WLAN System Toolbox™

Reference

MATLAB®

R2015b

MathWorks®

How to Contact MathWorks

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

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

WLAN System Toolbox™

Revision History

==Month and year of printing batch, e.g., September 2000==	Online only	New for Version 1.0 (R2015b)

Contents

functions/functions — Alphabetical List

wlanCoarseCFOEstimate

Coarse estimate of carrier frequency offset

Syntax

```
fOffset = wlanCoarseCFOEstimate(rxSig,cbw)
fOffset = wlanCoarseCFOEstimate(rxSig,cbw,corrOffset)
```

Description

`fOffset = wlanCoarseCFOEstimate(rxSig,cbw)` returns a coarse estimate of the carrier frequency offset (CFO) given received time-domain “L-STF” on page 1-8[1] samples `rxSig` and channel bandwidth `cbw`.

`fOffset = wlanCoarseCFOEstimate(rxSig,cbw,corrOffset)` returns a coarse estimate given correlation offset `corrOffset`.

Examples

Coarse Estimate of CFO for Non-HT Waveform

Create non-HT and generator configuration objects.

```
nht = wlanNonHTConfig;
wgc = wlanGeneratorConfig;
```

Generate a non-HT waveform.

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

Create a phase and frequency offset object and introduce a 2 kHz frequency offset.

```
pfo = comm.PhaseFrequencyOffset('SampleRate',20e6,'FrequencyOffset',2000);
```

1. IEEE® Std 802.11™-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.

```
rxSig = step(pfo,txSig);
```

Extract the L-STF.

```
ind = wlanFieldIndices(nht,'L-STF');
rxLSTF = rxSig(ind(1):ind(2),:);
```

Estimate the frequency offset from the L-STF.

```
freqOffsetEst = wlanCoarseCFOEstimate(rxLSTF,'CBW20')
```

```
freqOffsetEst =

   2.0000e+03
```

Estimate and Correct CFO for VHT Waveform with Correlation Offset

Estimate the frequency offset for a VHT signal passing through a noisy, TGac channel. Correct for the frequency offset.

Create a VHT configuration object and create the L-STF.

```
vht = wlanVHTConfig;
txstf = wlanLSTF(vht);
```

Set the channel bandwidth and sample rate.

```
cbw = 'CBW80';
fs = 80e6;
```

Create TGac and thermal noise channel objects. Set the delay profile of the TGac channel to `'Model-C'`. Set the noise figure of the thermal noise channel to 9 dB.

```
tgac = wlanTGacChannel('SampleRate',fs,'ChannelBandwidth',cbw, ...
    'DelayProfile','Model-C','LargeScaleFadingEffect','Pathloss');

noise = comm.ThermalNoise('SampleRate',fs,'NoiseMethod','Noise figure', ...
    'NoiseFigure',9);
```

Pass the L-STF through the noisy TGac channel.

```
rxstfNoNoise = step(tgac,txstf);
```

```
rxstf = step(noise,rxstfNoNoise);
```

Create a phase and frequency offset object and introduce a 750 Hz frequency offset.

```
pfo = comm.PhaseFrequencyOffset('SampleRate',fs, ...
    'FrequencyOffsetSource','Input port');
rxstf = step(pfo,rxstf,750);
```

For the model-C delay profile, the RMS delay spread is 30 ns, which is 3/8 of the 80 ns short training symbol duration. As such, set the correlation offset to 0.375.

```
corrOffset = 0.375;
```

Estimate the frequency offset. Your results may differ slightly.

```
fOffsetEst = wlanCoarseCFOEstimate(rxstf,cbw,corrOffset)
```

```
fOffsetEst =

  749.8770
```

The estimate is very close to the introduced CFO of 750 Hz.

Change the delay profile to `'Model-E'`, which has an RMS delay spread of 100 ns.

```
release(tgac)
tgac.DelayProfile = 'Model-E';
```

Pass the transmitted signal through the modified channel and apply the 750 Hz CFO.

```
rxstfNoNoise = step(tgac,txstf);
rxstf = step(noise,rxstfNoNoise);
rxstf = step(pfo,rxstf,750);
```

Estimate the frequency offset.

```
fOffsetEst = wlanCoarseCFOEstimate(rxstf,cbw,corrOffset)
```

```
fOffsetEst =

  682.9147
```

The estimate is inaccurate because the RMS delay spread is greater than the duration of the training symbol.

Set the correlation offset to the maximum value of 1 and estimate the CFO.

```
corrOffset = 1;
fOffsetEst = wlanCoarseCFOEstimate(rxstf,cbw,corrOffset)
```

```
fOffsetEst =

  747.8501
```

The estimate is accurate because the autocorrelation does not use the first training symbol. The channel delay renders this symbol useless.

Correct for the estimated frequency offset.

```
rxstfCorrected = step(pfo,rxstf,-fOffsetEst);
```

Estimate the frequency offset of the corrected signal.

```
fOffsetEstCorr = wlanCoarseCFOEstimate(rxstfCorrected,cbw,corrOffset)
```

```
fOffsetEstCorr =

  -3.5283e-11
```

The corrected signal has negligible frequency offset.

Two-Step CFO Estimation and Correction

Estimate and correct for a significant carrier frequency offset in two steps. Estimate the frequency offset after all corrections have been made.

Set the channel bandwidth and the corresponding sample rate.

```
cbw = 'CBW40';
fs = 40e6;
```

Coarse Frequency Correction

Generate an HT PPDU by creating `wlanHTConfig` and `wlanGeneratorConfig` objects.

```
cfg = wlanHTConfig('ChannelBandwidth',cbw);
wgc = wlanGeneratorConfig;
```

Generate the transmit waveform.

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

Create TGn and thermal noise channel objects. Set the noise figure of the receiver to 9 dB.

```
tgn = wlanTGnChannel('SampleRate',fs,'DelayProfile','Model-D', ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
noise = comm.ThermalNoise('SampleRate',fs, ...
    'NoiseMethod','Noise figure', ...
    'NoiseFigure',9);
```

Pass the waveform through the TGn channel and add noise.

```
rxSigNoNoise = step(tgn,txSig);
rxSig = step(noise,rxSigNoNoise);
```

Create a phase and frequency offset object to introduce a carrier frequency offset. Introduce a 2 kHz frequency offset.

```
pfo = comm.PhaseFrequencyOffset('SampleRate',fs,'FrequencyOffsetSource','Input port');
rxSig = step(pfo,rxSig,2e3);
```

Extract the L-STF signal for coarse frequency offset estimation.

```
istf = wlanFieldIndices(cfg,'L-STF');
rxstf = rxSig(istf(1):istf(2),:);
```

Perform a coarse estimate of the frequency offset. Your results may differ.

```
foffset1 = wlanCoarseCFOEstimate(rxstf,cbw)
```

```
foffset1 =

   2.0002e+03
```

Correct for the estimated offset.

```
rxSigCorr1 = step(pfo,rxSig,-foffset1);
```

Fine Frequency Correction

Extract the L-LTF signal for fine offset estimation.

```
iltf = wlanFieldIndices(cfg,'L-LTF');
rxltf1 = rxSigCorr1(iltf(1):iltf(2),:);
```

Perform a fine estimate of the corrected signal.

```
foffset2 = wlanFineCFOEstimate(rxltf1,cbw)
```

```
foffset2 =

    6.6079
```

The corrected signal offset is reduced from 2000 Hz to approximately 7 Hz.

Correct for the remaining offset.

```
rxSigCorr2 = step(pfo,rxSigCorr1,-foffset2);
```

Determine the frequency offset of the twice corrected signal.

```
rxltf2 = rxSigCorr2(iltf(1):iltf(2),:);
deltaFreq = wlanFineCFOEstimate(rxltf2,cbw)
```

```
deltaFreq =

     0
```

The CFO is zero.

Input Arguments

`rxSig` — Received signal
matrix

Received signal containing an L-STF, specified as an N_S-by-N_R matrix. N_S is the number of samples in the L-STF and N_R is the number of receive antennas.

Note: If the number of samples in `rxSig` is greater than the number of samples in the L-STF, the trailing samples are not used to estimate the carrier frequency offset.

Data Types: `double`
Complex Number Support: Yes

`cbw` — Channel bandwidth
string

Channel bandwidth in MHz, specified as one of these strings: `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`.

Data Types: `char`

`corrOffset` — Correlation offset
0.75 (default) | real scalar from 0 to 1

Correlation offset as a fraction of a short training symbol, specified as a real scalar from 0 to 1. Each short training symbol has a duration of 0.8 μs.

Data Types: `double`

Output Arguments

`fOffset` — Frequency offset
real scalar

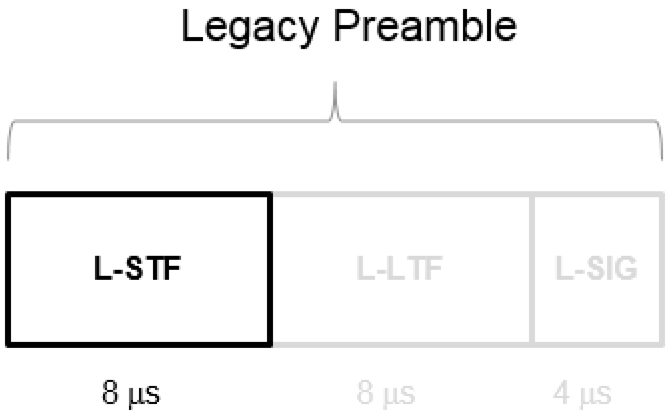
Frequency offset in Hz, returned as a real scalar.

Data Types: `double`

More About

L-STF

The legacy short training field (L-STF) is the first field of the legacy preamble for 802.11a/g.



It is also the first field of the HT-mixed format preamble used in 802.11n™ and the VHT format preamble used in 802.11ac™. The L-STF is 8 µs in length and consists of 10 repeated 0.8 µs symbols. Because the sequence has good correlation properties, it is used for start-of-packet detection, for coarse frequency correction, and for setting the AGC. The sequence uses 12 of the 52 subcarriers that are available in 20 MHz. IEEE Std 802.11-2012, Equation 18-6 and Equation 18-7 define the L-STF.

References

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

[2] Li, Jian. “Carrier Frequency Offset Estimation for OFDM-Based WLANs.” *IEEE Signal Processing Letters*. Vol. 8, Issue 3, Mar 2001, pp. 80–82.

[3] Terry, J. and J. Heiskala. *OFDM Wireless LANs: A Theoretical and Practical Guide*. Indianapolis, IN: Sams Publishing, 2001.

[4] Moose, P. H. *A technique for orthogonal frequency division multiplexing frequency offset correction. IEEE Transactions on Communications*. Vol. 42, Issue 10, Oct 1994, pp. 2908–2914.

[5] 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.

See Also

`comm.PhaseFrequencyOffset` | `wlanFineCFOEstimate` | `wlanLSTF`

Introduced in R2015b

wlanFineCFOEstimate

Fine estimate of carrier frequency offset

Syntax

```
fOffset = wlanFineCFOEstimate(rxSig,cbw)
fOffset = wlanFineCFOEstimate(rxSig,cbw,corrOffset)
```

Description

`fOffset = wlanFineCFOEstimate(rxSig,cbw)` returns a fine estimate of the carrier frequency offset (CFO) given received time-domain “L-LTF” on page 1-16[2] samples `rxSig` and channel bandwidth `cbw`.

`fOffset = wlanFineCFOEstimate(rxSig,cbw,corrOffset)` returns the estimated frequency offset given correlation offset `corrOffset`.

Examples

Fine Estimate of Carrier Frequency Offset

Create non-HT and generator configuration objects.

```
nht = wlanNonHTConfig;
wgc = wlanGeneratorConfig;
```

Generate a non-HT waveform.

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

Create a phase and frequency offset object and introduce a 2 Hz frequency offset.

```
pfo = comm.PhaseFrequencyOffset('SampleRate',20e6,'FrequencyOffset',2);
```

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

```
rxSig = step(pfo,txSig);
```

Extract the L-LTF and estimate the frequency offset.

```
ind = wlanFieldIndices(nht,'L-LTF');
rxlltf = rxSig(ind(1):ind(2),:);
freqOffsetEst = wlanFineCFOEstimate(rxlltf,'CBW20')
```

```
freqOffsetEst =

    2.0000
```

Estimate and Correct CFO for VHT Waveform

Estimate the frequency offset for a VHT signal passing through a noisy, TGac channel. Correct for the frequency offset.

Create a VHT configuration object and create the L-LTF.

```
vht = wlanVHTConfig;
txltf = wlanLLTF(vht);
```

Set the sample rate to correspond to the default bandwidth of the VHT configuration object.

```
fs = 80e6;
```

Create TGac and thermal noise channel objects. Set the noise figure of the AWGN channel to 10 dB.

```
tgac = wlanTGacChannel('SampleRate',fs, ...
    'ChannelBandwidth',vht.ChannelBandwidth, ...
    'DelayProfile','Model-C','LargeScaleFadingEffect','Pathloss');

noise = comm.ThermalNoise('SampleRate',fs, ...
    'NoiseMethod','Noise figure', ...
    'NoiseFigure',10);
```

Pass the L-LTF through the noisy TGac channel.

```
rxltfNoNoise = step(tgac,txltf);
rxltf = step(noise,rxltfNoNoise);
```

Create a phase and frequency offset object and introduce a 25 Hz frequency offset.

```
pfo = comm.PhaseFrequencyOffset('SampleRate',fs,'FrequencyOffsetSource','Input port');
rxltf = step(pfo,rxltf,25);
```

Perform a fine estimate the frequency offset using a correlation offset of 0.6. Your results may differ slightly.

```
fOffsetEst = wlanFineCFOEstimate(rxltf,vht.ChannelBandwidth,0.6)
```

```
fOffsetEst =

   24.1252
```

Correct for the estimated frequency offset.

```
rxltfCorr = step(pfo,rxltf,-fOffsetEst);
```

Estimate the frequency offset of the corrected signal.

```
fOffsetEstCorr = wlanFineCFOEstimate(rxltfCorr,vht.ChannelBandwidth,0.6)
```

```
fOffsetEstCorr =

   6.8187e-13
```

The corrected signal has negligible frequency offset.

Two-Step CFO Estimation and Correction

Estimate and correct for a significant carrier frequency offset in two steps. Estimate the frequency offset after all corrections have been made.

Set the channel bandwidth and the corresponding sample rate.

```
cbw = 'CBW40';
fs = 40e6;
```

Coarse Frequency Correction

Generate an HT PPDU by creating `wlanHTConfig` and `wlanGeneratorConfig` objects.

```
cfg = wlanHTConfig('ChannelBandwidth',cbw);
wgc = wlanGeneratorConfig;
```

Generate the transmit waveform.

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

Create TGn and thermal noise channel objects. Set the noise figure of the receiver to 9 dB.

```
tgn = wlanTGnChannel('SampleRate',fs,'DelayProfile','Model-D', ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
noise = comm.ThermalNoise('SampleRate',fs, ...
    'NoiseMethod','Noise figure', ...
    'NoiseFigure',9);
```

Pass the waveform through the TGn channel and add noise.

```
rxSigNoNoise = step(tgn,txSig);
rxSig = step(noise,rxSigNoNoise);
```

Create a phase and frequency offset object to introduce a carrier frequency offset. Introduce a 2 kHz frequency offset.

```
pfo = comm.PhaseFrequencyOffset('SampleRate',fs,'FrequencyOffsetSource','Input port');
rxSig = step(pfo,rxSig,2e3);
```

Extract the L-STF signal for coarse frequency offset estimation.

```
istf = wlanFieldIndices(cfg,'L-STF');
rxstf = rxSig(istf(1):istf(2),:);
```

Perform a coarse estimate of the frequency offset. Your results may differ.

```
foffset1 = wlanCoarseCFOEstimate(rxstf,cbw)
```

```
foffset1 =

   2.0002e+03
```

Correct for the estimated offset.

```
rxSigCorr1 = step(pfo,rxSig,-foffset1);
```

Fine Frequency Correction

Extract the L-LTF signal for fine offset estimation.

```
iltf = wlanFieldIndices(cfg,'L-LTF');
rxltf1 = rxSigCorr1(iltf(1):iltf(2),:);
```

Perform a fine estimate of the corrected signal.

```
foffset2 = wlanFineCFOEstimate(rxltf1,cbw)
```

```
foffset2 =

    6.6079
```

The corrected signal offset is reduced from 2000 Hz to approximately 7 Hz.

Correct for the remaining offset.

```
rxSigCorr2 = step(pfo,rxSigCorr1,-foffset2);
```

Determine the frequency offset of the twice corrected signal.

```
rxltf2 = rxSigCorr2(iltf(1):iltf(2),:);
deltaFreq = wlanFineCFOEstimate(rxltf2,cbw)
```

```
deltaFreq =

     0
```

The CFO is zero.

Input Arguments

`rxSig` — Received signal
matrix

Received signal containing an L-LTF, specified as an N_S-by-N_R matrix. N_S is the number of samples in the L-LTF and N_R is the number of receive antennas.

Note: If the number of samples in `rxSig` is greater than the number of samples in the L-LTF, the trailing samples are not used to estimate the carrier frequency offset.

Data Types: `double`
Complex Number Support: Yes

`cbw` — Channel bandwidth
string

Channel bandwidth in MHz, specified as one of these strings: `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`.

Data Types: `char`

`corrOffset` — Correlation offset
0.75 (default) | real scalar from 0 to 1

Correlation offset as a fraction of the L-LTF cyclic prefix, specified as a real scalar from 0 to 1. The cyclic prefix has a duration of 1.6 μs.

Data Types: `double`

Output Arguments

`fOffset` — Frequency offset
real scalar

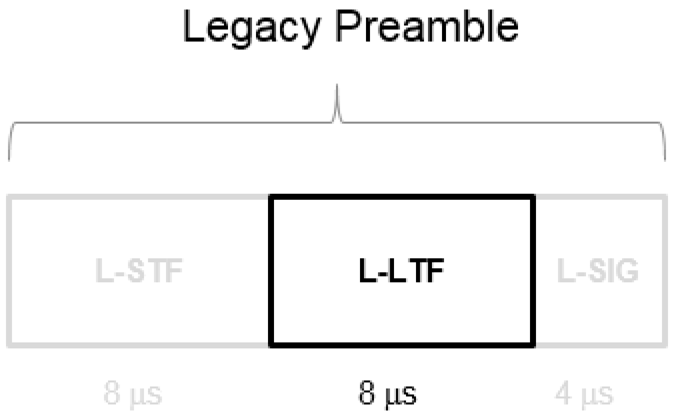
Frequency offset in Hz, returned as a real scalar.

Data Types: `double`

More About

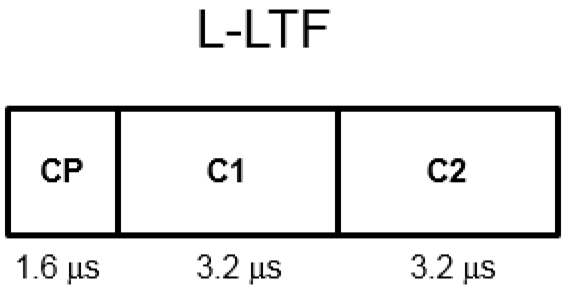
L-LTF

The legacy long training field (L-LTF) is the second field in the legacy preamble for 802.11a/g.



It is also the second field of the HT-mixed format preamble used in 802.11n and the VHT format preamble used in 802.11ac. Channel estimation, frequency offset estimation, and time synchronization rely on the L-LTF. The long OFDM training symbol consists of 53 subcarriers. IEEE Std 802.11-2012, Equation 18-8 and Equation 18-9 define the L-LTF.

The L-LTF, which is 8 μs in length, is composed of a 1.6 μs cyclic prefix (CP) followed by two identical 3.2 μs long training symbols (C1 and C2). The cyclic prefix (CP) consists of the second half of the long training symbol.



References

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

[2] Li, Jian. "Carrier Frequency Offset Estimation for OFDM-Based WLANs." *IEEE Signal Processing Letters*. Vol. 8, Issue 3, Mar 2001, pp. 80–82.

[3] Terry, J. and J. Heiskala. *OFDM Wireless LANs: A Theoretical and Practical Guide*. Indianapolis, IN: Sams Publishing, 2001.

[4] Moose, P. H. *A technique for orthogonal frequency division multiplexing frequency offset correction. IEEE Transactions on Communications*. Vol. 42, Issue 10, Oct 1994, pp. 2908–2914.

See Also

`comm.PhaseFrequencyOffset` | `wlanCoarseCFOEstimate` | `wlanLLTF`

Introduced in R2015b

wlanLLTFChannelEstimate

Channel estimation using L-LTF

Syntax

```
chEst = wlanLLTFChannelEstimate(demodSig,cfg)
chEst = wlanLLTFChannelEstimate(demodSig,cbw)
chEst = wlanLLTFChannelEstimate( ___ ,span)
```

Description

`chEst = wlanLLTFChannelEstimate(demodSig,cfg)` returns the channel estimate between the transmitter and all receive antennas using the demodulated "L-LTF" on page 1-28[3], `demodSig`, given the parameters specified in configuration object `cfg`.

`chEst = wlanLLTFChannelEstimate(demodSig,cbw)` returns the channel estimate given channel bandwidth `cbw`. The channel bandwidth can be used instead of the configuration object.

`chEst = wlanLLTFChannelEstimate( ___ ,span)` returns the channel estimate using any of the previous syntaxes and performs frequency smoothing over the specified filter span. See "Frequency Smoothing" on page 1-29.

Examples

Estimate SISO Channel Using L-LTF

Create VHT format and waveform generator configuration objects. Generate a time-domain waveform for an 802.11ac VHT packet.

```
vht = wlanVHTConfig;
```

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

```
wgc = wlanGeneratorConfig;
txWaveform = wlanWaveformGenerator([1;0;0;1],vht,wgc);
```

Multiply the transmitted VHT signal by -0.1 + 0.5i and pass it through an AWGN channel with a 30 dB signal-to-noise ratio.

```
rxWaveform = awgn(txWaveform*(-0.1+0.5i),30);
```

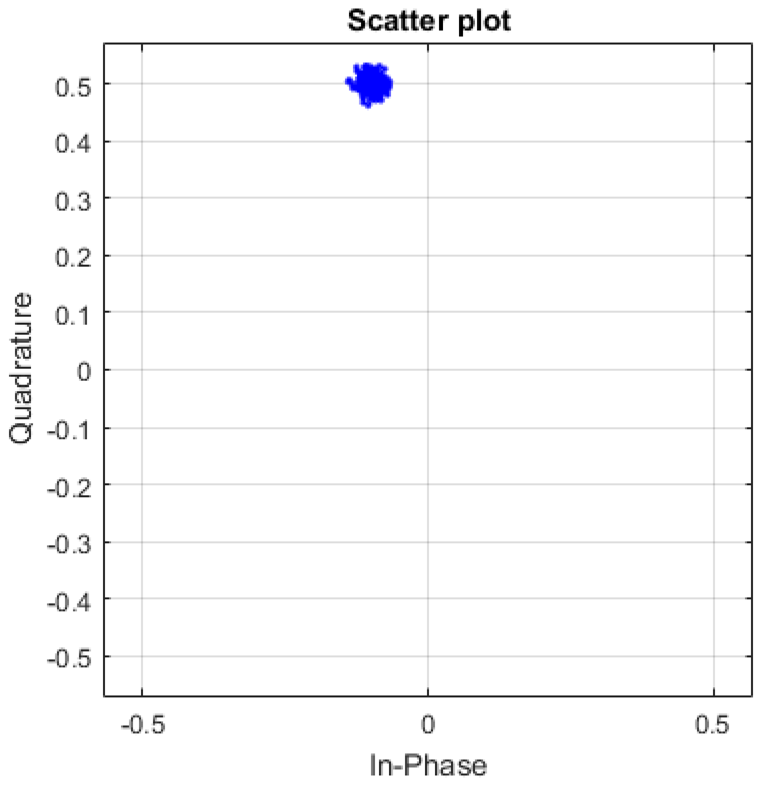
Extract the L-LTF field indices and demodulate the L-LTF. Perform channel estimation without frequency smoothing.

```
idxLLTF = wlanFieldIndices(vht,'L-LTF');
demodSig = wlanLLTFDemodulate(rxWaveform(idxLLTF(1):idxLLTF(2),:),vht);

est = wlanLLTFChannelEstimate(demodSig,vht);
```

Plot the channel estimate.

```
scatterplot(est)
grid
```



The channel estimate matches the complex channel multiplier.

Estimate 80 MHz SISO Channel Using L-LTF

Create VHT format and waveform generator configuration objects. Using these objects, generate a time-domain waveform for an 802.11ac VHT packet.

```
vht = wlanVHTConfig('ChannelBandwidth','CBW80');
wgc = wlanGeneratorConfig;
txWaveform = wlanWaveformGenerator([1;0;0;1],vht,wgc);
```

Multiply the transmitted VHT signal by -0.4 + 0.3i and pass it through an AWGN channel.

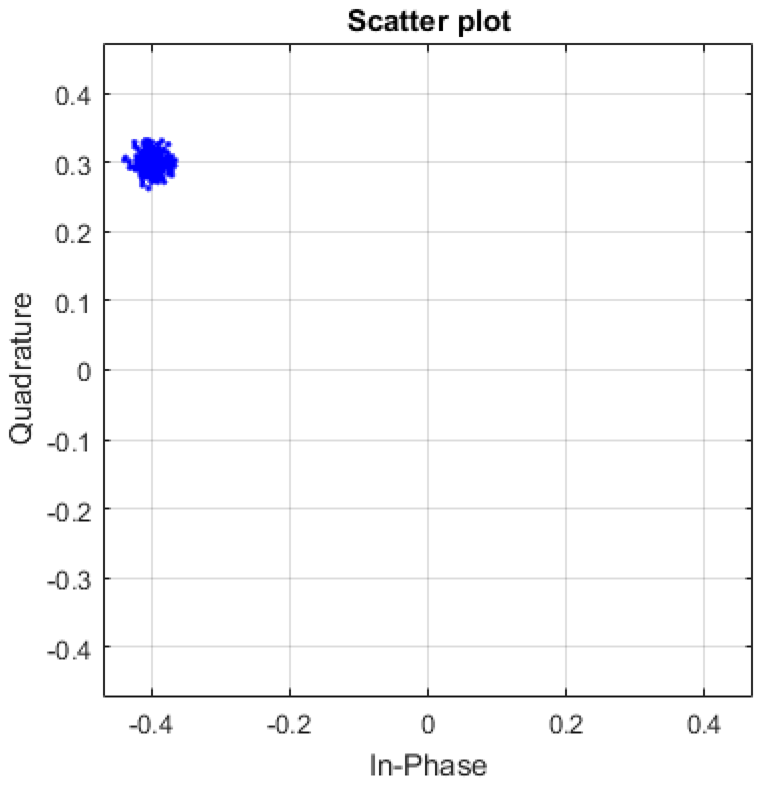
```
rxWaveform = awgn(txWaveform*(-0.4+0.3i),30);
```

Specify the channel bandwidth for demodulation and channel estimation. Extract the L-LTF field indices, demodulate the L-LTF, and perform channel estimation without frequency smoothing.

```
chanBW = 'CBW80';
idxLLTF = wlanFieldIndices(vht,'L-LTF');
demodSig = wlanLLTFDemodulate(rxWaveform(idxLLTF(1):idxLLTF(2),:),chanBW);
est = wlanLLTFChannelEstimate(demodSig,chanBW);
```

Plot the channel estimate.

```
scatterplot(est)
grid
```



The channel estimate matches the complex channel multiplier.

Estimate SISO Channel Using L-LTF and Smoothing Filter

Create VHT format and waveform generator configuration objects. Generate a time-domain waveform for an 802.11ac VHT packet.

```
vht = wlanVHTConfig;
wgc = wlanGeneratorConfig;
txWaveform = wlanWaveformGenerator([1;0;0;1],vht,wgc);
```

Multiply the transmitted VHT signal by 0.2 - 0.6i and pass it through an AWGN channel having a 10 dB SNR.

```
rxWaveform = awgn(txWaveform*complex(0.2,-0.6),10);
```

Extract the L-LTF from the received waveform. Demodulate the L-LTF.

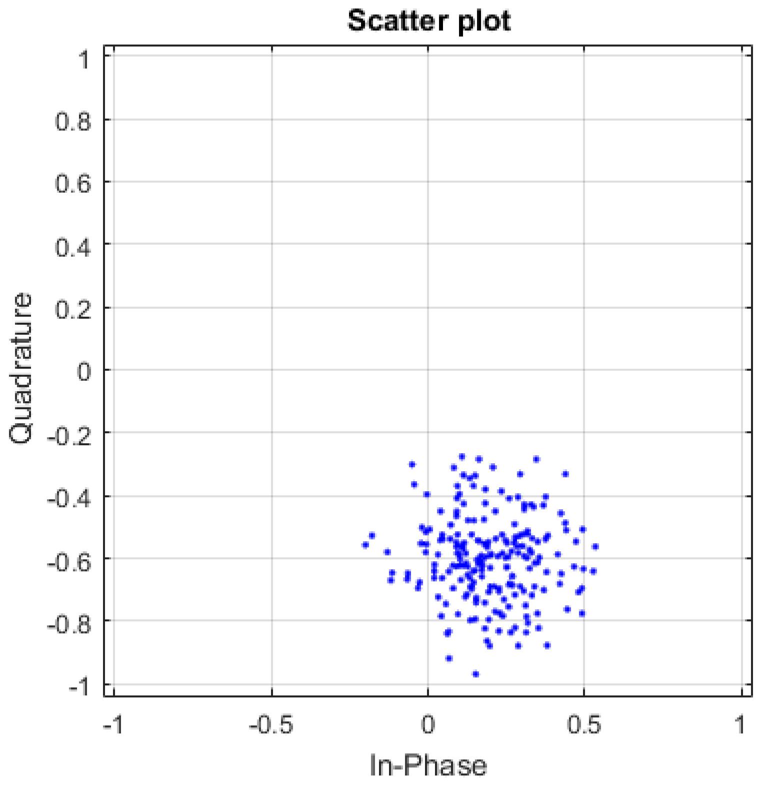
```
idxLLTF = wlanFieldIndices(vht, 'L-LTF');
lltfDemodSig = wlanLLTFDemodulate(rxWaveform(idxLLTF(1):idxLLTF(2),:),vht);
```

Use the demodulated L-LTF signal to generate the channel estimate.

```
est = wlanLLTFChannelEstimate(lltfDemodSig,vht);
```

Plot the channel estimate.

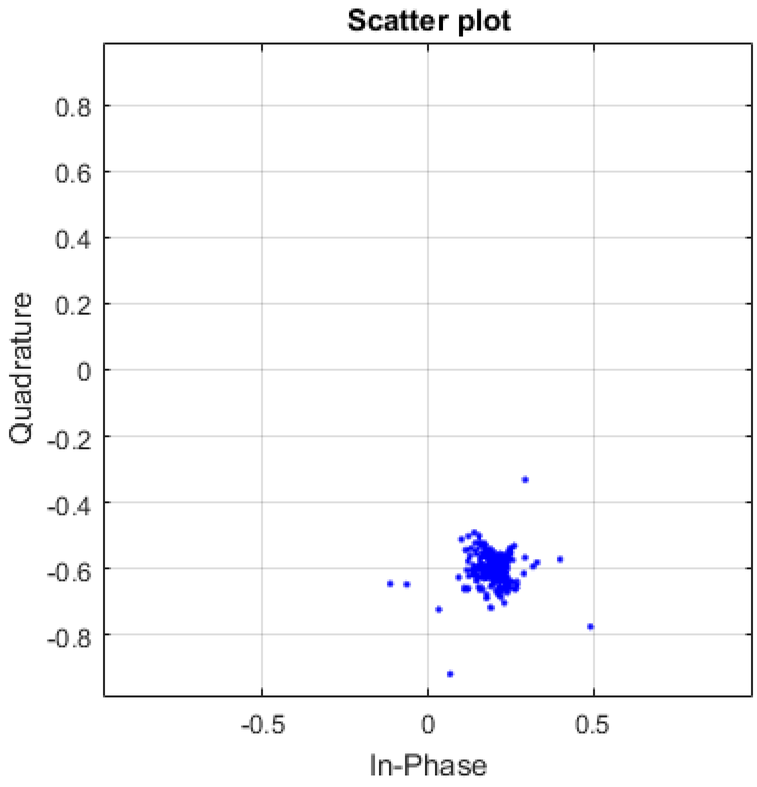
```
scatterplot(est)
grid
```



The channel estimate is noisy, which may lead to inaccurate data recovery.

Estimate the channel again with the filter span set to 11.

```
est = wlanLLTFChannelEstimate(lltfDemodSig,vht,11);
scatterplot(est)
grid
```



The filtering provides a better channel estimate.

Estimate Channel with L-LTF and Recover VHT-SIG-A

Create a VHT format configuration object. Generate L-LTF and VHT-SIG-A fields.

```
vht = wlanVHTConfig;
txLLTF = wlanLLTF(vht);
txSig = wlanVHTSIGA(vht);
```

Create a TGac channel for an 80 MHz bandwidth. Pass the transmitted L-LTF and VHT-SIG-A signals through the channel.

```
tgac = wlanTGacChannel('SampleRate',80e6,'ChannelBandwidth','CBW80', ...
    'LargeScaleFadingEffect','Pathloss and shadowing');

rxLLTFNoNoise = step(tgac,txLLTF);
rxSigNoNoise = step(tgac,txSig);
```

Create an AWGN channel having a noise variance corresponding to a 9 dB noise figure receiver. Pass the faded signals through the AWGN channel.

```
nVar = 10^((-228.6 + 10*log10(290) + 10*log10(80e6) + 9)/10);
chAWGN = comm.AWGNChannel('NoiseMethod','Variance','Variance',nVar);

rxLLTF = step(chAWGN,rxLLTFNoNoise);
rxSig = step(chAWGN,rxSigNoNoise);
```

Demodulate the received L-LTF.

```
demodLLTF = wlanLLTFDemodulate(rxLLTF,vht);
```

Estimate the channel using the demodulated L-LTF.

```
chEst = wlanLLTFChannelEstimate(demodLLTF,vht);
```

Recover the VHT-SIG-A signal and verify that there was no CRC failure.

```
[recBits,crcFail] = wlanVHTSIGARecover(rxSig,chEst,nVar,'CBW80');
crcFail
```

```
crcFail =

     0
```

Input Arguments

`demodSig` — Demodulated L-LTF OFDM symbols

3-D array

Demodulated L-LTF OFDM symbols, specified as an N_{ST}-by-N_{SYM}-by-N_R array. N_{ST} is the number of occupied subcarriers. N_{SYM} is the number of demodulated L-LTF symbols (one or two). N_R is the number of receive antennas. Each column of the 3-D array is a demodulated L-LTF OFDM symbol. If you specify two L-LTF symbols, `wlanLLTFChannelEstimate` averages the channel estimate over both symbols.

Data Types: `double`
Complex Number Support: Yes

`cfg` — Format configuration
`wlanVHTConfig` object | `wlanHTConfig` object | `wlanNonHTConfig` object

Format configuration, specified as one of these objects:

- `wlanVHTConfig` for VHT format
- `wlanHTConfig` for HT format
- `wlanNonHTConfig` for non-HT format

The `wlanLLTFChannelEstimate` function uses the `ChannelBandwidth` property of `cfg`.

`cbw` — Channel bandwidth
`'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth of the packet transmission waveform, specified as one of these strings: `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`.

PPDU Transmission Format	Acceptable Channel Bandwidth
VHT	`'CBW20'`, `'CBW40'`, `'CBW80'` (default), or `'CBW160'`
HT	`'CBW20'` (default) or `'CBW40'`
Non-HT	`'CBW20'`

Data Types: `char`

`span` — Filter span
positive odd integer

Filter span of the frequency smoothing filter, specified as a positive odd integer and expressed as a number of subcarriers. Frequency smoothing is applied only when `span` is specified and is greater than one. See “Frequency Smoothing” on page 1-29.

Note: Frequency smoothing is recommended only when a single transmit antenna is used.

Data Types: `double`

Output Arguments

`chEst` — Channel estimate

3-D array

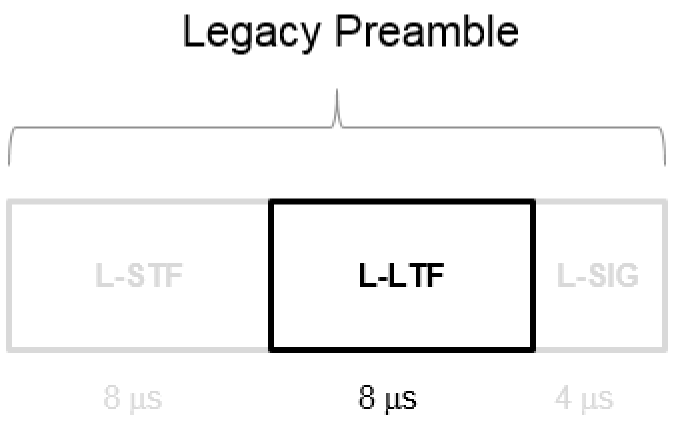
Channel estimate containing data and pilot subcarriers, returned as an N_{ST}-by-1-by-N_R array. N_{ST} is the number of occupied subcarriers. The value of 1 corresponds to the single transmitted stream in the L-LTF. N_R is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

More About

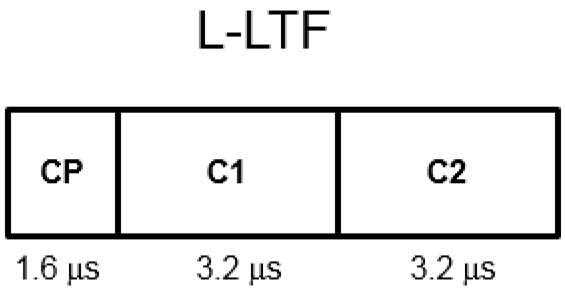
L-LTF

The legacy long training field (L-LTF) is the second field in the legacy preamble for 802.11a/g.



It is also the second field of the HT-mixed format preamble used in 802.11n and the VHT format preamble used in 802.11ac. Channel estimation, frequency offset estimation, and time synchronization rely on the L-LTF. The long OFDM training symbol consists of 53 subcarriers. IEEE Std 802.11-2012, Equation 18-8 and Equation 18-9 define the L-LTF.

The L-LTF, which is 8 µs in length, is composed of a 1.6 µs cyclic prefix (CP) followed by two identical 3.2 µs long training symbols (C1 and C2). The cyclic prefix (CP) consists of the second half of the long training symbol.



Frequency Smoothing

Frequency smoothing can improve channel estimation for highly correlated channels by averaging out white noise.

Frequency smoothing is recommended only for cases in which a single transmit antenna is used. Frequency smoothing consists of applying a moving-average filter that spans multiple adjacent subcarriers. Channel conditions dictate whether frequency smoothing is beneficial.

- If adjacent subcarriers are highly correlated, frequency smoothing results in significant noise reduction.
- In a highly frequency-selective channel, smoothing can degrade the quality of the channel estimate.

References

[1] Van de Beek, J-J., et al. "On channel estimation in OFDM systems." Vehicular Technology Conference. 1995 IEEE 45th, Vol. 2, 1995.

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

See Also

`wlanHTConfig` | `wlanHTLTFChannelEstimate` | `wlanLLTFDemodulate` | `wlanNonHTConfig` | `wlanVHTConfig` | `wlanVHTLTFChannelEstimate`

Introduced in R2015b

wlanHTLTFChannelEstimate

Channel estimation using HT-LTF

Syntax

```
chEst = wlanHTLTFChannelEstimate(demodSig,cfg)
chEst = wlanHTLTFChannelEstimate(demodSig,cfg,span)
```

Description

`chEst = wlanHTLTFChannelEstimate(demodSig,cfg)` returns the channel estimate using the demodulated "HT-LTF" on page 1-37[4] signal, `demodSig`, given the parameters specified in configuration object `cfg`.

`chEst = wlanHTLTFChannelEstimate(demodSig,cfg,span)` returns the channel estimate and specifies the span of a moving-average filter used to perform frequency smoothing.

Examples

Estimate SISO Channel Using HT-LTF

Estimate and plot the channel coefficients of an HT-mixed format channel by using the high throughput long training field.

Create an HT format configuration object. Generate the corresponding HT-LTF based on the object.

```
cfg = wlanHTConfig;
txSig = wlanHTLTF(cfg);
```

Multiply the transmitted HT-LTF signal by 0.2 + 0.1i and pass it through an AWGN channel. Demodulate the received signal.

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

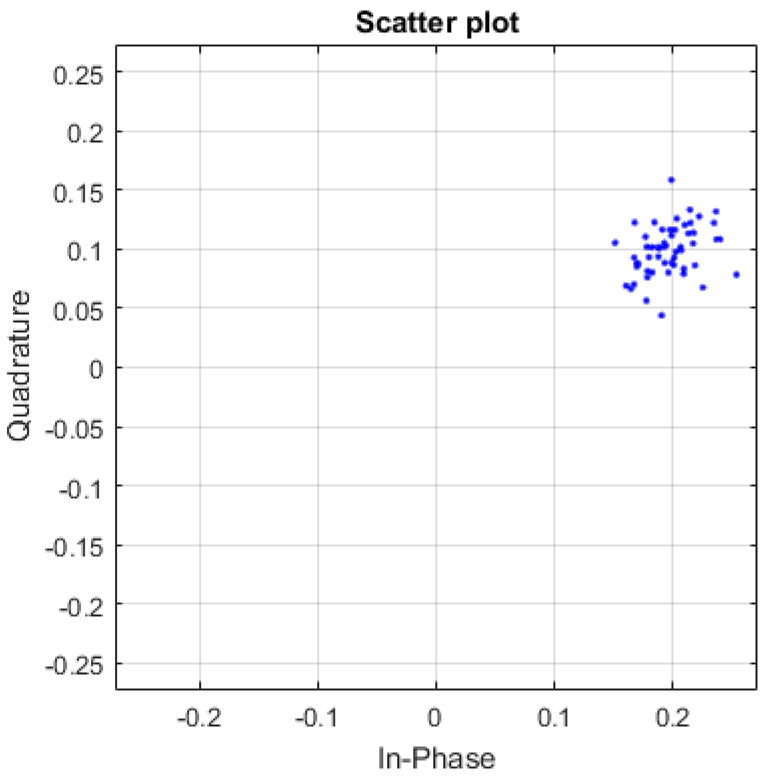
```
rxSig = awgn(txSig*(0.2+0.1i),30);
demodSig = wlanHTLTFDemodulate(rxSig,cfg);
```

Estimate the channel response using the demodulated HT-LTF.

```
est = wlanHTLTFChannelEstimate(demodSig,cfg);
```

Plot the channel estimate.

```
scatterplot(est)
grid
```



The channel estimate matches the complex channel multiplier.

Estimate MIMO Channel Using HT-LTF

Estimate the channel coefficients of a 2x2 MIMO channel by using the high throughput long training field. Recover the HT-data field and determine the number of bit errors.

Create an HT-mixed format configuration object for a channel having two spatial streams and four transmit antennas. Create the corresponding generator configuration object. Transmit a complete HT waveform.

```
cfg = wlanHTConfig('NumTransmitAntennas',2, ...
    'NumSpaceTimeStreams',2,'MCS',11);
wgc = wlanGeneratorConfig();
txPSDU = randi([0 1],8*cfg.PSDULength,1);
txWaveform = wlanWaveformGenerator(txPSDU,cfg,wgc);
```

Pass the transmitted waveform through a 2x2 TGn channel.

```
tgn = wlanTGnChannel('SampleRate',20e6, ...
    'NumTransmitAntennas',2, ...
    'NumReceiveAntennas',2, ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
rxWaveformNoNoise = step(tgn,txWaveform);
```

Create an AWGN channel with noise power, `nVar`, corresponding to a receiver having a 9 dB noise figure. The noise power is equal to $kTBF$, where k is Boltzmann's constant, T is the ambient noise temperature (290K), B is the bandwidth (20 MHz), and F is the noise figure (9 dB).

```
nVar = 10^((-228.6 + 10*log10(290) + 10*log10(20e6) + 9)/10);
chAWGN = comm.AWGNChannel('NoiseMethod','Variance', ...
    'Variance',nVar);
```

Pass the signal through the AWGN channel.

```
rxWaveform = step(chAWGN,rxWaveformNoNoise);
```

Determine the indices for the HT-LTF. Extract the HT-LTF from the received waveform. Demodulate the HT-LTF.

```
indLTF  = wlanFieldIndices(cfg,'HT-LTF');
rxLTF = rxWaveform(indLTF(1):indLTF(2),:);
```

```
ltfDemodSig = wlanHTLTFDemodulate(rxLTF,cfg);
```

Generate the channel estimate by using the demodulated HT-LTF signal. Specify a smoothing filter span of three subcarriers.

```
chEst = wlanHTLTFChannelEstimate(ltfDemodSig,cfg,3);
```

Extract the HT-data field from the received waveform.

```
indData = wlanFieldIndices(cfg,'HT-Data');
rxDataField = rxWaveform(indData(1):indData(2),:);
```

Recover the data and verify that there no bit errors occurred.

```
rxPSDU = wlanHTDataRecover(rxDataField,chEst,nVar,cfg);

numErrs = biterr(txPSDU,rxPSDU)
```

```
numErrs =

     0
```

Input Arguments

`demodSig` — Demodulated HT-LTF signal

3-D array

Demodulated HT-LTF signal, specified as an N_{ST}-by-N_{SYM}-by-N_R array. N_{ST} is the number of occupied subcarriers, N_{SYM} is the number of HT-LTF OFDM symbols, and N_R is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

`cfg` — Configuration information

`wlanHTConfig`

Configuration information, specified as a `wlanHTConfig` object. The function uses the following `wlanHTConfig` object properties:

`ChannelBandwidth` — Channel bandwidth

'CBW20' (default) | 'CBW40'

Channel bandwidth in MHz, specified as `'CBW20'` or `'CBW40'`.

Data Types: `char`

`NumSpaceTimeStreams` — Number of space-time streams

1 (default) | 2 | 3 | 4

Number of space-time streams in the transmission, specified as 1, 2, 3, or 4.

Data Types: `double`

`NumExtensionStreams` — Number of extension spatial streams

0 (default) | 1 | 2 | 3

Number of extension spatial streams in the transmission, specified as 0, 1, 2, or 3. When `NumExtensionStreams` is greater than 0, `SpatialMapping` must be `'Custom'`.

Data Types: `double`

`MCS` — Modulation and coding scheme

0 (default) | integer from 0 to 31

Modulation and coding scheme to use for transmitting the current packet, specified as an integer from 0 to 31. The MCS setting identifies which modulation and coding rate combination is used, and the number of spatial streams (N_{SS}).

MCS(Note 1)	N_{SS}(Note 1)	Modulation	Coding Rate
0, 8, 16, or 24	1, 2, 3, or 4	BPSK	1/2
1, 9, 17, or 25	1, 2, 3, or 4	QPSK	1/2
2, 10, 18, or 26	1, 2, 3, or 4	QPSK	3/4
3, 11, 19, or 27	1, 2, 3, or 4	16QAM	1/2
4, 12, 20, or 28	1, 2, 3, or 4	16QAM	3/4
5, 13, 21, or 29	1, 2, 3, or 4	64QAM	2/3
6, 14, 22, or 30	1, 2, 3, or 4	64QAM	3/4
7, 15, 23, or 31	1, 2, 3, or 4	64QAM	5/6

MCS[Note 1]	N_{SS}[Note 1]	Modulation	Coding Rate
Note-1 MCS from 0 to 7 have one spatial stream. MCS from 8 to 15 have two spatial streams. MCS from 16 to 23 have three spatial streams. MCS from 24 to 31 have four spatial streams.			

See IEEE 802.11-2012, Section 20.6 for further description of MCS dependent parameters.

When working with the HT-Data field, if the number of space-time streams is equal to the number of spatial streams, no space-time block coding (STBC) is used. See IEEE 802.11-2012, Section 20.3.11.9.2 for further description of STBC mapping.

Example: `22` indicates an MCS with three spatial streams, 64-QAM modulation, and a 3/4 coding rate.

Data Types: `double`

`span` — Filter span

positive odd integer

Filter span of the frequency smoothing filter, specified as an odd integer. The span is expressed as a number of subcarriers.

Note: If adjacent subcarriers are highly correlated, frequency smoothing will result in significant noise reduction. However, in a highly frequency selective channel, smoothing may degrade the quality of the channel estimate.

Data Types: `double`

Output Arguments

`chEst` — Channel estimate

3-D array

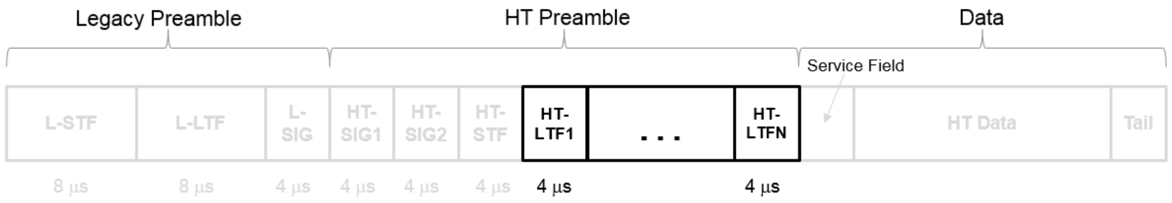
Channel estimate between all combinations of space-time streams and receive antennas, returned as an N_{ST}-by-(N_{STS}+N_{ESS})-by-N_R array. N_{ST} is the number of occupied subcarriers, N_{STS} is the number of space-time streams. N_{ESS} is the number of extension spatial streams. N_R is the number of receive antennas. Data and pilot subcarriers are included in the channel estimate.

Data Types: `double`
Complex Number Support: Yes

More About

HT-LTF

The high throughput long training field (HT-LTF) is located between the HT-STF and data field of an HT-mixed packet.



As described in IEEE Std 802.11-2012, Section 20.3.9.4.6, the receiver can use the HT-LTF to estimate the MIMO channel between the set of QAM mapper outputs (or, if STBC is applied, the STBC encoder outputs) and the receive chains. The HT-LTF portion has one or two parts. The first part consists of one, two, or four HT-LTFs that are necessary for demodulation of the HT-Data portion of the PPDU. These HT-LTFs are referred to as HT-DLTFs. The optional second part consists of zero, one, two, or four HT-LTFs that can be used to sound extra spatial dimensions of the MIMO channel not utilized by the HT-Data portion of the PPDU. These HT-LTFs are referred to as HT-ELTFs. Each HT long training symbol is 4 µs. The number of space-time streams and the number of extension streams determines the number of HT-LTF symbols transmitted.

Tables 20-12, 20-13 and 20-14 from IEEE Std 802.11-2012 are reproduced here.

N_{STS} Determination	N_{HTDLTF} Determination	N_{HTELTF} Determination
Table 20-12 defines the number of space-time streams (N_{STS}) based on the number of spatial streams (N_{SS}) from the MCS and the STBC field.	Table 20-13 defines the number of HT-DLTFs required for the N_{STS}.	Table 20-14 defines the number of HT-ELTFs required for the number of extension spatial streams (N_{ESS}). N_{ESS} is defined in HT-SIG$_2$.

N_{STS} Determination			N_{HTDLTF} Determination		N_{HTELTF} Determination	
N_{SS} from MCS	STBC field	N_{STS}	N_{STS}	N_{HTDLTF}	N_{ESS}	N_{HTELTF}
1	0	1	1	1	0	0
1	1	2	2	2	1	1
2	0	2	3	4	2	2
2	1	3	4	4	3	4
2	2	4				
3	0	3				
3	1	4				
4	0	4				

Additional constraints include:

- $N_{HTLTF} = N_{HTDLTF} + N_{HTELTF} \leq 5$.
- $N_{STS} + N_{ESS} \leq 4$.
  - When $N_{STS} = 3$, N_{ESS} cannot exceed one.
  - If $N_{ESS} = 1$ when $N_{STS} = 3$ then $N_{HTLTF} = 5$.

References

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

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

See Also

`wlanHTConfig` | `wlanHTLTF` | `wlanHTLTFDemodulate`

Introduced in R2015b

wlanVHTLTFChannelEstimate

Channel estimation using VHT-LTF

Syntax

```
chEst = wlanVHTLTFChannelEstimate(demodSig,cfg)
chEst = wlanVHTLTFChannelEstimate(demodSig,cfg,span)
```

Description

`chEst = wlanVHTLTFChannelEstimate(demodSig,cfg)` returns the channel estimate, using the demodulated "VHT-LTF" on page 1-43[5] signal, `demodSig`, given the parameters specified in `wlanVHTConfig` object `cfg`.

`chEst = wlanVHTLTFChannelEstimate(demodSig,cfg,span)` specifies the span of a moving-average filter used to perform frequency smoothing.

Examples

Estimate SISO Channel Using VHT-LTF

Display the channel estimate of the data and pilot subcarriers for a VHT format channel using its long training field.

Create a VHT format configuration object. Generate a VHT-LTF based on `cfg`.

```
cfg = wlanVHTConfig;
txSig = wlanVHTLTF(cfg);
```

Multiply the transmitted VHT-LTF signal by 0.3 - 0.15i and pass it through an AWGN channel having a 30 dB signal-to-noise ratio. Demodulate the received signal.

```
rxSig = awgn(txSig*(0.3-0.15i),30);
demodSig = wlanVHTLTFDemodulate(rxSig,cfg);
```

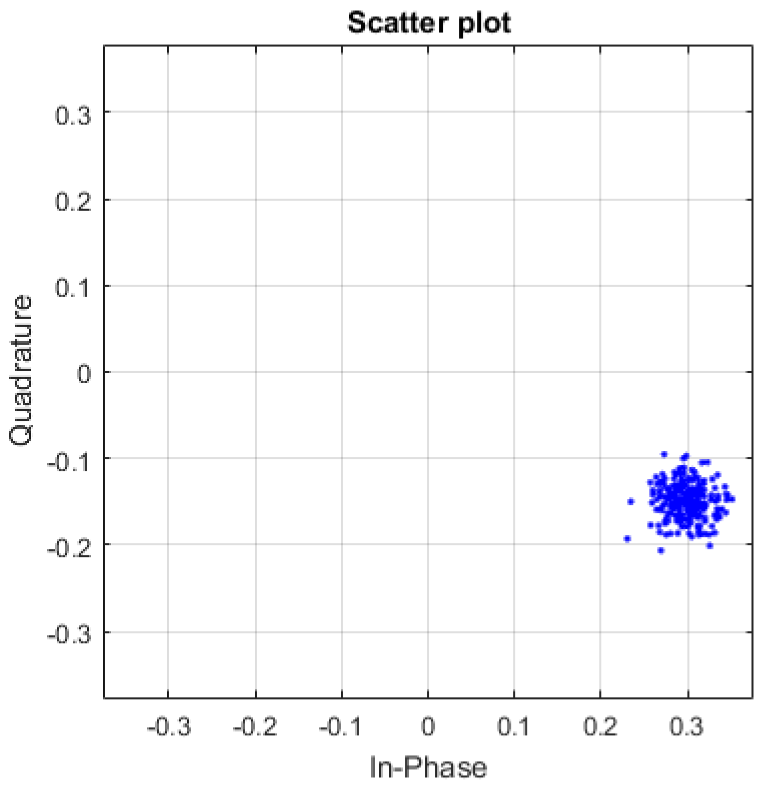
5. IEEE Std 802.11ac-2013 Adapted and reprinted with permission from IEEE. Copyright IEEE 2013. All rights reserved.

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

```
est = wlanVHTLTFChannelEstimate(demodSig,cfg);
```

Plot the channel estimate.

```
scatterplot(est)
grid
```



The channel estimate matches the complex channel multiplier.

Estimate MIMO Channel Using VHT-LTF

Estimate and display the channel coefficients of a 4x2 MIMO channel using the VHT-LTF.

Create a VHT format configuration object for a channel having four spatial streams and four transmit antennas. Create the corresponding generator configuration object. Transmit a complete VHT waveform.

```
cfg = wlanVHTConfig('NumTransmitAntennas',4, ...
    'NumSpaceTimeStreams',4,'MCS',5);
wgc = wlanGeneratorConfig;
txWaveform = wlanWaveformGenerator([1;0;0;1;1;0],cfg,wgc);
```

Set the sampling rate, and then pass the transmitted waveform through a 4x2 TGac channel.

```
fs = 80e6;
ch = wlanTGacChannel('SampleRate',fs, ...
    'NumTransmitAntennas',4,'NumReceiveAntennas',2);
rxWaveform = step(ch,txWaveform);
```

Determine the VHT-LTF field indices and demodulate the VHT-LTF from the received waveform.

