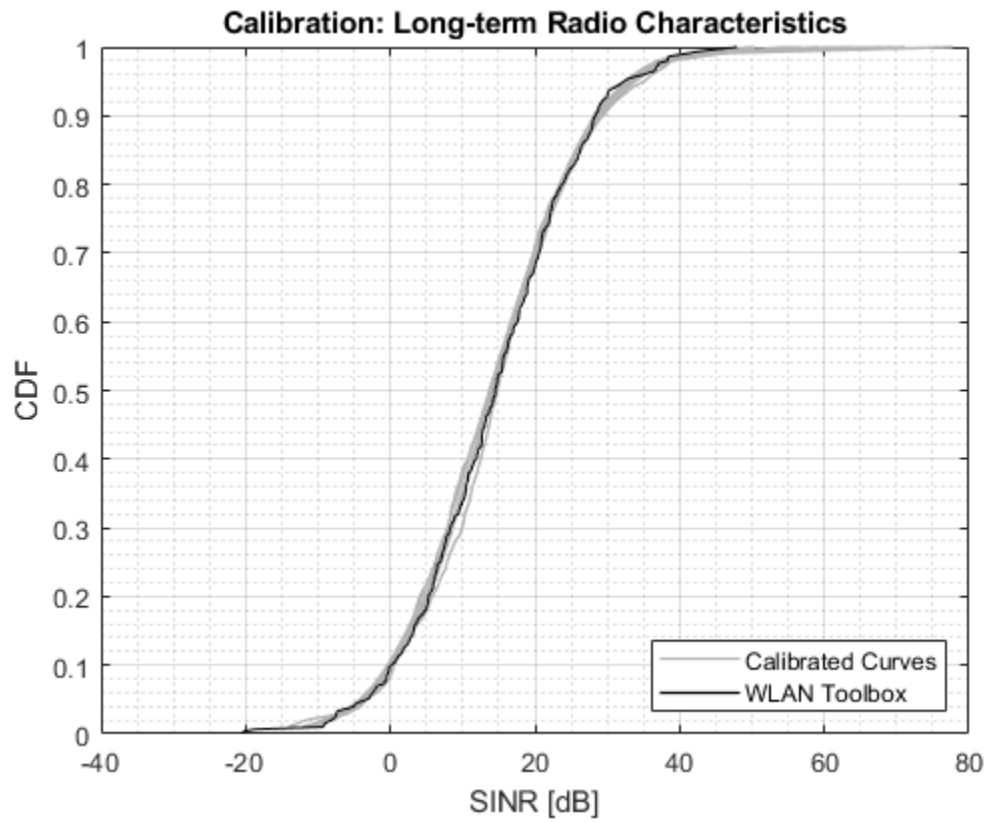
This example simulates a small number of drops. Therefore, for a more meaningful comparison the number of drops simulated should be increased. The calibration result for 100 drops is shown below:

Calibration of Long-term Radio Characteristics - 100 Drops
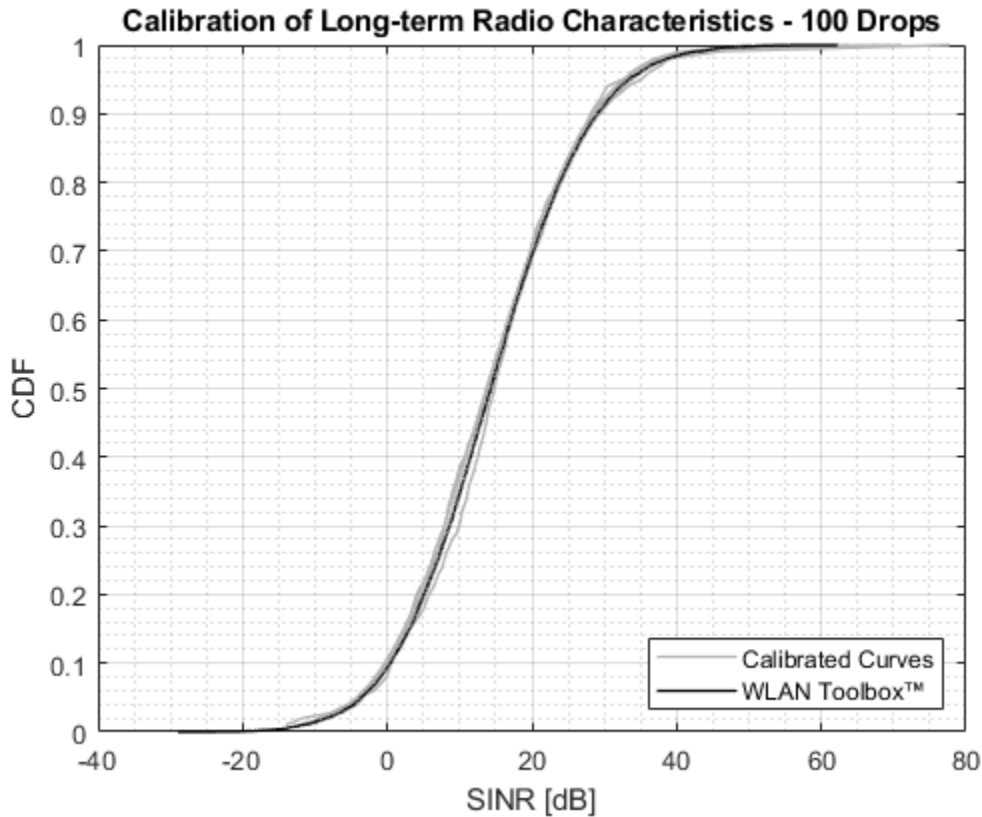
Part B - PHY-focused System-Level Simulation

In this section, the scenario and path-loss model calibrated in part A are used to perform a PHY-focused system-level simulation and determine the packet error rate for the network. This simulation is described as PHY-focused as the PHY is not abstracted and the MAC is simplified. Each active link is modeled using baseband transmitter and receiver processing. A very simple MAC assumes at each transmission event all transmitters (APs) wish to transmit, and one receiver (STA) per BSS is the recipient. A simple CCA algorithm is used to control channel access between transmitters as specified in Figure 4 of [3]. The CCA algorithm enables random transmitters if the signal power received from transmitters that have already been activated does not exceed the CCA threshold, `MACParameters.CCAThreshold`.

The received signal power from all possible interfering transmitters is calculated at each active receiver. If the received power from an interfering transmitter is above the noise floor of the receiver, then the link is modeled with full baseband transmitter and receiver processing. For each modeled link an HE single-user packet is generated and passed through a TGax Model-D NLOS stationary indoor channel model. At the receiver, the waveforms from the transmitter of interest and all interfering transmitters are scaled by the expected path-loss and combined to create a waveform containing the signal of interest plus interference. All waveforms are time aligned. The receiver performs synchronization, demodulation, and decoding to attempt to recover the payload. The decoded payload is compared to the PSDU transmitted in the BSS to determine if the packet has been recovered successfully.

In this example the transmission and channel parameters are assumed to be the same for all nodes. The transmission configuration for all packets is one space-time stream, no space-time block coding and 16-QAM rate-1/2 (MCS 3).

```matlab
if systemLevelSimulation

% Pre-allocate outputs
output.numPkts = zeros(numRx,1);
output.numPktErrors = zeros(numRx,1);
output.sinrMeas = nan(numRx,SimParameters.NumTxEventsPerDrop,SimParameters.NumDrops);
output.sinrEst = nan(numRx,SimParameters.NumTxEventsPerDrop,SimParameters.NumDrops);
output.pktErrorRate = 0;

% For each possible transmitter create a waveform configuration. In this
% example the link and radio parameters are the same for all nodes.
cfgHEBase = wlanHESUConfig;
cfgHEBase.ChannelBandwidth = PHYParameters.ChannelBandwidth; % Channel bandwidth
cfgHEBase.NumTransmitAntennas = PHYParameters.NumTxAntennas; % Number of transmit antennas
cfgHEBase.SpatialMapping = 'Fourier';  % Spatial mapping matrix
cfgHEBase.NumSpaceTimeStreams = 1;     % Number of space-time streams
cfgHEBase.GuardInterval = 0.8;         % Guard interval duration
cfgHEBase.HELTFType = 4;               % HE-LTF compression mode
cfgHEBase.APEPLength = 1e3;            % Payload length in bytes
cfgHEBase.ChannelCoding = 'LDPC';      % Channel coding
cfgHEBase.MCS = 3;                     % Modulation and coding scheme
cfgHE = cell(numTx,1);
for txidx = 1:numTx
    cfgHE{txidx} = cfgHEBase;
end

fprintf('Running System Level Simulation ...\n')
for drop = 1:SimParameters.NumDrops
    fprintf('  Running drop #%d/%d ...\n',drop,SimParameters.NumDrops);

    % Drop receivers in each room
    [association,txChannels,rxChannels,txPositions,rxPositions] = tgaxDropNodes( ...
        txs,rxs,ScenarioParameters,MACParameters.NumChannels);

    % Generate propagation model
    propModel = TGaxResidential('roomSize',ScenarioParameters.RoomSize);

    % Calculate signal strength for all links
    signalStrength = sigstrength(rxs,txs,propModel,'Type','power',...
        'ReceiverGain',PHYParameters.RxGain); % all signal strengths in dBm

    % Threshold signals below noise level to reduce simulation time
    signalStrength(signalStrength < rxNoisePower) = -Inf;

    % Threshold signals that are not on same non-overlapping channel
    signalStrength(~(txChannels == rxChannels')) = -Inf;

    % Mask the transmitter-receiver links that are non-negligible to
    % simulate and get the linear indices
    nonnegligibleMask = signalStrength > -Inf;

    % Reset the non-negligible channels to create a new realization for the
    % current drop
    nonnegligibleIdx = find(nonnegligibleMask)';
```

```
% For each possible active link in a drop create a channel
% configuration. In this example the link and radio parameters are the
% same for all nodes.
tgaxChan = cell(numel(nonnegligibleIdx),1);
for i = 1:numel(nonnegligibleIdx)
    % Index of transmitter for a given link
    txIdx = mod(nonnegligibleIdx(i)-1,numTx)+1;
    % Channel configuration. The channel realization for each link is
    % different as the global random stream is used.
    tgaxChanBase = wlanTGaxChannel;
    tgaxChanBase.DelayProfile = 'Model-D';
    tgaxChanBase.NumTransmitAntennas = cfgHE{txIdx}.NumTransmitAntennas;
    tgaxChanBase.NumReceiveAntennas = PHYParameters.NumRxAntennas;
    tgaxChanBase.TransmitReceiveDistance = 10; % Distance in meters for NLOS
    tgaxChanBase.ChannelBandwidth = cfgHE{txIdx}.ChannelBandwidth;
    tgaxChanBase.LargeScaleFadingEffect = 'None';
    tgaxChanBase.EnvironmentalSpeed = 0; % m/s, stationary
    tgaxChanBase.SampleRate = fs;
    tgaxChanBase.NormalizeChannelOutputs = false;

    % Store in cell array and reset the channel to generate a new
    % response
    tgaxChan{i} = tgaxChanBase;
    reset(tgaxChan{i});
end

for txevent = 1:SimParameters.NumTxEventsPerDrop
    fprintf('    Running transmission event #%d/%d ...\n',txevent,SimParameters.NumTxEventsPe

    % Determine active transmitters and receivers with Clear Channel Assessment
    [activeTx,activeRx] = tgaxCCA(signalStrength,MACParameters.CCALevel);

    % Plot scenario and links
    if showScenarioPlot
        tgaxUpdatePlot(hGrid,txPositions,rxPositions,activeTx,activeRx,nonnegligibleMask,txCl
            sprintf('PHY System-Level Simulation, Drop #%d/%d, Transmission Event #%d/%d', .
            drop,SimParameters.NumDrops,txevent,SimParameters.NumTxEventsPerDrop));
    end

    % Extract elements for active links using activeTx and activeRx
    cfgHEActive = cfgHE(activeTx);
    associationActive = association(activeTx,activeRx);
    nonnegligibleMaskActive = nonnegligibleMask(activeTx,activeRx);
    signalStrengthActive = signalStrength(activeTx,activeRx);

    % Create array containing active channels
    tgaxChanActive = cell(size(associationActive));
    matchIdx = nonnegligibleIdx==find(activeTx&activeRx');
    tgaxChanActive(nonnegligibleMaskActive) = tgaxChan(any(matchIdx,1));

    % Generate a waveform for each non-negligible active link and
    % combine waveforms for each receiver
    [rxWavs,txPSDUActive,signalPower,interfPower] = tgaxGenerateRxWaveforms( ...
        cfgHEActive,tgaxChanActive,nonnegligibleMaskActive,signalStrengthActive,associationAc

    % Run PHY link simulation for each link and determine if the packet
    % has been decoded successfully. The estimated interference power
```

```matlab
            % is passed to the receiver in the place of an interference power
            % measurement algorithm.
            numActiveRxs = sum(activeRx);
            pktError = false(numActiveRxs,1);
            sinrMeas = nan(numActiveRxs,1);
            for rxIdx = 1:numActiveRxs
                [pktError(rxIdx),sinrMeas(rxIdx)] = tgaxModelPHYLink( ...
                    rxWavs{rxIdx},cfgHEActive{rxIdx},rxNoisePower,interfPower(rxIdx),txPSDUActive{rx}
            end

            % Store output for active receivers
            output.numPktErrors(activeRx) = output.numPktErrors(activeRx)+pktError;
            output.numPkts(activeRx) = output.numPkts(activeRx) + 1;
            output.sinrMeas(activeRx,txevent,drop) = sinrMeas;

            % Calculate the expected SINR at each receiver
            sinrEst = 10*log10(signalPower./(interfPower+10^((rxNoisePower-30)/10)));
            output.sinrEst(activeRx,txevent,drop) = sinrEst;
        end
    end

% Calculate average packet error rate
output.pktErrorRate = sum(output.numPktErrors)/sum(output.numPkts);

disp('Simulation complete')
disp(['Average packet error rate for transmitters: ' num2str(output.pktErrorRate)]);

end

rng(seed); % Restore random state
```

```
Running System Level Simulation ...
  Running drop #1/3 ...
    Running transmission event #1/2 ...
    Running transmission event #2/2 ...
  Running drop #2/3 ...
    Running transmission event #1/2 ...
    Running transmission event #2/2 ...
  Running drop #3/3 ...
    Running transmission event #1/2 ...
    Running transmission event #2/2 ...
Simulation complete
Average packet error rate for transmitters: 0.054152
```

PHY System-Level Simulation, Drop #3/3, Transmission Event #2/2
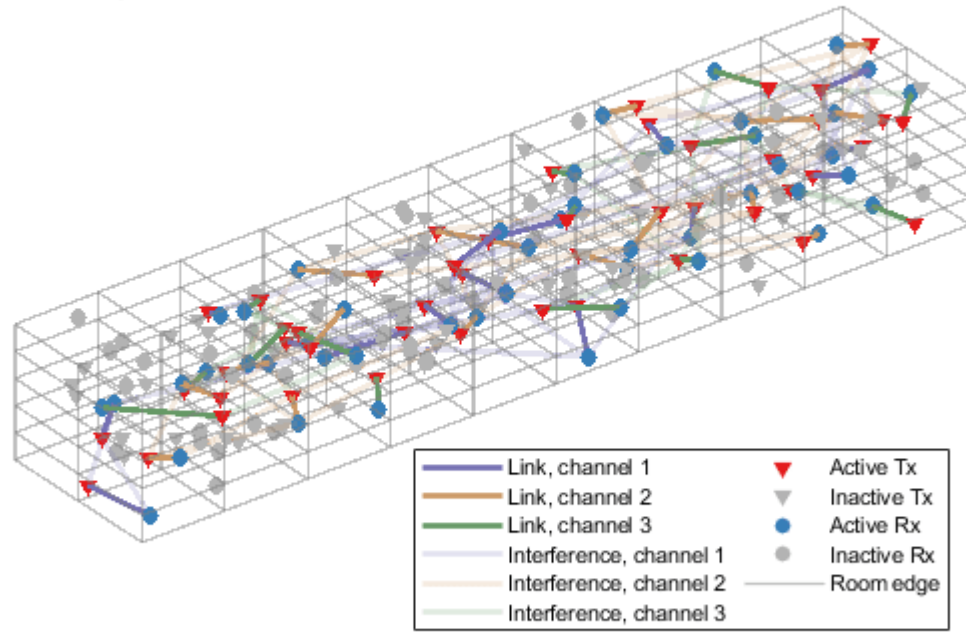
Further Exploration

The PHY-focused system-level simulation demonstrated in this example can be used to explore the impact of PHY-level parameters on system performance. For example, the plot below shows the network average packet error rate for different values of CCA threshold for 50 drops and 2 transmission events per drop.

Residential Scenario, MCS 3
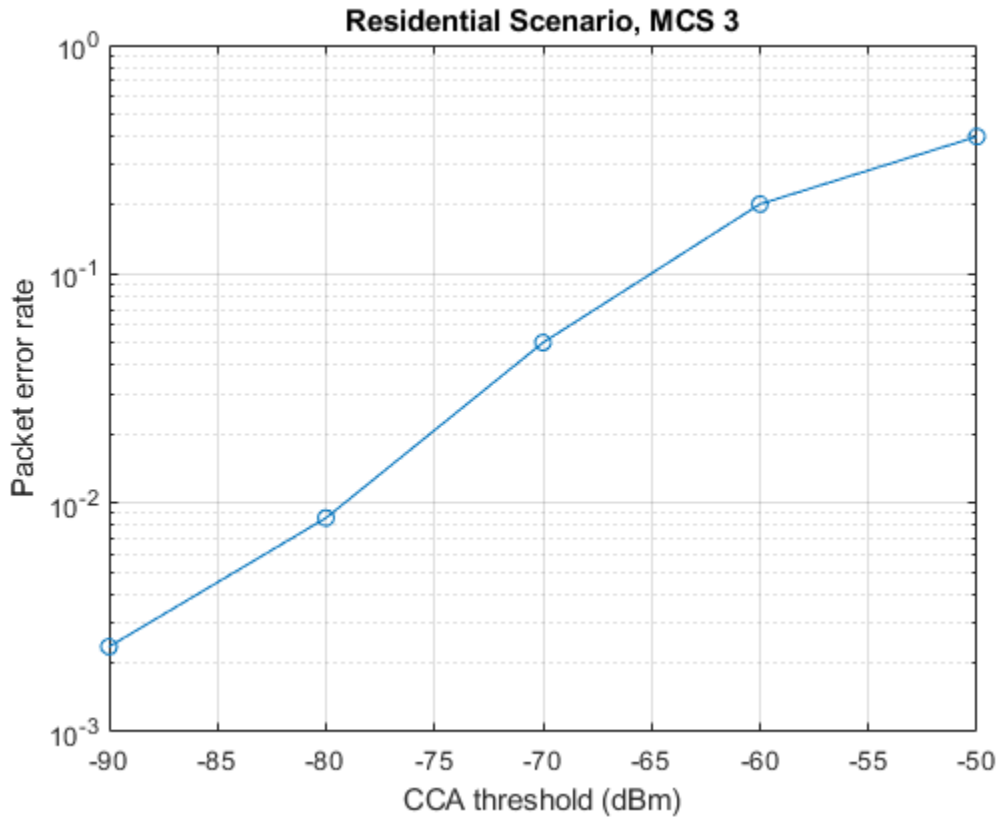
Appendix

This example uses the following helper functions:

- tgaxBuildResidentialGrid.m
- tgaxCalibrationCDF.m
- tgaxCCA.m
- tgaxDropNodes.m
- tgaxGenerateRxWaveforms.m
- tgaxModelPHYLink.m
- TGaxResidential.m
- tgaxUpdatePlot.m

Selected Bibliography

1   IEEE P802.11ax™/D4.1 Draft 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 6: Enhancements for High Efficiency WLAN.

2   IEEE 802.11-14/0980r16 - TGax Simulation Scenarios.

3   IEEE 802.11-14/0571r12 - 11ax Evaluation Methodology.

4   IEEE 802.11-14/0800r30 - Box 1 and Box 2 Calibration Results.

Physical Layer Abstraction for System-Level Simulation

This example demonstrates IEEE® 802.11ax™ physical layer abstraction for system-level simulation. A link quality model and link performance model based on the TGax evaluation methodology are presented and validated by comparing with published results.

Introduction

Modeling the full physical layer processing at each transmitter and receiver when simulating large networks is computationally expensive. Physical layer abstraction, or link-to-system mapping is a method to run simulations in a timely manner by accurately predicting the performance of a link in a computationally efficient way.

This example demonstrates physical layer abstraction for the data portion of an 802.11ax [1 on page 7-0   ] packet based on the TGax evaluation methodology [2 on page 7-0   ].

There are two parts to the physical layer abstraction model [3 on page 7-0   , 4 on page 7-0   ]:

- **The link quality model** calculates the post-equalizer signal to interference and noise ratio (SINR) per subcarrier. For a receiver, this is based on the location and transmission characteristics of the transmitter of interest, and interfering transmissions, and the impact of large- and small-scale fading.

- **The link performance model** predicts the instantaneous packet error rate (PER), and therefore transmission success of an individual packet, given the SINR per subcarrier and coding parameters used for the transmission.

The example is split into two parts:

- Part A on page 7-0   demonstrates the link quality model used to obtain the SINR per subcarrier and validates it by comparing the results for a residential scenario as per the box 2 tests in the TGax evaluation methodology. The objective of box 2 tests is to align the distribution of small and large scale fading channels with MIMO configurations of TGax contributors.

- Part B on page 7-0   demonstrates the link performance model used to estimate the PER, and compares the result of using the abstraction against a link-level simulation with a fading TGax channel model as per the box 0 tests in the TGax evaluation methodology. The objective of box 0 tests is to align PHY abstractions of TGax contributors.

Part A - Link Quality Model

The link quality model implements the box 2 SINR equation from the TGax Evaluation Methodology. The multiple-input-multiple-output (MIMO) SINR per subcarrier (index m) and spatial stream (index j) between the transmitter and receiver of interest is given by

$$SINR_{RX}^{TX}(m, j) = \frac{S_{RX}^{TX}(m, j)}{Is_{RX}^{TX}(m, j) + Io_{RX}^{TX}(m, j) + N(m, j)}.$$

The SINR takes into account the path-loss and fading channels between all transmitters and the receiver, and precoding applied at the transmitters. The power of the signal of interest is given by

$$S_{RX}^{TX}(m, j) = P_{RX}^{TX} \left| [\mathbf{T}_{RX}(m)]_j^H \mathbf{H}_{RX}^{TX}(m) \left[\mathbf{W}^{TX}(m) \right]_j \right|^2,$$

where P_{RX}^{TX} is the received power of the signal of interest, $\mathbf{T}_{RX}$ is the linear receiver filter, $\mathbf{H}_{RX}^{TX}$ is the channel matrix between the transmitter and receiver of interest, and $\mathbf{W}^{TX}$ is the precoding matrix applied at the transmitter.

The power of intra-user interference is given by

$$Is_{RX}^{TX}(m, j) = P_{RX}^{TX} \left\| [\mathbf{T}_{RX}(m)]_j^H \mathbf{H}_{RX}^{TX}(m) \mathbf{W}^{TX}(m) \right\|^2 - S_{RX}^{TX}(m, j).$$

The power of inter-user interference is given by

$$Io_{RX}^{TX}(m, j) = \sum_k \sum_{i \in \Omega(k)} P_{RX}^{TX} \left\| [\mathbf{T}_{RX}(m)]_j^H \mathbf{H}_{RX}^{TX_i}(m) \mathbf{W}^{TX_i}(m) \right\|^2,$$

where $\Omega(k)$ is the set of interfering transmitters in the kth basic service set (BSS)

The noise power is given by

$$N(m, j) = \left\| [\mathbf{T}_{RX}(m)]_j \right\|^2 N_0,$$

where N_0 is the noise power spectral density.

Generate a Channel Matrix per Subcarrier

The link quality model requires a channel matrix per subcarrier. Calculate the channel matrix from the path gains returned from the fading channel model `wlanTGaxChannel` by using the `helperPerfectChannelEstimate()` helper function. Efficiently generate path gains by setting the `ChannelFiltering` property of `wlanTGaxChannel` to `false`.

```
sprev = rng('default'); % Seed random number generator and store previous state

% Get an HE OFDM configuration: 80 MHz channel bandwidth, 3.2 us guard
% interval
ofdmInfo = wlanHEOFDMInfo('HE-Data','CBW80',3.2);
k = ofdmInfo.ActiveFrequencyIndices;

% Configure channel to return path gains for one OFDM symbol
tgax = wlanTGaxChannel;
tgax.ChannelBandwidth = 'CBW80';
tgax.SampleRate = 80e6; % MHz
tgax.ChannelFiltering = false;
tgax.NumSamples = ofdmInfo.FFTLength+ofdmInfo.CPLength;

% Generate channel matrix per subcarrier for signal of interest
pathGains = tgax();    % Get path gains
chanInfo = info(tgax); % Get channel info for filter coefficients
chanFilter = chanInfo.ChannelFilterCoefficients;
Hsoi = helperPerfectChannelEstimate(pathGains,chanFilter, ...
    ofdmInfo.FFTLength,ofdmInfo.CPLength,ofdmInfo.ActiveFFTIndices);
```

```
% Generate channel matrix per subcarrier for interfering signal
reset(tgax); % Get a new channel realization
pathGains = tgax();
Hint = helperPerfectChannelEstimate(pathGains,chanFilter, ...
    ofdmInfo.FFTLength,ofdmInfo.CPLength,ofdmInfo.ActiveFFTIndices);
```

SINR Calculation

Calculate and visualize the post-equalizer SINR per subcarrier with the `calculateSINR` and `plotSINR` helper functions.

```
Psoi = -20; % Signal of interest received power (dBm)
Pint = -45; % Interfering signal received power (dBm)
N0 = -85;   % Noise power (dBm)
W = ones(ofdmInfo.NumTones,1); % Precoding matrix (assume no precoding)

sinr = calculateSINR(Hsoi,db2pow(Psoi-30),W,db2pow(N0-30),{Hint},db2pow(Pint-30),{W});

plotSINR(sinr,Hsoi,Psoi,Hint,Pint,N0,k);
```
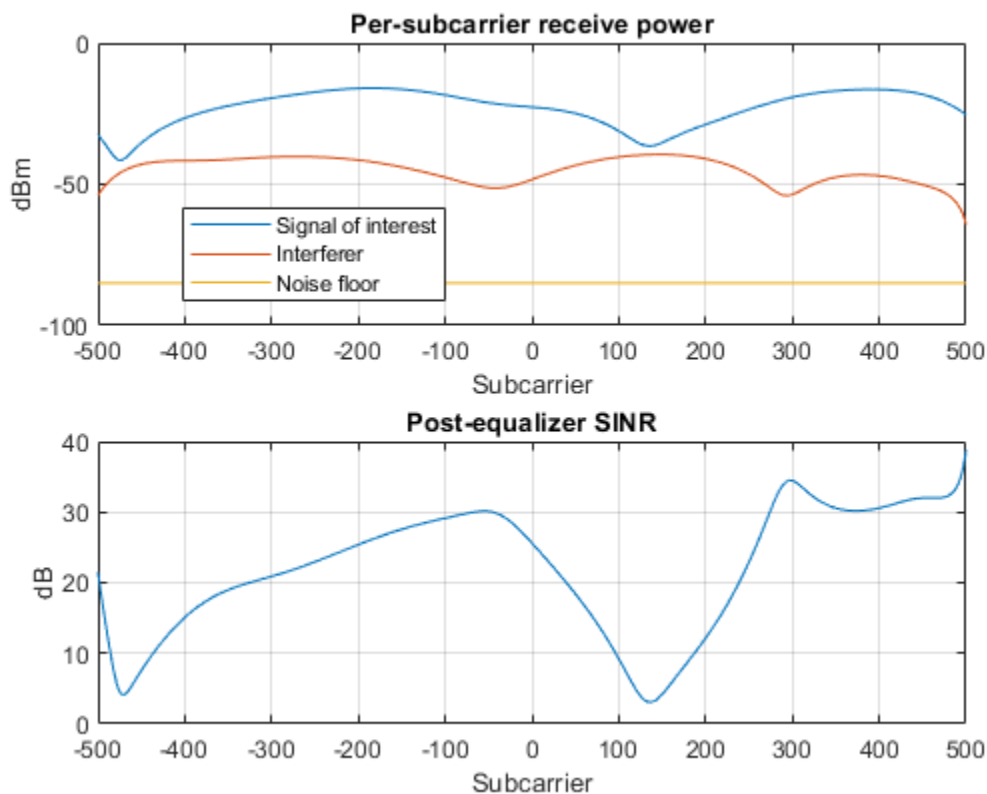

TGax Evaluation Methodology Box 2 - Verify SINR Calibration

This section validates the SINR calculation by comparing the cumulative density function (CDF) of per-subcarrier SINRs with calibration results provided by the TGax working group. We compare the SINR calculation with results published by TGax [5 on page 7-0   ] for box 2, test 3: "downlink transmission per basic channel access rule" for the residential scenario.

For more information about the scenario, and for the results of long-term SINR calibration see the "802.11ax PHY-Focused System-Level Simulation" on page 7-72 example.

The major simulation parameters are defined as either belonging to Physical Layer (PHY), Medium Access Control Layer (MAC), scenario, or simulation. In this example the PHY and MAC parameters are assumed to be the same for all nodes.

```
sinrCalibration = ☑ ; % Disable box 2 calibration
if sinrCalibration

PHYParams = struct;
PHYParams.TxPower = 20;      % Transmitter power in dBm
PHYParams.TxGain = 0;        % Transmitter antenna gain in dBi
PHYParams.RxGain = -2;       % Receiver antenna gain in dBi
PHYParams.NoiseFigure = 7;   % Receiver noise figure in dB
PHYParams.NumTxAntennas = 1; % Number of transmitter antennas
PHYParams.NumSTS = 1;        % Number of space-time streams
PHYParams.NumRxAntennas = 1; % Number of receiver antennas
PHYParams.ChannelBandwidth = 'CBW80'; % Bandwidth of system
PHYParams.TransmitterFrequency = 5e9; % Transmitter frequency in Hz

MACParams = struct;
MACParams.NumChannels = 3; % Number of non-overlapping channels
MACParams.CCALevel = -70;  % Transmission threshold in clear channel assessment algorithm (dBm)
```

The scenario parameters define the size and layout of the residential building as per [6 on page 7-0]. Note only one receiver (STA) can be active at any given time.

```
ScenarioParams = struct;
ScenarioParams.BuildingLayout = [10 2 5]; % Number of rooms in [x,y,z] directions
ScenarioParams.RoomSize = [10 10 3];      % Size of each room in metres [x,y,z]
ScenarioParams.NumRxPerRoom = 1;          % Number of receivers per room.
```

The `NumDrops` and `NumTxEventsPerDrop` parameters control the length of the simulation. In this example, these parameters are configured for a short simulation. A 'drop' randomly places transmitters and receivers within the scenario and selects the channel for a BSS. A 'transmission' event randomly selects transmitters and receivers for transmissions according to the basic clear channel assessment (CCA) rules defined in the evaluation methodology.

```
SimParams = struct;
SimParams.Test = 3; % Downlink transmission per basic channel access rule
SimParams.NumDrops = 3;
SimParams.NumTxEventsPerDrop = 2;
```

The function `box2Simulation` runs the simulation by performing these steps:

1 Randomly drop transmitters (APs) and receivers (STAs) within the scenario.
2 Calculate the large-scale path loss and generate frequency-selective TGax fading channels for all non-negligible links.
3 For each transmission event, determine active transmitters and receivers based on CCA rules.
4 Calculate and return the SINR per subcarrier and the effective SINR for each active receiver as per box 2, test 3 in the TGax evaluation methodology.

```
box2Results = box2Simulation(PHYParams,MACParams,ScenarioParams,SimParams);
```

Plot the CDF of the SINR per subcarrier and effective SINR (as defined in box 2, test 3) against submitted calibration results.

```
tgaxCalibrationCDF(box2Results.sinr(:), ...
    ['SS1Box2Test' num2str(SimParams.Test) 'A'],'CDF of SINR per subcarrier');
tgaxCalibrationCDF(box2Results.sinrEff(:), ...
    ['SS1Box2Test' num2str(SimParams.Test) 'B'],'CDF of effective SINR per reception');

end
```

```
  Running drop #1/3 ...
    Generating 3518 fading channel realizations ...
    Running transmission event #1/2 ...
    Running transmission event #2/2 ...
  Running drop #2/3 ...
    Generating 3366 fading channel realizations ...
    Running transmission event #1/2 ...
    Running transmission event #2/2 ...
  Running drop #3/3 ...
    Generating 3750 fading channel realizations ...
    Running transmission event #1/2 ...
    Running transmission event #2/2 ...
```

Calibration: CDF of effective SINR per reception
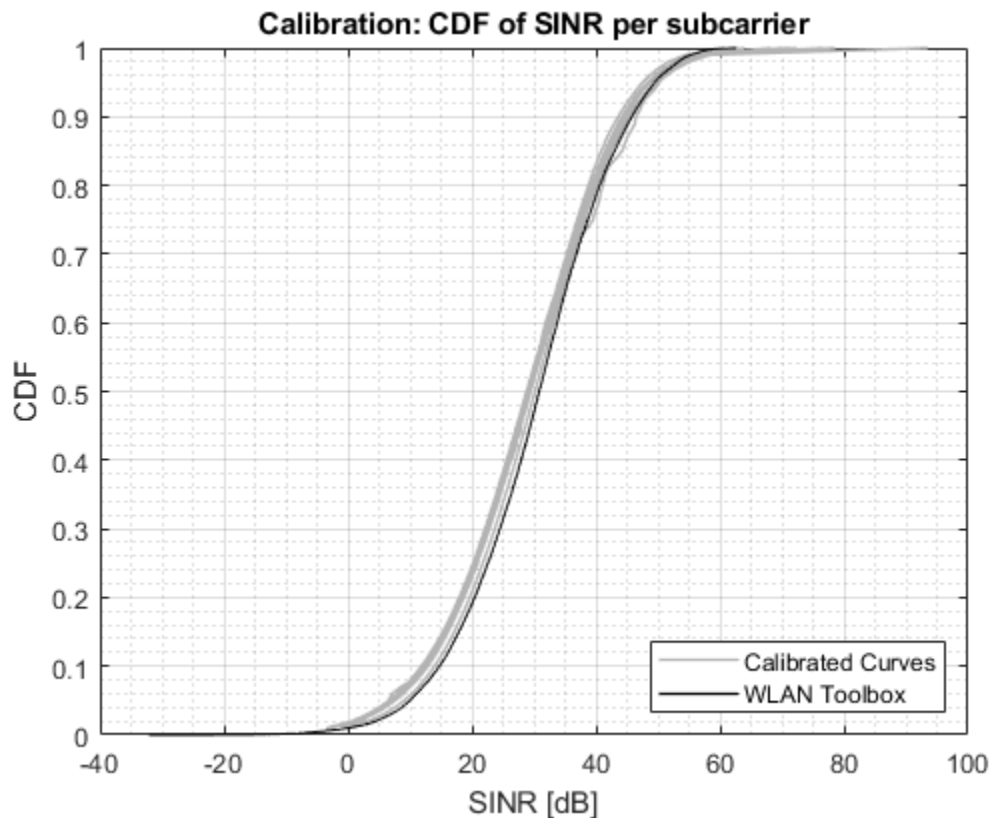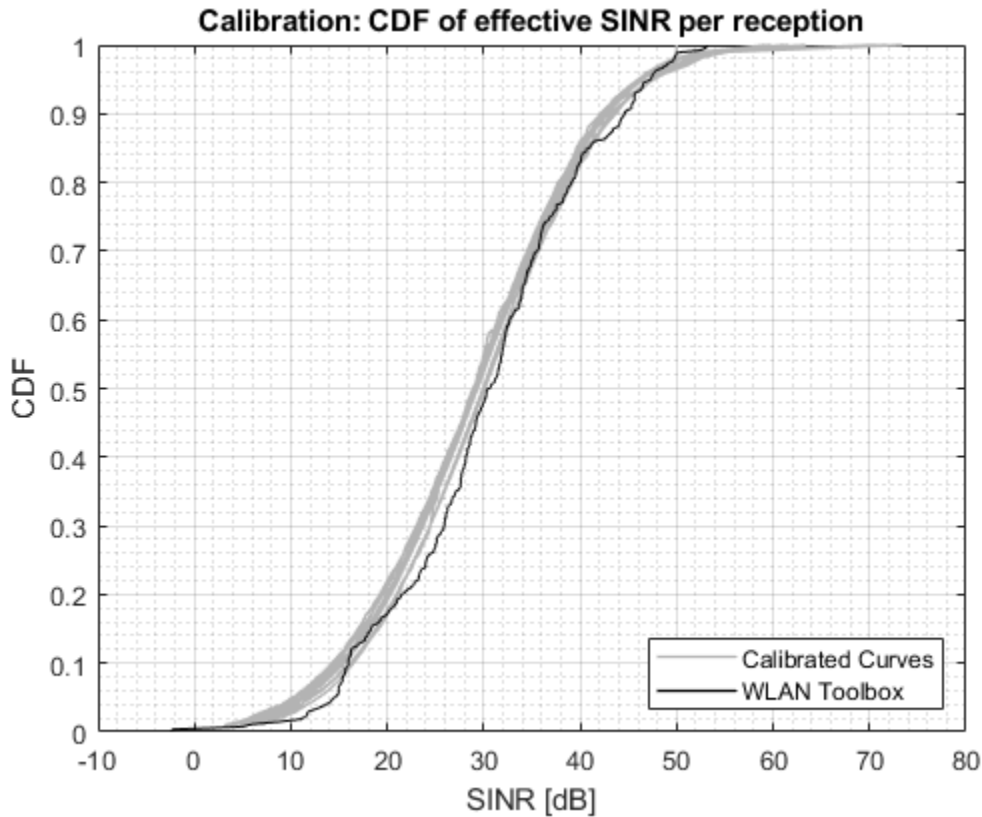
Increase the number of drops for a more accurate comparison.

Part B - Link Performance Model

The link performance model predicts the instantaneous PER given the SINR per subcarrier calculated in Part A on page 7-0   and coding parameters used for the transmission.

Effective SINR mapping and averaging is used to compress the post-equalizer SINR per subcarrier into a single effective SNR. The effective SNR is the SNR that provides an equivalent PER performance with an additive white Gaussian noise (AWGN) channel as with the fading channel. A pre-computed lookup table, generated with WLAN Toolbox™, provides the PER for an SNR under an AWGN channel for a given channel coding, modulation scheme, and coding rate. Once the PER is obtained, a random variable determines whether the packet has been received in error.
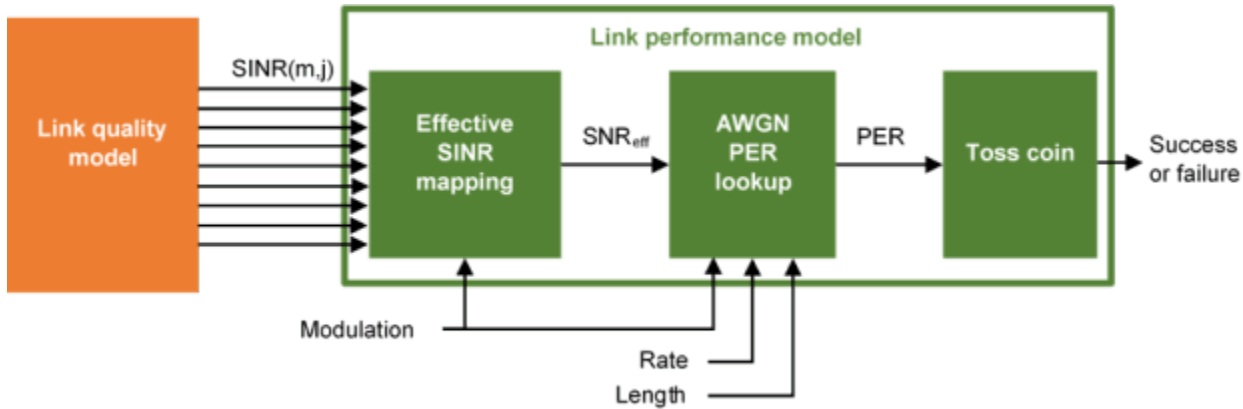
The TGax evaluation methodology PER estimation procedure is used in this example considering a single interference event.

The effective SINR is calculated using the received bit mutual information rate (RBIR) mapping function;

$$SINR_{eff} = \alpha\phi^{-1}\left\{\frac{1}{N_{ss}N_{sc}}\Sigma_{j=1}^{N_{ss}}\Sigma_{m=1}^{N_{sc}}\phi\left(\frac{SINR_{(m,j)}}{\beta}, M\right), M\right\}.$$

- $\phi(x, M)$ is the RBIR mapping function, which transforms the SINR of each subcarrier to an "information measure" for the modulation scheme M. The RBIR mapping function for BPSK, QPSK, 16QAM, 64QAM and 256QAM is provided in [7 on page 7-0   ].

- $\phi^{-1}(x, M)$ is the inverse RBIR mapping function, which transforms an "information measure" back to the SNR domain.

- N_{ss} is the number of spatial-streams.

- N_{sc} is the number of subcarriers.

- $SINR$ is the post-equalizer SINR of the mth subcarrier and jth spatial-stream.

- α and β are tuning parameters. The TGax evaluation methodology assumes no tuning therefore in this example we assume these are set to 1.

The PER for a reference data length PER_{PL_0} is obtained by looking up the appropriate AWGN table, $\overline{LUT}$, given the modulation and coding scheme (MCS), channel coding scheme, and reference data length (PL_0)

$$PER_{PL_0} = \overline{LUT}\left(SNR_{eff}; MCS, \text{ coding scheme, } PL_0\right),$$

where the reference data length depends on the channel coding and data length for the transmission PL.

$$PL_0 = \begin{cases} 32 \text{ } bytes, & BCC \text{ and } PL < 400 \text{ } bytes \\ 1458 \text{ } bytes, & BCC \text{ and } PL \geq 400 \text{ } bytes \\ 1458 \text{ } bytes, & LDPC \end{cases}.$$

The final estimated PER is then adjusted for the data length:

$$PER_{PL} = \left(1 - PER_{PL_0}\right)^{\frac{PL}{PL0}}.$$

The described method assumes the SINR is constant for the duration of the packet. The TGax evaluation methodology describes techniques to deal with time-varying interference and estimate the error rate of media access control protocol data units (MPDUs) within an aggregate MPDU (A-MPDU).