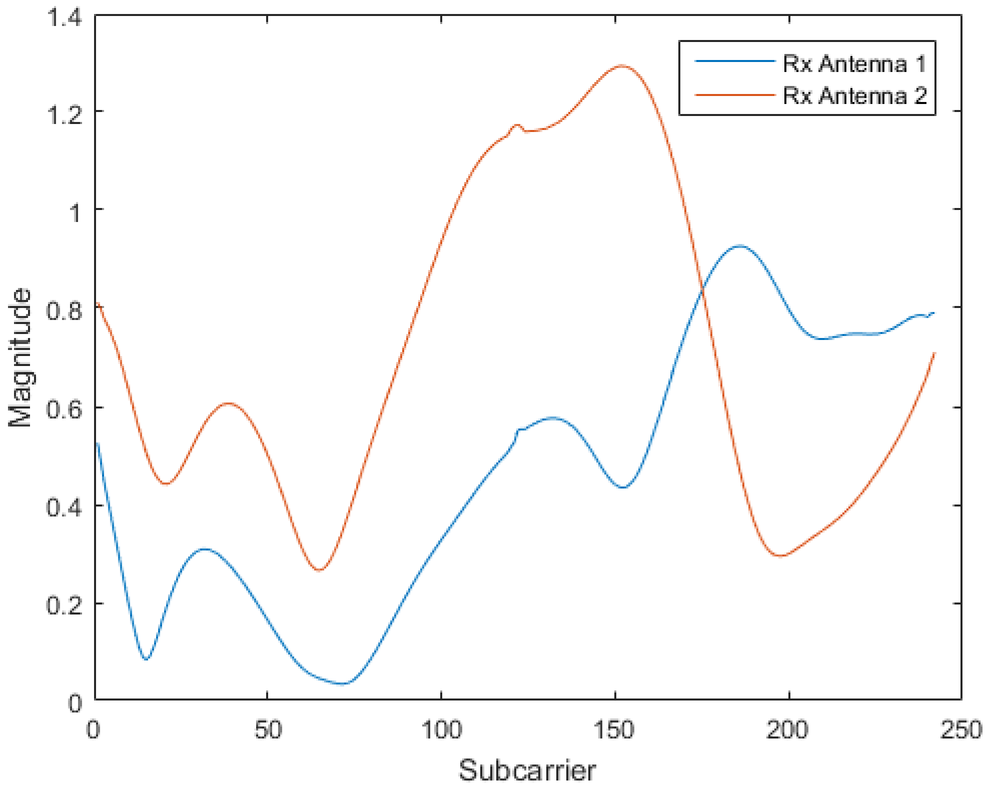
```
indVHTLTF = wlanFieldIndices(cfg,'VHT-LTF');
ltfDemodSig = wlanVHTLTFDemodulate(rxWaveform(indVHTLTF(1):indVHTLTF(2),:), cfg);
```

Generate the channel estimate by using the demodulated VHT-LTF signal. Specify a smoothing filter span of five subcarriers.

```
est = wlanVHTLTFChannelEstimate(ltfDemodSig,cfg,5);
```

Plot the magnitude response of the first space-time stream for both receive antennas. Due to the random nature of the fading channel, your results may vary.

```
plot(abs(est(:,1,1)))
hold on
plot(abs(est(:,1,2)))
xlabel('Subcarrier')
ylabel('Magnitude')
legend('Rx Antenna 1','Rx Antenna 2')
```



Input Arguments

`demodSig` — Demodulated VHT-LTF signal
3-D array

Demodulated VHT-LTF signal, specified as an N_{ST}-by-N_{SYM}-by-N_R array. N_{ST} is the number of occupied subcarriers, N_{SYM} is the number of VHT-LTF OFDM symbols, and N_R is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

`cfg` — Format configuration
`wlanVHTConfig`

Format configuration, specified as a `wlanVHTConfig` object.

`span` — Filter span
positive odd integer

Filter span of the frequency smoothing filter, specified as an odd integer. The span is expressed as a number of subcarriers.

Note: If adjacent subcarriers are highly correlated, frequency smoothing results in significant noise reduction. However, in a highly frequency-selective channel, smoothing can degrade the quality of the channel estimate.

Data Types: `double`

Output Arguments

`chEst` — Channel estimate
3-D array

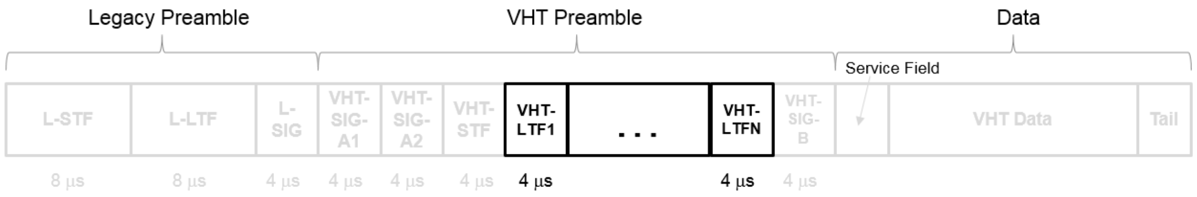
Channel estimate between all combinations of space-time streams and receive antennas, returned as an N_{ST}-by-N_{STS}-by-N_R array. N_{ST} is the number of occupied subcarriers. N_{STS} is the number of space-time streams. N_R is the number of receive antennas. The channel estimate includes coefficients for both the data and pilot subcarriers.

Data Types: `double`
Complex Number Support: Yes

More About

VHT-LTF

The very high throughput long training field (VHT-LTF) is located between the VHT-STF and VHT-SIG-B portion of the VHT packet.



It is used for MIMO channel estimation and pilot subcarrier tracking. The VHT-LTF includes one VHT long training symbol for each spatial stream indicated by the selected MCS. Each symbol is 4 μs long. A maximum of eight symbols are permitted in the VHT-LTF.

The VHT-LTF is defined in IEEE Std 802.11ac-2013, Section 22.3.8.3.5.

References

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

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

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

See Also

`wlanVHTConfig` | `wlanVHTDataRecover` | `wlanVHTLTFDemodulate`

Introduced in R2015b

wlanEqualize

Perform MIMO channel equalization

Syntax

```
[Y, CSI] = wlanEqualize(X,CHANEST, 'ZF')
[Y, CSI] = wlanEqualize(X,CHANEST, 'MMSE',NOISEVAR)
```

Description

`[Y, CSI] = wlanEqualize(X,CHANEST, 'ZF')` performs equalization using the signal input X and the channel estimation input CHANEST, and returns the estimation of transmitted signal in Y and the soft channel state information in CSI. The zero-forcing (ZF) method is used. The inputs X and CHANEST can be double precision 2-D matrices or 3-D arrays with real or complex values. X is of size Nsd x Nsym x Nr, where Nsd represents the number of data subcarriers (frequency domain), Nsym represents the number of OFDM symbols (time domain), and Nr represents the number of receive antennas (spatial domain). CHANEST is of size Nsd x Nsts x Nr, where Nsts represents the number of space-time streams. The double precision output Y is of size Nsd x Nsym x Nsts. Y is complex when either X or CHANEST is complex and is real otherwise. The double precision output CSI is of size Nsd x Nsts.

`[Y, CSI] = wlanEqualize(X,CHANEST, 'MMSE',NOISEVAR)` performs the equalization using the minimum-mean-square-error (MMSE) method. The noise variance input NOISEVAR is a double precision, nonnegative, scalar or row vector of length Nr.

Input Arguments

`X` —
(default) |

Example:

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `char` | `struct` | `table` | `cell` | `function_handle`

Complex Number Support: Yes

CHANEST —
(default) |

Example:

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `char` | `struct` | `table` | `cell` | `function_handle`
Complex Number Support: Yes

NOISEVAR —
(default) |

Example:

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `char` | `struct` | `table` | `cell` | `function_handle`
Complex Number Support: Yes

Output Arguments

Y —

CSI —

See Also
`wlanSTBCCombine`

wlanFieldIndices

Generate PPDU field indices

Syntax

```
ind = wlanFieldIndices(cfg)
ind = wlanFieldIndices(cfg,field)
```

Description

`ind = wlanFieldIndices(cfg)` returns a structure, `ind`, containing the start and stop indices of the individual component fields that comprise the PPDU. The format is specified by configuration object `cfg`.

`ind = wlanFieldIndices(cfg,field)` returns a row vector containing the start and stop indices for the specified field type.

Examples

Extract PPDU Fields From VHT Waveform

Extract the VHT-STF from a VHT waveform.

Create VHT configuration object for a MIMO transmission using a 160 MHz channel bandwidth.

```
cfg = wlanVHTConfig('MCS',8,'ChannelBandwidth','CBW160','NumTransmitAntennas',2,'NumSpa
```

Configure the waveform generator and generate the corresponding VHT waveform.

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

Determine the component PPDU field indices for the VHT format.

```
ind = wlanFieldIndices(cfg)
ind = 

       LSTF: [1 1280]
       LLTF: [1281 2560]
       LSIG: [2561 3200]
    VHTSIGA: [3201 4480]
     VHTSTF: [4481 5120]
     VHTLTF: [5121 6400]
    VHTSIGB: [6401 7040]
    VHTData: [7041 8320]
```

The VHT PPDU waveform is comprised of eight fields, including seven preamble fields and one data field.

Extract the VHT-STF from the transmitted waveform.

```
stf = txSig(ind.VHTSTF(1):ind.VHTSTF(2),:);
```

Verify that the VHT-STF has dimensions of 640-by-2 corresponding to the number of samples (80 for each 20 MHz bandwidth segment) and the number of transmit antennas.

```
size(stf)
```

```
ans =

   640     2
```

Extract VHT-LTF and Recover VHT Data

Generate a VHT waveform. Extract and demodulate the VHT-LTF to estimate the channel coefficients. Recover the data field using the channel estimate and use this to determine the number of bit errors.

Configure a VHT channel and a waveform generator.

```
vht = wlanVHTConfig('NumTransmitAntennas',2,'NumSpaceTimeStreams',2);
wgc = wlanGeneratorConfig;
```

Generate a random PSDU and create the corresponding VHT waveform.

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

Pass the signal through a TGac 2x2 MIMO channel.

```
tgac = wlanTGacChannel('NumTransmitAntennas',2,'NumReceiveAntennas',2, ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
rxSigNoNoise = step(tgac,txSig);
```

Add AWGN to the received signal. Set the noise variance for the case in which the receiver has a 9 dB noise figure.

```
nVar = 10^((-228.6+10*log10(290)+10*log10(80e6)+9)/10);
chAWGN = comm.AWGNChannel('NoiseMethod','Variance','Variance',nVar);
rxSig = step(chAWGN,rxSigNoNoise);
```

Determine the indices for the VHT-LTF and extract the field from the received signal.

```
indVHT = wlanFieldIndices(vht,'VHT-LTF');
rxLTF = rxSig(indVHT(1):indVHT(2),:);
```

Demodulate the VHT-LTF and estimate the channel coefficients.

```
dLTF = wlanVHTLTFDemodulate(rxLTF,vht);
chEst = wlanVHTLTFChannelEstimate(dLTF,vht);
```

Extract the data field and recover the information bits.

```
indData = wlanFieldIndices(vht,'VHT-Data');
rxData = rxSig(indData(1):indData(2),:);
rxPSDU = wlanVHTDataRecover(rxData,chEst,nVar,vht);
```

Determine the number of bit errors.

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

```
numErrs =

     0
```

Input Arguments

`cfg` — Transmission format

`wlanVHTConfig` | `wlanHTConfig` | `wlanNonHTConfig`

Transmission format , specified as a `wlanVHTConfig`, `wlanHTConfig`, or `wlanNonHTConfig` configuration object.

Example: `txformat` = `wlanVHTConfig`

`field` — PPDU field
string

PPDU field, specified as one of these strings: `'L-STF'`, `'L-LTF'`, `'L-SIG'`, `'HT-SIG'`, `'VHT-SIG-A'`, `'HT-STF'`, `'VHT-STF'`, `'HT-LTF'`, `'VHT-LTF'`, `'VHT-SIG-B'`, `'NonHT-Data'`, `'HT-Data'`, or `'VHT-Data'`.

- `'VHT-SIG-A'`, `'VHT-STF'`, `'VHT-LTF'`, `'VHT-SIG-B'`, and `'VHT-Data'` are valid for the VHT format only.
- `'HT-SIG'`, `'HT-STF'`, `'HT-LTF'`, and `'HT-Data'` are valid for the HT format only.
- `'NonHT-Data'` is valid for the non-HT format only.
- `'L-STF'`, `'L-LTF'`, and `'L-SIG'` are common to all formats.

Data Types: `char`

Output Arguments

`ind` — Start and stop indices
structure | vector

Start and stop indices of all PPDU fields, returned as a structure. If you specify `field`, the function returns `ind` as a 1-by-2 vector consisting of the start and stop indices of the PPDU field.

Note: In null data packet (NDP) mode, `ind` is returned as an empty matrix. NDP mode occurs when either of these conditions are met:

- For VHT, `APEPLength` = 0.
- For HT, `PSDULength` = 0.

Data Types: `struct` | `double`

References

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

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

See Also

`wlanGeneratorConfig` | `wlanHTConfig` | `wlanNonHTConfig` | `wlanVHTConfig` | `wlanWaveformGenerator`

Introduced in R2015b

wlanGeneratorConfig

Create waveform generator configuration object

Syntax

```
cfgWaveGen = wlanGeneratorConfig
cfgWaveGen = wlanGeneratorConfig(Name,Value)
```

Description

`cfgWaveGen = wlanGeneratorConfig` creates a waveform generator configuration object. Use an instance of this object to configure the `wlanWaveformGenerator` function.

`cfgWaveGen = wlanGeneratorConfig(Name,Value)` creates a waveform generator configuration object that overrides default values using one or more `Name,Value` pair arguments.

At runtime, the calling function validates object settings for properties relevant to the operation of the function.

Examples

Generate VHT Waveform with Random Scrambler State

Configure `wlanWaveformGenerator` to produce a time-domain signal for an 802.11ac VHT transmission with five packets and a 30 microsecond idle period between packet. Use a random scrambler initial state for each packet.

Create a generator configuration object. Update the default settings of the object to generate five packets with a 30 microsecond idle period between each packet.

```
wgc = wlanGeneratorConfig;
wgc.NumPackets = 5;
wgc.IdleTime = 30e-6;
```

Use a random scrambler initial state for each packet.

```
wgc.ScramblerInitialization = randi([1 127],wgc.NumPackets,1);
```

Create a VHT configuration object and confirm the channel bandwidth for scaling the *x*-axis of the plot.

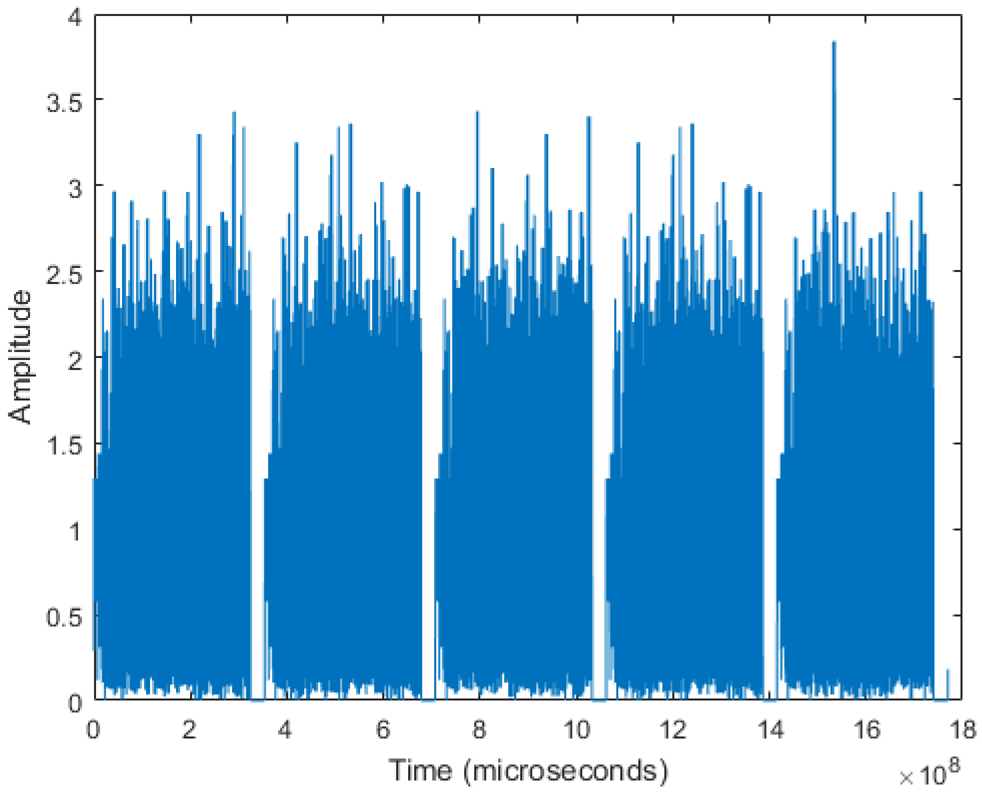
```
vht = wlanVHTConfig;
vht.ChannelBandwidth
```

```
ans =

CBW80
```

Generate and plot the waveform. Display the time in microseconds on the *x*-axis.

```
txWaveform = wlanWaveformGenerator([1;0;0;1],vht,wgc);
time = [0:length(txWaveform)-1]/80e-6;
plot(time,abs(txWaveform))
xlabel ('Time (microseconds)');
ylabel('Amplitude');
```



Five packets separated by 30 microsecond idle periods.

Generate Non-HT Waveform

Configure `wlanWaveformGenerator` to produce a waveform containing a single 802.11a packet without any windowing.

Create a generator configuration object and disable windowing.

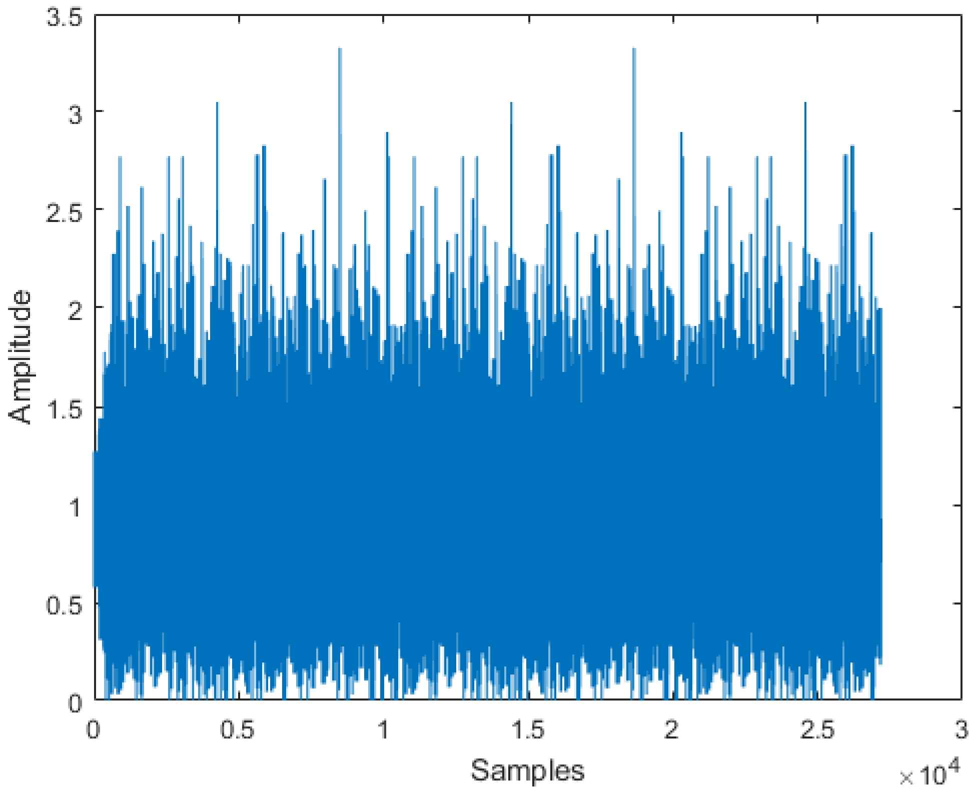
```
cfgWaveGen = wlanGeneratorConfig('Windowing',false);
```

Create a non-HT format configuration object.

```
cfgNonHT = wlanNonHTConfig;
```

Generate and plot the waveform.

```
txWaveform = wlanWaveformGenerator([1;0;0;1],cfgNonHT,cfgWaveGen);
plot(abs(txWaveform));
xlabel('Samples');
ylabel('Amplitude');
```



Input Arguments

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'NumPackets',21,'ScramblerInitialization',[52,17]`

`'NumPackets'` — Number of packets
1 (default) | positive integer

Number of packets to generate in a single function call, specified as a positive integer.

Data Types: `double`

`'IdleTime'` — Idle time added after each packet
0 (default) | nonnegative scalar

Idle time added after each packet, specified as a nonnegative scalar in seconds. If `IdleTime` is greater than the default value of zero, it cannot be less than 2 µs.

Example: `20e-6`

Data Types: `double`

`'ScramblerInitialization'` — Initial scrambler state
93 (default) | integer from 1 to 127 | matrix

Initial scrambler state of the data scrambler for each packet generated, specified as an integer from 1 to 127, or as an N_P-*by*-N_{Users} matrix of integers with values from 1 to 127. N_P is the number of packets, and N_{Users} is the number of users. The default value of 93 is the example state given in IEEE Std 802.11-2012, Section L.1.5.2.

- When specified as a scalar, the same scrambler initialization value is used to generate each packet for each user of a multipacket waveform.
- When specified as a matrix, each element represents an initial state of the scrambler for packets in the multipacket waveform generated for each user. Each column specifies the initial states for a single user, therefore up to four columns are supported. If a single column is provided, the same initial states are used for all users. Each row represents the initial state of each packet to generate. Therefore, a matrix with multiple rows enables you to use a different initial state per packet, where the first row contains the initial state of the first packet. If the number of packets to generate exceeds the number of rows of the matrix provided, the rows are looped internally.

The waveform generator configuration object does not validate the initial state of the scrambler.

Note: `ScramblerInitialization` applies to OFDM-based formats only.

Example: `[3 56 120]`

Data Types: `double` | `int8`

`'Windowing'` — Option to disable windowing
`true` (default) | `false`

Option to disable windowing between consecutive OFDM symbols, specified as a logical. Windowing is the process in which the OFDM symbol is multiplied by a cosine function before transmission to reduce adjacent channel power.

Data Types: `logical`

`'WindowTransitionTime'` — Duration of the window transition
1.0e-07 (default) | positive scalar less than or equal to 1.6e-06

Duration of the window transition applied to each OFDM symbol, specified in seconds as a positive scalar less than or equal to 1.6e-06. For a transition time of zero, no windowing is applied.

The maximum permitted `WindowTransitionTime` value depends on the type of guard interval:

- For a long guard interval, the maximum permitted setting is 1.6e-06 seconds.
- For a short guard interval, the maximum permitted setting is 8.0e-07 seconds.

Data Types: `double`

Output Arguments

`cfgWaveGen` — Waveform generator configuration
`wlanGeneratorConfig` object

Waveform generator configuration, returned as a `wlanGeneratorConfig` object. The properties of `cfgWaveGen` are specified in wlanGeneratorConfig Properties.

See Also

`wlanWaveformGenerator`

Introduced in R2015b

wlanHTConfig

Create HT format configuration object

Syntax

```
cfgHT = wlanHTConfig
cfgHT = wlanHTConfig(Name,Value)
```

Description

`cfgHT = wlanHTConfig` creates a configuration object that initializes parameters for an IEEE 802.11 high throughput (HT) format “PPDU” on page 1-64.

`cfgHT = wlanHTConfig(Name,Value)` creates an HT format configuration object that overrides the default settings using one or more `Name,Value` pair arguments.

At runtime, the calling function validates object settings for properties relevant to the operation of the function.

Examples

Create HT Configuration Object with Default Settings

Create an HT configuration object with default settings.

```
cfgHT = wlanHTConfig
```

```
cfgHT = 

  wlanHTConfig with properties:

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

Update the number of antennas to two, and number of space-time streams to four.

```
cfgHT.NumTransmitAntennas = 2;
cfgHT.NumSpaceTimeStreams = 4
```

```
cfgHT = 

  wlanHTConfig with properties:

       ChannelBandwidth: 'CBW20'
    NumTransmitAntennas: 2
    NumSpaceTimeStreams: 4
         SpatialMapping: 'Direct'
                    MCS: 0
          GuardInterval: 'Long'
          ChannelCoding: 'BCC'
             PSDULength: 1024
     RecommendSmoothing: true
```

Create wlanHTConfig Object

Create a `wlanHTConfig` object for single user operation with a PSDU length of 2048 bytes, and using BCC forward error correction.

```
cfgHT = wlanHTConfig('PSDULength',2048);
cfgHT.ChannelBandwidth = 'CBW20'
```

```
cfgHT = 

  wlanHTConfig with properties:

       ChannelBandwidth: 'CBW20'
    NumTransmitAntennas: 1
    NumSpaceTimeStreams: 1
         SpatialMapping: 'Direct'
                    MCS: 0
          GuardInterval: 'Long'
          ChannelCoding: 'BCC'
          PSDULength: 2048
   RecommendSmoothing: true
```

Input Arguments

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'ChannelBandwidth','CBW40','NumTransmitAntennas',2`

`'ChannelBandwidth'` — Channel bandwidth
`'CBW20'` (default) | `'CBW40'`

Channel bandwidth in MHz, specified as `'CBW20'` or `'CBW40'`.

Data Types: `char`

`'NumTransmitAntennas'` — Number of transmit antennas
1 (default) | 2 | 3 | 4

Number of transmit antennas, specified as 1, 2, 3, or 4.

Data Types: `double`

`'NumSpaceTimeStreams'` — Number of space-time streams
1 (default) | 2 | 3 | 4

Number of space-time streams in the transmission, specified as 1, 2, 3, or 4.

Data Types: `double`

`'NumExtensionStreams'` — Number of extension spatial streams
0 (default) | 1 | 2 | 3

Number of extension spatial streams in the transmission, specified as 0, 1, 2, or 3. When `NumExtensionStreams` is greater than 0, `SpatialMapping` must be `'Custom'`.

Data Types: `double`

'SpatialMapping' — Spatial mapping scheme

'Direct' (default) | 'Hadamard' | 'Fourier' | 'Custom'

Spatial mapping scheme, specified as 'Direct', 'Hadamard', 'Fourier', or 'Custom'. The default value 'Direct', applies when NumTransmitAntennas and NumSpaceTimeStreams are equal.

Data Types: char

'SpatialMappingMatrix' — Spatial mapping matrix

1 (default) | scalar | matrix | 3-D array

Spatial mapping matrix, specified as a scalar, matrix, or 3-D array. Use this property to rotate and scale the constellation mapper output vector. This property applies when the SpatialMapping property is set to 'Custom'. The spatial mapping matrix is used for beamforming and mixing space-time streams over the transmit antennas.

- When specified as a scalar, NumTransmitAntennas = NumSpaceTimeStreams = 1 and a constant value applies to all the subcarriers.
- When specified as a matrix, the size must be (N_{STS} + N_{ESS})-by-N_T. N_{STS} is the number of space-time streams. N_{ESS} is the number of extension spatial streams. N_T is the number of transmit antennas. The spatial mapping matrix applies to all the subcarriers. The first N_{STS} and last N_{ESS} rows apply to the space-time streams and extension spatial streams respectively.
- When specified as a 3-D array, the size must be N_{ST}-by-(N_{STS} + N_{ESS})-by-N_T. N_{ST} is the sum of the data and pilot subcarriers, as determined by ChannelBandwidth. N_{STS} is the number of space-time streams. N_{ESS} is the number of extension spatial streams. N_T is the number of transmit antennas. In this case, each data and pilot subcarrier can have its own spatial mapping matrix.

  The table shows the ChannelBandwidth setting and the corresponding N_{ST}.

ChannelBandwidth	N_{ST}
'CBW20'	56
'CBW40'	114

The calling function normalizes the spatial mapping matrix for each subcarrier.

Example: [0.5 0.3; 0.4 0.4; 0.5 0.8] represents a spatial mapping matrix having three space-time streams and two transmit antennas.

Data Types: `double`
Complex Number Support: Yes

`'MCS'` — Modulation and coding scheme

0 (default) | integer from 0 to 31

Modulation and coding scheme to use for transmitting the current packet, specified as an integer from 0 to 31. The MCS setting identifies which modulation and coding rate combination is used, and the number of spatial streams (N_{SS}).

MCS(Note 1)	N_{SS}(Note 1)	Modulation	Coding Rate
0, 8, 16, or 24	1, 2, 3, or 4	`BPSK`	`1/2`
1, 9, 17, or 25	1, 2, 3, or 4	`QPSK`	`1/2`
2, 10, 18, or 26	1, 2, 3, or 4	`QPSK`	`3/4`
3, 11, 19, or 27	1, 2, 3, or 4	`16QAM`	`1/2`
4, 12, 20, or 28	1, 2, 3, or 4	`16QAM`	`3/4`
5, 13, 21, or 29	1, 2, 3, or 4	`64QAM`	`2/3`
6, 14, 22, or 30	1, 2, 3, or 4	`64QAM`	`3/4`
7, 15, 23, or 31	1, 2, 3, or 4	`64QAM`	`5/6`
Note-1 MCS from 0 to 7 have one spatial stream. MCS from 8 to 15 have two spatial streams. MCS from 16 to 23 have three spatial streams. MCS from 24 to 31 have four spatial streams.			

See IEEE 802.11-2012, Section 20.6 for further description of MCS dependent parameters.

When working with the HT-Data field, if the number of space-time streams is equal to the number of spatial streams, no space-time block coding (STBC) is used. See IEEE 802.11-2012, Section 20.3.11.9.2 for further description of STBC mapping.

Example: `22` indicates an MCS with three spatial streams, 64-QAM modulation, and a 3/4 coding rate.

Data Types: `double`

`'GuardInterval'` — Cyclic prefix length for the data field within a packet

`'Long'` (default) | `'Short'`

Cyclic prefix length for the data field within a packet, specified as `'Long'` or `'Short'`.

- The long guard interval length is 800 ns.
- The short guard interval length is 400 ns.

Data Types: `char`

`'PSDULength'` — Number of bytes carried in the user payload
1024 (default) | integer from 0 to 65,535

Number of bytes carried in the user payload, specified as an integer from 0 to 65,535. A `PSDULength` of 0 implies a sounding packet for which there are no data bits to recover.

Example: `512`

Data Types: `double`

`'RecommendSmoothing'` — Indication of whether frequency-domain smoothing is recommended
true (default) | false

Indication of whether frequency-domain smoothing is recommended as part of channel estimation, specified as a logical. If the frequency profile across the channel is nonvarying, the receiver sets this property to `true`.

Data Types: `logical`

Output Arguments

`cfgHT` — HT PPDU configuration
`wlanHTConfig` object

HT “PPDU” on page 1-64 configuration, returned as a `wlanHTConfig` object. The properties of `cfgHT` are described in wlanHTConfig Properties.

More About

PPDU

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

References

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

See Also

`wlanHTDataRecover` | `wlanNonHTConfig` | `wlanVHTConfig` | `wlanWaveformGenerator`

Introduced in R2015b

wlanHTData

Generate HT data field waveform

Syntax

```
y = wlanHTData(psdu,cfg)
y = wlanHTData(psdu,cfg,scramInit)
```

Description

`y = wlanHTData(psdu,cfg)` generates the “HT-Data field” on page 1-71[6] time-domain waveform for the input PLCP service data unit, `psdu`, and specified configuration object, `cfg`. See “HT-Data Field Processing” on page 1-72 for waveform generation details.

`y = wlanHTData(psdu,cfg,scramInit)` uses `scramInit` for the scrambler initialization state.

Examples

Generate HT-Data Waveform

Generate the waveform signal for a 40 MHz HT-mixed data field with multiple transmit antennas. Create an HT format configuration object. Specify 40 MHz channel bandwidth, two transmit antennas, and two space-time streams.

```
cfgHT = wlanHTConfig('ChannelBandwidth','CBW40','NumTransmitAntennas',2,'NumSpaceTimeSt
```

```
cfgHT = 

  wlanHTConfig with properties:

       ChannelBandwidth: 'CBW40'
```

6. IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.

```
    NumTransmitAntennas: 2
    NumSpaceTimeStreams: 2
         SpatialMapping: 'Direct'
                    MCS: 12
          GuardInterval: 'Long'
          ChannelCoding: 'BCC'
             PSDULength: 1024
     RecommendSmoothing: true
```

Assign `PSDULength` bytes of random data to a bit stream and generate the HT data waveform.

```
PSDU =  randi([0 1],cfgHT.PSDULength*8,1);
y = wlanHTData(PSDU,cfgHT);
```

Determine the size of the waveform.

```
size(y)
```

```
ans =

        2080           2
```

The function returns a complex two-column time-domain waveform. Each column contains 2080 samples, corresponding to the HT-Data field for each transmit antenna.

Input Arguments

`psdu` — PLCP Service Data Unit

vector

PLCP Service Data Unit (“PSDU” on page 1-72), specified as an N_b-by-1 vector. N_b is the number of bits and equals `PSDULength` × 8.

Data Types: `double`

`cfg` — Format configuration

`wlanHTConfig` object

Format configuration, specified as a `wlanHTConfig` object. The `wlanHTData` function uses the object properties indicated.

`ChannelBandwidth` — Channel bandwidth
`'CBW20'` (default) | `'CBW40'`

Channel bandwidth in MHz, specified as `'CBW20'` or `'CBW40'`.

Data Types: `char`

`NumTransmitAntennas` — Number of transmit antennas
1 (default) | 2 | 3 | 4

Number of transmit antennas, specified as 1, 2, 3, or 4.

Data Types: `double`

`NumSpaceTimeStreams` — Number of space-time streams
1 (default) | 2 | 3 | 4

Number of space-time streams in the transmission, specified as 1, 2, 3, or 4.

Data Types: `double`

`NumExtensionStreams` — Number of extension spatial streams
0 (default) | 1 | 2 | 3

Number of extension spatial streams in the transmission, specified as 0, 1, 2, or 3. When `NumExtensionStreams` is greater than 0, `SpatialMapping` must be `'Custom'`.

Data Types: `double`

`SpatialMapping` — Spatial mapping scheme
`'Direct'` (default) | `'Hadamard'` | `'Fourier'` | `'Custom'`

Spatial mapping scheme, specified as `'Direct'`, `'Hadamard'`, `'Fourier'`, or `'Custom'`. The default value `'Direct'`, applies when `NumTransmitAntennas` and `NumSpaceTimeStreams` are equal.

Data Types: `char`

`SpatialMappingMatrix` — Spatial mapping matrix
1 (default) | scalar | matrix | 3-D array

Spatial mapping matrix, specified as a scalar, matrix, or 3-D array. Use this property to rotate and scale the constellation mapper output vector. This property applies when the `SpatialMapping` property is set to `'Custom'`. The spatial mapping matrix is used for beamforming and mixing space-time streams over the transmit antennas.

- When specified as a scalar, `NumTransmitAntennas` = `NumSpaceTimeStreams` = 1 and a constant value applies to all the subcarriers.
- When specified as a matrix, the size must be (N_{STS} + N_{ESS})-by-N_T. N_{STS} is the number of space-time streams. N_{ESS} is the number of extension spatial streams. N_T is the number of transmit antennas. The spatial mapping matrix applies to all the subcarriers. The first N_{STS} and last N_{ESS} rows apply to the space-time streams and extension spatial streams respectively.
- When specified as a 3-D array, the size must be N_{ST}-by-(N_{STS} + N_{ESS})-by-N_T. N_{ST} is the sum of the data and pilot subcarriers, as determined by `ChannelBandwidth`. N_{STS} is the number of space-time streams. N_{ESS} is the number of extension spatial streams. N_T is the number of transmit antennas. In this case, each data and pilot subcarrier can have its own spatial mapping matrix.

  The table shows the `ChannelBandwidth` setting and the corresponding N_{ST}.

`ChannelBandwidth`	N_{ST}
`'CBW20'`	56
`'CBW40'`	114

The calling function normalizes the spatial mapping matrix for each subcarrier.

Example: `[0.5 0.3; 0.4 0.4; 0.5 0.8]` represents a spatial mapping matrix having three space-time streams and two transmit antennas.

Data Types: `double`
Complex Number Support: Yes

`MCS` — Modulation and coding scheme

0 (default) | integer from 0 to 31

Modulation and coding scheme to use for transmitting the current packet, specified as an integer from 0 to 31. The MCS setting identifies which modulation and coding rate combination is used, and the number of spatial streams (N_{SS}).

MCS(Note 1)	N_{SS}(Note 1)	Modulation	Coding Rate
0, 8, 16, or 24	1, 2, 3, or 4	BPSK	1/2
1, 9, 17, or 25	1, 2, 3, or 4	QPSK	1/2
2, 10, 18, or 26	1, 2, 3, or 4	QPSK	3/4

MCS(Note 1)	N_{SS}(Note 1)	Modulation	Coding Rate
3, 11, 19, or 27	1, 2, 3, or 4	`16QAM`	`1/2`
4, 12, 20, or 28	1, 2, 3, or 4	`16QAM`	`3/4`
5, 13, 21, or 29	1, 2, 3, or 4	`64QAM`	`2/3`
6, 14, 22, or 30	1, 2, 3, or 4	`64QAM`	`3/4`
7, 15, 23, or 31	1, 2, 3, or 4	`64QAM`	`5/6`
Note-1 MCS from 0 to 7 have one spatial stream. MCS from 8 to 15 have two spatial streams. MCS from 16 to 23 have three spatial streams. MCS from 24 to 31 have four spatial streams.			

See IEEE 802.11-2012, Section 20.6 for further description of MCS dependent parameters.

When working with the HT-Data field, if the number of space-time streams is equal to the number of spatial streams, no space-time block coding (STBC) is used. See IEEE 802.11-2012, Section 20.3.11.9.2 for further description of STBC mapping.

Example: `22` indicates an MCS with three spatial streams, 64-QAM modulation, and a 3/4 coding rate.

Data Types: `double`

`GuardInterval` — Cyclic prefix length for the data field within a packet

`'Long'` (default) | `'Short'`

Cyclic prefix length for the data field within a packet, specified as `'Long'` or `'Short'`.

- The long guard interval length is 800 ns.
- The short guard interval length is 400 ns.

Data Types: `char`

`ChannelCoding` — Type of forward error correction coding

`'BCC'` (default)

This property is read only.

Type of forward error correction coding for the data field, specified as `'BCC'` to indicate binary convolutional coding. LDPC coding is not currently supported.

Data Types: `char`

`PSDULength` — Number of bytes carried in the user payload

1024 (default) | integer from 0 to 65,535

Number of bytes carried in the user payload, specified as an integer from 0 to 65,535. A `PSDULength` of 0 implies a sounding packet for which there are no data bits to recover.

Example: `512`

Data Types: `double`

`scramInit` — Scrambler initialization state

93 (default) | integer from 1 to 127 | binary column vector

Scrambler initialization state for each packet generated, specified as an integer from 1 to 127 or as the corresponding binary column vector of length seven. The default value of 93 is the example state given in IEEE Std 802.11-2012, Section L.1.5.2.

Example: `[1; 0; 1; 1; 1; 0; 1]` conveys the scrambler initialization state of 93 as a binary column vector.

Data Types: `double` | `int8`

Output Arguments

y — HT-Data field time-domain waveform

matrix

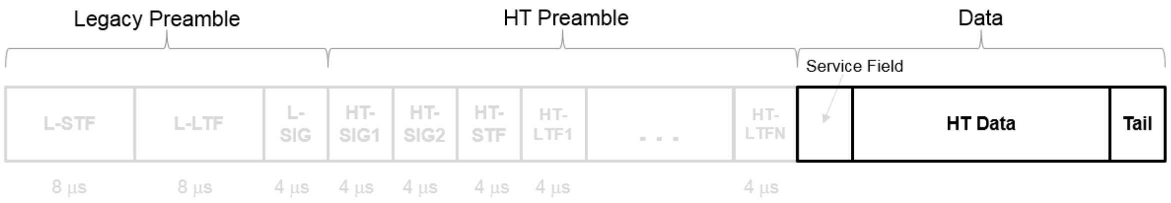
"HT-Data field" on page 1-71 time-domain waveform for HT-mixed format, returned as an N_S-by-N_T matrix. N_S is the number of time domain samples, and N_T is the number of transmit antennas.

Data Types: `double`
Complex Number Support: Yes

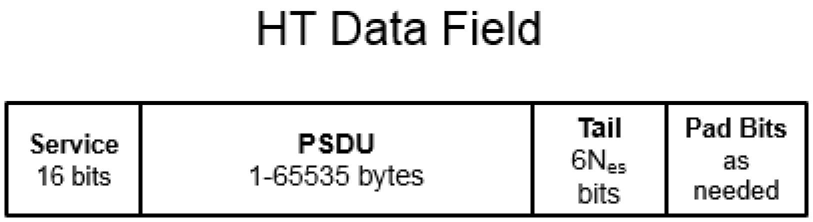
More About

HT-Data field

The high throughput data field (HT data) follows the last HT-LTF of an HT-mixed packet.



The high throughput data field is used to transmit one or more frames from the MAC layer and consists of four subfields.



- **Service field** — Contains 16 zeros to initialize the data scrambler.
- **PSDU** — Variable-length field containing the PLCP service data unit (PSDU). In 802.11, the PSDU can consist of an aggregate of several MAC service data units.
- **Tail** — Tail bits required to terminate a convolutional code. The field uses six zeros for each encoding stream.
- **Pad Bits** — Variable-length field required to ensure that the HT data field consists of an integer number of symbols.

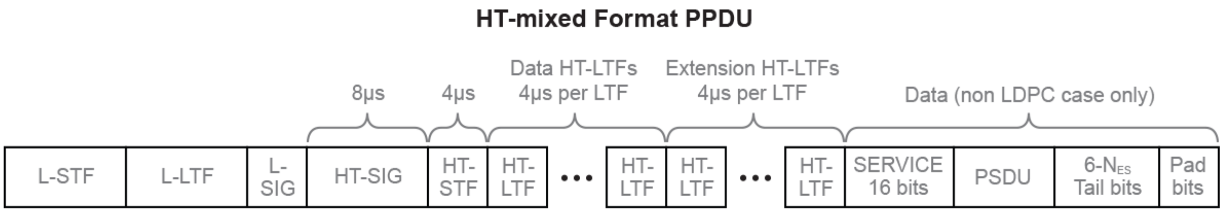
PSDU

Physical layer convergence procedure (PLCP) service data unit (PSDU). This field is composed of a variable number of octets. The minimum is 0 (zero) and the maximum is 2500. For more information see IEEE Std 802.11™-2012, Section 15.3.5.7.

Algorithms

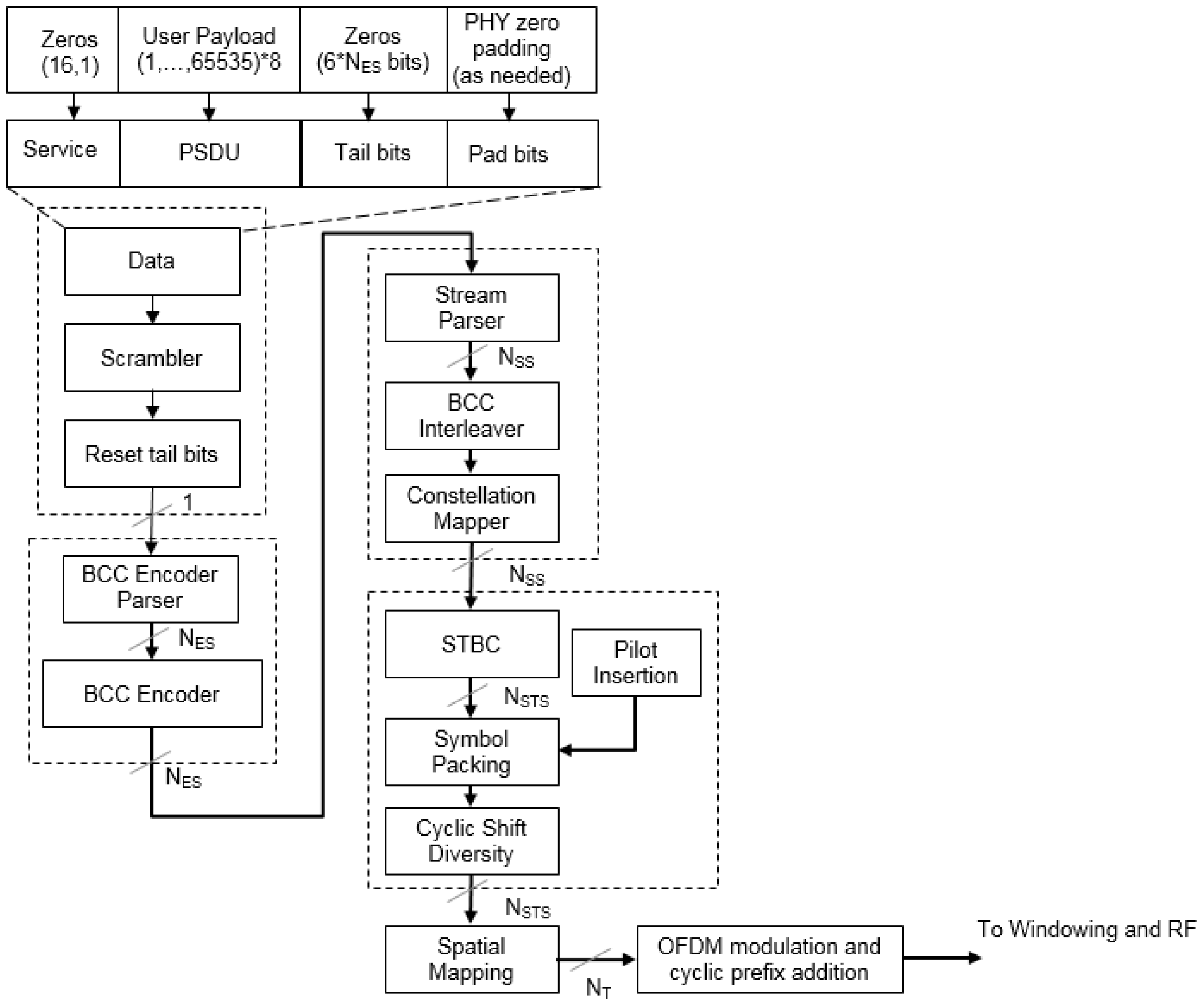
HT-Data Field Processing

The “HT-Data field” on page 1-71 follows the last HT-LTF in the packet structure.



The "HT-Data field" on page 1-71 includes the user payload in the PSDU, plus 16 service bits, 6 × N_{ES} tail bits, and additional padding bits as required to fill out the last OFDM symbol.

For algorithm details, refer to IEEE Std 802.11™-2012 [1], Section 20.3.11. The `wlanHTData` function performs transmitter processing on the "HT-Data field" on page 1-71 and outputs the time-domain waveform for N_T transmit antennas.



N_{ES} is the number of BCC encoders.
N_{SS} is the number of spatial streams.
N_{STS} is the number of space-time streams.
N_T is the number of transmit antennas.

BCC channel coding is shown. STBC and spatial mapping are optional modes for HT format.

References

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

See Also

`wlanHTConfig` | `wlanHTDataRecover` | `wlanHTLTF` | `wlanWaveformGenerator`

Introduced in R2015b

wlanHTDataRecover

Recover HT data

Syntax

```
recData = wlanHTDataRecover(rxSig,chEst,noiseVarEst,cfg)
recData = wlanHTDataRecover(rxSig,chEst,noiseVarEst,cfg,cfgRec)
[recData,eqSym] = wlanHTDataRecover( ___ )
[recData,eqSym,cpe] = wlanHTDataRecover( ___ )
```

Description

`recData = wlanHTDataRecover(rxSig,chEst,noiseVarEst,cfg)` returns the recovered “HT-Data field” on page 1-83[7], `recData`, for input signal `rxSig`. Specify a channel estimate for the occupied subcarriers, `chEst`, a noise variance estimate, `noiseVarEst`, and an “HT-Mixed” on page 1-83 format configuration object, `cfg`.

`recData = wlanHTDataRecover(rxSig,chEst,noiseVarEst,cfg,cfgRec)` specifies algorithm information using `wlanRecoveryConfig` object `cfgRec`.

`[recData,eqSym] = wlanHTDataRecover( ___ )` also returns the equalized symbols, `eqSym`, using the arguments from the previous syntaxes.

`[recData,eqSym,cpe] = wlanHTDataRecover( ___ )` also returns the common phase error, `cpe`.

Examples

Recover HT-Data Bits

Create an HT configuration object having a PSDU length of 1024 bytes. Generate an HTData sequence from a binary sequence whose length is eight times the length of the PSDU.

7. IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.

```
cfgHT = wlanHTConfig('PSDULength',1024);
txBits = randi([0 1],8*cfgHT.PSDULength,1);
txHTSig = wlanHTData(txBits,cfgHT);
```

Pass the signal through an AWGN channel with a signal-to-noise ratio of 10 dB.

```
rxHTSig = awgn(txHTSig,10);
```

Specify a channel estimate. Because fading was not introduced, a vector of ones is a perfect estimate. For a 20 MHz bandwidth, there are 52 data subcarriers and 4 pilot subcarriers in the HT-SIG field.

```
chEst = ones(56,1);
```

Recover the data bits and determine the number of bit errors. Display the number of bit errors and the associated bit error rate.

```
rxBits = wlanHTDataRecover(rxHTSig,chEst,0.1,cfgHT);
[numerr,ber] = biterr(rxBits,txBits)
```

```
numerr =

     0


ber =

     0
```

Recover HT-Data Field Signal Using Zero-Forcing Algorithm

Create an HT configuration object having a 40 MHz channel bandwidth and a 1024-byte PSDU length. Generate the corresponding HT-Data sequence.

```
cfgHT = wlanHTConfig('ChannelBandwidth','CBW40','PSDULength',1024);
txBits = randi([0 1],8*cfgHT.PSDULength,1);
txHTSig = wlanHTData(txBits, cfgHT);
```

Pass the signal through an AWGN channel with a signal-to-noise ratio of 7 dB.

```
rxHTSig = awgn(txHTSig,7);
```

Create a data recovery object that specifies the use of the zero-forcing algorithm.

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

Recover the data and determine the number of bit errors. Because fading was not introduced, the channel estimate is set to a vector of ones whose length is equal to the number of occupied subcarriers.

```
rxBits = wlanHTDataRecover(rxHTSig,ones(114,1),0.2,cfgHT,cfgRec);
[numerr,ber] = biterr(rxBits,txBits)
```

```
numerr =

     0


ber =

     0
```

Input Arguments

`rxSig` — Received HT-Data signal

vector | matrix

Received HT-Data signal, specified as an N_S-by-N_R vector or matrix. N_S is the number of samples, and N_R is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

`chEst` — Channel estimate

vector | matrix | 3-D array

Channel estimate, specified as an N_{ST}-by-N_{STS}-by-N_R array. N_{ST} is the number of occupied subcarriers, N_{STS} is the number of space-time streams, and N_R is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

`noiseVarEst` — Noise variance estimate

scalar

Noise variance estimate, specified as a nonnegative scalar.

Example: 0.7071

Data Types: `double`

`cfg` — Format configuration
`wlanHTConfig` object

Format configuration, specified as a `wlanHTConfig` object. The `wlanHTDataRecover` function uses the following `wlanHTConfig` object properties:

`ChannelBandwidth` — Channel bandwidth
`'CBW20'` (default) | `'CBW40'`

Channel bandwidth in MHz, specified as `'CBW20'` or `'CBW40'`.

Data Types: `char`

`NumSpaceTimeStreams` — Number of space-time streams
1 (default) | 2 | 3 | 4

Number of space-time streams in the transmission, specified as 1, 2, 3, or 4.

Data Types: `double`

`MCS` — Modulation and coding scheme
0 (default) | integer from 0 to 31

Modulation and coding scheme to use for transmitting the current packet, specified as an integer from 0 to 31. The MCS setting identifies which modulation and coding rate combination is used, and the number of spatial streams (N_{SS}).

MCS(Note 1)	N_{SS}(Note 1)	Modulation	Coding Rate
0, 8, 16, or 24	1, 2, 3, or 4	BPSK	1/2
1, 9, 17, or 25	1, 2, 3, or 4	QPSK	1/2
2, 10, 18, or 26	1, 2, 3, or 4	QPSK	3/4
3, 11, 19, or 27	1, 2, 3, or 4	16QAM	1/2
4, 12, 20, or 28	1, 2, 3, or 4	16QAM	3/4
5, 13, 21, or 29	1, 2, 3, or 4	64QAM	2/3

MCS[Note 1]	N_{SS}[Note 1]	Modulation	Coding Rate
6, 14, 22, or 30	1, 2, 3, or 4	64QAM	3/4
7, 15, 23, or 31	1, 2, 3, or 4	64QAM	5/6
Note-1 MCS from 0 to 7 have one spatial stream. MCS from 8 to 15 have two spatial streams. MCS from 16 to 23 have three spatial streams. MCS from 24 to 31 have four spatial streams.			

See IEEE 802.11-2012, Section 20.6 for further description of MCS dependent parameters.

When working with the HT-Data field, if the number of space-time streams is equal to the number of spatial streams, no space-time block coding (STBC) is used. See IEEE 802.11-2012, Section 20.3.11.9.2 for further description of STBC mapping.

Example: `22` indicates an MCS with three spatial streams, 64-QAM modulation, and a 3/4 coding rate.

Data Types: `double`

`GuardInterval` — Cyclic prefix length for the data field within a packet

`'Long'` (default) | `'Short'`

Cyclic prefix length for the data field within a packet, specified as `'Long'` or `'Short'`.

- The long guard interval length is 800 ns.
- The short guard interval length is 400 ns.

Data Types: `char`

`ChannelCoding` — Type of forward error correction coding

`'BCC'` (default)

This property is read only.

Type of forward error correction coding for the data field, specified as `'BCC'` to indicate binary convolutional coding. LDPC coding is not currently supported.

Data Types: `char`

`PSDULength` — Number of bytes carried in the user payload

1024 (default) | integer from 0 to 65,535

Number of bytes carried in the user payload, specified as an integer from 0 to 65,535. A `PSDULength` of 0 implies a sounding packet for which there are no data bits to recover.

Example: `512`

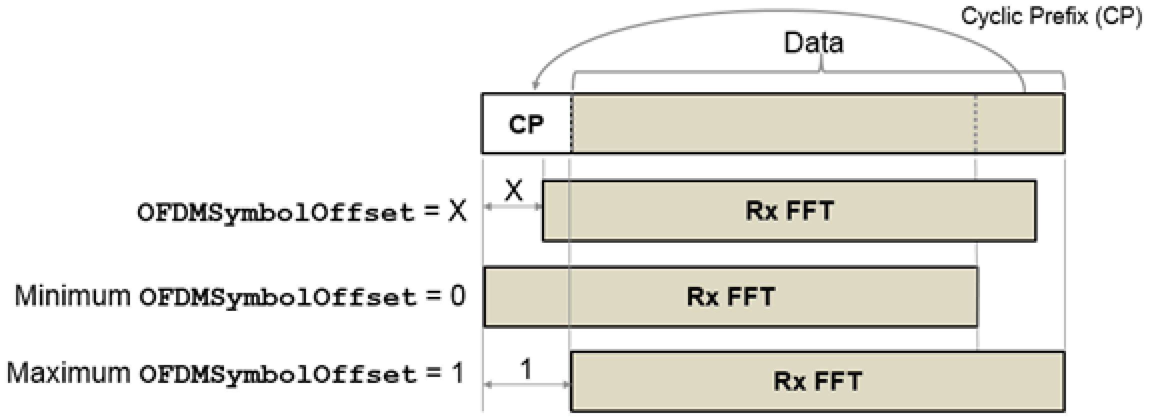
Data Types: `double`

`cfgRec` — Algorithm parameters
`wlanRecoveryConfig` object

Algorithm parameters, specified as a `wlanRecoveryConfig` object. The object properties include:

`OFDMSymbolOffset` — OFDM symbol sampling offset
0.75 (default) | scalar value from 0 to 1

OFDM symbol sampling offset represented as a fraction of the cyclic prefix (CP) length, specified as a scalar value from 0 to 1. This value indicates the start location for OFDM demodulation, relative to the beginning of the cyclic prefix. `OFDMSymbolOffset` = 0 represents the start of the cyclic prefix and `OFDMSymbolOffset` = 1 represents the end of the cyclic prefix.



Data Types: `double`

`EqualizationMethod` — Equalization method
`'MMSE'` (default) | `'ZF'`

Equalization method, specified as `'MMSE'` or `'ZF'`.

- `'MMSE'` indicates that the receiver uses a minimum mean square error equalizer.
- `'ZF'` indicates that the receiver uses a zero-forcing equalizer.

Example: `'ZF'`

Data Types: `char`

`PilotPhaseTracking` — Pilot phase tracking
`'PreEQ'` (default) | `'None'`

Pilot phase tracking, specified as either of these strings: `'PreEQ'` or `'None'`. If `'PreEQ'` is specified, pilot phase tracking is enabled and is performed prior to any equalization operation. If `'None'` is specified, pilot phase tracking does not occur.

Data Types: `char`

Output Arguments

`recData` — Recovered binary output data
binary column vector

Recovered binary output data, returned as a column vector of length $8 \times N_{PSDU}$, where N_{PSDU} is the length of the PSDU in bytes. See wlanHTConfig Properties for `PSDULength` details.

Data Types: `int8`

`eqSym` — Equalized symbols
column vector | matrix | 3-D array

Equalized symbols, returned as an N_{SD}-by-N_{SYM}-by-N_{SS} array. N_{SD} is the number of data subcarriers, N_{SYM} is the number of OFDM symbols in the HT-Data field, and N_{SS} is the number of spatial streams.

Data Types: `double`
Complex Number Support: Yes

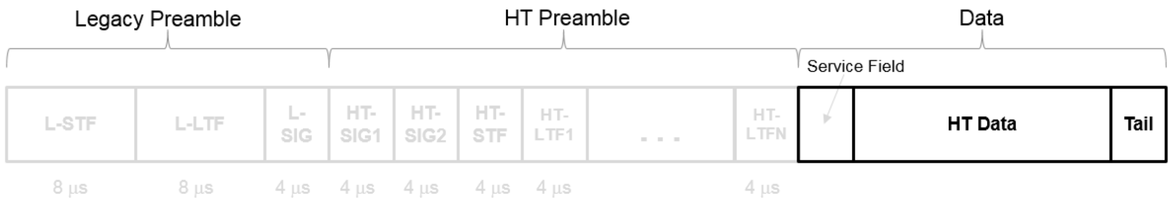
`cpe` — Common phase error
column vector

Common phase error in radians, returned as a column vector having length N_{SYM}. N_{SYM} is the number of OFDM symbols in the HT-Data field.

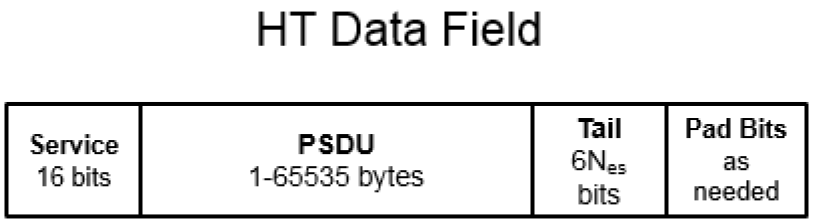
More About

HT-Data field

The high throughput data field (HT data) follows the last HT-LTF of an HT-mixed packet.



The high throughput data field is used to transmit one or more frames from the MAC layer and consists of four subfields.



- **Service field** — Contains 16 zeros to initialize the data scrambler.
- **PSDU** — Variable-length field containing the PLCP service data unit (PSDU). In 802.11, the PSDU can consist of an aggregate of several MAC service data units.
- **Tail** — Tail bits required to terminate a convolutional code. The field uses six zeros for each encoding stream.
- **Pad Bits** — Variable-length field required to ensure that the HT data field consists of an integer number of symbols.

HT-Mixed

High throughput mixed (HT-mixed) format devices support a mixed mode in which the PLCP header is compatible with HT and Non-HT modes.

References

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

See Also

`wlanHTConfig` | `wlanRecoveryConfig`

Introduced in R2015b

wlanHTLTF

Generate HT-LTF waveform

Syntax

```
y = wlanHTLTF(cfg)
```

Description

`y = wlanHTLTF(cfg)` generates an “HT-LTF” on page 1-89[8] time-domain waveform for “HT-mixed” on page 1-91 format transmissions given the parameters specified in `cfg`.

Examples

Generate Single-Stream HT-LTF Waveform

Create a `wlanHTConfig` object having a channel bandwidth of 40 MHz.