Calculate Effective SINR

Calculate the effective SINR and the PER using the `tgaxLinkPerformanceModel` example helper object.

```
Abstraction = tgaxLinkPerformanceModel;
```

The `effectiveSINR` method calculates the effective SINR given the modulation scheme and post equalizer SINR per subcarrier and spatial stream.

```
format =  HE SU           ▼  ;
mcs =     6                          ; % MCS 6 is 64-QAM
[snreff,rbir_sc,rbir_av] = effectiveSINR(Abstraction,sinr,format,mcs);
```

The RBIR (information measure) per subcarrier obtained by mapping the SINR per subcarrier, and the average RBIR are shown in the first figure subplot. The effective SINR per subcarrier is obtained by inverse mapping the average RBIR and is shown in the second subplot.

```
plotRBIR(sinr,snreff,rbir_av,rbir_sc,k);
```
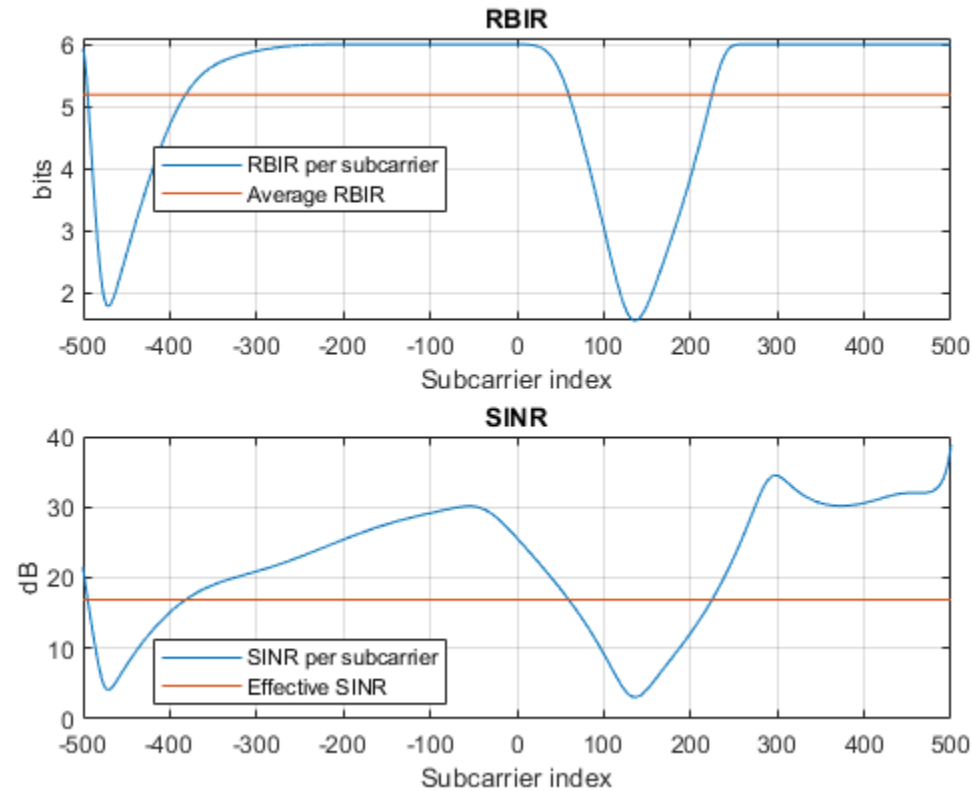
Estimate Packet Error Rate

Given the effective SNR, estimate the PER by linearly interpolating and extrapolating a pre-computed AWGN link-level curve in the logarithmic domain, and adjusting for the data length. The `estimatePER` method returns the final PER, `per`, and the AWGN lookup table used, `lut` .

```
channelCoding = LDPC              ▼  ;
dataLength =     1458             ; % Bytes
[per,~,~,lut] = estimatePER(Abstraction,snreff,format,mcs,channelCoding,dataLength);

% Plot the AWGN lookup table and the estimated PER
figure;
semilogy(lut(:,1),lut(:,2));
grid on
hold on
plot(snreff,per,'d');
legend('AWGN LDPC lookup table','Estimated PER from effective SINR')
title('Packet error rate')
xlabel('SNR (dB)')
ylabel('PER')
```
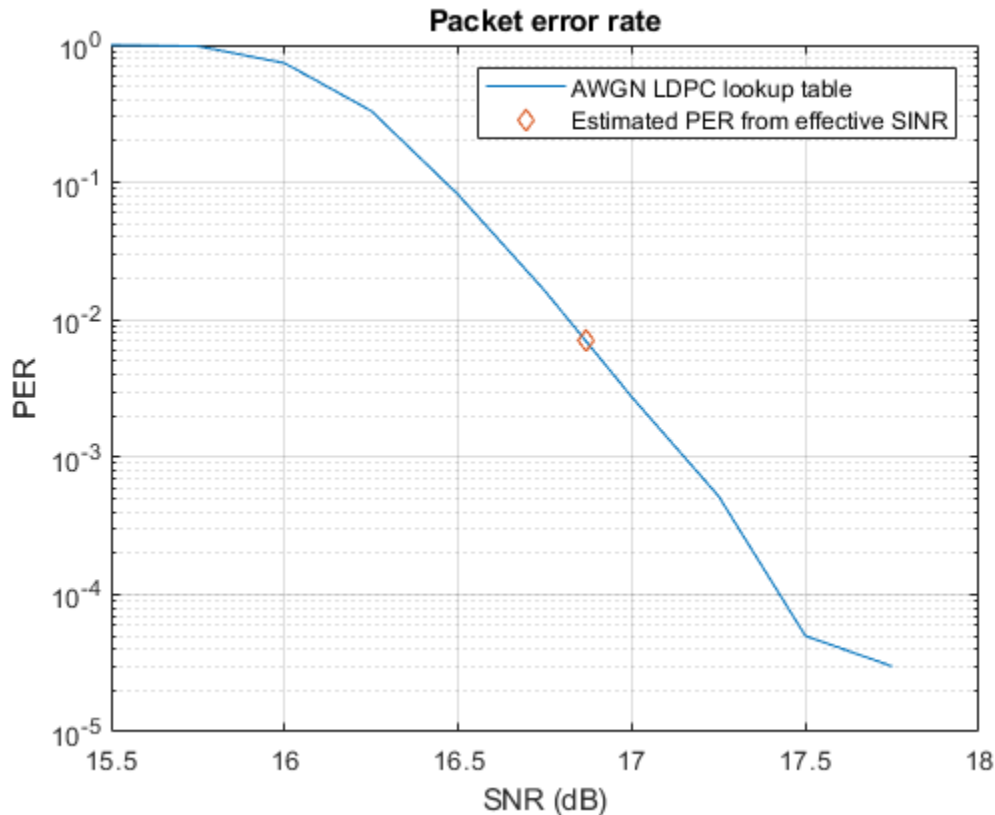
TGax Evaluation Methodology Box 0 - Verify Effective SNR vs PER Performance

To verify the entire physical-layer abstraction method, the PER from a link-level simulation is compared with the PER estimates using the abstraction. This follows steps 2 and 3 of box 0 testing in the TGax Evaluation Methodology. An 802.11ax single-user link is modeled with perfect synchronization, channel estimation, and no impairments apart from a fading TGax channel model and AWGN. Only errors within data portion of a packet are considered.

In this example the SNRs to simulate are selected based on the MCS, number of transmit and receive antennas, and channel model for the given PHY configuration. The number of space-time streams is assumed to equal the number of transmit antennas. The simulation is configured for a short run; for more meaningful results you should increase the number of packets to simulate.

```
verifyAbstraction = ☑ ; % Disable box 0 simulation
if verifyAbstraction

% Simulation Parameters
mcs = [4 8]; % Vector of MCS to simulate between 0 and 11
numTxRx = [1 1]; % Matrix of MIMO schemes, each row is [numTx numRx]
chan = "Model-D"; % String array of delay profiles to simulate
maxNumErrors = 1e1;  % The maximum number of packet errors at an SNR point
maxNumPackets = 1e2; % The maximum number of packets at an SNR point

% Fixed PHY configuration for all simulations
cfgHE = wlanHESUConfig;
cfgHE.ChannelBandwidth = 'CBW20'; % Channel bandwidth
```

```
cfgHE.APEPLength = 1000;          % Payload length in bytes
cfgHE.ChannelCoding = 'LDPC';     % Channel coding

% Generate a structure array of simulation configurations. Each element is
% one SNR point to simulate.
simParams = getBox0SimParams(chan,numTxRx,mcs,cfgHE,maxNumErrors,maxNumPackets);

% Simulate each configuration
results = cell(1,numel(simParams));
% parfor isim = 1:numel(simParams) % Use 'parfor' to speed up the simulation
for isim = 1:numel(simParams)
    results{isim} = box0Simulation(simParams(isim));
end
```

The suitability of the abstraction is determined by comparing the PER calculated by link-level simulation and abstraction. The first figure compares the PERs at each SNR simulated.

```
plotPERvsSNR(simParams,results);
```

The second figure compares the number of successfully decoded link-level simulation packets with an effective SNR against the reference AWGN curve. If the abstraction is successful the PER should follow the AWGN curve.

```
plotPERvsEffectiveSNR(simParams,results);
end
```

```
Model-D 1-by-1, MCS 4, SNR 11 completed after 14 packets, PER:0.78571
Model-D 1-by-1, MCS 4, SNR 15 completed after 22 packets, PER:0.5
Model-D 1-by-1, MCS 4, SNR 19 completed after 100 packets, PER:0.05
Model-D 1-by-1, MCS 4, SNR 23 completed after 100 packets, PER:0.02
Model-D 1-by-1, MCS 4, SNR 27 completed after 100 packets, PER:0
Model-D 1-by-1, MCS 8, SNR 21.5 completed after 13 packets, PER:0.84615
Model-D 1-by-1, MCS 8, SNR 25.5 completed after 23 packets, PER:0.47826
Model-D 1-by-1, MCS 8, SNR 29.5 completed after 100 packets, PER:0.03
Model-D 1-by-1, MCS 8, SNR 33.5 completed after 100 packets, PER:0.02
Model-D 1-by-1, MCS 8, SNR 37.5 completed after 100 packets, PER:0
```

Abstracted vs Simulated PHY, 1x1, Model-D

Simulated link PER vs effective SNR, 1x1, Model-D
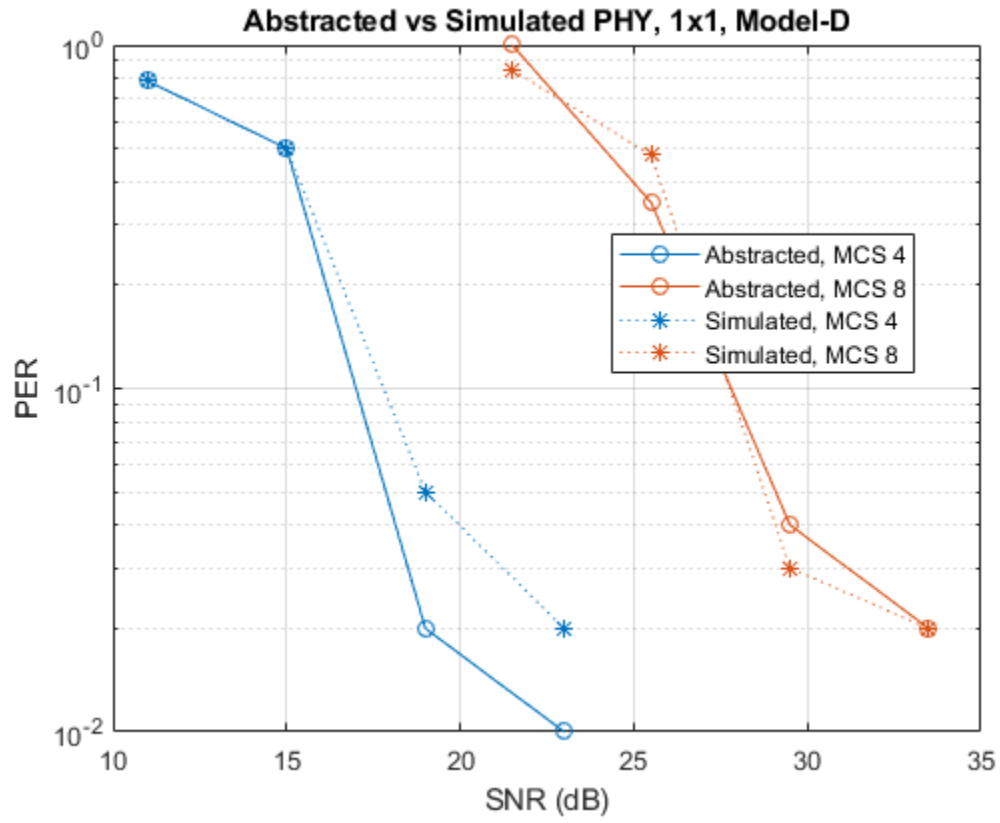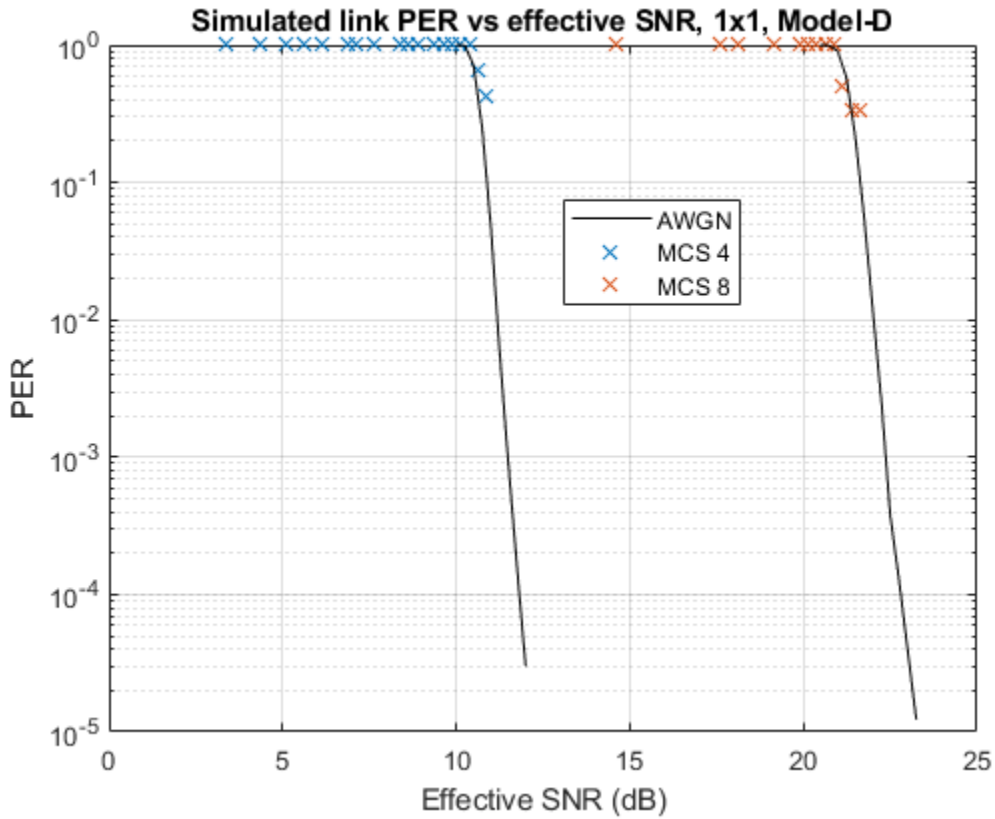
```
rng(sprev) % Restore random state
```

In this example the tuning parameters α and β are set to 1. These could be tuned to further improve the accuracy of the abstraction if desired. The results when simulating 1000 packet errors or 100000 packets for MCS 0 to 9 for a 1458-byte packet without tuning is shown.

Abstracted vs Simulated PHY, 1x1, Model-D

Simulated link PER vs effective SNR, 1x1, Model-D
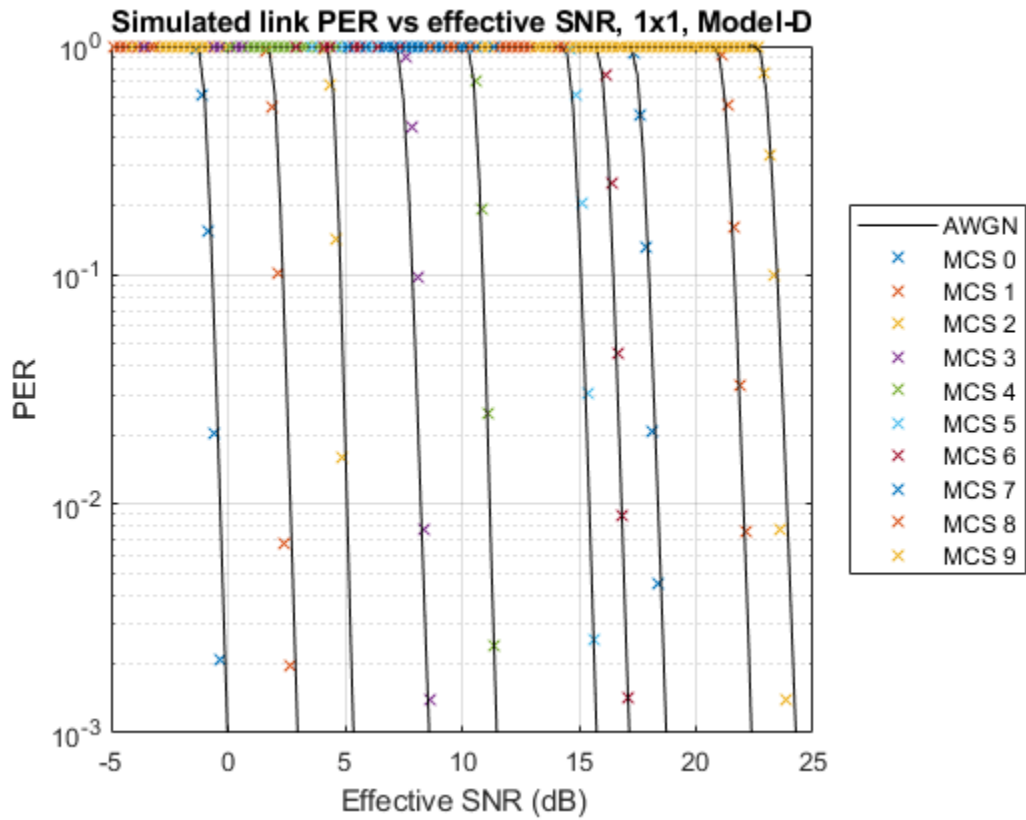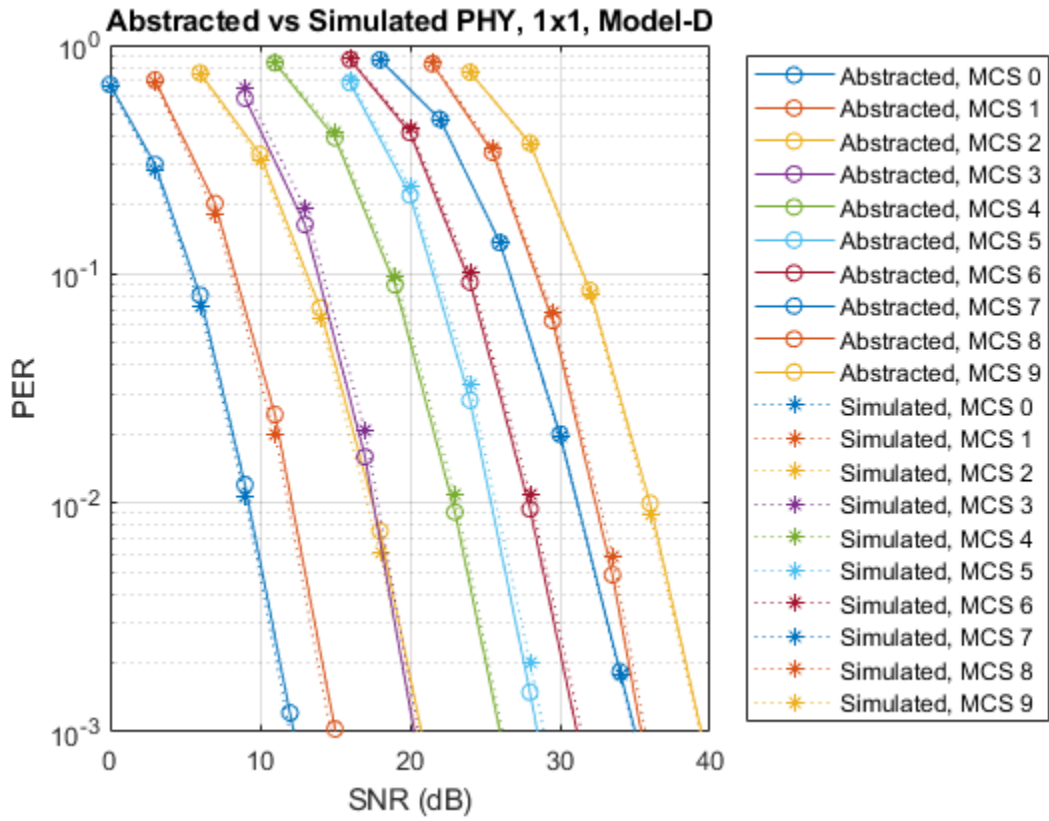
Further Exploration

To see how the 802.11ax physical layer abstraction described in this example can be used in a system-level simulation, see the "802.11ax System-Level Simulation with Physical Layer Abstraction" on page 7-52 example.

Selected Bibliography

1   IEEE P802.11ax™/D4.1 Draft 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 6: Enhancements for High Efficiency WLAN.

2   IEEE 802.11-14/0571r12 - 11ax Evaluation Methodology.

3   Brueninghaus, Karsten, et al. "Link performance models for system level simulations of broadband radio access systems." *2005 IEEE 16th International Symposium on Personal, Indoor and Mobile Radio Communications*. Vol. 4. IEEE, 2005.

4   Mehlführer, Christian, et al. "The Vienna LTE simulators-Enabling reproducibility in wireless communications research." *EURASIP Journal on Advances in Signal Processing* 2011.1 (2011): 29.

5   IEEE 802.11-14/0800r30 - Box 1 and Box 2 Calibration Results.

6   IEEE 802.11-14/0980r16 - TGax Simulation Scenarios.

7   IEEE 802.11-14/1450r0 - Box 0 Calibration Results

802.11ax Downlink Throughput Comparison of OFDM and OFDMA Through System-level Simulation

This example demonstrates how to model a multinode downlink orthogonal frequency-division multiple access (OFDMA) IEEE® 802.11ax™ [1] scenario using SimEvents®, Stateflow®, and WLAN Toolbox™. In OFDMA, independent groups of subcarriers are allocated to distinct users to realize simultaneous transmission. This multiplexing technique provides significant advantages in terms of spectrum efficiency, contention, latency, and network jitter. This example provides a model to enable downlink (DL) OFDMA communication in a 802.11ax network. The network topology consists of an access point (AP) and four linearly placed stations. The model enables you to configure multiple applications with same access category (AC) for different destination stations. In this example, the allocation of resource units (RUs) to the stations is fixed based on the number of users. The AP schedules transmissions to the stations in a round-robin fashion. The derived plot shows that the DL throughput at the AP is higher with OFDMA as compared to OFDM.

802.11ax OFDMA

The IEEE 802.11ax introduced significant enhancements over the existing 802.11ac standard [2]. One of the key improvements is the OFDMA, which is an extension of orthogonal frequency division multiplexing (OFDM) digital modulation technology into a multiuser environment. The principle of OFDMA is to efficiently use the available frequency space. OFDMA partitions the channel bandwidth into multiple mutually exclusive subbands, called RUs. By partitioning the channel bandwidth, multiple users can access the air interface simultaneously. As a result, concurrent transmissions of small frames to multiple users is possible at the same time. For example, a conventional 20 MHz channel can be partitioned into a maximum of nine subchannels. Subsequently, using OFDMA, an 802.11ax AP can simultaneously transmit small frames to nine 802.11ax stations. The simultaneous transmission of frames not only cuts down on excessive overhead at the MAC but also minimizes the contention overhead. In OFDMA, the allocation of RUs is completely controlled by the AP. The 802.11ax standard specifies two types of OFDMA transmissions, namely downlink (DL) and uplink (UL) OFDMA.

- **DL OFDMA:** AP transmits data to multiple stations simultaneously using a different RU for each station.
- **UL OFDMA:** Multiple stations transmit data to AP at the same time with each station using a different RU.

Comparison of OFDM and OFDMA

This section illustrates the difference between OFDM and OFDMA. In this figure, the 802.11n/ac/ax AP transmits DL to 4 OFDM stations independently over time. The entire channel bandwidth is used for the DL communication between the AP and a single OFDMA station. The same holds true for any uplink transmission from a single 802.11n/ac/ax client to the 802.11n/ac/ax AP.
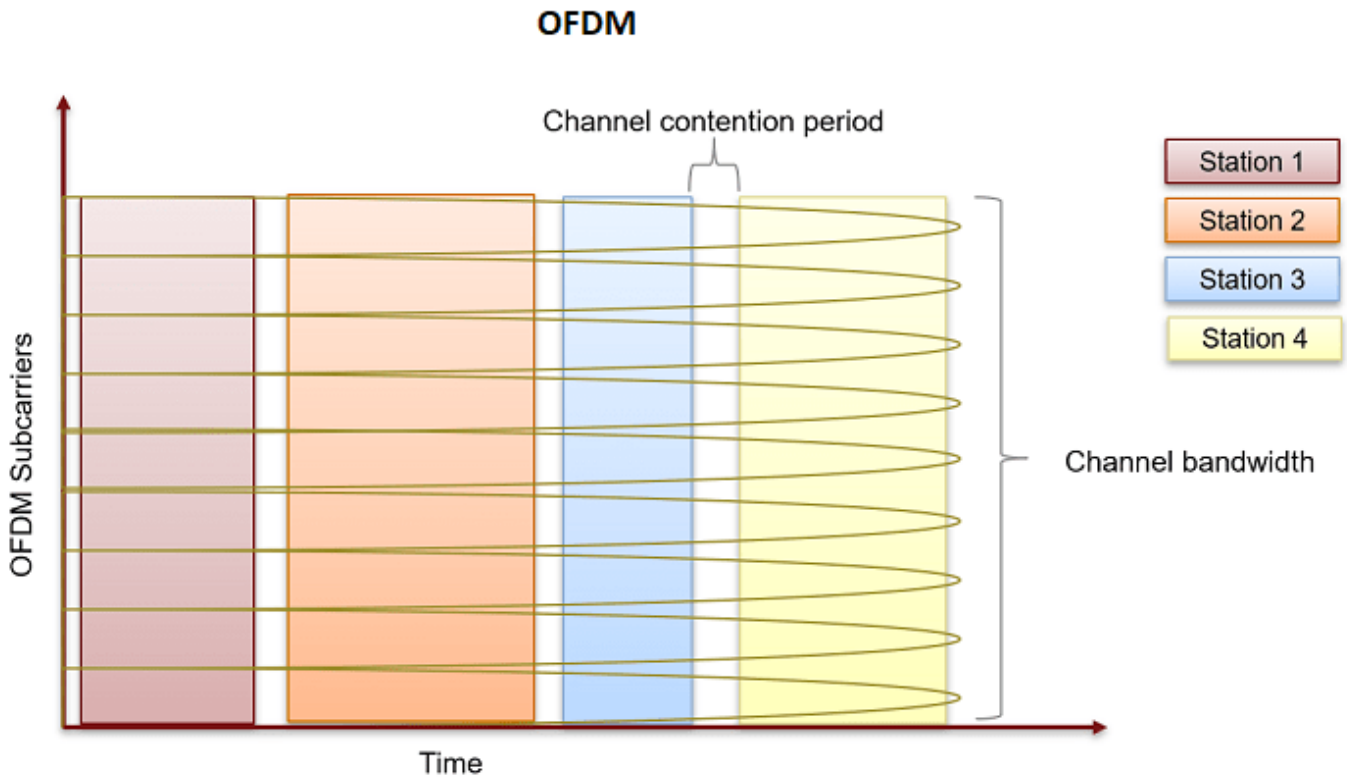
OFDM

This figure shows that when OFDMA is used, the 802.11ax AP partitions the channel bandwidth into RUs for multiple OFDMA stations on a continuous basis for simultaneous DL transmissions. By partitioning the channel bandwidth, OFDMA makes efficient use of the available frequency spectrum, resulting in reduced MAC contention and PHY preamble overhead.
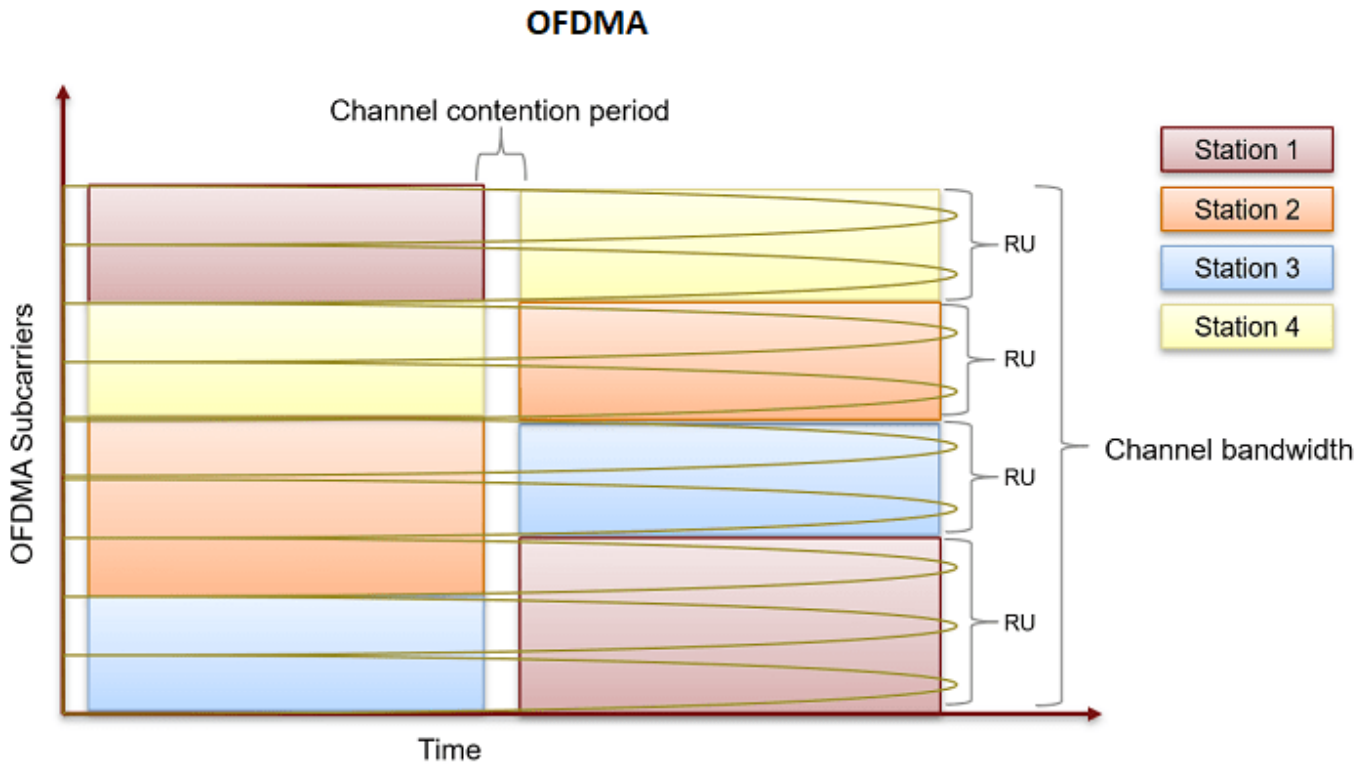
An 802.11ax AP can also coordinate with 802.11ax OFDMA stations for simultaneous uplink transmissions.

Model 802.11ax Network with OFDMA

This example demonstrates DL OFDMA communication in an 802.11ax network with one AP and four stations. These stations are linearly placed with AP located at the start. These stations implement carrier-sense multiple access with collision avoidance (CSMA/CA) with physical carrier sense. The physical carrier sensing uses the clear channel assessment (CCA) mechanism to determine whether the medium is busy before transmitting. This example is an enhancement over the "802.11 MAC and Application Throughput Measurement" on page 7-27 example. The enhancements are related to adding 802.11ax DL OFDMA support in the MAC and PHY library blocks. However, the `Node Position Allocator (NPA)`, `Visualizer` and the `Application Traffic Generator` blocks are the same as in the "802.11 MAC and Application Throughput Measurement" on page 7-27 example.
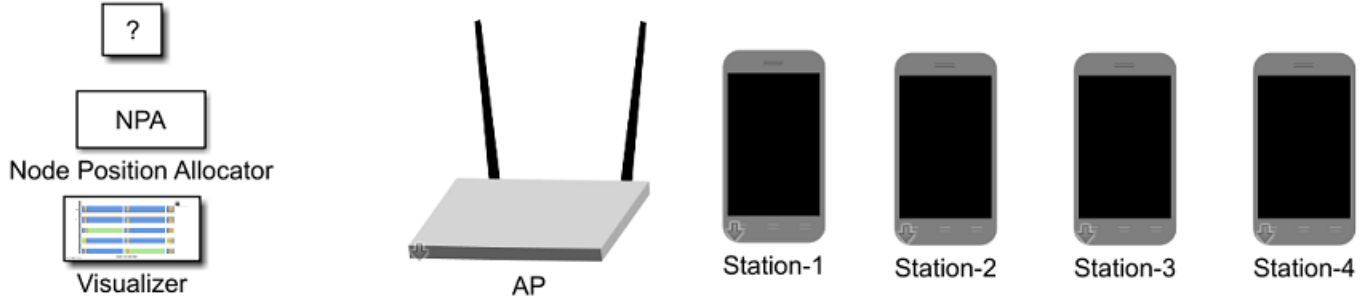
Copyright 2019 The MathWorks, Inc.

The components of a WLAN node are shown in this figure. The information is retrieved by pressing the arrow button for each node in the above figure.
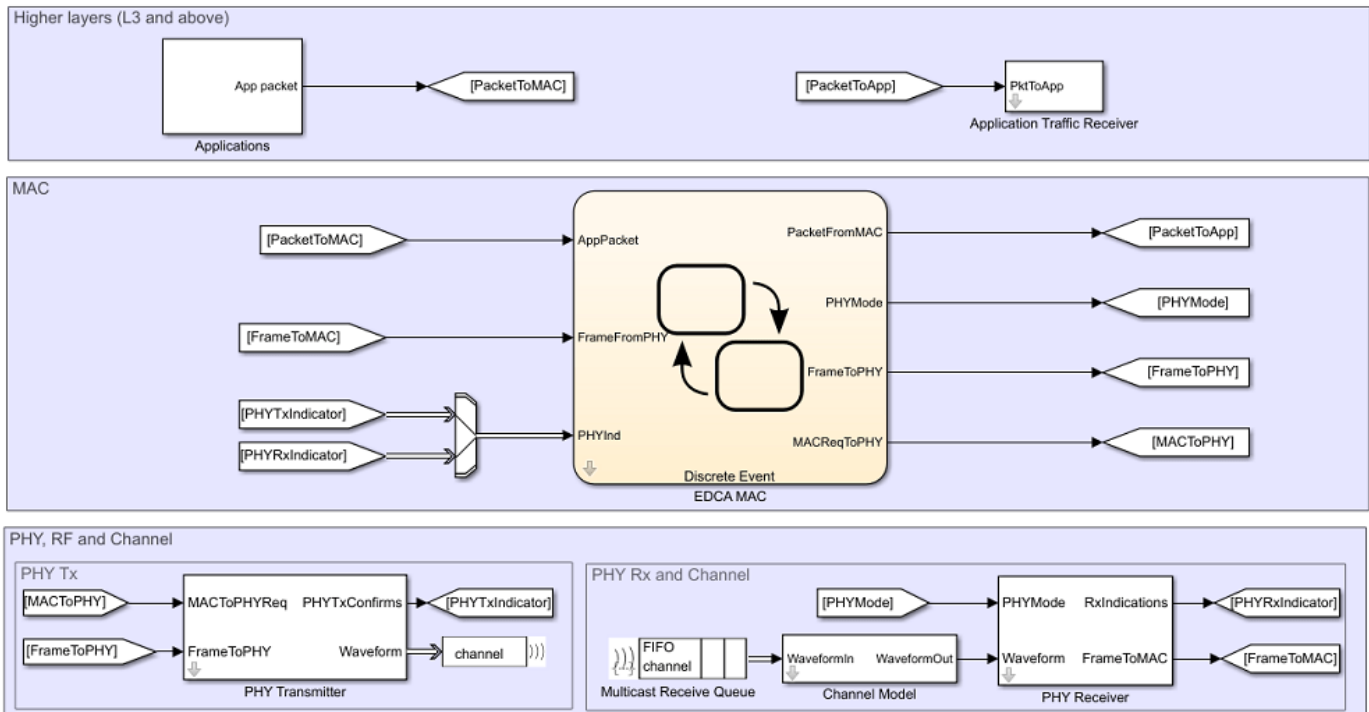
Application Configuration

This example uses the same application layer blocks (traffic generator and traffic receiver) as used in the "802.11 MAC and Application Throughput Measurement" on page 7-27 example. You can add or remove any number of application blocks with different configuration options like `Packet Size`, `Packet Interval`, `Destination Name`, and `Access Category (AC)`. Multiple applications with same AC can be enabled simultaneously with different `Destination Names`. To add or remove applications, enter the `Applications` subsystem block of a node by double clicking on it. This figure shows the applications inside the `Applications` subsystem block.
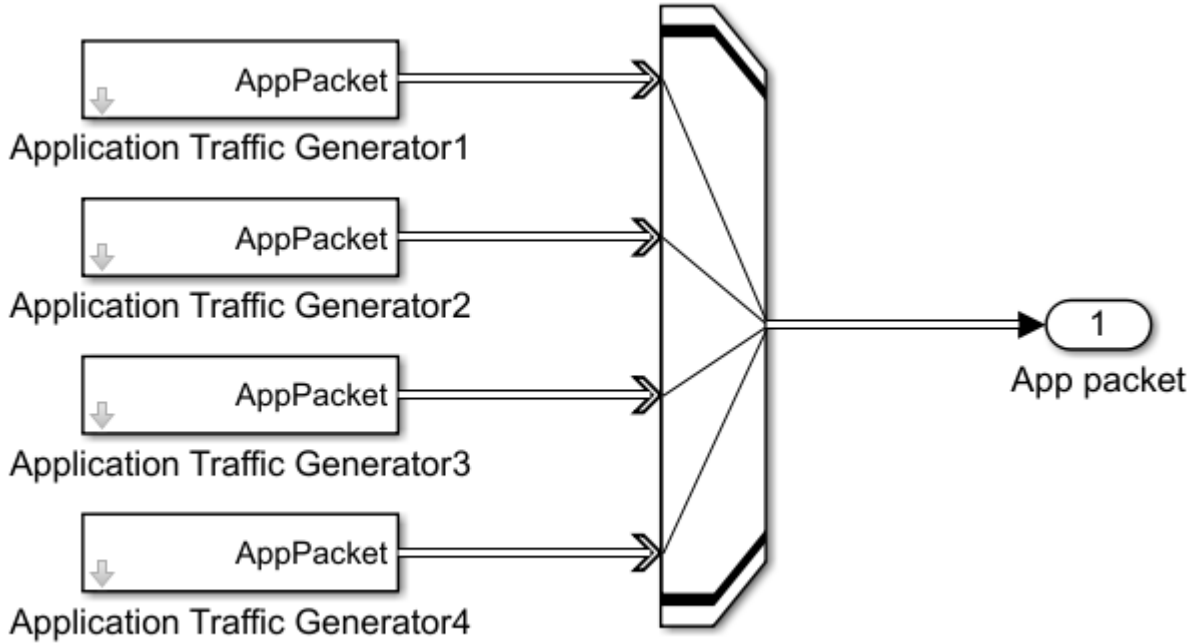
This figure shows the configuration options for an application. Double click on the `Application Traffic Generator` block to retrieve these options.
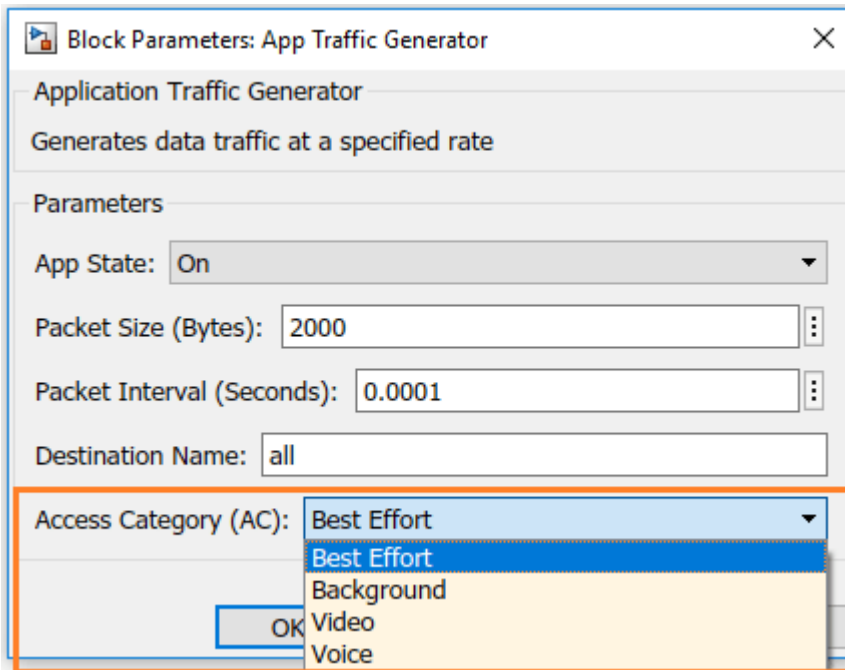
MAC Configuration

- The AP can be configured to transmit DL OFDMA multiuser format frames by setting `PHY Tx Format` to `HE-MU-OFDMA` in MAC configuration parameters. You can also limit the number of users in DL OFDMA transmission using the `Max Downlink Stations` property.

- Enable parallel transmissions between the basic service sets (BSSs) through the `Enable Spatial Reuse with BSS Color` property. This property is applicable only when `PHY Tx Format` property is set to `HE-SU`, `HE-EXT-SU`, or `HE-MU-OFDMA`. This model does not support the spatial reuse (SR) functionality. To study the impact of SR with BSS coloring on the network throughput, refer "Spatial Reuse with BSS Coloring in 802.11ax Residential Scenario" on page 7-12 example.

Model Limitations with HE-MU-OFDMA Format:

- This example supports only DL OFDMA communication. Any node can act as an AP or a station. Nodes with one or more applications enabled are considered APs. This model assumes that all the remaining stations are associated with APs.
- Uplink acknowledgments are not supported.
- Channel reservation using RTS & CTS is not supported.
- OFDMA in combination with MIMO is not supported.
- OFDMA in combination with rate adaptation is not supported.

When the `PHY Tx Format` is `HE-MU-OFDMA`, the `Ack Policy`, `RTS Threshold`, and `Number of Transmit Chains` options are disabled.

Simulation Results

To measure the network throughput, simulations are carried out for these two scenarios:

1 **AP as transmitter**: Simulating the model for the AP serving 1, 2, 4, 6, 8, and 9 stations using the OFDM and OFDMA transmissions.

2 **AP and stations as transmitters**: The same scenario as above but with stations also communicating with an AP using OFDM transmissions.

The throughput results obtained through these simulations are plotted as a function of the number of DL stations for both the OFDM (HE-SU) and the OFDMA (HE-MU-OFDMA) configurations.

Simulation Configuration

Create an 802.11ax network with nine stations and one AP. At the AP, add nine `Application Traffic Generator` blocks generating data to the nine stations. To enable the stations as transmitters, configure `AppState` of an `Application Traffic Generator` block to On. Configure the AP/station(s) with the values shown in this table.