```
cfg = wlanHTConfig('ChannelBandwidth','CBW40');
```

Generate the corresponding HT-LTF.

```
hltfOut = wlanHTLTF(cfg);
size(hltfOut)
```

```
ans =

   160     1
```

8. IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.

The `cfg` parameters result in a 160-sample waveform having only one column corresponding to a single stream transmission.

Generate HT-LTF with Four Space-Time Streams

Generate an HT-LTF having four transmit antennas and four space-time streams.

Create a `wlanHTConfig` object having an MCS of 31, four transmit antennas, and four space-time streams.

```
cfg = wlanHTConfig('MCS',31,'NumTransmitAntennas',4,'NumSpaceTimeStreams',4)
```

```
cfg = 

  wlanHTConfig with properties:

       ChannelBandwidth: 'CBW20'
    NumTransmitAntennas: 4
    NumSpaceTimeStreams: 4
         SpatialMapping: 'Direct'
                    MCS: 31
          GuardInterval: 'Long'
          ChannelCoding: 'BCC'
             PSDULength: 1024
     RecommendSmoothing: true
```

Generate the corresponding HT-LTF.

```
hltfOut = wlanHTLTF(cfg);
```

Verify that the HT-LTF output consists of four streams (one for each antenna).

```
size(hltfOut)
```

```
ans =

   320     4
```

Because the channel bandwidth is 20 MHz and has four space-time streams, the output waveform has four HT-LTF and 320 time-domain samples.

Input Arguments

`cfg` — Format configuration

`wlanHTConfig` object

Format configuration, specified as a `wlanHTConfig` object. The `wlanHTLTF` function uses these properties:

`ChannelBandwidth` — Channel bandwidth

`'CBW20'` (default) | `'CBW40'`

Channel bandwidth in MHz, specified as `'CBW20'` or `'CBW40'`.

Data Types: `char`

`NumTransmitAntennas` — Number of transmit antennas

1 (default) | 2 | 3 | 4

Number of transmit antennas, specified as 1, 2, 3, or 4.

Data Types: `double`

`NumSpaceTimeStreams` — Number of space-time streams

1 (default) | 2 | 3 | 4

Number of space-time streams in the transmission, specified as 1, 2, 3, or 4.

Data Types: `double`

`NumExtensionStreams` — Number of extension spatial streams

0 (default) | 1 | 2 | 3

Number of extension spatial streams in the transmission, specified as 0, 1, 2, or 3. When `NumExtensionStreams` is greater than 0, `SpatialMapping` must be `'Custom'`.

Data Types: `double`

`SpatialMapping` — Spatial mapping scheme

`'Direct'` (default) | `'Hadamard'` | `'Fourier'` | `'Custom'`

Spatial mapping scheme, specified as `'Direct'`, `'Hadamard'`, `'Fourier'`, or `'Custom'`. The default value `'Direct'`, applies when `NumTransmitAntennas` and `NumSpaceTimeStreams` are equal.

Data Types: `char`

`SpatialMappingMatrix` — Spatial mapping matrix
1 (default) | scalar | matrix | 3-D array

Spatial mapping matrix, specified as a scalar, matrix, or 3-D array. Use this property to rotate and scale the constellation mapper output vector. This property applies when the `SpatialMapping` property is set to `'Custom'`. The spatial mapping matrix is used for beamforming and mixing space-time streams over the transmit antennas.

- When specified as a scalar, `NumTransmitAntennas` = `NumSpaceTimeStreams` = 1 and a constant value applies to all the subcarriers.
- When specified as a matrix, the size must be (N_{STS} + N_{ESS})-by-N_T. N_{STS} is the number of space-time streams. N_{ESS} is the number of extension spatial streams. N_T is the number of transmit antennas. The spatial mapping matrix applies to all the subcarriers. The first N_{STS} and last N_{ESS} rows apply to the space-time streams and extension spatial streams respectively.
- When specified as a 3-D array, the size must be N_{ST}-by-(N_{STS} + N_{ESS})-by-N_T. N_{ST} is the sum of the data and pilot subcarriers, as determined by `ChannelBandwidth`. N_{STS} is the number of space-time streams. N_{ESS} is the number of extension spatial streams. N_T is the number of transmit antennas. In this case, each data and pilot subcarrier can have its own spatial mapping matrix.

  The table shows the `ChannelBandwidth` setting and the corresponding N_{ST}.

`ChannelBandwidth`	N_{ST}
`'CBW20'`	56
`'CBW40'`	114

The calling function normalizes the spatial mapping matrix for each subcarrier.

Example: `[0.5 0.3; 0.4 0.4; 0.5 0.8]` represents a spatial mapping matrix having three space-time streams and two transmit antennas.

Data Types: `double`
Complex Number Support: Yes

Output Arguments

y — HT-LTF waveform

matirx

HT-LTF waveform, returned as an ($N_S \times N_{HTLTF}$)-by-N_T matrix. N_S is the number of time domain samples per N_{HTLTF}, where N_{HTLTF} is the number of OFDM symbols in the "HT-LTF" on page 1-89. N_T is the number of transmit antennas.

N_S is proportional to the channel bandwidth. Each symbol contains 80 time samples per 20 MHz channel.

`ChannelBandwidth`	N_S
`'CBW20'`	80
`'CBW40'`	160

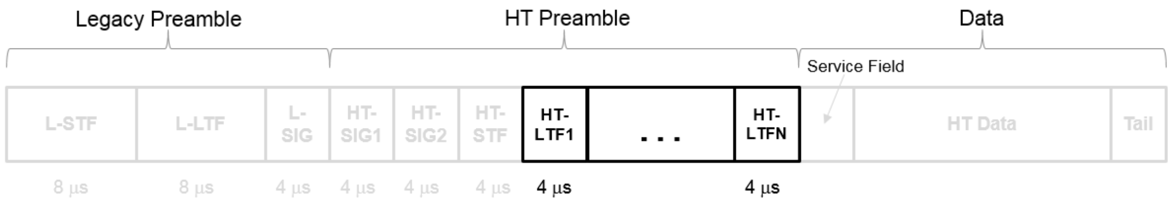
Determination of the number of N_{HTLTF} is described in "HT-LTF" on page 1-89.

Data Types: `double`
Complex Number Support: Yes

More About

HT-LTF

The high throughput long training field (HT-LTF) is located between the HT-STF and data field of an HT-mixed packet.



As described in IEEE Std 802.11-2012, Section 20.3.9.4.6, the receiver can use the HT-LTF to estimate the MIMO channel between the set of QAM mapper outputs (or, if STBC is applied, the STBC encoder outputs) and the receive chains. The HT-LTF portion has one or two parts. The first part consists of one, two, or four HT-LTFs that are necessary for demodulation of the HT-Data portion of the PPDU. These HT-LTFs are referred to as HT-DLTFs. The optional second part consists of zero, one, two, or four HT-LTFs that can be used to sound extra spatial dimensions of the MIMO channel not utilized by the HT-Data portion of the PPDU. These HT-LTFs are referred to as HT-ELTFs. Each HT long training symbol is 4 µs. The number of space-time streams and the number of extension streams determines the number of HT-LTF symbols transmitted.

Tables 20-12, 20-13 and 20-14 from IEEE Std 802.11-2012 are reproduced here.

N_{STS} Determination	N_{HTDLTF} Determination	N_{HTELTF} Determination
Table 20-12 defines the number of space-time streams (N_{STS}) based on the number of spatial streams (N_{SS}) from the MCS and the STBC field.	Table 20-13 defines the number of HT-DLTFs required for the N_{STS}.	Table 20-14 defines the number of HT-ELTFs required for the number of extension spatial streams (N_{ESS}). N_{ESS} is defined in HT-SIG$_2$.

N_{SS} from MCS	STBC field	N_{STS}
1	0	1
1	1	2
2	0	2
2	1	3
2	2	4
3	0	3
3	1	4
4	0	4

N_{STS}	N_{HTDLTF}
1	1
2	2
3	4
4	4

N_{ESS}	N_{HTELTF}
0	0
1	1
2	2
3	4

Additional constraints include:

- $N_{HTLTF} = N_{HTDLTF} + N_{HTELTF} \leq 5$.
- $N_{STS} + N_{ESS} \leq 4$.
  - When $N_{STS} = 3$, N_{ESS} cannot exceed one.

- If $N_{ESS} = 1$ when $N_{STS} = 3$ then $N_{HTLTF} = 5$.

HT-mixed

As described in IEEE Std 802.11-2012, Section 20.1.4, high throughput mixed (HT-mixed) format packets contain a preamble compatible with IEEE Std 802.11-2012, Section 18 and Section 19 receivers. Non-HT (Section 18 and Section19) STAs can decode the non-HT fields (L-STF, L-LTF, and L-SIG). The remaining preamble fields (HT-SIG, HT-STF, and HT-LTF) are for HT transmission, so the Section 18 and Section 19 STAs cannot decode them. The HT portion of the packet is described in IEEE Std 802.11-2012, Section 20.3.9.4. Support for the HT-mixed format is mandatory.

PPDU

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

References

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

See Also

`wlanHTConfig` | `wlanHTData` | `wlanHTLTFChannelEstimate` | `wlanHTLTFDemodulate` | `wlanLLTF`

Introduced in R2015b

wlanHTLTFDemodulate

Demodulate HT-LTF waveform

Syntax

```
y = wlanHTLTFDemodulate(x,cfg)
y = wlanHTLTFDemodulate(x,cfg,OFDMSymbolOffset)
```

Description

`y = wlanHTLTFDemodulate(x,cfg)` returns the demodulated “HT-LTF” on page 1-96[9], `y`, given received HT-LTF `x`. The input signal is a component of the “HT-mixed” on page 1-97 format “PPDU” on page 1-97. The function demodulates the signal using the information in the `wlanHTConfig` object, `cfg`.

`y = wlanHTLTFDemodulate(x,cfg,OFDMSymbolOffset)` specifies the OFDM symbol sampling offset.

Examples

Demodulate HT-LTF in AWGN

Create an HT configuration object.

```
cfg = wlanHTConfig;
```

Generate an HT-LTF signal based on the object.

```
x = wlanHTLTF(cfg);
```

Pass the HT-LTF signal through an AWGN channel.

```
y = awgn(x,20);
```

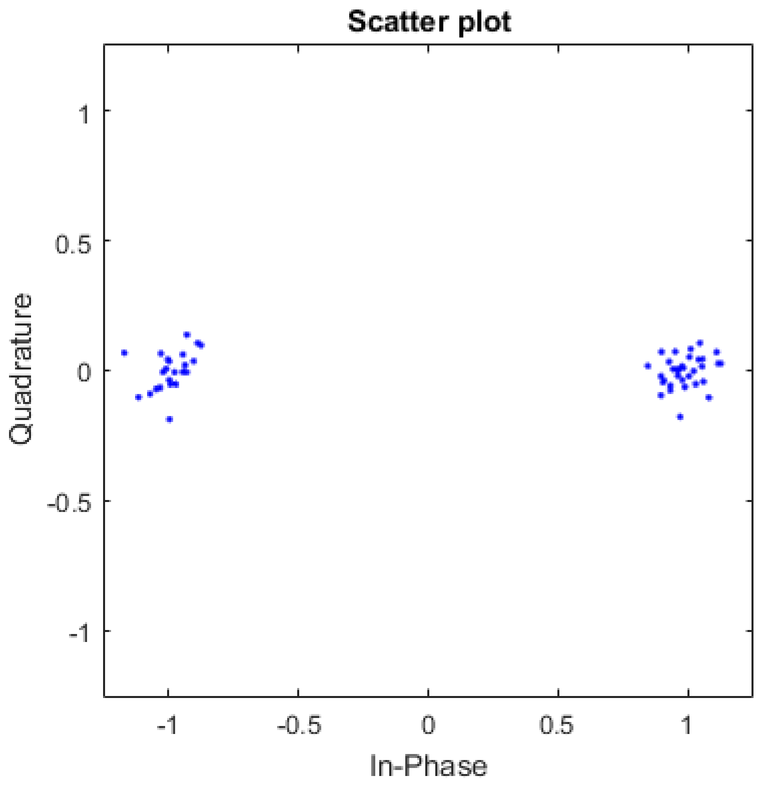
Demodulate the received signal.

9. IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.

```
z = wlanHTLTFDemodulate(y,cfg);
```

Display the scatter plot of the demodulated signal.

```
scatterplot(z)
```



Demodulate 2x2 HT-LTF with OFDM Symbol Offset

Create an HT configuration object having two transmit antennas and two space-time streams.

```
cfg = wlanHTConfig('NumTransmitAntennas',2,'NumSpaceTimeStreams',2, ...
    'MCS',8);
```

Generate the HT-LTF based on the configuration object.

```
x = wlanHTLTF(cfg);
```

Pass the HT-LTF signal through an AWGN channel.

```
y = awgn(x,10);
```

Demodulate the received signal. Set the OFDM symbol offset to `0.5`, which corresponds to 1/2 of the cyclic prefix length.

```
z = wlanHTLTFDemodulate(y,cfg,0.5);
```

Input Arguments

`x` — Input signal

matrix

Input signal comprising an "HT-LTF" on page 1-96, specified as an N_S-by-N_R matrix. N_S is the number of samples and N_R is the number of receive antennas. You can generate the signal by using the `wlanHTLTF` function.

Data Types: `double`
Complex Number Support: Yes

`cfg` — HT format configuration

`wlanHTConfig` object

HT format configuration, specified as a `wlanHTConfig` object. The function uses the following `wlanHTConfig` object properties:

`ChannelBandwidth` — Channel bandwidth

`'CBW20'` (default) | `'CBW40'`

Channel bandwidth in MHz, specified as `'CBW20'` or `'CBW40'`.

Data Types: `char`

`NumSpaceTimeStreams` — Number of space-time streams

1 (default) | 2 | 3 | 4

Number of space-time streams in the transmission, specified as 1, 2, 3, or 4.

Data Types: `double`

`NumExtensionStreams` — Number of extension spatial streams

0 (default) | 1 | 2 | 3

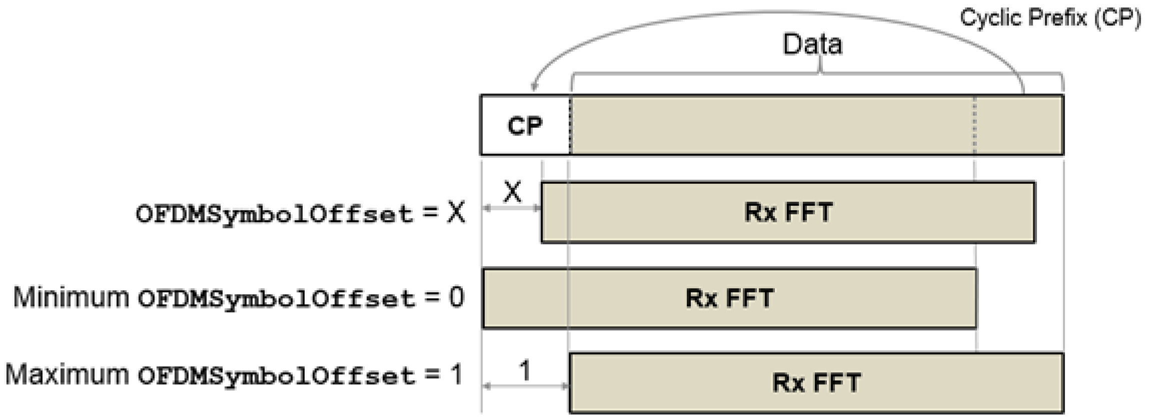
Number of extension spatial streams in the transmission, specified as 0, 1, 2, or 3. When `NumExtensionStreams` is greater than 0, `SpatialMapping` must be `'Custom'`.

Data Types: `double`

`OFDMSymbolOffset` — OFDM symbol sampling offset

0.75 (default) | scalar value from 0 to 1

OFDM symbol sampling offset represented as a fraction of the cyclic prefix (CP) length, specified as a scalar value from 0 to 1. This value indicates the start location for OFDM demodulation, relative to the beginning of the cyclic prefix. `OFDMSymbolOffset` = 0 represents the start of the cyclic prefix and `OFDMSymbolOffset` = 1 represents the end of the cyclic prefix.



Data Types: `double`

Output Arguments

y — Demodulated HT-LTF signal

matrix | 3-D array

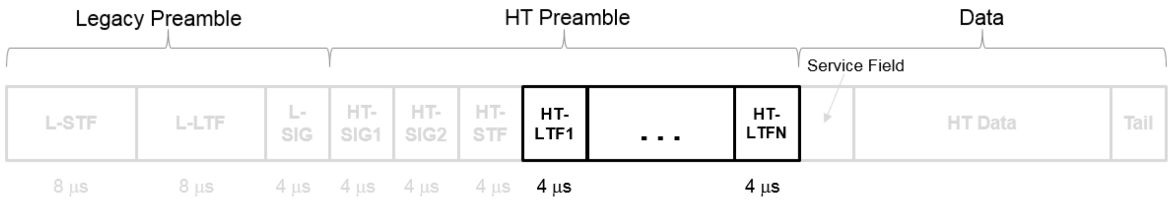
Demodulated HT-LTF signal for an HT-Mixed PPDU, returned as an N_{ST}-by-N_{SYM}-by-N_R matrix or array. N_{ST} is the number of data and pilot subcarriers. N_{SYM} is the number of OFDM symbols in the HT-LTF. N_R is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

More About

HT-LTF

The high throughput long training field (HT-LTF) is located between the HT-STF and data field of an HT-mixed packet.



As described in IEEE Std 802.11-2012, Section 20.3.9.4.6, the receiver can use the HT-LTF to estimate the MIMO channel between the set of QAM mapper outputs (or, if STBC is applied, the STBC encoder outputs) and the receive chains. The HT-LTF portion has one or two parts. The first part consists of one, two, or four HT-LTFs that are necessary for demodulation of the HT-Data portion of the PPDU. These HT-LTFs are referred to as HT-DLTFs. The optional second part consists of zero, one, two, or four HT-LTFs that can be used to sound extra spatial dimensions of the MIMO channel not utilized by the HT-Data portion of the PPDU. These HT-LTFs are referred to as HT-ELTFs. Each HT long training symbol is 4 µs. The number of space-time streams and the number of extension streams determines the number of HT-LTF symbols transmitted.

Tables 20-12, 20-13 and 20-14 from IEEE Std 802.11-2012 are reproduced here.

N_{STS} Determination	N_{HTDLTF} Determination	N_{HTELTF} Determination
Table 20-12 defines the number of space-time streams (N_{STS}) based on the number of spatial streams (N_{SS}) from the MCS and the STBC field.	Table 20-13 defines the number of HT-DLTFs required for the N_{STS}.	Table 20-14 defines the number of HT-ELTFs required for the number of extension spatial streams (N_{ESS}). N_{ESS} is defined in HT-SIG$_2$.

N_{STS} Determination		
N_{SS} from MCS	**STBC field**	**N_{STS}**
1	0	1
1	1	2
2	0	2
2	1	3
2	2	4
3	0	3
3	1	4
4	0	4

N_{HTDLTF} Determination	
N_{STS}	**N_{HTDLTF}**
1	1
2	2
3	4
4	4

N_{HTELTF} Determination	
N_{ESS}	**N_{HTELTF}**
0	0
1	1
2	2
3	4

Additional constraints include:

- $N_{HTLTF} = N_{HTDLTF} + N_{HTELTF} \leq 5$.
- $N_{STS} + N_{ESS} \leq 4$.
  - When $N_{STS} = 3$, N_{ESS} cannot exceed one.
  - If $N_{ESS} = 1$ when $N_{STS} = 3$ then $N_{HTLTF} = 5$.

HT-mixed

High throughput mixed (HT-mixed) format devices support a mixed mode in which the PLCP header is compatible with HT and non-HT modes.

PPDU

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

References

[1] IEEE Std 802.11™-2012 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and

metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

See Also

`wlanHTConfig` | `wlanHTLTF` | `wlanHTLTFChannelEstimate`

Introduced in R2015b

wlanHTSig

Generate HT-SIG waveform

Syntax

```
y = wlanHTSIG(cfg)
[y,bits] = wlanHTSIG(cfg)
```

Description

`y = wlanHTSIG(cfg)` generates an “HT-SIG” on page 1-104[10] time-domain waveform for “HT-mixed” on page 1-105 format transmissions given the parameters specified in `cfg`.

`[y,bits] = wlanHTSIG(cfg)` returns the information bits, `bits`, that comprise the HT-SIG field.

Examples

Generate HT-SIG Waveform

Generate an HT-SIG waveform for a single transmit antenna.

Create an HT configuration object. Specify a 40 MHz channel bandwidth.

```
cfg = wlanHTConfig;
cfg.ChannelBandwidth = 'CBW40'
```

```
cfg = 

  wlanHTConfig with properties:

       ChannelBandwidth: 'CBW40'
```

10. IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.

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

Generate the HT-SIG waveform. Determine the size of the waveform.

```
y = wlanHTSIG(cfg);
size(y)
```

```
ans =

   320     1
```

The function returns a waveform having a complex output of 320 samples corresponding to two 160-sample OFDM symbols.

Display MCS Information from HT-SIG

Generate an HT-SIG waveform and display the MCS information. Change the MCS and display the updated information.

Create a `wlanHTConfig` object having two spatial streams and two transmit antennas. Specify an MCS value of 8, corresponding to BPSK modulation and a coding rate of 1/2.

```
cfg = wlanHTConfig('NumSpaceTimeStreams',2,'NumTransmitAntennas',2,'MCS',8);
```

Generate the information bits from the HT-SIG waveform.

```
[~,sigBits] = wlanHTSIG(cfg);
```

Extract the MCS field from `sigBits` and convert it to its decimal equivalent. The MCS information is contained in bits 1-7.

```
mcsBits = sigBits(1:7);
bi2de(mcsBits')
```

```
ans =

     8
```

The MCS matches the specified value.

Change the MCS to 13, which corresponds to 64-QAM modulation with a 2/3 coding rate. Generate the HT-SIG waveform.

```
cfg.MCS = 13;
[~,sigBits] = wlanHTSIG(cfg);
```

Verify that the MCS bits are the binary equivalent of 13.

```
mcsBits = sigBits(1:7);
bi2de(mcsBits')
```

```
ans =

    13
```

Input Arguments

`cfg` — Format configuration

`wlanHTConfig` object

Format configuration, specified as a `wlanHTConfig` object. The `wlanHTSIG` function uses these properties.

`MCS` — Modulation and coding scheme

0 (default) | integer from 0 to 31

Modulation and coding scheme to use for transmitting the current packet, specified as an integer from 0 to 31. The MCS setting identifies which modulation and coding rate combination is used, and the number of spatial streams (N_{SS}).

MCS(Note 1)	N_{SS}(Note 1)	Modulation	Coding Rate
0, 8, 16, or 24	1, 2, 3, or 4	BPSK	1/2

MCS(Note 1)	N_{SS}(Note 1)	Modulation	Coding Rate
1, 9, 17, or 25	1, 2, 3, or 4	`QPSK`	`1/2`
2, 10, 18, or 26	1, 2, 3, or 4	`QPSK`	`3/4`
3, 11, 19, or 27	1, 2, 3, or 4	`16QAM`	`1/2`
4, 12, 20, or 28	1, 2, 3, or 4	`16QAM`	`3/4`
5, 13, 21, or 29	1, 2, 3, or 4	`64QAM`	`2/3`
6, 14, 22, or 30	1, 2, 3, or 4	`64QAM`	`3/4`
7, 15, 23, or 31	1, 2, 3, or 4	`64QAM`	`5/6`
Note-1 MCS from 0 to 7 have one spatial stream. MCS from 8 to 15 have two spatial streams. MCS from 16 to 23 have three spatial streams. MCS from 24 to 31 have four spatial streams.			

See IEEE 802.11-2012, Section 20.6 for further description of MCS dependent parameters.

When working with the HT-Data field, if the number of space-time streams is equal to the number of spatial streams, no space-time block coding (STBC) is used. See IEEE 802.11-2012, Section 20.3.11.9.2 for further description of STBC mapping.

Example: `22` indicates an MCS with three spatial streams, 64-QAM modulation, and a 3/4 coding rate.

Data Types: `double`

`ChannelBandwidth` — Channel bandwidth

`'CBW20'` (default) | `'CBW40'`

Channel bandwidth in MHz, specified as `'CBW20'` or `'CBW40'`.

Data Types: `char`

`PSDULength` — Number of bytes carried in the user payload

1024 (default) | integer from 0 to 65,535

Number of bytes carried in the user payload, specified as an integer from 0 to 65,535. A `PSDULength` of 0 implies a sounding packet for which there are no data bits to recover.

Example: `512`

Data Types: `double`

`RecommendSmoothing` — Indication of whether frequency-domain smoothing is recommended

true (default) | false

Indication of whether frequency-domain smoothing is recommended as part of channel estimation, specified as a logical. If the frequency profile across the channel is nonvarying, the receiver sets this property to `true`.

Data Types: `logical`

`NumSpaceTimeStreams` — Number of space-time streams

1 (default) | 2 | 3 | 4

Number of space-time streams in the transmission, specified as 1, 2, 3, or 4.

Data Types: `double`

`ChannelCoding` — Type of forward error correction coding

`'BCC'` (default)

This property is read only.

Type of forward error correction coding for the data field, specified as `'BCC'` to indicate binary convolutional coding. LDPC coding is not currently supported.

Data Types: `char`

`GuardInterval` — Cyclic prefix length for the data field within a packet

`'Long'` (default) | `'Short'`

Cyclic prefix length for the data field within a packet, specified as `'Long'` or `'Short'`.

- The long guard interval length is 800 ns.
- The short guard interval length is 400 ns.

Data Types: `char`

`NumExtensionStreams` — Number of extension spatial streams

0 (default) | 1 | 2 | 3

Number of extension spatial streams in the transmission, specified as 0, 1, 2, or 3. When `NumExtensionStreams` is greater than 0, `SpatialMapping` must be `'Custom'`.

Data Types: `double`

Output Arguments

y — HT-SIG waveform

matrix

HT-SIG waveform, returned as an N_S-by-N_T matrix. N_S is the number of time-domain samples, and N_T is the number of transmit antennas.

Data Types: `double`
Complex Number Support: Yes

`bits` — HT-SIG information bits

vector

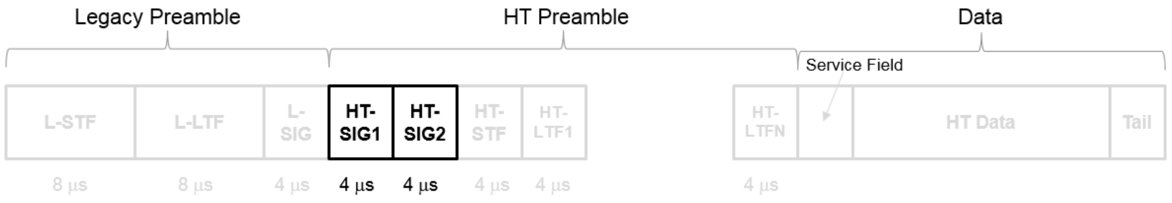
HT-SIG information bits, returned as a 48-by-1 vector.

Data Types: `int8`

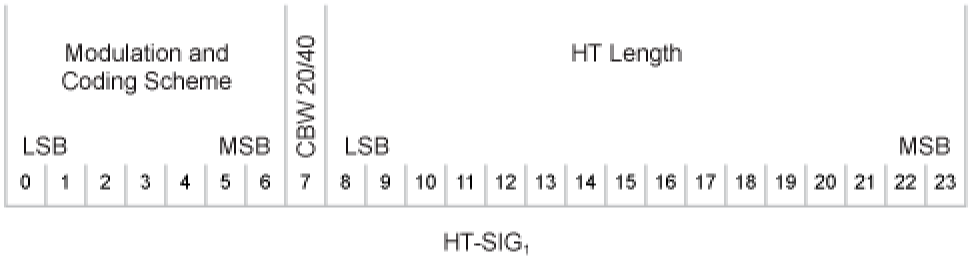
More About

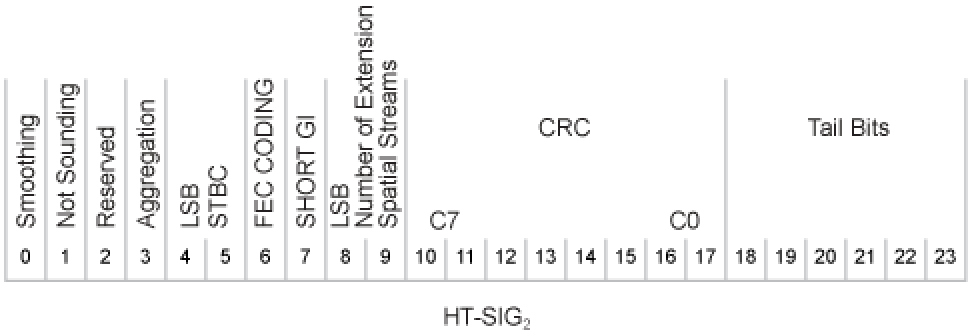
HT-SIG

The high throughput signal (HT-SIG) field is located between the L-SIG field and HT-STF and is part of the HT-mixed format preamble. It is composed of two symbols, HT-SIG$_1$ and HT-SIG$_2$.



HT-SIG carries information used to decode the HT packet, including the MCS, packet length, FEC coding type, guard interval, number of extension spatial streams, and whether there is payload aggregation. The HT-SIG symbols are also used for auto-detection between HT-mixed format and legacy OFDM packets.





Refer to IEEE Std 802.11-2012, Section 20.3.9.4.3 for a detailed description of the HT-SIG field.

HT-mixed

As described in IEEE Std 802.11-2012, Section 20.1.4, high throughput mixed (HT-mixed) format packets contain a preamble compatible with IEEE Std 802.11-2012, Section 18 and Section 19 receivers. Non-HT (Section 18 and Section19) STAs can decode the non-HT fields (L-STF, L-LTF, and L-SIG). The remaining preamble fields (HT-SIG, HT-STF, and HT-LTF) are for HT transmission, so the Section 18 and Section 19 STAs cannot decode them. The HT portion of the packet is described in IEEE Std 802.11-2012, Section 20.3.9.4. Support for the HT-mixed format is mandatory.

References

[1] IEEE Std 802.11™-2012 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and

metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

See Also

`wlanHTConfig` | `wlanHTSIGRecover` | `wlanHTSTF` | `wlanLSIG`

Introduced in R2015b

wlanHTSIGRecover

Recover HT-SIG information bits

Syntax

```
recBits = wlanHTSIGRecover(rxSig,chEst,noiseVarEst,cbw)
recBits = wlanHTSIGRecover(rxSig,chEst,noiseVarEst,cbw,cfgRec)
[recBits,failCRC] = wlanHTSIGRecover( ___ )
[recBits,failCRC,eqSym] = wlanHTSIGRecover( ___ )
[recBits,failCRC,eqSym,cpe] = wlanHTSIGRecover( ___ )
```

Description

`recBits = wlanHTSIGRecover(rxSig,chEst,noiseVarEst,cbw)` returns the recovered information bits from the "HT-SIG" on page 1-115[11] field and performs a CRC check. Inputs include the channel estimate data `chEst`, noise variance estimate `noiseVarEst`, and channel bandwidth `cbw`.

`recBits = wlanHTSIGRecover(rxSig,chEst,noiseVarEst,cbw,cfgRec)` specifies algorithm parameters using `wlanRecoveryConfig` object `cfgRec`.

`[recBits,failCRC] = wlanHTSIGRecover( ___ )` returns the result of the CRC check, `failCRC`, using any of the arguments from the previous syntaxes.

`[recBits,failCRC,eqSym] = wlanHTSIGRecover( ___ )` returns the equalized symbols, `eqSym`.

`[recBits,failCRC,eqSym,cpe] = wlanHTSIGRecover( ___ )` returns the common phase error, `cpe`.

11. IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.

Examples

Recover HT-SIG Information Bits in Perfect Channel

Create a `wlanHTConfig` object having a channel bandwidth of 40 MHz. Use the object to create an HT-SIG field.

```
cfg = wlanHTConfig('ChannelBandwidth','CBW40');
[txSig,txBits] = wlanHTSIG(cfg);
```

Because a perfect channel is assumed, specify the channel estimate as a column vector of ones and the noise variance estimate as zero.

```
chEst = ones(104,1);
noiseVarEst = 0;
```

Recover the HT-SIG information bits. Verify that the received information bits are identical to the transmitted bits.

```
rxBits = wlanHTSIGRecover(txSig,chEst,noiseVarEst,'CBW40');
numerr = biterr(txBits,rxBits)
```

```
numerr =

     0
```

Recover HT-SIG Using Zero-Forcing Equalizer

Create a `wlanHTConfig` object having a channel bandwidth of 40 MHz. Use the object to create an HT-SIG field.

```
cfg = wlanHTConfig('ChannelBandwidth','CBW40');
[txSig,txBits] = wlanHTSIG(cfg);
```

Pass the transmitted HT-SIG through an AWGN channel.

```
ch = comm.AWGNChannel('NoiseMethod','Variance', ...
    'Variance',0.1);

rxSig = step(ch,txSig);
```

Use a zero-forcing equalizer by creating a `wlanRecoveryConfig` object with its `EqualizationMethod` property set to `'ZF'`.

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

Recover the HT-SIG field. Verify that the received information has no bit errors.

```
rxBits = wlanHTSIGRecover(rxSig,ones(104,1),0.1,'CBW40',cfgRec);
biterr(txBits,rxBits)
```

```
ans =

     0
```

Recover HT-SIG in 2x2 MIMO Channel

Recover HT-SIG in a 2x2 MIMO channel with AWGN. Confirm that the `CRC` check passes.

Configure a 2x2 MIMO TGn channel.

```
chanBW = 'CBW20';
cfg = wlanHTConfig( ...
    'ChannelBandwidth',chanBW, ...
    'NumTransmitAntennas',2, ...
    'NumSpaceTimeStreams',2);
```

Generate L-LTF and HT-SIG waveforms.

```
txLLTF  = wlanLLTF(cfg);
txHTSIG = wlanHTSIG(cfg);
```

Set the sample rate to correspond to the channel bandwidth. Create a TGn 2x2 MIMO channel without large scale fading effects.

```
fsamp = 20e6;
ch = wlanTGnChannel('SampleRate',fsamp, ...
    'LargeScaleFadingEffect','None', ...
    'NumTransmitAntennas',2, ...
    'NumReceiveAntennas',2);
```

Pass the L-LTF and HT-SIG waveforms through a TGn channel with white noise.

```
rxLLTF = awgn(step(ch,txLLTF),20);
rxHTSIG = awgn(step(ch,txHTSIG),20);
```

Demodulate the L-LTF signal. Generate a channel estimate by using the demodulated L-LTF.

```
demodLLTF = wlanLLTFDemodulate(rxLLTF,chanBW,1);
chanEst = wlanLLTFChannelEstimate(demodLLTF,chanBW);
```

Recover the information bits, the CRC failure status, and the equalized symbols from the received HT-SIG field.

```
[recHTSIGBits,failCRC,eqSym] = wlanHTSIGRecover(rxHTSIG, ...
    chanEst,0.01,chanBW);
```

Verify that HT-SIG passed a CRC check by examining the status of `failCRC`.

```
failCRC
```

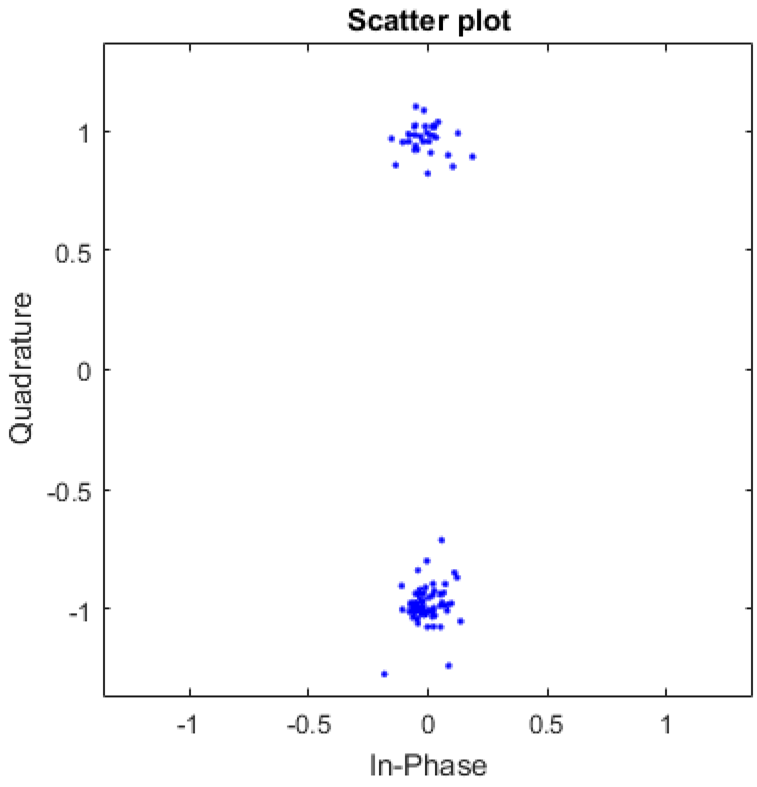
```
failCRC =

     0
```

Because `failCRC` is `0`, HT-SIG passed the CRC check.

Visualize the scatter plot of the equalized symbols, `eqSym`.

```
scatterplot(eqSym(:))
```



Input Arguments

`rxSig` — Received HT-SIG field

matrix

Received HT-SIG field, specified as an N_S-by-N_R matrix. N_S is the number of samples and increases with channel bandwidth.

Channel Bandwidth	N_S
`'CBW20'`	160

Channel Bandwidth	N_S
'CBW40'	320

N_R is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

`chEst` — Channel estimate
vector | 3-D array

Channel estimate, specified as an N_{ST}-by-1-by-N_R array. N_{ST} is the number of occupied subcarriers and increases with channel bandwidth.

Channel Bandwidth	N_{ST}
'CBW20'	52
'CBW40'	104

N_R is the number of receive antennas.

The channel estimate is based on the “L-LTF” on page 1-116.

`noiseVarEst` — Noise variance estimate
nonnegative scalar

Noise variance estimate, specified as a nonnegative scalar.

Data Types: `double`

`cbw` — Channel bandwidth
`'CBW20'` | `'CBW40'`

Channel bandwidth in MHz, specified as one of these strings: `'CBW20'` or `'CBW40'`.

Data Types: `char`

`cfgRec` — Algorithm parameters
`wlanRecoveryConfig` object

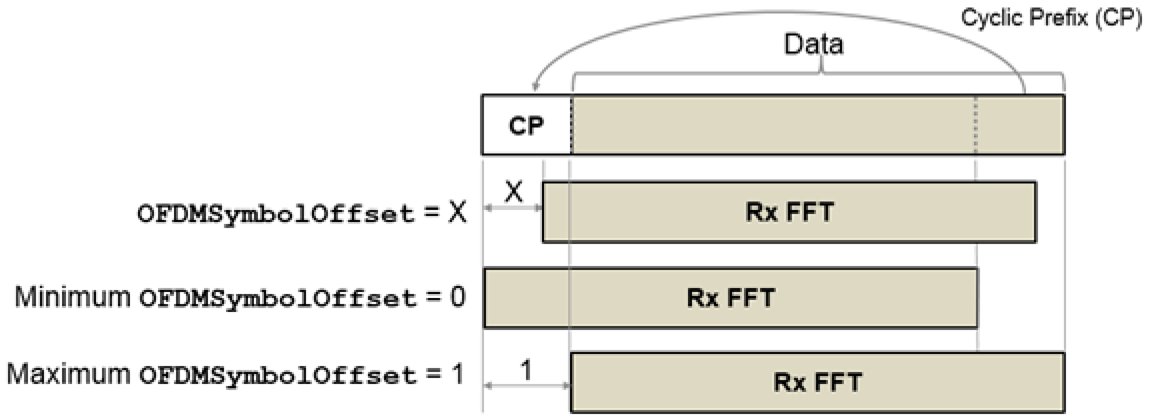
Algorithm parameters, specified as a `wlanRecoveryConfig` object. The function uses these properties.

Note: If `cfgRec` is not provided, the function uses the default values of the `wlanRecoveryConfig` object.

`OFDMSymbolOffset` — OFDM symbol sampling offset

0.75 (default) | scalar value from 0 to 1

OFDM symbol sampling offset represented as a fraction of the cyclic prefix (CP) length, specified as a scalar value from 0 to 1. This value indicates the start location for OFDM demodulation, relative to the beginning of the cyclic prefix. `OFDMSymbolOffset` = 0 represents the start of the cyclic prefix and `OFDMSymbolOffset` = 1 represents the end of the cyclic prefix.



Data Types: `double`

`EqualizationMethod` — Equalization method

`'MMSE'` (default) | `'ZF'`

Equalization method, specified as `'MMSE'` or `'ZF'`.

- `'MMSE'` indicates that the receiver uses a minimum mean square error equalizer.
- `'ZF'` indicates that the receiver uses a zero-forcing equalizer.

Example: `'ZF'`

Data Types: `char`

`PilotPhaseTracking` — Pilot phase tracking
`'PreEQ'` (default) | `'None'`

Pilot phase tracking, specified as either of these strings: `'PreEQ'` or `'None'`. If `'PreEQ'` is specified, pilot phase tracking is enabled and is performed prior to any equalization operation. If `'None'` is specified, pilot phase tracking does not occur.

Data Types: `char`

Output Arguments

`recBits` — Recovered HT-SIG information
vector

Recovered HT-SIG information bits, returned as a 48-element column vector. The number of elements corresponds to the length of the HT-SIG field.

Data Types: `int8`

`failCRC` — CRC failure status
`true` | `false`

CRC failure status, returned as a logical scalar. If HT-SIG fails the CRC check, `failCRC` is `true`.

Data Types: `logical`

`eqSym` — Equalized symbols
matrix

Equalized symbols, returned as a 48-by-2 matrix corresponding to 48 data subcarriers and 2 OFDM symbols.

Data Types: `double`
Complex Number Support: Yes

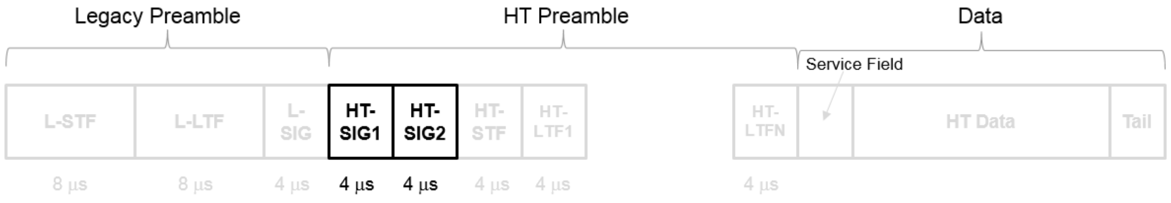
`cpe` — Common phase error
column vector

Common phase error in radians, returned as a 2-by-1 column vector.

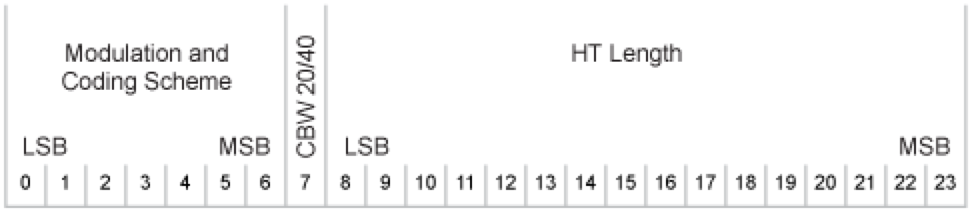
More About

HT-SIG

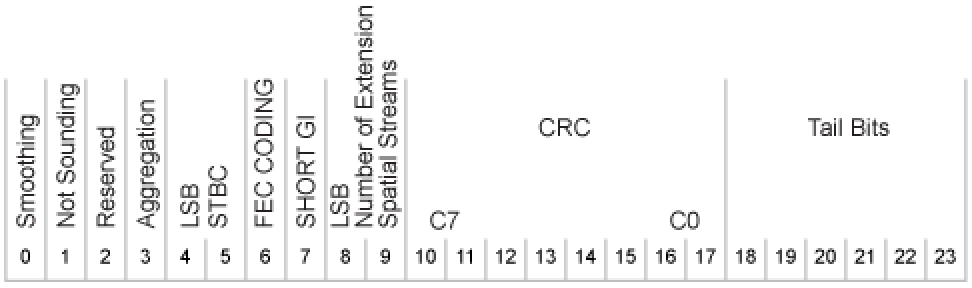
The high throughput signal (HT-SIG) field is located between the L-SIG field and HT-STF and is part of the HT-mixed format preamble. It is composed of two symbols, HT-SIG$_1$ and HT-SIG$_2$.



HT-SIG carries information used to decode the HT packet, including the MCS, packet length, FEC coding type, guard interval, number of extension spatial streams, and whether there is payload aggregation. The HT-SIG symbols are also used for auto-detection between HT-mixed format and legacy OFDM packets.



HT-SIG$_1$

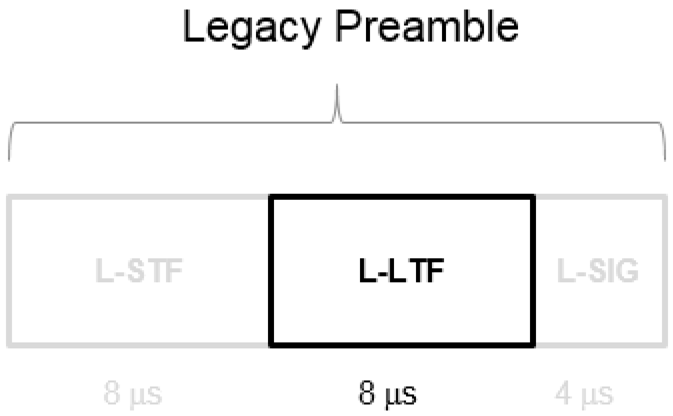


HT-SIG$_2$

Refer to IEEE Std 802.11-2012, Section 20.3.9.4.3 for a detailed description of the HT-SIG field.

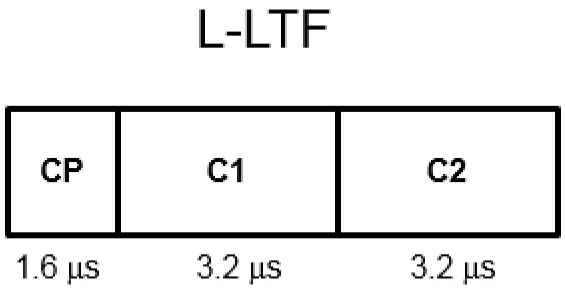
L-LTF

The legacy long training field (L-LTF) is the second field in the legacy preamble for 802.11a/g.



It is also the second field of the HT-mixed format preamble used in 802.11n and the VHT format preamble used in 802.11ac. Channel estimation, frequency offset estimation, and time synchronization rely on the L-LTF. The long OFDM training symbol consists of 53 subcarriers. IEEE Std 802.11-2012, Equation 18-8 and Equation 18-9 define the L-LTF.

The L-LTF, which is 8 μs in length, is composed of a 1.6 μs cyclic prefix (CP) followed by two identical 3.2 μs long training symbols (C1 and C2). The cyclic prefix (CP) consists of the second half of the long training symbol.



References

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

See Also

`wlanHTConfig` | `wlanHTSIG` | `wlanRecoveryConfig`

Introduced in R2015b

wlanHTSTF

Generate HT-STF waveform

Syntax

```
y = wlanHTSTF(cfg)
```

Description

`y = wlanHTSTF(cfg)` generates an "HT-STF" on page 1-122[12] time-domain waveform for "HT-mixed" on page 1-122 format transmissions, given the parameters specified in `cfg`.

Examples

Generate HT Short Training Field

Create a `wlanHTConfig` object with a 40 MHz bandwidth.

```
cfg = wlanHTConfig('ChannelBandwidth','CBW40');
```

Generate an HT-STF. The function returns a complex output of 160 samples.

```
stf = wlanHTSTF(cfg);
size(stf)


ans =

   160     1

```

Change the channel bandwidth to 20 MHz and create a new HT-STF.

```
cfg.ChannelBandwidth = 'CBW20';
```

12. IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.

```
stf = wlanHTSTF(cfg);
```

Verify that the number of samples has been halved due to the bandwidth reduction.

```
size(stf)
```

```
ans =

    80     1
```

Input Arguments

`cfg` — Format configuration
`wlanHTConfig` object

Format configuration, specified as a `wlanHTConfig` object. The `wlanHTSTF` function uses these properties.

`ChannelBandwidth` — Channel bandwidth
`'CBW20'` (default) | `'CBW40'`

Channel bandwidth in MHz, specified as `'CBW20'` or `'CBW40'`.

Data Types: `char`

`NumTransmitAntennas` — Number of transmit antennas
1 (default) | 2 | 3 | 4

Number of transmit antennas, specified as 1, 2, 3, or 4.

Data Types: `double`

`NumSpaceTimeStreams` — Number of space-time streams
1 (default) | 2 | 3 | 4

Number of space-time streams in the transmission, specified as 1, 2, 3, or 4.

Data Types: `double`

`SpatialMapping` — Spatial mapping scheme
`'Direct'` (default) | `'Hadamard'` | `'Fourier'` | `'Custom'`

Spatial mapping scheme, specified as `'Direct'`, `'Hadamard'`, `'Fourier'`, or `'Custom'`. The default value `'Direct'`, applies when `NumTransmitAntennas` and `NumSpaceTimeStreams` are equal.

Data Types: `char`

`SpatialMappingMatrix` — Spatial mapping matrix

1 (default) | scalar | matrix | 3-D array

Spatial mapping matrix, specified as a scalar, matrix, or 3-D array. Use this property to rotate and scale the constellation mapper output vector. This property applies when the `SpatialMapping` property is set to `'Custom'`. The spatial mapping matrix is used for beamforming and mixing space-time streams over the transmit antennas.

- When specified as a scalar, `NumTransmitAntennas` = `NumSpaceTimeStreams` = 1 and a constant value applies to all the subcarriers.
- When specified as a matrix, the size must be (N_{STS} + N_{ESS})-by-N_T. N_{STS} is the number of space-time streams. N_{ESS} is the number of extension spatial streams. N_T is the number of transmit antennas. The spatial mapping matrix applies to all the subcarriers. The first N_{STS} and last N_{ESS} rows apply to the space-time streams and extension spatial streams respectively.
- When specified as a 3-D array, the size must be N_{ST}-by-(N_{STS} + N_{ESS})-by-N_T. N_{ST} is the sum of the data and pilot subcarriers, as determined by `ChannelBandwidth`. N_{STS} is the number of space-time streams. N_{ESS} is the number of extension spatial streams. N_T is the number of transmit antennas. In this case, each data and pilot subcarrier can have its own spatial mapping matrix.

  The table shows the `ChannelBandwidth` setting and the corresponding N_{ST}.

ChannelBandwidth	N_{ST}
`'CBW20'`	56
`'CBW40'`	114

The calling function normalizes the spatial mapping matrix for each subcarrier.

Example: `[0.5 0.3; 0.4 0.4; 0.5 0.8]` represents a spatial mapping matrix having three space-time streams and two transmit antennas.

Data Types: `double`
Complex Number Support: Yes

Output Arguments

y — HT-STF waveform
matrix

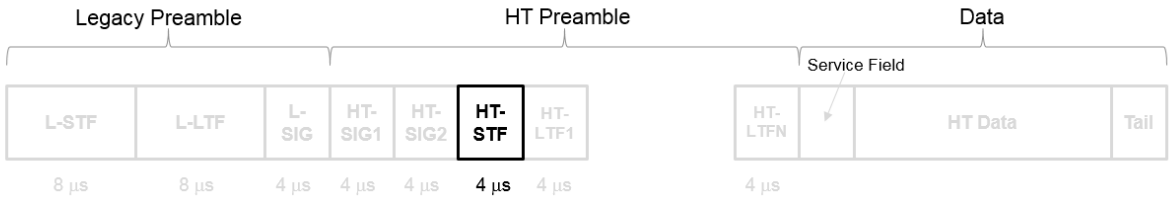
HT-STF waveform, returned as an N_S-by-N_T matrix. N_S is the number of samples, and N_T is the number of transmit antennas.

Data Types: `double`
Complex Number Support: Yes

More About

HT-STF

The high throughput short training field (HT-STF) is located between the HT-SIG and HT-LTF fields of an HT-mixed packet. The HT-STF is 4 µs in length and is used to improve automatic gain control estimation for a MIMO system. For a 20 MHz transmission, the frequency sequence used to construct the HT-STF is identical to that of the L-STF. For a 40 MHz transmission, the upper subcarriers of the HT-STF are constructed from a frequency-shifted and phase-rotated version of the L-STF.



HT-mixed

As described in IEEE Std 802.11-2012, Section 20.1.4, high throughput mixed (HT-mixed) format packets contain a preamble compatible with IEEE Std 802.11-2012, Section 18 and Section 19 receivers. Non-HT (Section 18 and Section19) STAs can decode the non-HT fields (L-STF, L-LTF, and L-SIG). The remaining preamble fields (HT-SIG, HT-STF, and HT-LTF) are for HT transmission, so the Section 18 and Section 19 STAs cannot decode them. The HT portion of the packet is described in IEEE Std 802.11-2012, Section 20.3.9.4. Support for the HT-mixed format is mandatory.

References

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

See Also

`wlanHTConfig` | `wlanHTLTF` | `wlanHTSIG` | `wlanLSTF`

Introduced in R2015b

wlanLLTF

Generate L-LTF waveform

Syntax

```
y = wlanLLTF(cfg)
```

Description

`y = wlanLLTF(cfg)` generates an "L-LTF" on page 1-126[13] time-domain waveform for the specified configuration object.

Examples

Generate L-LTF Waveform

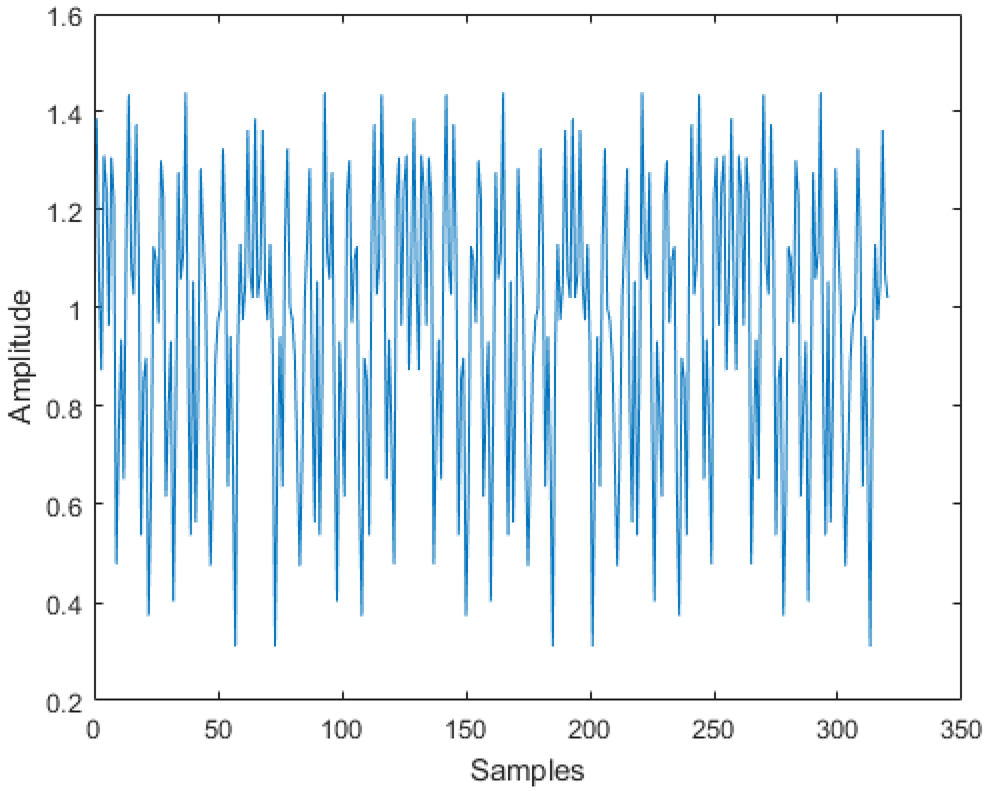
Generate the L-LTF for a 40 MHz single antenna VHT packet.