Parameter	Value
Packet size	1000 bytes
Packet Interval	0.00001 seconds
Access Category	Best Effort
Max A-MPDU Subframes	1
MCS	11
Ack Policy	No Ack

Simulate the network for `HE-SU` and `HE-MU-OFDMA` formats by varying the number of applications enabled as 1, 2, 4, 6, 8, and 9 for each simulation. At the end of each simulation run, the throughput values for each node are retrieved from the `statistics.mat` file and summed to obtain the total throughput.

Plot throughput results for OFDM and OFDMA configurations.

AP as transmitter

This code plots OFDM and OFDMA configurations using only AP as transmitters:
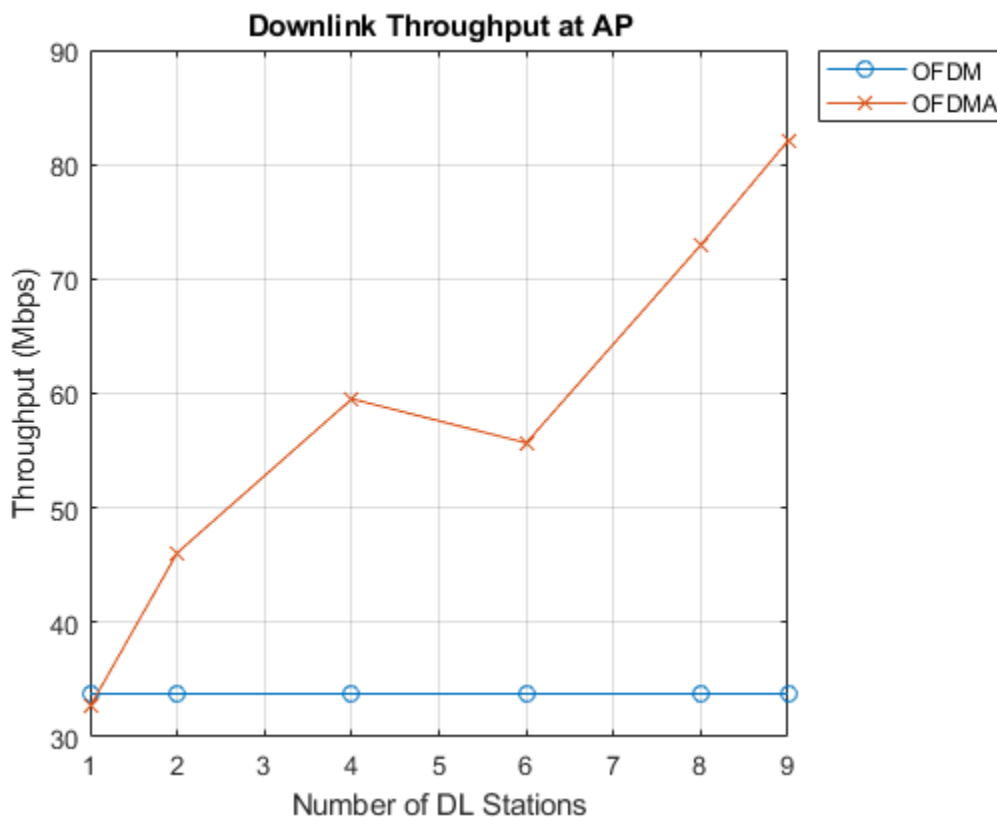
```
figure;
```

```matlab
% Number of DL stations
numStations = [1 2 4 6 8 9];

% Throughput results for OFDMA configuration (Mbps)
throughputOFDMA = [32.64 46.08 59.52 55.68 72.96 82.08];

% Throughput results for OFDM configuration (Mbps)
throughputOFDM = [33.76 33.76 33.76 33.76 33.76 33.76];

% Plot throughput obtained from OFDM simulations
plot(numStations, throughputOFDM,'-o');
% Retain OFDM throughput plot
hold on;
% Plot throughput obtained from OFDMA simulations
plot(numStations, throughputOFDMA,'-x');
grid on;
xlabel('Number of DL Stations');
ylabel('Throughput (Mbps)');
legend('OFDM', 'OFDMA', 'Location', 'northeastoutside');
title('Downlink Throughput at AP');
```


AP and stations as transmitters

This code plots OFDM and OFDMA configurations using both APs and stations as transmitters:

```matlab
figure;
```

```matlab
% Number of DL stations
```

```
numStations = [1 2 4 6 8 9];

% Throughput results for OFDMA configuration (Mbps)
throughputOFDMA = [20.8 27.52 32.64 29.76 40.96 38.88];
% Throughput results for OFDM configuration (Mbps)
throughputOFDM = [21.44 16.64 10.72 8.16 8 7.2];

% Plot throughput obtained from OFDM simulations
plot(numStations, throughputOFDM,'-o');
% Retain OFDM throughput plot
hold on;
% Plot throughput obtained from OFDMA simulations
plot(numStations, throughputOFDMA,'-x');
grid on;
xlabel('Number of DL Stations');
ylabel('Throughput (Mbps)');
legend('OFDM', 'OFDMA', 'Location', 'northeastoutside');
title('Downlink Throughput at AP');
```
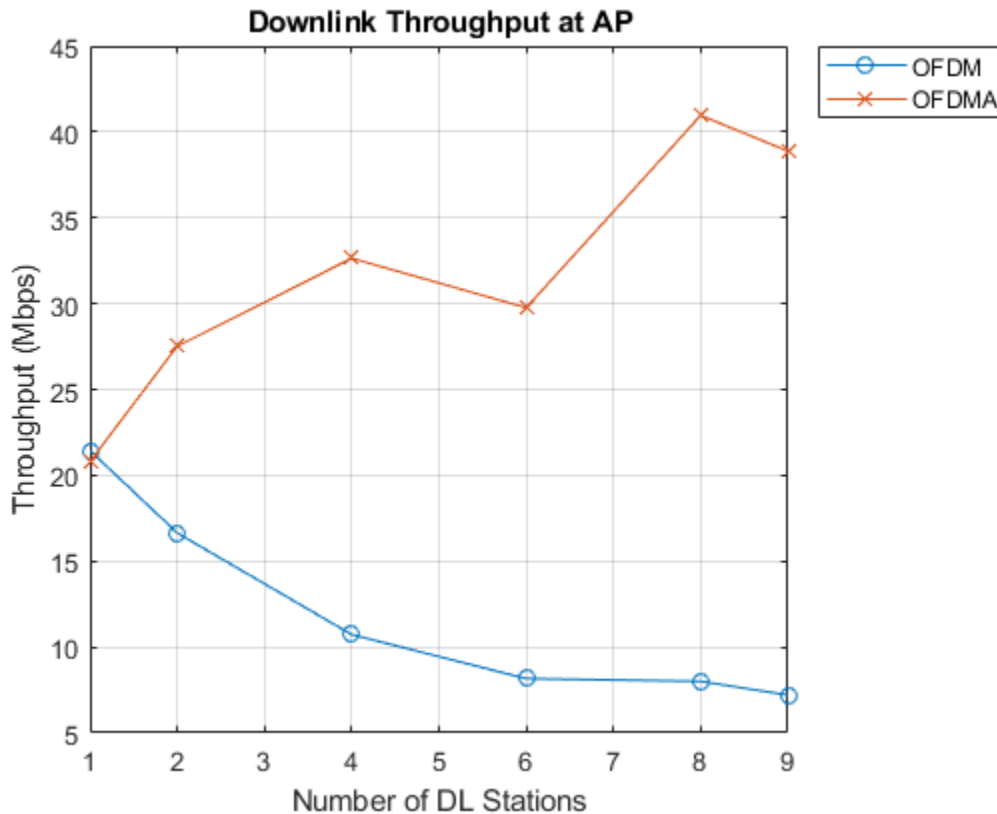

The above plots show the 802.11ax throughput comparison of OFDM and OFDMA. As OFDMA reduces the MAC contention and the PHY preamble overhead, the throughput obtained with OFDMA is greater than that obtained using OFDM. When only AP is configured as the transmitter, the DL throughput shows no change for OFDM. However, when both the AP and the stations are configured as the transmitters, the DL throughput shows a steady decline. This is because the AP gets fewer opportunities to transmit as the number of transmitting stations increase in the network.

This example enables you to model a DL OFDMA communication in a multinode IEEE 802.11ax network. The DL OFDMA support is added to the PHY and MAC library blocks. The application layer configuration allows you to enable multiple applications with same AC using different destination stations. Round-robin scheduling strategy is used to select stations for next transmission. The assignment of RUs is fixed based on the number of users. The plots confirm that the throughput at AP with OFDMA is greater than that obtained using OFDM.

Further Exploration

Scheduling Stations for Transmission

This example uses round-robin scheduling algorithm to select stations for next transmission.

RU allocation

In a transmission, the allocation index defines the assignment of RUs. The allocation index is defined in Table 27-24 of [1]. For each 20 MHz subchannel, an 8-bit index describes the number of RUs, the size of RUs, and the number of users transmitted on each RU. In this example, the allocation of RUs to the DL stations is fixed based on the number of users as shown in this table. In both the above plots, the throughput drop for OFDMA from four stations to six stations can be affiliated to the RU size. Transmission time is shorter in a 52-tone RU as compared to 26-tone RU. When an MU frame is transmitted, all the PSDUs in the MU frame are padded to align at the same transmission time. The presence of 26-tone RU for six users results in a longer transmission time, thereby causing the throughput drop between four and six users.

Number of stations	Allocation index	Sizes of RU (tones)								
1	192	242								
2	96	106				106				
3	128	106		26		106				
4	112	52		52		52		52		
5	15	52	52		26		52		52	
6	7	26	26		52		26	52		52
7	3	26	26	26	26		26	52		52
8	1	26	26	26	26	26	26	26		52
9	0	26	26	26	26	26	26	26	26	26

The model used in this example implements four stations (allocation index 112). You can modify the station scheduling algorithm and RU allocation by updating the helper function roundRobinScheduler used by the EDCA MAC block.

Appendix

The example uses these helpers:

1 edcaFrameFormats.m: Create an enumeration for PHY frame formats.
2 edcaNodeInfo.m: Return MAC address of a node.

3 edcaPlotQueueLengths.m: Plot MAC queue lengths in the simulation.

4 edcaPlotStats.m: Plot MAC state transitions with respect to simulation times.

5 edcaStats.m: Create an enumeration for simulation statistics.

6 edcaUpdateStats.m: Update statistics of the simulation.

7 helperAggregateMPDUs.m: Aggregate MPDUs to form an A-MPDU.

8 helperSubframeBoundaries.m : Return subframe boundaries of an A-MPDU.

9 phyRx.m: Model PHY operations related to packet reception.

10 phyTx.m Model PHY operations related to packet transmission.

11 edcaApplyFading.m: Apply Rayleigh fading effect on the waveform.

12 heSIGBUserFieldDecode.m: Decode HE-SIG-B user field.

13 heCPECorrection.m: Estimate and correct common phase error.

14 heSIGBCommonFieldDecode.m: Decode HE-SIG-B common field.

15 heSIGBMergeSubchannels.m: Merge 20MHz HE-SIG-B subchannels.

16 addMUPadding.m: Add multiuser PSDU padding.

17 macQueueManagement.m: Create a WLAN MAC queue management object.

18 roundRobinScheduler.m: Create a round-robin scheduler object.

19 calculateSubframesCount.m: Return number of subframes to be aggregated.

20 interpretVHTSIGABitsFailCheck.m: Interprets the bits in VHT-SIG-A field

21 rateAdaptationARF.m: Create an auto rate fallback (ARF) algorithm object.

22 rateAdaptationMinstrelNonHT.m: Create a minstrel algorithm object.

References

1 IEEE P802.11ax™/D4.1. "Amendment 6: Enhancements for High Efficiency WLAN" Draft Standard for Information technology - Telecommunications and information exchange between systems Local and metropolitan area networks - Specific requirements -Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

2 IEEE Std 802.11ac™ -2016. "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 Jiang, Tao, et al., editors. *Orthogonal Frequency Division Multiple Access Fundamentals and Applications*. Auerbach, 2010.

Generate and Visualize FTP Application Traffic Pattern

This example shows how to generate a file transfer protocol (FTP) application traffic pattern based on the IEEE® 802.11ax™ Evaluation Methodology [1 on page 7-0   ] and the 3GPP TR 36.814 specification [2 on page 7-0   ].

FTP Application Traffic Model

Multinode communication systems involve modeling of different application traffic models. Each application is characterized by parameters such as the data rate, packet inter arrival time, and packet size. To evaluate various algorithms and protocols, standardization bodies such as IEEE and 3GPP define certain application traffic patterns such as Voice over Internet Protocol (VoIP), video conferencing, and FTP. This example generates and visualizes an FTP application traffic pattern.

The FTP application traffic pattern is modeled as a sequence of file transfers separated by reading time. The reading time specifies the time interval between two successive file transfers. The file is generated as multiple packets separated by packet inter arrival time. The packet inter arrival time specifies the time interval between two successive packet transfers.

The 11ax Evaluation Methodology [1 on page 7-0   ] specifies this FTP application traffic model:

- **Local FTP traffic model -** This model is characterized by truncated Lognormal file size and exponential reading time.

The 3GPP TR 36.814 specification [2 on page 7-0   ] specifies these FTP application traffic models:

- **FTP traffic model 2 -** This model is characterized by 2/0.5 megabytes file size and exponential reading time. This figure shows the traffic pattern of this FTP model.
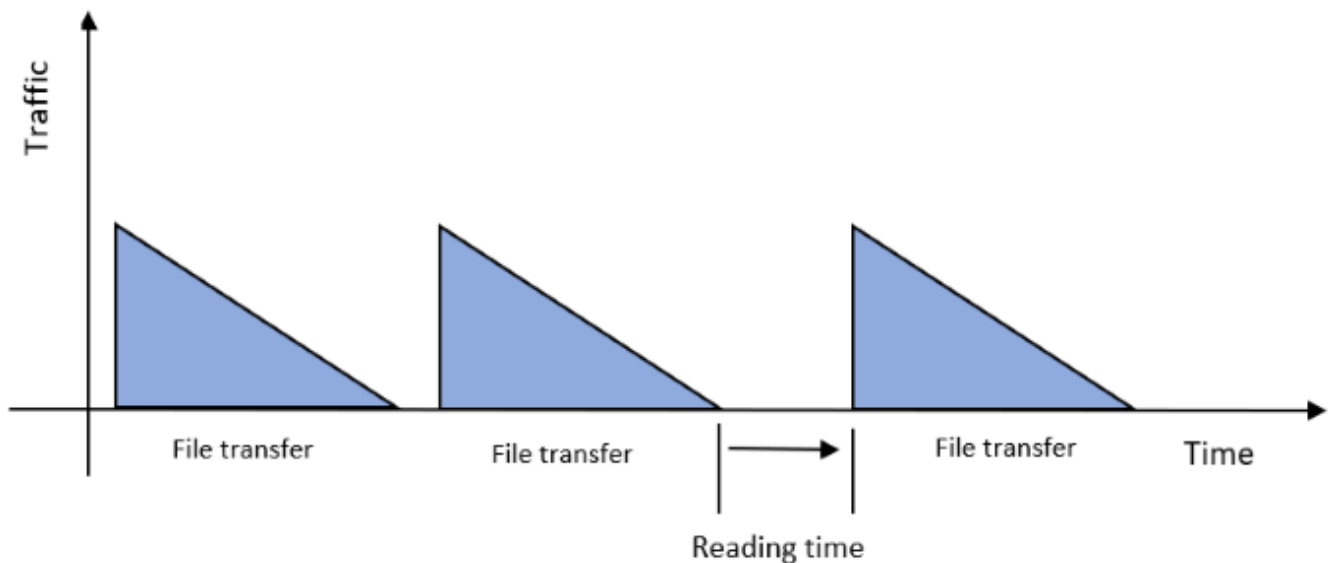
- **FTP traffic model 3 -** This model is characterized by a 0.5 megabytes file, exponential reading time, and Poisson packet arrival rate. This figure shows the traffic pattern of this FTP model.
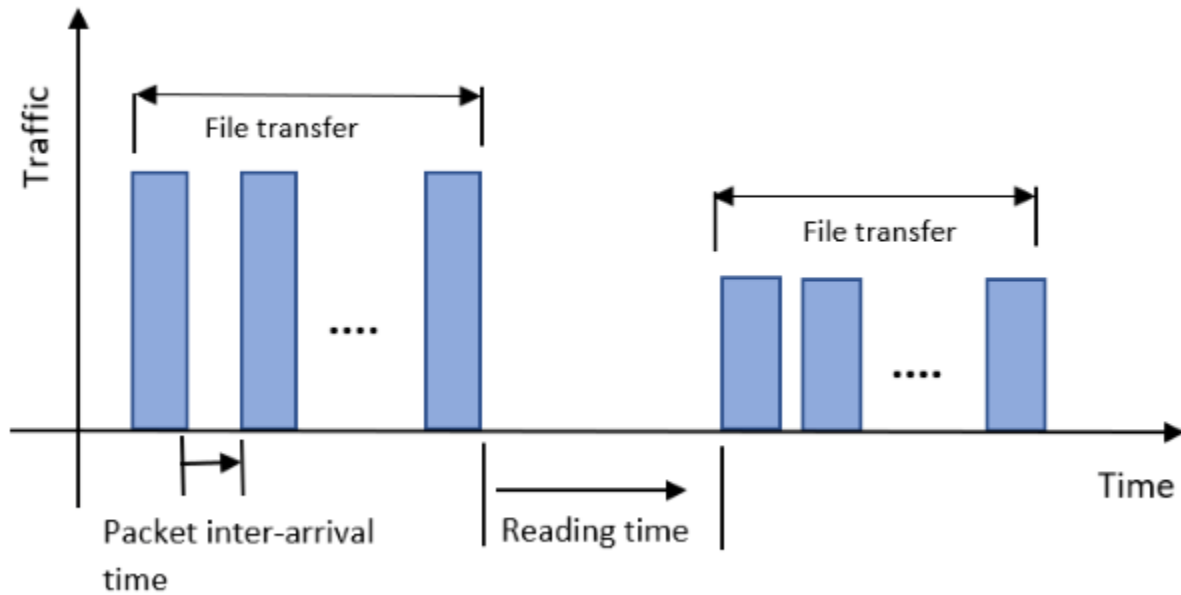
This example demonstrates the local FTP traffic model specified in 11-ax Evaluation Methodology [1 on page 7-0   ]. Similarly, you can use the FTP traffic models 2 and 3 specified in 3GPP TR 36.814 specification [2 on page 7-0   ] using the file size and packet arrival rate properties.

Configure FTP Application Traffic Pattern Object

Create a configuration object to generate the FTP application traffic pattern.

```
% Create FTP application traffic pattern object with default properties
ftpObj = networkTrafficFTP;

% Set exponential distribution mean value for reading time in milliseconds
ftpObj.ExponentialMean = 50;

% Set truncated Lognormal distribution mu value for file size calculation
ftpObj.LogNormalMu = 10;

% Set truncated Lognormal distribution sigma value for file size calculation
ftpObj.LogNormalSigma = 1;

% Set truncated Lognormal distribution upper limit in megabytes
ftpObj.UpperLimit = 5;

% Display object
disp(ftpObj);

    networkTrafficFTP with properties:

            LogNormalMu: 10
         LogNormalSigma: 1
             UpperLimit: 5
         ExponentialMean: 50
```

```
      PacketInterArrivalTime: 0
              GeneratePacket: 0
```

Generate and Visualize FTP Application Traffic Pattern

Generate FTP application traffic pattern using the `generate` object function of the `networkTrafficFTP` object.

```
% Set simulation time in milliseconds
simTime = 10000;

% Set step time in milliseconds
stepTime = 1;

% Validate simTime, simTime must be greater than or equal to stepTime
validateattributes(simTime, {'numeric'}, {'real', 'scalar', 'finite', '>=', stepTime});
rng('default');

% Time after which the generate method must be invoked again
nextInvokeTime = 0;

% Generated packet count
packetCount = 0;

% Initialize arrays to store outputs for visualization
% Packet generation times in milliseconds
generationTime = zeros(5000, 1);

% Time interval between two consecutive packet transfers in milliseconds
packetIntervals = zeros(5000, 1);

% Packet sizes in bytes
packetSizes = zeros(5000, 1);

% Loop over the simulation time, generating FTP application traffic pattern
% and saving the dt and packet size values for visualization
while simTime
    if nextInvokeTime <= 0          % Time to generate the packet
        packetCount = packetCount+1; % Increment packet count
        % Call generate method and store outputs for visualization
        [packetIntervals(packetCount), packetSizes(packetCount)] = generate(ftpObj);
        % Set next invoke time
        nextInvokeTime = packetIntervals(packetCount);
        % Store packet generation time for visualization
        generationTime(packetCount+1) = generationTime(packetCount) + packetIntervals(packetCount
    end

    % Update next invoke time
    nextInvokeTime = nextInvokeTime - stepTime;

    % Update simulation time
    simTime = simTime - stepTime;
end
```

Visualize the generated FTP application traffic pattern. In this plot, dt is the time interval between two successive FTP application packets.

```
% Packet Number Versus Packet Intervals (dt)
% Stem graph to see packet intervals
pktIntervalsFig = figure('Name', 'Packet intervals', 'NumberTitle', 'off');
pktIntervalsAxes = axes(pktIntervalsFig);
stem(pktIntervalsAxes, packetIntervals(1:packetCount));
title(pktIntervalsAxes, 'Packet Number Versus dt');
xlabel(pktIntervalsAxes, 'Packet Number');
ylabel(pktIntervalsAxes, 'dt in Milliseconds');
```
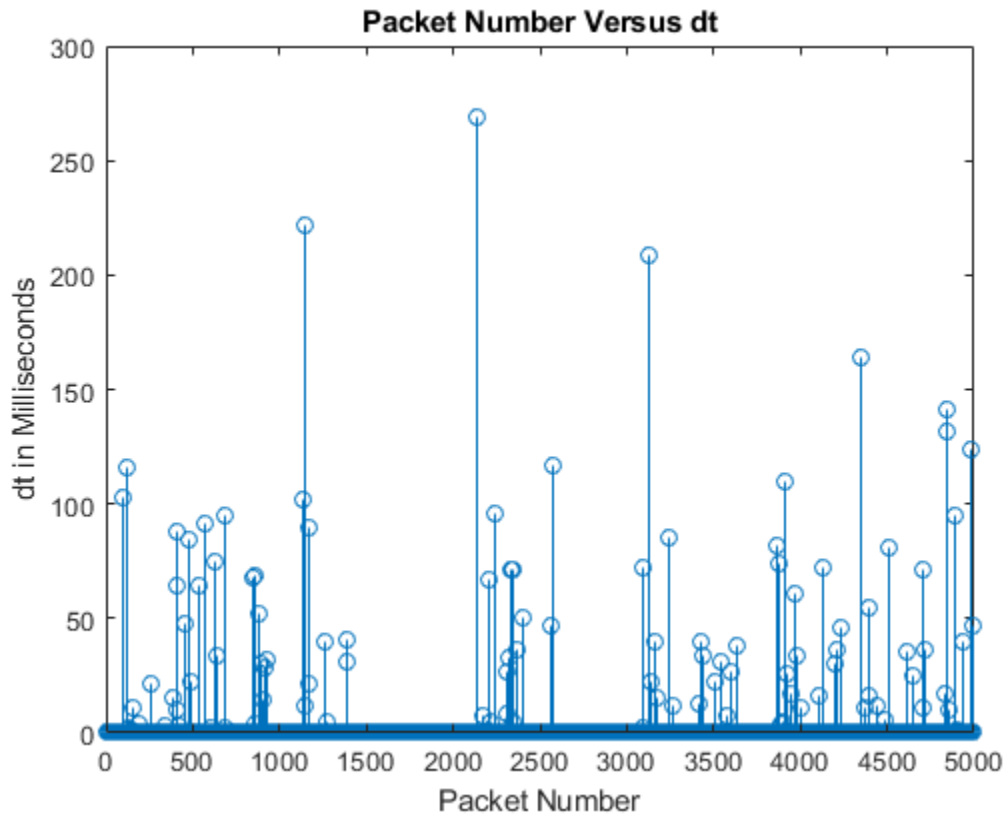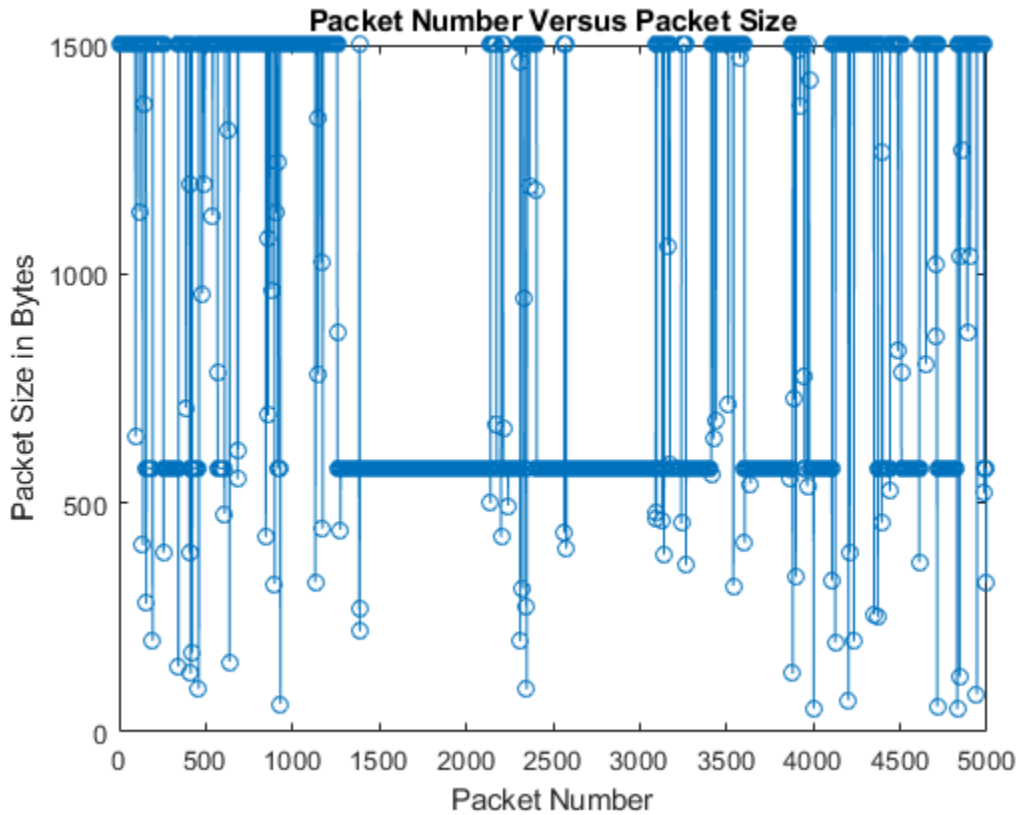


```
% Plot to see different packet sizes
pktSizesFig = figure('Name', 'Packet sizes', 'NumberTitle', 'off');
pktSizesAxes = axes(pktSizesFig);
plot(pktSizesAxes, packetSizes(1:packetCount), 'marker', 'o');
title(pktSizesAxes, 'Packet Number Versus Packet Size');
xlabel(pktSizesAxes, 'Packet Number');
ylabel(pktSizesAxes, 'Packet Size in Bytes');
```

```
% Stem graph of FTP application traffic pattern (Packet sizes of different files
% at different packet generation times)
ftpPatternFig = figure('Name', 'FTP application traffic pattern', 'NumberTitle', 'off');
ftpPatternAxes = axes(ftpPatternFig);
stem(ftpPatternAxes, generationTime(1:packetCount), packetSizes(1:packetCount), 'Marker', 'o');
title(ftpPatternAxes, 'Packet Generation Time Versus Packet Size');
ylabel(ftpPatternAxes, 'Packet Size in Bytes');
xlabel(ftpPatternAxes, 'Time in Milliseconds');
```
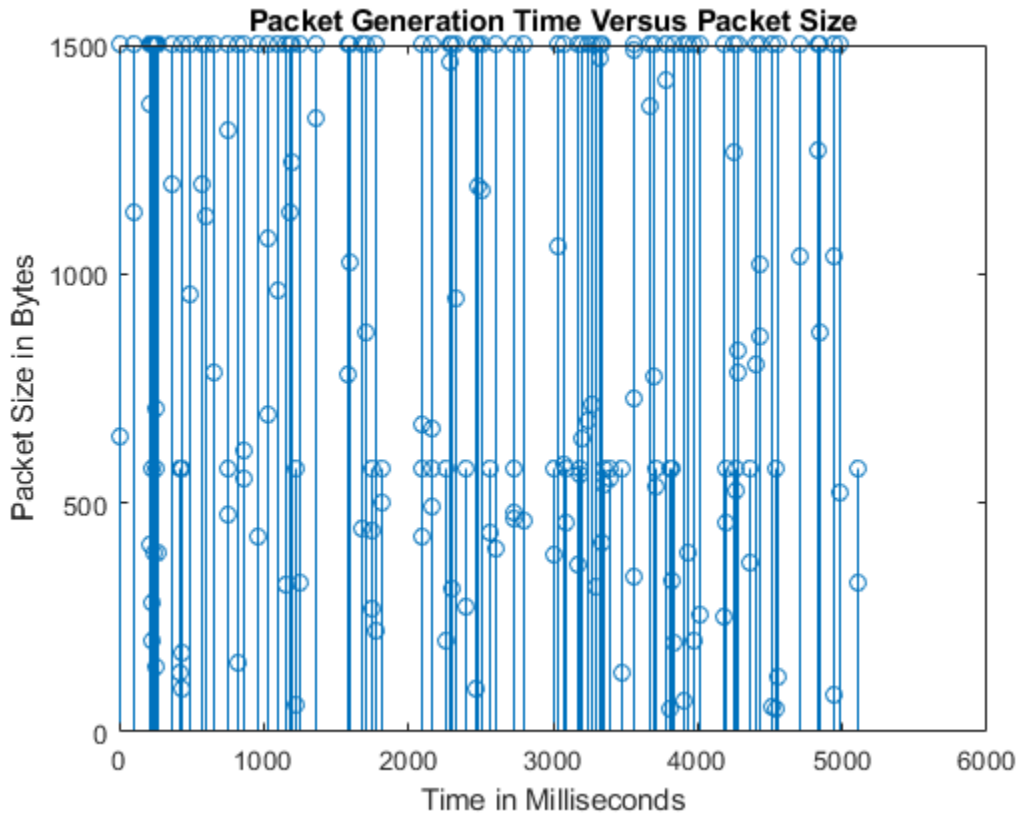
Further Exploration

This example generates an FTP traffic pattern as per the 11ax Evaluation Methodology [1 on page 7-0] and 3GPP specification [2 on page 7-0]. Similarly, you can use `networkTrafficVoIP` and `networkTrafficOnOff` objects to generate VoIP and On-Off application traffic patterns, respectively. You can use these different application traffic patterns in system-level simulations to accurately model the real-world data traffic.

References

1 IEEE 802.11-14/0571r12 . "11ax Evaluation Methodology". IEEE P802.11. Wireless LANs.

2 3GPP TR 36.814. "Evolved Universal Terrestrial Radio Access (E-UTRA). Further advancements for E-UTRA physical layer aspects". *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Test and Measurement

Modeling and Testing an 802.11ax RF Receiver with 5G Interference

The example shows how to characterize the impact of RF impairments, such as phase noise and power amplifier (PA) nonlinearities, in the RF reception of an IEEE® 802.11ax™ waveform coexisting with an adjacent 5G or 802.11ax interferer. The example generates the baseband waveforms by using WLAN Toolbox™ and 5G Toolbox™ and models the RF receiver by using RF Blockset™. The example does not require 5G Toolbox if it models an 802.11ax interferer.

Introduction

This example characterizes the impact of receiver RF impairments, such as phase noise and power amplifier (PA) nonlinearities, and the impact of an adjacent 5G [1] or 802.11ax [2] interferer in the RF reception of an 802.11ax waveform. To evaluate the impact of the interference, the example performs these measurements:

- Error vector magnitude (EVM): vector difference between the ideal (transmitted) signal and the measured (received) signal
- Adjacent channel rejection (ACR): power difference between the desired signal and an interfering signal in the adjacent channel
- Packet error rate (PER): number of packets containing errors divided by the total number of received packets

The example works on a packet-by-packet basis. For each desired HE packet, the workflow consists of these steps:

1 Generate the baseband 802.11ax waveform (desired) by using WLAN Toolbox.
2 Generate the baseband 5G waveform (interferer) by using 5G Toolbox. You can generate an 802.11ax interferer by using WLAN Toolbox instead. Alternatively, you can remove the interference.
3 Oversample and filter the waveforms by using a Finite Impulse Response (FIR) Interpolation block.
4 Measure and display the ACR by calculating the power difference between both waveforms.
5 Aggregate both waveforms by using the Frequency Shift and Aggregation block.
6 Convert the baseband waveform into an RF signal by using the RF Receiver block. The block uses an RF carrier frequency to carry the baseband information in RF Blockset. You can select the RF carrier frequency of your choice.
7 Downconvert the waveform to an intermediate frequency by using an RF superheterodyne receiver. You can accurately model the impairments introduced by an actual RF receiver by using the RF components available in RF Blockset. You can also explore the impact of altering the RF impairments or replace the RF superheterodyne receiver with another RF subsystem of your choice.
8 Downsample and filter the waveform by using an FIR Decimation block.
9 Extract the data symbols and measure the EVM by demodulating the baseband waveform.
10 Calculate the PER by extracting the received bits and comparing them to the transmitted bits.

This example performs these operations by using a Simulink® model. The Simulink model carries out the baseband signal processing (steps 1, 2, 9 and 10) by using MATLAB® Function blocks and

performs the RF receiver modeling (steps 6 and 7) by using RF Blockset. This model supports `Normal` and `Accelerator` simulation modes.

Simulink Model Structure

The model contains three main parts:

- Baseband Waveform Generation: generates the baseband 802.11ax and 5G waveforms
- RF Reception: downconverts the waveform to an intermediate frequency by modeling the effect of RF components
- Baseband Waveform Reception: calculates EVM and PER

```
modelName = 'HERFReceiverNRInterfererModel';
open_system(modelName);
```
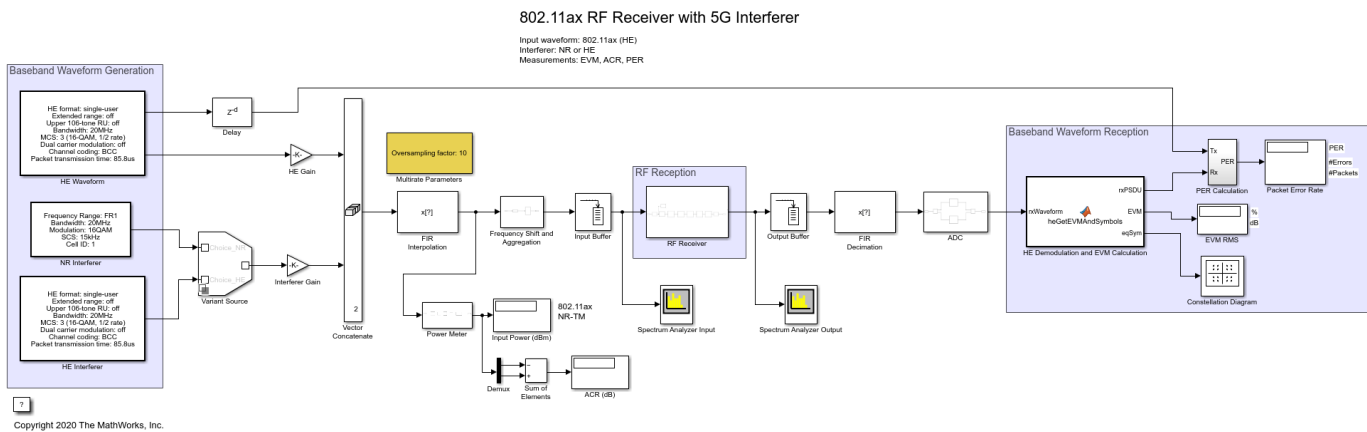

Baseband Waveform Generation

The HE Waveform block generates standard-compliant high-efficiency single-user (HE SU) waveforms [2]. For the waveform generation, set transmission and configuration parameters by specifying options in the HE Waveform block.

The HE Waveform block contains two tabs:

- HE SU Format: configure the transmission parameters selected in this section by using a `wlanHESUConfig` object.
- Generator Configuration: generate each packet, which contains random data, with the `wlanWaveformGenerator` function for the specified HE SU configuration and parameters.

Block Parameters: HE Waveform

HE SU Format | Generator Configuration

Parameters to configure an HE single-user (SU) packet, which is a full-band transmission to a single user.

☐ Extended range single-user

☐ Upper 106-tone RU

Channel bandwidth | 20

Modulation/coding | 3 (16-QAM, 1/2 rate)

☐ Dual carrier modulation

Channel coding | BCC

APEP length (bytes) | 1000

Guard interval (μs) | 0.8

HE-LTF type | 4x

☐ Uplink

BSS color | 0

Spatial reuse | 0

TXOP duration | 127

Packet padding (μs) | 0

OK | Cancel | Help | Apply

Similarly, the NR Interferer block transmits standard-compliant 5G NR waveforms for frequency range 1 (FR1) [1]. For the NR waveform generation, you can specify the channel bandwidth, modulation, subcarrier spacing (SCS), and cell identity in the NR Interferer block. The NR Interferer block transmits a full band and uniform PDSCH. The model resamples the NR waveform so that the sampling rate of the NR waveform matches the sampling rate of the 802.11ax waveform.
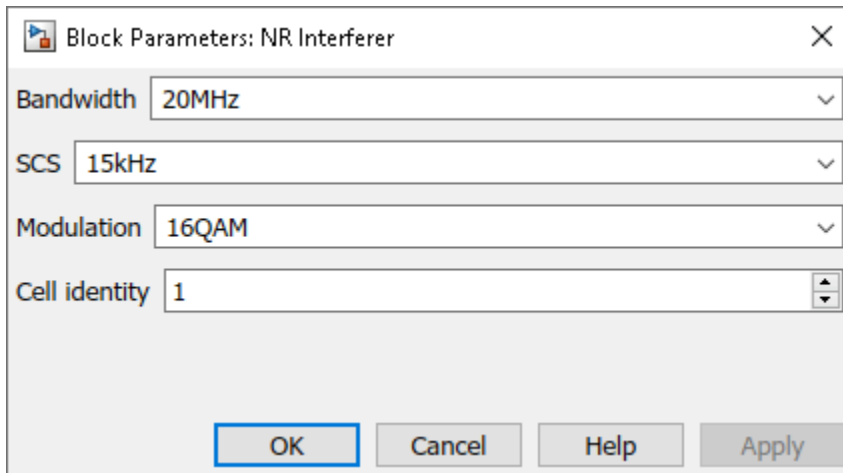
Alternatively, you can model an 802.11ax interferer instead of a 5G interferer by selecting **Choice_HE** in the Variant Source block. The model shifts the HE interferer so that it is not synchronized with the desired HE waveform.

Control the power of both waveforms by setting the HE Gain and Interferer Gain blocks. To cancel the transmission of the interferer, set the **Gain** parameter of the Interferer Gain block to 0.

After generating the waveforms, a Vector Concatenate block concatenates both waveforms horizontally, one column per waveform. Then, an FIR Interpolation block oversamples and filters the waveforms to show the effect of the nonlinear impairments on the adjacent channels. To capture at least third order and fifth order nonlinearities, oversample the combined bandwidth (both waveforms) around 5 times. As the combined bandwidth is 40MHz by default (20 MHz each waveform and a 20 MHz spacing between them), set an **Oversampling factor** of 10 to provide a sample rate of 200 MHz, which is 5 times the combined bandwidth. You can set the **Oversampling factor** in the Multirate Parameters block, which provides an interface to easily configure the parameters of the FIR Interpolation and Decimation blocks.
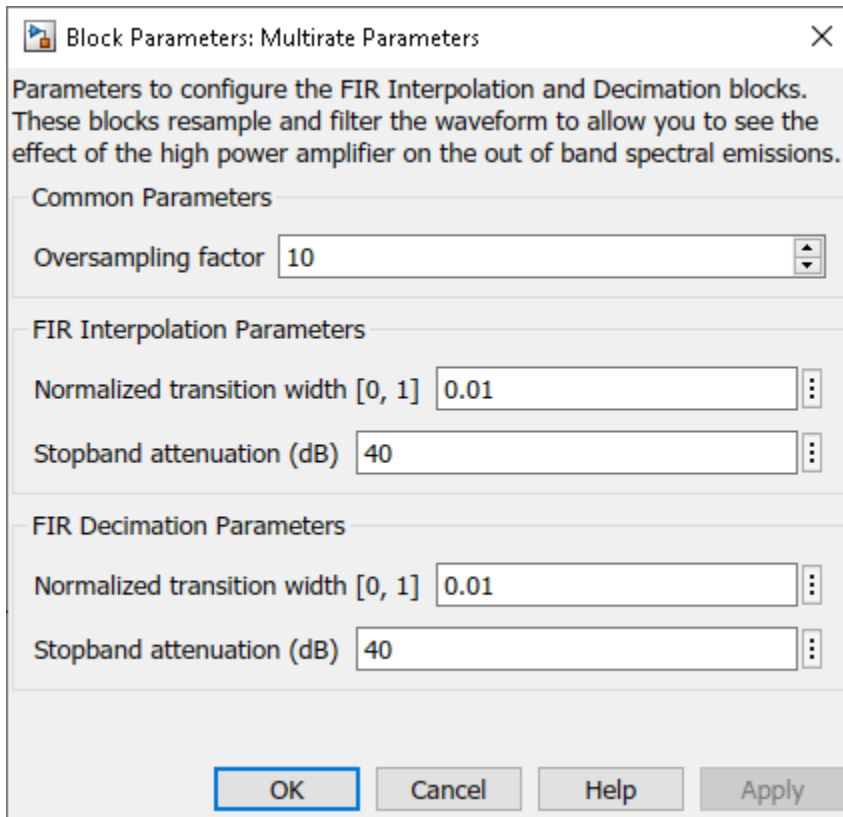
Once the waveforms have been oversampled, the Frequency Shift and Aggregation block frequency shifts and aggregates them. To measure the ACR, the center frequency of the adjacent channel shall be placed 20, 40, 80, or 160 MHz away from the center frequency of the desired signal [2]. By default, the example centers the HE waveform at baseband (0 Hz) and sets the spacing between the HE and interfering waveforms to 20 MHz. You can adjust the center frequencies by specifying the **Desired output center frequencies (Hz)** parameter in the Frequency Shift and Aggregation block. The ACR measurement is displayed in the ACR (dB) block.

Specify Simulation Time

The **Packet transmission time (μs)** parameter in the HE Waveform block calculates the time required to transmit each HE packet. Hence, the **Stop Time** value in the Simulink model must be equal to or higher than the value specified in **Packet transmission time (μs)** to obtain the EVM results and constellation diagram of at least one packet. As the filters in the FIR Interpolation and Decimation blocks introduce a delay, you can use the **Idle time (s)** parameter in the HE Packet block to compensate for the delay.

RF Reception

The RF Receiver block is based on a superheterodyne receiver architecture. This architecture applies passband filtering and amplification and downconverts the received waveform to an intermediate frequency. The RF components of this superheterodyne receiver are:

• RF and IF bandpass filters
• Low-noise and IF amplifiers

- Demodulator consisting of mixers, phase shifter, and local oscillator

```
set_param(modelName,'Open','off');
set_param([modelName '/RF Receiver'],'Open','on');
```
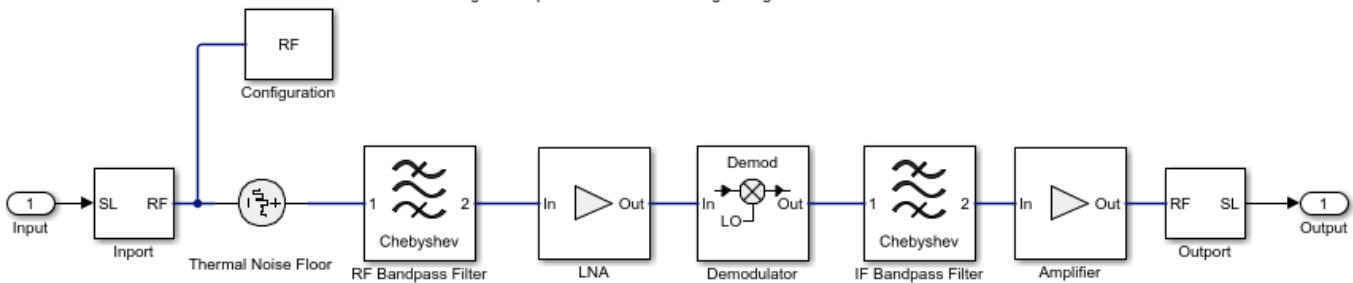

The Inport block inside the RF Receiver converts the complex baseband waveform into the RF domain. You can vary the center frequency of this RF signal by modifying the **Carrier frequency** parameter of this block. By default, the **Carrier frequency** parameter corresponds to the center frequency of the desired HE waveform and the carrier frequency of the NR waveform is located 20 MHz from the HE carrier. The Outport block converts the RF signal back to complex baseband.

You can configure the RF Receiver components by using the RF Receiver block mask.

Block Parameters: RF Receiver ✕

RF Receiver (mask)

Estimate the impact of non-linearity, phase noise, impedance mismatch and IQ imbalance.

| RF Stage | RF Filter | LNA | IQ Demodulator | IF Filter | Amplifier |

Radio Frequency Stage Parameters

HE carrier frequency (MHz): `5950`

HE intermediate frequency (MHz): `70`

Source impedance (Ohm): `50`

Load impedance (Ohm): `50`

System temperature (K): `290`

OK | Cancel | Help | Apply

The RF Receiver block exhibits typical impairments, including:

- Phase noise as an effect directly related to the thermal noise within the active devices of the oscillator
- Amplifier nonlinearities due to DC power limitation when the amplifiers work in the saturation region

Use an Input Buffer block before the RF Receiver block to send fewer samples at a time to the RF Receiver block. For simplicity, the Input Buffer in the current configuration sends one sample at a time, resulting in the RF Receiver block being sample-based.

As the current RF Receiver block configuration sends one sample at a time, the Output Buffer block (after the RF Receiver block) collects all samples within the baseband HE waveform before sending the samples to the HE Demodulation and EVM Calculation block. At the output of the RF Receiver block, the FIR Decimation block downsamples the waveform back to its original sampling rate. Additionally, the ADC block digitizes the signal. You can modify the ADC block parameters using its mask.
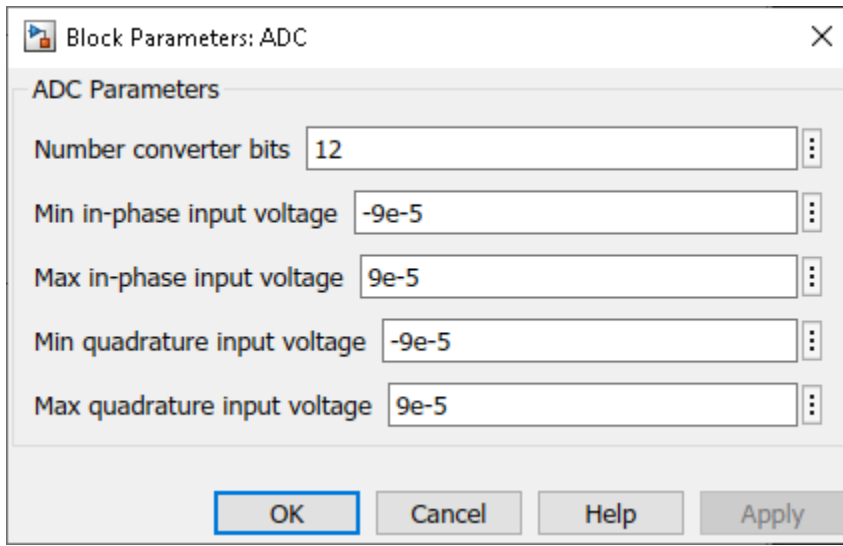
Baseband Waveform Reception

The HE Demodulation and EVM Calculation block recovers and plots the HE-Data symbols in the Constellation Diagram block by performing frequency and packet offset corrections, channel estimation, pilot phase tracking, OFDM demodulation, and equalization. This block performs these EVM measurements:

- EVM per subcarrier (dB): EVM averaged over the allocated HE-Data symbols within a subcarrier
- EVM per OFDM symbol (dB)
- Overall EVM (dB and %): EVM averaged over all transmitted HE-Data symbols

This block also decodes each packet to recover the transmitted bits. The example compares the recovered bits to those transmitted for each packet to determine the packet error rate for the simulation duration by using the PER Calculation block.

The ACR measurement is displayed in the ACR (dB) block. You can also measure the ACR by calculating the power difference between the **Channel Power** levels of each waveform in the Spectrum Analyzer Input block. To check the **Channel Power** levels of each waveform, set this configuration in the Spectrum Analyzer Input block:

- The **Span (Hz):** must be the bandwidth of the waveform to measure. By default, the example sets this value to 20 MHz, which is the bandwidth of both waveforms, the desired HE and the interferer.
- The **CF (Hz):** must be 0 for the desired HE waveform or the spacing between both waveforms (defined in the Frequency Shift and Aggregation block) for the interferer. By default, the example sets this value to 0 Hz to measure the channel power of the desired waveform.

To measure ACR according to IEEE P802.11ax/D7.0, set the desired waveform power 3 dB above the rate-dependent sensitivity specified in Table 27-51 (-71 dBm for the default configuration) and adjust the power level of the interferer waveform to achieve a 10% PER for a PSDU length of 4096 octets.

Model Performance

To characterize the impact of the NR interference on the HE reception you can compare the EVM for two different cases: 1) without interference, for example, transmit only the HE waveform; and 2) with

interference, for example, transmit both HE and NR waveforms. You can also measure the ACR in the second case.
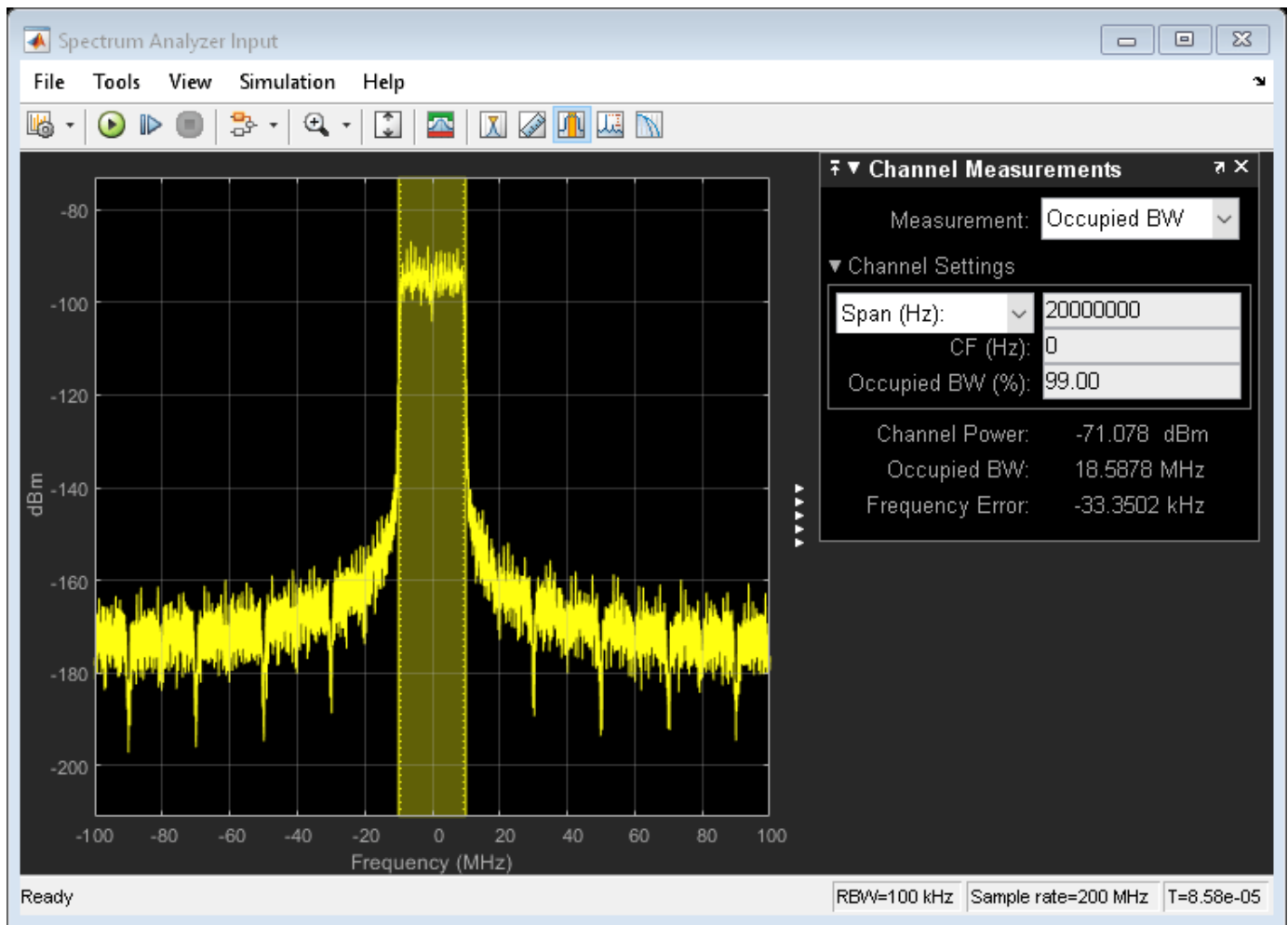
- *Without NR interference (NR gain = 0).* To eliminate the NR interference, set the **Gain** parameter of the Interferer Gain block to 0. To calculate the EVM and plot the constellation diagram, run the simulation long enough to capture one packet (**Stop Time** equal to 85.5 microseconds for the default configuration).
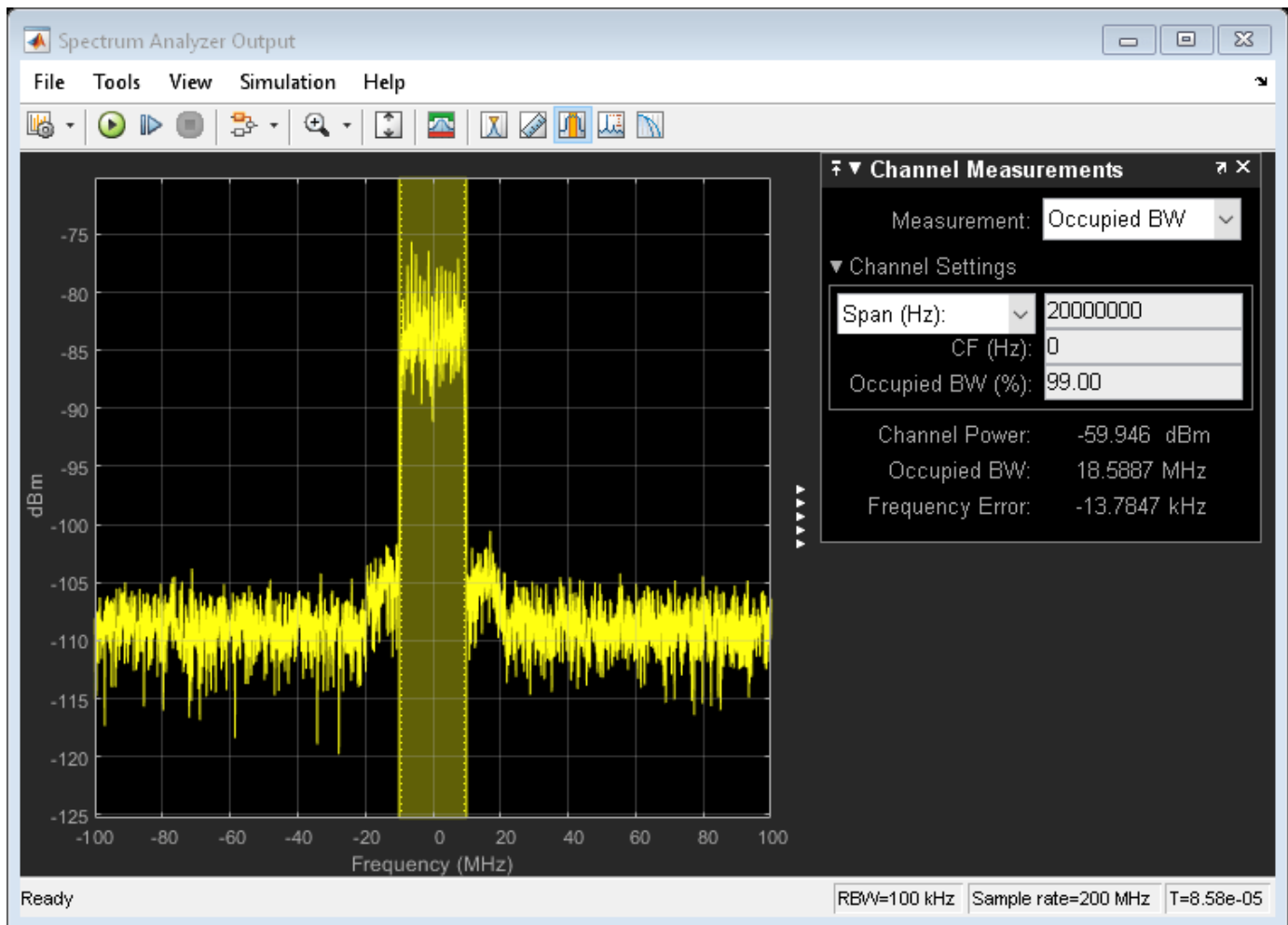
```
set_param([modelName '/Interferer Gain'],'Gain','0');
sim(modelName);
```
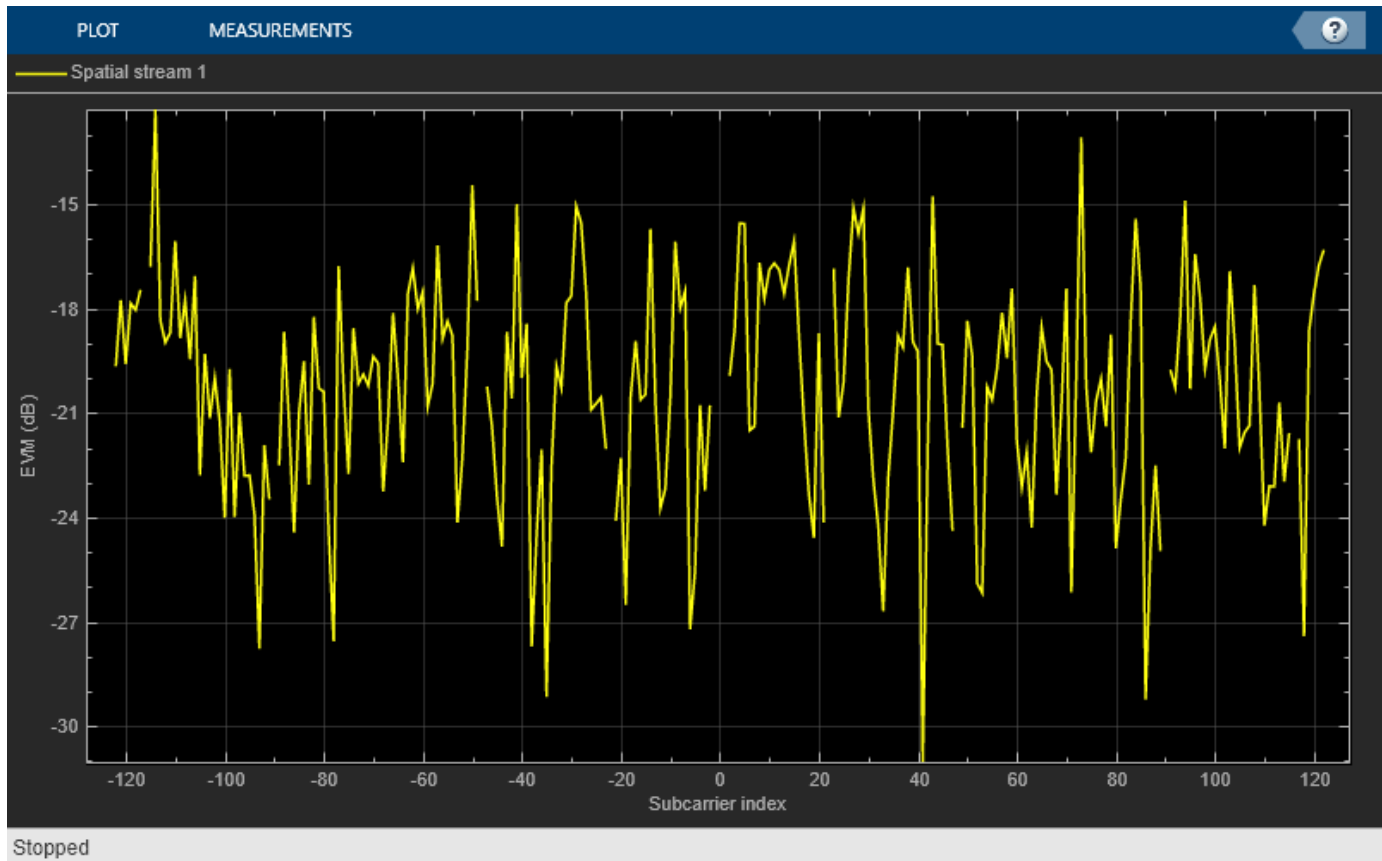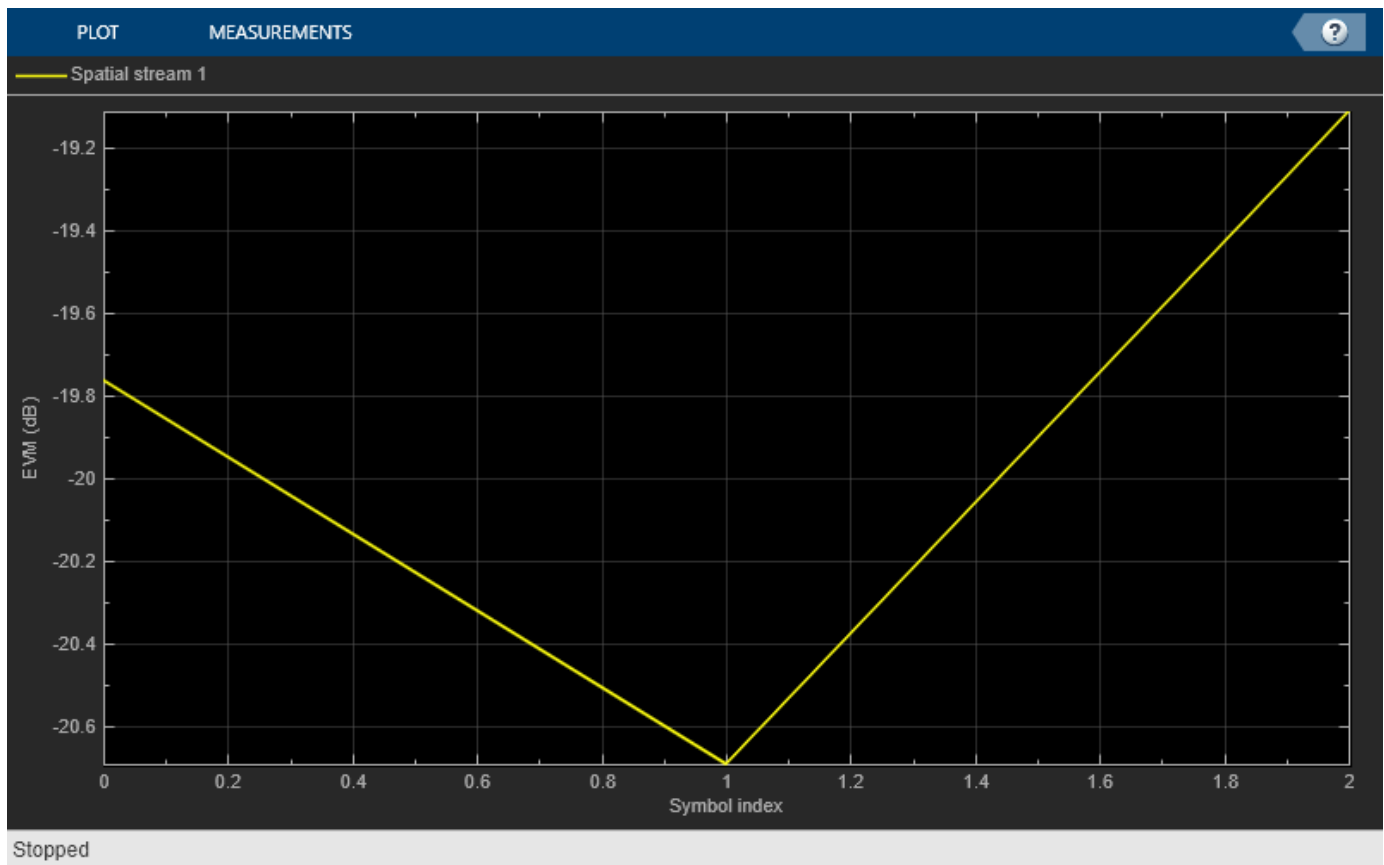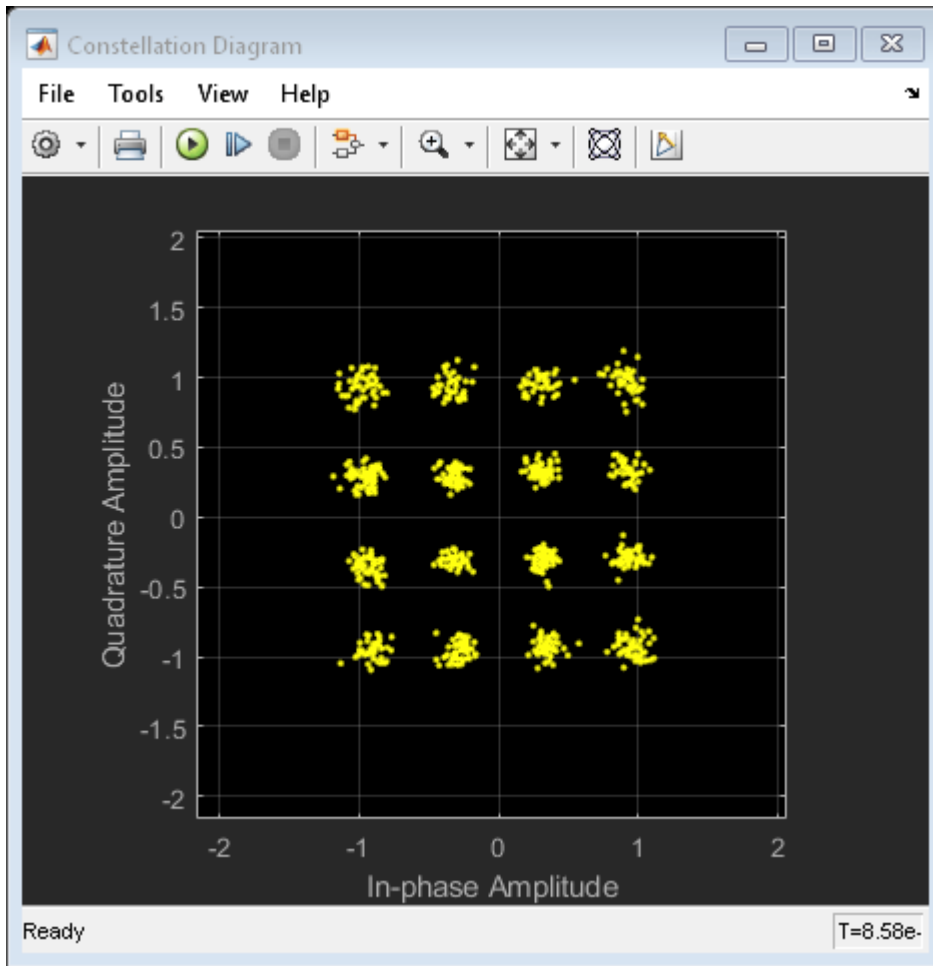
When you disable the interference, the overall EVM is around -20 dB.

- *With NR interference (NR gain = -37.72 dB)*. To activate the NR interference, set the **Gain** parameter of the Interferer Gain block to any value greater than 0. For example, in order to measure the ACR when the PER is approximately 10% for a PSDU length of 4096 octets [2], choose a gain value of around -37.72 dB and increase the APEP length. If you want to measure the PER for several packet transmissions, for example 100 packets, multiply the current **Stop Time** value by 100. By default, the example transmits one packet and sets the APEP length to 50 bytes.
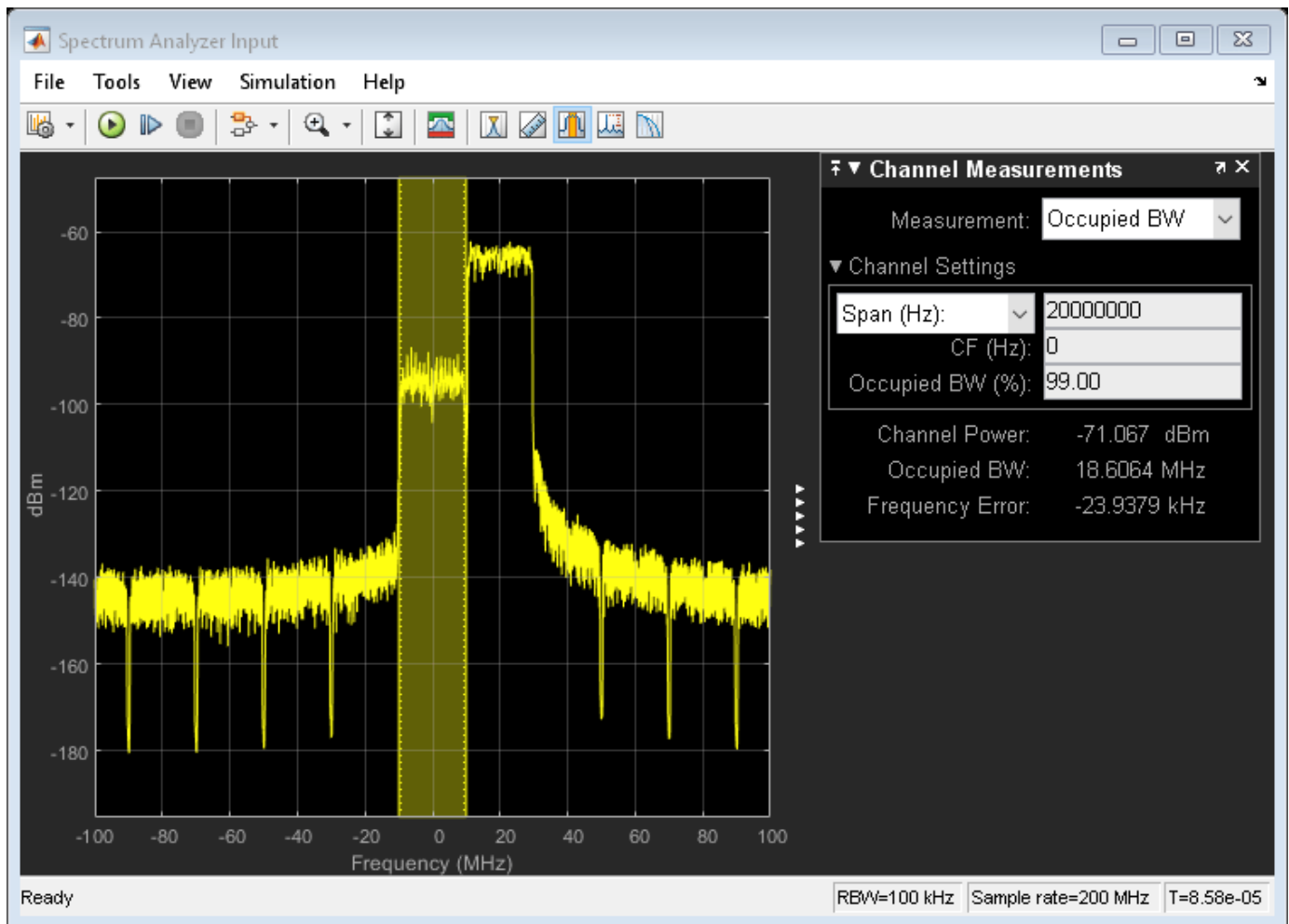
```
set_param([modelName '/Interferer Gain'],'Gain','db2mag(-37.72)');
sim(modelName);
```
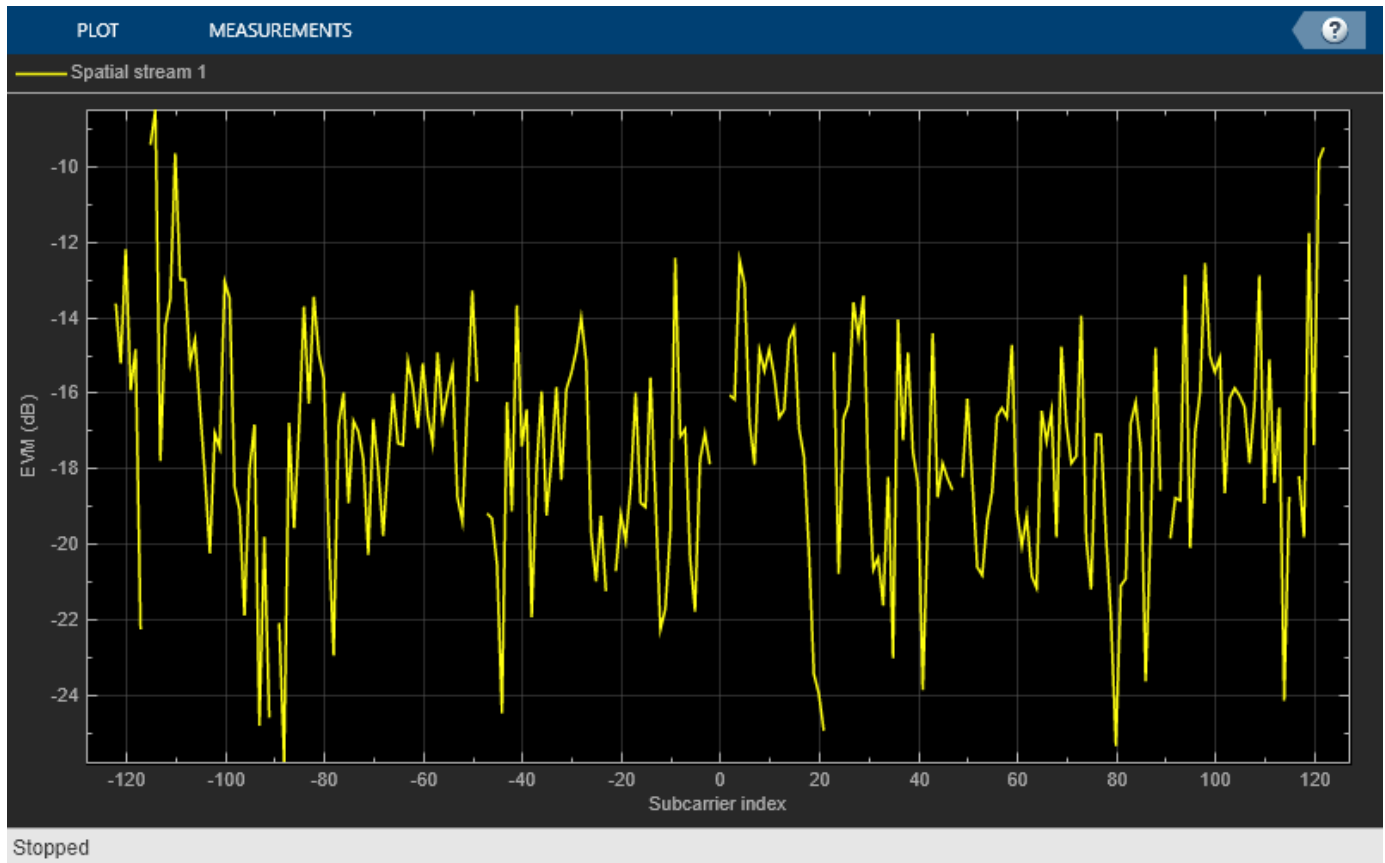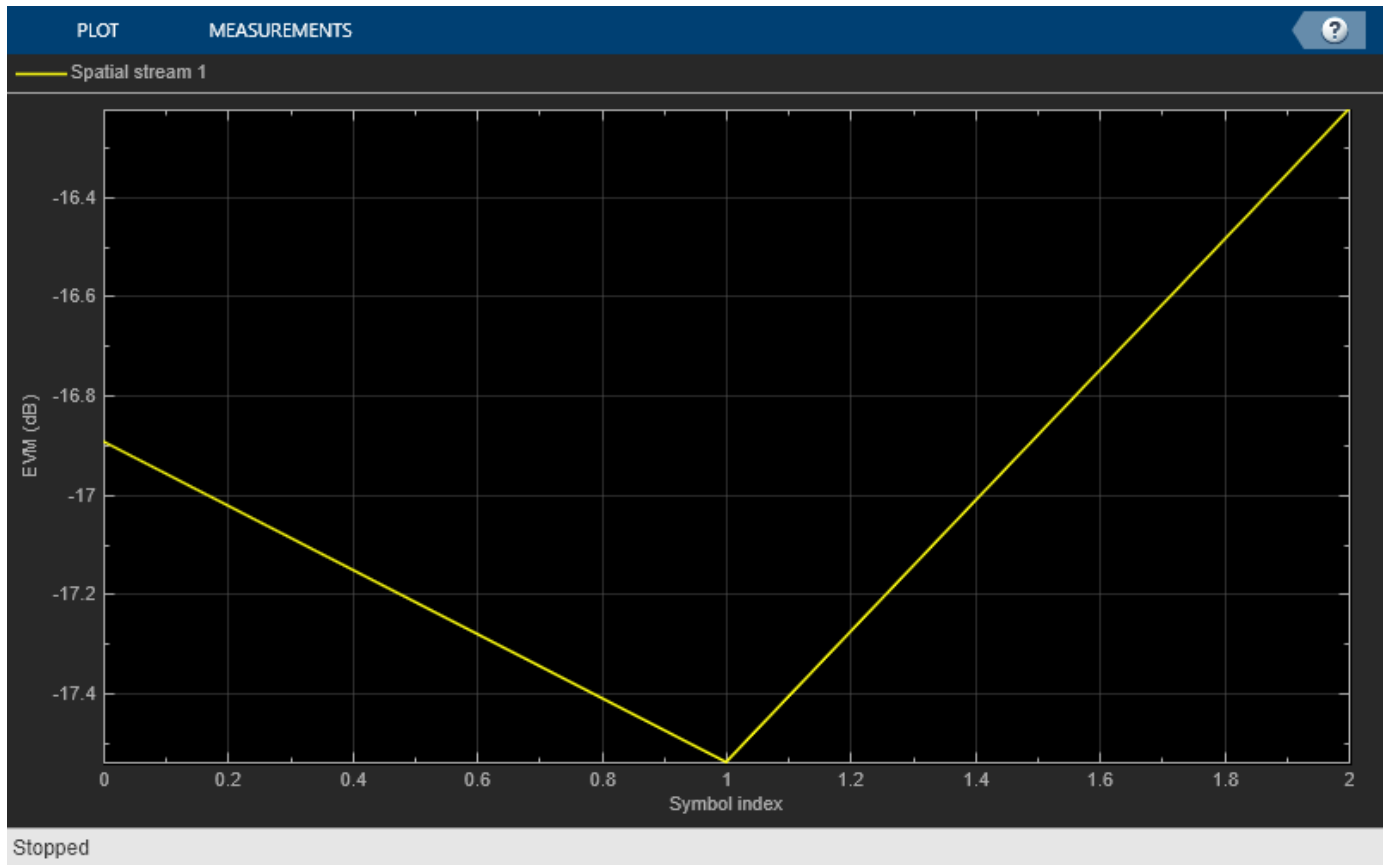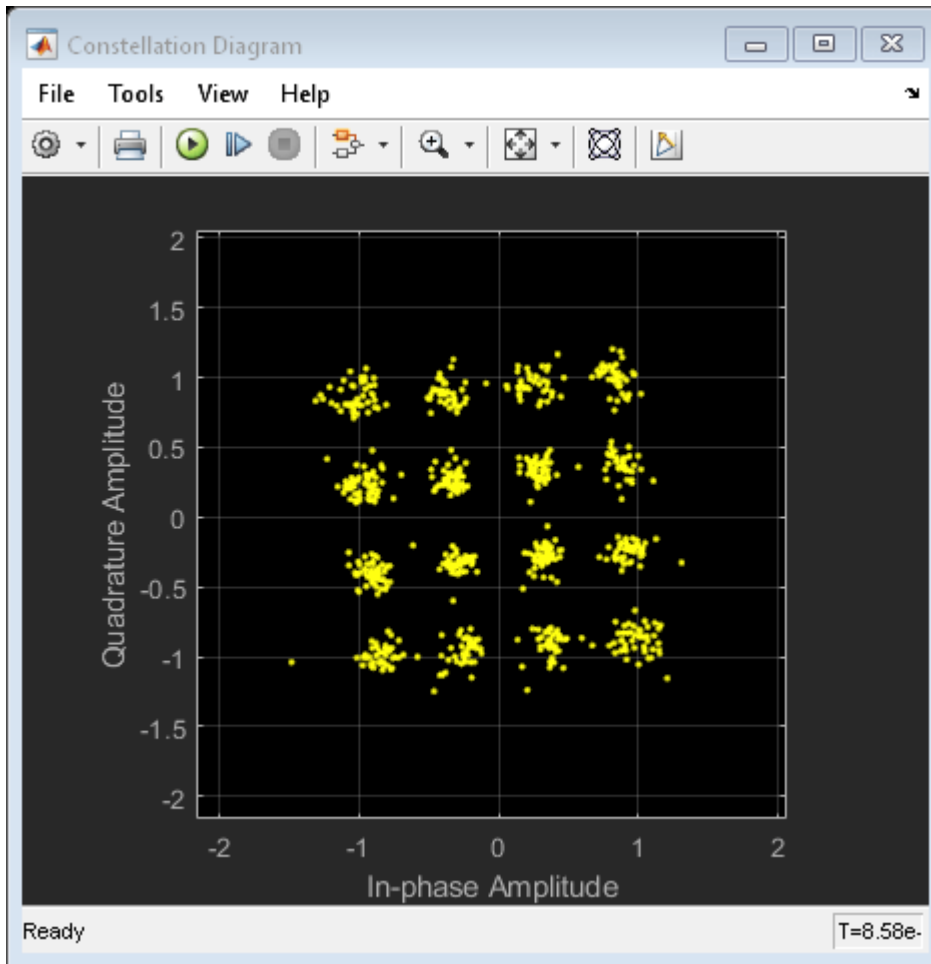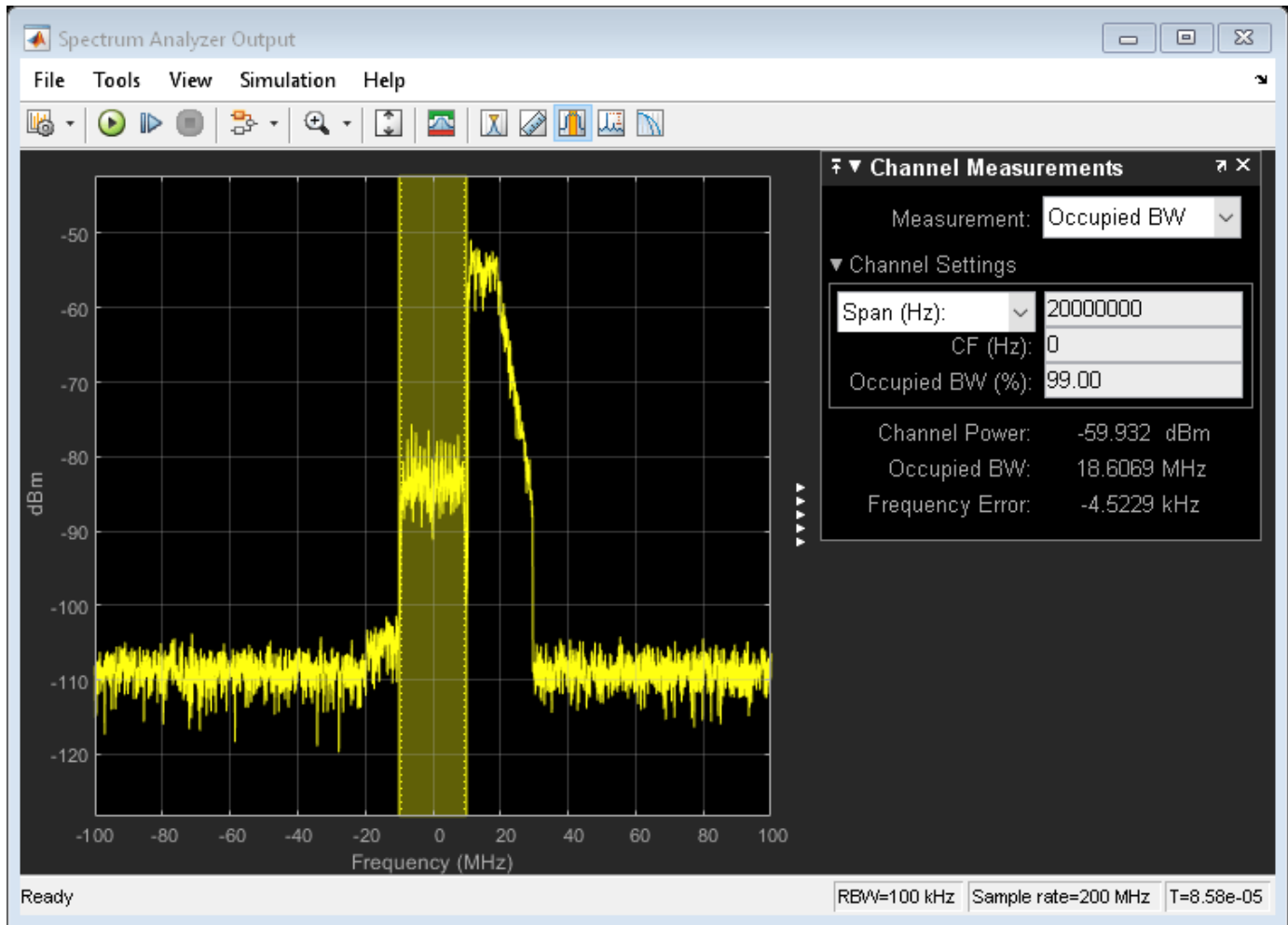
Compared to the case without interference, the constellation diagram is more distorted and the overall EVM is around -17 dB.