```
cfgVHT = wlanVHTConfig('ChannelBandwidth', 'CBW40');
y = wlanLLTF(cfgVHT);
size(y)
plot(abs(y));
xlabel('Samples');
ylabel('Amplitude');
```

```
ans =

   320     1
```

13. IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.



The output L-LTF waveform contains 320 time-domain samples for a 40 MHz channel bandwidth.

Input Arguments

`cfg` — Format configuration

`wlanVHTConfig` object | `wlanHTConfig` object | `wlanNonHTConfig` object

Format configuration, specified as a `wlanVHTConfig`, `wlanHTConfig`, or `wlanNonHTConfig` object. For a specified format, the `wlanLLTF` function uses only the object properties indicated.

Transmission Format	Applicable Object Properties
VHT	`ChannelBandwidth`, `NumTransmitAntennas`
HT	`ChannelBandwidth`, `NumTransmitAntennas`
non-HT	`NumTransmitAntennas`

Example: `wlanVHTConfig`

Output Arguments

y — L-LTF time-domain waveform
matrix

"L-LTF" on page 1-126 time-domain waveform, returned as an N_S-by-N_T matrix. N_S is the number of time-domain samples, and N_T is the number of transmit antennas.

N_S is proportional to the channel bandwidth. The time-domain waveform consists of two symbols. Each symbol contains 80 time samples per 20 MHz channel.

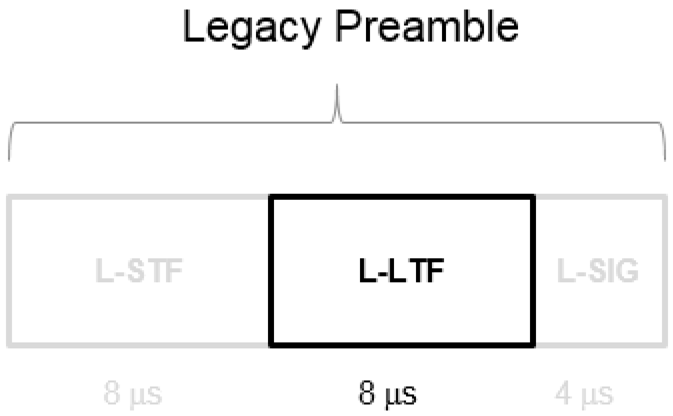
`ChannelBandwidth`	N_S
'CBW20'	160
'CBW40'	320
'CBW80'	640
'CBW160'	1280

Data Types: `double`
Complex Number Support: Yes

More About

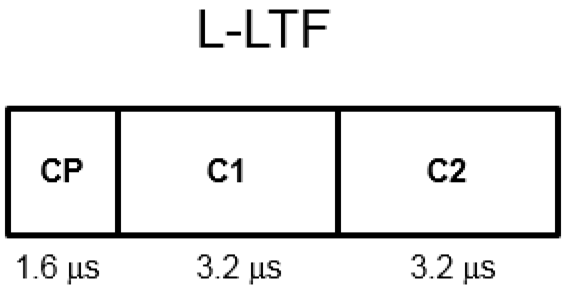
L-LTF

The legacy long training field (L-LTF) is the second field in the legacy preamble for 802.11a/g.



It is also the second field of the HT-mixed format preamble used in 802.11n and the VHT format preamble used in 802.11ac. Channel estimation, frequency offset estimation, and time synchronization rely on the L-LTF. The long OFDM training symbol consists of 53 subcarriers. IEEE Std 802.11-2012, Equation 18-8 and Equation 18-9 define the L-LTF.

The L-LTF, which is 8 µs in length, is composed of a 1.6 µs cyclic prefix (CP) followed by two identical 3.2 µs long training symbols (C1 and C2). The cyclic prefix (CP) consists of the second half of the long training symbol.



Algorithms

The “L-LTF” on page 1-126 is two OFDM symbols long and follows the L-STF of the preamble in the packet structure for the VHT, HT, and non-HT formats. For algorithm details, refer to IEEE Std 802.11ac-2013 [1], Section 22.3.8.2.3 and IEEE Std 802.11-2012 [2], Section 20.3.9.3.4.

References

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

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

See Also

`wlanHTConfig` | `wlanLLTFChannelEstimate` | `wlanLLTFDemodulate` | `wlanLSIG` | `wlanLSTF` | `wlanNonHTConfig` | `wlanVHTConfig`

Introduced in R2015b

wlanLLTFDemodulate

Demodulate L-LTF waveform

Syntax

```
y = wlanLLTFDemodulate(x,cbw)
y = wlanLLTFDemodulate(x,cfg)
y = wlanLLTFDemodulate( ___ ,symOffset)
```

Description

`y = wlanLLTFDemodulate(x,cbw)` returns the demodulated "L-LTF" on page 1-132[14] waveform given time-domain input signal `x` and channel bandwidth `cbw`.

`y = wlanLLTFDemodulate(x,cfg)` returns the demodulated L-LTF given the format configuration object, `cfg`.

`y = wlanLLTFDemodulate( ___ ,symOffset)` specifies the OFDM symbol offset, `symOffset`, using any of the arguments from the previous syntaxes.

Examples

Demodulate L-LTF for Non-HT Format Transmission

Demodulate the L-LTF used in a non-HT OFDM transmission, after passing the L-LTF through an AWGN channel.

Create a non-HT configuration object and use it to generate an L-LTF signal.

```
cfg = wlanNonHTConfig;
txSig = wlanLLTF(cfg);
```

14. IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.

Pass the L-LTF signal through an AWGN channel. Demodulate the received signal.

```
rxSig = awgn(txSig,15,'measured');
y = wlanLLTFDemodulate(rxSig,'CBW20');
```

Demodulate L-LTF for VHT Format Transmission

Demodulate the L-LTF used in a VHT transmission, after passing the L-LTF through an AWGN channel.

Create a VHT configuration object and use it to generate an L-LTF signal.

```
cfg = wlanVHTConfig;
txSig = wlanLLTF(cfg);
```

Pass the L-LTF signal through an AWGN channel.

```
rxSig = awgn(txSig,5);
```

Demodulate the received L-LTF using the information from the `wlanVHTConfig` object.

```
y = wlanLLTFDemodulate(rxSig,cfg);
```

Demodulate L-LTF with OFDM Symbol Offset

Demodulate the L-LTF for the HT-mixed transmission format, given a custom OFDM symbol offset.

Set the channel bandwidth to 40 MHz and the OFDM symbol offset to 1. That way, the FFT takes place after the guard interval.

```
cbw = 'CBW40';
ofdmSymOffset = 1;
```

Create an HT configuration object and use it to generate an L-LTF signal.

```
cfg = wlanHTConfig('ChannelBandwidth',cbw);
txSig = wlanLLTF(cfg);
```

Pass the L-LTF signal through an AWGN channel.

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

Demodulate the received L-LTF using a custom OFDM symbol offset.

```
y = wlanLLTFDemodulate(rxSig,'CBW40',ofdmSymOffset);
```

Input Arguments

`x` — Time-domain input signal

vector | matrix

Time-domain input signal corresponding to the L-LTF of the “PPDU” on page 1-134, specified as an N_S-by-N_R vector or matrix. N_S is the number of samples and N_R is the number of receive antennas.

The number of samples depends on the channel bandwidth.

- N_S = 160 for `'CBW20'`
- N_S = 320 for `'CBW40'`
- N_S = 640 for `'CBW80'`
- N_S = 1280 for `'CBW160'`

Data Types: `double`
Complex Number Support: Yes

`cbw` — Channel bandwidth

`'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth in MHz, specified as one of these strings: `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`.

Data Types: `char`

`cfg` — Format information

`wlanNonHTConfig` | `wlanHTConfig` | `wlanVHTConfig`

Format information, specified as a WLAN configuration object. To create these objects, see `wlanNonHTConfig`, `wlanHTConfig`, or `wlanVHTConfig`.

`symOffset` — OFDM symbol offset

0.75 (default) | real scalar from 0 to 1

OFDM symbol offset as a fraction of the cyclic prefix length, specified as a real scalar from 0 to 1.

Data Types: `double`

Output Arguments

y — Demodulated L-LTF signal
3-D array

Demodulated L-LTF signal, returned as an N_{ST}-by-N_{SYM}-by-N_{R} array. N_{ST} is the number of occupied subcarriers, N_{SYM} is the number of OFDM symbols, and N_{R} is the number of receive antennas. For the L-LTF, N_{SYM} is always 2.

The number of subcarriers depends on the channel bandwidth.

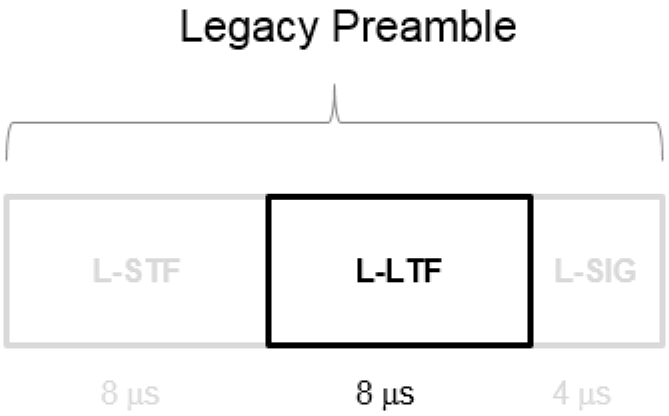
- $N_{\mathrm{ST}} = 52$ for `'CBW20'`
- $N_{\mathrm{ST}} = 104$ for `'CBW40'`
- $N_{\mathrm{ST}} = 208$ for `'CBW80'`
- $N_{\mathrm{ST}} = 416$ for `'CBW160'`

Data Types: `double`
Complex Number Support: Yes

More About

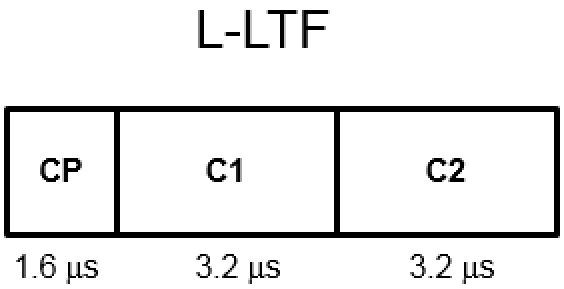
L-LTF

The legacy long training field (L-LTF) is the second field in the legacy preamble for 802.11a/g.



It is also the second field of the HT-mixed format preamble used in 802.11n and the VHT format preamble used in 802.11ac. Channel estimation, frequency offset estimation, and time synchronization rely on the L-LTF. The long OFDM training symbol consists of 53 subcarriers. IEEE Std 802.11-2012, Equation 18-8 and Equation 18-9 define the L-LTF.

The L-LTF, which is 8 µs in length, is composed of a 1.6 µs cyclic prefix (CP) followed by two identical 3.2 µs long training symbols (C1 and C2). The cyclic prefix (CP) consists of the second half of the long training symbol.



PPDU

The PLCP protocol data unit (PPDU) is the complete “PLCP” on page 1-134 frame, including PLCP headers, MAC headers, the MAC data field, and the MAC and PLCP trailers [2].

PLCP

The physical layer convergence procedure (PLCP) is the upper component of the physical layer in 802.11 networks. Each physical layer has its own PLCP, which provides auxiliary framing to the MAC [2].

References

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

[2] Gast, Matthew S. *802.11n: A Survival Guide*. Sebastopol, CA: O’Reilly Media Inc., 2012, p. 120.

See Also

`wlanLLTF` | `wlanLLTFChannelEstimate`

Introduced in R2015b

wlanLSIG

Generate L-SIG waveform

Syntax

```
[y, bits] = wlanLSIG(cfgFormat)
```

Description

`[y, bits] = wlanLSIG(cfgFormat)` generates an “L-SIG” on page 1-139[15] time-domain waveform using the specified configuration object.

Examples

Generate L-SIG Waveform for 80 MHz VHT Packet

Generate the L-SIG waveform for an 80 MHz VHT transmission format packet.

```
cfgVHT = wlanVHTConfig;
cfgVHT.ChannelBandwidth = 'CBW80';
lsigOut = wlanLSIG(cfgVHT);
size(lsigOut)
```

```
ans =

   320     1
```

The L-SIG waveform returned contains one symbol with 320 complex samples for an 80 MHz channel bandwidth.

Extract Rate Information from L-SIG

Create a non-HT configuration object. The default `MCS` is 0.

15. IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.

```
cfg = wlanNonHTConfig
```

```
cfg = 

  wlanNonHTConfig with properties:

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

Generate the L-SIG waveform and information bits. Extract the rate from the returned bits. The rate information is contained in the first four bits.

```
[y,bits] = wlanLSIG(cfg);
rateBits = bits(1:4)
```

```
rateBits =

    1
    1
    0
    1
```

As defined in IEEE Std 802.11™-2012, Table 18-6, a value of `[1 1 0 1]` corresponds to a rate of 6 Mbps for 20 MHz channel spacing.

Change `MCS` to 7 then regenerate the L-SIG waveform and information bits. Extract the rate from the returned bits and analyze. The rate information is contained in the first four bits.

```
cfg.MCS = 7
[y,bits] = wlanLSIG(cfg);

rateBits = bits(1:4)
```

```
cfg = 

  wlanNonHTConfig with properties:
```

```
           Modulation: 'OFDM'
                  MCS: 7
           PSDULength: 1000
    NumTransmitAntennas: 1


rateBits =

    0
    0
    1
    1
```

As defined in IEEE Std 802.11-2012, Table 18-6, a value of `[0 0 1 1]` corresponds to a rate of 54 Mbps for 20 MHz channel spacing.

Input Arguments

`cfgFormat` — Format configuration

`wlanVHTConfig` object | `wlanHTConfig` object | `wlanNonHTConfig` object

Format configuration, specified as a `wlanVHTConfig`, `wlanHTConfig`, or `wlanNonHTConfig` object. For a specified format, the `wlanLSIG` function uses only the object properties indicated.

Transmission Format	Applicable Object Properties
VHT	`ChannelBandwidth`, `NumUsers`, `NumTransmitAntennas`, `NumSpaceTimeStreams`, `STBC`, `MCS`, `ChannelCoding`, `APEPLength`, `GuardInterval`
HT	`ChannelBandwidth`, `NumTransmitAntennas`, `NumSpaceTimeStreams`, `MCS`, `GuardInterval`, `ChannelCoding`, `PSDULength`

Transmission Format	Applicable Object Properties
non-HT (Only OFDM modulation is supported for a `wlanNonHTConfig` object input.)	`Modulation`, `MCS`, `PSDULength`, `NumTransmitAntennas`

Example: `wlanVHTConfig`

Output Arguments

y — L-SIG time-domain waveform

matrix

"L-SIG" on page 1-139 time-domain waveform, returned as an N_S-by-N_T matrix. N_S is the number of time-domain samples, and N_T is the number of transmit antennas.

N_S is proportional to the channel bandwidth. Each symbol contains 80 time samples per 20 MHz channel.

`ChannelBandwidth`	N_S
'CBW20'	80
'CBW40'	160
'CBW80'	320
'CBW160'	640

Data Types: `double`
Complex Number Support: Yes

`bits` — Signaling bits

column vector

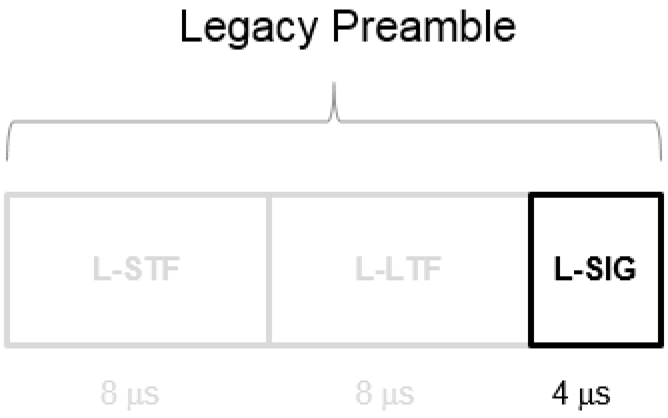
Signaling bits from the legacy signal field, returned as a 24-by-1 bit column vector. See "L-SIG" on page 1-139 for the bit field description.

Data Types: `int8`

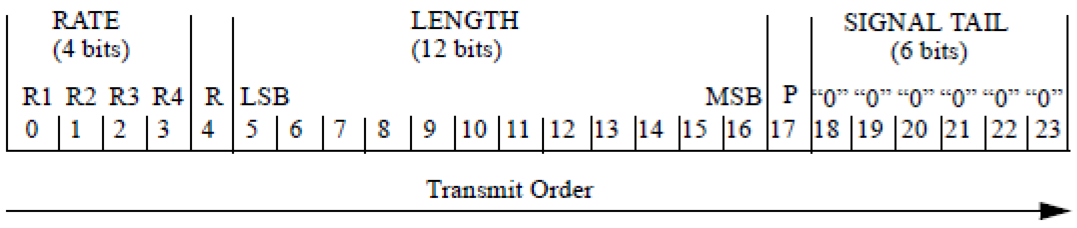
More About

L-SIG

The legacy signal (L-SIG) field is part of the legacy preamble. It consists of 24 bits that contain rate, length, and parity information. The L-SIG is a component of VHT, HT, and non-HT PPDUs. It is transmitted using BPSK modulation with rate 1/2 binary convolutional coding (BCC).



The L-SIG consists of one 4 μs symbol, which is a 3.2 μs OFDM symbol prepended with a 0.8 μs cyclic prefix. It contains packet information for the received configuration,



- Bits 0 through 3 specify the data rate (modulation and coding rate) for the non-HT format.

Rate (bits 0–3)	Modulation	Coding Rate (R)	Data Rate (20 MHz BW) (Mb/s)
1101	BPSK	1/2	6
1111	BPSK	3/4	9
0101	QPSK	1/2	12
0111	QPSK	3/4	18
1001	16-QAM	1/2	24
1011	16-QAM	3/4	36
0001	64-QAM	2/3	48
0011	64-QAM	3/4	54

  For HT and VHT formats, the L-SIG rate bits are set to `'1 1 0 1'`. Data rate information for HT and VHT formats is signaled in format-specific signaling fields.
- Bit 4 is reserved for future use.
- Bits 5 through 16:
  - For non-HT, specify the data length (amount of data transmitted in octets) as described in IEEE Std 802.11-2012, Table 18-1 and Section 9.23.4.
  - For HT-mixed, specify the transmission time as described in IEEE Std 802.11-2012, Section 20.3.9.3.5 and Section 9.23.4.
  - For VHT, specify the transmission time as described in IEEE Std 802.11ac-2013, Section 22.3.8.2.4.
- Bit 17 has the even parity of bits 0 through 16.
- Bits 18 through 23 contain all zeros for the signal tail bits.

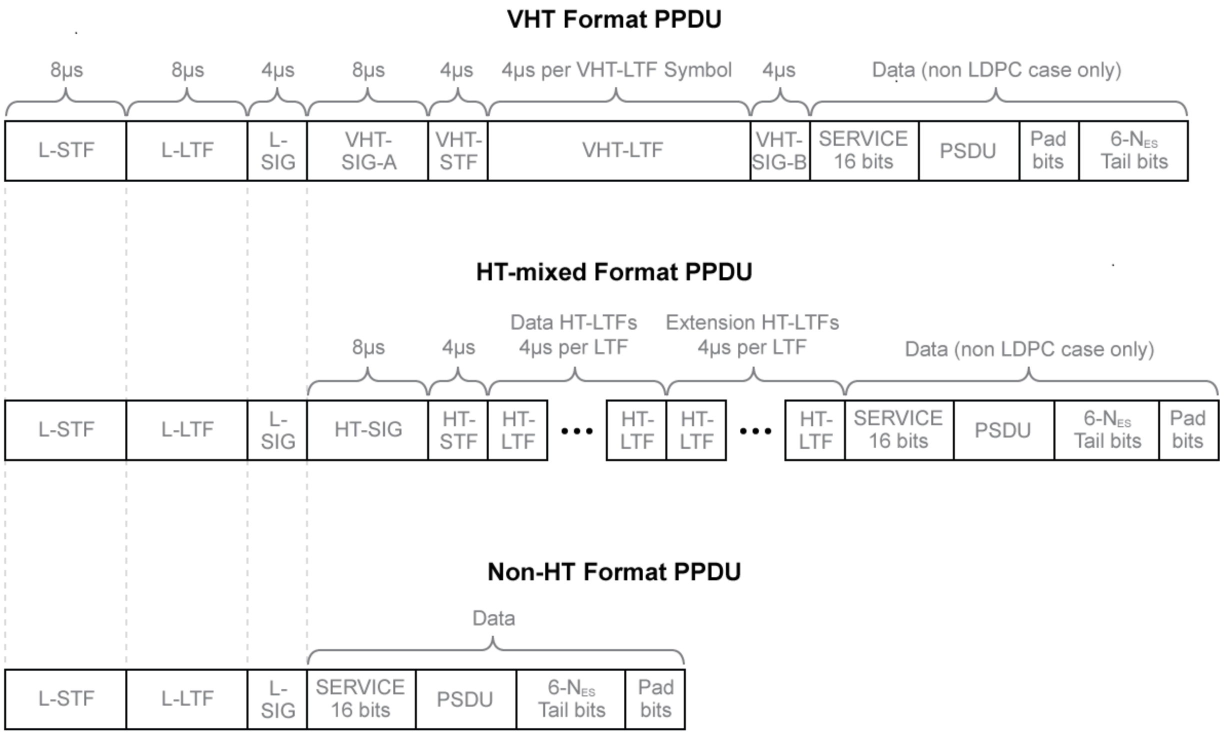
Note: Signaling fields added for HT (`wlanHTSIG`) and VHT (`wlanVHTSIGA`, `wlanVHTSIGB`) formats provide data rate and configuration information for those formats.

- IEEE Std 802.11-2012, Section 20.3.9.4.3 describes HT-SIG bit settings for the HT-mixed format.

- IEEE Std 802.11ac-2013, Section 22.3.8.3.3 and Section 22.3.8.3.6 describe bit settings for VHT-SIG-A and VHT-SIG-B, respectively, for the VHT format.

Algorithms

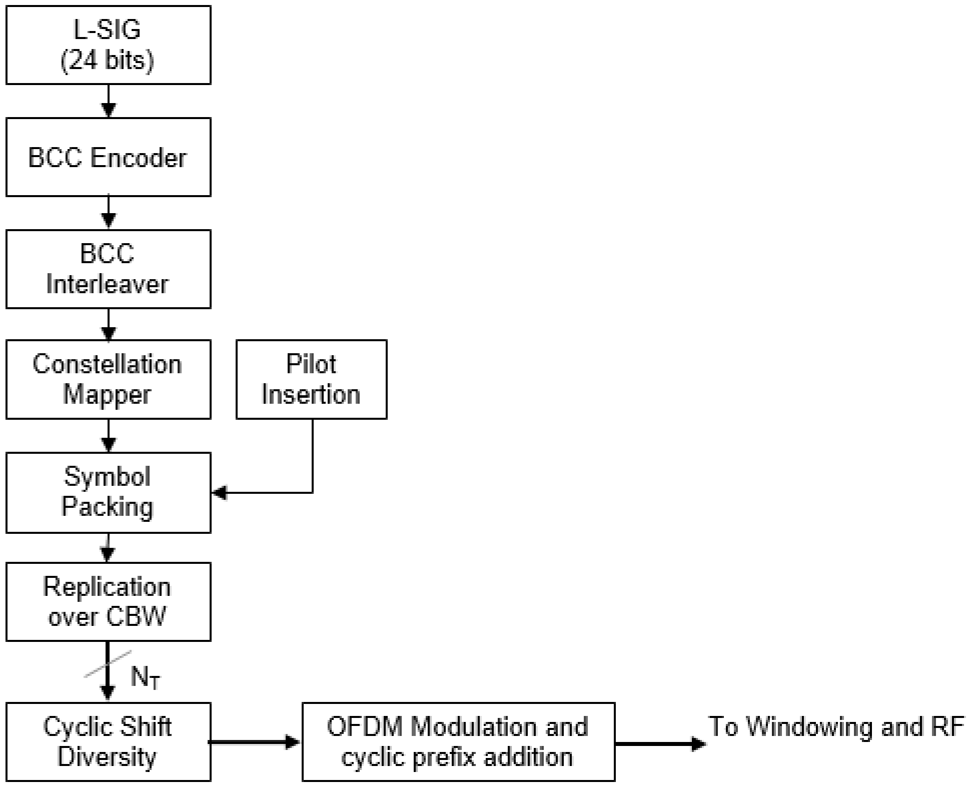
The “L-SIG” on page 1-139 follows the L-STF and L-LTF of the preamble in the packet structure.



For “L-SIG” on page 1-139 transmission processing algorithm details, see:

- VHT format – refer to IEEE Std 802.11ac-2013 [1], Section 22.3.8.2.4
- HT format – refer to IEEE Std 802.11-2012 [2], Sections 20.3.9.3.5
- non-HT format – refer to IEEE Std 802.11-2012 [2], Sections 18.3.4

The `wlanLSIG` function performs transmitter processing on the “L-SIG” on page 1-139 field and outputs the time-domain waveform.



References

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

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

See Also

`wlanHTConfig` | `wlanLLTF` | `wlanLSIGRecover` | `wlanNonHTConfig` | `wlanVHTConfig`

Introduced in R2015b

wlanLSIGRecover

Recover L-SIG information bits

Syntax

```
recBits = wlanLSIGRecover(rxSig,chEst,noiseVarEst,cbw)
recBits = wlanLSIGRecover(rxSig,chEst,noiseVarEst,cbw,cfgRec)
[recBits,failCheck] = wlanLSIGRecover( ___ )
[recBits,failCheck,eqSym] = wlanLSIGRecover( ___ )
[recBits,failCheck,eqSym,cpe] = wlanLSIGRecover( ___ )
```

Description

`recBits = wlanLSIGRecover(rxSig,chEst,noiseVarEst,cbw)` returns the recovered “L-SIG” on page 1-152[16] information bits, `recBits`, given the time-domain L-SIG waveform, `rxSig`. Specify the channel estimate, `chEst`, the noise variance estimate, `noiseVarEst`, and the channel bandwidth, `cbw`.

`recBits = wlanLSIGRecover(rxSig,chEst,noiseVarEst,cbw,cfgRec)` returns information bits and specifies algorithm information using `wlanRecoveryConfig` object `cfgRec`.

`[recBits,failCheck] = wlanLSIGRecover( ___ )` returns the status of a validity check, `failCheck`, using the arguments from previous syntaxes.

`[recBits,failCheck,eqSym] = wlanLSIGRecover( ___ )` returns the equalized symbols, `eqSym`.

`[recBits,failCheck,eqSym,cpe] = wlanLSIGRecover( ___ )` returns the common phase error, `cpe`.

16. IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.

Examples

Recover L-SIG Information from 2x2 MIMO Channel

Recover L-SIG information transmitted in a noisy 2x2 MIMO channel, and calculate the number of bit errors present in the received information bits.

Set the channel bandwidth and sample rate.

```
chanBW = 'CBW40';
fs = 40e6;
```

Create a VHT configuration object corresponding to a 40 MHz 2x2 MIMO channel.

```
vht = wlanVHTConfig( ...
    'ChannelBandwidth',chanBW, ...
    'NumTransmitAntennas',2, ...
    'NumSpaceTimeStreams',2);
```

Generate the L-LTF and L-SIG field signals.

```
txLLTF = wlanLLTF(vht);
[txLSIG,txLSIGData] = wlanLSIG(vht);
```

Pass the signals through a 2x2 multipath fading channel.

```
tgac = wlanTGacChannel('SampleRate',fs,'ChannelBandwidth',chanBW, ...
    'NumTransmitAntennas',2,'NumReceiveAntennas',2, ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
rxLLTFNoNoise = step(tgac,txLLTF);
rxLSIGNoNoise = step(tgac,txLSIG);
```

Create an AWGN channel for a 40 MHz channel with a 9 dB noise figure. The noise variance is equal to $kTBF$, where k is Boltzmann's constant, T is the ambient temperature of 290 K, B is the bandwidth (sample rate), and F is the receiver noise figure.

```
nVar = 10^((-228.6 + 10*log10(290) + 10*log10(fs) + 9)/10);
chAWGN = comm.AWGNChannel('NoiseMethod','Variance','Variance',nVar);
```

Pass the signal from the TGac channel through the AWGN channel.

```
rxLLTF = step(chAWGN,rxLLTFNoNoise);
```

```
rxLSIG = step(chAWGN,rxLSIGNoNoise);
```

Perform channel estimation based on the L-LTF.

```
demodLLTF = wlanLLTFDemodulate(rxLLTF,chanBW,1);
chanEst = wlanLLTFChannelEstimate(demodLLTF,chanBW);
```

Recover the L-SIG information bits.

```
rxLSIGData = wlanLSIGRecover(rxLSIG,chanEst,0.1,chanBW);
```

Verify that there are no bit errors in the recovered L-SIG data.

```
numErrors = biterr(txLSIGData,rxLSIGData)
```

```
numErrors =

     0
```

Recover L-SIG with Zero Forcing Equalizer

Recover L-SIG information using the zero-forcing equalizer algorithm. Calculate the number of bit errors in the received data.

Create an HT configuration object.

```
cfgHT = wlanHTConfig;
```

Create a recovery object with `EqualizationMethod` property set to zero forcing, `'ZF'`.

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

Generate the L-SIG field and pass it through an AWGN channel.

```
[txLSIG,txLSIGData] = wlanLSIG(cfgHT);
rxLSIG = awgn(txLSIG,20);
```

Recover the L-SIG using the zero-forcing algorithm set in `cfgRec`. The channel estimate is a vector of ones because fading was not introduced.

```
rxLSIGData = wlanLSIGRecover(rxLSIG,ones(52,1),0.01,'CBW20',cfgRec);
```

Verify that there are no bit errors in the recovered L-SIG data.

```
numErrors = biterr(txLSIGData,rxLSIGData)
```

```
numErrors =

     0
```

Recover L-SIG from Phase and Frequency Offset

Recover the L-SIG from a channel that introduces a fixed phase and frequency offset.

Create a VHT configuration object corresponding to a 160 MHz SISO channel. Generate the transmitted L-SIG field.

```
cfgVHT = wlanVHTConfig('ChannelBandwidth','CBW160');
txLSIG = wlanLSIG(cfgVHT);
```

Create a recovery configuration object and disable pilot phase tracking.

```
cfgRec = wlanRecoveryConfig('PilotPhaseTracking','None');
```

To introduce a 45 degree phase offset and a 100 Hz frequency offset, create a phase and frequency offset System object.

```
pfo = comm.PhaseFrequencyOffset('SampleRate',160e6,'PhaseOffset',45, ...
    'FrequencyOffset',100);
```

Introduce phase and frequency offsets to the transmitted L-SIG. Pass the L-SIG through an AWGN channel.

```
rxLSIG = awgn(step(pfo,txLSIG),20);
```

Recover the L-SIG information bits, the failure check status, and the equalized symbols.

```
[recLSIGData,failCheck,eqSym] = wlanLSIGRecover(rxLSIG,ones(416,1),0.01,'CBW160',cfgRec
```

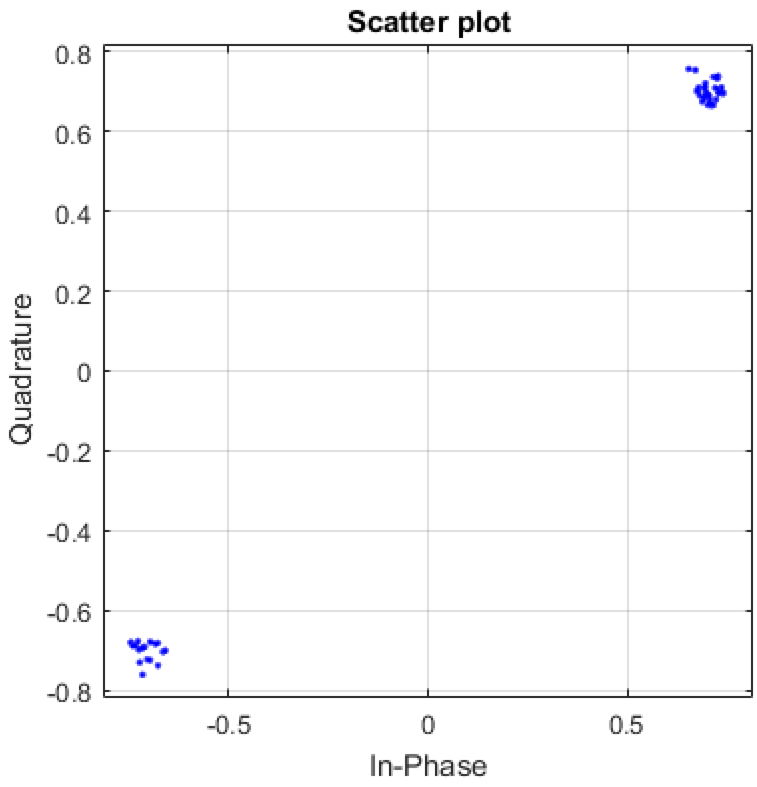
Verify that the L-SIG passed the failure checks.

```
failCheck
```

```
failCheck =
0
```

Plot the equalized symbols. The 45 degree phase offset is visible.

```
scatterplot(eqSym)
grid
```



Input Arguments

`rxSig` — Received L-SIG field
vector | matrix

Received L-SIG field, specified as an N_S-by-N_R matrix. N_S is the number of samples, and N_R is the number of receive antennas.

Channel Bandwidth	N_S
`'CBW20'`	80
`'CBW40'`	160
`'CBW80'`	320
`'CBW160'`	640

Data Types: `double`
Complex Number Support: Yes

`chEst` — Channel estimate

vector | 3-D array

Channel estimate, specified as an N_{ST}-by-1-by-N_R array. N_{ST} is the number of occupied subcarriers, and N_R is the number of receive antennas.

Channel Bandwidth	N_S
`'CBW20'`	52
`'CBW40'`	104
`'CBW80'`	208
`'CBW160'`	416

Data Types: `double`
Complex Number Support: Yes

`noiseVarEst` — Noise variance estimate

nonnegative scalar

Noise variance estimate, specified as a nonnegative scalar.

Data Types: `double`

`cbw` — Channel bandwidth

`'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth in MHz, specified as one of these strings: `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`.

Example: `'CBW80'` corresponds to a channel bandwidth of 80 MHz

Data Types: `char`

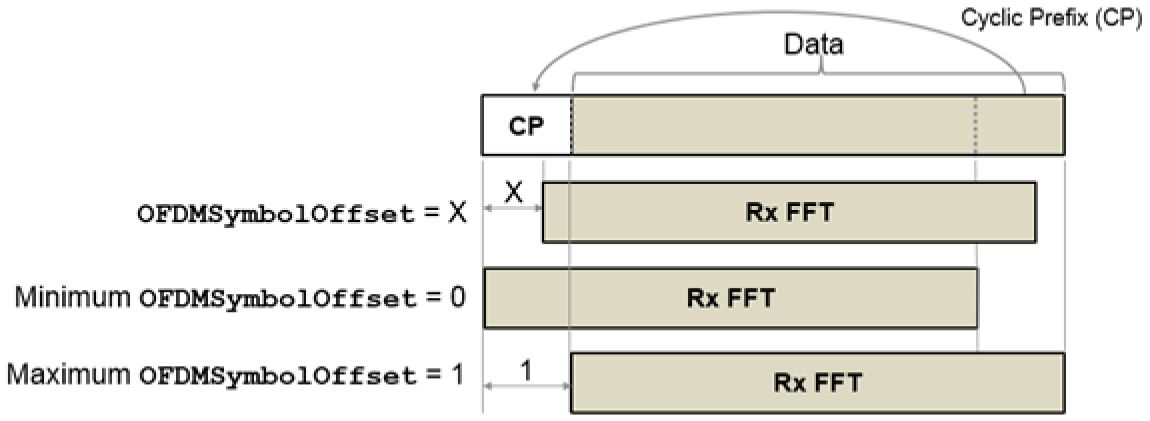
`cfgRec` — Algorithm parameters
`wlanRecoveryConfig` object

Algorithm parameters, specified as a `wlanRecoveryConfig` object. The function uses these properties:

Note: If `cfgRec` is not provided, the function uses the default values of the `wlanRecoveryConfig` object.

`OFDMSymbolOffset` — OFDM symbol sampling offset
0.75 (default) | scalar value from 0 to 1

OFDM symbol sampling offset represented as a fraction of the cyclic prefix (CP) length, specified as a scalar value from 0 to 1. This value indicates the start location for OFDM demodulation, relative to the beginning of the cyclic prefix. `OFDMSymbolOffset` = 0 represents the start of the cyclic prefix and `OFDMSymbolOffset` = 1 represents the end of the cyclic prefix.



Data Types: `double`

`EqualizationMethod` — Equalization method

`'MMSE'` (default) | `'ZF'`

Equalization method, specified as `'MMSE'` or `'ZF'`.

- `'MMSE'` indicates that the receiver uses a minimum mean square error equalizer.
- `'ZF'` indicates that the receiver uses a zero-forcing equalizer.

Example: `'ZF'`

Data Types: `char`

`PilotPhaseTracking` — Pilot phase tracking

`'PreEQ'` (default) | `'None'`

Pilot phase tracking, specified as either of these strings: `'PreEQ'` or `'None'`. If `'PreEQ'` is specified, pilot phase tracking is enabled and is performed prior to any equalization operation. If `'None'` is specified, pilot phase tracking does not occur.

Data Types: `char`

Output Arguments

`recBits` — Recovered L-SIG information

binary vector

Recovered L-SIG information bits, returned as a 24-element column vector containing binary data. The 24 elements correspond to the length of the L-SIG field.

Data Types: `int8`

`failCheck` — Failure check status

`true` | `false`

Failure check status, returned as a logical scalar. If L-SIG fails the parity check, or if its first four bits do not correspond to one of the eight allowable data rates, `failCheck` is `true`.

Data Types: `logical`

`eqSym` — Equalized symbols

vector

Equalized symbols, returned as 48-by-1 vector. There are 48 data subcarriers in the L-SIG field.

Data Types: `double`
Complex Number Support: Yes

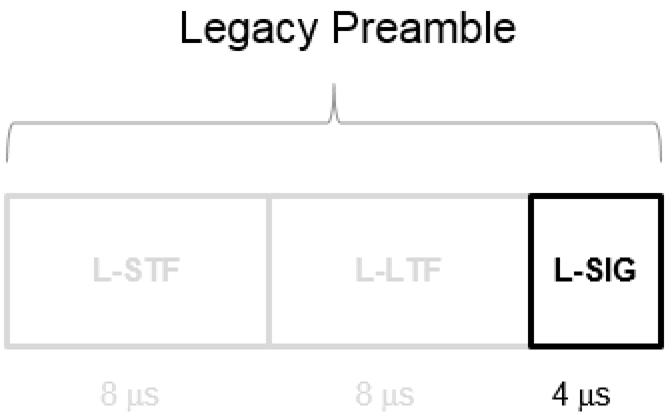
`cpe` — Common phase error
column vector

Common phase error in radians, returned as a scalar.

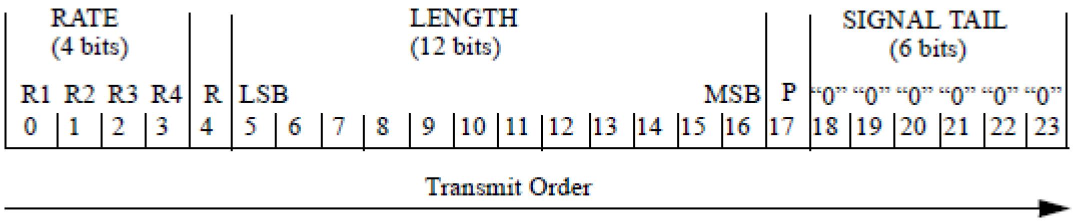
More About

L-SIG

The legacy signal (L-SIG) field is part of the legacy preamble. It consists of 24 bits that contain rate, length, and parity information. The L-SIG is a component of VHT, HT, and non-HT PPDUs. It is transmitted using BPSK modulation with rate 1/2 binary convolutional coding (BCC).



The L-SIG consists of one 4 µs symbol, which is a 3.2 µs OFDM symbol prepended with a 0.8 µs cyclic prefix. It contains packet information for the received configuration,



- Bits 0 through 3 specify the data rate (modulation and coding rate) for the non-HT format.

Rate (bits 0–3)	Modulation	Coding Rate (R)	Data Rate (20 MHz BW) (Mb/s)
1101	BPSK	1/2	6
1111	BPSK	3/4	9
0101	QPSK	1/2	12
0111	QPSK	3/4	18
1001	16-QAM	1/2	24
1011	16-QAM	3/4	36
0001	64-QAM	2/3	48
0011	64-QAM	3/4	54

For HT and VHT formats, the L-SIG rate bits are set to `'1 1 0 1'`. Data rate information for HT and VHT formats is signaled in format-specific signaling fields.

- Bit 4 is reserved for future use.
- Bits 5 through 16:
  - For non-HT, specify the data length (amount of data transmitted in octets) as described in IEEE Std 802.11-2012, Table 18-1 and Section 9.23.4.
  - For HT-mixed, specify the transmission time as described in IEEE Std 802.11-2012, Section 20.3.9.3.5 and Section 9.23.4.
  - For VHT, specify the transmission time as described in IEEE Std 802.11ac-2013, Section 22.3.8.2.4.

- Bit 17 has the even parity of bits 0 through 16.
- Bits 18 through 23 contain all zeros for the signal tail bits.

Note: Signaling fields added for HT (`wlanHTSIG`) and VHT (`wlanVHTSIGA`, `wlanVHTSIGB`) formats provide data rate and configuration information for those formats.

- IEEE Std 802.11-2012, Section 20.3.9.4.3 describes HT-SIG bit settings for the HT-mixed format.
- IEEE Std 802.11ac-2013, Section 22.3.8.3.3 and Section 22.3.8.3.6 describe bit settings for VHT-SIG-A and VHT-SIG-B, respectively, for the VHT format.

References

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

See Also

`wlanLLTF` | `wlanLLTFChannelEstimate` | `wlanLLTFDemodulate` | `wlanLSIG`

Introduced in R2015b

wlanLSTF

Generate L-STF waveform

Syntax

```
y = wlanLSTF(cfg)
```

Description

`y = wlanLSTF(cfg)` generates an “L-STF” on page 1-157[17] time-domain waveform using the specified configuration object.

Examples

Generate L-STF Waveform

Generate the L-STF waveform for a 40 MHz single antenna VHT packet.

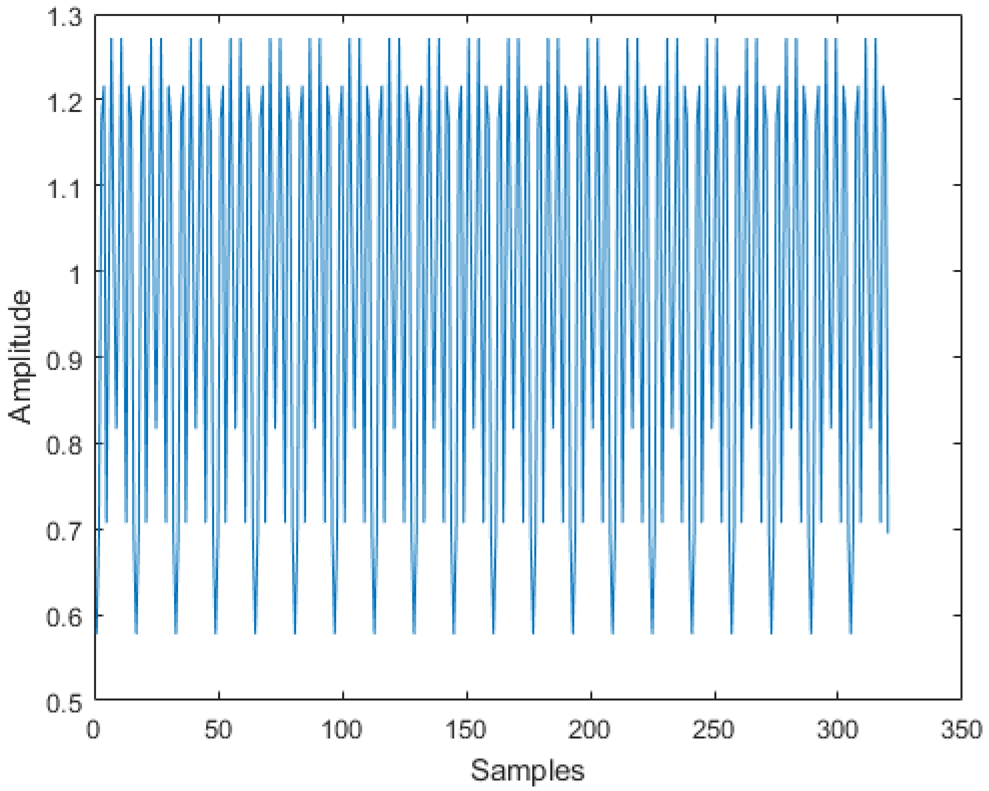
Create a VHT configuration object. Use this object to generate the L-STF waveform.

```
cfgVHT = wlanVHTConfig('ChannelBandwidth','CBW40');
y = wlanLSTF(cfgVHT);
size(y)
plot(abs(y));
xlabel('Samples');
ylabel('Amplitude');
```

```
ans =

   320     1
```

17. IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.



The output L-STF waveform contains 320 samples for a 40 MHz channel bandwidth.

Input Arguments

`cfg` — Format configuration

`wlanVHTConfig` object | `wlanHTConfig` object | `wlanNonHTConfig` object

Format configuration, specified as a `wlanVHTConfig`, `wlanHTConfig`, or `wlanNonHTConfig` object. For a specified format, the `wlanLSTF` function uses only the object properties indicated.

Transmission Format	Applicable Object Properties
VHT	`ChannelBandwidth`, `numTransmitAntennas`
HT	`ChannelBandwidth`, `numTransmitAntennas`
non-HT	`numTransmitAntennas`

Example: `wlanVHTConfig`

Output Arguments

y — L-STF time-domain waveform

matrix

("L-STF" on page 1-157) time-domain waveform, returned as an N_S-by-N_T matrix. N_S is the number of time-domain samples, and N_T is the number of transmit antennas.

N_S is proportional to the channel bandwidth. The time-domain waveform consists of two symbols. Each symbol contains 80 time samples per 20 MHz channel.

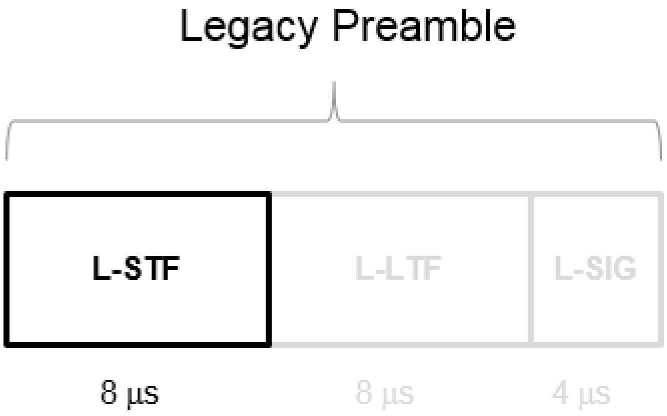
`ChannelBandwidth`	N_S
'CBW20'	160
'CBW40'	320
'CBW80'	640
'CBW160'	1280

Data Types: `double`
Complex Number Support: Yes

More About

L-STF

The legacy short training field (L-STF) is the first field of the legacy preamble for 802.11a/g.



It is also the first field of the HT-mixed format preamble used in 802.11n and the VHT format preamble used in 802.11ac. The L-STF is 8 µs in length and consists of 10 repeated 0.8 µs symbols. Because the sequence has good correlation properties, it is used for start-of-packet detection, for coarse frequency correction, and for setting the AGC. The sequence uses 12 of the 52 subcarriers that are available in 20 MHz. IEEE Std 802.11-2012, Equation 18-6 and Equation 18-7 define the L-STF.

Algorithms

The “L-STF” on page 1-157 is two OFDM symbols long and is the first field in the packet structure for the VHT, HT, and non-HT OFDM formats. For algorithm details, refer to IEEE Std 802.11ac-2013 [1], Section 22.3.8.2.2.

References

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

See Also

`wlanHTConfig` | `wlanLLTF` | `wlanNonHTConfig` | `wlanVHTConfig`

Introduced in R2015b

wlanNonHTConfig

Create non-HT format configuration object

Syntax

```
cfgNonHT = wlanNonHTConfig
cfgNonHT = wlanNonHTConfig(Name,Value)
```

Description

`cfgNonHT = wlanNonHTConfig` creates a configuration object that initializes parameters for an IEEE 802.11 non-high throughput (non-HT) format "PPDU" on page 1-164.

`cfgNonHT = wlanNonHTConfig(Name,Value)` creates a non-HT format configuration object that overrides the default settings using one or more `Name,Value` pair arguments.

At runtime, the calling function validates object settings for properties relevant to the operation of the function.

Examples

Create Non-HT Configuration Object with Default Settings

Create a non-HT configuration object with default settings.

```
cfgNHT = wlanHTConfig
```

```
cfgNHT = 

  wlanHTConfig with properties:

       ChannelBandwidth: 'CBW20'
    NumTransmitAntennas: 1
    NumSpaceTimeStreams: 1
         SpatialMapping: 'Direct'
```

```
                    MCS: 0
          GuardInterval: 'Long'
          ChannelCoding: 'BCC'
             PSDULength: 1024
     RecommendSmoothing: true
```

Update the number of transmit antennas to two.

```
cfgNHT.NumTransmitAntennas = 2
```

```
cfgNHT = 

  wlanHTConfig with properties:

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

Create Non-HT Format Configuration Object

Create a `wlanNonHTConfig` object for OFDM operation for a PSDU length of 2048 bytes.

```
cfgNHT = wlanNonHTConfig('Modulation','OFDM');
cfgNHT.PSDULength = 2048;
cfgNHT
```

```
cfgNHT = 

  wlanNonHTConfig with properties:

             Modulation: 'OFDM'
                    MCS: 0
             PSDULength: 2048
    NumTransmitAntennas: 1
```

Input Arguments

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Modulation','OFDM','MCS',7` specifies OFDM modulation with a modulation and coding scheme of 7, which assigns 64QAM and a 3/4 coding rate for the non-HT format packet.

`'Modulation'` — Modulation type for non-HT transmission
`'OFDM'` (default) | `'DSSS'`

Modulation type for the non-HT transmission packet, specified as `'OFDM'` or `'DSSS'`.

Data Types: `char`

`'MCS'` — OFDM modulation and coding scheme
0 (default) | integer from 0 to 7 | integer

OFDM modulation and coding scheme to use for transmitting the current packet, specified as an integer from 0 to 7. The system configuration associated with an `MCS` setting maps to the specified data rate.

MCS	Modulation	Coding Rate	Data Rate (Mbps)
0	BPSK	1/2	6
1	BPSK	3/4	9
2	QPSK	1/2	12
3	QPSK	3/4	18
4	16QAM	1/2	24
5	16QAM	3/4	36
6	64QAM	2/3	48
7	64QAM	3/4	54

See IEEE Std 802.11-2012, Table 18-4.

Data Types: `double`

`'DataRate'` — DSSS modulation data rate

`'1Mbps'` (default) | `'2Mbps'` | `'5.5Mbps'` | `'11Mbps'`

DSSS modulation data rate, specified as `'1Mbps'`, `'2Mbps'`, `'5.5Mbps'`, or `'11Mbps'`.

- `'1Mbps'` uses differential binary phase shift keying (DBPSK) modulation with a 1 Mbps data rate.
- `'2Mbps'` uses differential quadrature phase shift keying (DQPSK) modulation with a 2 Mbps data rate.
- `'5.5Mbps'` uses complementary code keying (CCK) modulation with a 5.5 Mbps data rate.
- `'11Mbps'` uses complementary code keying (CCK) modulation with an 11 Mbps data rate.

For IEEE Std 802.11-2012, Section 16, only `'1Mbps'` and `'2Mbps'` apply

Data Types: `char`

`'Preamble'` — Preamble type for DSSS modulation

`'Long'` (default) | `'Short'`

Preamble type for DSSS modulation, specified as `'Long'` or `'Short'`.

- When `DataRate` is `'1Mbps'`, the `Preamble` setting is ignored and `'Long'` is used.
- When `DataRate` is greater than `'1Mbps'`, the `Preamble` property is available and can be set to `'Long'` or `'Short'`.

For IEEE Std 802.11-2012, Section 16, `'Short'` does not apply.

Data Types: `char`

`'LockedClocks'` — Clock locking indication for DSSS modulation

true (default) | false

Clock locking indication for DSSS modulation, specified as a logical. Bit 2 of the SERVICE field is the *Locked Clock Bit*. A `true` setting indicates that the PHY implementation derives its transmit frequency clock and symbol clock from the same oscillator. For more information, see IEEE Std 802.11-2012, Section 17.2.3.5 and Section 19.1.3.

Note:

- IEEE Std 802.11-2012, Section 19.3.2.2, specifies locked clocks is required for all ERP systems when transmitting at the ERP-PBCC rate or at a data rate described in Section 17. Therefore to model ERP systems, set `LockedClocks` to `true`.

Data Types: `logical`

`'PSDULength'` — Number of bytes carried in the user payload
1000 (default) | integer from 1 to 4095 | integer

Number of bytes carried in the user payload, specified as an integer from 1 to 4095.

Data Types: `double`

`'NumTransmitAntennas'` — Number of transmit antennas
1 (default) | integer from 1 to 8

Number of transmit antennas, specified as a scalar integer from 1 to 8.

Data Types: `double`

Output Arguments

`cfgNonHT` — Non-HT PPDU configuration
`wlanNonHTConfig` object

Non-HT “PPDU” on page 1-164 configuration, returned as a `wlanNonHTConfig` object. The properties of `cfgNonHT` are specified in wlanNonHTConfig Properties.

More About

PPDU

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

References

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

See Also

`wlanHTConfig` | `wlanVHTConfig` | `wlanWaveformGenerator`

Introduced in R2015b

wlanNonHTData

Generate non-HT-Data field waveform

Syntax

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

Description

`y = wlanNonHTData(psdu,cfg)` generates the “non-HT-Data field” on page 1-169[18] time-domain waveform for the input “PSDU” on page 1-169 bits.

`y = wlanNonHTData(psdu,cfg,scramInit)` uses `scramInit` for the scrambler initialization state.

Examples

Generate Non-HT-Data Waveform

Generate the waveform for a 20MHz non-HT-Data field for 36 Mbps.

Create a non-HT configuration object and assign `MCS` to 5.

```
cfg = wlanNonHTConfig('MCS',5);
```

Assign random data to the PSDU and generate the data field waveform.