The ACR is around 28 dB. You can also measure the ACR when the interferer is an HE waveform. In this case, to measure the ACR when the PER is approximately 10% for a PSDU length of 4096 octets [2], set the **Gain** value of the Interferer block to around -72.4 dB.

Summary and Further Exploration

This example demonstrates how to model and test the reception of an HE waveform coexisting with an NR waveform or another HE waveform. The RF receiver consists of bandpass filters, amplifiers, and a demodulator. To evaluate the impact of the NR interference, the example modifies the gain of the NR waveform and performs EVM, PER, and ACR measurements. You can explore the impact of altering the RF impairments. For example:

- Increase the phase noise by using **Phase noise offset (Hz)** and **Phase noise level (dBc/Hz)** parameters on the **Demodulator** tab of the RF Receiver block.

- Decrease the LO to RF isolation by using the **LO to RF isolation (dB):** parameter on the **Demodulator** tab of the RF Receiver block.

This example configures the RF Receiver block to work with the default values of the HE Waveform and NR Interferer blocks and with the HE and NR carriers centered at 5950 MHz and 5970 MHz, respectively. These carriers are within the IEEE 802.11 HE frequency bands (between 1 GHz and 7.125 GHz [2]) and the NR operating band n96 [3]. If you change the carrier frequencies or the waveform configurations, you may need to update the parameters of the RF Receiver block as these parameters have been selected to work for the default configuration of the example. For instance, a change in the HE carrier frequency requires revising the bandwidth of the filters. Modifying the waveform bandwidth may require updating the **Impulse response duration** and **Phase noise frequency offset (Hz)** parameters of the Demodulator block. The phase noise offset determines the lower limit of the impulse response duration. If the phase noise frequency offset resolution is too high for a given impulse response duration, a warning message appears, specifying the minimum duration suitable for the required resolution. For more information, see Demodulator (RF Blockset).

This example could be the basis for testing the coexistence between HE and NR or HE waveforms for different RF configurations. You can replace the RF Receiver block with another RF subsystem of your choice and configure the model accordingly.

Bibliography

1   3GPP TS 38.141-1. "NR; Base Station (BS) conformance testing Part 1: Conducted conformance testing." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network.*

2   IEEE P802.11ax™/D7.0 Draft 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 6: Enhancements for High Efficiency WLAN.

3   3GPP TS 38.101-1. "NR; User Equipment (UE) radio transmission and reception." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network.*

Modeling and Testing an 802.11ax RF Transmitter

This example shows how to characterize the impact of radio frequency (RF) impairments in an 802.11ax transmitter. The example generates a baseband IEEE® 802.11ax™ waveform by using WLAN Toolbox™ and models the RF transmitter by using RF Blockset™.

Introduction

This example characterizes the impact of RF impairments such as in-phase and quadrature (IQ) imbalance, phase noise, and power amplifier (PA) nonlinearities in the transmission of an 802.11ax waveform [1]. To evaluate the impact of these impairments, the example performs these measurements:

- Error vector magnitude (EVM): vector difference at a given time between the ideal (transmitted) signal and the measured (received) signal
- Spectral mask: test that ensures a transmission in one channel does not cause substantial interference in adjacent channels
- Occupied bandwidth: bandwidth that contains 99% of the total integrated power of the signal, centered on the assigned channel frequency
- Channel power: filtered mean power centered on the assigned channel frequency
- Complementary cumulative distribution function (CCDF): probability that the signal's instantaneous power is at a specified level above its average power
- Peak-to-average power ratio (PAPR): relation between the peak power of the signal and its average power

The example works on a packet-by-packet basis. For each packet, the workflow consists of these steps:

1. Generate the baseband 802.11ax waveform by using WLAN Toolbox.
2. Oversample and filter the waveform by using a Finite Impulse Response (FIR) Interpolation block.
3. Import the baseband waveform as an RF signal into the RF Transmitter block implemented by using RF Blockset. The model uses an RF intermediate frequency to carry the baseband information in RF Blockset.
4. Upconvert the waveform to the carrier frequency by using an RF transmitter. You can accurately model the impairments introduced by an actual RF transmitter by using the RF components available in RF Blockset.
5. Calculate the spectral mask, occupied bandwidth, channel power, CCDF and PAPR by using the Spectrum Analyzer block.
6. Downsample and filter the waveform by using an FIR Decimation block.
7. Extract the data symbols and measure the EVM by demodulating the baseband waveform.

This example performs these operations by using a Simulink® model. The Simulink model carries out the baseband signal processing (steps 1, 2, 6 and 7) by using MATLAB® Function blocks and performs the RF transmitter modeling (steps 3 and 4) by using RF Blockset. This model supports `Normal` and `Accelerator` simulation modes.

Simulink Model Structure

The model contains three main parts:

- Baseband Waveform Generation: generates the baseband 802.11ax waveforms
- RF Transmission: upconverts the waveform to the carrier frequency
- Baseband Waveform Reception: performs the RF measurements and calculates EVM by demodulating the baseband waveform

```
modelName = 'HERFTransmitterModel';
open_system(modelName);
```

802.11ax RF Transmitter

Input waveform: 802.11ax (HE)
Measurements: EVM, channel power, occupied bandwidth, spectral mask, CCDF, PAPR
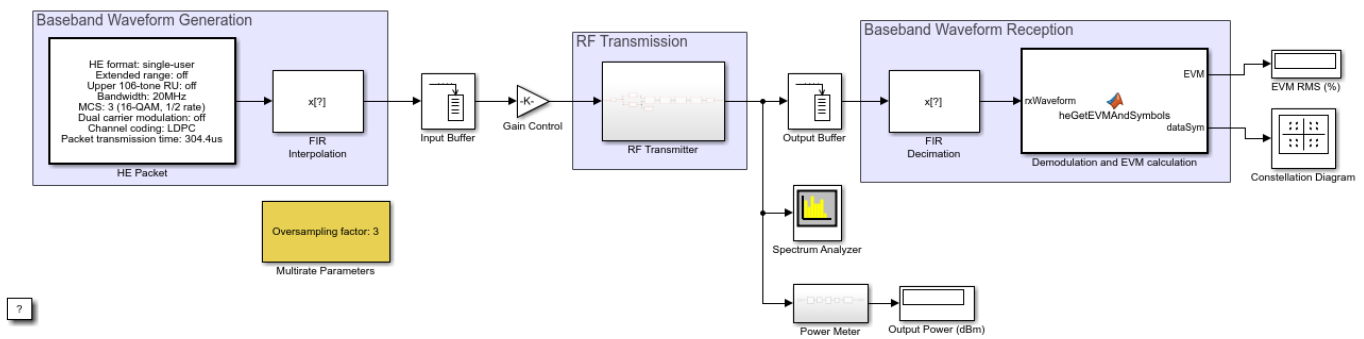
Copyright 2020 The MathWorks, Inc.

Baseband Waveform Generation

The HE Packet block generates standard-compliant high-efficiency single-user (HE SU) waveforms [1]. For the waveform generation, set transmission and configuration parameters by specifying options in the HE Packet block.

The HE Packet block contains two tabs:

- HE SU Format: configure the transmission parameters selected in this section by using a `wlanHESUConfig` object.
- Generator Configuration: generate each packet, which contains random data, with the `wlanWaveformGenerator` function for the specified HE SU configuration and parameters.

After generating each packet, the FIR Interpolation block oversamples and filters the waveform to allow you to see the effect of the high-power amplifier (HPA) on the out-of-band spectral emissions. At the output of the RF Transmitter block, the FIR Decimation block downsamples the waveform back to its original sampling rate. The Multirate Parameters block provides an interface to easily configure the parameters of the FIR Interpolation and Decimation blocks.
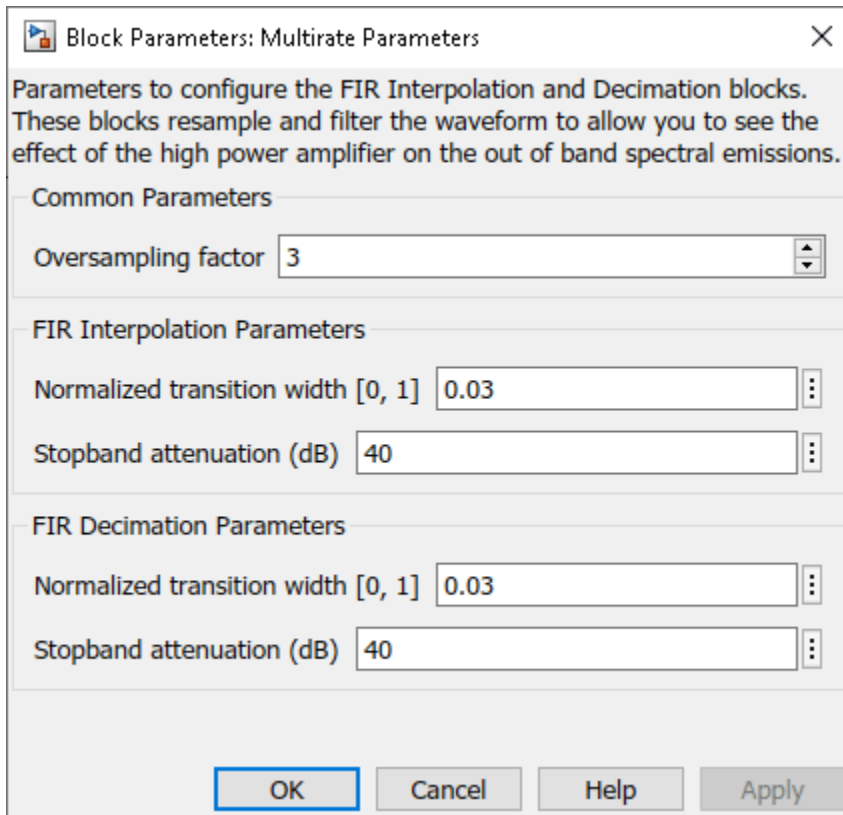
Specifying Simulation Time

The **Packet transmission time (μs)** parameter in the HE Packet block calculates the time required to transmit each 802.11ax packet. Hence, the **Stop Time** value in the Simulink model must be equal to or higher than the value depicted in **Packet transmission time (μs)** to obtain the EVM results and constellation diagram of at least one packet. As the filters in the FIR Interpolation and Decimation blocks introduce a delay, you can use the **Idle time (s)** parameter in the HE Packet block to compensate for the delay.

RF Transmission

The RF Transmitter block is based on a superheterodyne transmitter architecture. This architecture upconverts the waveform to the carrier frequency and applies passband filtering and amplification. The RF components of this superheterodyne transmitter are:

- IQ modulator consisting of mixers, a phase shifter, and a local oscillator
- Bandpass filter
- Power amplifier

In addition to these components, this RF Transmitter block also includes a variable gain amplifier (VGA) to control the input back-off (IBO) level of the HPA.

```
set_param(modelName,'Open','off');
set_param([modelName '/RF Transmitter'],'Open','on');
```

RF Superheterodyne Transmitter

This architecture upconverts the waveform to the carrier frequency and applies RF filtering and amplification before transmitting the signal.
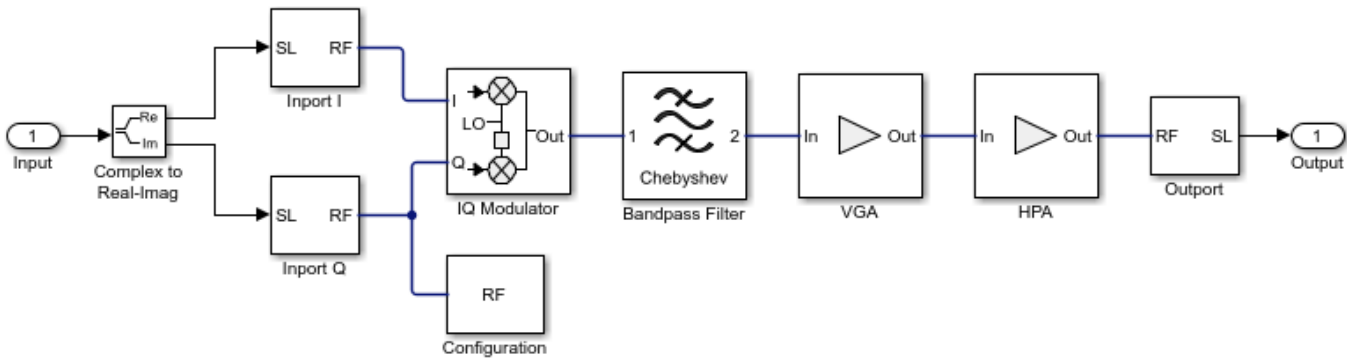
The Inport block inside the RF Transmitter converts the complex baseband waveform into the RF domain. You can vary the center frequency of this RF signal by modifying the **Carrier frequency** parameter of this block (the default frequency, which is considered to be an intermediate frequency in this example, is 70 MHz). The Outport block converts the RF signal back into complex baseband.

You can configure the RF Transmitter components by using the RF Transmitter block mask.
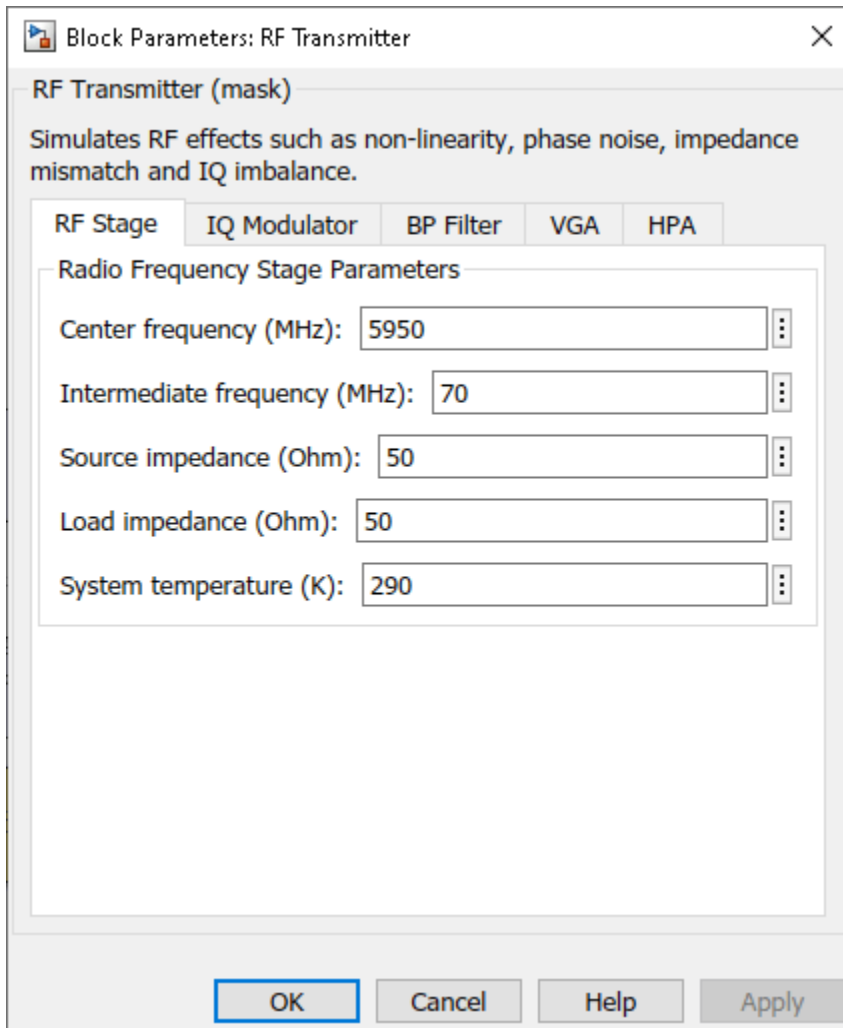
The RF Transmitter block exhibits typical impairments, including:

- I/Q imbalance as a result of gain or phase mismatches between the parallel sections of the transmitter chain dealing with the IQ signal paths
- Phase noise as an effect directly related to the thermal noise within the active devices of the oscillator
- PA nonlinearities due to DC power limitation when the amplifier works in the saturation region

You can use an Input Buffer block before the RF Transmitter block to reduce the number of samples sent to the RF Transmitter block. For simplicity, the Input Buffer in the current configuration sends one sample at a time, resulting in the RF Transmitter block being sample-based.

Adapt the power level of the baseband waveform to the RF configuration by adding a Gain Control block after the Input Buffer block.

As the current RF Transmitter block configuration sends one sample at a time, the Output Buffer block (after the RF Transmitter block) collects all samples within the baseband HE waveform before sending the samples onto the Demodulation and EVM calculation block.

Baseband Waveform Reception

The Demodulation and EVM calculation block recovers and plots the HE Data symbols in the Constellation Diagram block by performing frequency and packet offset corrections, channel estimation, pilot phase tracking, OFDM demodulation, and equalization. This block also performs these EVM measurements:

* EVM per subcarrier (dB): EVM averaged over the allocated HE Data symbols within a subcarrier
* EVM per OFDM symbol (dB)
* Overall EVM (dB and %): EVM averaged over all transmitted HE Data symbols

The default configuration of the Spectrum Analyzer block depicts the spectral mask according to IEEE P802.11ax/D7.0 Section 27.3.19.1 [1]. The Spectrum Analyzer block also provides additional measurements such as occupied bandwidth, channel power, CCDF and PAPR. The Power Meter block measures the RF waveform channel power, which is displayed in the Output Power (dBm) block.

Effect of Power Amplifier Nonlinearities

To characterize the impact of HPA nonlinearities in the EVM evaluation, you can measure the amplitude-to-amplitude modulation (AM/AM) of the HPA. The AM/AM refers to the output power levels in terms of the input power levels. The helper function `hePlotHPACurve` displays the AM/AM characteristic of the HPA selected for this model.

```
hePlotHPACurve();
figHPA = gcf;
```
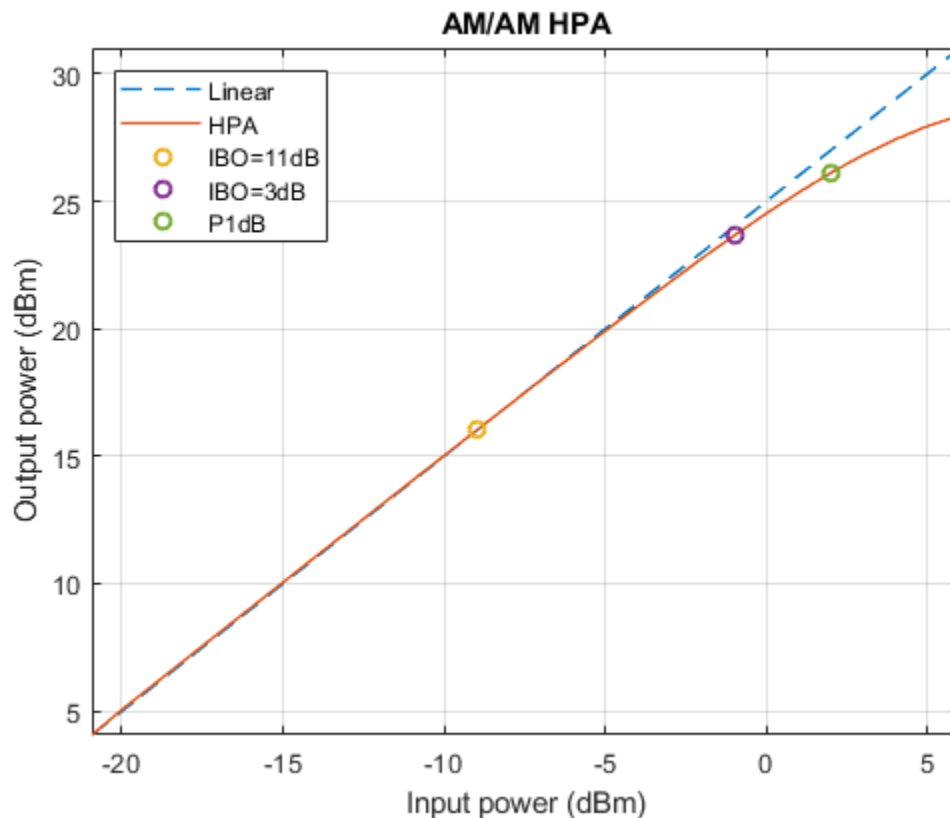
P1dB is the power at 1 dB compression point and is normally used as a reference when selecting the IBO level of the HPA. You can see the HPA impact on the RF Transmitter block by analyzing the EVM results for different operating points of the HPA. For example, compare the case when IBO = 11 dB, corresponding to HPA operating in the linear region, with the case when IBO = 3 dB, corresponding to HPA operating in saturation. The gain of the VGA controls the IBO level. To keep a VGA linear behavior using the default parameters, select gain values lower than 15 dB.
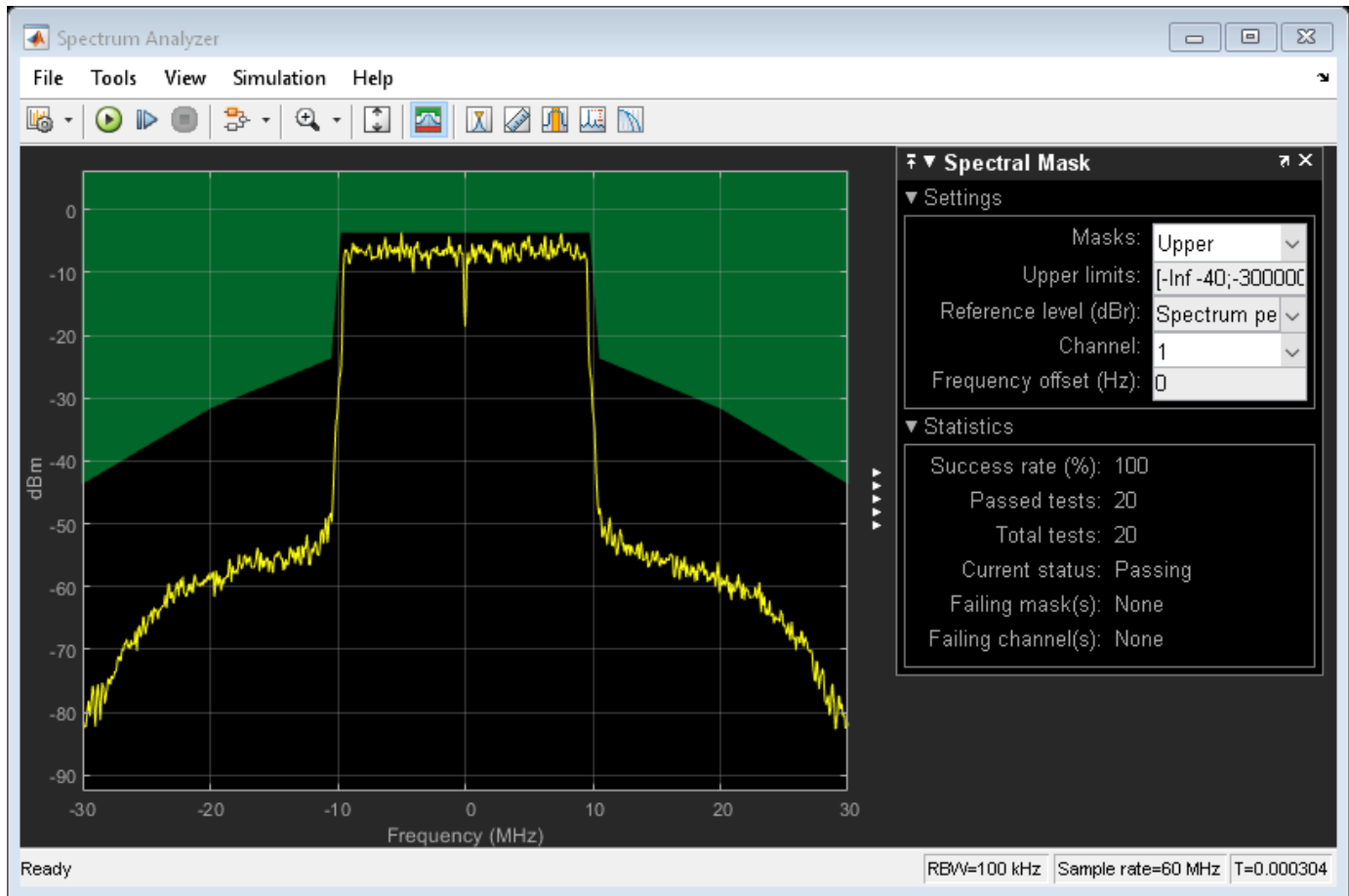
- *Linear HPA (IBO = 11 dB)*. To operate at an IBO level of 11 dB, set the **Available power gain** parameter of the VGA block to 5 dB. To calculate the EVM and plot the constellation diagram, run the simulation long enough to capture one packet (Stop Time equal to 304.4 us for the default configuration).

```
set_param([modelName '/RF Transmitter'],'vgaGain','5');
sim(modelName);
```
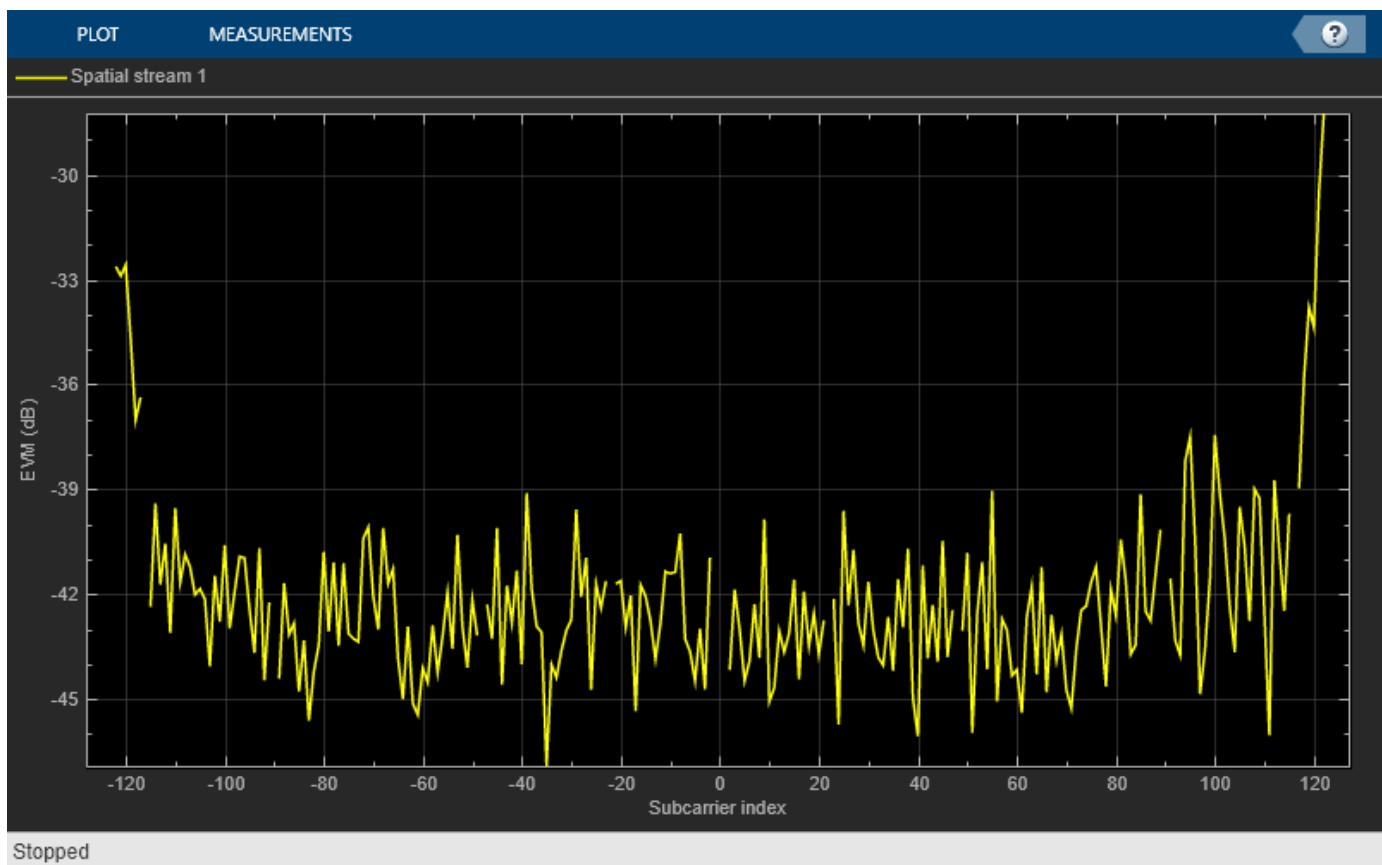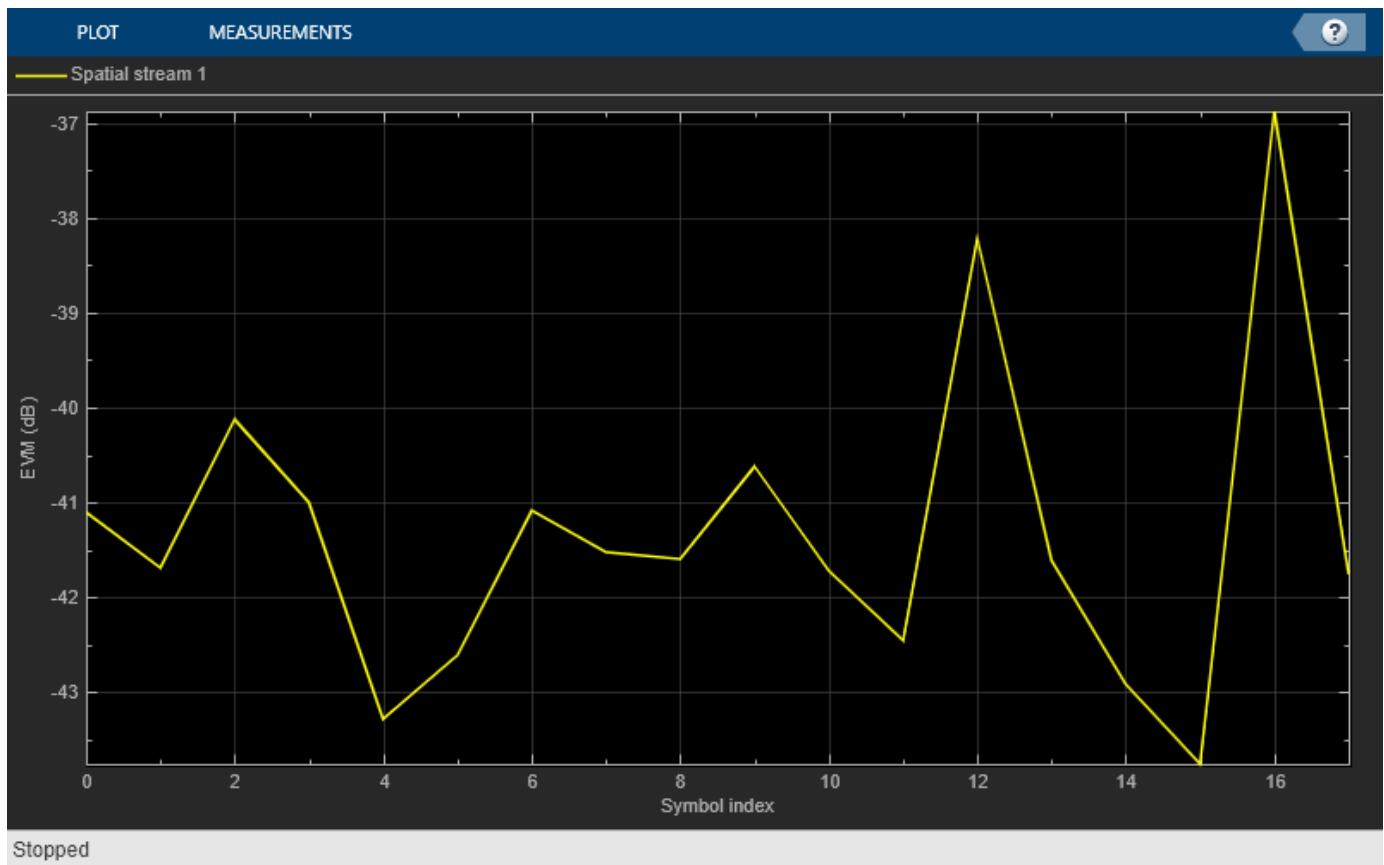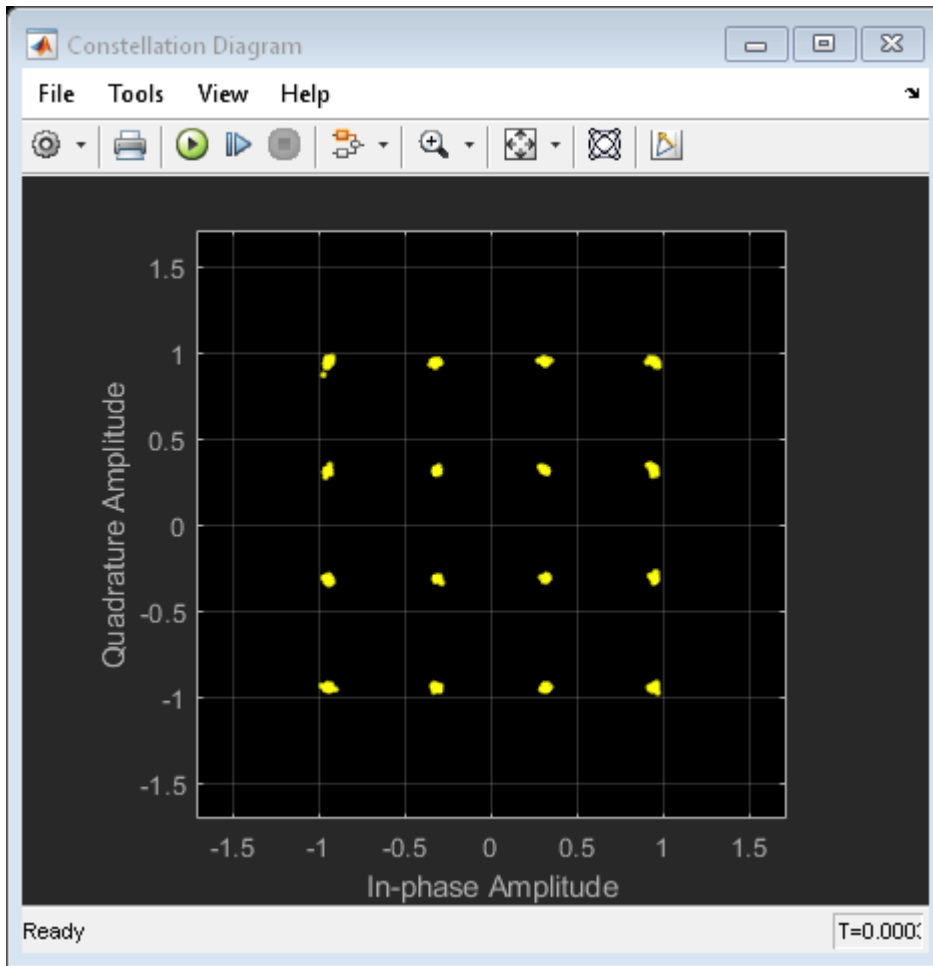
According to IEEE P802.11ax/D7.0 Table 27-49 [1], the allowed relative constellation error (EVM) in an HE SU PPDU when the **Dual carrier modulation** parameter is disabled and the **Modulation/ coding** is equal to 3 (16-QAM, 1/2) is -16 dB. As the overall EVM, around -41 dB, is lower than -16 dB, this architecture falls within the requirements of IEEE P802.11ax/D7.0 [1].

- *Nonlinear HPA (IBO = 3 dB).* To operate at an IBO level of 3 dB, set the **Available power gain** parameter of the VGA block to 13 dB.

```
set_param([modelName '/RF Transmitter'],'vgaGain','13');
sim(modelName);
slmsgviewer.DeleteInstance();

% Restore to default parameters
set_param([modelName '/RF Transmitter'],'vgaGain','5');
```
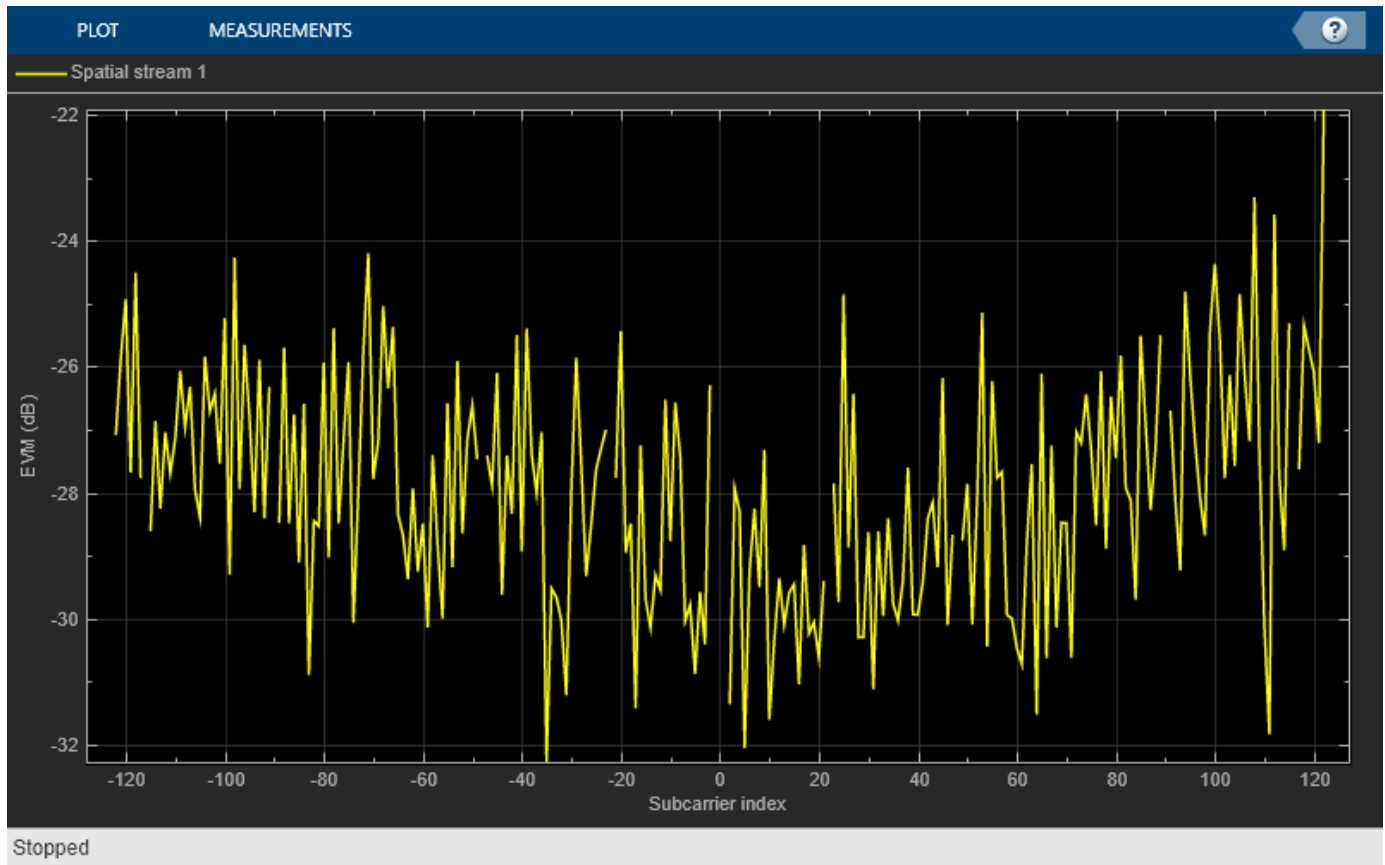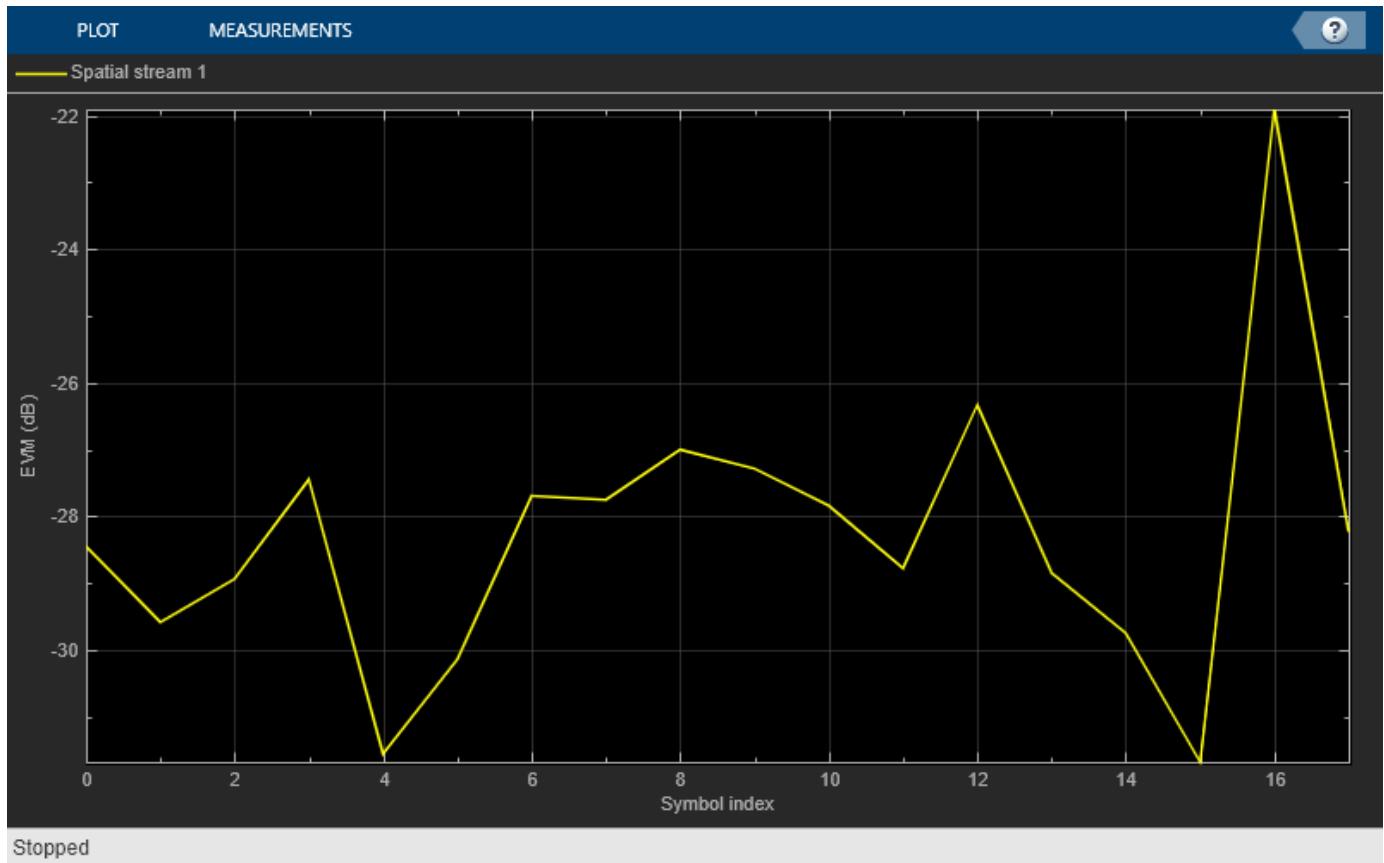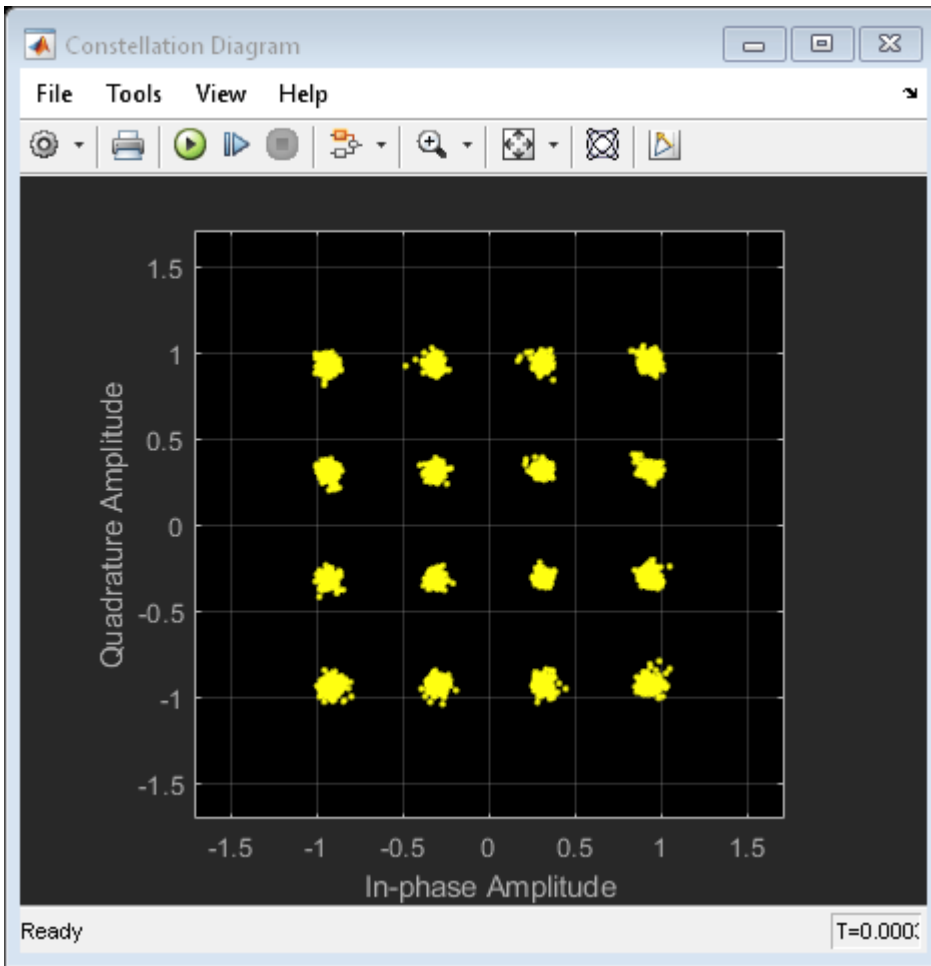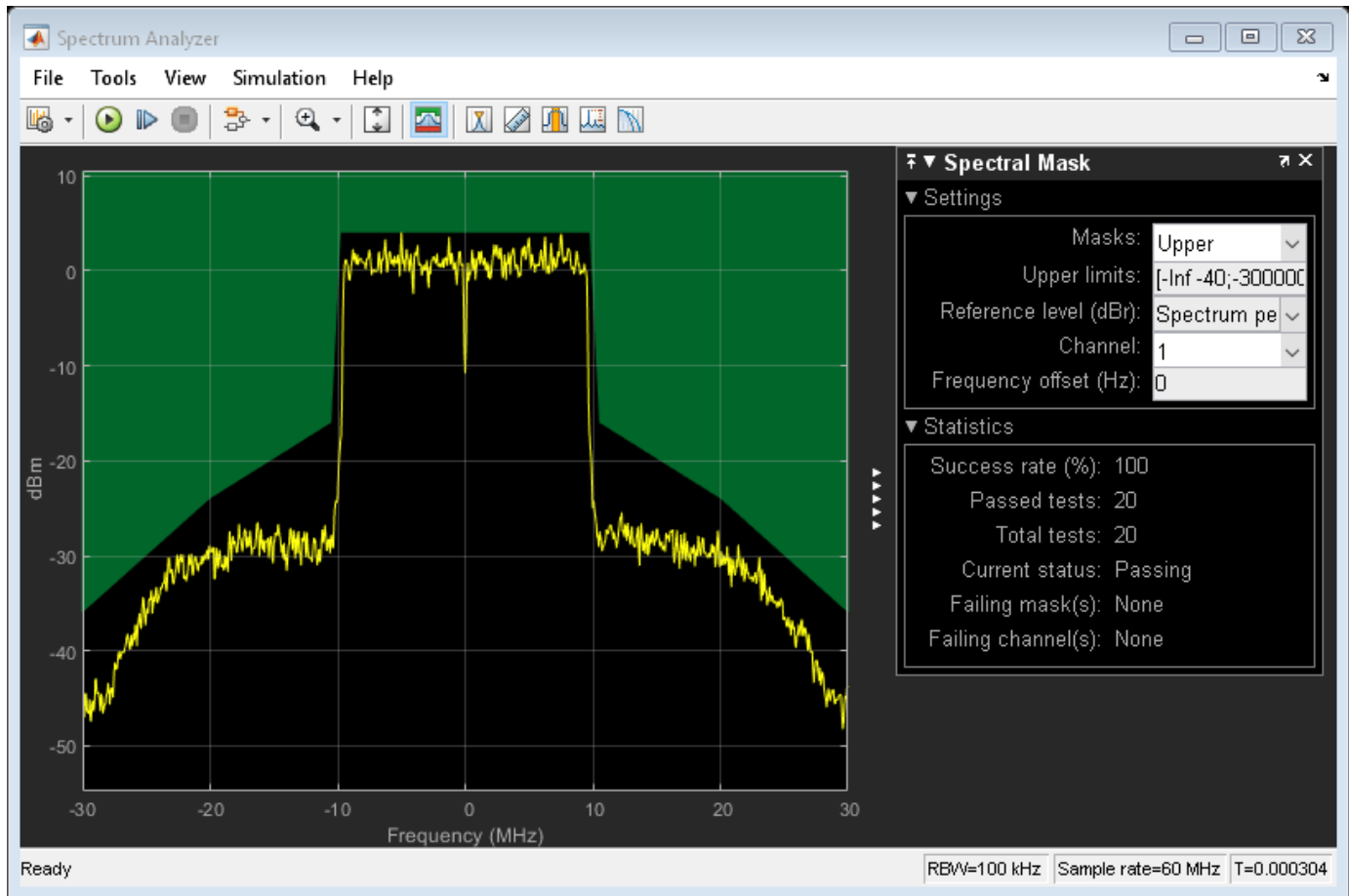
Compared to the previous case, the constellation diagram is more distorted. In terms of measurements, the overall EVM, around -28 dB, is still lower than -16 dB, so it also falls within the requirements of IEEE P802.11ax/D7.0 [1].

Summary and Further Exploration

This example demonstrates how to model and test the transmission of an 802.11ax waveform. The RF Transmitter block consists of a bandpass filter, amplifiers and an IQ modulator. The example highlights the effect of HPA nonlinearities on the performance of the RF Transmitter block. You can explore the impact of altering other impairments as well. For example:

- Increase I/Q imbalance by using the **I/Q gain mismatch (dB)** and **I/Q phase mismatch (Deg)** parameters on the **IQ Modulator** tab of the RF Transmitter block.
- Increase the phase noise by using **Phase noise offset (Hz)** and **Phase noise level (dBc/Hz)** parameters on the **IQ Modulator** tab of the RF Transmitter block.

Additionally, you can check the occupied bandwidth and the CCDF and PAPR measurements by using the Spectrum Analyzer window: select the Channel Measurements button on the toolbar for the occupied bandwidth and the CCDF Measurements button for the CCDF and PAPR.

The RF Transmitter block is configured to work with the default values of the HE Packet block and with the RF carrier centered at 5950 MHz. This carrier is within the IEEE 802.11 HE STA frequency bands (between 1 GHz and 7.125 GHz [1]). If you change the carrier frequency or the values in the

HE Packet block, you may need to update the parameters of the RF Transmitter block components, as these parameters have been selected to work for the default configuration of the example. For instance, a change in the carrier frequency requires revising the bandwidth of the filters. Modifying the waveform bandwidth may require updating the **Impulse response duration** and **Phase noise frequency offset (Hz)** parameters of the IQ Modulator block. The phase noise offset determines the lower limit of the impulse response duration. If the phase noise frequency offset resolution is too high for a given impulse response duration, a warning message appears, specifying the minimum duration suitable for the required resolution. For more information, see IQ Modulator (RF Blockset).

This example could be the basis for testing HE waveforms for different RF configurations. You can replace the RF Transmitter block by another RF subsystem of your choice and configuring the model accordingly.

References

1   IEEE P802.11ax™/D7.0 Draft 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 6: Enhancements for High Efficiency WLAN.

802.11ac Receiver Minimum Input Sensitivity Test

This example shows how to measure the receiver minimum input sensitivity as specified in Section 22.3.19.1 of the IEEE® 802.11ac™ standard [1].

Introduction

The receiver minimum sensitivity test ensures a device under test (DUT) receives data with a defined maximum packet error rate (PER) of 10% at a defined minimum signal power. The minimum signal power depends on the channel bandwidth and modulation and coding scheme (MCS) as specified in Table 22-25 of the IEEE 802.11ac standard [1]:

MCS	Modulation	Rate	Minimum Sensitivity [dBm]			
			20 MHz	40 MHz	80 MHz	160 MHz
0	BPSK	1/2	-82	-79	-76	-73
1	QPSK	1/2	-79	-76	-73	-70
2	QPSK	3/4	-77	-74	-71	-68
3	16-QAM	1/2	-74	-71	-68	-65
4	16-QAM	3/4	-70	-67	-64	-61
5	64-QAM	2/3	-66	-63	-60	-57
6	64-QAM	3/4	-65	-62	-59	-56
7	64-QAM	5/6	-64	-61	-58	-55
8	256-QAM	3/4	-59	-56	-53	-50
9	256-QAM	5/6	-57	-54	-51	-48

When the test is performed with hardware, each input antenna port on the DUT is connected through a cable to a single output antenna port of a transmitter. To perform the test, specify these parameters for the test waveform:

- Number of spatial streams - equal to the number of transmit antennas
- PSDU length, in bytes - 4096
- Space-time block coding (STBC) - disabled
- Guard interval, in nanoseconds - 800
- Channel coding - binary convolutional coding (BCC)

This example shows how to simulate the test by using WLAN Toolbox™. VHT packets stimulate a receiver at a range of input levels below the minimum sensitivity level. The example then measures the packet error rate for each sensitivity level.

The example simulates the test by performing these steps over a range of sensitivity levels:

- Generate and scale packets to the desired signal level
- Add white Gaussian noise is to create a noise floor at the receiver
- Demodulate the noisy packets to recover PSDUs.
- Compare recovered PSDUs to those transmitted to determine the number of packet errors and hence the packet error rate.

Automatic gain control (AGC), packet detection, timing synchronization, carrier frequency offset correction, noise estimation and phase tracking are performed by the example receiver. This diagram demonstrates processing for each packet:
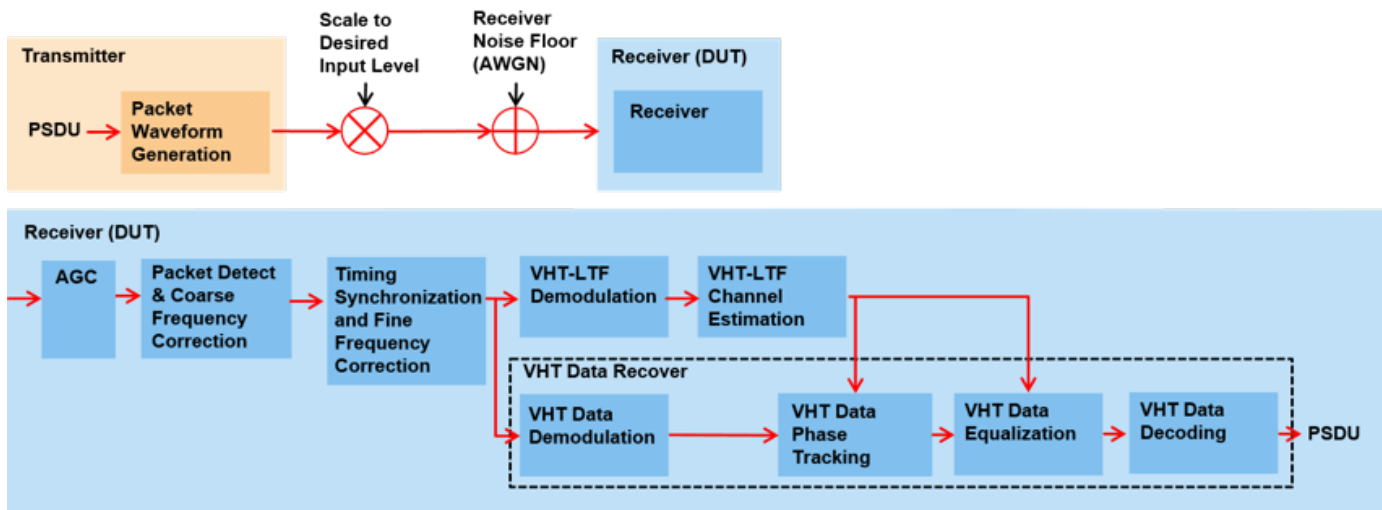
Test Parameters

Configure a transmission for the test by using a VHT configuration object. This example measures the minimum sensitivity for a 160 MHz transmission with 64-QAM rate 5/6 modulation and coding. The simulated DUT has 2 receive antennas. Test different configurations by changing these parameters.

```
cfgVHT = wlanVHTConfig;                  % Create VHT transmission configuration
cfgVHT.ChannelBandwidth = 'CBW160'; % Bandwidth
cfgVHT.MCS = 7;                          % 64-QAM, rate 5/6
NumReceiveAntennas = 2;                  % Number of receive antennas
```

The test requires these fixed transmission parameters.

```
cfgVHT.APEPLength = 4096; % Bytes
cfgVHT.STBC = false;
cfgVHT.NumTransmitAntennas = NumReceiveAntennas;
cfgVHT.NumSpaceTimeStreams = NumReceiveAntennas;
cfgVHT.SpatialMapping = 'Direct';
cfgVHT.GuardInterval = 'Long';
```

Simulation Parameters

A receiver processes VHT packets at a range of input levels below the minimum input sensitivity level. Specify the range of offsets to test in the vector `testInputLevelOffsets`.

```
testInputLevelOffsets = [-10 -9 -8 -7]; % dB
```

Control the number of packets tested at each sensitivity by specifying these parameters:

1 `maxNumErrors` is the maximum number of packet errors simulated at each input level. When the number of packet errors reaches this limit, the simulation at this sensitivity level is complete.

2 `maxNumPackets` is the maximum number of packets simulated at each input level and limits the length of the simulation if the packet error limit is not reached.

The numbers chosen in this example lead to a very short simulation. Increase `maxNumErrors` and `maxNumPackets` for meaningful results.

```
maxNumErrors = 20;
maxNumPackets = 200;
```

Signal Power Setup

The minimum sensitivity test specifies a maximum PER for a measured input level per receive antenna. In this simulation the receiver processes a test signal with a specified input level in dBm. Generate the test signal using the `wlanWaveformGenerator` function. The `wlanWaveformGenerator` function normalizes the waveform such that the power for all antennas sums to 0 dBm. Therefore, scale the output of the waveform generator to create the desired input level.

```
% Receiver minimum input level sensitivity for 20 MHz, Table 22-25. The
% sensitivity increases by 3dB for double the bandwidth.
rxMinSensitivityTable = [-82 -79 -77 -74 -70 -66 -65 -64 -59 -57]; % dBm

% Get minimum input sensitivity given MCS and bandwidth
fs = wlanSampleRate(cfgVHT);  % Baseband sampling rate (Hz)
B = floor(10*log10((fs/20e6))); % Scalar for bandwidth
rxMinSensitivity = rxMinSensitivityTable(cfgVHT.MCS+1)+B; % dBm
disp(['Minimum sensitivity for MCS' num2str(cfgVHT.MCS) ', ' ...
    num2str(fs/1e6) ' MHz: ' num2str(rxMinSensitivity,'%2.1f') ' dBm'])
```

```
Minimum sensitivity for MCS7, 160 MHz: -55.0 dBm
```

Define the range of input levels below the minimum level to test using `testInputLevels`.

```
testInputLevels = rxMinSensitivity+testInputLevelOffsets; % dBm
```

Calculate a voltage scalar, `A`, to scale the generated waveform for each test level. The power per receive antenna port is measured during the simulation to confirm the input signal level is correct.

```
A = 10.^((testInputLevels-30)/20);       % Voltage gain (attenuation)
A = A*sqrt(cfgVHT.NumTransmitAntennas); % Account for generator scaling
```

Noise Configuration

Add thermal noise at the receiver. The height of the noise floor determines the SNR at the receiver, as the input signal level is fixed for this test. The noise figure of the receiver determines the level of noise floor.

```
NF = 6;          % Noise figure (dB)
T = 290;         % Ambient temperature (K)
BW = fs;         % Bandwidth (Hz)
k = 1.3806e-23; % Boltzmann constant (J/K)
noiseFloor = 10*log10(k*T*BW)+NF; % dB
disp(['Receiver noise floor: ' num2str(noiseFloor+30,'%2.1f') ' dBm'])
```

```
Receiver noise floor: -85.9 dBm
```

Add noise to the waveform using an AWGN channel, `comm.AWGNChannel`.

```
awgnChannel = comm.AWGNChannel('NoiseMethod','Variance', ...
    'Variance',10^(noiseFloor/10));
```

Input Level Sensitivity Simulation

Calculate the packet error rate for each input level by simulating multiple packets.

For each packet perform the following processing steps:

1   Create and encode a PSDU to create a single packet waveform.

2   Create the desired input level in dBm by scaling the waveform.

3   Measure the power of the received waveform.

4   Add AWGN to the received waveform.

5   Boost the signal prior to processing by passing through an automatic gain control.

6   Detect the packet.

7   Estimate and correct coarse carrier frequency offset.

8   Establish fine timing synchronization.

9   Estimate and correct fine carrier frequency offset.

10   Extract and OFDM demodulate the VHT-LTF and perform channel estimation.

11   Extract the VHT Data field and recover the PSDU.

```matlab
ind = wlanFieldIndices(cfgVHT); % For accessing fields within the packet
chanBW = cfgVHT.ChannelBandwidth;
rng(0);  % Set random state for repeatability

agc = comm.AGC; % Automatic gain control

S = numel(testInputLevels);
packetErrorRate = zeros(S,1);
rxAntennaPower = zeros(S,1);
for i=1:S
    disp(['Simulating ' num2str(testInputLevels(i),'%2.1f') ...
        ' dBm input level...']);

    % Loop to simulate multiple packets
    numPacketErrors = 0;
    measuredPower = zeros(maxNumPackets,1); % Average power per antenna
    numPkt = 1; % Index of packet transmitted
    while numPacketErrors<=maxNumErrors && numPkt<=maxNumPackets
        % Generate a packet waveform
        txPSDU = randi([0 1],cfgVHT.PSDULength*8,1); % PSDULength in bytes
        tx = wlanWaveformGenerator(txPSDU,cfgVHT);

        % Scale input signal to desired level
        rx = tx.*A(i);

        % Measure the average power at the antenna connector in Watts
        measuredPower(numPkt) = mean(mean(rx.*conj(rx)));

        % Add noise floor at receiver
        rx = awgnChannel(rx);

        % Pass each channel through AGC
        for ic = 1:size(rx,2)
            rx(:,ic) = agc(rx(:,ic));
            reset(agc);
        end

        % Packet detect and determine coarse packet offset
        coarsePktOffset = wlanPacketDetect(rx,chanBW);
        if isempty(coarsePktOffset) % If empty no L-STF detected; packet error
            numPacketErrors = numPacketErrors+1;
```

```matlab
        numPkt = numPkt+1;
        continue; % Go to next loop iteration
    end

    % Extract L-STF and perform coarse frequency offset correction
    lstf = rx(coarsePktOffset+(ind.LSTF(1):ind.LSTF(2)),:);
    coarseFreqOff = wlanCoarseCFOEstimate(lstf,chanBW);
    rx = helperFrequencyOffset(rx,fs,-coarseFreqOff);

    % Extract the non-HT fields and determine fine packet offset
    nonhtfields = rx(coarsePktOffset+(ind.LSTF(1):ind.LSIG(2)),:);
    finePktOffset = wlanSymbolTimingEstimate(nonhtfields,chanBW);

    % Determine final packet offset
    pktOffset = coarsePktOffset+finePktOffset;
    % if packet detected out of a reasonable range (>50 samples);
    % packet error
    if pktOffset>50
        numPacketErrors = numPacketErrors+1;
        numPkt = numPkt+1;
        continue; % Go to next loop iteration
    end

    % Extract L-LTF and perform fine frequency offset correction
    lltf = rx(pktOffset+(ind.LLTF(1):ind.LLTF(2)),:);
    fineFreqOff = wlanFineCFOEstimate(lltf,chanBW);
    rx = helperFrequencyOffset(rx,fs,-fineFreqOff);

    % Extract VHT-LTF samples from the waveform, demodulate and perform
    % channel estimation
    vhtltf = rx(pktOffset+(ind.VHTLTF(1):ind.VHTLTF(2)),:);
    vhtltfDemod = wlanVHTLTFDemodulate(vhtltf,cfgVHT);
    chanEst = wlanVHTLTFChannelEstimate(vhtltfDemod,cfgVHT);

    % Get single stream channel estimate
    chanEstSSPilots = vhtSingleStreamChannelEstimate(vhtltfDemod,cfgVHT);

    % Extract VHT Data samples from the waveform
    vhtdata = rx(pktOffset+(ind.VHTData(1):ind.VHTData(2)),:);

    % Estimate the noise power in VHT data field
    nEstVHT = vhtNoiseEstimate(vhtdata,chanEstSSPilots,cfgVHT);

    % Recover the transmitted PSDU in VHT Data
    rxPSDU = wlanVHTDataRecover(vhtdata,chanEst,nEstVHT,cfgVHT);

    % Determine if any bits are in error, i.e. a packet error
    packetError = any(biterr(txPSDU,rxPSDU));
    numPacketErrors = numPacketErrors+packetError;
    numPkt = numPkt+1;
end

% Calculate packet error rate (PER) at input level point
packetErrorRate(i) = numPacketErrors/(numPkt-1);
disp(['  Completed after ' ...
    num2str(numPkt-1) ' packets, PER: ' ...
    num2str(packetErrorRate(i))]);
```

```
    % Calculate average input power per antenna
    rxAntennaPower(i) = 10*log10(mean(measuredPower(1:(numPkt-1))))+30;
    disp(['  Measured antenna connector power: ' ...
        num2str(rxAntennaPower(i),'%2.1f') ' dBm']);
end

Simulating -65.0 dBm input level...
  Completed after 21 packets, PER: 1
  Measured antenna connector power: -65.0 dBm
Simulating -64.0 dBm input level...
  Completed after 26 packets, PER: 0.80769
  Measured antenna connector power: -64.0 dBm
Simulating -63.0 dBm input level...
  Completed after 130 packets, PER: 0.16154
  Measured antenna connector power: -63.0 dBm
Simulating -62.0 dBm input level...
  Completed after 200 packets, PER: 0.02
  Measured antenna connector power: -62.0 dBm
```

Analysis and Further Exploration

Plot the PER for tested input signal levels with the maximum PER at minimum sensitivity.

```
figure
semilogy(rxAntennaPower,packetErrorRate,'o-')
hold on
semilogy(rxMinSensitivity,0.1,'rx')
currentxlim = xlim(gca);
xlim([currentxlim(1) currentxlim(2)+1])
grid on
xlabel('Measured power per antenna connector (dBm)');
ylabel('PER');
legend('Simulated PER performance','Maximum PER at minimum sensitivity');
title(sprintf(['Minimum Input Sensitivity Test: MCS%d, %d MHz, ' ...
    '%d Antennas'],cfgVHT.MCS,fs/1e6,cfgVHT.NumTransmitAntennas))
```
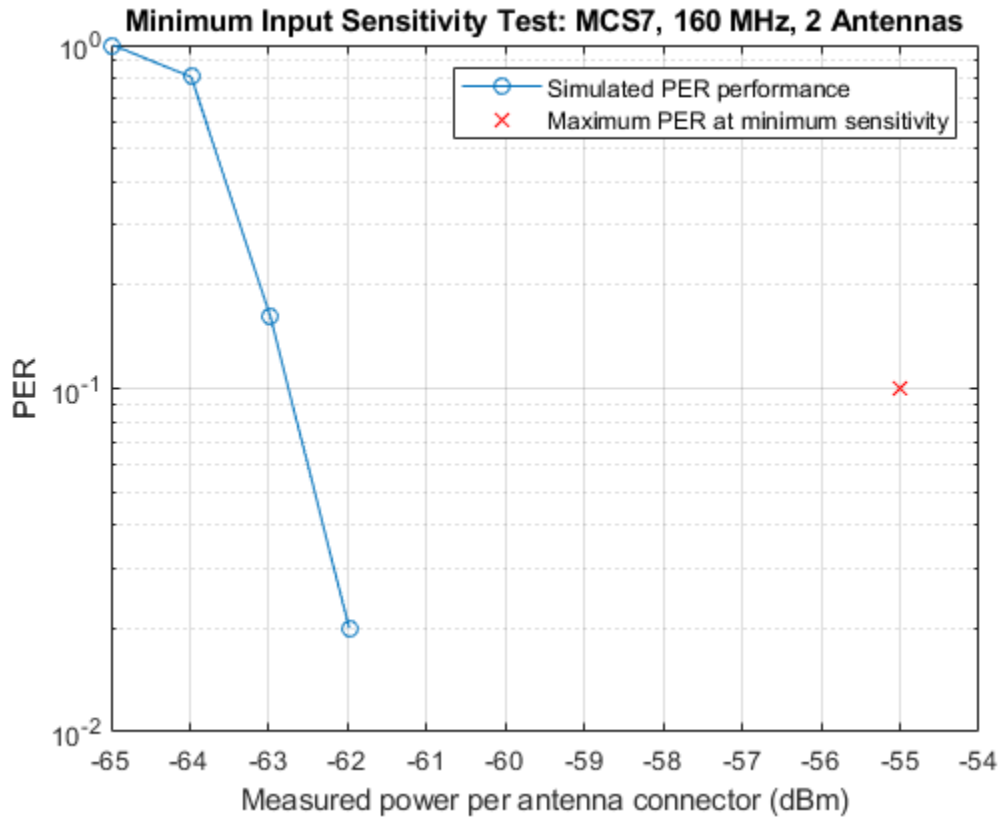
The plot reveals the simulated 10% PER is just under 8 dB lower than the minimum sensitivity specified by the test. This difference is due to the implementation margin allowed by the test. The implementation margin allows for algorithmic degradations due to impairments and the receiver noise figure when compared to ideal AWGN performance [2]. In this example only AWGN is added as an impairment. Therefore, only the algorithmic performance of front-end synchronization, channel estimation and phase tracking in the presence of AWGN use the implementation margin. If more impairments are included in the simulation the PER waterfall in the plot will move right towards the minimum sensitivity and the margin will decrease.

The number of packets tested at each SNR point is controlled by two parameters; `maxNumErrors` and `maxNumPackets`. For meaningful results, you should use larger numbers than those used in this example.

Appendix

This example uses the following helper functions:

- helperFrequencyOffset.m
- vhtNoiseEstimate.m
- vhtSingleStreamChannelEstimate.m

Selected Bibliography

1   IEEE Std 802.11ac™-2013 IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific

requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz.

2 Perahia, Eldad, and Robert Stacey. Next Generation Wireless LANS: 802.11n and 802.11ac. Cambridge University Press, 2013.

802.11ac Transmitter Measurements

This example shows how to perform these transmitter measurements on an IEEE® 802.11ac™ waveform:

- Modulation accuracy
- Spectrum emission mask
- Spectrum flatness
- In-phase and quadrature (IQ) gain and phase imbalance

Introduction

The transmitter modulation accuracy, required spectrum mask, and required spectral flatness for a given configuration is specified in [Section 21.3.17 of 1]. This example shows how to perform these measurements on a waveform. This example also models, measures, and corrects the IQ gain and phase imbalance. Generate the waveform with WLAN Toolbox™ or use a captured waveform with a spectrum analyzer.

The example generates 20 VHT packets with an 80 MHz channel bandwidth and a 10 microsecond gap between packets. Each packet contains random data and uses 256-QAM modulation. To reduce the out-of-band emmissions and meet the spectral mask requirement, upsample and filter the baseband waveform. Add the IQ gain and phase imbalance to the filtered waveform. Use a high-power amplifier (HPA) model to introduce inband distortion and spectral regrowth. Perform the spectral emission mask measurement on the upsampled waveform after the high-power amplifier modeling. Downsample and correct the waveform with the estimated IQ gain and phase imbalance. Measure the error vector magnitude (EVM) of the VHT Data field to determine the modulation accuracy. Additionally, measure the spectral flatness. This diagram shows the workflow contained in the example.
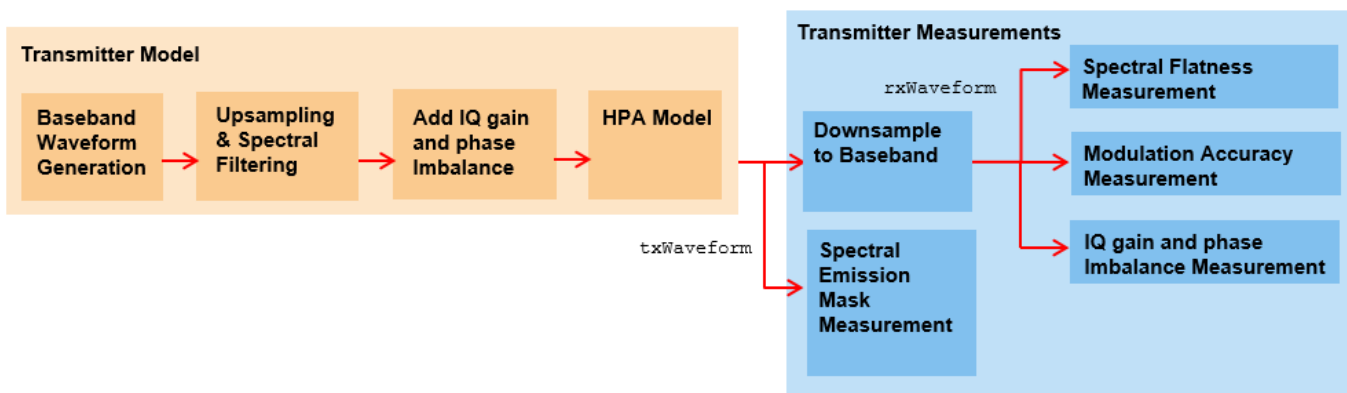
IEEE 802.11ac VHT Packet Configuration

This example generates an IEEE 802.11ac waveform consisting of multiple VHT format packets. Use the VHT format configuration object, `wlanVHTConfig`, to configure transmission properties of a VHT packet. This example configures the VHT waveform for a 80 MHz bandwidth. Because this example does not use space-time block coding, it can measure the modulation accuracy per spatial stream.

```
cfgVHT = wlanVHTConfig;            % Create packet configuration
cfgVHT.ChannelBandwidth = 'CBW80'; % 80 MHz
```

```
cfgVHT.NumTransmitAntennas = 1;    % One transmit antenna
cfgVHT.NumSpaceTimeStreams = 1;    % One space-time stream
cfgVHT.STBC = false;               % No STBC so one spatial stream
cfgVHT.MCS = 8;                    % Modulation: 256-QAM
cfgVHT.APEPLength = 3000;          % A-MPDU length pre-EOF padding in bytes
```

Baseband Waveform Generation

Generate the VHT waveform for the specified bits and configuration by using the `wlanWaveformGenerator` function, specifying the desired oversampling factor, number of packets, and idle time between each packet.

```
numPackets = 20;  % Generate 20 packets
idleTime = 10e-6; % 10 microseconds idle time between packets
```

Create random bits for all packets, `data`, and pass as an argument to `wlanWaveformGenerator` along with the VHT packet configuration object `cfgVHT`. This configures the waveform generator to synthesize an 802.11ac VHT waveform. Additionally, configure the waveform generator by using name-value pairs to generate multiple packets with a specified idle time between each packet.

```
% Create random data; PSDULength is in bytes
savedState = rng(0); % Set random state
data = randi([0 1],cfgVHT.PSDULength*8*numPackets,1);

% Generate a multi-packet waveform
txWaveform = wlanWaveformGenerator(data,cfgVHT, ...
    'NumPackets',numPackets,'IdleTime',idleTime);

% Get the sampling rate of the waveform
fs = wlanSampleRate(cfgVHT);
disp(['Baseband sampling rate: ' num2str(fs/1e6) ' Msps']);
```

```
Baseband sampling rate: 80 Msps
```

Oversampling and Filtering

Use spectral filtering to reduce the out-of-band spectral emissions owing to the implicit rectangular pulse shaping in the OFDM modulation, and spectral regrowth caused by the high-power amplifier model. Upsample the waveform to model the effect of a high-power amplifier and to view the out-of-band spectral emissions. Oversampling requires an interpolation filter to remove spectral images caused by upsampling. This example oversamples the waveform with an interpolation filter, which acts as a spectral filter and allows the waveform to meet spectral mask requirements. Oversample and filter the waveform by using the `dsp.FIRInterpolator` System object (TM).

```
% Oversample the waveform
osf = 3;          % Oversampling factor
filterLen = 120;  % Filter length
beta = 0.5;       % Design parameter for Kaiser window

% Generate filter coefficients
coeffs = osf.*firnyquist(filterLen,osf,kaiser(filterLen+1,beta));
coeffs = coeffs(1:end-1); % Remove trailing zero
interpolationFilter = dsp.FIRInterpolator(osf,'Numerator',coeffs);
txWaveform = interpolationFilter(txWaveform);

% Plot the magnitude and phase response of the filter applied after
% oversampling
```
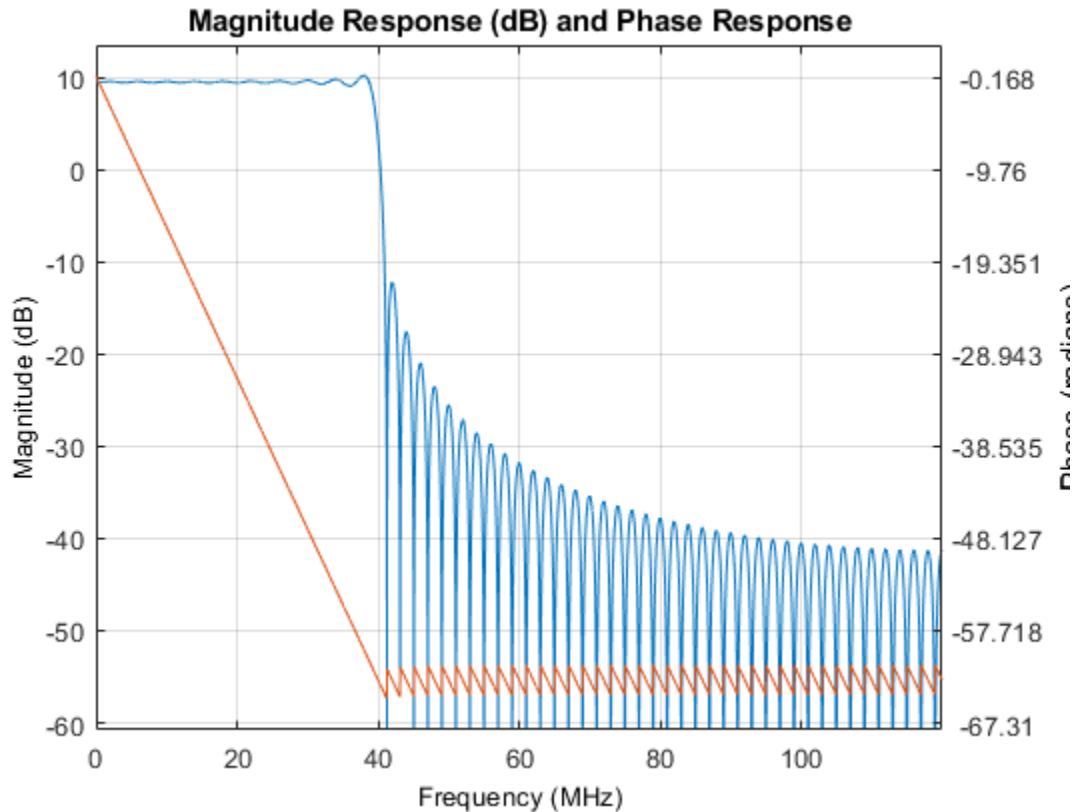
```
h = fvtool(interpolationFilter);
h.Analysis = 'freq';         % Plot magnitude and phase responses
h.FS = osf*fs;               % Set sampling rate
h.NormalizedFrequency = 'off'; % Plot responses against frequency
```


IQ Imbalance Modeling

IQ imbalance arises when a front-end component does not respect the power balance or the orthogonality between the I and Q branches. This example adds IQ gain and phase imbalance to the transmitted waveform based on the flag `modelIQImbalance`. At the receiver, estimate the IQ gain and phase imbalance and correct the waveform as per the compensation scheme specified in [5].

```
modelIQImbalance = true; % Set to true to add IQ gain and phase imbalance

if modelIQImbalance
    iqGaindB = 1; % IQ gain imbalance in dB, specify from the range [-1 1]
    iqPhaseDeg = 1; % IQ phase imbalance in degrees, specify from the range [-2 2]
    iqGainLin = db2mag(iqGaindB); % Convert gain from dB to linear value
    txWaveform = real(txWaveform) + 1i*imag(txWaveform)*iqGainLin*exp(1j*iqPhaseDeg*pi/180); % As
end
```

High-Power Amplifier Modeling

The high-power amplifier introduces nonlinear behavior in the form of inband distortion and spectral regrowth. This example simulates the power amplifiers by using the Rapp model in 802.11ac [2], which introduces AM/AM distortion.

Model the amplifier by using the `comm.MemorylessNonlinearity` object, and configure reduced distortion by specifying a backoff, `hpaBackoff`, such that the amplifier operates below its saturation point. You can increase the backoff to reduce EVM for higher MCS values.

```
pSaturation = 25; % Saturation power of a power amplifier in dBm
hpaBackoff = 13; % Power amplifier backoff in dB

% Create and configure a memoryless nonlinearity to model the amplifier
nonLinearity = comm.MemorylessNonlinearity;
nonLinearity.Method = 'Rapp model';
nonLinearity.Smoothness = 3; % p parameter
nonLinearity.LinearGain = -hpaBackoff;
nonLinearity.OutputSaturationLevel = db2mag(pSaturation-30);

% Apply the model to each transmit antenna
for i=1:cfgVHT.NumTransmitAntennas
    txWaveform(:,i) = nonLinearity(txWaveform(:,i));
end
```

Add thermal noise to the waveform, specifying a 6 dB receiver noise figure [3].

```
NF = 6;          % Noise figure (dB)
BW = fs*osf;     % Bandwidth (Hz)
k = 1.3806e-23;  % Boltzman constant (J/K)
T = 290;         % Ambient temperature (K)
noisePower = 10*log10(k*T*BW)+NF;

awgnChannel = comm.AWGNChannel('NoiseMethod','Variance', ...
    'Variance',10^(noisePower/10));
txWaveform = awgnChannel(txWaveform);
```

Modulation Accuracy (EVM), Spectral Flatness and IQ Imbalance Measurements

Resample the oversampled waveform down to baseband for physical layer processing and EVM, spectral flatness, and IQ imbalance measurements. As part of the resampling, apply a low-pass anti-aliasing filter before downsampling. The impact of the low-pass filter is visible in the spectral flatness measurement. Resample the waveform to baseband using the `dsp.FIRDecimator` System object (TM) with the same coefficients, `coeffs`, used for oversampling in the 'Oversampling and Filtering' section.

```
% Resample the waveform to baseband
decimationFilter = dsp.FIRDecimator(osf,'Numerator',coeffs);
rxWaveform = decimationFilter(txWaveform);
```

This section detects, synchronizes, and extracts each packet in `rxWaveform`, then measures the EVM, spectral flatness, and IQ imbalance. For each packet, the example performs these steps:

- Detect the start of the packet
- Extract the non-HT fields
- Estimate and correct coarse carrier frequency offset (CFO)
- Perform fine symbol timing estimate by using the frequency-corrected non-HT fields
- Extract the packet from the waveform by using the fine symbol timing offset
- Correct the extracted packet with the coarse CFO estimate
- Extract the L-LTF, then estimate the fine CFO and correct for the whole packet

- Extract the VHT-LTF and perform channel estimation for each of the transmit streams
- Measure the IQ imbalance from the channel estimate and perform correction on the channel estimate
- Measure the spectral flatness by using the channel estimate
- Extract and OFDM demodulate the VHT data field
- Perform noise estimation by using the demodulated data field pilots and single-stream channel estimate at pilot subcarriers
- Phase-correct and equalize the VHT data field by using the channel and noise estimates
- Correct the equalized data subcarriers with the IQ imbalance estimate
- For each data-carrying subcarrier in each spatial stream, find the closest constellation point and measure the EVM

The diagram shows the processing chain:
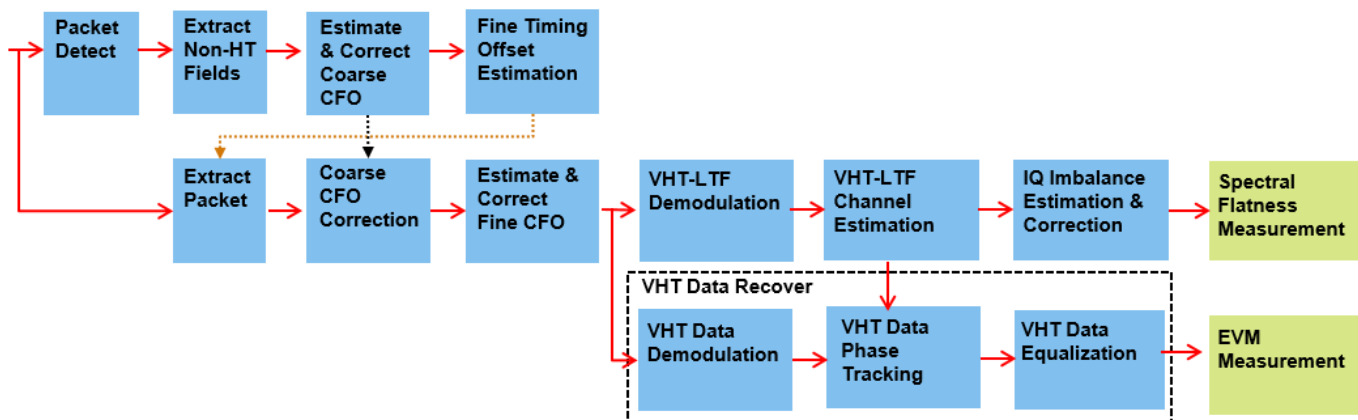
The VHT-LTF symbols include pilot symbols to allow for phase tracking, but this example does not perform phase tracking.