```
psdu = randi([0 1],cfg.PSDULength*8,1);
y = wlanNonHTData(psdu,cfg);
size(y)
```

```
ans =

        4480           1
```

18. IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.

Input Arguments

`psdu` — PLCP service data unit
vector

PLCP service data unit ("PSDU" on page 1-169), specified as an N_{bits}*-by-1* vector, where N_{bits} = *PSDULength*8*. "PSDU" on page 1-169 vector can range from 1 byte to 4095 bytes, as specified by `PSDULength`.

Data Types: `double`

`cfg` — Format configuration
`wlanNonHTConfig` object

Format configuration, specified as a `wlanNonHTConfig` object. The `wlanNonHTData` function uses the `wlanNonHTConfig` object properties associated with the `'OFDM'` setting for `Modulation`.

Non-HT Format Configuration

`Modulation` — Modulation type for non-HT transmission
`'OFDM'` (default) | `'DSSS'`

Modulation type for the non-HT transmission packet, specified as `'OFDM'` or `'DSSS'`.

Data Types: `char`

`MCS` — OFDM modulation and coding scheme
0 (default) | integer from 0 to 7 | integer

OFDM modulation and coding scheme to use for transmitting the current packet, specified as an integer from 0 to 7. The system configuration associated with an `MCS` setting maps to the specified data rate.

MCS	Modulation	Coding Rate	Data Rate (Mbps)
0	BPSK	1/2	6
1	BPSK	3/4	9
2	QPSK	1/2	12

MCS	Modulation	Coding Rate	Data Rate (Mbps)
3	QPSK	3/4	18
4	16QAM	1/2	24
5	16QAM	3/4	36
6	64QAM	2/3	48
7	64QAM	3/4	54

See IEEE Std 802.11-2012, Table 18-4.

Data Types: `double`

`PSDULength` — Number of bytes carried in the user payload
1000 (default) | integer from 1 to 4095 | integer

Number of bytes carried in the user payload, specified as an integer from 1 to 4095.

Data Types: `double`

`NumTransmitAntennas` — Number of transmit antennas
1 (default) | integer from 1 to 8

Number of transmit antennas, specified as a scalar integer from 1 to 8.

Data Types: `double`

`scramInit` — Scrambler initialization state
93 (default) | integer from 1 to 127 | binary column vector

Scrambler initialization state for each packet generated, specified as an integer from 1 to 127 or as the corresponding binary column vector of length seven. The default value of 93 is the example state given in IEEE Std 802.11-2012, Section L.1.5.2.

Example: `[1; 0; 1; 1; 1; 0; 1]` conveys the scrambler initialization state of 93 as a binary column vector.

Data Types: `double` | `int8`

Output Arguments

y — Non-HT-Data field time-domain waveform
matrix

Non-HT-Data field time-domain waveform, returned as an N_S-by-N_T matrix. N_S is the number of time domain samples, and N_T is the number of transmit antennas.

Data Types: `double`
Complex Number Support: Yes

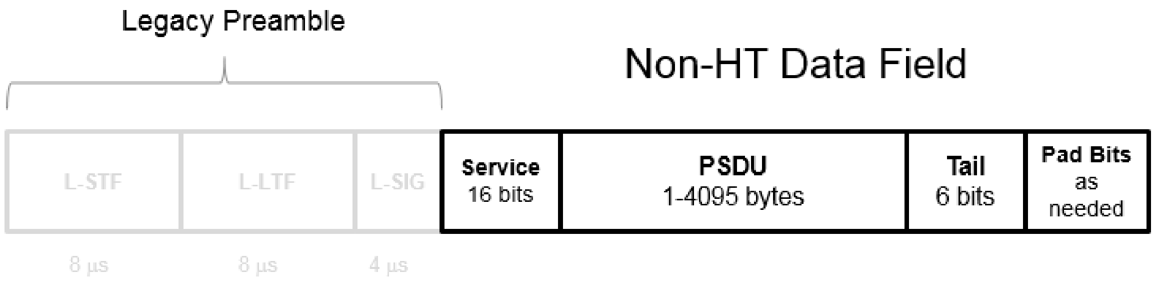
More About

PSDU

Physical layer convergence procedure (PLCP) service data unit (PSDU). This field is composed of a variable number of octets. The minimum is 0 (zero) and the maximum is 2500. For more information see IEEE Std 802.11™-2012, Section 15.3.5.7.

non-HT-Data field

The non-high throughput data (non-HT data) field is used to transmit MAC frames and is composed of a service field, a PSDU, tail bits, and pad bits.

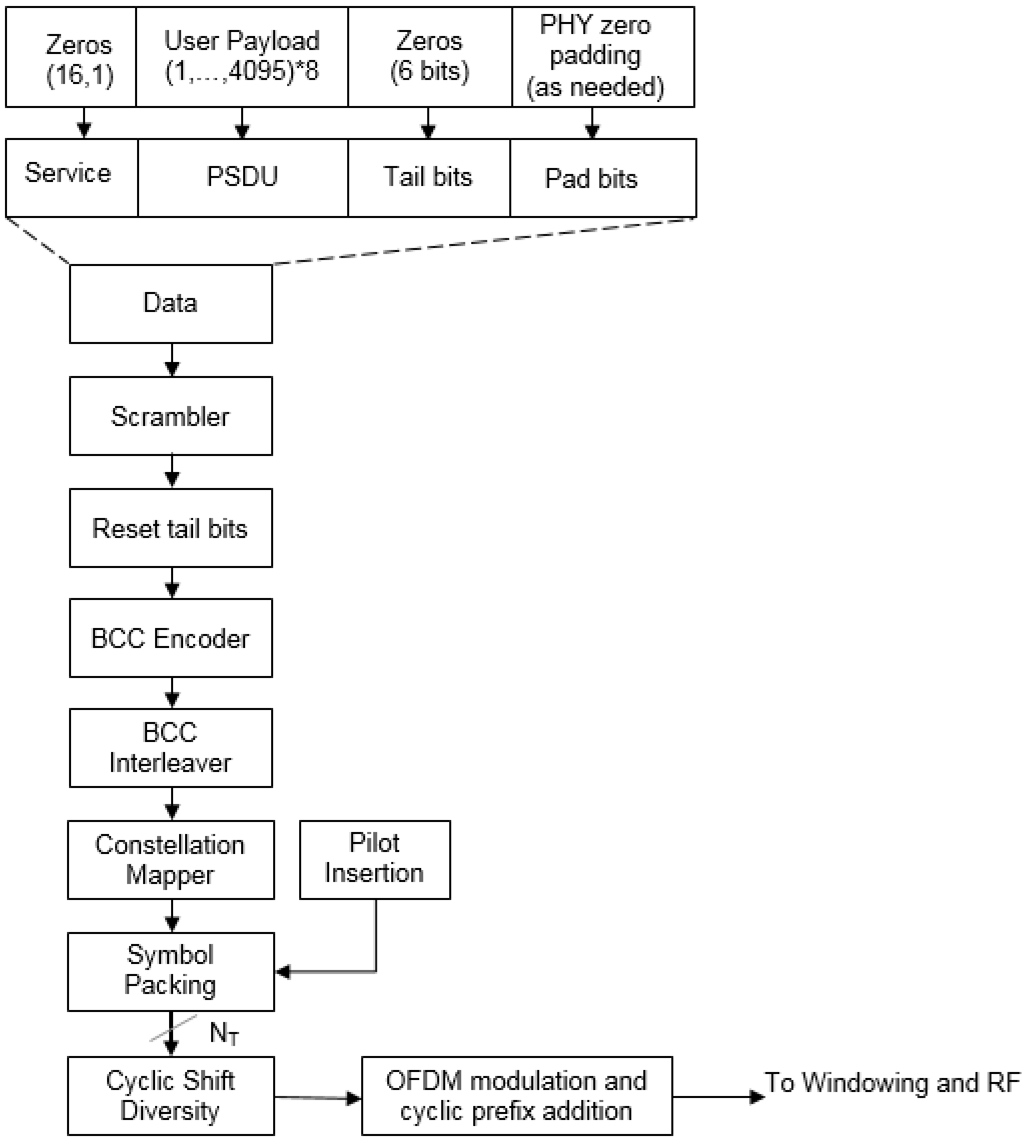


- **Service field** — Contains 16 zeros to initialize the data scrambler.
- **PSDU** — Variable-length field containing the PLCP service data unit (PSDU).
- **Tail** — Tail bits required to terminate a convolutional code. The field uses six zeros for the single encoding stream.
- **Pad Bits** — Variable-length field required to ensure that the non-HT data field contains an integer number of symbols.

Algorithms

non-HT-Data Field Processing

The “non-HT-Data field” on page 1-169 follows the L-SIG in the packet structure. For algorithm details, refer to IEEE Std 802.11-2012 [1], Section 18.3.5. The “non-HT-Data field” on page 1-169 includes the user payload in the *PSDU* plus 16 service bits, 6 tail bits, and additional padding bits as required to fill out the last OFDM symbol. The `wlanNonHTData` function performs transmitter processing on the “non-HT-Data field” on page 1-169 and outputs the time-domain waveform.


Zeros (16,1)
User Payload (1,...,4095)*8
Zeros (6 bits)
PHY zero padding (as needed)
Service
PSDU
Tail bits
Pad bits
Data
Scrambler
Reset tail bits
BCC Encoder
BCC Interleaver
Constellation Mapper
Pilot Insertion
Symbol Packing
N_T
Cyclic Shift Diversity
OFDM modulation and cyclic prefix addition
To Windowing and RF

References

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

See Also

`wlanLSIG` | `wlanNonHTConfig` | `wlanNonHTDataRecover`

Introduced in R2015b

wlanNonHTDataRecover

Recover non-HT data

Syntax

```
recData = wlanNonHTDataRecover(rxSig,chEst,noiseVarEst,cfg)
recData = wlanNonHTDataRecover(rxSig,chEst,noiseVarEst,cfg,cfgRec)
[recData,eqSym] = wlanNonHTDataRecover( ___ )
[recData,eqSym,cpe] = wlanNonHTDataRecover( ___ )
```

Description

`recData = wlanNonHTDataRecover(rxSig,chEst,noiseVarEst,cfg)` returns the recovered "Non-HT-Data field" on page 1-179[19] bits, given received signal `rxSig`, channel estimate data `chEst`, noise variance estimate `noiseVarEst`, and `wlanNonHTConfig` object `cfg`.

`recData = wlanNonHTDataRecover(rxSig,chEst,noiseVarEst,cfg,cfgRec)` specifies the recovery algorithm parameters using `wlanRecoveryConfig` object `cfgRec`.

`[recData,eqSym] = wlanNonHTDataRecover( ___ )` returns the equalized symbols, `eqSym`, using the arguments from the previous syntaxes.

`[recData,eqSym,cpe] = wlanNonHTDataRecover( ___ )` also returns the common phase error, `cpe`.

Examples

Recover Non-HT Data Bits

Create a non-HT configuration object having a PSDU length of 2048 bytes. Generate the corresponding data sequence.

19. IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.

```
cfg = wlanNonHTConfig('PSDULength',2048);
txBits = randi([0 1],8*cfg.PSDULength,1);
txSig = wlanNonHTData(txBits,cfg);
```

Pass the signal through an AWGN channel with a signal-to-noise ratio of 15 dB.

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

Recover the data and determine the number of bit errors.

```
rxBits = wlanNonHTDataRecover(rxSig,ones(52,1),0.05,cfg);
[numerr,ber] = biterr(rxBits,txBits)
```

```
numerr =

     0


ber =

     0
```

Recover Non-HT Data Bits Using Zero-Forcing Algorithm

Create a non-HT configuration object having a 1024-byte PSDU length. Generate the corresponding non-HT data sequence.

```
cfg = wlanNonHTConfig('PSDULength',1024);
txBits = randi([0 1],8*cfg.PSDULength,1);
txSig = wlanNonHTData(txBits,cfg);
```

Pass the signal through an AWGN channel with a signal-to-noise ratio of 10 dB.

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

Create a data recovery object that specifies the use of the zero-forcing algorithm.

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

Recover the data and determine the number of bit errors.

```
rxBits = wlanNonHTDataRecover(rxSig,ones(52,1),0.1,cfg,cfgRec);
[numerr,ber] = biterr(rxBits,txBits)
```

```
numerr =

     0


ber =

     0
```

Recover Non-HT Data in Fading Channel

Configure a non-HT data object.

```
cfg = wlanNonHTConfig;
```

Generate and transmit a non-HT PSDU.

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

Generate an L-LTF for channel estimation.

```
txLLTF = wlanLLTF(cfg);
```

Create an 802.11g channel with a 3 Hz maximum Doppler shift and a 100 ns RMS path delay. Disable the reset before filtering option so that the L-LTF and data fields use the same channel realization.

```
ch802 = stdchan(1/20e6,3,'802.11g',100e-9);
ch802.ResetBeforeFiltering = 0;
```

Pass the L-LTF and data signals through an 802.11g channel with AWGN.

```
rxLLTF = awgn(filter(ch802,txLLTF),10);
rxSig = awgn(filter(ch802,txSig),10);
```

Demodulate the L-LTF and use it to estimate the fading channel.

```
dLLTF = wlanLLTFDemodulate(rxLLTF,cfg);
chEst = wlanLLTFChannelEstimate(dLLTF,cfg);
```

Recover the non-HT data using the L-LTF channel estimate and determine the number of bit errors in the transmitted packet.

```
rxPSDU = wlanNonHTDataRecover(rxSig,chEst,0.1,cfg);

[numErr,ber] = biterr(txPSDU,rxPSDU)


numErr =

     0


ber =

     0
```

Input Arguments

`rxSig` — Received non-HT data signal
vector | matrix

Received non-HT data signal, specified as a matrix of size N_S-by-N_R. N_S is the number of samples and N_R is the number of receive antennas. N_S can be greater than the length of the data field signal.

Data Types: `double`
Complex Number Support: Yes

`chEst` — Channel estimate data
vector | 3-D array

Channel estimate data, specified as an N_{ST}-by-1-by-N_R array. N_{ST} is the number of occupied subcarriers, and N_{R} is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

`noiseVarEst` — Noise variance estimate
nonnegative scalar

Estimate of the noise variance, specified as a nonnegative scalar.

Example: 0.7071

Example: [0.33 0.51 0.11 1.13]

Data Types: `double`

`cfg` — Configure non-HT format parameters
`wlanNonHTConfig` object

Non-HT format configuration, specified as a `wlanNonHTConfig` object. The `wlanHTDataRecover` function uses the following `wlanNonHTConfig` object properties:

`MCS` — OFDM modulation and coding scheme
0 (default) | integer from 0 to 7 | integer

OFDM modulation and coding scheme to use for transmitting the current packet, specified as an integer from 0 to 7. The system configuration associated with an `MCS` setting maps to the specified data rate.

MCS	Modulation	Coding Rate	Data Rate (Mbps)
0	BPSK	1/2	6
1	BPSK	3/4	9
2	QPSK	1/2	12
3	QPSK	3/4	18
4	16QAM	1/2	24
5	16QAM	3/4	36
6	64QAM	2/3	48
7	64QAM	3/4	54

See IEEE Std 802.11-2012, Table 18-4.

Data Types: `double`

`PSDULength` — Number of bytes carried in the user payload
1000 (default) | integer from 1 to 4095 | integer

Number of bytes carried in the user payload, specified as an integer from 1 to 4095.

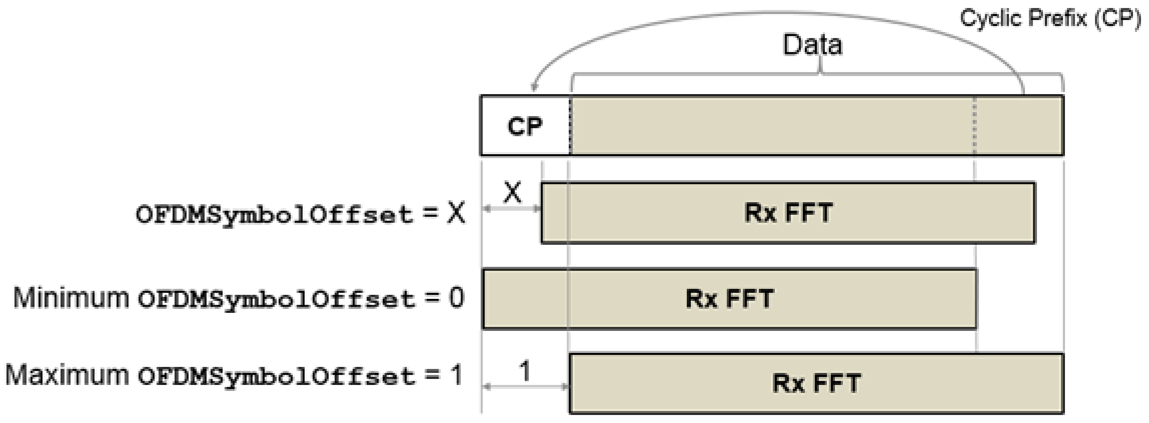
Data Types: `double`

`cfgRec` — Algorithm parameters
`wlanRecoveryConfig` object

Algorithm parameters, specified as a `wlanRecoveryConfig` object. The object properties include:

`OFDMSymbolOffset` — OFDM symbol sampling offset
0.75 (default) | scalar value from 0 to 1

OFDM symbol sampling offset represented as a fraction of the cyclic prefix (CP) length, specified as a scalar value from 0 to 1. This value indicates the start location for OFDM demodulation, relative to the beginning of the cyclic prefix. `OFDMSymbolOffset` = 0 represents the start of the cyclic prefix and `OFDMSymbolOffset` = 1 represents the end of the cyclic prefix.



Data Types: `double`

`EqualizationMethod` — Equalization method
`'MMSE'` (default) | `'ZF'`

Equalization method, specified as `'MMSE'` or `'ZF'`.

- `'MMSE'` indicates that the receiver uses a minimum mean square error equalizer.
- `'ZF'` indicates that the receiver uses a zero-forcing equalizer.

Example: `'ZF'`

Data Types: `char`

`PilotPhaseTracking` — Pilot phase tracking

`'PreEQ'` (default) | `'None'`

Pilot phase tracking, specified as either of these strings: `'PreEQ'` or `'None'`. If `'PreEQ'` is specified, pilot phase tracking is enabled and is performed prior to any equalization operation. If `'None'` is specified, pilot phase tracking does not occur.

Data Types: `char`

Output Arguments

`recData` — Recovered binary output data

binary column vector

Recovered binary output data, returned as a column vector of length 8×N_{PSDU}, where N_{PSDU} is the length of the PSDU in bytes. See wlanNonHTConfig Properties for `PSDULength` details.

Data Types: `int8`

`eqSym` — Equalized symbols

column vector | matrix

Equalized symbols, returned as an N_{SD}-by-N_{SYM} matrix. N_{SD} is the number of data subcarriers, and N_{SYM} is the number of OFDM symbols in the non-HT data field.

Data Types: `double`
Complex Number Support: Yes

`cpe` — Common phase error

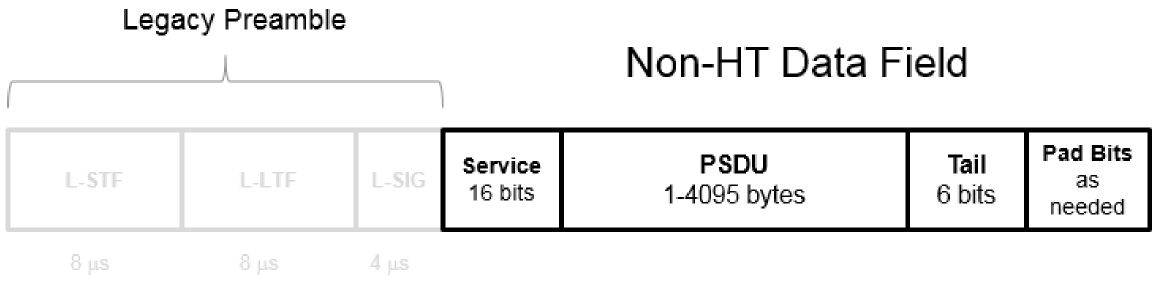
column vector

Common phase error in radians, returned as a column vector having length N_{SYM}. N_{SYM} is the number of OFDM symbols in the "Non-HT-Data field" on page 1-179.

More About

Non-HT-Data field

The non-high throughput data (non-HT data) field is used to transmit MAC frames and is composed of a service field, a PSDU, tail bits, and pad bits.



- **Service field** — Contains 16 zeros to initialize the data scrambler.
- **PSDU** — Variable-length field containing the PLCP service data unit (PSDU).
- **Tail** — Tail bits required to terminate a convolutional code. The field uses six zeros for the single encoding stream.
- **Pad Bits** — Variable-length field required to ensure that the non-HT data field contains an integer number of symbols.

References

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

See Also

`wlanNonHTConfig` | `wlanNonHTData` | `wlanRecoveryConfig`

Introduced in R2015b

wlanPacketDetect

not found

Syntax

Description

Use the Help browser search field to `matlab:docsearch wlanPacketDetect`, or type "`matlab:help help`" for help command options, such as help for methods.

wlanPPDUfield

Extract PPDU fields from received packets

Syntax

```
s = wlanPPDUfield(x,cfg,frmInd)
```

Description

`s = wlanPPDUfield(x,cfg,frmInd)` returns the structure, `s`, containing the individual component fields that comprise the received signal, `x`. Specify the transmission format, `cfg`, and the frame index, `frmInd`.

Examples

Extract PPDU Fields From VHT Waveform

Extract the VHT-STF from a VHT waveform.

Create VHT configuration object for a MIMO transmission using a 160 MHz channel bandwidth.

```
cfg = wlanVHTConfig('MCS',8,'ChannelBandwidth','CBW160','NumTransmitAntennas',2,'NumSpa
```

Configure the waveform generator and generate the corresponding VHT waveform.

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

Determine the component PPDU field indices for the VHT format.

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

```
ind =

       LSTF: [1 1280]
       LLTF: [1281 2560]
       LSIG: [2561 3200]
    VHTSIGA: [3201 4480]
     VHTSTF: [4481 5120]
     VHTLTF: [5121 6400]
    VHTSIGB: [6401 7040]
    VHTData: [7041 8320]
```

The VHT PPDU waveform is comprised of eight fields, including seven preamble fields and one data field.

Extract the VHT-STF from the transmitted waveform.

```
stf = txSig(ind.VHTSTF(1):ind.VHTSTF(2),:);
```

Verify that the VHT-STF has dimensions of 640-by-2 corresponding to the number of samples (80 for each 20 MHz bandwidth segment) and the number of transmit antennas.

```
size(stf)
```

```
ans =

   640     2
```

Input Arguments

x — Received signal

vector | matrix

Received signal, specified as an N_S-by-N_T vector or matrix, where N_S is the number of time-domain samples and N_T is the number of transmit antennas.

Data Types: `double`
Complex Number Support: Yes

`cfg` — Transmission format

`wlanVHTConfig` | `wlanHTConfig` | `wlanNonHTConfig`

Transmission format , specified as a `wlanVHTConfig`, `wlanHTConfig`, or a `wlanNonHTConfig` configuration object.

Example: `txformat` = `wlanVHTConfig`

`frmInd` — Frame index
positive integer

Frame index of the sample offset, specified as a positive integer.

Data Types: `double`

Output Arguments

`s` — Component fields
structure

Component fields of the received signal, returned as a structure. Each component, for example, VHT-LTF, is returned in a field of the structure, `s`.

Data Types: `struct`

References

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

See Also

`wlanFieldExtraction` | `wlanGeneratorConfig` | `wlanHTConfig` | `wlanNonHTConfig` | `wlanVHTConfig` | `wlanWaveformGenerator`

Introduced in R2015b

wlanRecoveryConfig

Create data recovery configuration object

Syntax

```
cfgRec = wlanRecoveryConfig
cfgRec = wlanRecoveryConfig(Name,Value)
```

Description

`cfgRec = wlanRecoveryConfig` creates a configuration object that initializes parameters for use in recovery of signal and data information.

`cfgRec = wlanRecoveryConfig(Name,Value)` creates an information recovery configuration object that overrides the default settings using one or more `Name,Value` pair arguments.

At runtime, the calling function validates object settings for properties relevant to the operation of the function.

Examples

Create wlanRecoveryConfig Object

Create an information recovery configuration object using a Name,Value pairs to update the equalization method and OFDM symbol sampling offset.

```
cfgRec = wlanRecoveryConfig('EqualizationMethod','ZF','OFDMSymbolOffset',0.5)
```

```
cfgRec = 

  wlanRecoveryConfig with properties:

      OFDMSymbolOffset: 0.5
    EqualizationMethod: 'ZF'
    PilotPhaseTracking: 'PreEQ'
```

Input Arguments

Name-Value Pair Arguments

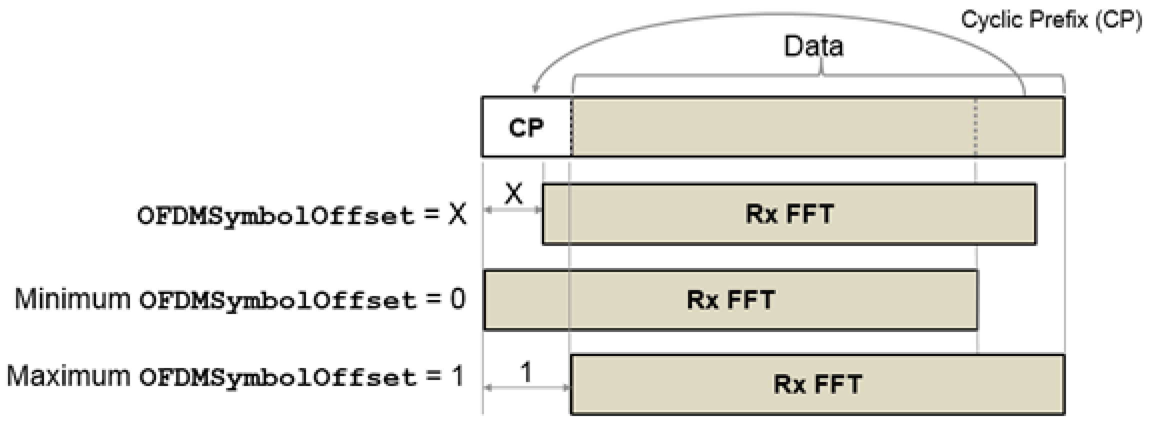
Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'OFDMSymbolOffset',0.25,'EqualizationMethod','ZF'`

`'OFDMSymbolOffset'` — OFDM symbol sampling offset

0.75 (default) | scalar value from 0 to 1

OFDM symbol sampling offset represented as a fraction of the cyclic prefix (CP) length, specified as a scalar value from 0 to 1. This value indicates the start location for OFDM demodulation, relative to the beginning of the cyclic prefix. `OFDMSymbolOffset` = 0 represents the start of the cyclic prefix and `OFDMSymbolOffset` = 1 represents the end of the cyclic prefix.



Data Types: `double`

`'EqualizationMethod'` — Equalization method
`'MMSE'` (default) | `'ZF'`

Equalization method, specified as `'MMSE'` or `'ZF'`.

- `'MMSE'` indicates that the receiver uses a minimum mean square error equalizer.
- `'ZF'` indicates that the receiver uses a zero-forcing equalizer.

Example: `'ZF'`

Data Types: `char`

`'PilotPhaseTracking'` — Pilot phase tracking
`'PreEQ'` (default) | `'None'`

Pilot phase tracking, specified as either of these strings: `'PreEQ'` or `'None'`. If `'PreEQ'` is specified, pilot phase tracking is enabled and is performed prior to any equalization operation. If `'None'` is specified, pilot phase tracking does not occur.

Data Types: `char`

Output Arguments

`cfgRec` — Data recovery configuration
`wlanRecoveryConfig` object

Data recovery configuration, returned as a `wlanRecoveryConfig` object. The properties of `cfgRec` are specified in wlanRecoveryConfig Properties.

See Also

`wlanHTDataRecover` | `wlanHTSIGRecover` | `wlanLSIGRecover` | `wlanNonHTDataRecover` | `wlanVHTDataRecover` | `wlanVHTSIGARecover` | `wlanVHTSIGBRecover`

Introduced in R2015b

wlanSTBCCombine

Perform space-time block code (STBC) combining

Syntax

```
[Y, CSI] = wlanSTBCCombine(X,CHANEST, 'ZF')
[Y, CSI] = wlanSTBCCombine(X,CHANEST, 'MMSE',NOISEVAR)
```

Description

`[Y, CSI] = wlanSTBCCombine(X,CHANEST, 'ZF')` performs the STBC combining using the signal input X and the channel estimation input CHANEST, according to the STBC scheme for VHT format, and returns the estimation of transmitted signal in Y and the soft channel state information in CSI. The zero-forcing (ZF) equalization is performed for the combining. The inputs X and CHANEST can be double precision 2-D matrices or 3-D arrays with real or complex values. X is of size Nsd x Nsym x Nr, where Nsd represents the number of data subcarriers (frequency domain), Nsym represents the number of OFDM symbols (time domain), and Nr represents the number of receive antennas (spatial domain). CHANEST is of size Nsd x Nsts x Nr, where Nsts represents the number of space-time streams which must be an even number. The double precision output Y is of size Nsd x Nsym x Nsts/2. Y is complex when either X or CHANEST is complex and is real otherwise. The double precision output CSI is of size Nsd x Nsts/2.

`[Y, CSI] = wlanSTBCCombine(X,CHANEST, 'MMSE',NOISEVAR)` performs the STBC combining using the minimum-mean-square-error (MMSE) equalization method. The noise variance input NOISEVAR is a double precision, nonnegative, scalar or row vector of length Nr.

Input Arguments

`X` —

(default) |

Example:

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `char` | `struct` | `table` | `cell` | `function_handle`
Complex Number Support: Yes

`CHANEST` —
(default) |

Example:

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `char` | `struct` | `table` | `cell` | `function_handle`
Complex Number Support: Yes

`NOISEVAR` —
(default) |

Example:

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `char` | `struct` | `table` | `cell` | `function_handle`
Complex Number Support: Yes

Output Arguments

`Y` —

`CSI` —

See Also
`wlanSTBCEncode`

wlanSTBCEncode

Perform space-time block coding (STBC)

Syntax

```
Y = wlanSTBCEncode(X)
```

Description

`Y = wlanSTBCEncode(X)` encodes the input X according to the STBC scheme for VHT format, and returns the result in Y. The input X can be a double precision 2-D matrix or 3-D array with real or complex values. X is of size Nsd x Nsym x Nss, where Nsd represents the number of data subcarriers (frequency domain), Nsym represents the number of OFDM symbols (time domain), and Nss represents the number of spatial streams (spatial domain). The output Y is of size Nsd x Nsym x (2Nss). Y has the same data type and complexity as X.

Input Arguments

`X` —
(default) |

Example:

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `char` | `struct` | `table` | `cell` | `function_handle`
Complex Number Support: Yes

Output Arguments

`Y` —

See Also

`wlanSTBCCombine`

wlanSymbolTiming

not found

Syntax

Description

Use the Help browser search field to `matlab:docsearch wlanSymbolTiming`, or type "`matlab:help help`" for help command options, such as help for methods.

wlanVHTConfig

Create VHT format configuration object

Syntax

```
cfgVHT = wlanVHTConfig
cfgVHT = wlanVHTConfig(Name,Value)
```

Description

`cfgVHT = wlanVHTConfig` creates a configuration object that initializes parameters for an IEEE 802.11 very high throughput (VHT) format “PPDU” on page 1-199.

`cfgVHT = wlanVHTConfig(Name,Value)` creates a VHT format configuration object that overrides the default settings using one or more `Name,Value` pair arguments.

At runtime, the calling function validates object settings for properties relevant to the operation of the function.

Examples

Create wlanVHTConfig Object for Single User

Create a VHT configuration object with the default settings.

```
cfgVHT = wlanVHTConfig
```

```
cfgVHT = 

  wlanVHTConfig with properties:

       ChannelBandwidth: 'CBW80'
               NumUsers: 1
    NumTransmitAntennas: 1
    NumSpaceTimeStreams: 1
         SpatialMapping: 'Direct'
```

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

Update the channel bandwidth.

```
cfgVHT.ChannelBandwidth = 'CBW40'
```

```
cfgVHT = 

  wlanVHTConfig with properties:

       ChannelBandwidth: 'CBW40'
               NumUsers: 1
    NumTransmitAntennas: 1
    NumSpaceTimeStreams: 1
         SpatialMapping: 'Direct'
                   STBC: false
                    MCS: 0
          ChannelCoding: 'BCC'
             APEPLength: 1024
          GuardInterval: 'Long'
                GroupID: 63
             PartialAID: 275
```

Create wlanVHTConfig Object for Two Users

Create a VHT configuration object for a 20MHz two-user configuration with two transmit antennas.

Create a `wlanVHTConfig` object using a combination of `Name,Value` pairs and in-line initialization to change default settings. Each element of vector-valued properties apply to a specific user.

```
cfgMU = wlanVHTConfig('ChannelBandwidth','CBW20','NumUsers',2,'GroupID',2,'NumTransmit

cfgMU.NumSpaceTimeStreams = [1 1];
cfgMU.MCS = [4 8];
cfgMU.APEPLength = [1024 2048]
```

```
cfgMU = 

  wlanVHTConfig with properties:

       ChannelBandwidth: 'CBW20'
               NumUsers: 2
          UserPositions: [0 1]
    NumTransmitAntennas: 2
    NumSpaceTimeStreams: [1 1]
         SpatialMapping: 'Direct'
                    MCS: [4 8]
          ChannelCoding: 'BCC'
             APEPLength: [1024 2048]
          GuardInterval: 'Long'
                GroupID: 2
```

`NumUsers` is set to `2` and the user-dependent properties are two-element vectors.

Input Arguments

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'ChannelBandwidth','CBW160','NumUsers',2` specifies a channel bandwidth of 160 MHz and two users for the VHT format packet.

`'ChannelBandwidth'` — Channel bandwidth
`'CBW80'` (default) | `'CBW20'` | `'CBW40'` | `'CBW160'`

Channel bandwidth, specified as `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`. If the transmission has multiple users, the same channel bandwidth is applied to all users. The default value of `'CBW80'` sets the channel bandwidth to 80 MHz.

Data Types: `char`

`'NumUsers'` — Number of users
1 (default) | 2 | 3 | 4

Number of users, specified as 1, 2, 3, or 4.

Data Types: `double`

`'UserPositions'` — Position of users
[0 1] (default) | row vector of integers for 0 to 3 in strictly increasing order

Position of users, specified as an integer row vector with length equal to `NumUsers` and element values from 0 to 3 in a strictly increasing order. This property applies when `NumUsers` property is set to 2, 3 or 4. The default value of this property is [0 1].

Example: `[0 1 2]`indicates that users occupy the specified positions.

Data Types: `double`

`'NumTransmitAntennas'` — Number of transmit antennas
1 (default) | integer from 1 to 8

Number of transmit antennas, specified as a scalar integer from 1 to 8.

Data Types: `double`

`'NumSpaceTimeStreams'` — Number of space-time streams
1 (default) | integer from 1 to 8 | 1-by-N_{Users} vector of integers from 1 to 4

Number of space-time streams in the transmission, specified as a scalar or vector.

- For a single user, the number of space-time streams is a scalar integer from 1 to 8.
- For multiple users, the number of space-time streams is a 1-by-N_{Users} vector of integers from 1 to 4, where the vector length, N_{Users}, is an integer from 1 to 4.

Example: `[1 3 2]` is the number of space-time streams for each user.

Note: The sum of the space-time stream vector elements must not exceed eight.

Data Types: `double`

`'SpatialMapping'` — Spatial mapping scheme
`'Direct'` (default) | `'Hadamard'` | `'Fourier'` | `'Custom'`

Spatial mapping scheme, specified as `'Direct'`, `'Hadamard'`, `'Fourier'`, or `'Custom'`. The default value of `'Direct'` applies when `NumTransmitAntennas` and `NumSpaceTimeStreams` are equal.

Data Types: `char`

`'SpatialMappingMatrix'` — Spatial mapping matrix
1 (default) | scalar | matrix | 3-D array

Spatial mapping matrix, specified as a scalar, matrix, or 3-D array. Use this property to apply a beamforming steering matrix, and to rotate and scale the constellation mapper output vector. If applicable, scale the space-time block coder output instead. `SpatialMappingMatrix` applies when the `SpatialMapping` property is set to `'Custom'`. For more information, see IEEE Std 802.11-2012, Section 20.3.11.11.2.

- When specified as a scalar, a constant value applies to all the subcarriers.
- When specified as a matrix, the size must be N_{STS_Total}-by-N_T. The spatial mapping matrix applies to all the subcarriers. N_{STS_Total} is the sum of space-time streams for all users, and N_T is the number of transmit antennas.
- When specified as a 3-D array, the size must be N_{ST}-by-N_{STS_Total}-by-N_T. N_{ST} is the sum of the occupied data (N_{SD}) and pilot (N_{SP}) subcarriers, as determined by `ChannelBandwidth`. N_{STS_Total} is the sum of space-time streams for all users. N_T is the number of transmit antennas.

  N_{ST} increases with channel bandwidth.

`ChannelBandwidth`	Number of Occupied Subcarriers (N_{ST})	Number of Data Subcarriers (N_{SD})	Number of Pilot Subcarriers (N_{SP})
'CBW20'	56	52	4
'CBW40'	114	108	6
'CBW80'	242	234	8
'CBW160'	484	468	16

The calling function normalizes the spatial mapping matrix for each subcarrier.

Example: [0.5 0.3 0.4; 0.4 0.5 0.8] represents a spatial mapping matrix having two space-time streams and three transmit antennas.

Data Types: `double`
Complex Number Support: Yes

`'Beamforming'` — Option to disable the signaling of a transmission with beamforming

true (default) | false

Option to disable the signaling of a transmission with beamforming, specified as a logical. Beamforming is performed when setting is `true`. This property applies when `NumUsers` equals 1 and `SpatialMapping` is set to `'Custom'`. The `SpatialMappingMatrix` property specifies the beamforming steering matrix.

Data Types: `logical`

`'STBC'` — Option to enable space-time block coding

false (default) | true

Option to enable space-time block coding (STBC) of the PPDU data field, specified as a logical. STBC transmits multiple copies of the data stream across assigned antennas.

- When set to `false`, no STBC is applied to the data field, and the number of space-time streams is equal to the number of spatial streams.
- When set to `true`, STBC is applied to the data field, and the number of space-time streams is double the number of spatial streams.

See IEEE 802.11ac-2013, Section 22.3.10.9.4 for further description.

Note: `STBC` is relevant for single-user transmissions only.

Data Types: `logical`

`'MCS'` — Modulation and coding scheme

0 (default) | integer from 0 to 9 | 1-by-N_{Users} vector of integers

Modulation and coding scheme used in transmitting the current packet, specified as a scalar or vector.

- For a single user, the MCS value is a scalar integer from 0 to 9.
- For multiple users, MCS is a 1-by-N_{Users} vector of integers or a scalar with values from 0 to 9, where the vector length, N_{Users}, is an integer from 1 to 4.

MCS	Modulation	Coding Rate
0	BPSK	1 / 2

MCS	Modulation	Coding Rate
1	`QPSK`	`1/2`
2	`QPSK`	`3/4`
3	`16QAM`	`1/2`
4	`16QAM`	`3/4`
5	`64QAM`	`2/3`
6	`64QAM`	`3/4`
7	`64QAM`	`5/6`
8	`256QAM`	`3/4`
9	`256QAM`	`5/6`

Data Types: `double`

`'APEPLength'` — Number of bytes in the A-MPDU pre-EOF padding
1024 (default) | integer from 0 to 1,048,575 | vector of integers

Number of bytes in the A-MPDU pre-EOF padding, specified as a scalar integer or vector of integers.

- For a single user, `APEPLength` is a scalar integer from 0 to 1,048,575.
- For multi-user, `APEPLength` is a 1-by-N_{Users} vector of integers or a scalar with values from 0 to 1,048,575, where the vector length, N_{Users}, is an integer from 1 to 4.
- `APEPLength = 0` for a null data packet (NDP).

`APEPLength` is used internally to determine the number of OFDM symbols in the data field. For more information, see IEEE Std 802.11ac-2013, Table 22-1.

Data Types: `double`

`'GuardInterval'` — Cyclic prefix length for the data field within a packet
`'Long'` (default) | `'Short'`

Cyclic prefix length for the data field within a packet, specified as `'Long'` or `'Short'`.

- The long guard interval length is 800 ns.
- The short guard interval length is 400 ns.

Data Types: `char`

`'GroupID'` — Group identification number
63 (default) | integer from 0 to 63

Group identification number, specified as a scalar integer from 0 to 63.

- A group identification number of either 0 or 63 indicates a VHT single-user PPDU.
- A group identification number from 1 to 62 indicates a VHT multi-user PPDU.

Data Types: `double`

`'PartialAID'` — Abbreviated indication of the PSDU recipient
275 (default) | integer from 0 to 511

Abbreviated indication of the PSDU recipient, specified as a scalar integer from 0 to 511.

- For an uplink transmission, the partial identification number is the last nine bits of the basic service set identifier (BSSID).
- For a downlink transmission, the partial identification of a client is an identifier that combines the association ID with the BSSID of its serving AP.

For more information, see IEEE Std 802.11ac-2013, Table 22-1.

Data Types: `double`

Output Arguments

`cfgVHT` — VHT PPDU configuration
`wlanVHTConfig` object

VHT “PPDU” on page 1-199 configuration, returned as a `wlanVHTConfig` object. The properties of `cfgVHT` are described in wlanVHTConfig Properties.

More About

PPDU

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

References

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

See Also

`wlanHTConfig` | `wlanNonHTConfig` | `wlanVHTDataRecover` | `wlanVHTLTFDemodulate` | `wlanWaveformGenerator`

Introduced in R2015b

wlanVHTData

Generate VHT-Data field

Syntax

```
y = wlanVHTData(psdu,cfg)
y = wlanVHTData(psdu,cfg,scramInit)
```

Description

`y = wlanVHTData(psdu,cfg)` generates a “VHT-Data field” on page 1-208[20] time-domain waveform from the input user data bits, `psdu`, for the specified configuration object, `cfg`. See “VHT-Data Field Processing” on page 1-209 for waveform generation details.

`y = wlanVHTData(psdu,cfg,scramInit)` uses `scramInit` for the scrambler initialization state.

Examples

Generate VHT-Data Waveform

Generate the waveform for a MIMO 20 MHz VHT-Data field.

Create a VHT configuration object. Assign a 20 MHz channel bandwidth, two transmit antennas, two space-time streams, and set MCS to four.

```
cfgVHT = wlanVHTConfig('ChannelBandwidth','CBW20','NumTransmitAntennas',2,'NumSpaceTime
```

Generate the user payload data and the VHT-Data field waveform.

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

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

```
size(y)
```

```
ans =

        2160           2
```

The 20 MHz waveform is an array with two columns, corresponding to two transmit antennas. There are 2160 complex samples in each column.

```
y(1:10,:)
```

```
ans =

  -0.0598 + 0.1098i  -0.1904 + 0.1409i
   0.6971 - 0.3068i  -0.0858 - 0.2701i
  -0.1284 + 0.9268i  -0.8318 + 0.3314i
  -0.1180 + 0.0731i   0.1313 + 0.4956i
   0.3591 + 0.5485i   0.9749 + 0.2859i
  -0.9751 + 1.3334i   0.0559 + 0.4248i
   0.0881 - 0.8230i  -0.1878 - 0.2959i
  -0.2952 - 0.4433i  -0.1005 - 0.4035i
  -0.5562 - 0.3940i  -0.1292 - 0.5976i
   1.0999 + 0.3292i  -0.2036 - 0.0200i
```

Input Arguments

`psdu` — PHY service data unit

vector

PHY service data unit (“PSDU” on page 1-209), specified as an N_b-by-1 vector. N_b is the number of bits and equals `PSDULength` × 8.

Data Types: `double`

`cfg` — Format configuration

`wlanVHTConfig` object

Format configuration, specified as a `wlanVHTConfig` object. The `wlanVHTData` function uses the object properties indicated.

`ChannelBandwidth` — Channel bandwidth
`'CBW80'` (default) | `'CBW20'` | `'CBW40'` | `'CBW160'`

Channel bandwidth, specified as `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`. If the transmission has multiple users, the same channel bandwidth is applied to all users. The default value of `'CBW80'` sets the channel bandwidth to 80 MHz.

Data Types: `char`

`NumTransmitAntennas` — Number of transmit antennas
1 (default) | integer from 1 to 8

Number of transmit antennas, specified as a scalar integer from 1 to 8.

Data Types: `double`

`NumSpaceTimeStreams` — Number of space-time streams
1 (default) | integer from 1 to 8 | 1-by-N_{Users} vector of integers from 1 to 4

Number of space-time streams in the transmission, specified as a scalar or vector.

- For a single user, the number of space-time streams is a scalar integer from 1 to 8.
- For multiple users, the number of space-time streams is a 1-by-N_{Users} vector of integers from 1 to 4, where the vector length, N_{Users}, is an integer from 1 to 4.

Example: `[1 3 2]` is the number of space-time streams for each user.

Note: The sum of the space-time stream vector elements must not exceed eight.

Data Types: `double`

`SpatialMapping` — Spatial mapping scheme
`'Direct'` (default) | `'Hadamard'` | `'Fourier'` | `'Custom'`

Spatial mapping scheme, specified as `'Direct'`, `'Hadamard'`, `'Fourier'`, or `'Custom'`. The default value of `'Direct'` applies when `NumTransmitAntennas` and `NumSpaceTimeStreams` are equal.

Data Types: `char`

`SpatialMappingMatrix` — Spatial mapping matrix
1 (default) | scalar | matrix | 3-D array

Spatial mapping matrix, specified as a scalar, matrix, or 3-D array. Use this property to apply a beamforming steering matrix, and to rotate and scale the constellation mapper output vector. If applicable, scale the space-time block coder output instead. `SpatialMappingMatrix` applies when the `SpatialMapping` property is set to `'Custom'`. For more information, see IEEE Std 802.11-2012, Section 20.3.11.11.2.

- When specified as a scalar, a constant value applies to all the subcarriers.
- When specified as a matrix, the size must be N_{STS_Total}-by-N_T. The spatial mapping matrix applies to all the subcarriers. N_{STS_Total} is the sum of space-time streams for all users, and N_T is the number of transmit antennas.
- When specified as a 3-D array, the size must be N_{ST}-by-N_{STS_Total}-by-N_T. N_{ST} is the sum of the occupied data (N_{SD}) and pilot (N_{SP}) subcarriers, as determined by `ChannelBandwidth`. N_{STS_Total} is the sum of space-time streams for all users. N_T is the number of transmit antennas.

  N_{ST} increases with channel bandwidth.

`ChannelBandwidth`	Number of Occupied Subcarriers (N_{ST})	Number of Data Subcarriers (N_{SD})	Number of Pilot Subcarriers (N_{SP})
'CBW20'	56	52	4
'CBW40'	114	108	6
'CBW80'	242	234	8
'CBW160'	484	468	16

The calling function normalizes the spatial mapping matrix for each subcarrier.

Example: [0.5 0.3 0.4; 0.4 0.5 0.8] represents a spatial mapping matrix having two space-time streams and three transmit antennas.

Data Types: `double`
Complex Number Support: Yes

`STBC` — Option to enable space-time block coding
false (default) | true

Option to enable space-time block coding (STBC) of the PPDU data field, specified as a logical. STBC transmits multiple copies of the data stream across assigned antennas.

- When set to `false`, no STBC is applied to the data field, and the number of space-time streams is equal to the number of spatial streams.

- When set to `true`, STBC is applied to the data field, and the number of space-time streams is double the number of spatial streams.

See IEEE 802.11ac-2013, Section 22.3.10.9.4 for further description.

Note: `STBC` is relevant for single-user transmissions only.

Data Types: `logical`

`MCS` — Modulation and coding scheme

0 (default) | integer from 0 to 9 | 1-by-N_{Users} vector of integers

Modulation and coding scheme used in transmitting the current packet, specified as a scalar or vector.

- For a single user, the MCS value is a scalar integer from 0 to 9.
- For multiple users, MCS is a 1-by-N_{Users} vector of integers or a scalar with values from 0 to 9, where the vector length, N_{Users}, is an integer from 1 to 4.

MCS	Modulation	Coding Rate
0	`BPSK`	`1/2`
1	`QPSK`	`1/2`
2	`QPSK`	`3/4`
3	`16QAM`	`1/2`
4	`16QAM`	`3/4`
5	`64QAM`	`2/3`
6	`64QAM`	`3/4`
7	`64QAM`	`5/6`
8	`256QAM`	`3/4`
9	`256QAM`	`5/6`

Data Types: `double`

`ChannelCoding` — Type of forward error correction coding

`'BCC'` (default)

This property is read only.

Type of forward error correction coding for the data field, specified as `'BCC'` to indicate binary convolutional coding. LDPC coding is not currently supported.

Data Types: `char`

`GuardInterval` — Cyclic prefix length for the data field within a packet
`'Long'` (default) | `'Short'`

Cyclic prefix length for the data field within a packet, specified as `'Long'` or `'Short'`.

- The long guard interval length is 800 ns.
- The short guard interval length is 400 ns.

Data Types: `char`

`APEPLength` — Number of bytes in the A-MPDU pre-EOF padding
1024 (default) | integer from 0 to 1,048,575 | vector of integers

Number of bytes in the A-MPDU pre-EOF padding, specified as a scalar integer or vector of integers.

- For a single user, `APEPLength` is a scalar integer from 0 to 1,048,575.
- For multi-user, `APEPLength` is a 1-by-N_{Users} vector of integers or a scalar with values from 0 to 1,048,575, where the vector length, N_{Users}, is an integer from 1 to 4.
- `APEPLength = 0` for a null data packet (NDP).

`APEPLength` is used internally to determine the number of OFDM symbols in the data field. For more information, see IEEE Std 802.11ac-2013, Table 22-1.

Data Types: `double`

`PSDULength` — Number of bytes carried in the user payload
integer | vector of integers

This property is read only.

Number of bytes carried in the user payload, including the A-MPDU and any MAC padding. For a null data packet (NDP) the PSDU length is zero.

- For a single user, the PSDU length is a scalar integer from 1 to 1,048,575.

- For multiple users, the PSDU length is a 1-by-N_{Users} vector of integers from 1 to 1,048,575, where the vector length,N_{Users}, is an integer from 1 to 4.

`PSDULength` is a read-only property and is calculated internally based on the `APEPLength` property and other coding-related properties, as specified in IEEE Std 802.11ac-2013, Section 22.4.3. It is accessible by direct property call. When accessing `PSDULength`, the object is validated.

Example: `[1035 4150]` is the PSDU length vector for a `wlanVHTConfig` object with two users, where the MCS for the first user is 0 and the MCS for the second user is 3.

Data Types: `double`

`scramInit` — Scrambler initialization state

93 (default) | integer from 1 to 127 | integer row vector | binary column vector | binary matrix

Initial scrambler state of the data scrambler for each packet generated, specified as an integer, a binary column vector, a 1-by-N_U integer row vector, or a 7-by-N_U binary matrix. N_U is the number of users, from 1 to 4. If specified as an integer or binary column vector, the setting applies to all users. If specified as a row vector or binary matrix, the setting for each user is specified in the corresponding column, as a scalar integer from 1 to 127 or the corresponding binary column vector.

Example: `[1;0;1;1;1;0;1]` conveys the scrambler initialization state of 93 as a binary column vector.

Data Types: `double` | `int8`

Output Arguments

y — VHT-Data field time-domain waveform

matrix

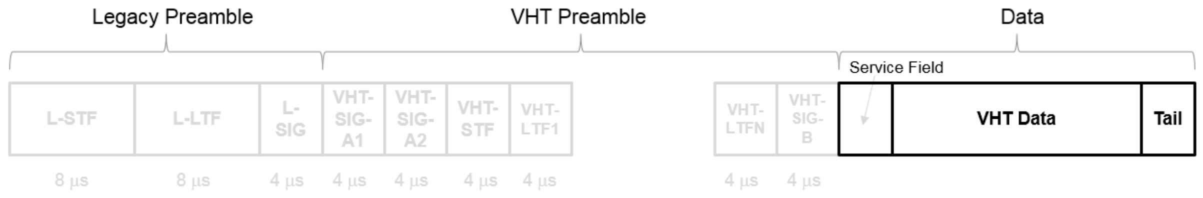
"VHT-Data field" on page 1-208 time-domain waveform, returned as an N_S-by-N_T matrix. N_S is the number of time-domain samples and N_T is the number of transmit antennas. See "VHT-Data Field Processing" on page 1-209 for waveform generation details.

Data Types: `double`
Complex Number Support: Yes

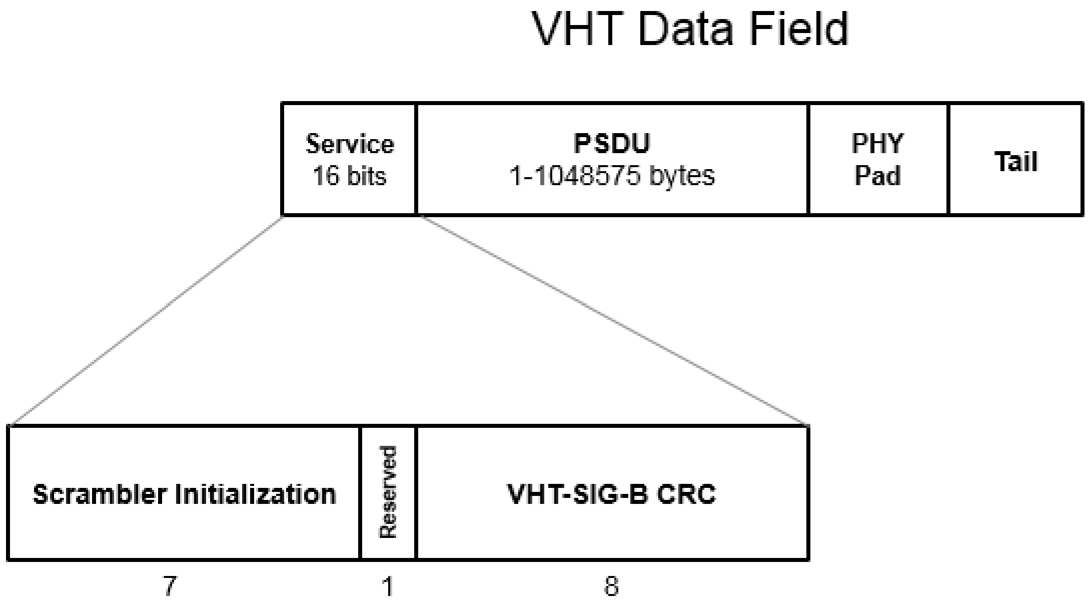
More About

VHT-Data field

The very high throughput data (VHT data) field is used to transmit one or more frames from the MAC layer. It follows the VHT-SIG-B field in the packet structure for the VHT format PPDUs.



The VHT data field is defined in IEEE Std 802.11ac-2013, Section 22.3.10. It is composed of four subfields.



- **Service field** — Contains a seven-bit scrambler initialization state, one bit reserved for future considerations, and eight bits for the VHT-SIG-B CRC field.
- **PSDU** — Variable-length field containing the PLCP service data unit. In 802.11, the PSDU can consist of an aggregate of several MAC service data units.
- **PHY Pad** — Variable number of bits passed to the transmitter to create a complete OFDM symbol.
- **Tail** — Bits used to terminate a convolutional code. Tail bits are not needed when LDPC is used.

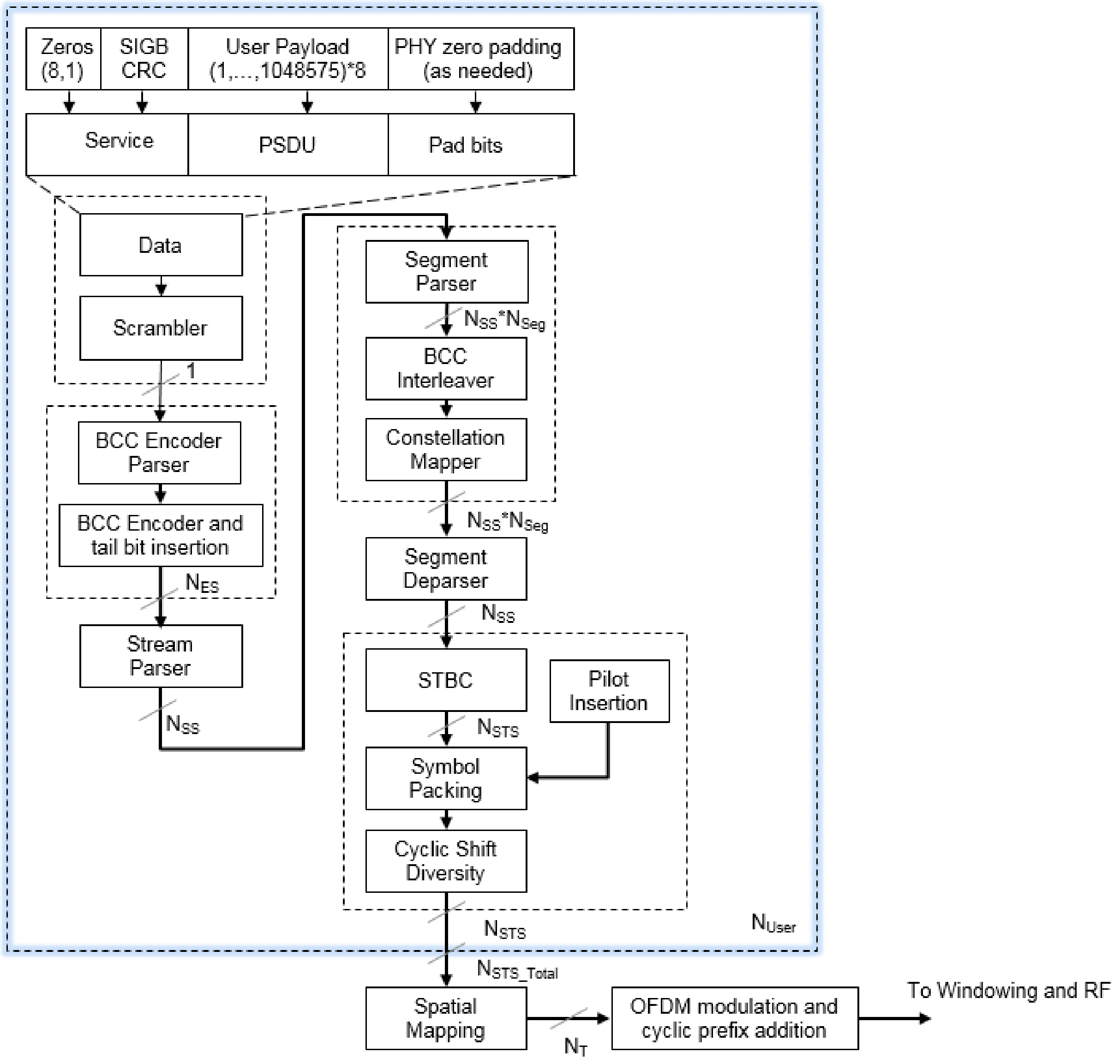
PSDU

Physical layer (PHY) Service Data Unit (PSDU). A PSDU can consist of one medium access control (MAC) protocol data unit (MPDU) or several MPDUs in an aggregate MPDU (A-MPDU). In a single user scenario, the VHT-Data field contains one PSDU. In a multi-user scenario, the VHT-Data field carries up to four PSDUs for up to four users.

Algorithms

VHT-Data Field Processing

The “VHT-Data field” on page 1-208 encodes the service, “PSDU” on page 1-209, pad bits, and tail bits. The `wlanVHTData` function performs transmitter processing on the “VHT-Data field” on page 1-208 and outputs the time-domain waveform for N_T transmit antennas.



For algorithm details, refer to IEEE Std 802.11ac-2013 [1], Section 22.3.4.9 and 22.3.4.10, respectively, single user and multi-user.

References

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

See Also

`wlanHTConfig` | `wlanVHTDataRecover` | `wlanWaveformGenerator`

Introduced in R2015b

wlanVHTDataRecover

Recover VHT data

Syntax

```
recBits = wlanVHTDataRecover(rxSig,chEst,noiseVarEst,cfg)
recBits = wlanVHTDataRecover(rxSig,chEst,noiseVarEst,cfg,cfgRec)
[recBits,crcBits] = wlanVHTDataRecover(___)
[recBits,crcBits,eqSym] = wlanVHTDataRecover(___)
[recBits,crcBits,eqSym,cpe] = wlanVHTDataRecover(___)
```

Description

`recBits = wlanVHTDataRecover(rxSig,chEst,noiseVarEst,cfg)` returns the recovered payload bits from the "VHT data field" on page 1-221[21]. Inputs include the received "VHT data field" on page 1-221 signal, the channel estimate, the noise variance estimate, and the format configuration object, `cfg`.

`recBits = wlanVHTDataRecover(rxSig,chEst,noiseVarEst,cfg,cfgRec)` returns the recovered bits using the algorithm parameters specified in `cfgRec`.

`[recBits,crcBits] = wlanVHTDataRecover(___)` also returns the VHT-SIG-B checksum bits, `crcBits`, using the arguments from the previous syntaxes.

`[recBits,crcBits,eqSym] = wlanVHTDataRecover(___)` also returns the equalized symbols, `eqSym`.

`[recBits,crcBits,eqSym,cpe] = wlanVHTDataRecover(___)` also returns the common phase error, `cpe`.

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

Examples

Recover VHT-Data Field Over 2x2 Fading Channel

Recover bits in the VHT-Data field using channel estimation on a VHT-LTF field over a 2 x 2 quasi-static fading channel.

Create a VHT configuration object with 160 MHz channel bandwidth and two transmission paths.

```
cbw = 'CBW160';
vht = wlanVHTConfig('ChannelBandwidth',cbw,'NumTransmitAntennas',2,'NumSpaceTimeStream
```

Generate VHT-LTF and VHT-Data field signals.

```
txDataBits = randi([0 1],8*vht.PSDULength,1);
txVHTLTF  = wlanVHTLTF(vht);
txVHTData = wlanVHTData(txDataBits,vht);
```

Pass the transmitted waveform through a 2 x 2 quasi-static fading channel with AWGN.

```
snr = 10;
H = 1/sqrt(2)*complex(randn(2,2),randn(2,2));
rxVHTLTF  = awgn(txVHTLTF*H,snr);
rxVHTData = awgn(txVHTData*H,snr);
```

Calculate the received signal power and use it to estimate the noise variance.

```
powerDB = 10*log10(var(rxVHTData));
noiseVarEst = mean(10.^(0.1*(powerDB-snr)));
```

Perform channel estimation based on the VHT-LTF field.

```
demodVHTLTF = wlanVHTLTFDemodulate(rxVHTLTF,vht,1);
chanEst = wlanVHTLTFChannelEstimate(demodVHTLTF,vht);
```

Recover payload bits in the VHT-Data field and compare against the original payload bits.

```
rxDataBits = wlanVHTDataRecover(rxVHTData,chanEst,noiseVarEst,vht);
numErr = biterr(txDataBits,rxDataBits)
numErr =

     0
```

Recover VHT-Data Field Signal

Recover a VHT-Data field signal through a SISO AWGN channel using ZF equalization.

Configure VHT format object, generate random payload bits, and generate the VHT-Data field.

```
cfgVHT = wlanVHTConfig('APEPLength',512);
txBits = randi([0 1], 8*cfgVHT.PSDULength,1);
txVHTData = wlanVHTData(txBits,cfgVHT);
```

Pass the transmitted VHT data through an AWGN channel.

```
ch = comm.AWGNChannel('NoiseMethod','Variance','Variance',0.1);
rxVHTData = step(ch,txVHTData);
```

Configure the recovery object and recover the payload bits using a perfect channel estimate of all ones. Compare the recovered bits against the transmitted bits.

```
cfgRec = wlanRecoveryConfig('EqualizationMethod','ZF');
recBits = wlanVHTDataRecover(rxVHTData,ones(242,1),0.1,cfgVHT,cfgRec);
numErrs = biterr(txBits,recBits)
```

```
numErrs =

     0
```

Input Arguments

`rxSig` — Received VHT-Data field signal

matrix

Received VHT-Data field signal in the time domain, specified as an N_S-by-N_R matrix. N_R is the number of receive antennas. N_S must be greater than or equal to the number of time-domain samples in the VHT-Data field input.

Note: `wlanVHTDataRecover` processes one PPDU data field per entry. If N_S is greater than the field length, extra samples at the end of `rxSig` are not processed. To process a concatenated stream of PPDU data fields, multiple calls to `wlanVHTDataRecover` are required. If `rxSig` is shorter than the length of the VHT-Data field, an error occurs.

Data Types: `double`
Complex Number Support: Yes

`chEst` — Channel estimation

matrix | 3-D array

Channel estimation for data and pilot subcarriers, specified as a matrix or array of size N_{ST}-by-N_{STS}-by-N_R. N_{ST} is the number of occupied subcarriers. N_{STS} is the number of space-time streams. N_R is the number of receive antennas. N_{ST} and N_{STS} must match the `cfg` configuration object settings for channel bandwidth and number of space-time streams.

N_{ST} increases with channel bandwidth.

`ChannelBandwidth`	Number of Occupied Subcarriers (N_{ST})	Number of Data Subcarriers (N_{SD})	Number of Pilot Subcarriers (N_{SP})
'CBW20'	56	52	4
'CBW40'	114	108	6
'CBW80'	242	234	8
'CBW160'	484	468	16

Data Types: `double`
Complex Number Support: Yes

`noiseVarEst` — Noise variance estimate

nonnegative scalar

Noise variance estimate, specified as a nonnegative scalar.

Data Types: `double`

`cfg` — VHT PPDU configuration

`wlanVHTConfig` object

VHT PPDU configuration, specified as a `wlanVHTConfig` object. The `wlanVHTDataRecover` function uses the following `wlanVHTConfig` object properties:

`ChannelBandwidth` — Channel bandwidth

`'CBW80'` (default) | `'CBW20'` | `'CBW40'` | `'CBW160'`

Channel bandwidth, specified as `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`. If the transmission has multiple users, the same channel bandwidth is applied to all users. The default value of `'CBW80'` sets the channel bandwidth to 80 MHz.

Data Types: `char`

`NumSpaceTimeStreams` — Number of space-time streams

1 (default) | integer from 1 to 8 | 1-by-N_{Users} vector of integers from 1 to 4

Number of space-time streams in the transmission, specified as a scalar or vector.

- For a single user, the number of space-time streams is a scalar integer from 1 to 8.
- For multiple users, the number of space-time streams is a 1-by-N_{Users} vector of integers from 1 to 4, where the vector length, N_{Users}, is an integer from 1 to 4.

Example: `[1 3 2]` is the number of space-time streams for each user.

Note: The sum of the space-time stream vector elements must not exceed eight.

Data Types: `double`

`STBC` — Option to enable space-time block coding

false (default) | true

Option to enable space-time block coding (STBC) of the PPDU data field, specified as a logical. STBC transmits multiple copies of the data stream across assigned antennas.

- When set to `false`, no STBC is applied to the data field, and the number of space-time streams is equal to the number of spatial streams.
- When set to `true`, STBC is applied to the data field, and the number of space-time streams is double the number of spatial streams.

See IEEE 802.11ac-2013, Section 22.3.10.9.4 for further description.

Note: `STBC` is relevant for single-user transmissions only.

Data Types: `logical`

`GuardInterval` — Cyclic prefix length for the data field within a packet

`'Long'` (default) | `'Short'`

Cyclic prefix length for the data field within a packet, specified as `'Long'` or `'Short'`.

- The long guard interval length is 800 ns.
- The short guard interval length is 400 ns.

Data Types: `char`

`MCS` — Modulation and coding scheme

0 (default) | integer from 0 to 9 | 1-by-N_{Users} vector of integers

Modulation and coding scheme used in transmitting the current packet, specified as a scalar or vector.

- For a single user, the MCS value is a scalar integer from 0 to 9.
- For multiple users, MCS is a 1-by-N_{Users} vector of integers or a scalar with values from 0 to 9, where the vector length, N_{Users}, is an integer from 1 to 4.

MCS	Modulation	Coding Rate
0	BPSK	1/2
1	QPSK	1/2
2	QPSK	3/4
3	16QAM	1/2
4	16QAM	3/4
5	64QAM	2/3
6	64QAM	3/4
7	64QAM	5/6
8	256QAM	3/4
9	256QAM	5/6

Data Types: `double`

`APEPLength` — Number of bytes in the A-MPDU pre-EOF padding
1024 (default) | integer from 0 to 1,048,575 | vector of integers

Number of bytes in the A-MPDU pre-EOF padding, specified as a scalar integer or vector of integers.

- For a single user, `APEPLength` is a scalar integer from 0 to 1,048,575.
- For multi-user, `APEPLength` is a 1-by-N_{Users} vector of integers or a scalar with values from 0 to 1,048,575, where the vector length, N_{Users}, is an integer from 1 to 4.
- `APEPLength = 0` for a null data packet (NDP).

`APEPLength` is used internally to determine the number of OFDM symbols in the data field. For more information, see IEEE Std 802.11ac-2013, Table 22-1.

Data Types: `double`

`cfgRec` — Algorithm parameters
`wlanRecoveryConfig` object

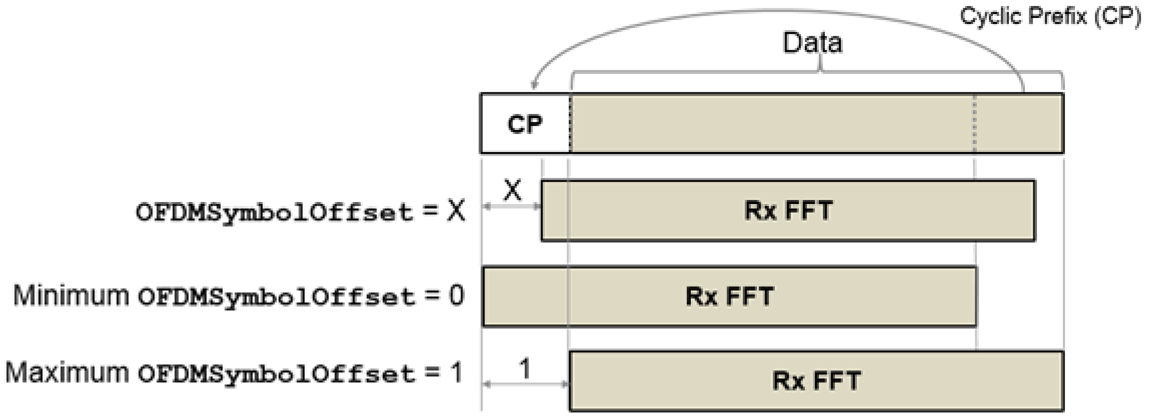
Algorithm parameters containing properties used during data receovery, specified as a `wlanRecoveryConfig` object. The configurable properties include OFDM symbol sampling offset, equalization method, and the type of pilot phase tracking. If you do not specify a `cfgRec` object, the default object property values as described in wlanRecoveryConfig Properties are used in the data recovery.

Note: Use `cfgRec.EqualizationMethod='ZF'` when either of the following conditions are met:

- `cfg.NumSpaceTimeStreams=1`
- `cfg.NumSpaceTimeStreams=2` and `cfg.STBC=true`

`OFDMSymbolOffset` — OFDM symbol sampling offset
0.75 (default) | scalar value from 0 to 1

OFDM symbol sampling offset represented as a fraction of the cyclic prefix (CP) length, specified as a scalar value from 0 to 1. This value indicates the start location for OFDM demodulation, relative to the beginning of the cyclic prefix. `OFDMSymbolOffset` = 0 represents the start of the cyclic prefix and `OFDMSymbolOffset` = 1 represents the end of the cyclic prefix.



Data Types: `double`

`EqualizationMethod` — Equalization method
`'MMSE'` (default) | `'ZF'`

Equalization method, specified as `'MMSE'` or `'ZF'`.

- `'MMSE'` indicates that the receiver uses a minimum mean square error equalizer.
- `'ZF'` indicates that the receiver uses a zero-forcing equalizer.

Example: `'ZF'`

Data Types: `char`

`PilotPhaseTracking` — Pilot phase tracking
`'PreEQ'` (default) | `'None'`

Pilot phase tracking, specified as either of these strings: `'PreEQ'` or `'None'`. If `'PreEQ'` is specified, pilot phase tracking is enabled and is performed prior to any equalization operation. If `'None'` is specified, pilot phase tracking does not occur.

Data Types: `char`

Output Arguments

`recBits` — Recovered payload bits in the VHT-Data field
1 | 0 | column vector

Recovered payload bits in the VHT-Data field, returned as a column vector of length 8 × `cfgVHT.PSDULength`. See wlanVHTConfig Properties for `PSDULength` details.

Data Types: `int8`

`crcBits` — Checksum bits for VHT-SIG-B field
binary column vector

Checksum bits for VHT-SIG-B field, returned as a binary column vector of length 8.

Data Types: `int8`

`eqSym` — Equalized symbols
matrix | 3-D array

Equalized symbols, returned as an N_{SD}-by-N_{SYM}-by-N_{SS} matrix or array. N_{SD} is the number of data subcarriers. N_{SYM} is the number of OFDM symbols in the VHT-Data field. N_{SS} is the number of spatial streams. When STBC is `false`, $N_{SS} = N_{STS}$. When STBC is `true`, $N_{SS} = N_{STS}/2$.

Data Types: `double`
Complex Number Support: Yes

`cpe` — Common phase error
column vector

Common phase error in radians, returned as a column vector having length N_{SYM}. N_{SYM} is the number of OFDM symbols in the “VHT data field” on page 1-221.

Limitations

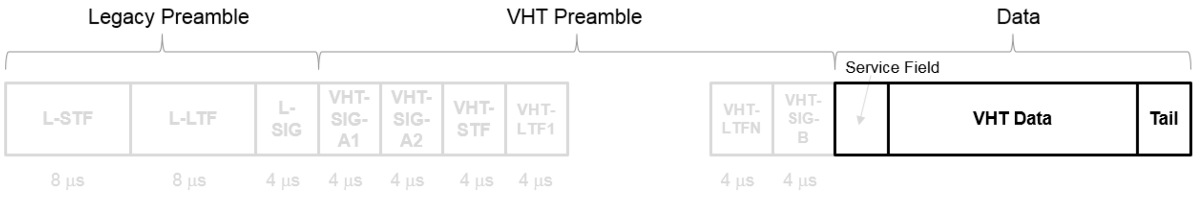
`wlanVHTDataRecover` processing limitations, restrictions, and recommendations:

- If only VHT format PPDUs are processed, then `isa(cfgVHT, 'wlanVHTConfig')` must be `true`.
- For single-user scenarios, `cfgVHT.NumUsers` must equal 1.
- When STBC is enabled, the number of space-time streams must be even.
- `cfgRec.EqualizationMethod = 'ZF'` is recommended when `cfgVHT.STBC = true` and `cfgVHT.NumSpaceTimeStreams = 2`
- `cfgRec.EqualizationMethod = 'ZF'` is recommended when `cfgVHT.NumSpaceTimeStreams = 1`

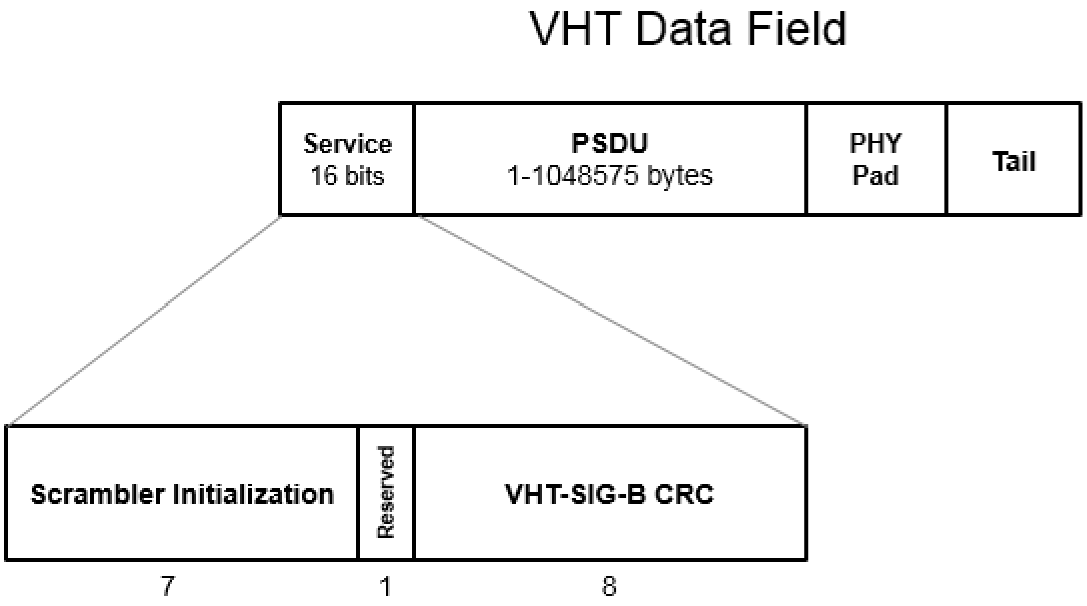
More About

VHT data field

The very high throughput data (VHT data) field is used to transmit one or more frames from the MAC layer. It follows the VHT-SIG-B field in the packet structure for the VHT format PPDUs.



The VHT data field is defined in IEEE Std 802.11ac-2013, Section 22.3.10. It is composed of four subfields.



- **Service field** — Contains a seven-bit scrambler initialization state, one bit reserved for future considerations, and eight bits for the VHT-SIG-B CRC field.
- **PSDU** — Variable-length field containing the PLCP service data unit. In 802.11, the PSDU can consist of an aggregate of several MAC service data units.
- **PHY Pad** — Variable number of bits passed to the transmitter to create a complete OFDM symbol.
- **Tail** — Bits used to terminate a convolutional code. Tail bits are not needed when LDPC is used.

References

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

See Also

`wlanRecoveryConfig` | `wlanVHTConfig` | `wlanVHTData` | `wlanVHTLTFChannelEstimate` | `wlanVHTLTFDemodulate`

Introduced in R2015b

wlanVHTLTF

Generate VHT-LTF waveform

Syntax

```
y = wlanVHTLTF(cfg)
```

Description

`y = wlanVHTLTF(cfg)` generates a "VHT-LTF" on page 1-226 [22] time-domain waveform for the specified configuration object. See "VHT-LTF Processing" on page 1-227 for waveform generation details.

Examples

Generate VHT-LTF Waveform

Create a VHT configuration object with an 80 MHz channel bandwidth.

```
cfgVHT = wlanVHTConfig;
cfgVHT.ChannelBandwidth = 'CBW80';
```

Generate a VHT-LTF waveform.

```
vltfOut = wlanVHTLTF(cfgVHT);
size(vltfOut)
```

```
ans =

   320     1
```

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

The 80 MHz waveform is a single OFDM symbol with 320 complex output samples.

Input Arguments

`cfg` — Format configuration

`wlanVHTConfig` object

Format configuration, specified as a `wlanVHTConfig` object. The `wlanVHTLTF` function uses the object properties indicated.

`ChannelBandwidth` — Channel bandwidth

`'CBW80'` (default) | `'CBW20'` | `'CBW40'` | `'CBW160'`

Channel bandwidth, specified as `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`. If the transmission has multiple users, the same channel bandwidth is applied to all users. The default value of `'CBW80'` sets the channel bandwidth to 80 MHz.

Data Types: `char`

`NumTransmitAntennas` — Number of transmit antennas

1 (default) | integer from 1 to 8

Number of transmit antennas, specified as a scalar integer from 1 to 8.

Data Types: `double`

`NumSpaceTimeStreams` — Number of space-time streams

1 (default) | integer from 1 to 8 | 1-by-N_{Users} vector of integers from 1 to 4

Number of space-time streams in the transmission, specified as a scalar or vector.

- For a single user, the number of space-time streams is a scalar integer from 1 to 8.
- For multiple users, the number of space-time streams is a 1-by-N_{Users} vector of integers from 1 to 4, where the vector length, N_{Users}, is an integer from 1 to 4.

Example: `[1 3 2]` is the number of space-time streams for each user.

Note: The sum of the space-time stream vector elements must not exceed eight.

Data Types: `double`

`SpatialMapping` — Spatial mapping scheme

`'Direct'` (default) | `'Hadamard'` | `'Fourier'` | `'Custom'`

Spatial mapping scheme, specified as `'Direct'`, `'Hadamard'`, `'Fourier'`, or `'Custom'`. The default value of `'Direct'` applies when `NumTransmitAntennas` and `NumSpaceTimeStreams` are equal.

Data Types: `char`

`SpatialMappingMatrix` — Spatial mapping matrix

1 (default) | scalar | matrix | 3-D array

Spatial mapping matrix, specified as a scalar, matrix, or 3-D array. Use this property to apply a beamforming steering matrix, and to rotate and scale the constellation mapper output vector. If applicable, scale the space-time block coder output instead. `SpatialMappingMatrix` applies when the `SpatialMapping` property is set to `'Custom'`. For more information, see IEEE Std 802.11-2012, Section 20.3.11.11.2.

- When specified as a scalar, a constant value applies to all the subcarriers.
- When specified as a matrix, the size must be N_{STS_Total}-by-N_T. The spatial mapping matrix applies to all the subcarriers. N_{STS_Total} is the sum of space-time streams for all users, and N_T is the number of transmit antennas.
- When specified as a 3-D array, the size must be N_{ST}-by-N_{STS_Total}-by-N_T. N_{ST} is the sum of the occupied data (N_{SD}) and pilot (N_{SP}) subcarriers, as determined by `ChannelBandwidth`. N_{STS_Total} is the sum of space-time streams for all users. N_T is the number of transmit antennas.

  N_{ST} increases with channel bandwidth.

`ChannelBandwidth`	Number of Occupied Subcarriers (N_{ST})	Number of Data Subcarriers (N_{SD})	Number of Pilot Subcarriers (N_{SP})
'CBW20'	56	52	4
'CBW40'	114	108	6
'CBW80'	242	234	8
'CBW160'	484	468	16

The calling function normalizes the spatial mapping matrix for each subcarrier.

Example: [0.5 0.3 0.4; 0.4 0.5 0.8] represents a spatial mapping matrix having two space-time streams and three transmit antennas.

Data Types: `double`
Complex Number Support: Yes

Output Arguments

y — VHT-LTF time-domain waveform

matrix

“VHT-LTF” on page 1-226 time-domain waveform, returned as an ($N_S \times N_{VHTLTF}$)-by-N_T matrix. N_S is the number of time-domain samples per N_{VHTLTF}, where N_{VHTLTF} is the number of OFDM symbols in the VHT-LTF. N_T is the number of transmit antennas.

N_S is proportional to the channel bandwidth. Each symbol contains 80 time samples per 20 MHz channel.

`ChannelBandwidth`	N_S
`'CBW20'`	80
`'CBW40'`	160
`'CBW80'`	320
`'CBW160'`	640

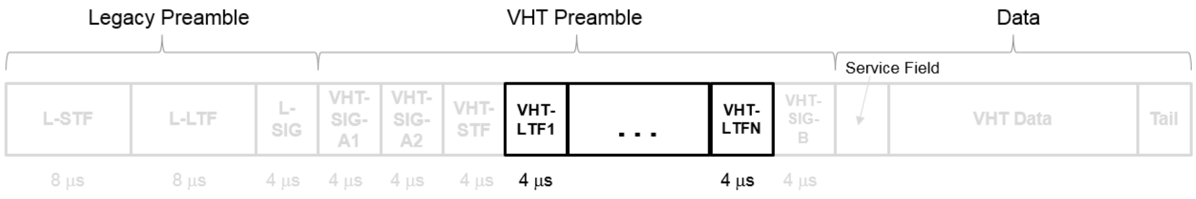
See “VHT-LTF Processing” on page 1-227 for waveform generation details.

Data Types: `double`
Complex Number Support: Yes

More About

VHT-LTF

The very high throughput long training field (VHT-LTF) is located between the VHT-STF and VHT-SIG-B portion of the VHT packet.



It is used for MIMO channel estimation and pilot subcarrier tracking. The VHT-LTF includes one VHT long training symbol for each spatial stream indicated by the selected MCS. Each symbol is 4 µs long. A maximum of eight symbols are permitted in the VHT-LTF.

The VHT-LTF is defined in IEEE Std 802.11ac-2013, Section 22.3.8.3.5.

Algorithms

VHT-LTF Processing

The “VHT-LTF” on page 1-226 is used for MIMO channel estimation and pilot subcarrier tracking. The number of OFDM symbols in the “VHT-LTF” on page 1-226 (N_{VHTLTF}) is derived from the total number of space-time streams (N_{STS_Total}). $N_{STS\_Total} = \Sigma N_{STS}(u)$ for user u, $u = 0,\ldots, N_{Users}–1$ and $N_{STS}(u)$ is the number of space-time streams per user.

N_{STS_Total}	N_{VHTLTF}
1	1
2	2
3	4
4	4
5	6
6	6
7	8
8	8

For algorithm details refer to IEEE Std 802.11ac-2013 [1], Section 22.3.4.7.

References

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

See Also

`wlanLLTF` | `wlanVHTConfig` | `wlanVHTData` | `wlanVHTLTFChannelEstimate` | `wlanVHTLTFDemodulate` | `wlanVHTSTF`

Introduced in R2015b

wlanVHTLTFDemodulate

Demodulate VHT-LTF waveform

Syntax

```
y = wlanVHTLTFDemodulate(x,cfg)
y = wlanVHTLTFDemodulate(x,cfg,OFDMSymbolOffset)
```

Description

`y = wlanVHTLTFDemodulate(x,cfg)` returns demodulated "VHT-LTF" on page 1-235[23] waveform `y` given time-domain input signal `x` and `wlanVHTConfig` object `cfg`.

`y = wlanVHTLTFDemodulate(x,cfg,OFDMSymbolOffset)` specifies the OFDM symbol offset as a fraction of the cyclic prefix length.

Examples

Demodulate Received VHT-LTF Signal

Create a VHT format configuration object.

```
vht = wlanVHTConfig;
```

Generate a VHT-LTF signal.

```
txVHTLTF = wlanVHTLTF(vht);
```

Add white noise to the signal.

```
rxVHTLTF = awgn(txVHTLTF,1);
```

Demodulate the received signal.

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

```
y = wlanVHTLTFDemodulate(rxVHTLTF,vht);
```

Demodulate VHT-LTF and Estimate Channel Coefficients

Specify a VHT format configuration object and generate a VHT-LTF.

```
vht = wlanVHTConfig;
txltf = wlanVHTLTF(vht);
```

Multiply the transmitted VHT-LTF by 0.1 + 0.1i . Pass the signal through an AWGN channel.

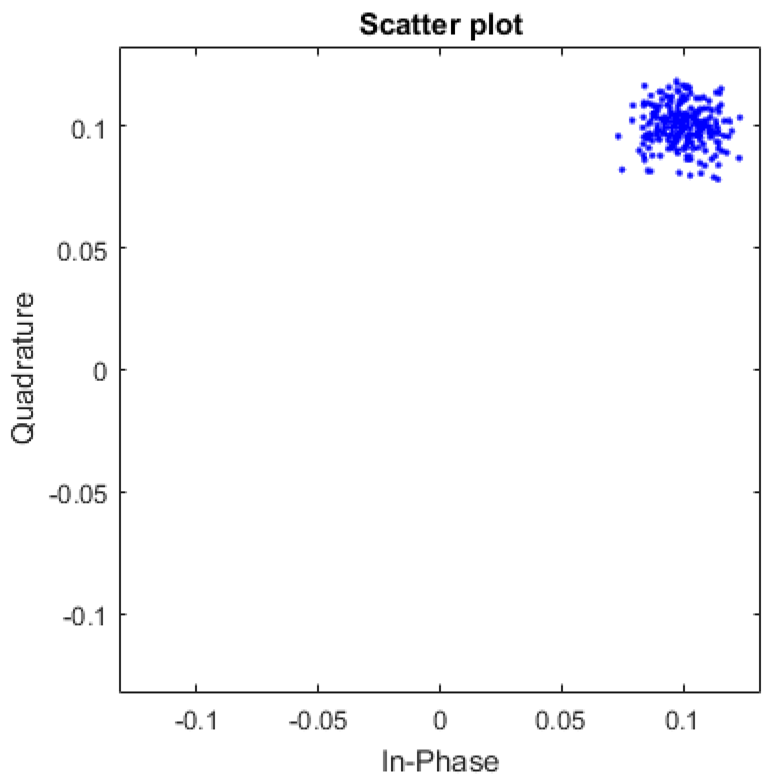
```
rxltfNoNoise = txltf * complex(0.1,0.1);
rxltf = awgn(rxltfNoNoise,20,'measured');
```

Demodulated the received VHT-LTF with a symbol offset of 0.5.

```
dltf = wlanVHTLTFDemodulate(rxltf,vht,0.5);
```

Estimate the channel using the demodulated VHT-LTF. Plot the result.

```
chEst = wlanVHTLTFChannelEstimate(dltf,vht);
scatterplot(chEst)
```



The estimate is very close to the previously introduced 0.1+0.1i multiplier.

Extract VHT-LTF and Recover VHT Data

Generate a VHT waveform. Extract and demodulate the VHT-LTF to estimate the channel coefficients. Recover the data field using the channel estimate and use this to determine the number of bit errors.

Configure a VHT channel and a waveform generator.

```
vht = wlanVHTConfig('NumTransmitAntennas',2,'NumSpaceTimeStreams',2);
wgc = wlanGeneratorConfig;
```

Generate a random PSDU and create the corresponding VHT waveform.

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

Pass the signal through a TGac 2x2 MIMO channel.

```
tgac = wlanTGacChannel('NumTransmitAntennas',2,'NumReceiveAntennas',2, ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
rxSigNoNoise = step(tgac,txSig);
```

Add AWGN to the received signal. Set the noise variance for the case in which the receiver has a 9 dB noise figure.

```
nVar = 10^((-228.6+10*log10(290)+10*log10(80e6)+9)/10);
chAWGN = comm.AWGNChannel('NoiseMethod','Variance','Variance',nVar);
rxSig = step(chAWGN,rxSigNoNoise);
```

Determine the indices for the VHT-LTF and extract the field from the received signal.

```
indVHT = wlanFieldIndices(vht,'VHT-LTF');
rxLTF = rxSig(indVHT(1):indVHT(2),:);
```

Demodulate the VHT-LTF and estimate the channel coefficients.

```
dLTF = wlanVHTLTFDemodulate(rxLTF,vht);
chEst = wlanVHTLTFChannelEstimate(dLTF,vht);
```

Extract the data field and recover the information bits.

```
indData = wlanFieldIndices(vht,'VHT-Data');
rxData = rxSig(indData(1):indData(2),:);
rxPSDU = wlanVHTDataRecover(rxData,chEst,nVar,vht);
```

Determine the number of bit errors.

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

```
numErrs =

     0
```

Input Arguments

x — Time-domain input signal
matrix

Time-domain input signal corresponding to the VHT-LTF of the PPDU, specified as a matrix of size N_S-by-N_R. N_S is the number of samples. N_R is the number of receive antennas. N_S can be greater than or equal to the VHT-LTF length as indicated by `cfg`. Trailing samples at the end of `x` are not used.

Data Types: `double`
Complex Number Support: Yes

`cfg` — VHT format configuration
`wlanVHTConfig` object

VHT format configuration, specified as a `wlanVHTConfig` object. The function uses the following `wlanVHTConfig` object properties:

`ChannelBandwidth` — Channel bandwidth
`'CBW80'` (default) | `'CBW20'` | `'CBW40'` | `'CBW160'`

Channel bandwidth, specified as `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`. If the transmission has multiple users, the same channel bandwidth is applied to all users. The default value of `'CBW80'` sets the channel bandwidth to 80 MHz.

Data Types: `char`

`NumSpaceTimeStreams` — Number of space-time streams
1 (default) | integer from 1 to 8 | 1-by-N_{Users} vector of integers from 1 to 4

Number of space-time streams in the transmission, specified as a scalar or vector.

- For a single user, the number of space-time streams is a scalar integer from 1 to 8.
- For multiple users, the number of space-time streams is a 1-by-N_{Users} vector of integers from 1 to 4, where the vector length, N_{Users}, is an integer from 1 to 4.

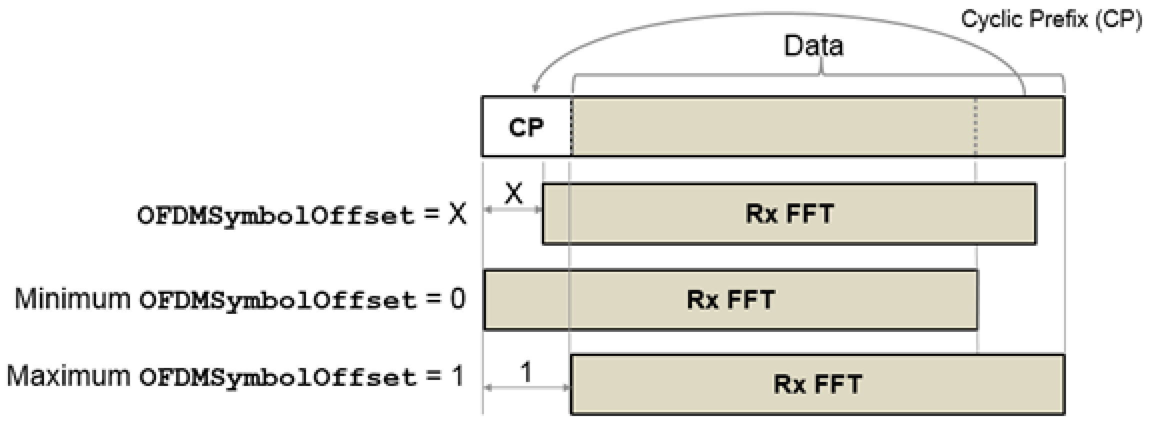
Example: `[1 3 2]` is the number of space-time streams for each user.

Note: The sum of the space-time stream vector elements must not exceed eight.

Data Types: `double`

`OFDMSymbolOffset` — OFDM symbol sampling offset
0.75 (default) | scalar value from 0 to 1

OFDM symbol sampling offset represented as a fraction of the cyclic prefix (CP) length, specified as a scalar value from 0 to 1. This value indicates the start location for OFDM demodulation, relative to the beginning of the cyclic prefix. `OFDMSymbolOffset` = 0 represents the start of the cyclic prefix and `OFDMSymbolOffset` = 1 represents the end of the cyclic prefix.



Data Types: `double`

Output Arguments

y — Demodulated VHT-LTF waveform

matrix | 3-D array

Demodulated VHT-LTF waveform, returned as an N_{ST}-by-N_{SYM}-by-N_R array. N_{ST} is the number of data and pilot subcarriers, N_{SYM} is the number of OFDM symbols in the VHT-LTF, and N_R is the number of receive antennas.

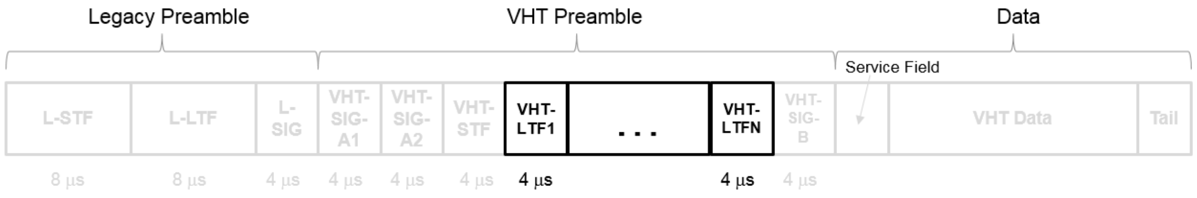
If the received VHT-LTF signal, `x`, is empty, then the output is also empty.

Data Types: `double`
Complex Number Support: Yes

More About

VHT-LTF

The very high throughput long training field (VHT-LTF) is located between the VHT-STF and VHT-SIG-B portion of the VHT packet.



It is used for MIMO channel estimation and pilot subcarrier tracking. The VHT-LTF includes one VHT long training symbol for each spatial stream indicated by the selected MCS. Each symbol is 4 µs long. A maximum of eight symbols are permitted in the VHT-LTF.

The VHT-LTF is defined in IEEE Std 802.11ac-2013, Section 22.3.8.3.5.

References

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

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

See Also

`wlanVHTConfig` | `wlanVHTLTF` | `wlanVHTLTFChannelEstimate`

Introduced in R2015b

wlanVHTSIGA

Generate VHT-SIG-A waveform

Syntax

```
y= wlanVHTSIGA(cfg)
[y,bits] = wlanVHTSIGA(cfg)
```

Description

`y= wlanVHTSIGA(cfg)` generates a “VHT-SIG-A” on page 1-243[24] time-domain waveform for the specified configuration object. See “VHT-SIG-A Processing” on page 1-245 for waveform generation details.

`[y,bits] = wlanVHTSIGA(cfg)` also outputs “VHT-SIG-A” on page 1-243 information bits.

Examples

Generate VHT-SIG-A Waveform

Generate the VHT-SIG-A waveform for an 80 MHz transmission packet.

Create a VHT configuration object, assign an 80 MHz channel bandwidth, and generate the waveform.

```
cfgVHT = wlanVHTConfig;
cfgVHT.ChannelBandwidth = 'CBW80';
y = wlanVHTSIGA(cfgVHT);
size(y)
```

```
ans =
```

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

```
   640      1
```

The 80 MHz waveform has two OFDM symbols and is a total of 640 samples long. Each symbol contains 320 samples.

Extract VHT-SIG-A Bandwidth Information

Generate the VHT-SIG-A waveform for a 40 MHz transmission packet.

Create a VHT configuration object, and assign a 40 MHz channel bandwidth.

```
cfgVHT = wlanVHTConfig;
cfgVHT.ChannelBandwidth = 'CBW40';
```

Generate the VHT-SIG-A waveform and information bits.

```
[y,bits] = wlanVHTSIGA(cfgVHT);
```

Extract the bandwidth from the returned bits and analyze. The bandwidth information is contained in the first two bits.

```
bwBits = bits(1:2);
bi2de(bwBits)
```

```
ans =

    1
    0
```

As defined in IEEE Std 802.11ac-2013, Table 22-12, a value of `'1'` corresponds to 40 MHz bandwidth.

Input Arguments

`cfg` — Format configuration
`wlanVHTConfig` object

Format configuration, specified as a `wlanVHTConfig` object. The `wlanVHTSIGA` function uses the object properties indicated.

User Scenario	Applicable Object Properties
Multi-user	`ChannelBandwidth`, `NumUsers`, `UserPositions`, `NumTransmitAntennas`, `NumSpaceTimeStreams`, `SpatialMapping`, `STBC`, `ChannelCoding`, `GuardInterval`, and `GroupID`
Single user	`ChannelBandwidth`, `NumUsers`, `NumTransmitAntennas`, `NumSpaceTimeStreams`, `SpatialMapping`, `STBC`, `MCS`, `ChannelCoding`, `GuardInterval`, `GroupID`, `Beamforming`, and `PartialAID`

`ChannelBandwidth` — Channel bandwidth

`'CBW80'` (default) | `'CBW20'` | `'CBW40'` | `'CBW160'`

Channel bandwidth, specified as `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`. If the transmission has multiple users, the same channel bandwidth is applied to all users. The default value of `'CBW80'` sets the channel bandwidth to 80 MHz.

Data Types: `char`

`NumUsers` — Number of users

1 (default) | 2 | 3 | 4

Number of users, specified as 1, 2, 3, or 4.

Data Types: `double`

`UserPositions` — Position of users

[0 1] (default) | row vector of integers for 0 to 3 in strictly increasing order

Position of users, specified as an integer row vector with length equal to `NumUsers` and element values from 0 to 3 in a strictly increasing order. This property applies when `NumUsers` property is set to 2, 3 or 4. The default value of this property is [0 1].

Example: `[0 1 2]`indicates that users occupy the specified positions.

Data Types: `double`

`NumTransmitAntennas` — Number of transmit antennas

1 (default) | integer from 1 to 8

Number of transmit antennas, specified as a scalar integer from 1 to 8.

Data Types: `double`

`NumSpaceTimeStreams` — Number of space-time streams
1 (default) | integer from 1 to 8 | 1-by-N_{Users} vector of integers from 1 to 4

Number of space-time streams in the transmission, specified as a scalar or vector.

- For a single user, the number of space-time streams is a scalar integer from 1 to 8.
- For multiple users, the number of space-time streams is a 1-by-N_{Users} vector of integers from 1 to 4, where the vector length, N_{Users}, is an integer from 1 to 4.

Example: `[1 3 2]` is the number of space-time streams for each user.

Note: The sum of the space-time stream vector elements must not exceed eight.

Data Types: `double`

`SpatialMapping` — Spatial mapping scheme
`'Direct'` (default) | `'Hadamard'` | `'Fourier'` | `'Custom'`

Spatial mapping scheme, specified as `'Direct'`, `'Hadamard'`, `'Fourier'`, or `'Custom'`. The default value of `'Direct'` applies when `NumTransmitAntennas` and `NumSpaceTimeStreams` are equal.

Data Types: `char`

`Beamforming` — Option to disable the signaling of a transmission with beamforming
true (default) | false

Option to disable the signaling of a transmission with beamforming, specified as a logical. Beamforming is performed when setting is `true`. This property applies when `NumUsers` equals 1 and `SpatialMapping` is set to `'Custom'`. The `SpatialMappingMatrix` property specifies the beamforming steering matrix.

Data Types: `logical`

`STBC` — Option to enable space-time block coding
false (default) | true

Option to enable space-time block coding (STBC) of the PPDU data field, specified as a logical. STBC transmits multiple copies of the data stream across assigned antennas.

- When set to `false`, no STBC is applied to the data field, and the number of space-time streams is equal to the number of spatial streams.
- When set to `true`, STBC is applied to the data field, and the number of space-time streams is double the number of spatial streams.

See IEEE 802.11ac-2013, Section 22.3.10.9.4 for further description.

Note: `STBC` is relevant for single-user transmissions only.

Data Types: `logical`

`MCS` — Modulation and coding scheme

0 (default) | integer from 0 to 9 | 1-by-N_{Users} vector of integers

Modulation and coding scheme used in transmitting the current packet, specified as a scalar or vector.

- For a single user, the MCS value is a scalar integer from 0 to 9.
- For multiple users, MCS is a 1-by-N_{Users} vector of integers or a scalar with values from 0 to 9, where the vector length, N_{Users}, is an integer from 1 to 4.

MCS	Modulation	Coding Rate
0	`BPSK`	`1/2`
1	`QPSK`	`1/2`
2	`QPSK`	`3/4`
3	`16QAM`	`1/2`
4	`16QAM`	`3/4`
5	`64QAM`	`2/3`
6	`64QAM`	`3/4`
7	`64QAM`	`5/6`
8	`256QAM`	`3/4`
9	`256QAM`	`5/6`

Data Types: `double`

`ChannelCoding` — Type of forward error correction coding
`'BCC'` (default)

This property is read only.

Type of forward error correction coding for the data field, specified as `'BCC'` to indicate binary convolutional coding. LDPC coding is not currently supported.

Data Types: `char`

`GuardInterval` — Cyclic prefix length for the data field within a packet
`'Long'` (default) | `'Short'`

Cyclic prefix length for the data field within a packet, specified as `'Long'` or `'Short'`.

- The long guard interval length is 800 ns.
- The short guard interval length is 400 ns.

Data Types: `char`

`GroupID` — Group identification number
63 (default) | integer from 0 to 63

Group identification number, specified as a scalar integer from 0 to 63.

- A group identification number of either 0 or 63 indicates a VHT single-user PPDU.
- A group identification number from 1 to 62 indicates a VHT multi-user PPDU.

Data Types: `double`

`PartialAID` — Abbreviated indication of the PSDU recipient
275 (default) | integer from 0 to 511

Abbreviated indication of the PSDU recipient, specified as a scalar integer from 0 to 511.

- For an uplink transmission, the partial identification number is the last nine bits of the basic service set identifier (BSSID).
- For a downlink transmission, the partial identification of a client is an identifier that combines the association ID with the BSSID of its serving AP.

For more information, see IEEE Std 802.11ac-2013, Table 22-1.

Data Types: `double`

Output Arguments

y — VHT-SIG-A time-domain waveform

matrix

"VHT-SIG-A" on page 1-243 time-domain waveform, returned as an N_S-by-N_T matrix. N_S is the number of time-domain samples, and N_T is the number of transmit antennas.

N_S is proportional to the channel bandwidth. The time-domain waveform consists of two symbols. Each symbol contains 80 time samples per 20 MHz channel.

`ChannelBandwidth`	N_S
`'CBW20'`	160
`'CBW40'`	320
`'CBW80'`	640
`'CBW160'`	1280

See "VHT-SIG-A Processing" on page 1-245 for waveform generation details.

Data Types: `double`
Complex Number Support: Yes

`bits` — Signaling bits used for the VHT-SIG-A field

48-bit column vector

Signaling bits used for the "VHT-SIG-A" on page 1-243, returned as a 48-bit column vector.

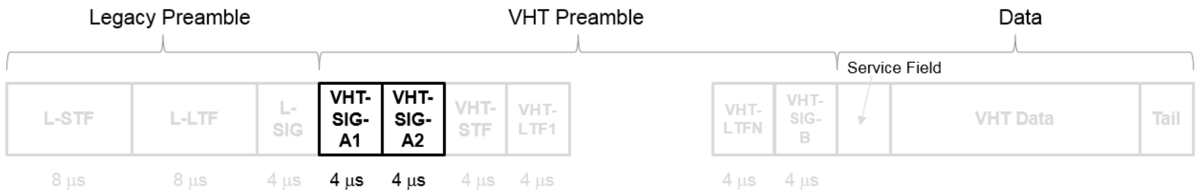
Data Types: `int8`

More About

VHT-SIG-A

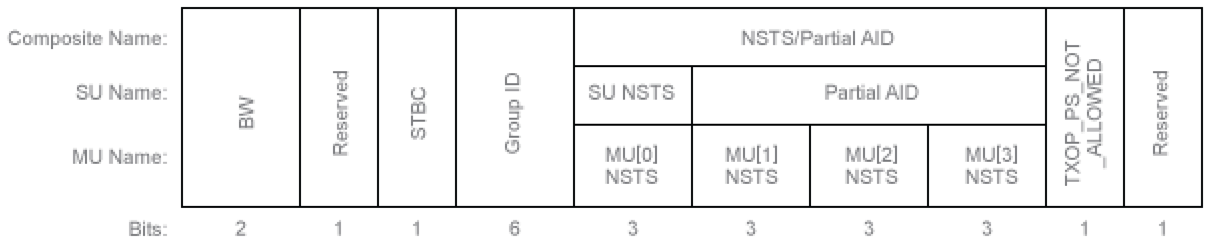
The very high throughput signal A (VHT-SIG-A) field contains information required to interpret VHT format packets. Similar to the non-HT signal (L-SIG) field for the non-HT OFDM format, this field stores the actual rate value, channel coding, guard interval, MIMO scheme, and other configuration details for the VHT format packet. Unlike the HT-SIG field, this field does not store the packet length information. Packet length information is derived from L-SIG and is captured in the VHT-SIG-B field for the VHT format.

The VHT-SIG-A field consists of two symbols: VHT-SIG-A1 and VHT-SIG-A2. These symbols are located between the L-SIG and the VHT-STF portion of the VHT format PPDU.

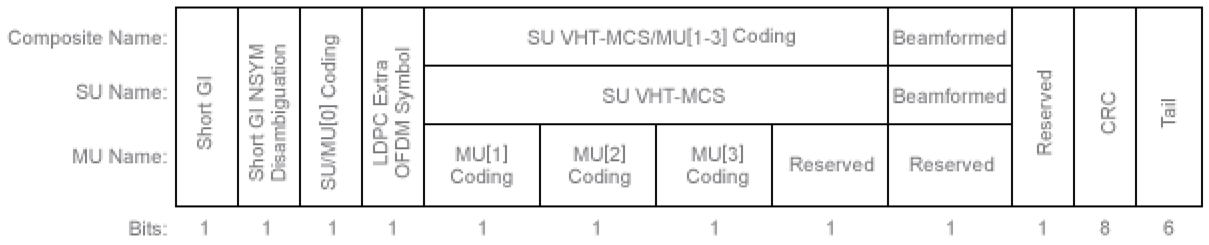


The VHT-SIG-A field is defined in IEEE Std 802.11ac-2013, Section 22.3.8.3.3.

VHT-SIG-A1 Structure



VHT-SIG-A2 Structure



The VHT-SIG-A field includes these components. The bit field structures for VHT-SIG-A1 and VHT-SIG-A2 vary for single user or multi-user transmissions.

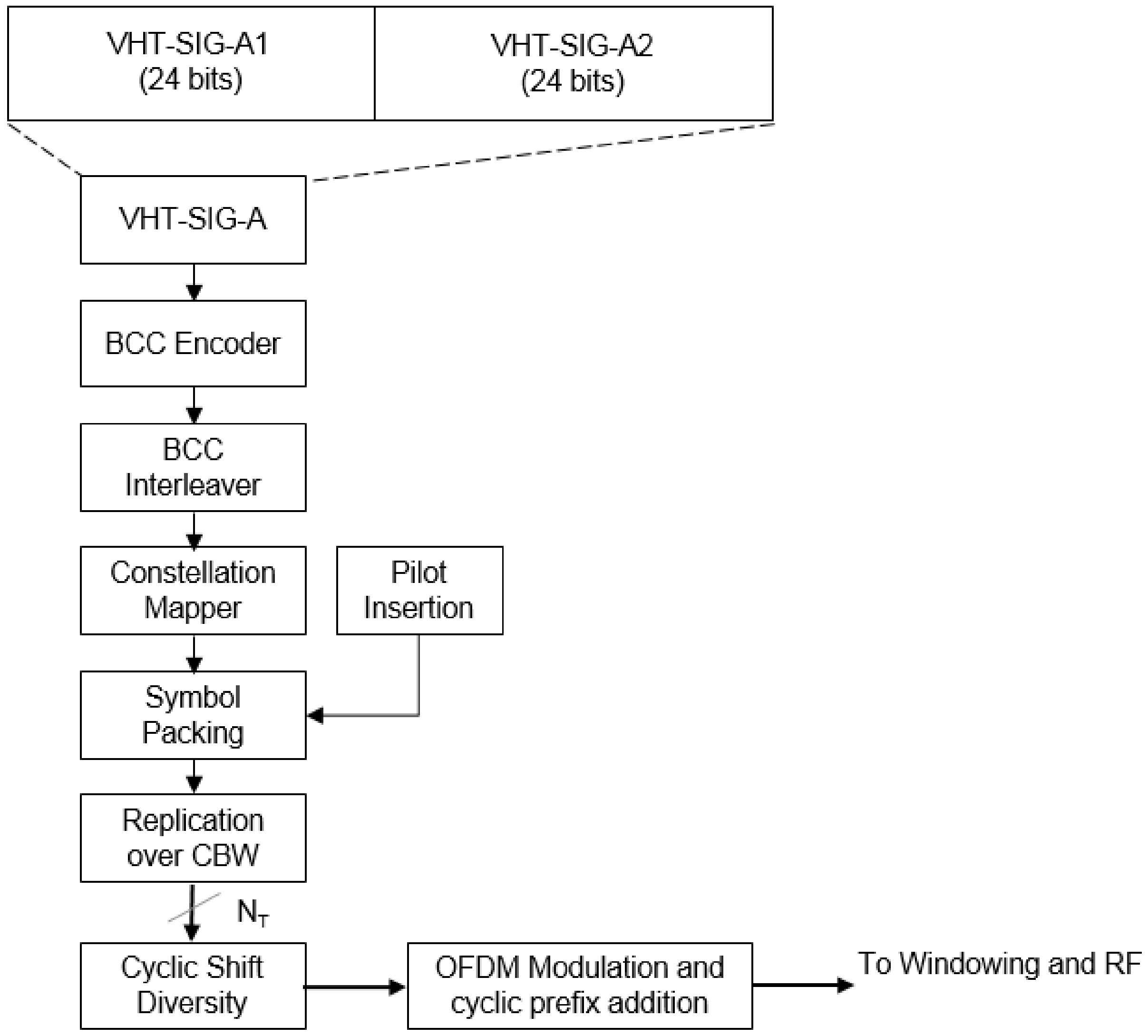
- **BW** — A two-bit field that indicates 0 for 20 MHz, 1 for 40 MHz, 2 for 80 MHz, or 3 for 160 MHz.
- **STBC** — A bit that indicates the presence of space-time block coding.
- **Group ID** — A six-bit field that indicates the group and user position assigned to a STA.
- **N_{STS}** — A three-bit field for a single user or 4 three-bit fields for a multi-user scenario, that indicates the number of space-time streams per user.
- **Partial AID** — An identifier that combines the association ID and the BSSID.
- **TXOP_PS_NOT_ALLOWED** — An indicator bit that shows if client devices are allowed to enter dose state. This bit is set to false when the VHT-SIG-A structure is populated, indicating that the client device is allowed to enter dose state.
- **Short GI** — A bit that indicates use of the 400 ns guard interval.
- **Short GI NSYM Disambiguation** — A bit that indicates if an extra symbol is required when the short GI is used.
- **SU/MU[0] Coding** — A bit field that indicates if convolutional or LDPC coding is used for a single user or for user MU[0] in a multi-user scenario.
- **LDPC Extra OFDM Symbol** — A bit that indicates if an extra OFDM symbol is required to transmit the data field.
- **MCS** — A four-bit field.
  - For a single user scenario, it indicates the modulation and coding scheme used.
  - For a multi-user scenario, it indicates use of convolutional or LDPC coding and the MCS setting is conveyed in the VHT-SIG-B field.
- **Beamformed** — An indicator bit set to 1 when a beamforming matrix is applied to the transmission.
- **CRC** — An eight-bit field used to detect errors in the VHT-SIG-A transmission.
- **Tail** — A six-bit field used to terminate the convolutional code.

Algorithms

VHT-SIG-A Processing

The “VHT-SIG-A” on page 1-243 field includes information required to process VHT format packets.

For algorithm details, refer to IEEE Std 802.11ac-2013 [1], Section 22.3.4.5. The `wlanVHTSIGA` function performs transmitter processing on the “VHT-SIG-A” on page 1-243 field and outputs the time-domain waveform.



References

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

See Also

`wlanLSIG` | `wlanVHTConfig` | `wlanVHTSIGARecover` | `wlanVHTSTF`

Introduced in R2015b

wlanVHTSIGARecover

Recover VHT-SIG-A information bits

Syntax

```
recBits = wlanVHTSIGARecover(rxSig,chEst,noiseVarEst,cbw)
recBits = wlanVHTSIGARecover(rxSig,chEst,noiseVarEst,cbw,cfgRec)
[recBits,failCRC] = wlanVHTSIGARecover( ___ )
[recBits,failCRC,eqSym] = wlanVHTSIGARecover( ___ )
[recBits,failCRC,eqSym,cpe] = wlanVHTSIGARecover( ___ )
```

Description

`recBits = wlanVHTSIGARecover(rxSig,chEst,noiseVarEst,cbw)` returns the recovered information bits from the “VHT-SIG-A” on page 1-256[25] field. Inputs include the received “VHT-SIG-A” on page 1-256 field, the channel estimate, the noise variance estimate, and the channel bandwidth.

`recBits = wlanVHTSIGARecover(rxSig,chEst,noiseVarEst,cbw,cfgRec)` specifies algorithm information using `wlanRecoveryConfig` object `cfgRec`.

`[recBits,failCRC] = wlanVHTSIGARecover( ___ )` returns the failure status of the CRC check, `failCRC`, using the arguments from previous syntaxes.

`[recBits,failCRC,eqSym] = wlanVHTSIGARecover( ___ )` returns the equalized symbols, `eqSym`.

`[recBits,failCRC,eqSym,cpe] = wlanVHTSIGARecover( ___ )` returns the common phase error, `cpe`.

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

Examples

Recover VHT-SIG-A Information Bits

Recover the information bits in the VHT-SIG-A field by performing channel estimation on the L-LTF over a 1x2 quasi-static fading channel

Create a `wlanVHTConfig` object having a channel bandwidth of 80 MHz. Generate L-LTF and VHT-SIG-A field signals using this object.

```
cfg = wlanVHTConfig('ChannelBandwidth','CBW80');
txLLTF = wlanLLTF(cfg);
[txVHTSIGA, txBits] = wlanVHTSIGA(cfg);
chanBW = cfg.ChannelBandwidth;
noiseVarEst = 0.1;
```

Pass the L-LTF and VHT-SIG-A signals through a 1x2 quasi-static fading channel with AWGN.

```
H = 1/sqrt(2)*complex(randn(1,2),randn(1,2));
rxLLTF    = awgn(txLLTF*H,10);
rxVHTSIGA = awgn(txVHTSIGA*H,10);
```

Perform channel estimation based on the L-LTF.

```
demodLLTF = wlanLLTFDemodulate(rxLLTF,chanBW,1);
chanEst = wlanLLTFChannelEstimate(demodLLTF,chanBW);
```

Recover the VHT-SIG-A. Verify that the CRC check was successful.

```
[rxBits,failCRC] = wlanVHTSIGARecover(rxVHTSIGA,chanEst,noiseVarEst,'CBW80');
failCRC
```

```
failCRC =

     0
```

The CRC failure check returns a `0`, indicating that the CRC passed.

Compare the transmitted bits to the received bits. Confirm that the reported CRC result is correct because the output matches the input.

```
isequal(txBits,rxBits)
```

```
ans =

     1
```

Recover VHT-SIG-A Using Zero-Forcing Equalizer

Recover the VHT-SIG-A in an AWGN channel. Configure the VHT signal to have a 160 MHz channel bandwidth, one space-time stream, and one receive antenna.

Create a `wlanVHTConfig` object having a channel bandwidth of 160 MHz. Using the object to create a VHT-SIG-A waveform.

```
cfg = wlanVHTConfig('ChannelBandwidth','CBW160');
```

Generate L-LTF and VHT-SIG-A field signals.

```
txLLTF = wlanLLTF(cfg);
[txSig,txBits] = wlanVHTSIGA(cfg);
chanBW = cfg.ChannelBandwidth;
noiseVar = 0.1;
```

Pass the transmitted VHT-SIG-A through an AWGN channel.

```
ch = comm.AWGNChannel('NoiseMethod','Variance','Variance',noiseVar);
rxLLTF = step(ch,txLLTF);
rxSig = step(ch,txSig);
```

Using `wlanRecoveryConfig`, set the equalization method to zero-forcing, `'ZF'`.

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

Perform channel estimation based on the L-LTF.

```
demodLLTF = wlanLLTFDemodulate(rxLLTF,chanBW,1);
chanEst = wlanLLTFChannelEstimate(demodLLTF,chanBW);
```

Recover the VHT-SIG-A. Verify that there are no bit errors in the received information.

```
[rxBits,crcFail] = wlanVHTSIGARecover(rxSig,chanEst,noiseVar,'CBW160',cfgRec);
crcFail
```

```
crcFail =
     0
```

The CRC failure check returns a `0`, indicating the CRC passed. Comparing the transmitted bits to the received bits reconfirms the reported CRC result because the output matches the input.

```
biterr(txBits,rxBits)
```

```
ans =

     0
```

Recover VHT-SIG-A in 2x2 MIMO Channel

Recover VHT-SIG-A in a 2x2 MIMO channel with AWGN. Confirm that the `CRC` check passes.

Configure a 2x2 MIMO VHT channel.

```
chanBW = 'CBW20';
cfgVHT = wlanVHTConfig('ChannelBandwidth', chanBW, 'NumTransmitAntennas', 2, 'NumSpace
```

Generate L-LTF and VHT-SIG-A waveforms.

```
txLLTF  = wlanLLTF(cfgVHT);
txVHTSIGA = wlanVHTSIGA(cfgVHT);
```

Pass the L-LTF and VHT-SIG-A waveforms through a 2×2 MIMO channel with white noise.