Test the spectral flatness for each packet by measuring the deviation in the magnitude of individual subcarriers in the channel estimate against the average [1]. Plot these deviations for each packet using the helper function vhtTxSpectralFlatnessMeasurement. Plot the average EVM per data-carrying subcarrier and the equalized symbols for each packet.

Demodulate, equalize, and decode the VHT Data symbols by using the `wlanVHTDataRecover` function. Parameterize this function to perform pilot phase tracking and zero-forcing equalization as required by the standard. This example measures the modulation accuracy from the equalized symbols.

This example makes two different EVM measurements using two instances of `comm.EVM`.

- RMS EVM per packet, which comprises averaging the EVM over subcarriers, OFDM symbols, and spatial streams.
- RMS EVM per subcarrier per spatial stream for a packet. Because this configuration maps spatial streams directly to antennas, this measurement can help detect frequency-dependent impairments, which may affect individual RF chains differently. This measurement averages the EVM over OFDM symbols only.

```
% Setup EVM measurements
[EVMPerPkt,EVMPerSC] = vhtEVMSetup(cfgVHT);
```

This code configures objects and variables for processing.

```
% Get indices for accessing each field within the time-domain packet
ind = wlanFieldIndices(cfgVHT);

rxWaveformLength = size(rxWaveform,1);
pktLength = double(ind.VHTData(2));

% Define the minimum length of data we can detect; length of the L-STF in
% samples
minPktLen = double(ind.LSTF(2)-ind.LSTF(1))+1;

% Setup the measurement plots
[hSF,hCon,hEVM] = vhtTxSetupPlots(cfgVHT);

rmsEVM = zeros(numPackets,1);
pktOffsetStore = zeros(numPackets,1);

rng(savedState); % Restore random state
```

Detect and process packets within the received waveform, rxWaveform by using a while loop, which performs these steps.

- Detect a packet by indexing into rxWaveform with the sample offset, searchOffset
- Detect and process the first packet within rxWaveform
- Detect and process the next packet by incrementing the sample index offset, searchOffset
- Repeat until no further packets are detected

```
pktNum = 0;
searchOffset = 0; % Start at first sample (no offset)
while (searchOffset+minPktLen)<=rxWaveformLength
    % Packet detect
    pktOffset = wlanPacketDetect(rxWaveform,cfgVHT.ChannelBandwidth);
    % Packet offset from start of waveform
    pktOffset = searchOffset+pktOffset;
    % If no packet detected or offset outwith bounds of waveform then stop
    if isempty(pktOffset) || (pktOffset<0) || ...
            ((pktOffset+ind.LSIG(2))>rxWaveformLength)
        break;
    end

    % Extract non-HT fields and perform coarse frequency offset correction
    % to allow for reliable symbol timing
    nonht = rxWaveform(pktOffset+(ind.LSTF(1):ind.LSIG(2)),:);
    coarsefreqOff = wlanCoarseCFOEstimate(nonht,cfgVHT.ChannelBandwidth);
    nonht = helperFrequencyOffset(nonht,fs,-coarsefreqOff);

    % Determine offset between the expected start of L-LTF and actual start
    % of L-LTF
    lltfOffset = wlanSymbolTimingEstimate(nonht,cfgVHT.ChannelBandwidth);
    % Determine packet offset
    pktOffset = pktOffset+lltfOffset;
    % If offset is without bounds of waveform skip samples and continue
    % searching within remainder of the waveform
```

```
if (pktOffset<0) || ((pktOffset+pktLength)>rxWaveformLength)
    searchOffset = pktOffset+double(ind.LSTF(2))+1;
    continue;
end

% Timing synchronization complete; extract the detected packet
rxPacket = rxWaveform(pktOffset+(1:pktLength),:);
pktNum = pktNum+1;
disp(['  Packet ' num2str(pktNum) ' at index: ' num2str(pktOffset+1)]);

% Apply coarse frequency correction to the extracted packet
rxPacket = helperFrequencyOffset(rxPacket,fs,-coarsefreqOff);

% Perform fine frequency offset correction on the extracted packet
lltf = rxPacket(ind.LLTF(1):ind.LLTF(2),:); % Extract L-LTF
fineFreqOff = wlanFineCFOEstimate(lltf,cfgVHT.ChannelBandwidth);
rxPacket = helperFrequencyOffset(rxPacket,fs,-fineFreqOff);

% Extract VHT-LTF samples, demodulate and perform channel estimation
vhtltf = rxPacket(ind.VHTLTF(1):ind.VHTLTF(2),:);
vhtltfDemod = wlanVHTLTFDemodulate(vhtltf,cfgVHT);

% Get single stream channel estimate
chanEstSSPilots = vhtSingleStreamChannelEstimate(vhtltfDemod,cfgVHT);

% Channel estimate
chanEst = wlanVHTLTFChannelEstimate(vhtltfDemod,cfgVHT);

% Perform IQ gain and phase imbalance estimation
[gainEst,phaseEst,alphaEst,betaEst,gamma,dataRot] = ...
                        helperIQImbalanceEstimate(chanEst,cfgVHT);
fprintf('    Measured IQ gain & phase imbalance: %2.2f dB, %2.2f deg\n',gainEst,phaseEst);

% Perform IQ gain and phase imbalance correction on channel
% estimates
chanEst = chanEst./(alphaEst + betaEst.*gamma); % As specified in Equation-29 of [5]

% Spectral flatness measurement
vhtTxSpectralFlatnessMeasurement(chanEst,cfgVHT,pktNum,hSF);

% Extract VHT Data samples from the waveform
vhtdata = rxPacket(ind.VHTData(1):ind.VHTData(2),:);

% Estimate the noise power in VHT data field
noiseVarVHT = vhtNoiseEstimate(vhtdata,chanEstSSPilots,cfgVHT);

% Extract VHT Data samples and perform OFDM demodulation, equalization
% and phase tracking
[~,~,eqSym] = wlanVHTDataRecover(vhtdata,chanEst,noiseVarVHT,cfgVHT,...
    'EqualizationMethod','ZF','PilotPhaseTracking','PreEQ'); % Use zero forcing algorithm fo

% Perform IQ gain and phase imbalance correction on VHT data
eqSym = eqSym.*dataRot; % Carrier rotation on data subcarriers
eqSym = ((conj(alphaEst)*eqSym)-(betaEst*conj(eqSym(end:-1:1,:,:))))/((abs(alphaEst)^2)-(abs

% Compute RMS EVM over all spatial streams for packet
rmsEVM(pktNum) = EVMPerPkt(eqSym);
fprintf('    RMS EVM: %2.2f%%, %2.2fdB\n',rmsEVM(pktNum),20*log10(rmsEVM(pktNum)/100));
```

8-51

```
        % Compute RMS EVM per subcarrier and spatial stream for the packet
        evmPerSC = EVMPerSC(eqSym); % Nst-by-1-by-Nss

        % Plot RMS EVM per subcarrier and equalized constellation
        vhtTxEVMConstellationPlots(eqSym,evmPerSC,cfgVHT,pktNum,hCon,hEVM);

        % Store the offset of each packet within the waveform
        pktOffsetStore(pktNum) = pktOffset;

        % Increment waveform offset and search remaining waveform for a packet
        searchOffset = pktOffset+pktLength+minPktLen;
    end

    if pktNum>0
        fprintf('Average EVM for %d packets: %2.2f%%, %2.2fdB\n', ...
            pktNum,mean(rmsEVM(1:pktNum)),20*log10(mean(rmsEVM(1:pktNum))/100));
    else
        disp('No complete packet detected');
    end

      Packet 1 at index: 41
        Measured IQ gain & phase imbalance: 0.98 dB, 0.96 deg
        Spectral flatness passed
        RMS EVM: 3.38%, -29.43dB
      Packet 2 at index: 9801
        Measured IQ gain & phase imbalance: 0.98 dB, 0.96 deg
        Spectral flatness passed
        RMS EVM: 3.14%, -30.05dB
      Packet 3 at index: 19561
        Measured IQ gain & phase imbalance: 0.98 dB, 0.96 deg
        Spectral flatness passed
        RMS EVM: 2.96%, -30.56dB
      Packet 4 at index: 29321
        Measured IQ gain & phase imbalance: 0.98 dB, 0.96 deg
        Spectral flatness passed
        RMS EVM: 3.09%, -30.20dB
      Packet 5 at index: 39081
        Measured IQ gain & phase imbalance: 0.98 dB, 0.96 deg
        Spectral flatness passed
        RMS EVM: 3.03%, -30.36dB
      Packet 6 at index: 48841
        Measured IQ gain & phase imbalance: 0.98 dB, 0.96 deg
        Spectral flatness passed
        RMS EVM: 2.73%, -31.27dB
      Packet 7 at index: 58601
        Measured IQ gain & phase imbalance: 0.98 dB, 0.96 deg
        Spectral flatness passed
        RMS EVM: 3.00%, -30.46dB
      Packet 8 at index: 68361
        Measured IQ gain & phase imbalance: 0.98 dB, 0.96 deg
        Spectral flatness passed
        RMS EVM: 2.89%, -30.79dB
      Packet 9 at index: 78121
        Measured IQ gain & phase imbalance: 0.98 dB, 0.96 deg
        Spectral flatness passed
        RMS EVM: 3.02%, -30.39dB
      Packet 10 at index: 87881
```

```
      Measured IQ gain & phase imbalance: 0.98 dB, 0.96 deg
      Spectral flatness passed
      RMS EVM: 2.79%, -31.09dB
    Packet 11 at index: 97641
      Measured IQ gain & phase imbalance: 0.98 dB, 0.96 deg
      Spectral flatness passed
      RMS EVM: 2.98%, -30.52dB
    Packet 12 at index: 107401
      Measured IQ gain & phase imbalance: 0.98 dB, 0.96 deg
      Spectral flatness passed
      RMS EVM: 3.00%, -30.47dB
    Packet 13 at index: 117161
      Measured IQ gain & phase imbalance: 0.98 dB, 0.96 deg
      Spectral flatness passed
      RMS EVM: 2.88%, -30.80dB
    Packet 14 at index: 126921
      Measured IQ gain & phase imbalance: 0.98 dB, 0.96 deg
      Spectral flatness passed
      RMS EVM: 2.96%, -30.59dB
    Packet 15 at index: 136681
      Measured IQ gain & phase imbalance: 0.98 dB, 0.96 deg
      Spectral flatness passed
      RMS EVM: 3.17%, -29.98dB
    Packet 16 at index: 146441
      Measured IQ gain & phase imbalance: 0.98 dB, 0.96 deg
      Spectral flatness passed
      RMS EVM: 2.69%, -31.42dB
    Packet 17 at index: 156201
      Measured IQ gain & phase imbalance: 0.98 dB, 0.96 deg
      Spectral flatness passed
      RMS EVM: 3.03%, -30.36dB
    Packet 18 at index: 165961
      Measured IQ gain & phase imbalance: 0.98 dB, 0.96 deg
      Spectral flatness passed
      RMS EVM: 3.26%, -29.75dB
    Packet 19 at index: 175721
      Measured IQ gain & phase imbalance: 0.98 dB, 0.96 deg
      Spectral flatness passed
      RMS EVM: 3.29%, -29.65dB
    Packet 20 at index: 185481
      Measured IQ gain & phase imbalance: 0.98 dB, 0.96 deg
      Spectral flatness passed
      RMS EVM: 3.27%, -29.72dB
  Average EVM for 20 packets: 3.03%, -30.38dB
```
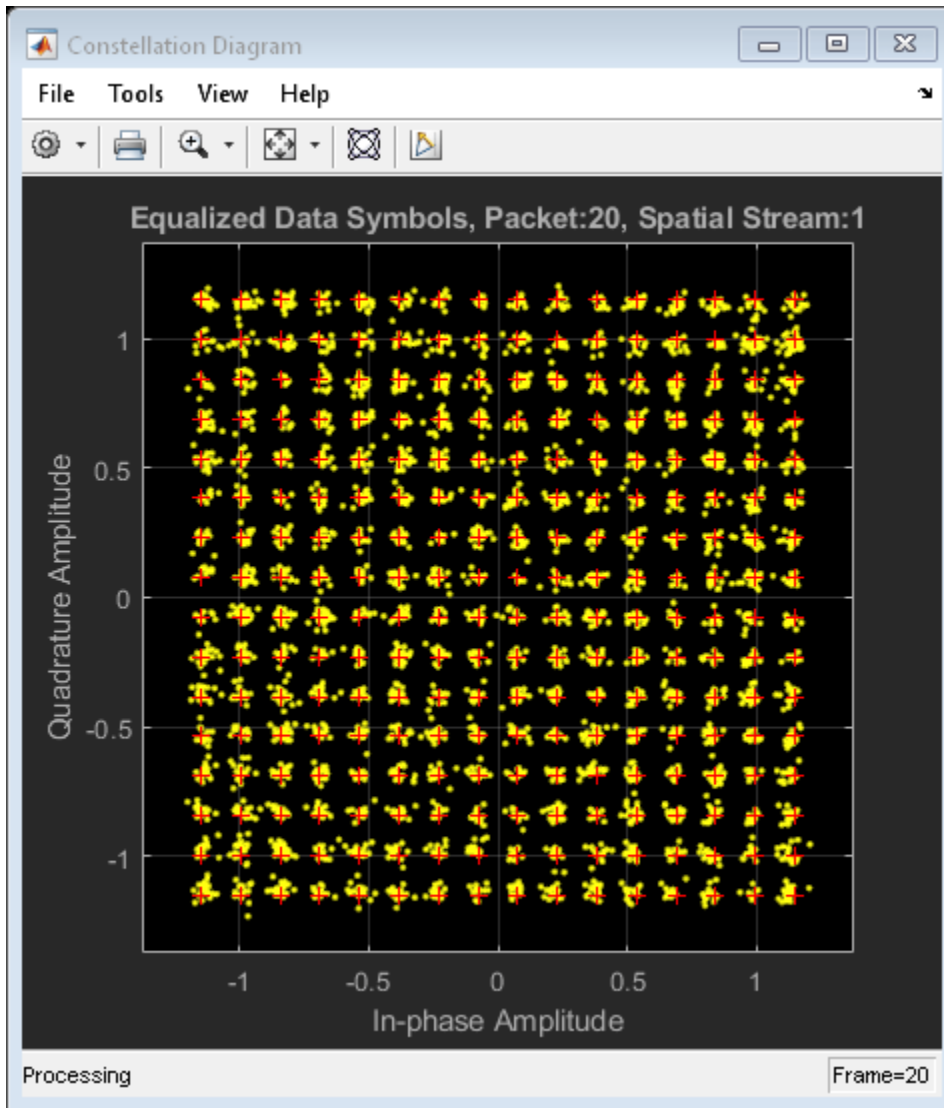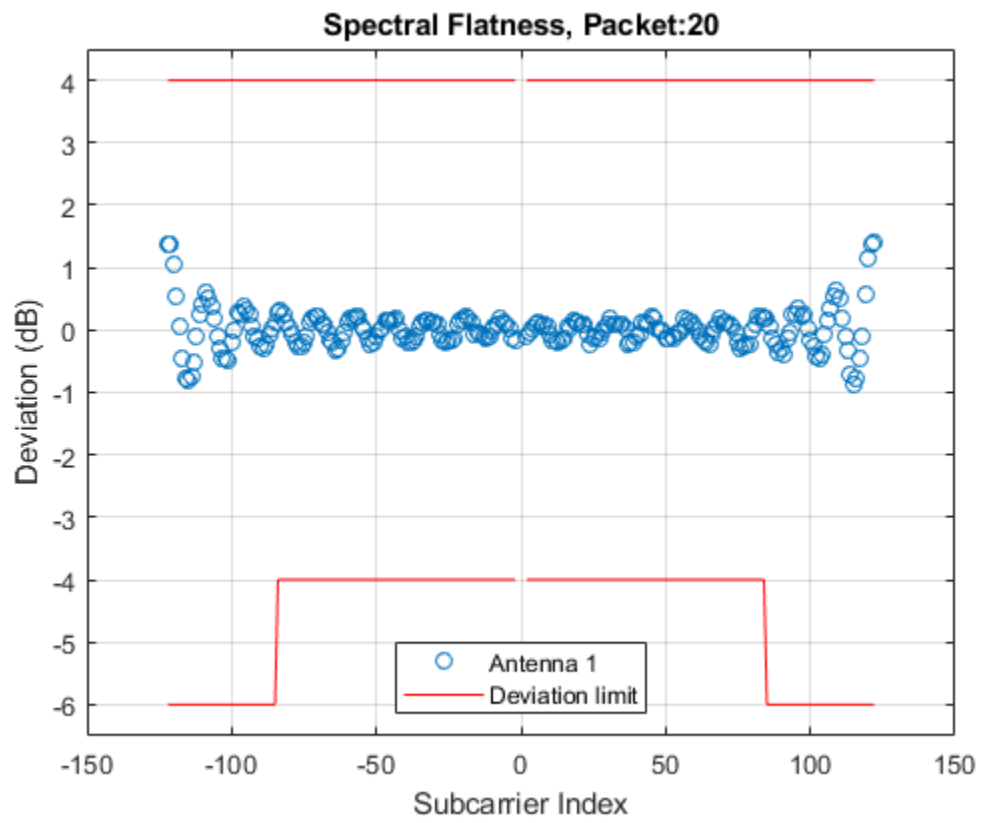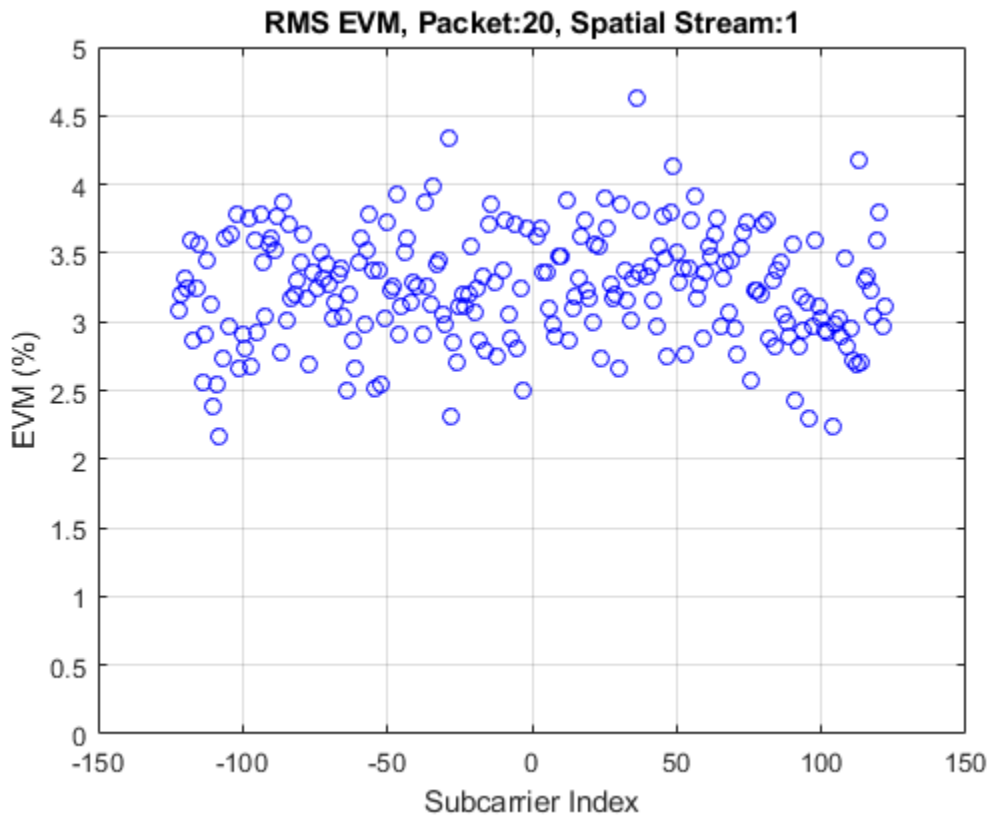
Spectral Flatness, Packet:20

RMS EVM, Packet:20, Spatial Stream:1

Transmit Spectrum Emission Mask Measurement

This section measures the spectral mask of the filtered and impaired waveform after high-power amplifier modeling.

The transmitter spectral mask test [4] uses a time-gated spectral measurement of the VHT Data field. The example extracts the VHT Data field of each packet from the oversampled waveform, `txWaveform`, by using the start indices of each packet within the baseband waveform. Any delay introduced in the baseband processing chain used to determine the packet indices must be accounted for when gating the VHT data field within `txWaveform`. Concatenate the extracted VHT Data fields in preparation for measurement.

```
startIdx = osf*(ind.VHTData(1)-1)+1;  % Upsampled start of VHT Data
endIdx = osf*ind.VHTData(2);          % Upsampled end of VHT Data
delay = grpdelay(decimationFilter,1); % Group delay of downsampling filter
idx = zeros(endIdx-startIdx+1,pktNum);
for i = 1:pktNum
    % Start of packet in txWaveform
    pktOffset = osf*pktOffsetStore(i)-delay;
    % Indices of VHT Data in txWaveform
    idx(:,i) = (pktOffset+(startIdx:endIdx));
end
gatedVHTData = txWaveform(idx(:),:);
```

The 802.11ac standard specifies the spectral mask relative to the peak power spectral density. The helper function helperSpectralMaskTest generates a plot which overlays the required mask with the measured PSD.
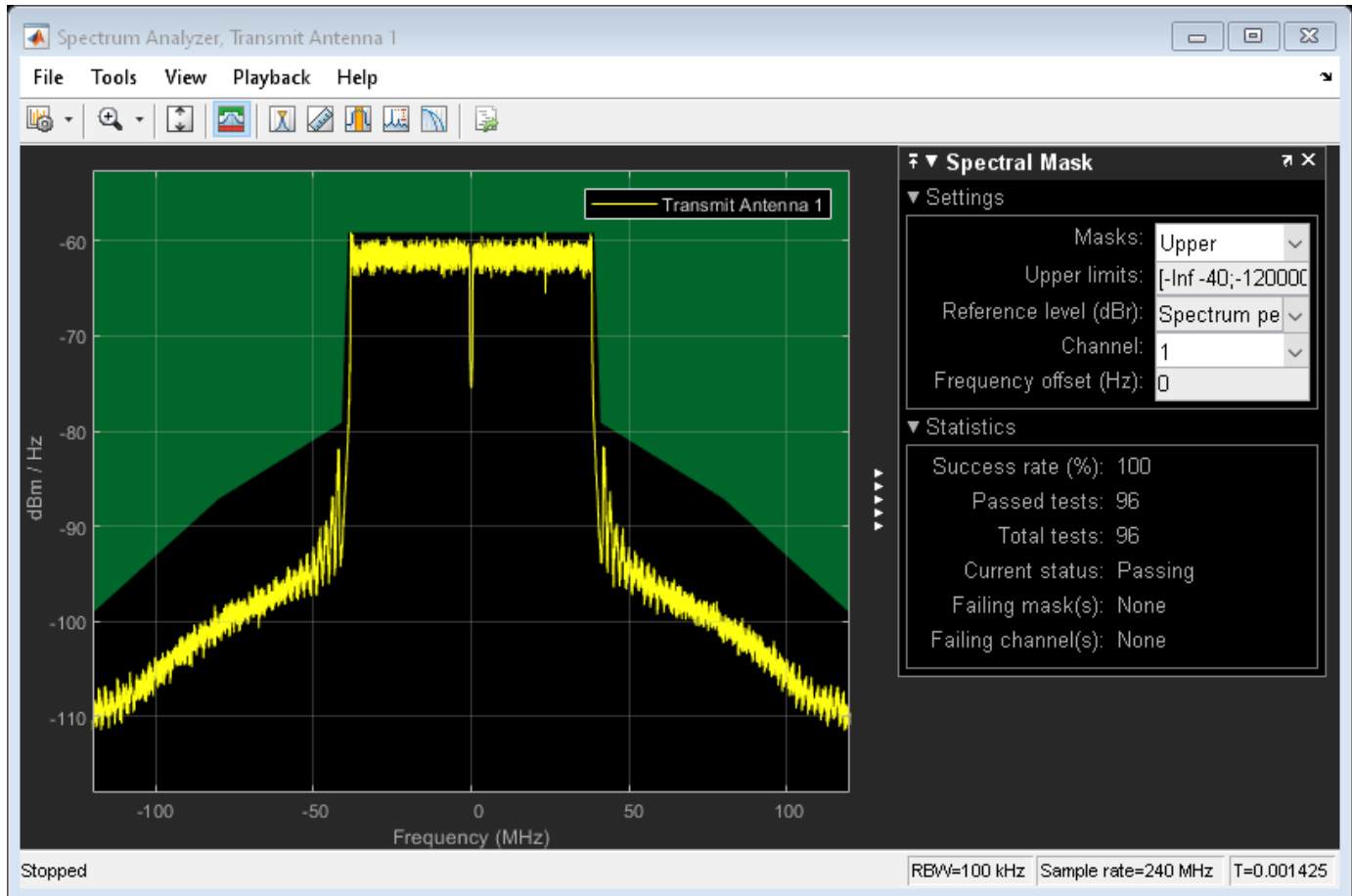
```
if pktNum>0
    helperSpectralMaskTest(gatedVHTData,fs,osf);
end
```

Spectrum mask passed

Conclusion and Further Exploration

This example plots four results: spectral flatness, RMS EVM per subcarrier, equalized constellation, and spectral mask.

The high-power amplifier model introduces significant inband distortion and spectral regrowth which is visible in the EVM results, noisy constellation and out-of-band emissions in the spectral mask plot. Try increasing the high-power amplifier backoff and note the improved EVM, constellation and lower out-of-band emissions.

Try using different values for `iqGaindB` and `iqPhaseDeg` and note the impact on EVM and constellation.

The spectral filtering and downsampling (to bring the waveform to baseband for processing) stages include filtering. These filter responses affect the spectral flatness measurement. The ripple in the spectral flatness measurement is mainly due to downsampling to baseband. Try using different filters or filter lengths and note the impact on the spectral flatness.

Selected Bibliography

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

2 Loc and Cheong. IEEE P802.11 Wireless LANs. TGac Functional Requirements and Evaluation Methodology Rev. 16. 2011-01-19.

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

4 Archambault, Jerry, and Shravan Surineni. "IEEE 802.11 spectral measurements using vector signal analyzers." RF Design 27.6 (2004): 38-49.

5 M. Janaswamy, N. K. Chavali and S. Batabyal, "Measurement of transmitter IQ parameters in HT and VHT wireless LAN systems," 2016 International Conference on Signal Processing and Communications (SPCOM), Bangalore.

# 802.11ad Transmitter Spectral Emission Mask Testing

This example shows how to perform pulse shaping and spectrum emission mask testing on an IEEE® 802.11ad™ transmitted waveform.

Introduction

IEEE 802.11ad [1] standard, commonly referred to as directional multi-gigabit (DMG), provides data throughput up to 7 Gbps using the 60 GHz industrial, scientific, and medical (ISM) frequency band. The DMG standard supports three PHY types:

- A Control PHY using MCS 0

- A Single Carrier (SC) PHY using MCS 1 to MCS 12 and low power SC PHY using MCS 25 to MCS 32

- An OFDM PHY using MCS 13 to MCS 24.

DMG defines four 2.16 GHz wide operating channels, typically in the 57-66 GHz band. The spectral mask test, as demonstrated in this example, ensures a transmission in one channel does not cause substantial interference in adjacent channels. The DMG channelization is shown in the figure below.
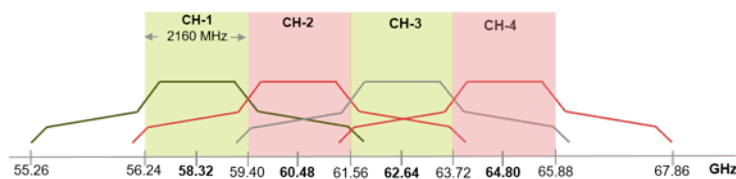
The SC DMG PHY uses single carrier modulation for low cost, short range applications. This example shows how pulse shaping and spectrum mask measurements can be performed on an SC DMG modulated waveform. The waveform is generated using WLAN Toolbox™, but a waveform captured with a spectrum analyzer could also be used. The transmitter spectrum mask and required spectral flatness for DMG configuration is specified in IEEE 802.11ad [1], Section 20.3.2.

This example generates five DMG SC packets, each separated by a one microsecond gap. Random data is used in each packet and pi/2-16QAM modulation is used. To meet the spectral mask requirements, the baseband waveform is upsampled and filtered to reduce the out-of-band emissions. A high power amplifier (HPA) model is used to introduce inband distortion and spectral regrowth. The spectral emission mask measurement is performed on the upsampled waveform after the HPA modeling. The test schematic is illustrated in the following diagram:
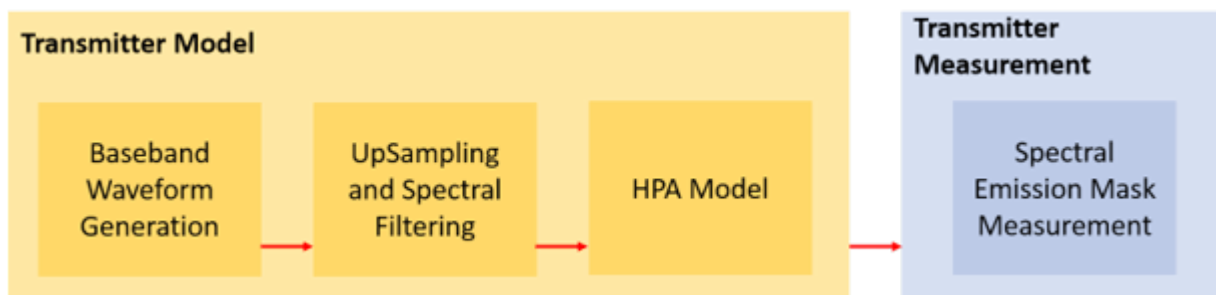
DMG, Single Carrier Packet Configuration

In this example, an IEEE 802.11ad waveform consisting of multiple DMG SC packets is generated. The DMG SC waveform properties are specified in a `wlanDMGConfig` configuration object. The object is configured for an MCS index of 12, with no `TrainingLength` fields appended to the packets. Per the test requirement (specified in IEEE 802.11ad Section 21.3.2), the `PSDULength` is set to 20000 for the packet to ensure that the transmit spectral mask is measured on a DMG packet longer than 10 microseconds.

```
cfgDMG = wlanDMGConfig;     % DMG packet configuration
cfgDMG.MCS = 12;            % SC PHY with pi/2-16QAM modulation
cfgDMG.PSDULength = 20000;  % Length in Bytes
```

Baseband Waveform Generation

The waveform generator can be configured to generate one or more packets with idle time between each packet. In this example, `wlanWaveformGenerator` is configured to generate five packets filled with random payload data. Each packet is separated by a one microsecond idle period in between and a random scrambler seed is used to generate each packet.

```
% Set random stream for repeatability of results
s = rng(98765);

% Generate a multi-packet waveform
idleTime = 1e-6; % One microsecond idle time between packets
numPackets = 5;  % Generate five packets

% Create random bits for all payload data; PSDULength is in bytes
psdu = randi([0 1],cfgDMG.PSDULength*8*numPackets,1);

% Override the ScramblerInitialization property of the DMG configuration
% object by specifying the scrambler initialization
genWaveform = wlanWaveformGenerator(psdu,cfgDMG,...
    'IdleTime',idleTime, ...
    'NumPackets',numPackets, ...
    'ScramblerInitialization',randi([1 127],numPackets,1));

% Get the sampling rate of the waveform
fs = wlanSampleRate(cfgDMG);
disp(['Baseband sampling rate: ' num2str(fs/1e6) ' Msps']);
```

```
Baseband sampling rate: 1760 Msps
```

Oversampling and Filtering

Spectral filtering is used to reduce the out-of-band spectral emissions due to the spread spectrum characteristics of the transmitted waveform and spectral regrowth caused by the HPA in an RF chain. The waveform must be oversampled to model the effect of the HPA on the waveform and view the out-of-band spectral emissions. In this example, the waveform is oversampled and filtered through a raised cosine filter using `comm.RaisedCosineTransmitFilter`. To meet the spectral mask requirements, the raised cosine filter is truncated to the duration of eight symbols and the roll-off factor is set to 0.5.

```
% Define the pulse shaping filter characteristics
Nsym = 8;   % Filter span in symbol durations
beta = 0.5; % Roll-off factor
osps = 4;   % Output samples per symbol
```

```
% Create raised cosine transmit filter system object
rcosFlt = comm.RaisedCosineTransmitFilter(...
    'Shape','Normal', ...
    'RolloffFactor',beta, ...
    'FilterSpanInSymbols',Nsym, ...
    'OutputSamplesPerSymbol',osps);

% Filter transmit signal for pulse shaping
filterWaveform = rcosFlt([genWaveform; zeros(Nsym/2,1)]);

% Plot the magnitude and phase response of the pulse shaping filter
h = fvtool(rcosFlt,'Analysis','freq');
h.FS = osps*fs;                 % Set sampling rate
h.NormalizedFrequency = 'off'; % Plot responses against frequency
```
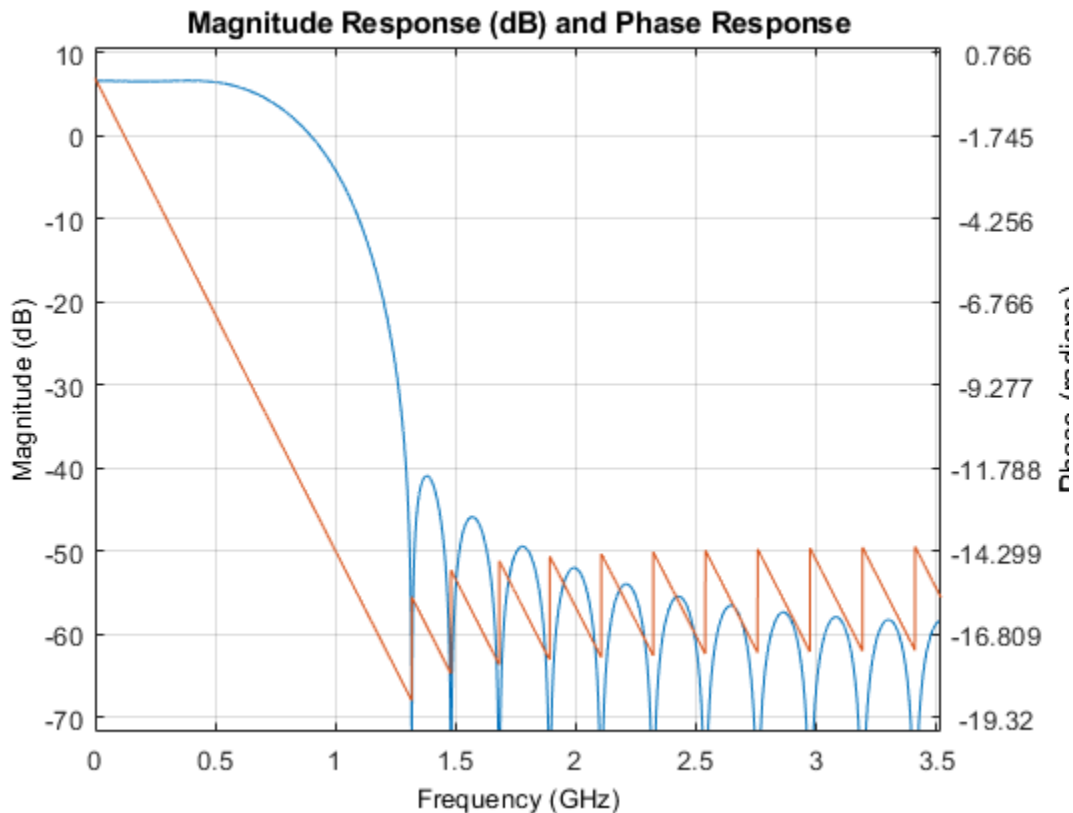

High Power Amplifier Modeling

Within an RF chain, the HPA is a necessary component but it introduces nonlinear behavior in the form of inband distortion and spectral regrowth. The Rapp model, described in [2], can be used to model an 802.11ad power amplifier. The Rapp model causes AM/AM distortion and is modeled with `comm.MemorylessNonlinearity`. The HPA is backed-off to operate below the saturation point to reduce distortion.

```
hpaBackoff = 0.5; % Power Amplifier backoff in dB

% Create and configure a memoryless nonlinearity to model HPA
```

```
nonLinearity = comm.MemorylessNonlinearity;
nonLinearity.Method = 'Rapp model';
nonLinearity.Smoothness = 0.81;                          % Smoothness factor
nonLinearity.LinearGain = 10*log10(4.65) - hpaBackoff;  % Small signal gain
nonLinearity.OutputSaturationLevel = 0.58;              % Saturation level

% Apply the model
txWaveform = nonLinearity(filterWaveform);
```

Transmit Spectrum Emission Mask Measurement

IEEE 802.11ad [1], Section 20.3.2, specifies the transmit spectral mask that all DMG waveforms must adhere to and describes the packet characteristics. According to the test definition, packets should have no training fields appended and be greater than 10 microseconds in duration.

```
dBrLimits = [-30 -30 -22 -17 0 0 -17 -22 -30 -30];
fLimits = [-Inf -3.06 -2.7 -1.2 -0.94 0.94 1.2 2.7 3.06 Inf] * 1e3;
rbw = 1e6; % Resolution bandwidth in Hz
vbw = 300e3; % Video bandwidth in Hz
```

Use the helper function `helperSpectralMaskTest` to generate a plot that overlays the required spectral mask with the measured PSD. It checks the transmitted PSD levels to be within the specified mask levels and displays a pass/fail status after the test.

```
helperSpectralMaskTest(txWaveform,fs,osps,dBrLimits,fLimits,rbw,vbw);
```

```
% Restore default stream
rng(s);
```
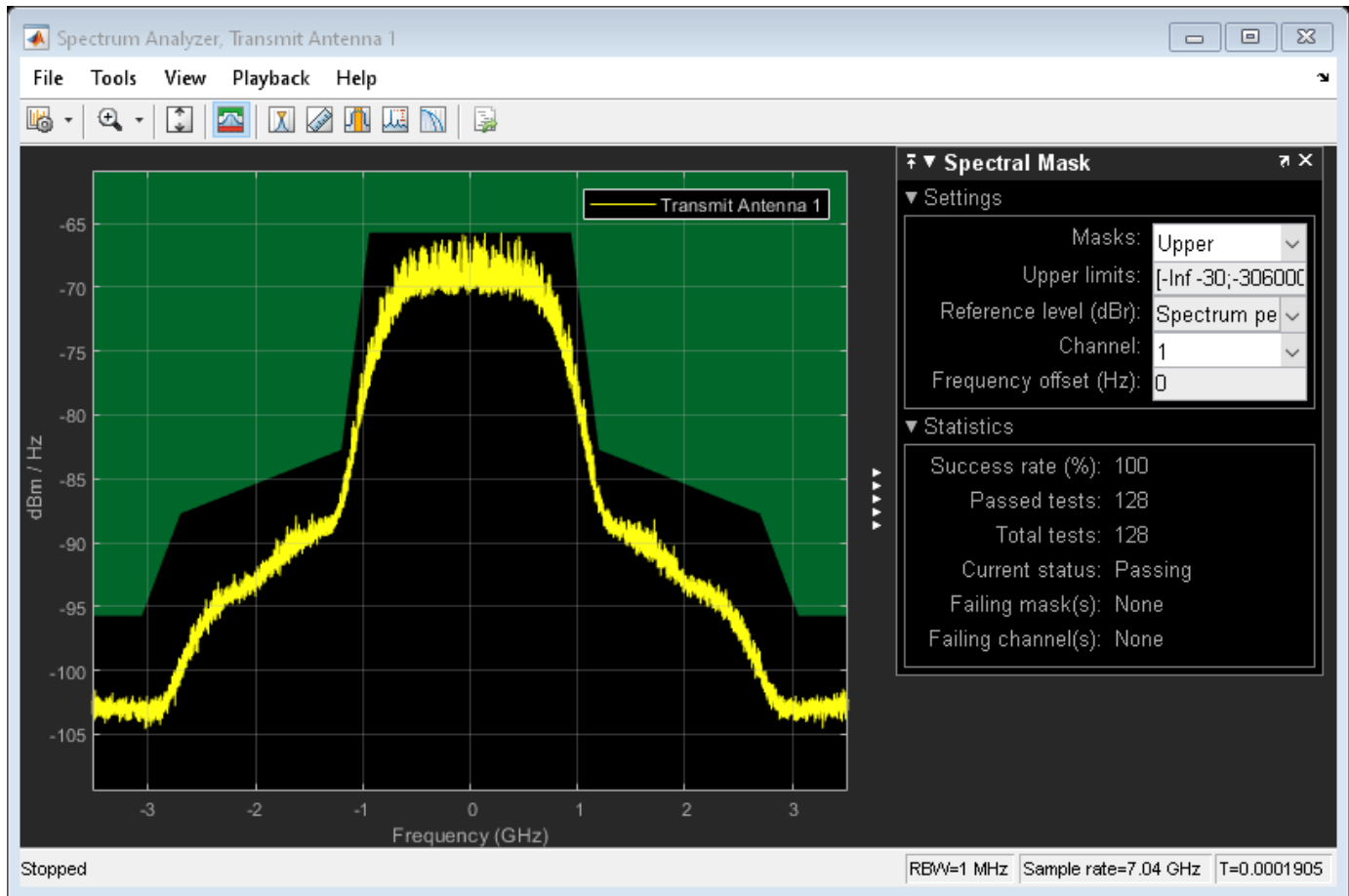
```
    Spectrum mask passed
```

Conclusion and Further Exploration

The transmit spectral mask for a DMG SC waveform in the 60 GHz band for a 2.16 GHz channel bandwidth is shown in this example. It also illustrates that the spectrum of the transmitted signal satisfies regulatory restrictions by falling within the spectral mask after pulse shaping. A similar result can be generated for DMG Control and OFDM PHYs.

The HPA model and the spectral filtering affect the out-of-band emissions in the spectral mask plot. For Single Carrier and Control PHY, you can try using different pulse shaping filter parameters and/or decrease or increase the smoothness factor.

For information on other transmitter measurements like modulation accuracy and spectral flatness, refer to the following examples:

- "802.11ac Transmitter Measurements" on page 8-45
- "802.11p Spectral Emission Mask Testing" on page 8-65

Appendix

This example uses the following helper functions:

- helperSpectralMaskTest.m

Selected Bibliography

1   IEEE Std 802.11™-2016: IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

2   Eldad Perahia, et. al. TGad Evaluation Methodology, IEEE 802.11-09/0296r16

802.11p Spectral Emission Mask Testing

This example shows how to perform spectrum emission mask tests for an IEEE® 802.11p™ transmitted waveform.

Introduction

IEEE 802.11p [2] is an approved amendment to the IEEE 802.11™ standard to enable support for wireless access in vehicular environments (WAVE). Using the half-clocked mode with a 10 MHz channel bandwidth, it operates at the 5.85-5.925 GHz bands for which additional spectral emission masks are defined [Annex D of 1].