```
ch = comm.MIMOChannel('SampleRate', 20e6);
rxLLTF = awgn(step(ch, txLLTF), 15);
rxVHTSIGA = awgn(step(ch, txVHTSIGA),15);
```

Demodulate the L-LTF signal. To generate a channel estimate, use the demodulated L-LTF.

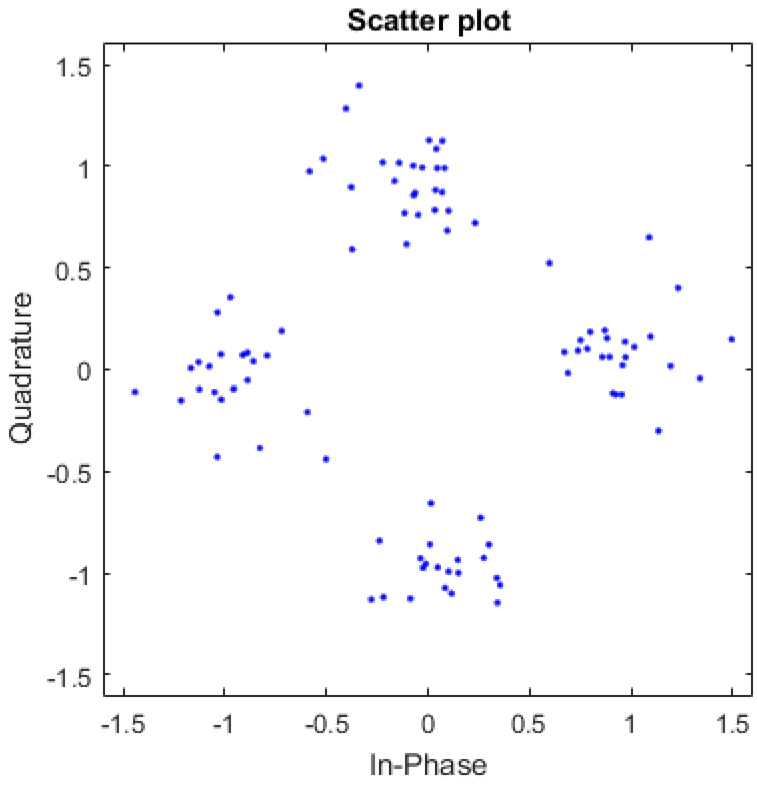
```
demodLLTF = wlanLLTFDemodulate(rxLLTF, chanBW, 1);
chanEst = wlanLLTFChannelEstimate(demodLLTF, chanBW);
```

Recover the information bits in VHT-SIG-A.

```
[recVHTSIGABits, failCRC, eqSym] = wlanVHTSIGARecover(rxVHTSIGA, chanEst, 0, chanBW);
```

Visualize the scatter plot of the equalized symbols, `eqSym`.

```
scatterplot(eqSym(:))
```



Input Arguments

`rxSig` — Received VHT-SIG-A

matrix

Received VHT-SIG-A field, specified as an N_S-by-N_R matrix. N_S is the number of samples and increases with channel bandwidth.

Channel Bandwidth	N_S
`'CBW20'`	160
`'CBW40'`	320
`'CBW80'`	640
`'CBW160'`	1280

N_R is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

`chEst` — Channel estimate
3-D array

Channel estimate, specified as an N_{ST}-by-1-by-N_{R} array. N_{ST} is the number of occupied subcarriers and increases with channel bandwidth.

Channel Bandwidth	N_{ST}
`'CBW20'`	52
`'CBW40'`	104
`'CBW80'`	208
`'CBW160'`	416

N_{R} is the number of receive antennas.

The channel estimate is based on the “L-LTF” on page 1-256.

Data Types: `double`
Complex Number Support: Yes

`noiseVarEst` — Noise variance estimate
nonnegative scalar

Noise variance estimate, specified as a nonnegative scalar.

Data Types: `double`

`cbw` — Channel bandwidth
`'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth in MHz, specified as one of these strings: `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`.

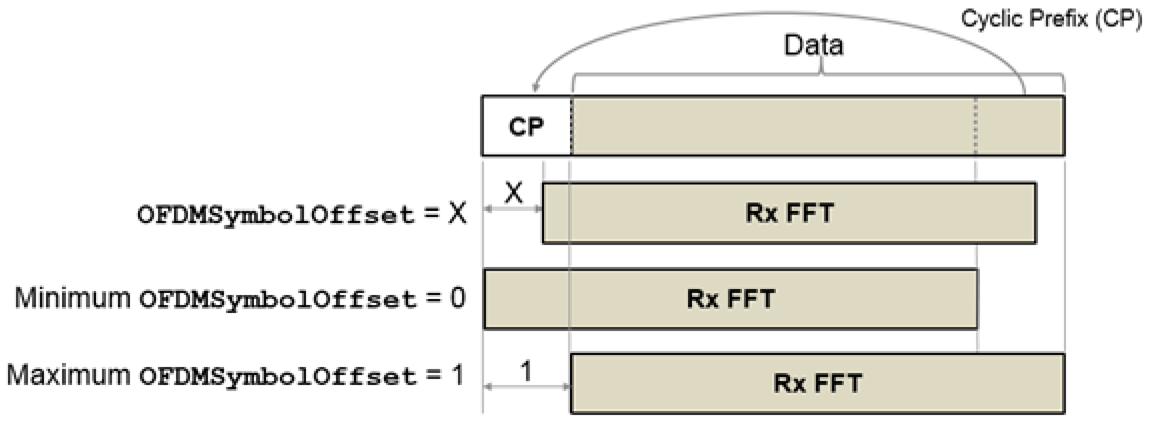
Data Types: `char`

`cfgRec` — Algorithm parameters
`wlanRecoveryConfig` object

Algorithm parameters, specified as a `wlanRecoveryConfig` object. The function uses these properties:

`OFDMSymbolOffset` — OFDM symbol sampling offset
0.75 (default) | scalar value from 0 to 1

OFDM symbol sampling offset represented as a fraction of the cyclic prefix (CP) length, specified as a scalar value from 0 to 1. This value indicates the start location for OFDM demodulation, relative to the beginning of the cyclic prefix. `OFDMSymbolOffset` = 0 represents the start of the cyclic prefix and `OFDMSymbolOffset` = 1 represents the end of the cyclic prefix.



Data Types: `double`

`EqualizationMethod` — Equalization method
`'MMSE'` (default) | `'ZF'`

Equalization method, specified as `'MMSE'` or `'ZF'`.

- `'MMSE'` indicates that the receiver uses a minimum mean square error equalizer.

- `'ZF'` indicates that the receiver uses a zero-forcing equalizer.

Example: `'ZF'`

Data Types: `char`

`PilotPhaseTracking` — Pilot phase tracking

`'PreEQ'` (default) | `'None'`

Pilot phase tracking, specified as either of these strings: `'PreEQ'` or `'None'`. If `'PreEQ'` is specified, pilot phase tracking is enabled and is performed prior to any equalization operation. If `'None'` is specified, pilot phase tracking does not occur.

Data Types: `char`

Output Arguments

`recBits` — Recovered VHT-SIG-A information bits

column vector

Recovered VHT-SIG-A information bits, returned as a 48-by-1 column vector. See “VHT-SIG-A” on page 1-256 for more information.

Data Types: `int8`

`failCRC` — CRC failure check

`true` | `false`

CRC failure check, returned as `true` if the CRC check fails or `false` if the CRC check passes.

Data Types: `logical`

`eqSym` — Equalized symbols

matrix

Equalized symbols at the data carrying subcarriers, returned as 48-by-2 matrix. Each 20 MHz channel bandwidth segment has two symbols and 48 data carrying subcarriers. These segments are combined into a single 48-by-2 matrix that comprises the “VHT-SIG-A” on page 1-256 field.

Data Types: `double`
Complex Number Support: Yes

cpe — Common phase error
column vector

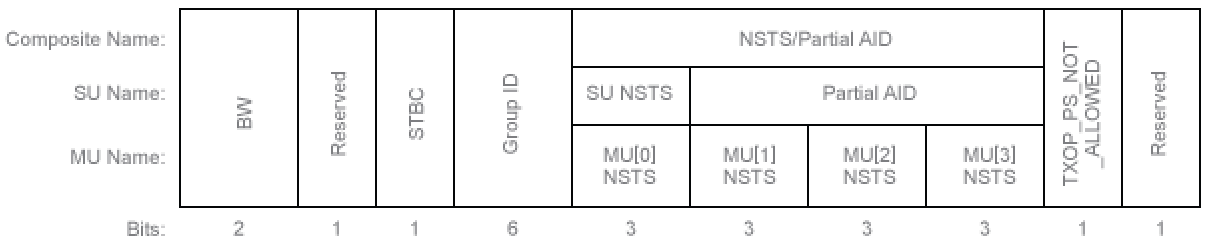
Common phase error in radians, returned as a 2-by-1 column vector.

More About

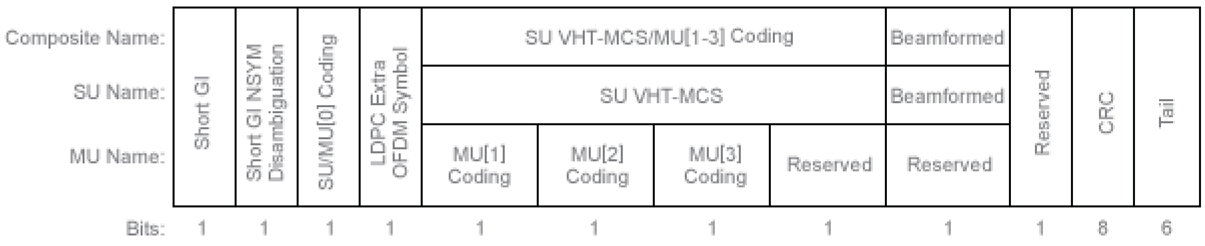
VHT-SIG-A

The very high throughput signal A (VHT-SIG-A) field consists of two symbols: VHT-SIG-A1 and VHT-SIG-A2. The VHT-SIG-A field carries information required to interpret VHT PPDU information.

VHT-SIG-A1 Structure



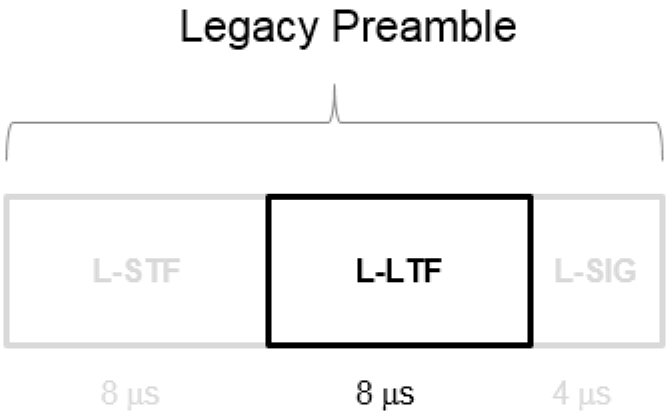
VHT-SIG-A2 Structure



For VHT-SIG-A field bit details, refer to IEEE Std 802.11ac-2013 [1], Table 22-12.

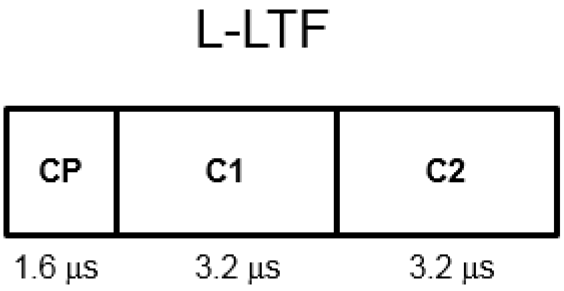
L-LTF

The legacy long training field (L-LTF) is the second field in the legacy preamble for 802.11a/g.



It is also the second field of the HT-mixed format preamble used in 802.11n and the VHT format preamble used in 802.11ac. Channel estimation, frequency offset estimation, and time synchronization rely on the L-LTF. The long OFDM training symbol consists of 53 subcarriers. IEEE Std 802.11-2012, Equation 18-8 and Equation 18-9 define the L-LTF.

The L-LTF, which is 8 µs in length, is composed of a 1.6 µs cyclic prefix (CP) followed by two identical 3.2 µs long training symbols (C1 and C2). The cyclic prefix (CP) consists of the second half of the long training symbol.



PPDU

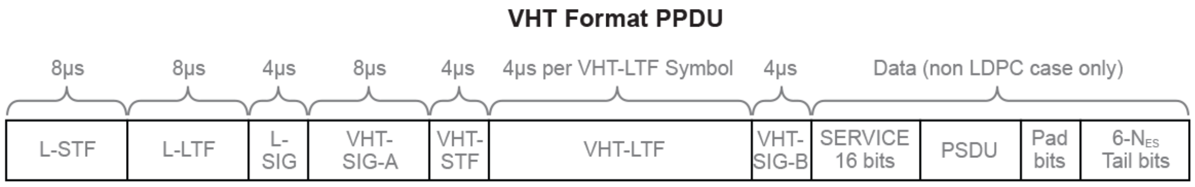
PLCP protocol data unit

The PPDU is the complete PLCP frame, including PLCP headers, MAC headers, the MAC data field, and the MAC and PLCP trailers.

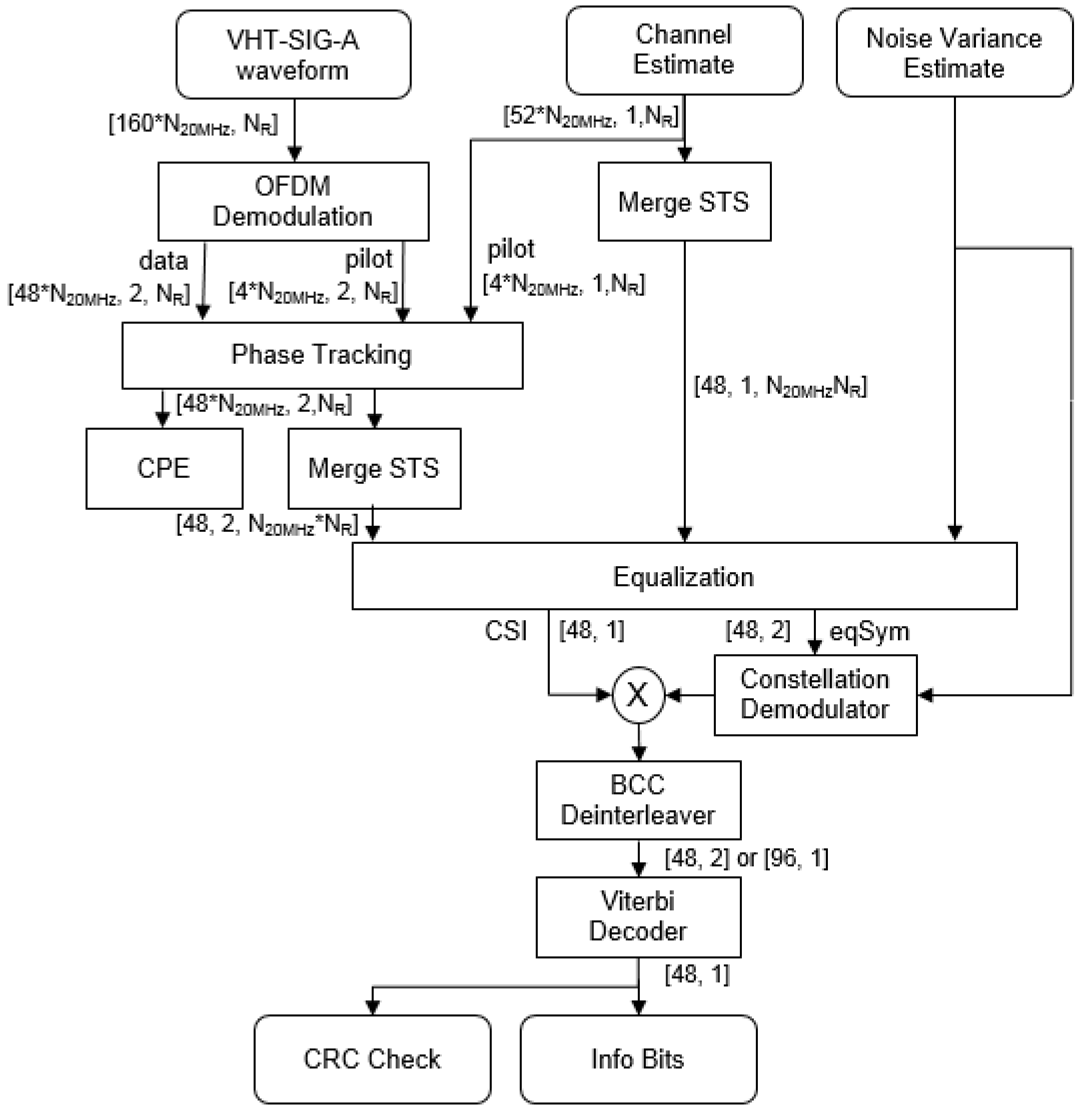
Algorithms

VHT-SIG-A Recovery

The “VHT-SIG-A” on page 1-256 field consists of two symbols and resides between the L-SIG field and the VHT-STF portion of the packet structure for the VHT format “PPDU” on page 1-258.



For single-user packets, you can recover the length information from the L-SIG and VHT-SIG-A field information. Therefore, it is not strictly required for the receiver to decode the “VHT-SIG-A” on page 1-256 field.



For “VHT-SIG-A” on page 1-256 details, refer to IEEE Std 802.11ac-2013 [1], Section 22.3.4.5, and Perahia [2], Section 7.3.2.1.

References

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

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

See Also

`wlanLLTF` | `wlanLLTFChannelEstimate` | `wlanLLTFDemodulate` | `wlanRecoveryConfig` | `wlanVHTSIGA`

Introduced in R2015b

wlanVHTSIGB

Generate VHT-SIG-B waveform

Syntax

```
y= wlanVHTSIGB(cfg)
[y,bits] = wlanVHTSIGB(cfg)
```

Description

`y= wlanVHTSIGB(cfg)` generates a "VHT-SIG-B" on page 1-266[26] time-domain waveform for the specified configuration object. See "VHT-SIG-B Processing" on page 1-268 for waveform generation details.

`[y,bits] = wlanVHTSIGB(cfg)` also outputs "VHT-SIG-B" on page 1-266 information bits.

Examples

Generate VHT-SIG-B Waveform

Generate the VHT-SIG-B waveform for an 80 MHz transmission packet.

Create a VHT configuration object, assign an 80 MHz channel bandwidth, and generate the waveform.

```
cfgVHT = wlanVHTConfig('ChannelBandwidth','CBW80');
vhtsigb = wlanVHTSIGB(cfgVHT);
size(vhtsigb)
```

```
ans =

   320     1
```

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

The 80 MHz waveform has one OFDM symbol and is a total of 320 samples long.

Input Arguments

`cfg` — Format configuration
`wlanVHTConfig` object

Format configuration, specified as a `wlanVHTConfig` object. The `wlanVHTSIGB` function uses the object properties indicated.

`ChannelBandwidth` — Channel bandwidth
`'CBW80'` (default) | `'CBW20'` | `'CBW40'` | `'CBW160'`

Channel bandwidth, specified as `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`. If the transmission has multiple users, the same channel bandwidth is applied to all users. The default value of `'CBW80'` sets the channel bandwidth to 80 MHz.

Data Types: `char`

`NumUsers` — Number of users
1 (default) | 2 | 3 | 4

Number of users, specified as 1, 2, 3, or 4.

Data Types: `double`

`NumTransmitAntennas` — Number of transmit antennas
1 (default) | integer from 1 to 8

Number of transmit antennas, specified as a scalar integer from 1 to 8.

Data Types: `double`

`NumSpaceTimeStreams` — Number of space-time streams
1 (default) | integer from 1 to 8 | 1-by-N_{Users} vector of integers from 1 to 4

Number of space-time streams in the transmission, specified as a scalar or vector.

- For a single user, the number of space-time streams is a scalar integer from 1 to 8.
- For multiple users, the number of space-time streams is a 1-by-N_{Users} vector of integers from 1 to 4, where the vector length, N_{Users}, is an integer from 1 to 4.

Example: `[1 3 2]` is the number of space-time streams for each user.

Note: The sum of the space-time stream vector elements must not exceed eight.

Data Types: `double`

`SpatialMapping` — Spatial mapping scheme

`'Direct'` (default) | `'Hadamard'` | `'Fourier'` | `'Custom'`

Spatial mapping scheme, specified as `'Direct'`, `'Hadamard'`, `'Fourier'`, or `'Custom'`. The default value of `'Direct'` applies when `NumTransmitAntennas` and `NumSpaceTimeStreams` are equal.

Data Types: `char`

`SpatialMappingMatrix` — Spatial mapping matrix

1 (default) | scalar | matrix | 3-D array

Spatial mapping matrix, specified as a scalar, matrix, or 3-D array. Use this property to apply a beamforming steering matrix, and to rotate and scale the constellation mapper output vector. If applicable, scale the space-time block coder output instead. `SpatialMappingMatrix` applies when the `SpatialMapping` property is set to `'Custom'`. For more information, see IEEE Std 802.11-2012, Section 20.3.11.11.2.

- When specified as a scalar, a constant value applies to all the subcarriers.
- When specified as a matrix, the size must be N_{STS_Total}-by-N_T. The spatial mapping matrix applies to all the subcarriers. N_{STS_Total} is the sum of space-time streams for all users, and N_T is the number of transmit antennas.
- When specified as a 3-D array, the size must be N_{ST}-by-N_{STS_Total}-by-N_T. N_{ST} is the sum of the occupied data (N_{SD}) and pilot (N_{SP}) subcarriers, as determined by `ChannelBandwidth`. N_{STS_Total} is the sum of space-time streams for all users. N_T is the number of transmit antennas.

  N_{ST} increases with channel bandwidth.

`ChannelBandwidth`	Number of Occupied Subcarriers (N_{ST})	Number of Data Subcarriers (N_{SD})	Number of Pilot Subcarriers (N_{SP})
`'CBW20'`	56	52	4

ChannelBandwidth	Number of Occupied Subcarriers (N_{ST})	Number of Data Subcarriers (N_{SD})	Number of Pilot Subcarriers (N_{SP})
'CBW40'	114	108	6
'CBW80'	242	234	8
'CBW160'	484	468	16

The calling function normalizes the spatial mapping matrix for each subcarrier.

Example: [0.5 0.3 0.4; 0.4 0.5 0.8] represents a spatial mapping matrix having two space-time streams and three transmit antennas.

Data Types: `double`
Complex Number Support: Yes

`MCS` — Modulation and coding scheme

0 (default) | integer from 0 to 9 | 1-by-N_{Users} vector of integers

Modulation and coding scheme used in transmitting the current packet, specified as a scalar or vector.

- For a single user, the MCS value is a scalar integer from 0 to 9.
- For multiple users, MCS is a 1-by-N_{Users} vector of integers or a scalar with values from 0 to 9, where the vector length, N_{Users}, is an integer from 1 to 4.

MCS	Modulation	Coding Rate
0	BPSK	1/2
1	QPSK	1/2
2	QPSK	3/4
3	16QAM	1/2
4	16QAM	3/4
5	64QAM	2/3
6	64QAM	3/4
7	64QAM	5/6
8	256QAM	3/4
9	256QAM	5/6

Data Types: `double`

`APEPLength` — Number of bytes in the A-MPDU pre-EOF padding

1024 (default) | integer from 0 to 1,048,575 | vector of integers

Number of bytes in the A-MPDU pre-EOF padding, specified as a scalar integer or vector of integers.

- For a single user, `APEPLength` is a scalar integer from 0 to 1,048,575.
- For multi-user, `APEPLength` is a 1-by-N_{Users} vector of integers or a scalar with values from 0 to 1,048,575, where the vector length, N_{Users}, is an integer from 1 to 4.
- `APEPLength = 0` for a null data packet (NDP).

`APEPLength` is used internally to determine the number of OFDM symbols in the data field. For more information, see IEEE Std 802.11ac-2013, Table 22-1.

Data Types: `double`

Output Arguments

y — VHT-SIG-B time-domain waveform

matrix

"VHT-SIG-B" on page 1-266 time-domain waveform, returned as an N_S-by-N_T matrix. N_S is the number of time-domain samples and N_T is the number of transmit antennas.

N_S is proportional to the channel bandwidth. Each symbol contains 80 time samples per 20 MHz channel.

`ChannelBandwidth`	N_S
`'CBW20'`	80
`'CBW40'`	160
`'CBW80'`	320
`'CBW160'`	640

See "VHT-SIG-B Processing" on page 1-268. for waveform generation details.

Data Types: `double`

Complex Number Support: Yes

`bits` — Signaling bits used for the VHT-SIG-B field

N_{bits} column vector

Signaling bits used for “VHT-SIG-B” on page 1-266 field, returned as an N_{bits} column vector. N_{bits} is the number of bits.

The number of output bits changes with the channel bandwidth.

`ChannelBandwidth`	N_b
`'CBW20'`	26
`'CBW40'`	27
`'CBW80'`	29
`'CBW160'`	29

See “VHT-SIG-B Processing” on page 1-268. for waveform generation details.

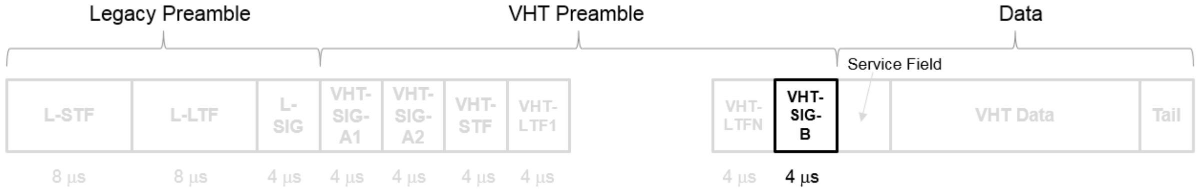
Data Types: `int8`

More About

VHT-SIG-B

The very high throughput signal B field (VHT-SIG-B) is used for multi-user scenario to set up the data rate and to fine-tune MIMO reception. It is modulated using MCS 0 and is transmitted in a single OFDM symbol.

The VHT-SIG-B field consists of a single OFDM symbol located between the VHT-LTF and the data portion of the VHT format PPDU.



The very high throughput signal B (VHT-SIG-B) field contains the actual rate and A-MPDU length value per user. The VHT-SIG-B is defined in IEEE Std 802.11ac-2013, Section 22.3.8.3.6, and Table 22–14. The number of bits in the VHT-SIG-B field varies with the channel bandwidth and the assignment depends on whether single user or multi-user scenario in allocated. For single user configurations, the same information is available in the L-SIG field but the VHT-SIG-B field is included for continuity purposes.

Field	VHT MU PPDU Allocation (bits)			VHT SU PPDU Allocation (bits)			Description
	20 MHz	40 MHz	80 MHz, 160 MHz	20 MHz	40 MHz	80 MHz, 160 MHz	
VHT-SIG-B	B0-15 (16)	B0-16 (17)	B0-18 (19)	B0-16 (17)	B0-18 (19)	B0-20 (21)	A variable-length field that indicates the size of the data payload in four-byte units. The length of the field depends on the channel bandwidth.
VHT-MCS	B16-19 (4)	B17-20 (4)	B19-22 (4)	N/A	N/A	N/A	A four-bit field that is included for multi-user scenarios only.
Reserved	N/A	N/A	N/A	B17–19 (3)	B19-20 (2)	B21-22 (2)	All ones
Tail	B20-25 (6)	B21-26 (6)	B23-28 (6)	B20-25 (6)	B21-26 (6)	B23-28 (6)	Six zero-bits

Field	VHT MU PPDU Allocation (bits)			VHT SU PPDU Allocation (bits)			Description
	20 MHz	40 MHz	80 MHz, 160 MHz	20 MHz	40 MHz	80 MHz, 160 MHz	
							used to terminate the convolutional code.
Total # bits	26	27	29	26	27	29	
Bit field repetitio	1	2	4 *For 160 MHz, the 80 MHz channel is repeated twice.*	1	2	4 *For 160 MHz, the 80 MHz channel is repeated twice.*	

For a null data packet (NDP), the VHT-SIG-B bits are set according to IEEE Std 802.11ac-2013, Table 22-15.

Algorithms

VHT-SIG-B Processing

The “VHT-SIG-B” on page 1-266 field is used to set up the data rate and to fine-tune MIMO reception. For single user packets, since the length information can be recovered from the L-SIG and VHT-SIG-A field information, it is not strictly required for the receiver to decode the “VHT-SIG-B” on page 1-266 field.

For algorithm details, refer to IEEE Std 802.11ac-2013 [1], Section 22.3.4.8.

References

[1] IEEE Std 802.11ac™-2013 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN

Medium Access Control (MAC) and Physical Layer (PHY) Specifications — Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz.

See Also

`wlanVHTConfig` | `wlanVHTData` | `wlanVHTLTF` | `wlanVHTSIGBRecover`

Introduced in R2015b

wlanVHTSIGBRecover

Recover VHT-SIG-B information bits

Syntax

```
recBits = wlanVHTSIGRecover(rxSig,chEst,noiseVarEst,cbw)
recBits = wlanVHTSIGRecover(rxSig,chEst,noiseVarEst,cbw,cfgRec)

[recBits,eqSym] = wlanVHTSIGRecover( ___ )
[recBits,eqSym,cpe] = wlanVHTSIGRecover( ___ )
```

Description

`recBits = wlanVHTSIGRecover(rxSig,chEst,noiseVarEst,cbw)` returns the recovered information bits from the "VHT-SIG-B" on page 1-278[27] field. Inputs include the received "VHT-SIG-B" on page 1-278 field, the channel estimate, the noise variance estimate, and the channel bandwidth.

`recBits = wlanVHTSIGRecover(rxSig,chEst,noiseVarEst,cbw,cfgRec)` specifies algorithm information using `wlanRecoveryConfig` object `cfgRec`.

`[recBits,eqSym] = wlanVHTSIGRecover( ___ )` returns the equalized symbols, `eqSym`, using the arguments from previous syntaxes.

`[recBits,eqSym,cpe] = wlanVHTSIGRecover( ___ )` returns the common phase error, `cpe`.

Examples

Recover VHT-SIG-B Information Bits

Recover VHT-SIG-B bits in a perfect channel having 80 MHz channel bandwidth, one space-time stream, and one receive antenna.

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

Create a `wlanVHTConfig` object having a channel bandwidth of 80 MHz. Using the object, create a VHT-SIG-B waveform.

```
cfg = wlanVHTConfig('ChannelBandwidth','CBW80');
[txSig,txBits] = wlanVHTSIGB(cfg);
```

For a channel bandwidth of 80 MHz, there are 242 occupied subcarriers. The channel estimate array dimensions for this example must be [Nst,Nsts,Nr] = [242,1,1]. The example assumes a perfect channel and one receive antenna. Therefore, specify the channel estimate as a column vector of ones and the noise variance estimate as zero.

```
chEst = ones(242,1);
noiseVarEst = 0;
```

Recover the VHT-SIG-B. Verify that the received information bits are identical to the transmitted bits.

```
rxBits = wlanVHTSIGBRecover(txSig,chEst,noiseVarEst,'CBW80');
isequal(txBits,rxBits)
```

```
ans =

     1
```

Recover VHT-SIG-B Using Zero-Forcing Equalizer

Recover the VHT-SIG-B using a zero-forcing equalizer in an AWGN channel having 160 MHz channel bandwidth, one space-time stream, and one receive antenna.

Create a `wlanVHTConfig` object having a channel bandwidth of 160 MHz. Using the object, create a VHT-SIG-B waveform.

```
cfg = wlanVHTConfig('ChannelBandwidth','CBW160');
[txSig,txBits] = wlanVHTSIGB(cfg);
```

Pass the transmitted VHT-SIG-B through an AWGN channel.

```
ch = comm.AWGNChannel('NoiseMethod','Variance','Variance',0.1);

rxSig = step(ch,txSig);
```

Using `wlanRecoveryConfig`, set the equalization method to zero-forcing, `'ZF'`.

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

Recover the VHT-SIG-B. Verify that the received information has no bit errors.

```
rxBits = wlanVHTSIGBRecover(rxSig,ones(484,1),0.1,'CBW160',cfgRec);
numErr = biterr(txBits,rxBits)
```

```
numErr =

     0
```

Recover VHT-SIG-B in 2x2 MIMO Channel

Recover VHT-SIG-B in a 2x2 MIMO channel with AWGN. Confirm that the information bits are recovered correctly.

Set the channel bandwidth and the corresponding sample rate.

```
cbw = 'CBW20';
fs = 20e6;
```

Create a VHT configuration object with 20 MHz bandwidth and two transmission paths. Generate the L-LTF and VHT-SIG-B waveforms.

```
vht = wlanVHTConfig('ChannelBandwidth',cbw,'NumTransmitAntennas',2,'NumSpaceTimeStream

txVHTLTF = wlanVHTLTF(vht);
[txVHTSIGB,txVHTSIGBBits] = wlanVHTSIGB(vht);
```

Pass the VHT-LTF and VHT-SIG-B waveforms through a 2x2 TGac channel.

```
tgac = wlanTGacChannel('NumTransmitAntennas',2,'NumReceiveAntennas',2, ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
rxVHTLTF = step(tgac,txVHTLTF);
rxVHTSIGB = step(tgac,txVHTSIGB);
```

Add 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 noise figure.

```
nVar = 10^((-228.6+10*log10(290)+10*log10(fs)+9)/10);
noise = comm.AWGNChannel('NoiseMethod','Variance','Variance',nVar);
```

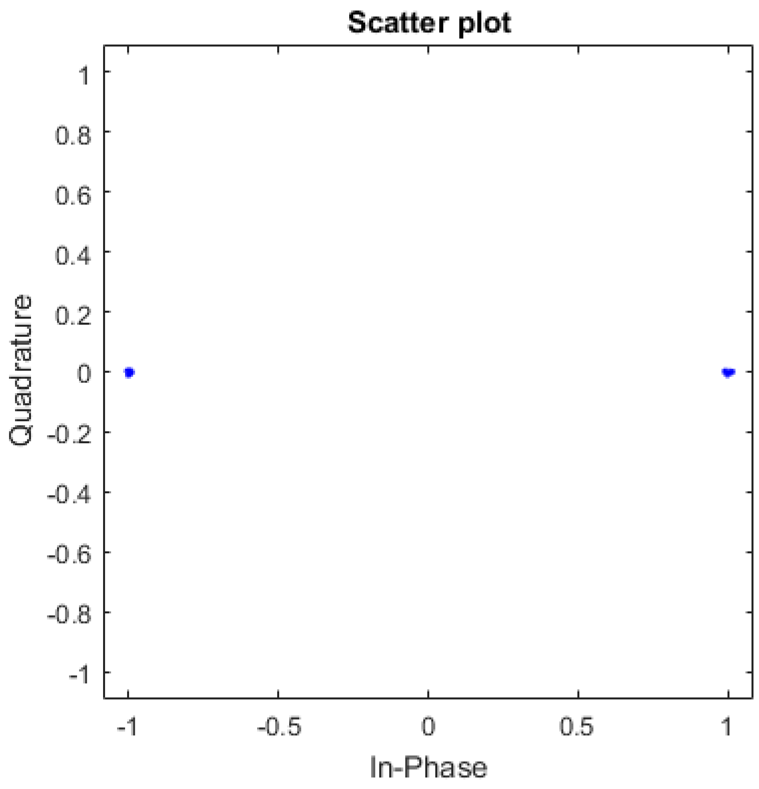
```
rxVHTLTF = step(noise,rxVHTLTF);
rxVHTSIGB = step(noise,rxVHTSIGB);
```

Demodulate the VHT-LTF signal and use it to generate a channel estimate.

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

Recover the VHT-SIG-B information bits. Display the scatter plot of the equalized symbols.

```
[recVHTSIGBBits,eqSym,cpe] = wlanVHTSIGBRecover(rxVHTSIGB,chEst,nVar,cbw);
scatterplot(eqSym)
```



Display the common phase error.

```
cpe
```

```
cpe =

   8.8435e-04
```

Determine the number of errors between the transmitted and received VHT-SIG-B information bits.

```
numErr = biterr(txVHTSIGBBits,recVHTSIGBBits)
```

```
numErr =

     0
```

Input Arguments

`rxSig` — Received VHT-SIG-B

matrix

Received VHT-SIG-B field, specified as an N_S-by-N_R matrix. N_S is the number of samples and increases with channel bandwidth.

Channel Bandwidth	N_S
`'CBW20'`	80
`'CBW40'`	160
`'CBW80'`	320
`'CBW160'`	640

N_R is the number of receive antennas.

Data Types: `double`

Complex Number Support: Yes

`chEst` — Channel estimate

3-D array

Channel estimate, specified as an N_{ST}-by-N_{STS}-by-N_{R} array. N_{ST} is the number of occupied subcarriers. N_{STS} is the number of space-time streams. N_{R} is the number of receive antennas.

N_{ST} increases with channel bandwidth.

`ChannelBandwidth`	Number of Occupied Subcarriers (N_{ST})	Number of Data Subcarriers (N_{SD})	Number of Pilot Subcarriers (N_{SP})
`'CBW20'`	56	52	4
`'CBW40'`	114	108	6
`'CBW80'`	242	234	8
`'CBW160'`	484	468	16

The channel estimate is based on the “VHT-LTF” on page 1-280.

`noiseVarEst` — Noise variance estimate

nonnegative scalar

Noise variance estimate, specified as a nonnegative scalar.

Data Types: `double`

`cbw` — Channel bandwidth

`'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth, specified as one of these strings: `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`.

Data Types: `char`

`cfgRec` — Algorithm parameters

`wlanRecoveryConfig` object

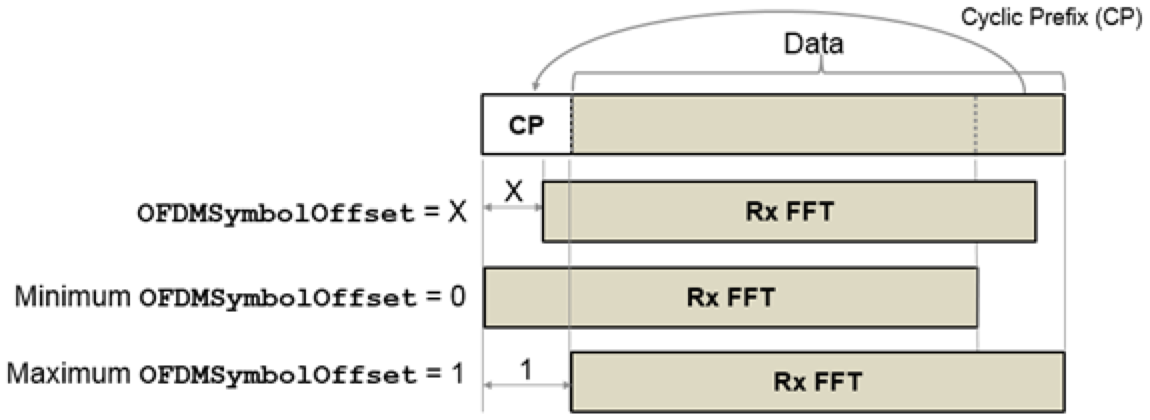
Algorithm parameters, specified as a `wlanRecoveryConfig` object. The function uses these properties:

Note: If `cfgRec` is not provided, the function uses the default values of the `wlanRecoveryConfig` object.

`OFDMSymbolOffset` — OFDM symbol sampling offset

0.75 (default) | scalar value from 0 to 1

OFDM symbol sampling offset represented as a fraction of the cyclic prefix (CP) length, specified as a scalar value from 0 to 1. This value indicates the start location for OFDM demodulation, relative to the beginning of the cyclic prefix. `OFDMSymbolOffset` = 0 represents the start of the cyclic prefix and `OFDMSymbolOffset` = 1 represents the end of the cyclic prefix.



Data Types: `double`

`EqualizationMethod` — Equalization method

`'MMSE'` (default) | `'ZF'`

Equalization method, specified as `'MMSE'` or `'ZF'`.

- `'MMSE'` indicates that the receiver uses a minimum mean square error equalizer.
- `'ZF'` indicates that the receiver uses a zero-forcing equalizer.

Example: `'ZF'`

Data Types: `char`

`PilotPhaseTracking` — Pilot phase tracking

`'PreEQ'` (default) | `'None'`

Pilot phase tracking, specified as either of these strings: `'PreEQ'` or `'None'`. If `'PreEQ'` is specified, pilot phase tracking is enabled and is performed prior to any equalization operation. If `'None'` is specified, pilot phase tracking does not occur.

Data Types: `char`

Output Arguments

`recBits` — Recovered VHT-SIG information

vector

Recovered VHT-SIG-B information bits, returned as an N_b-by-1 column vector. N_b is the number of recovered VHT-SIG-B information bits and increases with the channel bandwidth.

The number of output bits is proportional to the channel bandwidth.

ChannelBandwidth	N_b
'CBW20'	26
'CBW40'	27
'CBW80'	29
'CBW160'	29

See “VHT-SIG-B” on page 1-278 for information about the meaning of each bit in the field.

Data Types: `int8`

`eqSym` — Equalized symbols

matrix

Equalized symbols, returned as an N_{SD}-by-1 column vector. N_{SD} is the number of data subcarriers.

N_{SD} increases with the channel bandwidth.

`ChannelBandwidth`	N_{SD}
`'CBW20'`	52
`'CBW40'`	108
`'CBW80'`	234
`'CBW160'`	468

Data Types: `double`
Complex Number Support: Yes

`cpe` — Common phase error
column vector

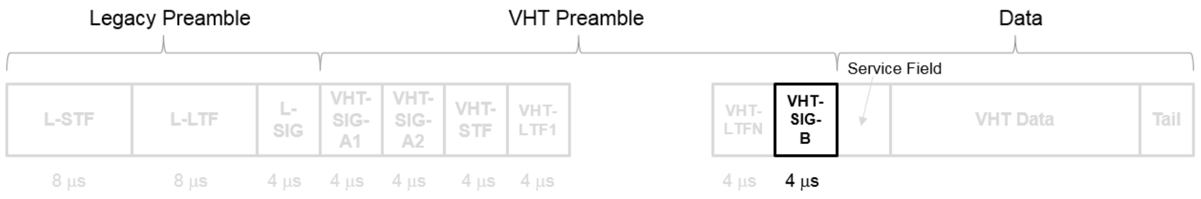
Common phase error in radians, returned as a scalar.

More About

VHT-SIG-B

The very high throughput signal B field (VHT-SIG-B) is used for multi-user scenario to set up the data rate and to fine-tune MIMO reception. It is modulated using MCS 0 and is transmitted in a single OFDM symbol.

The VHT-SIG-B field consists of a single OFDM symbol located between the VHT-LTF and the data portion of the VHT format PPDU.



The very high throughput signal B (VHT-SIG-B) field contains the actual rate and A-MPDU length value per user. The VHT-SIG-B is defined in IEEE Std 802.11ac-2013, Section 22.3.8.3.6, and Table 22–14. The number of bits in the VHT-SIG-B field varies with the channel bandwidth and the assignment depends on whether single user or multi-user scenario in allocated. For single user configurations, the same information is available in the L-SIG field but the VHT-SIG-B field is included for continuity purposes.

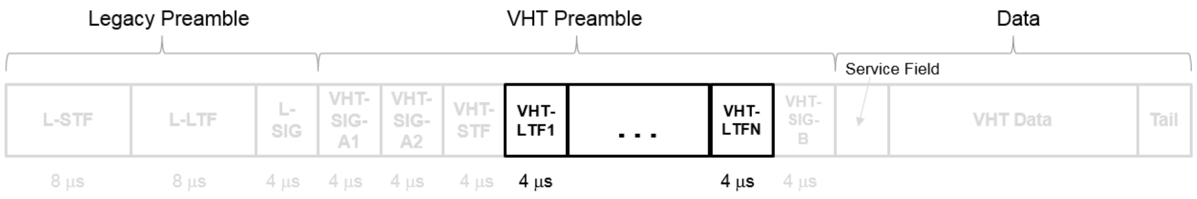
Field	VHT MU PPDU Allocation (bits)			VHT SU PPDU Allocation (bits)			Description
	20 MHz	40 MHz	80 MHz, 160 MHz	20 MHz	40 MHz	80 MHz, 160 MHz	
VHT-SIG-B	B0-15 (16)	B0-16 (17)	B0-18 (19)	B0-16 (17)	B0-18 (19)	B0-20 (21)	A variable-length field that indicates the size of the data payload in four-byte units. The length of the field depends on the channel bandwidth.
VHT-MCS	B16-19 (4)	B17-20 (4)	B19-22 (4)	N/A	N/A	N/A	A four-bit field that is included for multi-user scenarios only.
Reserved	N/A	N/A	N/A	B17–19 (3)	B19-20 (2)	B21-22 (2)	All ones
Tail	B20-25 (6)	B21-26 (6)	B23-28 (6)	B20-25 (6)	B21-26 (6)	B23-28 (6)	Six zero-bits used to terminate the

Field	VHT MU PPDU Allocation (bits)			VHT SU PPDU Allocation (bits)			Description
	20 MHz	40 MHz	80 MHz, 160 MHz	20 MHz	40 MHz	80 MHz, 160 MHz	
							convolutional code.
Total # bits	26	27	29	26	27	29	
Bit field repetitio	1	2	4 *For 160 MHz, the 80 MHz channel is repeated twice.*	1	2	4 *For 160 MHz, the 80 MHz channel is repeated twice.*	

For a null data packet (NDP), the VHT-SIG-B bits are set according to IEEE Std 802.11ac-2013, Table 22-15.

VHT-LTF

The very high throughput long training field (VHT-LTF) is located between the VHT-STF and VHT-SIG-B portion of the VHT packet.



It is used for MIMO channel estimation and pilot subcarrier tracking. The VHT-LTF includes one VHT long training symbol for each spatial stream indicated by the selected MCS. Each symbol is 4 μs long. A maximum of eight symbols are permitted in the VHT-LTF.

The VHT-LTF is defined in IEEE Std 802.11ac-2013, Section 22.3.8.3.5.

PPDU

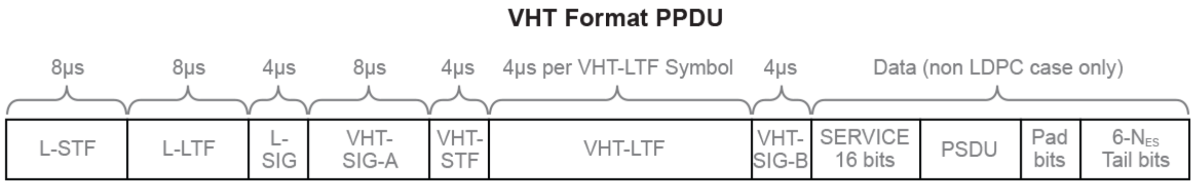
PLCP protocol data unit

The PPDU is the complete PLCP frame, including PLCP headers, MAC headers, the MAC data field, and the MAC and PLCP trailers.

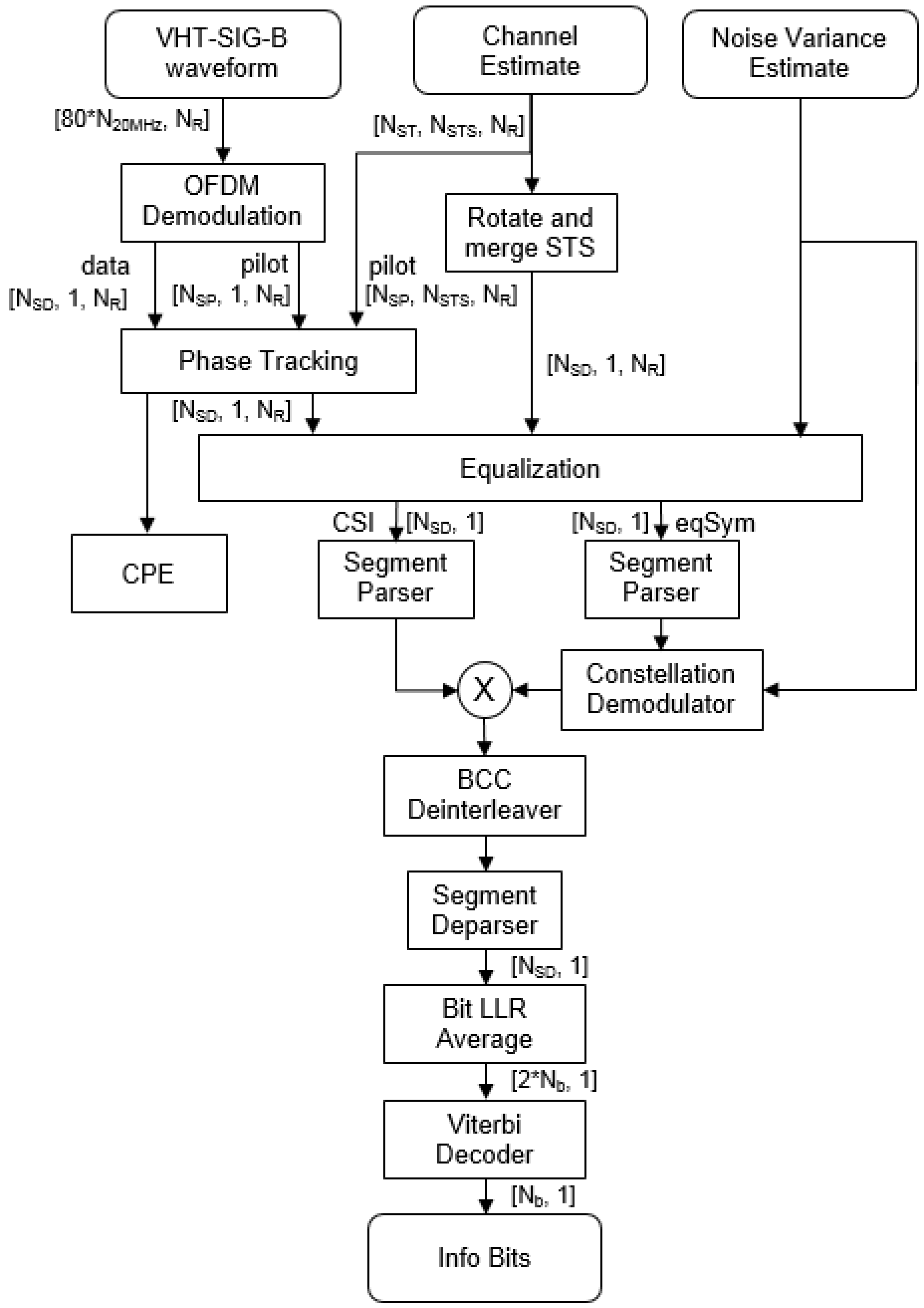
Algorithms

VHT-SIG-B Recovery

The “VHT-SIG-B” on page 1-278 field consists of one symbol and resides between the VHT-LTF field and the data portion of the packet structure for the VHT format PPDUs.



For single-user packets, you can recover the length information from the L-SIG and VHT-SIG-A field information. Therefore, it is not strictly required for the receiver to decode the “VHT-SIG-B” on page 1-278 field.


VHT-SIG-B waveform
Channel Estimate
Noise Variance Estimate
[80*N20MHz, NR]
[NST, NSTS, NR]
OFDM Demodulation
Rotate and merge STS
data
[NSD, 1, NR]
pilot
[NSP, 1, NR]
pilot
[NSP, NSTS, NR]
Phase Tracking
[NSD, 1, NR]
[NSD, 1, NR]
Equalization
CPE
CSI
[NSD, 1]
[NSD, 1]
eqSym
Segment Parser
Segment Parser
X
Constellation Demodulator
BCC Deinterleaver
Segment Deparser
[NSD, 1]
Bit LLR Average
[2*Nb, 1]
Viterbi Decoder
[Nb, 1]
Info Bits

For “VHT-SIG-B” on page 1-278 details, refer to IEEE Std 802.11ac™-2013 [1], Section 22.3.4.8, and Perahia [2], Section 7.3.2.4.

References

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

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

See Also

`wlanRecoveryConfig` | `wlanVHTConfig` | `wlanVHTLTFChannelEstimate` | `wlanVHTLTFDemodulate` | `wlanVHTSIGB`

Introduced in R2015b

wlanVHTSTF

Generate VHT-STF waveform

Syntax

```
y = wlanVHTSTF(cfg)
```

Description

`y = wlanVHTSTF(cfg)` generates a "VHT-STF" on page 1-288[28] time-domain waveform for the specified configuration object. See "VHT-STF Processing" on page 1-289 for waveform generation details.

Examples

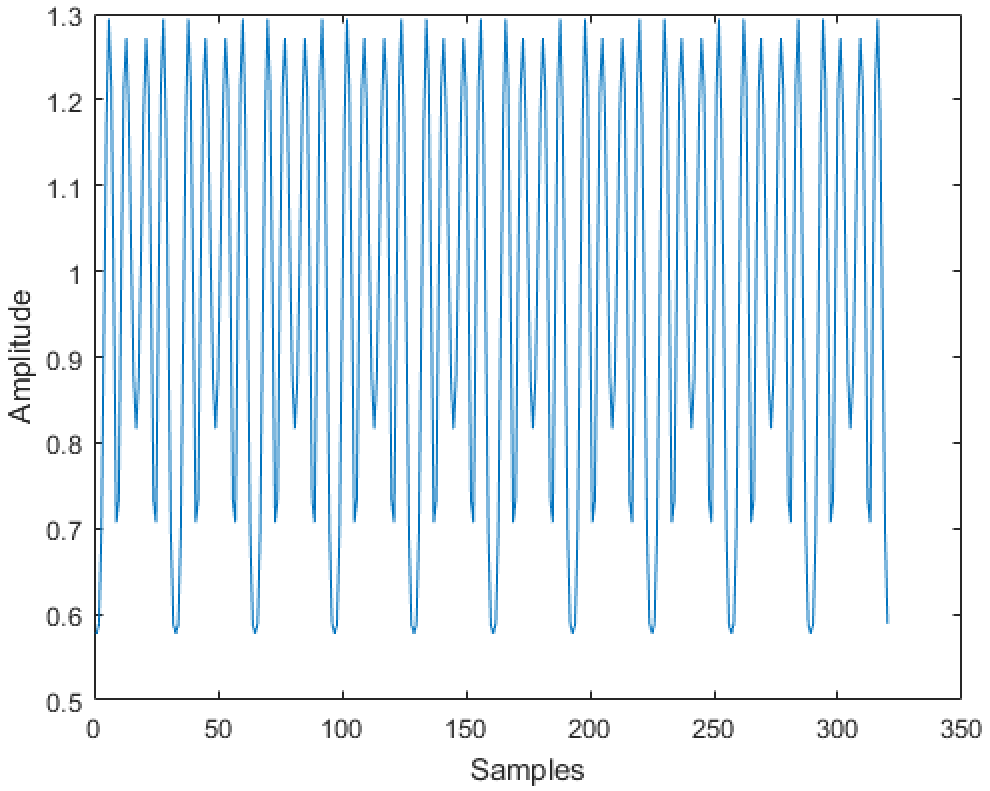
Generate VHT-STF Waveform

Create a VHT configuration object with an 80 MHz channel bandwidth. Generate and plot the VHT-STF waveform.

```
cfgVHT = wlanVHTConfig;
cfgVHT.ChannelBandwidth = 'CBW80';

vstfOut = wlanVHTSTF(cfgVHT);
size(vstfOut);
plot(abs(vstfOut));
xlabel('Samples');
ylabel('Amplitude');
```

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



The 80 MHz waveform is a single OFDM symbol with 320 complex time-domain output samples. The waveform contains the repeating short training field pattern.

Input Arguments

`cfg` — Format configuration

`wlanVHTConfig` object

Format configuration, specified as a `wlanVHTConfig` object. The `wlanVHTSTF` function uses the object properties indicated.

`ChannelBandwidth` — Channel bandwidth
`'CBW80'` (default) | `'CBW20'` | `'CBW40'` | `'CBW160'`

Channel bandwidth, specified as `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`. If the transmission has multiple users, the same channel bandwidth is applied to all users. The default value of `'CBW80'` sets the channel bandwidth to 80 MHz.

Data Types: `char`

`NumTransmitAntennas` — Number of transmit antennas
1 (default) | integer from 1 to 8

Number of transmit antennas, specified as a scalar integer from 1 to 8.

Data Types: `double`

`NumSpaceTimeStreams` — Number of space-time streams
1 (default) | integer from 1 to 8 | 1-by-N_{Users} vector of integers from 1 to 4

Number of space-time streams in the transmission, specified as a scalar or vector.

- For a single user, the number of space-time streams is a scalar integer from 1 to 8.
- For multiple users, the number of space-time streams is a 1-by-N_{Users} vector of integers from 1 to 4, where the vector length, N_{Users}, is an integer from 1 to 4.

Example: `[1 3 2]` is the number of space-time streams for each user.

Note: The sum of the space-time stream vector elements must not exceed eight.

Data Types: `double`

`SpatialMapping` — Spatial mapping scheme
`'Direct'` (default) | `'Hadamard'` | `'Fourier'` | `'Custom'`

Spatial mapping scheme, specified as `'Direct'`, `'Hadamard'`, `'Fourier'`, or `'Custom'`. The default value of `'Direct'` applies when `NumTransmitAntennas` and `NumSpaceTimeStreams` are equal.

Data Types: `char`

`SpatialMappingMatrix` — Spatial mapping matrix
1 (default) | scalar | matrix | 3-D array

Spatial mapping matrix, specified as a scalar, matrix, or 3-D array. Use this property to apply a beamforming steering matrix, and to rotate and scale the constellation mapper output vector. If applicable, scale the space-time block coder output instead. `SpatialMappingMatrix` applies when the `SpatialMapping` property is set to `'Custom'`. For more information, see IEEE Std 802.11-2012, Section 20.3.11.11.2.

- When specified as a scalar, a constant value applies to all the subcarriers.
- When specified as a matrix, the size must be N_{STS_Total}-by-N_T. The spatial mapping matrix applies to all the subcarriers. N_{STS_Total} is the sum of space-time streams for all users, and N_T is the number of transmit antennas.
- When specified as a 3-D array, the size must be N_{ST}-by-N_{STS_Total}-by-N_T. N_{ST} is the sum of the occupied data (N_{SD}) and pilot (N_{SP}) subcarriers, as determined by `ChannelBandwidth`. N_{STS_Total} is the sum of space-time streams for all users. N_T is the number of transmit antennas.

  N_{ST} increases with channel bandwidth.

`ChannelBandwidth`	Number of Occupied Subcarriers (N_{ST})	Number of Data Subcarriers (N_{SD})	Number of Pilot Subcarriers (N_{SP})
'CBW20'	56	52	4
'CBW40'	114	108	6
'CBW80'	242	234	8
'CBW160'	484	468	16

The calling function normalizes the spatial mapping matrix for each subcarrier.

Example: [0.5 0.3 0.4; 0.4 0.5 0.8] represents a spatial mapping matrix having two space-time streams and three transmit antennas.

Data Types: `double`
Complex Number Support: Yes

Output Arguments

y — VHT-STF time-domain waveform

matrix

"VHT-STF" on page 1-288 time-domain waveform, returned as an N_S-by-N_T matrix. N_S is the number of time-domain samples, and N_T is the number of transmit antennas.

N_S is proportional to the channel bandwidth. Each symbol contains 80 time samples per 20 MHz channel.

`ChannelBandwidth`	N_S
`'CBW20'`	80
`'CBW40'`	160
`'CBW80'`	320
`'CBW160'`	640

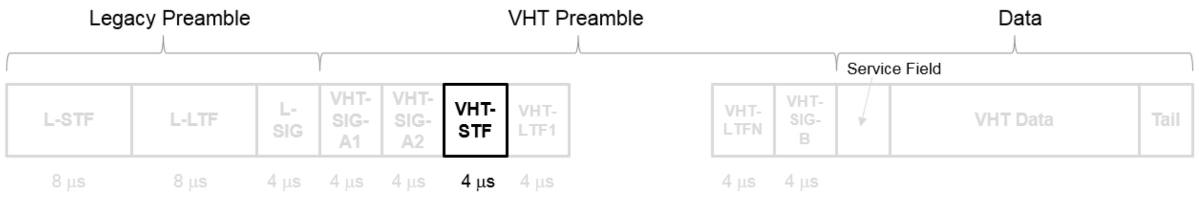
See "VHT-STF Processing" on page 1-289 for waveform generation details.

Data Types: `double`
Complex Number Support: Yes

More About

VHT-STF

The very high throughput short training field (VHT-STF) is a single OFDM symbol (4 μs in length) that is used to improve automatic gain control estimation in a MIMO transmission. It is located between the VHT-SIG-A and VHT-LTF portions of the VHT packet.



The frequency domain sequence used to construct the VHT-STF for a 20 MHz transmission is identical to the L-STF sequence. Duplicate L-STF sequences are frequency shifted and phase rotated to support VHT transmissions for the 40 MHz, 80

MHz, and 160 MHz channel bandwidths. As such, the L-STF and HT-STF are subsets of the VHT-STF.

The VHT-STF is defined in IEEE Std 802.11ac-2013, Section 22.3.8.3.4.

Algorithms

VHT-STF Processing

The "VHT-STF" on page 1-288 is one OFDM symbol long and is processed for improved gain control in MIMO configurations. For algorithm details, refer to IEEE Std 802.11ac-2013 [1], Section 22.3.4.6.

References

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

See Also

`wlanLSTF` | `wlanVHTConfig` | `wlanVHTLTF` | `wlanVHTSIGA`

Introduced in R2015b

wlanWaveformGenerator

Generate WLAN waveform

Syntax

```
waveform = wlanWaveformGenerator(bits,cfgFormat,cfgWaveGen)
```

Description

`waveform = wlanWaveformGenerator(bits,cfgFormat,cfgWaveGen)` generates a waveform for the specified information bits, format configuration, and waveform generator configuration.

Examples

Generate VHT Waveform

Generate a time-domain signal for an 802.11ac VHT transmission with ten packets and a 20 microsecond idle period between packets.

Create the format configuration object, `vht`. Assign two transmit antennas and two spatial streams, and disable STBC. Set the MCS to `1`, which assigns QPSK modulation and a 1/2 rate coding scheme per the 802.11 standard. Set the number of bytes in the A-MPDU pre-EOF padding, `APEPLength`, to `1024`.