This example shows how spectral mask measurements can be performed on a transmitted waveform. The waveform is generated with WLAN Toolbox™ for simplicity, but a waveform captured with a spectrum analyzer could be used as well.

A waveform consisting of three 10 MHz IEEE 802.11p packets separated by a 32 microsecond gap is generated. Random data is used for each packet and 16QAM modulation is used. The baseband waveform is upsampled and filtered to reduce the out of band emissions thereby meeting the spectral mask requirements. A high power amplifier (HPA) model is used, which introduces inband distortion and spectral regrowth. The spectral emission mask measurement is performed on the upsampled waveform after the high power amplifier modeling. The test schematic is illustrated in the following diagram:
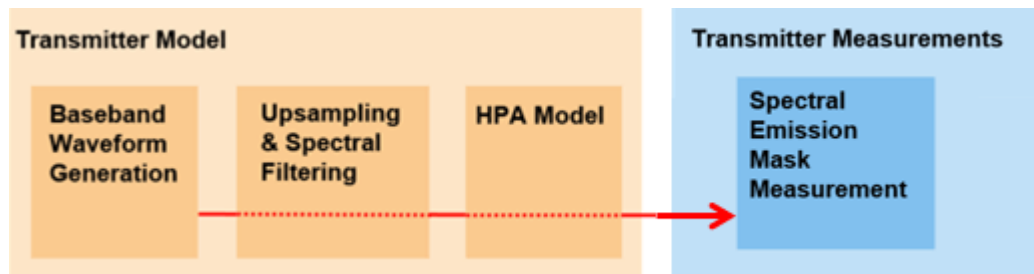
IEEE 802.11p non-HT Packet Configuration

In this example, an IEEE 802.11p waveform consisting of multiple non-HT format packets is generated. Format parameters of the non-HT waveform are described using a non-HT format configuration object. The object is created using the `wlanNonHTConfig` function. In this example, the object is configured for a 10 MHz bandwidth operation as used by IEEE 802.11p.

```
cfgNHT = wlanNonHTConfig;              % Create packet configuration
cfgNHT.ChannelBandwidth = 'CBW10';     % 10 MHz
cfgNHT.MCS = 4;                        % Modulation 16QAM, rate-1/2
cfgNHT.PSDULength = 1000;              % PSDU length in bytes
```

Baseband Waveform Generation

The waveform generator can be configured to generate one or more packets and add an idle time between each packet. In this example three packets with a 32 microsecond idle period will be created. Random bits for all packets `data` are created and passed as an argument to `wlanWaveformGenerator` along with the non-HT packet configuration object `cfgNHT` and additional waveform generation parameters. `cfgNHT` configures the waveform generator to create the IEEE 802.11p non-HT waveform.

```
% Set random stream for repeatability of results
s = rng(98765);

% Generate a multi-packet waveform
idleTime   = 32e-6;     % 32 microsecond idle time between packets
numPackets = 3;         % Generate 3 packets

% Create random data; PSDULength is in bytes
data = randi([0 1], cfgNHT.PSDULength*8*numPackets, 1);

genWaveform = wlanWaveformGenerator(data, cfgNHT, ...
              'NumPackets', numPackets,...
              'IdleTime', idleTime);

% Get the sampling rate of the waveform
fs = wlanSampleRate(cfgNHT);
disp(['Baseband sampling rate: ' num2str(fs/1e6) ' Msps']);

Baseband sampling rate: 10 Msps
```

Oversampling and Filtering

Spectral filtering is used to reduce the out of band spectral emissions due to the implicit rectangular pulse shaping in the OFDM modulation, and spectral regrowth caused by the high power amplifier in an RF chain. To model the effect of a high power amplifier on the waveform and view the out of band spectral emissions the waveform must be oversampled. In this example the waveform is oversampled with an interpolation filter which also acts as a spectral filter. This allows the waveform to meet spectral mask requirements. The waveform is oversampled and filtered using `dsp.FIRInterpolator`.

```
% Oversample the waveform
osf = 3;          % Oversampling factor
filterLen = 100;  % Filter length
r = 50;           % Design parameter for Chebyshev window (attenuation, dB)

% Generate filter coefficients and interpolate
coeffs = osf.*firnyquist(filterLen, osf, chebwin(filterLen+1, r));
coeffs = coeffs(1:end-1);   % Remove trailing zero
interpolationFilter = dsp.FIRInterpolator(osf, 'Numerator', coeffs);
filtWaveform = interpolationFilter([genWaveform; zeros(filterLen/2,1)]);

% Plot the magnitude and phase response of the filter applied after
% oversampling
h = fvtool(interpolationFilter);
h.Analysis = 'freq';            % Plot magnitude and phase responses
h.FS = osf*fs;                  % Set sampling rate
h.NormalizedFrequency = 'off';  % Plot responses against frequency
```
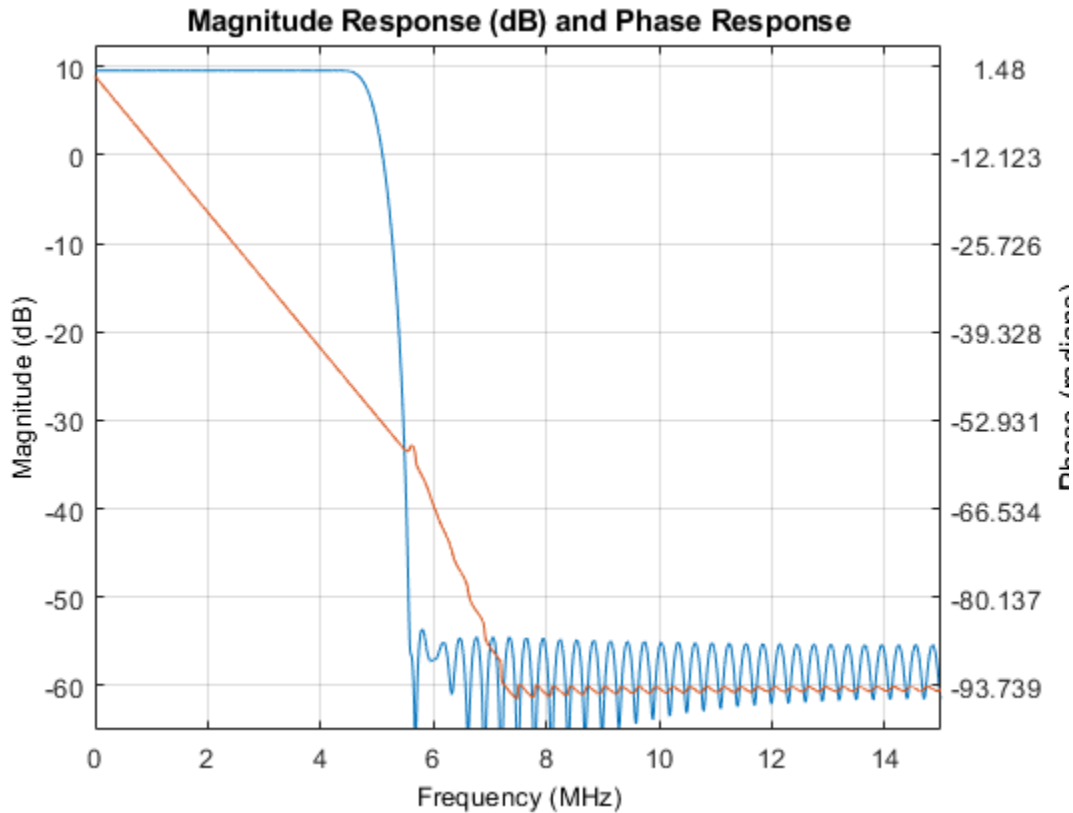
High Power Amplifier Modeling

The high power amplifier introduces nonlinear behavior in the form of inband distortion and spectral regrowth. This example simulates the power amplifiers by using the Rapp model in 802.11ac [2], which introduces AM/AM distortion.

Model the amplifier by using `comm.MemorylessNonlinearity` object, and configure reduced distortion by specifying a back-off, `hpaBackoff`, such that the amplifier operates below its saturation point. You can increase the backoff to reduce EVM for higher MCS values.

```
pSaturation = 25; % Saturation power of a power amplifier in dBm
hpaBackoff = 16; % dB

% Create and configure a memoryless nonlinearity to model the amplifier
nonLinearity = comm.MemorylessNonlinearity;
nonLinearity.Method = 'Rapp model';
nonLinearity.Smoothness = 3; % p parameter
nonLinearity.LinearGain = -hpaBackoff; % dB
nonLinearity.OutputSaturationLevel = db2mag(pSaturation-30);

% Apply the model to the transmit waveform
txWaveform = nonLinearity(filtWaveform);
```

Transmit Spectrum Emission Mask Measurement

Stations are classified according to the allowed maximum transmit powers (in mW). For the four different classes of stations, four different spectral emission masks are defined [Annex D of 1]. The spectral masks are defined relative to the peak power spectral density (PSD).

In this example the spectrum emission mask of the transmitted waveform after high power amplifier modeling is measured for a Class A station.

```
% IEEE Std 802.11-2012 Annex D.2.3, Table D-5: Class A STA
dBrLimits = [-40  -40 -28 -20  -10 0   0  -10 -20 -28 -40 -40];
fLimits   = [-Inf -15 -10 -5.5 -5 -4.5 4.5 5  5.5  10  15 Inf];
```

A time gated spectral measurement of the non-HT Data field is used for the transmitter spectrum emission mask test [3]. The non-HT Data field of each packet is extracted from the upsampled `txWaveform` using the start index of each packet. The extracted non-HT Data fields are concatenated in preparation for measurement.

```
% Indices for accessing each field within the time-domain packet
ind = wlanFieldIndices(cfgNHT);
startIdx = osf*(ind.NonHTData(1)-1)+1;   % Upsampled start of non-HT Data
endIdx = osf*ind.NonHTData(2);           % Upsampled end of non-HT Data
idleNSamps = osf*idleTime/(1/fs);        % Upsampled idle time samples
perPktLength = endIdx + idleNSamps;

idx = zeros(endIdx-startIdx+1, numPackets);
for i = 1:numPackets
    % Start of packet in txWaveform, accounting for the filter delay
    pktOffset = (i-1)*perPktLength+filterLen/2;
    % Indices of non-HT Data in txWaveform
    idx(:,i) = pktOffset+(startIdx:endIdx);
end
% Select the Data field for the individual packets
gatedNHTDataTx = txWaveform(idx(:),:);
```

The plot generated by the helper function `helperSpectralMaskTest` overlays the required spectral mask with the measured PSD. It checks the transmitted PSD levels to be within the specified mask levels and displays a pass/fail status after the test.

```
% Evaluate the PSD and check for compliance
helperSpectralMaskTest(gatedNHTDataTx, fs, osf, dBrLimits, fLimits);

% Restore default stream
rng(s);

    Spectrum mask passed
```
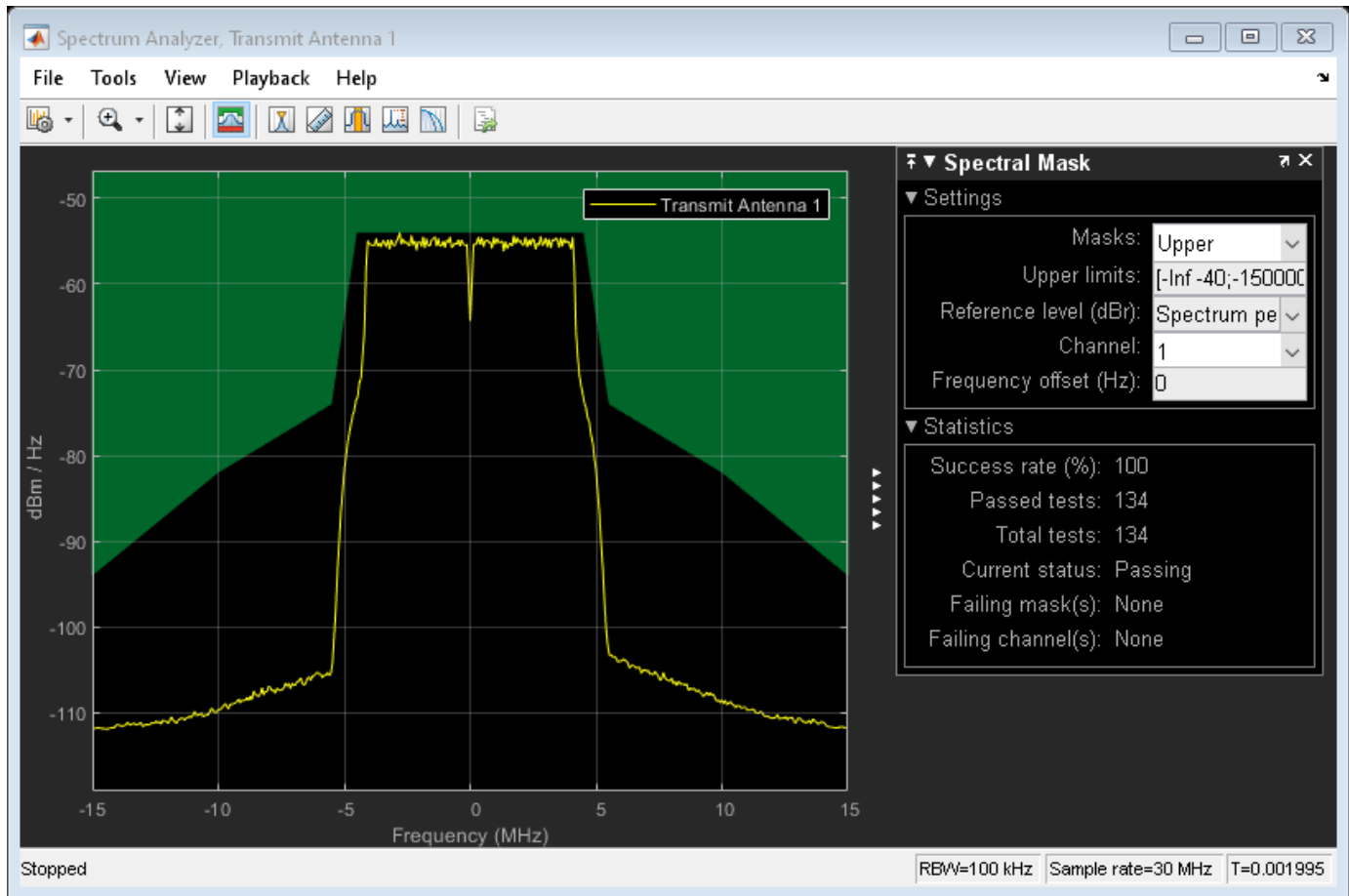
Conclusion and Further Exploration

The transmit spectral mask for Class A Stations at the 5.85-5.925 GHz bands for a 10 MHz channel spacing is shown in this example. It is also shown how the peak spectral density of the transmitted signal falls within the spectral mask to satisfy regulatory restrictions. A similar result can be generated for the 5 MHz channel spacing.

The high power amplifier model and the spectral filtering affect the out-of-band emissions in the spectral mask plot. For different station classes with higher relative dB values, try using different filters or filter lengths and/or increase the backoff for lower emissions.

For information on other transmitter measurements like modulation accuracy and spectral flatness, refer to the following example:

- "802.11ac Transmitter Measurements" on page 8-45

Selected Bibliography

1   IEEE Std 802.11-2012: IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE, New York, NY, USA, 1999-2013.

2   IEEE Std 802.11p-2010: 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 6: Wireless Access in Vehicular Environments, IEEE, New York, NY, USA, 2010.

3 Archambault, Jerry, and Shravan Surineni. "IEEE 802.11 spectral measurements using vector signal analyzers." RF Design 27.6 (2004): 38-49.

Code Generation and Deployment

What is C Code Generation from MATLAB?

You can use WLAN Toolbox together with MATLAB® Coder™ to:

- Create a MEX file to speed up your MATLAB application.
- Generate ANSI®/ISO® compliant C/C++ source code that implements your MATLAB functions and models.
- Generate a standalone executable that runs independently of MATLAB on your computer or another platform.

In general, the code you generate using the toolbox is portable ANSI C code. In order to use code generation, you need a MATLAB Coder license. For more information, see "Get Started with MATLAB Coder" (MATLAB Coder).

Using MATLAB Coder

Creating a MATLAB Coder MEX file can substantially accelerate your MATLAB code. It is also a convenient first step in a workflow that ultimately leads to completely standalone code. When you create a MEX file, it runs in the MATLAB environment. Its inputs and outputs are available for inspection just like any other MATLAB variable. You can then use MATLAB tools for visualization, verification, and analysis.

The simplest way to generate MEX files from your MATLAB code is by using the `codegen` function at the command line. For example, if you have an existing function, `myfunction.m`, you can type the commands at the command line to compile and run the MEX function. `codegen` adds a platform-specific extension to this name. In this case, the `"mex"` suffix is added.

```
codegen myfunction.m
myfunction_mex;
```

Within your code, you can run specific commands either as generated C code or by using the MATLAB engine. In cases where an isolated command does not yet have code generation support, you can use the `coder.extrinsic` command to embed the command in your code. This means that the generated code reenters the MATLAB environment when it needs to run that particular command. This is also useful if you want to embed commands that cannot generate code (such as plotting functions).

To generate standalone executables that run independently of the MATLAB environment, create a MATLAB Coder project inside the MATLAB Coder Integrated Development Environment (IDE). Alternatively, you can call the `codegen` command in the command line environment with appropriate configuration parameters. A standalone executable requires you to write your own `main.c` or `main.cpp` function. See "Generating Standalone C/C++ Executables from MATLAB Code" (MATLAB Coder) for more information.

C/C++ Compiler Setup

Before using `codegen` to compile your code, you must set up your C/C++ compiler. For 32-bit Windows platforms, MathWorks® supplies a default compiler with MATLAB. If your installation does not include a default compiler, you can supply your own compiler. For the current list of supported compilers, see Supported and Compatible Compilers on the MathWorks website. Install a compiler that is suitable for your platform, then read "Setting Up the C or C++ Compiler" (MATLAB Coder).

After installation, at the MATLAB command prompt, run `mex -setup`. You can then use the `codegen` function to compile your code.

Functions and System Objects That Support Code Generation

All WLAN Toolbox functions and System objects support code generation.

See Also

Functions
codegen | mex

More About

*   "Code Generation Workflow" (MATLAB Coder)
*   Generate C Code from MATLAB Code Video

Code Generation of WLAN Toolbox™ Features

This example shows how to generate MEX files and efficient C/C++ code for the WLAN Toolbox™ waveform generator function and verify its correct behaviour. Additionally, it shows how to work around limitations of code generation for WLAN Toolbox channel models. All WLAN Toolbox functions and System objects™ are supported for C/C++ code generation.

Introduction

With MATLAB Coder™, you can generate efficient, portable C source code, standalone executables, and MEX functions for deployment in desktop and embedded applications. You can speed up your MATLAB code using MEX functions or integrate generated source code and static or dynamic libraries into your C/C++ code projects. See "Setting Up the C or C++ Compiler" (MATLAB Coder) for more information about how to set up the C/C++ compiler. In order to use code generation, you need a MATLAB Coder license.

This example uses the `codegen` function to generate code for the WLAN Toolbox waveform generator and the TGax channel System object. First, prepare the required input arguments of the waveform generator and configure MATLAB Coder to generate MEX files. Next, compare the output of the generated MEX files and the original MATLAB code. Finally, the example shows how to work around limitations of code generation affecting WLAN Toolbox.

Input Arguments Definition

For code generation, you must specify the size and type of the input arguments to your entry-point function. In this case, you specify the input arguments of the `wlanWaveformGenerator` function and packet format configuration object. The mandatory arguments of the waveform generator are the data bits of the physical layer service data unit (PSDU) and the waveform format configuration. Additional inputs such as the idle time, scrambler initialization, and window transition time depend on the format configuration. Specify the additional inputs as name-value pairs. See `wlanWaveformGenerator` for more information.

You can use example values of the input arguments and the `codegen` function automatically derives their size, class, and complexity. However, the size of the input data vector depends on properties of the format configuration such as the channel bandwidth and coding. Use the `coder.typeof` function to define a variable-size input data vector that adapts to the format configuration properties as required.

Thus, you can run the generated code for multiple parameter sets of the format configuration object. For more information about specifying variable-size input arguments, see "Generate Code for Variable-Size Data" (MATLAB Coder). In this case, specify only the first dimension as variable-size.

```
variableDims = [1 0];
```

Variable-size vectors can be upper bounded or unbounded. To allow for full flexibility, you can select the upper bound to be the maximum aggregated MAC protocol data unit (A-MPDU) length, i.e., 6500631 bytes.

```
upperBound = 6500631*8; % Upper bound in bits
```

Next, use `coder.typeof` to define an input type as a column vector of doubles with a maximum size of `upperBound`-by-1, with the first dimension variable-size.

```
inputBits = coder.typeof(double(0),[upperBound 1],variableDims);
```

As the `inputBits` data type is double, you can only use double-precision values for the input bits of the generated MEX and C/C++ code.

Next, create a high-efficiency single-user (HE SU) format configuration object with default parameters and its corresponding configuration structure using the `coder.typeof` function. Use this structure to specify the type and size of any property of the HE SU format configuration object.

```
cfgHESU = wlanHESUConfig;
cfgHESUcg = coder.typeof(cfgHESU);
```

There are two possible forward-error-correction (FEC) coding types for the HE-Data: low-density parity-check, specified by setting the `ChannelCoding` property of `cfgHESU` to `'LDPC'`, and binary convolutional coding, specified as `'BCC'`. Since these are character arrays of different lengths, use the `coder.typeof` function to define a variable-length row vector of maximum size 1-by-4 characters.

```
cfgHESUcg.Properties.ChannelCoding = coder.typeof('LDPC',[1 4],[0 1]);
```

Use the same strategy for the channel bandwidth property. In this case, the longest character array corresponds to `'CBW160'`, so six characters are enough to cover all other possible cases, i.e., `'CBW20'`, `'CBW40'`, and `'CBW80'`.

```
cfgHESUcg.Properties.ChannelBandwidth = coder.typeof('CBW160',[1 6],[0 1]);
```

Specify optional arguments of the waveform generator such as the windowing transition time using name-value pairs. Use `coder.Constant` to define the name of the argument because it is a string literal that is not expected to change.

```
WindowTransitionTime_Name = coder.Constant('WindowTransitionTime');
```

Use an example value from which the `codegen` function derives its type and size. This technique is only suitable for fixed-size input arguments.

```
WindowTransitionTime_Value = 0;
```

Code Generation for the WLAN Toolbox Waveform Generator

Once the input arguments are specified for code generation, configure MATLAB Coder to generate MEX files and C/C++ code. A MEX file acts as an interface to the generated C/C++ code that can run in MATLAB. MEX file generation is usually the first step in the code generation workflow as it provides a convenient way for verifying the generated C/C++ code.

You can generate a MEX file, C/C++ code, a dynamic library, or a standalone executable by creating a MATLAB Coder configuration object and specifying the build type as `'MEX'`, `'LIB'`, `'DLL'`, or `'EXE'`, respectively.

```
BuildType = 'MEX';
cfgCoder = coder.config(BuildType);
```

Generate a report containing useful information about the code generation process.

```
cfgCoder.GenerateReport = true;
```

The name of the generated MEX file is the entry-point function name appended by the suffix 'mex', that is, `wlanWaveformGenerator_mex`. To specify a different name for your MEX file, use the option `-o output_file_name`. You can find the generated MEX files in your local folder and the generated C/C++ code in the folder codegen\mex\wlanWaveformGenerator.

Users of Microsoft Visual C++ product family may see the C4101 compiler warning indicating an unreferenced local variable.

```
inputArgs = {inputBits,cfgHESUcg,WindowTransitionTime_Name,WindowTransitionTime_Value}; %#ok<NASG
codegen wlanWaveformGenerator -args inputArgs -config cfgCoder -o wlanWaveformGenerator_HESU_mex
```

```
Code generation successful: To view the report, open('codegen\mex\wlanWaveformGenerator\html\repo
```

Next, verify that the generated MEX file behaves as expected when you use different configurations of the `ChannelBandwidth` and `ChannelCoding` by comparing its output to that of the `wlanWaveformGenerator`.

```
% Create a HE SU format configuration object specifying the channel
% bandwidth, coding, number of antennas and streams.
cfgHESU = wlanHESUConfig('ChannelBandwidth','CBW20','ChannelCoding','BCC', ...
                         'NumTransmitAntennas',2,'NumSpaceTimeStreams',2);

% Set the value of the window transition time
WindowTransitionTime = 1e-09;

% Create a PSDU of size defined by the getPSDULength function
inputBits = randi([0 1],getPSDULength(cfgHESU)*8,1);

% Run the wlanWaveformGenerator
waveformMAT = wlanWaveformGenerator(inputBits,cfgHESU,'WindowTransitionTime',WindowTransitionTime

% Run the generated MEX file
waveformMEX = wlanWaveformGenerator_HESU_mex(inputBits,cfgHESU,'WindowTransitionTime',WindowTrans

% Compare the outputs of the wlanWaveformGenerator and generated MEX file
difference = waveformMAT - waveformMEX;

% Check the results are consistent
if max(abs(difference),[],'all') > 1e-10
    error('The MEX file generated does not produce the same results as wlanWaveformGenerator.')
else
    disp('The outputs of the generated MEX file and the wlanWaveformGenerator are equal.')
end
```

```
The outputs of the generated MEX file and the wlanWaveformGenerator are equal.
```

Additional Options for Code Generation

In addition to the reporting options used above, you can configure MATLAB Coder with more advanced options.

- Generate C/C++ code only but do not build object code or MEX files setting `cfgCoder.GenCodeOnly = true`. This can save time when you iterate between modifying MATLAB code and inspecting the generated C/C++ code. To generate a MEX file, set this value to `false`.

- Configure MATLAB Coder for optimized C/C++ code generation by setting `cfgCoder.BuildConfiguration = 'Faster Runs'`. This option is available only for `'LIB'`, `'DLL'`, and `'EXE'` build types.

See the `coder.config` (MATLAB Coder) reference page for more information.

Limitations of Code Generation

"WLAN Channel Models" are System objects, which are designed specifically for implementing and simulating dynamic systems with inputs that change over time. Consider these limitations for code generation of System objects:

- Nontunable property values must be constant and can only be assigned once before a `step` method is executed, including the assignment in the constructor.

- You cannot pass in a System object™ to an entry-point function.

See "System Objects in MATLAB Code Generation" (MATLAB Coder) for more information about rules and limitations of System objects™ for code generation.

If you try to generate code for the entry-point function named `hCustomChannelNT`, which creates a `wlanTGaxChannel` with a bandwidth specified by the input argument `BW` and filters the signal `inputSignal`, the code generation process fails.

```
function signalOut = hCustomChannelNT(BW, signalIn)

    % Create a TGax channel with the appropriate bandwidth
    ch = wlanTGaxChannel('ChannelBandwidth',BW);
    % Filter the input signal
    signalOut = ch(signalIn);

end
```

The error message indicates: `Failed to compute constant value for nontunable property 'ChannelBandwidth'`.

The `ChannelBandwidth` property of the `wlanTGaxChannel` object is not a constant value because it depends on the input argument `BW` of the `hCustomChannelNT` function. You can work around this by setting `BW` to be a constant value.

```
inputSignal = coder.typeof(complex(0),[Inf 1],[1 0]); %#ok<NASGU>
codegen hCustomChannelNT -args {coder.Constant('CBW160'),inputSignal};
```

```
Code generation successful.
```

However, if you specify a constant value for the channel bandwidth, you cannot change the channel bandwidth at runtime. The following call `hCustomChannelNT_mex('CBW20',inputSignal)` fails because the specified argument `BW` is not `'CBW160'`. It is possible to work around this for a limited variability of the input arguments using a switch-case block. The function `hCustomChannel` contains a switch-case where each of the cases creates a `wlanTGaxChannel` with the appropriate channel bandwidth.

```
function signalOut = hCustomChannel(ChannelBandwidth, signalIn)

    % Create a TGax channel with the appropriate bandwidth and filter the input signal
    switch ChannelBandwidth
        case 'CBW20'
            ch = wlanTGaxChannel('ChannelBandwidth','CBW20');
            signalOut = ch(signalIn);
```

```matlab
            case 'CBW40'
                ch = wlanTGaxChannel('ChannelBandwidth','CBW40');
                signalOut = ch(signalIn);
            case 'CBW80'
                ch = wlanTGaxChannel('ChannelBandwidth','CBW80');
                signalOut = ch(signalIn);
            case 'CBW160'
                ch = wlanTGaxChannel('ChannelBandwidth','CBW160');
                signalOut = ch(signalIn);
            otherwise
                error('Invalid bandwidth configuration.')
        end

end
```

Note that `hCustomChannel` creates a new channel object every time it is used, so it does not preserve the state of the channel.

Next, generate code for `hCustomChannel` and select multiple channel bandwidths at runtime.

```matlab
inputSignal = coder.typeof(complex(0),[Inf 1],[1 0]);
BW = coder.typeof('CBW160',[1 6],[0 1]);
inputArgs = {BW,inputSignal};
codegen hCustomChannel -args inputArgs

cfgHESU = wlanHESUConfig;
cfgHESU.ChannelBandwidth = 'CBW20';

inputSignal = wlanWaveformGenerator_HESU_mex([1 0 1]', cfgHESU,'WindowTransitionTime',0);

rng('default') % Set the default random number generator for reproduction purposes
outputSignalMEX = hCustomChannel_mex('CBW20',inputSignal);
rng('default')
outputSignalMAT = hCustomChannel('CBW20',inputSignal);

% Calculate the difference between the MEX and MATLAB files outputs
difference = outputSignalMAT - outputSignalMEX;

% Check the results are consistent
if max(abs(difference),[],'all') > 1e-10
    error('The generated MEX file does not produce the same results as hCustomChannel.')
else
    disp('The outputs of the generated MEX file and hCustomChannel are equal.')
end
```

```
Code generation successful.

The outputs of the generated MEX file and hCustomChannel are equal.
```

Change the channel bandwidth and pass the signal through the channel.

```matlab
cfgHESU.ChannelBandwidth = 'CBW40';
inputSignal = wlanWaveformGenerator_HESU_mex([1 0 1]', cfgHESU,'WindowTransitionTime',0);

rng('default') % Set the default random number generator for reproduction purposes
outputSignalMEX = hCustomChannel_mex('CBW40',inputSignal);
rng('default')
```

```
outputSignalMAT = hCustomChannel('CBW40',inputSignal);

% Calculate the difference between the MEX and MATLAB files outputs
difference = outputSignalMAT - outputSignalMEX;

% Check the results are consistent
if max(abs(difference),[],'all') > 1e-10
    error('The MEX file generated does not produce the same results as hCustomChannel.')
else
    disp('The outputs of the generated MEX file and hCustomChannel are equal.')
end
```

```
The outputs of the generated MEX file and hCustomChannel are equal.
```

Further Investigation

You can integrate the generated code with the SystemVerilog Direct Programming Interface (DPI) to export MATLAB algorithms to ASIC or FPGA verification environments including Synopsys VCS®, Cadence Incisive or Xcelium, and Mentor Graphics ModelSim or Questa. You can automatically generate SystemVerilog DPI components from MATLAB functions using MATLAB Coder™ with HDL Verifier™. The following workflow can be used with MATLAB functions that generate stimuli and perform analysis or with a MATLAB function that is a behavioral golden reference for the DUT (Device Under Test) in the ASIC or FPGA verification environment. See "SystemVerilog DPI Component Generation" (HDL Verifier) (HDL Verifier) to learn more.
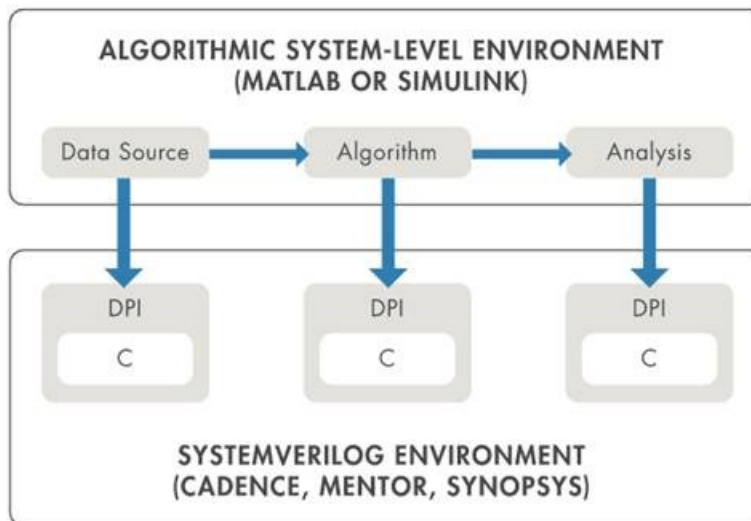
Software-Defined Radio

# 802.11a Transmission and Reception Using Analog Devices AD9361/AD9364

This example shows how to use the Xilinx® Zynq-based radio support package with MATLAB® and WLAN Toolbox™ to generate a simultaneous transmission and reception on a single SDR platform.

Example Summary

WLAN Toolbox can be used to generate standard-compliant baseband IQ waveforms. These baseband waveforms can be modulated for RF transmission using SDR radio hardware such as Xilinx Zynq-based radio.

In this example, an image file is imported and packed into multiple WLAN packets of a baseband waveform that is generated using WLAN Toolbox. A single antenna is used to generate an IEEE® 802.11a™ waveform. The RF WLAN waveform is then created, the baseband waveform is transferred to the hardware memory on the Zynq radio and transmitted over the air.

The RF card used in this example is capable of simultaneous transmission and reception. Therefore, the transmitted signal is captured using the same Zynq radio hardware platform. The diagram below shows the setup used.
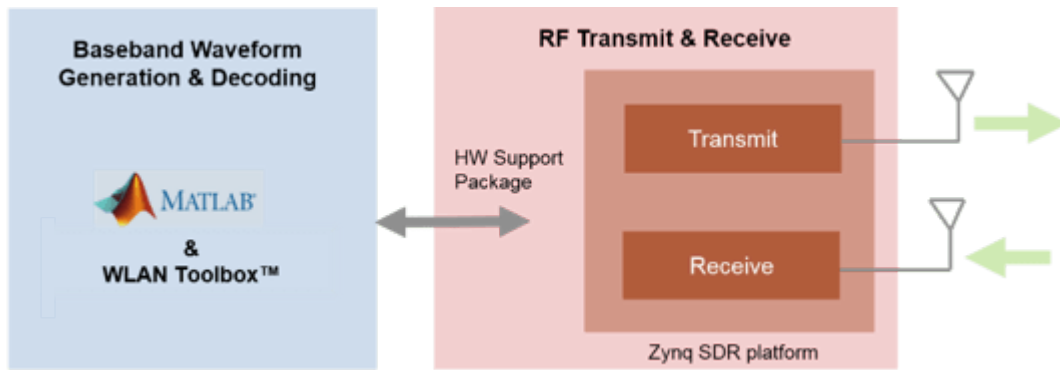
The receiver captures a number of WLAN packets and performs synchronization, channel estimation and equalization to retrieve packet parameters. The data field is then extracted and the transmitted payload is recovered using the retrieved packet parameters. After decoding the received waveform, the transmitted image is recovered.

SDR Support Packages

This example requires the Xilinx Zynq-based radio support package. This can be installed using the Add-On Explorer.

More information about other supported SDR platforms can be found here.

Full Example

The full example description and source code can be found in the list of examples using Xilinx Zynq-Based Radio under the name "802.11a Transmission and Reception Using Analog Devices AD9361/AD9364".

802.11 OFDM Beacon Receiver with USRP® Hardware

This example shows how to use the Universal Software Radio Peripheral (USRP®) device using SDRu (Software Defined Radio USRP®) System objects™ to implement a WLAN receiver. The receiver is able to recover 802.11™ OFDM non-HT beacon frames transmitted over the air from commercial 802.11 hardware.

Example Summary

WLAN Toolbox™ provides functions and tools to decode 802.11 waveforms. This example shows how to receive signals from commercial WLAN transmitters in MATLAB using USRP®. A receiver design is demonstrated including synchronization, transmission configuration recovery, and payload decoding for non-HT packets.
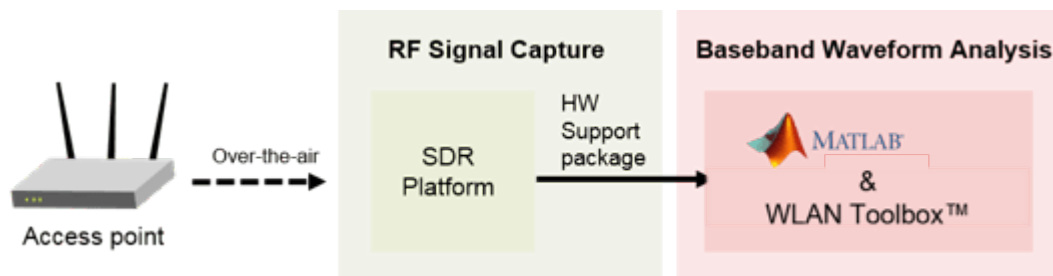
In this example, OFDM beacon packets corrupted by the transmission over the air are captured and processed to recover the payload contents. To recover beacon packets the receiver performs packet detection, symbol timing and frequency offset correction, and L-SIG and payload decoding. The resulting payload bits are then evaluated to determine whether the payload is a beacon frame and the contents are displayed as appropriate.

To view an example of a WLAN front end processing which does not require SDR hardware see "802.11 OFDM Beacon Receiver with Captured Data" on page 4-41.

SDR Support Packages

This example requires the USRP®-based radio support package. This can be installed using the Add-On Explorer.

More information about other supported SDR platforms can be found here.

Full Example

The full example description and source code can be found in the list of examples using USRP® under the name "IEEE 802.11 WLAN - OFDM Beacon Receiver with USRP® Hardware".

Copyright Notice

Universal Software Radio Peripheral® and USRP® are trademarks of National Instruments Corp.

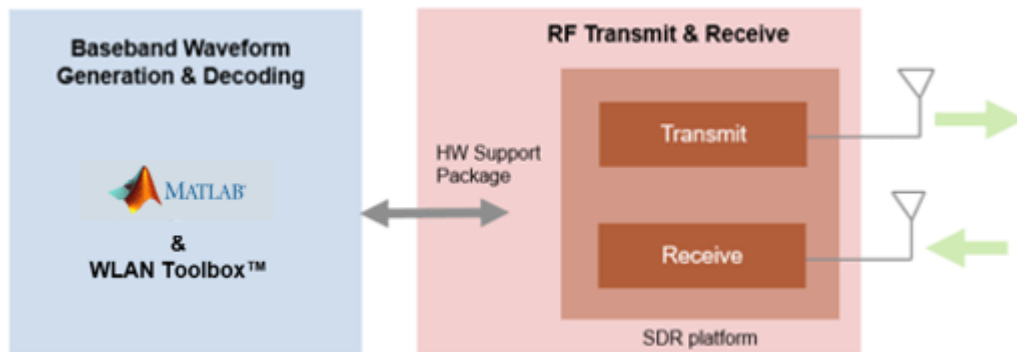# Transmission and Reception of an Image Using WLAN Toolbox and a Single USRP® E310

This example shows how to transmit and receive WLAN packets on a single radio platform, USRP® Embedded Series Radio Support Package with MATLAB® and WLAN Toolbox™. An image file is encoded and packed into WLAN packets for transmission, and subsequently decoded on reception.

Example Summary

WLAN Toolbox can be used to generate standard-compliant baseband IQ waveforms. These baseband waveforms can be modulated for RF transmission using SDR radio hardware such as USRP® E310 radio.

In this example, an image file is imported and packed into multiple WLAN packets of a baseband waveform that is generated using WLAN Toolbox. A single antenna is used to generate an IEEE® 802.11a™ waveform. The RF WLAN waveform is then created, the baseband waveform is transferred to the hardware memory on the USRP® E310 radio and transmitted over the air.

The Repeated Waveform Transmitter functionality with the USRP® Embedded Series radio hardware, allows a waveform to be transmitted over the air and is received using the same SDR hardware. The diagram below shows the setup used.

The receiver captures a number of WLAN packets and performs synchronization, channel estimation and equalization to retrieve packet parameters. The data field is then extracted and the transmitted payload is recovered using the retrieved packet parameters. After decoding the received waveform, the transmitted image is recovered.

SDR Support Packages

This example requires the Support Package for USRP® Embedded Series Radio. This can be installed using the Add-On Explorer.

More information about other supported SDR platforms can be found here.

Full Example

The full example description and source code can be found in the list of examples using USRP® Embedded Series Radio under the name "Transmission and Reception of an Image Using WLAN Toolbox and a Single USRP® E310".

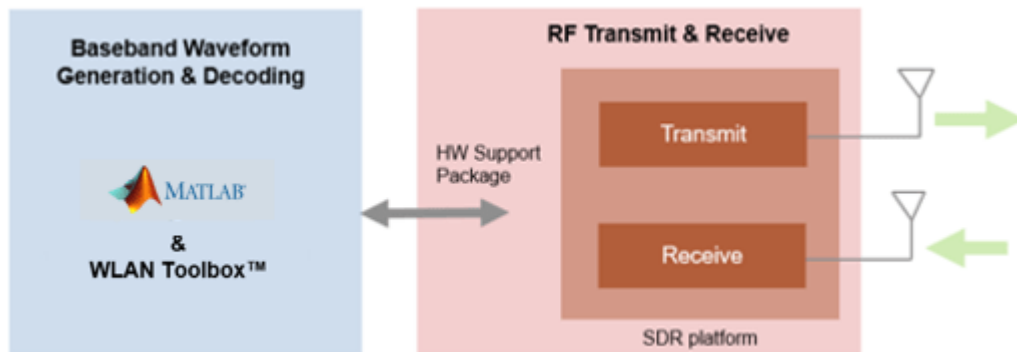# Image Transmission and Reception Using WLAN Toolbox and One PlutoSDR

This example shows how to transmit and receive WLAN packets on a single PlutoSDR device, using the Communications Toolbox™ Support Package for ADALM-PLUTO Radio and the WLAN Toolbox™. An image file is encoded and packed into WLAN packets for transmission, and subsequently decoded on reception.

Example Summary

WLAN Toolbox can be used to generate standard-compliant baseband IQ waveforms. These baseband waveforms can be modulated for RF transmission using SDR radio hardware such as PlutoSDR.

In this example, an image file is imported and packed into multiple WLAN packets of a baseband waveform that is generated using WLAN Toolbox. A single antenna is used to generate an IEEE® 802.11a™ waveform. The RF WLAN waveform is then created, the baseband waveform is transferred to the hardware memory on the PlutoSDR and transmitted over the air.

The Repeated Waveform Transmitter functionality with the PlutoSDR allows a waveform to be transmitted over the air and is received using the same SDR hardware. The diagram below shows the setup used.

The receiver captures a number of WLAN packets and performs synchronization, channel estimation and equalization to retrieve packet parameters. The data field is then extracted and the transmitted payload is recovered using the retrieved packet parameters. After decoding the received waveform, the transmitted image is recovered.

SDR Support Packages

This example requires the Support Package for ADALM-PLUTO Radio. This can be installed using the Add-On Explorer.

More information about other supported SDR platforms can be found here.

Full Example

The full example description and source code can be found in the list of examples using ADALM-PLUTO Radio under the name "Image Transmission and Reception Using WLAN Toolbox and One PlutoSDR".