```
vht = wlanVHTConfig;
vht.NumTransmitAntennas = 2;
vht.NumSpaceTimeStreams = 2;
vht.STBC = false;
vht.MCS = 1;
vht.APEPLength = 1024;
```

Create the generator configuration object and generate ten packets with 20 microsecond idle period.

```
wgc = wlanGeneratorConfig;
```

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

Generate the transmission waveform.

```
txWaveform = wlanWaveformGenerator([1;0;0;1],vht,wgc);
```

Generate VHT Waveform with Random Scrambler State

Configure `wlanWaveformGenerator` to produce a time-domain signal for an 802.11ac VHT transmission with five packets and a 30 microsecond idle period between packet. Use a random scrambler initial state for each packet.

Create a generator configuration object. Update the default settings of the object to generate five packets with a 30 microsecond idle period between each packet.

```
wgc = wlanGeneratorConfig;
wgc.NumPackets = 5;
wgc.IdleTime = 30e-6;
```

Use a random scrambler initial state for each packet.

```
wgc.ScramblerInitialization = randi([1 127],wgc.NumPackets,1);
```

Create a VHT configuration object and confirm the channel bandwidth for scaling the *x*-axis of the plot.

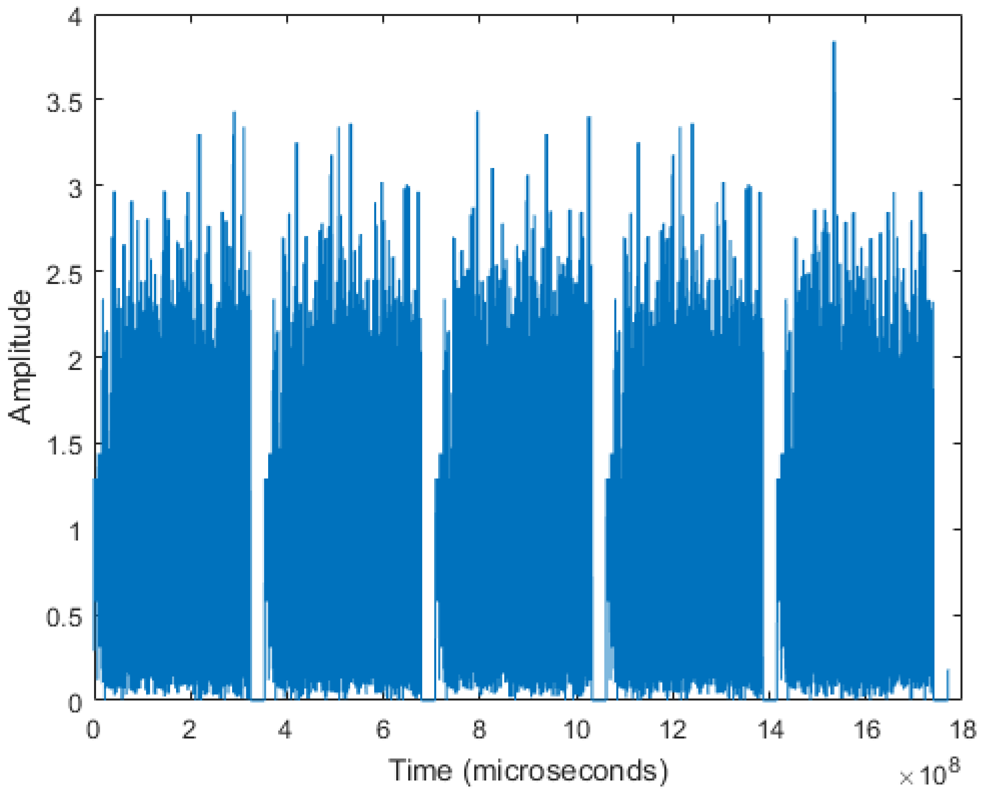
```
vht = wlanVHTConfig;
vht.ChannelBandwidth
```

```
ans =

CBW80
```

Generate and plot the waveform. Display the time in microseconds on the *x*-axis.

```
txWaveform = wlanWaveformGenerator([1;0;0;1],vht,wgc);
time = [0:length(txWaveform)-1]/80e-6;
plot(time,abs(txWaveform))
xlabel ('Time (microseconds)');
ylabel('Amplitude');
```



Five packets separated by 30 microsecond idle periods.

Input Arguments

`bits` — Information bits

0 | 1 | vector | cell array | vector cell array

Information bits for a single user, including any MAC padding representing multiple concatenated PSDUs, specified as a binary vector stream. Internally, the input `bits` vector is looped as required to generate the specified number of packets. The property `cfgFormat.PSDULength` specifies the number of data bits taken from the bit stream for each transmission packet generated. The property `cfgWaveGen.NumPackets` specifies the number of packets to generate.

- When `bits` is a cell array, each element of the cell array must be a `double` or `int8` typed binary vector.
- When `bits` is a vector or scalar cell array, the specified bits apply to all users.
- When `bits` is a vector cell array, each element applies to each user correspondingly. For each user, if the number of bits required across all packets of the generation exceeds the length of the vector provided, the applied bit vector is looped. Looping on the bits allows you to define a short pattern, for example. `[1;0;0;1]`, that is repeated as the input to the PSDU coding across packets and users. In each packet generation, for the *i*th user, the *i*th element of the `cfgFormat.PSDULength` indicates the number of data bytes taken from its stream. Multiple `PSDULength` by eight to compute the number of bits

Example: `[1 1 0 1 0 1 1]`

Data Types: `double` | `int8`

`cfgFormat` — Packet format configuration
`wlanVHTConfig` object | `wlanHTConfig` object | `wlanNonHTConfig` object

Packet format configuration, specified as a `wlanVHTConfig`, `wlanHTConfig`, or `wlanNonHTConfig` object. The type of `cfgFormat` object determines the IEEE 802.11 format of the generated waveform. For a description of the properties and valid settings for the various packet format configuration objects, see:

- wlanVHTConfig Properties
- wlanHTConfig Properties
- wlanNonHTConfig Properties

The data rate and PSDU length of generated PPDUs is determined based on the properties of the packet format configuration object.

`cfgWaveGen` — Waveform generator configuration
`wlanGeneratorConfig` object

Waveform generator configuration, specified as a `wlanGeneratorConfig` object. The `wlanWaveformGenerator` function uses these `wlanGeneratorConfig` object properties:

`NumPackets` — Number of packets

1 (default) | positive integer

Number of packets to generate in a single function call, specified as a positive integer.

Data Types: `double`

`IdleTime` — Idle time added after each packet

0 (default) | nonnegative scalar

Idle time added after each packet, specified as a nonnegative scalar in seconds. If `IdleTime` is greater than the default value of zero, it cannot be less than 2 μs.

Example: `20e-6`

Data Types: `double`

`ScramblerInitialization` — Initial scrambler state

93 (default) | integer from 1 to 127 | matrix

Initial scrambler state of the data scrambler for each packet generated, specified as an integer from 1 to 127, or as an N_P-*by*-N_{Users} matrix of integers with values from 1 to 127. N_P is the number of packets, and N_{Users} is the number of users. The default value of 93 is the example state given in IEEE Std 802.11-2012, Section L.1.5.2.

- When specified as a scalar, the same scrambler initialization value is used to generate each packet for each user of a multipacket waveform.
- When specified as a matrix, each element represents an initial state of the scrambler for packets in the multipacket waveform generated for each user. Each column specifies the initial states for a single user, therefore up to four columns are supported. If a single column is provided, the same initial states are used for all users. Each row represents the initial state of each packet to generate. Therefore, a matrix with multiple rows enables you to use a different initial state per packet, where the first row contains the initial state of the first packet. If the number of packets to generate exceeds the number of rows of the matrix provided, the rows are looped internally.

The waveform generator configuration object does not validate the initial state of the scrambler.

Note: `ScramblerInitialization` applies to OFDM-based formats only.

Example: `[3 56 120]`

Data Types: `double` | `int8`

`Windowing` — Option to disable windowing

`true` (default) | `false`

Option to disable windowing between consecutive OFDM symbols, specified as a logical. Windowing is the process in which the OFDM symbol is multiplied by a cosine function before transmission to reduce adjacent channel power.

Data Types: `logical`

`WindowTransitionTime` — Duration of the window transition

1.0e-07 (default) | positive scalar less than or equal to 1.6e-06

Duration of the window transition applied to each OFDM symbol, specified in seconds as a positive scalar less than or equal to 1.6e-06. For a transition time of zero, no windowing is applied.

The maximum permitted `WindowTransitionTime` value depends on the type of guard interval:

- For a long guard interval, the maximum permitted setting is 1.6e-06 seconds.
- For a short guard interval, the maximum permitted setting is 8.0e-07 seconds.

Data Types: `double`

Output Arguments

`waveform` — Packetized waveform

matrix

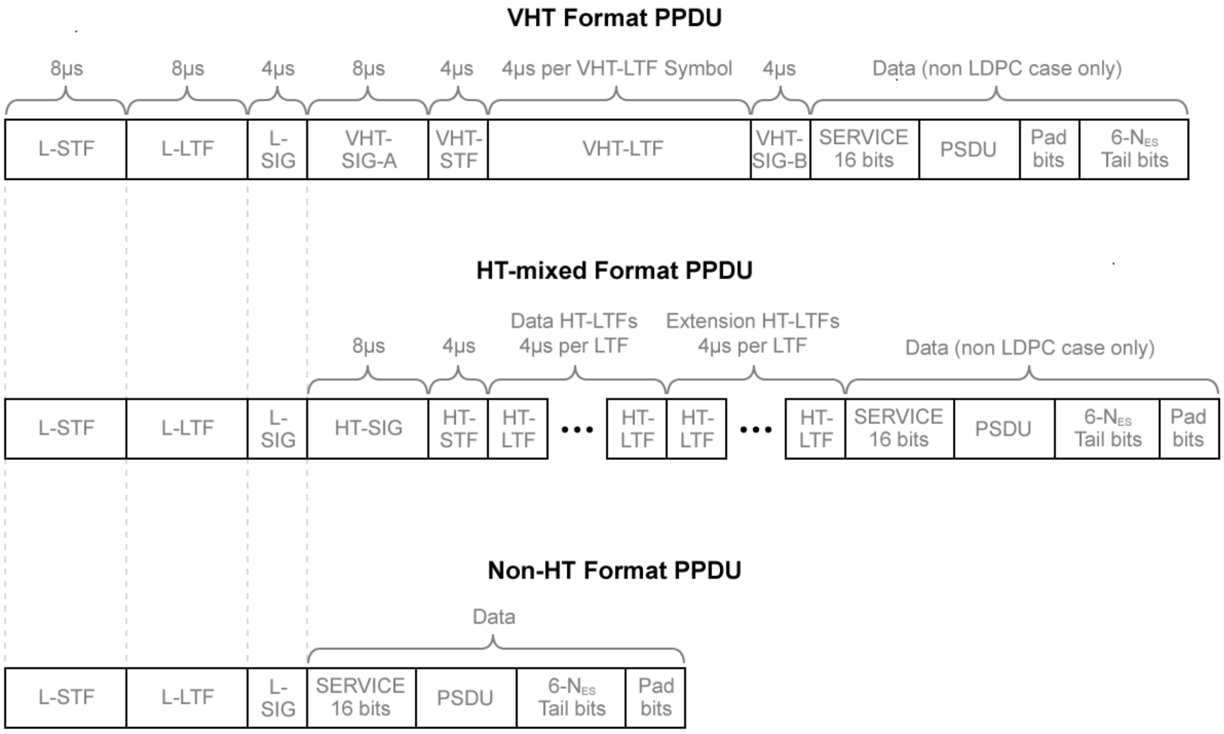
Packetized waveform, returned as an N_S-by-N_T matrix. N_S is the number of time-domain samples, and N_T is the number of transmit antennas. `waveform` contains one or more packets of the same “IEEE 802.11 PPDU format” on page 1-296. Each packet can contain different information bits. To window the OFDM symbols in each packet, enable `cfgWaveGen.Windowing`. By default, `Windowing` is enabled.

Data Types: `double`
Complex Number Support: Yes

More About

IEEE 802.11 PPDU format

IEEE 802.11[29][30] PPDU formats defined for transmission in include VHT, HT, and non-HT. The PPDU field structure consists of preamble and data portions.



See Also

`wlanGeneratorConfig` | `wlanHTConfig` | `wlanNonHTConfig` | `wlanVHTConfig`

Introduced in R2015b

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

wlanWindowing

not found

Syntax

Description

Use the Help browser search field to `matlab:docsearch wlanWindowing`, or type "`matlab:help help`" for help command options, such as help for methods.

Classes — Alphabetical List

wlanGeneratorConfig Properties

Define parameter values for waveform generation

The `wlanGeneratorConfig` object specifies the non-format-specific properties necessary for generating IEEE 802.11, [1] standards-compliant waveforms.

After you create an object, use dot notation to change or access the object parameters. For example:

Create a `wlanGeneratorConfig` object. Then modify the default setting for the `NumPackets` property.

```
cfgGen = wlanGeneratorConfig;
cfgGen.NumPackets = 5;
```

Waveform Generation Configuration

`NumPackets` — Number of packets

1 (default) | positive integer

Number of packets to generate in a single function call, specified as a positive integer.

Data Types: `double`

`IdleTime` — Idle time added after each packet

0 (default) | nonnegative scalar

Idle time added after each packet, specified as a nonnegative scalar in seconds. If `IdleTime` is greater than the default value of zero, it cannot be less than 2 µs.

Example: `20e-6`

Data Types: `double`

`ScramblerInitialization` — Initial scrambler state

93 (default) | integer from 1 to 127 | matrix

Initial scrambler state of the data scrambler for each packet generated, specified as an integer from 1 to 127, or as an N_P-*by*-N_{Users} matrix of integers with values from 1 to 127. N_P is the number of packets, and N_{Users} is the number of users. The default value of 93 is the example state given in IEEE Std 802.11-2012, Section L.1.5.2.

- When specified as a scalar, the same scrambler initialization value is used to generate each packet for each user of a multipacket waveform.
- When specified as a matrix, each element represents an initial state of the scrambler for packets in the multipacket waveform generated for each user. Each column specifies the initial states for a single user, therefore up to four columns are supported. If a single column is provided, the same initial states are used for all users. Each row represents the initial state of each packet to generate. Therefore, a matrix with multiple rows enables you to use a different initial state per packet, where the first row contains the initial state of the first packet. If the number of packets to generate exceeds the number of rows of the matrix provided, the rows are looped internally.

The waveform generator configuration object does not validate the initial state of the scrambler.

Note: `ScramblerInitialization` applies to OFDM-based formats only.

Example: `[3 56 120]`

Data Types: `double` | `int8`

`Windowing` — Option to disable windowing
`true` (default) | `false`

Option to disable windowing between consecutive OFDM symbols, specified as a logical. Windowing is the process in which the OFDM symbol is multiplied by a cosine function before transmission to reduce adjacent channel power.

Data Types: `logical`

`WindowTransitionTime` — Duration of the window transition
1.0e-07 (default) | positive scalar less than or equal to 1.6e-06

Duration of the window transition applied to each OFDM symbol, specified in seconds as a positive scalar less than or equal to 1.6e-06. For a transition time of zero, no windowing is applied.

The maximum permitted `WindowTransitionTime` value depends on the type of guard interval:

- For a long guard interval, the maximum permitted setting is 1.6e-06 seconds.

- For a short guard interval, the maximum permitted setting is 8.0e-07 seconds.

Data Types: `double`

See Also

`wlanGeneratorConfig` | `wlanHTConfig` | `wlanNonHTConfig` | `wlanVHTConfig` | `wlanWaveformGenerator`

Introduced in R2015b

wlanVHTConfig Properties

Define parameter values for VHT format packet

The `wlanVHTConfig` object specifies the transmission properties for the IEEE 802.11 very high throughput (VHT) format physical layer (PHY) packet.

After you create an object, use dot notation to change or access the object parameters. For example:

Create a `wlanVHTConfig` object. Then modify the default setting for the `ChannelBandwidth` property.

```
cfgVHT = wlanVHTConfig;
cfgVHT.ChannelBandwidth = 'CBW20';
```

VHT Format Configuration

`ChannelBandwidth` — Channel bandwidth

`'CBW80'` (default) | `'CBW20'` | `'CBW40'` | `'CBW160'`

Channel bandwidth, specified as `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`. If the transmission has multiple users, the same channel bandwidth is applied to all users. The default value of `'CBW80'` sets the channel bandwidth to 80 MHz.

Data Types: `char`

`NumUsers` — Number of users

1 (default) | 2 | 3 | 4

Number of users, specified as 1, 2, 3, or 4.

Data Types: `double`

`UserPositions` — Position of users

[0 1] (default) | row vector of integers for 0 to 3 in strictly increasing order

Position of users, specified as an integer row vector with length equal to `NumUsers` and element values from 0 to 3 in a strictly increasing order. This property applies when `NumUsers` property is set to 2, 3 or 4. The default value of this property is [0 1].

Example: `[0 1 2]`indicates that users occupy the specified positions.

Data Types: `double`

`NumTransmitAntennas` — Number of transmit antennas
1 (default) | integer from 1 to 8

Number of transmit antennas, specified as a scalar integer from 1 to 8.

Data Types: `double`

`NumSpaceTimeStreams` — Number of space-time streams
1 (default) | integer from 1 to 8 | 1-by-N_{Users} vector of integers from 1 to 4

Number of space-time streams in the transmission, specified as a scalar or vector.

- For a single user, the number of space-time streams is a scalar integer from 1 to 8.
- For multiple users, the number of space-time streams is a 1-by-N_{Users} vector of integers from 1 to 4, where the vector length, N_{Users}, is an integer from 1 to 4.

Example: `[1 3 2]` is the number of space-time streams for each user.

Note: The sum of the space-time stream vector elements must not exceed eight.

Data Types: `double`

`SpatialMapping` — Spatial mapping scheme
`'Direct'` (default) | `'Hadamard'` | `'Fourier'` | `'Custom'`

Spatial mapping scheme, specified as `'Direct'`, `'Hadamard'`, `'Fourier'`, or `'Custom'`. The default value of `'Direct'` applies when `NumTransmitAntennas` and `NumSpaceTimeStreams` are equal.

Data Types: `char`

`SpatialMappingMatrix` — Spatial mapping matrix
1 (default) | scalar | matrix | 3-D array

Spatial mapping matrix, specified as a scalar, matrix, or 3-D array. Use this property to apply a beamforming steering matrix, and to rotate and scale the constellation mapper output vector. If applicable, scale the space-time block coder output instead. `SpatialMappingMatrix` applies when the `SpatialMapping` property is set to `'Custom'`. For more information, see IEEE Std 802.11-2012, Section 20.3.11.11.2.

- When specified as a scalar, a constant value applies to all the subcarriers.

- When specified as a matrix, the size must be N_{STS_Total}-by-N_T. The spatial mapping matrix applies to all the subcarriers. N_{STS_Total} is the sum of space-time streams for all users, and N_T is the number of transmit antennas.
- When specified as a 3-D array, the size must be N_{ST}-by-N_{STS_Total}-by-N_T. N_{ST} is the sum of the occupied data (N_{SD}) and pilot (N_{SP}) subcarriers, as determined by `ChannelBandwidth`. N_{STS_Total} is the sum of space-time streams for all users. N_T is the number of transmit antennas.

  N_{ST} increases with channel bandwidth.

`ChannelBandwidth`	Number of Occupied Subcarriers (N_{ST})	Number of Data Subcarriers (N_{SD})	Number of Pilot Subcarriers (N_{SP})
'CBW20'	56	52	4
'CBW40'	114	108	6
'CBW80'	242	234	8
'CBW160'	484	468	16

The calling function normalizes the spatial mapping matrix for each subcarrier.

Example: [0.5 0.3 0.4; 0.4 0.5 0.8] represents a spatial mapping matrix having two space-time streams and three transmit antennas.

Data Types: `double`
Complex Number Support: Yes

`Beamforming` — Option to disable the signaling of a transmission with beamforming

true (default) | false

Option to disable the signaling of a transmission with beamforming, specified as a logical. Beamforming is performed when setting is `true`. This property applies when `NumUsers` equals 1 and `SpatialMapping` is set to `'Custom'`. The `SpatialMappingMatrix` property specifies the beamforming steering matrix.

Data Types: `logical`

`STBC` — Option to enable space-time block coding

false (default) | true

Option to enable space-time block coding (STBC) of the PPDU data field, specified as a logical. STBC transmits multiple copies of the data stream across assigned antennas.

- When set to `false`, no STBC is applied to the data field, and the number of space-time streams is equal to the number of spatial streams.
- When set to `true`, STBC is applied to the data field, and the number of space-time streams is double the number of spatial streams.

See IEEE 802.11ac-2013, Section 22.3.10.9.4 for further description.

Note: `STBC` is relevant for single-user transmissions only.

Data Types: `logical`

`MCS` — Modulation and coding scheme
0 (default) | integer from 0 to 9 | 1-by-N_{Users} vector of integers

Modulation and coding scheme used in transmitting the current packet, specified as a scalar or vector.

- For a single user, the MCS value is a scalar integer from 0 to 9.
- For multiple users, MCS is a 1-by-N_{Users} vector of integers or a scalar with values from 0 to 9, where the vector length, N_{Users}, is an integer from 1 to 4.

MCS	Modulation	Coding Rate
0	`BPSK`	`1/2`
1	`QPSK`	`1/2`
2	`QPSK`	`3/4`
3	`16QAM`	`1/2`
4	`16QAM`	`3/4`
5	`64QAM`	`2/3`
6	`64QAM`	`3/4`
7	`64QAM`	`5/6`
8	`256QAM`	`3/4`
9	`256QAM`	`5/6`

Data Types: `double`

`ChannelCoding` — Type of forward error correction coding

`'BCC'` (default)

This property is read only.

Type of forward error correction coding for the data field, specified as `'BCC'` to indicate binary convolutional coding. LDPC coding is not currently supported.

Data Types: `char`

`APEPLength` — Number of bytes in the A-MPDU pre-EOF padding

1024 (default) | integer from 0 to 1,048,575 | vector of integers

Number of bytes in the A-MPDU pre-EOF padding, specified as a scalar integer or vector of integers.

- For a single user, `APEPLength` is a scalar integer from 0 to 1,048,575.
- For multi-user, `APEPLength` is a 1-by-N_{Users} vector of integers or a scalar with values from 0 to 1,048,575, where the vector length, N_{Users}, is an integer from 1 to 4.
- `APEPLength = 0` for a null data packet (NDP).

`APEPLength` is used internally to determine the number of OFDM symbols in the data field. For more information, see IEEE Std 802.11ac-2013, Table 22-1.

Data Types: `double`

`PSDULength` — Number of bytes carried in the user payload

integer | vector of integers

This property is read only.

Number of bytes carried in the user payload, including the A-MPDU and any MAC padding. For a null data packet (NDP) the PSDU length is zero.

- For a single user, the PSDU length is a scalar integer from 1 to 1,048,575.
- For multiple users, the PSDU length is a 1-by-N_{Users} vector of integers from 1 to 1,048,575, where the vector length,N_{Users}, is an integer from 1 to 4.

`PSDULength` is a read-only property and is calculated internally based on the `APEPLength` property and other coding-related properties, as specified in IEEE Std 802.11ac-2013, Section 22.4.3. It is accessible by direct property call. When accessing `PSDULength`, the object is validated.

Example: `[1035 4150]` is the PSDU length vector for a `wlanVHTConfig` object with two users, where the MCS for the first user is 0 and the MCS for the second user is 3.

Data Types: `double`

`GuardInterval` — Cyclic prefix length for the data field within a packet

`'Long'` (default) | `'Short'`

Cyclic prefix length for the data field within a packet, specified as `'Long'` or `'Short'`.

- The long guard interval length is 800 ns.
- The short guard interval length is 400 ns.

Data Types: `char`

`GroupID` — Group identification number

63 (default) | integer from 0 to 63

Group identification number, specified as a scalar integer from 0 to 63.

- A group identification number of either 0 or 63 indicates a VHT single-user PPDU.
- A group identification number from 1 to 62 indicates a VHT multi-user PPDU.

Data Types: `double`

`PartialAID` — Abbreviated indication of the PSDU recipient

275 (default) | integer from 0 to 511

Abbreviated indication of the PSDU recipient, specified as a scalar integer from 0 to 511.

- For an uplink transmission, the partial identification number is the last nine bits of the basic service set identifier (BSSID).
- For a downlink transmission, the partial identification of a client is an identifier that combines the association ID with the BSSID of its serving AP.

For more information, see IEEE Std 802.11ac-2013, Table 22-1.

Data Types: `double`

See Also

`wlanHTConfig` | `wlanNonHTConfig` | `wlanVHTConfig` | `wlanWaveformGenerator`

Introduced in R2015b

wlanHTConfig Properties

Define parameter values for HT format packet

The `wlanHTConfig` object specifies the transmission properties for the IEEE 802.11 high throughput (HT) format physical layer (PHY) packet.

After you create an object, use dot notation to change or access the object parameters. For example:

Create a `wlanHTConfig` object. Then modify the default setting for the `NumTransmitAntennas` property.

```
cfgHT = wlanHTConfig;
cfgHT.numTransmitAntennas = 2;
```

HT Format Configuration

`ChannelBandwidth` — Channel bandwidth
`'CBW20'` (default) | `'CBW40'`

Channel bandwidth in MHz, specified as `'CBW20'` or `'CBW40'`.

Data Types: `char`

`NumTransmitAntennas` — Number of transmit antennas
1 (default) | 2 | 3 | 4

Number of transmit antennas, specified as 1, 2, 3, or 4.

Data Types: `double`

`NumSpaceTimeStreams` — Number of space-time streams
1 (default) | 2 | 3 | 4

Number of space-time streams in the transmission, specified as 1, 2, 3, or 4.

Data Types: `double`

`NumExtensionStreams` — Number of extension spatial streams
0 (default) | 1 | 2 | 3

Number of extension spatial streams in the transmission, specified as 0, 1, 2, or 3. When `NumExtensionStreams` is greater than 0, `SpatialMapping` must be `'Custom'`.

Data Types: `double`

`SpatialMapping` — Spatial mapping scheme

`'Direct'` (default) | `'Hadamard'` | `'Fourier'` | `'Custom'`

Spatial mapping scheme, specified as `'Direct'`, `'Hadamard'`, `'Fourier'`, or `'Custom'`. The default value `'Direct'`, applies when `NumTransmitAntennas` and `NumSpaceTimeStreams` are equal.

Data Types: `char`

`SpatialMappingMatrix` — Spatial mapping matrix

1 (default) | scalar | matrix | 3-D array

Spatial mapping matrix, specified as a scalar, matrix, or 3-D array. Use this property to rotate and scale the constellation mapper output vector. This property applies when the `SpatialMapping` property is set to `'Custom'`. The spatial mapping matrix is used for beamforming and mixing space-time streams over the transmit antennas.

- When specified as a scalar, `NumTransmitAntennas` = `NumSpaceTimeStreams` = 1 and a constant value applies to all the subcarriers.
- When specified as a matrix, the size must be (N_{STS} + N_{ESS})-by-N_T. N_{STS} is the number of space-time streams. N_{ESS} is the number of extension spatial streams. N_T is the number of transmit antennas. The spatial mapping matrix applies to all the subcarriers. The first N_{STS} and last N_{ESS} rows apply to the space-time streams and extension spatial streams respectively.
- When specified as a 3-D array, the size must be N_{ST}-by-(N_{STS} + N_{ESS})-by-N_T. N_{ST} is the sum of the data and pilot subcarriers, as determined by `ChannelBandwidth`. N_{STS} is the number of space-time streams. N_{ESS} is the number of extension spatial streams. N_T is the number of transmit antennas. In this case, each data and pilot subcarrier can have its own spatial mapping matrix.

  The table shows the `ChannelBandwidth` setting and the corresponding N_{ST}.

ChannelBandwidth	N_{ST}
`'CBW20'`	56
`'CBW40'`	114

The calling function normalizes the spatial mapping matrix for each subcarrier.

Example: `[0.5 0.3; 0.4 0.4; 0.5 0.8]` represents a spatial mapping matrix having three space-time streams and two transmit antennas.

Data Types: `double`
Complex Number Support: Yes

`MCS` — Modulation and coding scheme

0 (default) | integer from 0 to 31

Modulation and coding scheme to use for transmitting the current packet, specified as an integer from 0 to 31. The MCS setting identifies which modulation and coding rate combination is used, and the number of spatial streams (N_{SS}).

MCS(Note 1)	N_{SS}(Note 1)	Modulation	Coding Rate
0, 8, 16, or 24	1, 2, 3, or 4	`BPSK`	`1/2`
1, 9, 17, or 25	1, 2, 3, or 4	`QPSK`	`1/2`
2, 10, 18, or 26	1, 2, 3, or 4	`QPSK`	`3/4`
3, 11, 19, or 27	1, 2, 3, or 4	`16QAM`	`1/2`
4, 12, 20, or 28	1, 2, 3, or 4	`16QAM`	`3/4`
5, 13, 21, or 29	1, 2, 3, or 4	`64QAM`	`2/3`
6, 14, 22, or 30	1, 2, 3, or 4	`64QAM`	`3/4`
7, 15, 23, or 31	1, 2, 3, or 4	`64QAM`	`5/6`
Note-1 MCS from 0 to 7 have one spatial stream. MCS from 8 to 15 have two spatial streams. MCS from 16 to 23 have three spatial streams. MCS from 24 to 31 have four spatial streams.			

See IEEE 802.11-2012, Section 20.6 for further description of MCS dependent parameters.

When working with the HT-Data field, if the number of space-time streams is equal to the number of spatial streams, no space-time block coding (STBC) is used. See IEEE 802.11-2012, Section 20.3.11.9.2 for further description of STBC mapping.

Example: `22` indicates an MCS with three spatial streams, 64-QAM modulation, and a 3/4 coding rate.

Data Types: `double`

`GuardInterval` — Cyclic prefix length for the data field within a packet

`'Long'` (default) | `'Short'`

Cyclic prefix length for the data field within a packet, specified as `'Long'` or `'Short'`.

- The long guard interval length is 800 ns.
- The short guard interval length is 400 ns.

Data Types: `char`

`ChannelCoding` — Type of forward error correction coding

`'BCC'` (default)

This property is read only.

Type of forward error correction coding for the data field, specified as `'BCC'` to indicate binary convolutional coding. LDPC coding is not currently supported.

Data Types: `char`

`PSDULength` — Number of bytes carried in the user payload

1024 (default) | integer from 0 to 65,535

Number of bytes carried in the user payload, specified as an integer from 0 to 65,535. A `PSDULength` of 0 implies a sounding packet for which there are no data bits to recover.

Example: `512`

Data Types: `double`

`RecommendSmoothing` — Indication of whether frequency-domain smoothing is recommended

true (default) | false

Indication of whether frequency-domain smoothing is recommended as part of channel estimation, specified as a logical. If the frequency profile across the channel is nonvarying, the receiver sets this property to `true`.

Data Types: `logical`

See Also

`wlanHTConfig` | `wlanNonHTConfig` | `wlanVHTConfig` | `wlanWaveformGenerator`

Introduced in R2015b

wlanNonHTConfig Properties

Define parameter values for non-HT format packet

The `wlanNonHTConfig` object specifies the transmission properties for the IEEE 802.11 non-high throughput (non-HT) format physical layer (PHY) packet.

After you create an object, use dot notation to change or access the object parameters. For example:

Create a `wlanNonHTConfig` object. Then modify the default setting for the `PSDULength` property.

```
cfgNonHT = wlanNonHTConfig;
cfgNonHT.PSDULength = 3025;
```

Non-HT Format Configuration

`Modulation` — Modulation type for non-HT transmission

`'OFDM'` (default) | `'DSSS'`

Modulation type for the non-HT transmission packet, specified as `'OFDM'` or `'DSSS'`.

Data Types: `char`

`MCS` — OFDM modulation and coding scheme

0 (default) | integer from 0 to 7 | integer

OFDM modulation and coding scheme to use for transmitting the current packet, specified as an integer from 0 to 7. The system configuration associated with an `MCS` setting maps to the specified data rate.

MCS	Modulation	Coding Rate	Data Rate (Mbps)
0	BPSK	1/2	6
1	BPSK	3/4	9
2	QPSK	1/2	12
3	QPSK	3/4	18
4	16QAM	1/2	24
5	16QAM	3/4	36

MCS	Modulation	Coding Rate	Data Rate (Mbps)
6	64QAM	2/3	48
7	64QAM	3/4	54

See IEEE Std 802.11-2012, Table 18-4.

Data Types: `double`

`DataRate` — DSSS modulation data rate

`'1Mbps'` (default) | `'2Mbps'` | `'5.5Mbps'` | `'11Mbps'`

DSSS modulation data rate, specified as `'1Mbps'`, `'2Mbps'`, `'5.5Mbps'`, or `'11Mbps'`.

- `'1Mbps'` uses differential binary phase shift keying (DBPSK) modulation with a 1 Mbps data rate.
- `'2Mbps'` uses differential quadrature phase shift keying (DQPSK) modulation with a 2 Mbps data rate.
- `'5.5Mbps'` uses complementary code keying (CCK) modulation with a 5.5 Mbps data rate.
- `'11Mbps'` uses complementary code keying (CCK) modulation with an 11 Mbps data rate.

For IEEE Std 802.11-2012, Section 16, only `'1Mbps'` and `'2Mbps'` apply

Data Types: `char`

`Preamble` — Preamble type for DSSS modulation

`'Long'` (default) | `'Short'`

Preamble type for DSSS modulation, specified as `'Long'` or `'Short'`.

- When `DataRate` is `'1Mbps'`, the `Preamble` setting is ignored and `'Long'` is used.
- When `DataRate` is greater than `'1Mbps'`, the `Preamble` property is available and can be set to `'Long'` or `'Short'`.

For IEEE Std 802.11-2012, Section 16, `'Short'` does not apply.

Data Types: `char`

`LockedClocks` — Clock locking indication for DSSS modulation

true (default) | false

Clock locking indication for DSSS modulation, specified as a logical. Bit 2 of the SERVICE field is the *Locked Clock Bit*. A `true` setting indicates that the PHY implementation derives its transmit frequency clock and symbol clock from the same oscillator. For more information, see IEEE Std 802.11-2012, Section 17.2.3.5 and Section 19.1.3.

Note:

- IEEE Std 802.11-2012, Section 19.3.2.2, specifies locked clocks is required for all ERP systems when transmitting at the ERP-PBCC rate or at a data rate described in Section 17. Therefore to model ERP systems, set `LockedClocks` to `true`.

Data Types: `logical`

`PSDULength` — Number of bytes carried in the user payload
1000 (default) | integer from 1 to 4095 | integer

Number of bytes carried in the user payload, specified as an integer from 1 to 4095.

Data Types: `double`

`NumTransmitAntennas` — Number of transmit antennas
1 (default) | integer from 1 to 8

Number of transmit antennas, specified as a scalar integer from 1 to 8.

Data Types: `double`

See Also

`wlanLLFChannelEstimate` | `wlanLLTF` | `wlanLLTFDemodulate` | `wlanLSIG` | `wlanLSIGRecover` | `wlanLSTF` | `wlanNonHTConfig` | `wlanNonHTData` | `wlanNonHTDataRecover` | `wlanWaveformGenerator`

Introduced in R2015b

wlanRecoveryConfig Properties

Define parameter values for data recovery

The `wlanRecoveryConfig` object specifies properties for recovering data from IEEE 802.11 transmissions.

After you create an object, use dot notation to change or access the object parameters. For example:

Create a `wlanRecoveryConfig` object. Then modify the default setting for the `OFDMSymbolOffset` property.

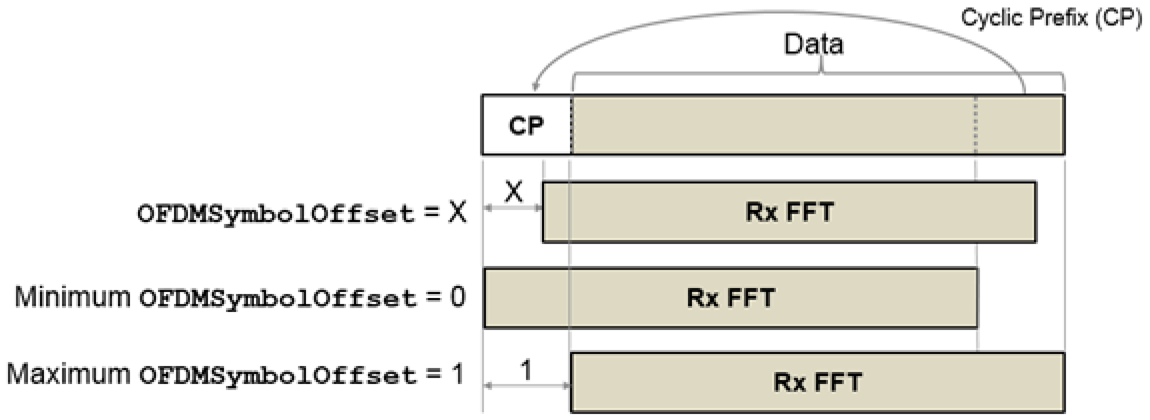
```
cfgRec = wlanRecoveryConfig;
cfgRec.OFDMSymbolOffset = 0.65;
```

Date Recovery Configuration

`OFDMSymbolOffset` — OFDM symbol sampling offset

0.75 (default) | scalar value from 0 to 1

OFDM symbol sampling offset represented as a fraction of the cyclic prefix (CP) length, specified as a scalar value from 0 to 1. This value indicates the start location for OFDM demodulation, relative to the beginning of the cyclic prefix. `OFDMSymbolOffset` = 0 represents the start of the cyclic prefix and `OFDMSymbolOffset` = 1 represents the end of the cyclic prefix.



Data Types: `double`

`EqualizationMethod` — Equalization method

`'MMSE'` (default) | `'ZF'`

Equalization method, specified as `'MMSE'` or `'ZF'`.

- `'MMSE'` indicates that the receiver uses a minimum mean square error equalizer.
- `'ZF'` indicates that the receiver uses a zero-forcing equalizer.

Example: `'ZF'`

Data Types: `char`

`PilotPhaseTracking` — Pilot phase tracking

`'PreEQ'` (default) | `'None'`

Pilot phase tracking, specified as either of these strings: `'PreEQ'` or `'None'`. If `'PreEQ'` is specified, pilot phase tracking is enabled and is performed prior to any equalization operation. If `'None'` is specified, pilot phase tracking does not occur.

Data Types: `char`

See Also

`wlanHTConfig` | `wlanNonHTConfig` | `wlanRecoveryConfig` | `wlanVHTConfig`

Introduced in R2015b

Classes — Alphabetical List

wlanTGacChannel System object

Filter signal through 802.11ac multipath fading channel

Description

The `wlanTGacChannel` System object™ filters an input signal through an 802.11ac (TGac) multipath fading channel.

The fading processing assumes the same parameters for all N_T-by-N_R links of the TGac channel, where N_T is the number of transmit antennas and N_R is the number of receive antennas. Each link comprises all multipaths for that link.

To filter an input signal using a TGac multipath fading channel:

1. Define and set up your TGac channel object. See “Construction” on page 3-2.
2. Call step to filter the input signal through a TGac multipath fading channel according to the properties of `wlanTGacChannel`.

Construction

`tgac = wlanTGacChannel` creates a TGac fading channel System object, `tgac`. This object filters a real or complex input signal through the TGac channel to obtain the channel-impaired signal.

`tgac = wlanTGacChannel(Name,Value)` creates a TGac channel object, `tgac`, with the specified property `Name` set to the specified `Value`. You can specify additional name-value pair arguments in any order as `Name1`,`Value1`,...,`NameN`,`ValueN`.

Properties

`SampleRate`

Input signal sample rate (Hz)

Sample rate of the input signal in Hz, specified as a real positive scalar. The default is `80e6`.

`DelayProfile`

Delay profile model

Delay profile model, specified as a string. Permissible values are `'Model-A'`, `'Model-B'`, `'Model-C'`, `'Model-D'`, `'Model-E'`, and `'Model-F'`. The default is `'Model-B'`. To enable the `FluorescentEffect` property, select either `'Model-D'` or `'Model-E'`.

The table summarizes the models.

Parameter	Model					
	A	B	C	D	E	F
Breakpoint distance (m)	5	5	5	10	20	30
RMS delay spread (ns)	0	15	30	50	100	150
Maximum delay (ns)	0	80	200	390	730	1050
Rician K-factor (dB)	0	0	0	3	6	6
Number of taps	1	9	14	18	18	18
Number of clusters	1	2	2	3	4	6

The number of clusters represents the number of independently modeled propagation paths.

`ChannelBandwidth`

Channel bandwidth

Channel bandwidth in MHz, specified as one of these strings: `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`. The default is `'CBW80'`, which corresponds to an 80 MHz channel bandwidth.

`CarrierFrequency`

RF carrier frequency

RF carrier frequency in Hz, specified as a real positive scalar. The default is `5.25e9`.

`NormalizePathGains`

Normalize path gains

To normalize the fading processes such that the total power of the path gains, averaged over time, is 0 dB, set this property to `true` (default). When you set this property to `false`, the path gains are not normalized.

`UserIndex`

User index

User index, specified as a nonnegative integer scalar. The default is `0`.

`TransmissionDirection`

Transmission direction

Transmission direction of the active link, specified as either `'Uplink'` or `'Downlink'`. The default is `'Downlink'`.

`NumTransmitAntennas`

Number of transmit antennas

Number of transmit antennas, specified as a positive integer scalar from `1` to `8`. The default is `1`.

`TransmitAntennaSpacing`

Distance between transmit antenna elements

Distance between transmit antenna elements, specified as a real positive scalar expressed in wavelengths. The default is `0.5`. This property is available when `NumTransmitAntennas` is greater than `1`.

`NumReceiveAntennas`

Number of receive antennas

Number of receive antennas, specified as a positive integer scalar from `1` to `8`. The default is `1`.

`ReceiveAntennaSpacing`

Distance between receive antenna elements

Distance between receive antenna elements, specified as a real positive scalar expressed in wavelengths. The default is `0.5`. This property is available when `NumReceiveAntennas` is greater than `1`.

`LargeScaleFadingEffect`

Large-scale fading effects

Type of large-scale fading effects, specified as `'None'`, `'Pathloss'`, `'Shadowing'`, or `'Pathloss and shadowing'`. The default is `'None'`.

`TransmitReceiveDistance`

Distance between the transmitter and receiver (m)

Distance in meters between the transmitter and receiver, specified as a real positive scalar. The default is `3`.

`FluorescentEffect`

Enable fluorescent effect

To include Doppler effects due to fluorescent lighting, set this property to `true` (default). This property is available when you set `DelayProfile` to `'Model-D'` or `'Model-E'`.

`PowerLineFrequency`

Frequency of the power line (Hz)

Frequency of the power line in Hz, specified as a one of these strings: `'50Hz'` or `'60Hz'`. The default is `'60Hz'`. This property is available when you set `FluorescentEffect` to `true` and `DelayProfile` to `'Model-D'` or `'Model-E'`.

`RandomStream`

Source of random number stream

Source of random number stream, specified as `'Global stream'` or `'mt19937ar with seed'`. The default is `'Global stream'`.

If you set `RandomStream` to `'Global stream'`, the current global random number stream is used for normally distributed random number generation. In this case, the `reset` method resets the filters only.

If you set `RandomStream` to `'mt19937ar with seed'`, the mt19937ar algorithm is used for normally distributed random number generation. In this case, the `reset` method not only resets the filters but also reinitializes the random number stream to the value of the `Seed` property.

`Seed`

Initial seed of mt19937ar random number stream

Initial seed of an mt19937ar random number generator algorithm, specified as a real, nonnegative integer scalar. The default is `73`. This property applies when you set the `RandomStream` property to `'mt19937ar with seed'`. The `Seed` property reinitializes the mt19937ar random number stream in the `reset` method.

`NormalizeChannelOutputs`

Normalize channel outputs

To normalize the channel outputs by the number of receive antennas, set this property to `true` (default).

Methods

clone	Create `wlanTGacChannel` object with same property values
info	Characteristic information about TGac Channel
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes

reset	Reset states of the `wlanTGacChannel` object
step	Filter signal through 802.11ac multipath fading channel

Examples

Transmit VHT Waveform Through TGac Channel

Generate a VHT waveform and pass it through a TGac SISO channel. Display the spectrum of the resultant signal.

Set the channel bandwidth and the corresponding sample rate.

```
bw = 'CBW80';
fs = 80e6;
```

Generate a VHT waveform.

```
cfg = wlanVHTConfig;
wgc = wlanGeneratorConfig;
txSig = wlanWaveformGenerator(randi([0 1],1000,1),cfg,wgc);
```

Create a TGac SISO channel with path loss and shadowing enabled.

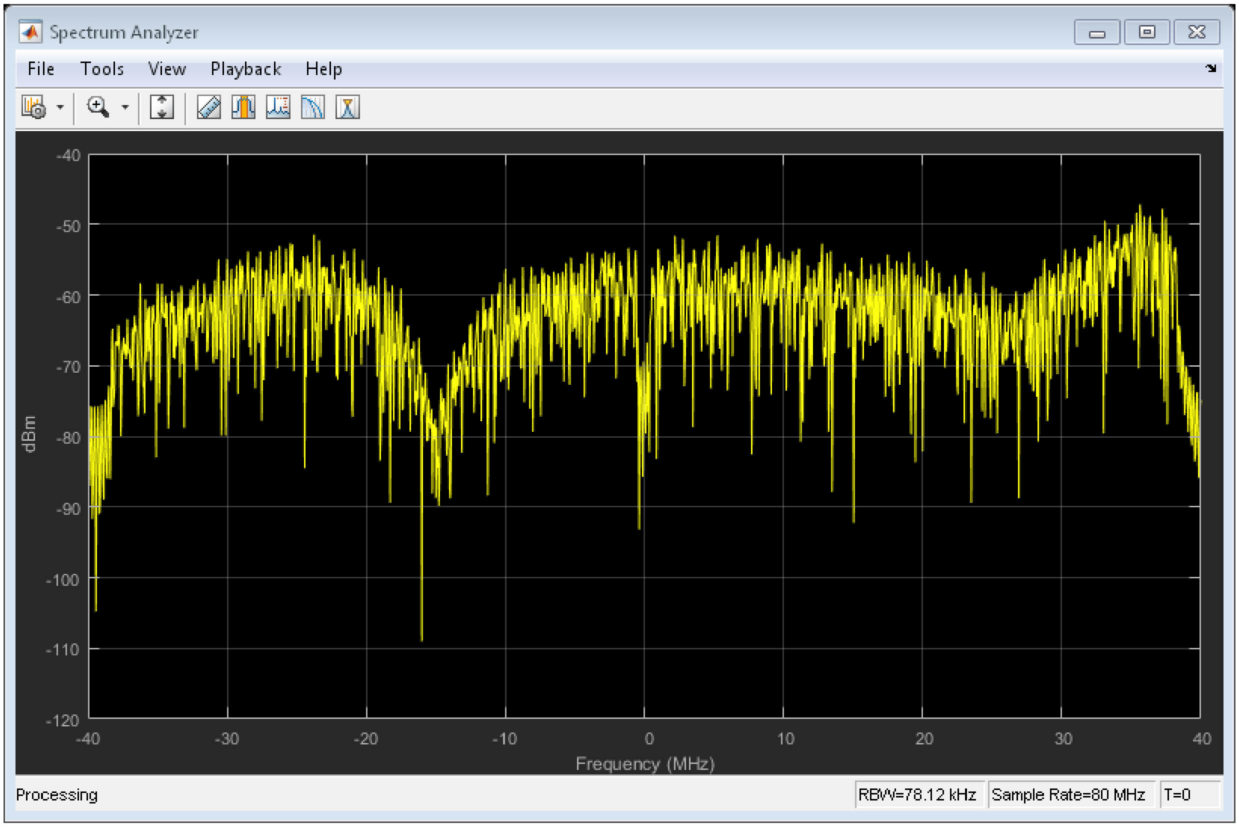
```
chan = wlanTGacChannel('SampleRate',fs,'ChannelBandwidth',bw, ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
```

Pass the VHT waveform through the channel.

```
rxSig = step(chan,txSig);
```

Plot the spectrum of the received waveform.

```
sa = dsp.SpectrumAnalyzer('SampleRate',fs,'YLimits',[-120 -40]);
step(sa,rxSig)
```



Because path loss and shadowing are enabled, the mean received power across the spectrum is approximately -60 dBm.

Transmit VHT Waveform Through 4x2 MIMO Channel

Create a VHT waveform having four transmit antennas and two space-time streams.

```
cfg = wlanVHTConfig('NumTransmitAntennas',4,'NumSpaceTimeStreams',2, ...
    'SpatialMapping','Fourier');
wgc = wlanGeneratorConfig;
txSig = wlanWaveformGenerator([1;0;0;1],cfg,wgc);
```

Create a 4x2 MIMO TGac channel and disable large-scale fading effects.

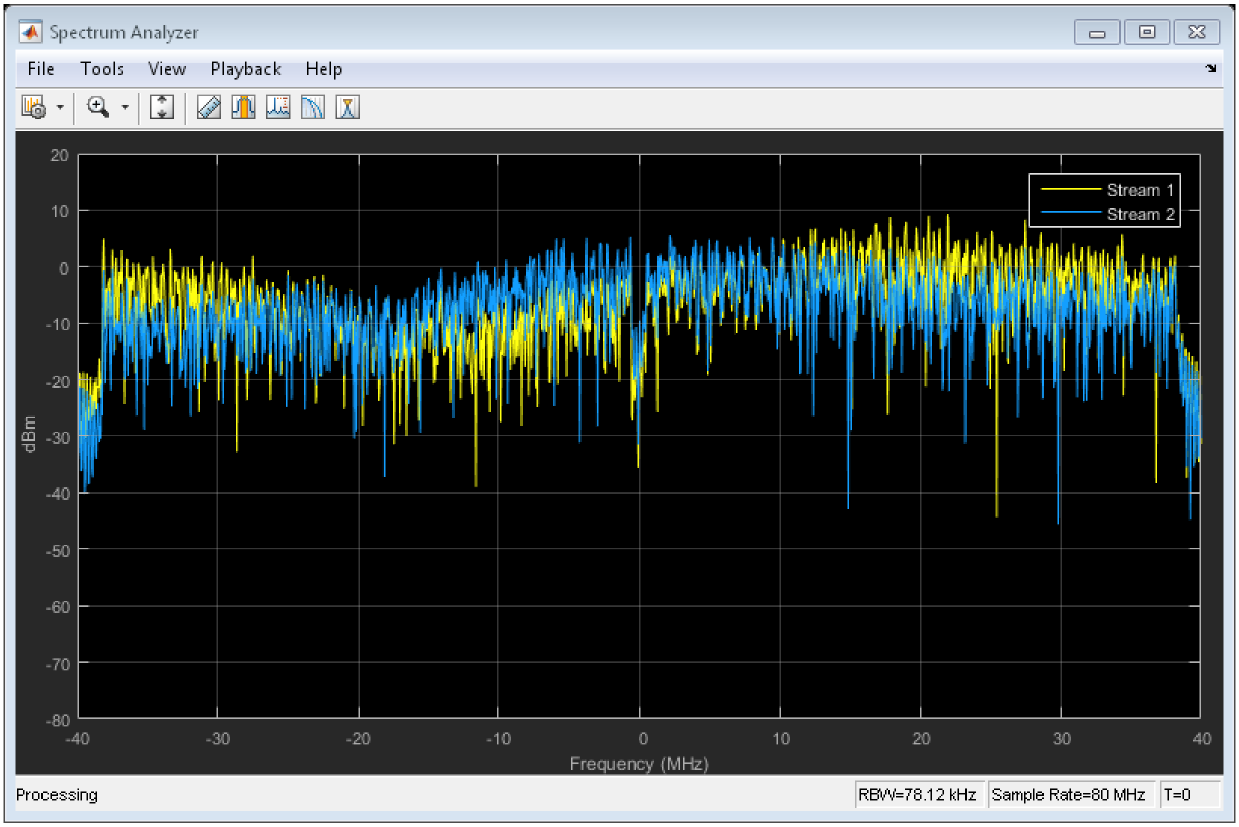
```
chan = wlanTGacChannel('SampleRate',80e6,'ChannelBandwidth','CBW80', ...
    'NumTransmitAntennas',4,'NumReceiveAntennas',2, ...
    'LargeScaleFadingEffect','None');
```

Pass the transmit waveform through the channel.

```
rxSig = step(chan,txSig);
```

Display the spectrum of the two received space-time streams.

```
sa = dsp.SpectrumAnalyzer('SampleRate',80e6, ...
    'ShowLegend',true, ...
    'ChannelNames',{'Stream 1','Stream 2'});
step(sa,rxSig)
```



Recover VHT Data from 2x2 MIMO Channel

Transmit a VHT-LTF and a VHT data field through a noisy 2x2 MIMO channel. Demodulate the received VHT-LTF to estimate the channel coefficients. Recover the VHT data and determine the number of bit errors.

Set the channel bandwidth and corresponding sample rate.

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

Create VHT-LTF and VHT data fields having two transmit antennas and two space-time streams.

```
cfg = wlanVHTConfig('ChannelBandwidth',bw, ...
    'NumTransmitAntennas',2,'NumSpaceTimeStreams',2);
txPSDU = randi([0 1],8*cfg.PSDULength,1);
txLTF = wlanVHTLTF(cfg);
txDataSig = wlanVHTData(txPSDU,cfg);
```

Create a 2x2 MIMO TGac channel.

```
tgac = wlanTGacChannel('SampleRate',fs,'ChannelBandwidth',bw, ...
    'NumTransmitAntennas',2,'NumReceiveAntennas',2, ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
```

Create an AWGN channel for a 160 MHz channel with a 9 dB noise figure. The noise variance, `nVar`, is equal to $kTBF$, where k is Boltzmann's constant, T is the ambient temperature of 290 K, B is the bandwidth (sample rate), and F is the receiver noise figure.

```
nVar = 10^((-228.6 + 10*log10(290) + 10*log10(fs) + 9)/10);
chAWGN = comm.AWGNChannel('NoiseMethod','Variance','Variance',nVar);
```

Pass the signals through the channel.

```
rxLTF = step(chAWGN,step(tgac,txLTF));
rxDataSig = step(chAWGN,step(tgac,txDataSig));
```

Demodulate the VHT-LTF. Use the demodulated signal to estimate the channel coefficients.

```
dLTF = wlanVHTLTFDemodulate(rxLTF,cfg);
chEst = wlanVHTLTFChannelEstimate(dLTF,cfg);
```

Recover the data and determine the number of bit errors.

```
rxPSDU = wlanVHTDataRecover(rxDataSig,chEst,nVar,cfg);
numErr = biterr(txPSDU,rxPSDU)
```

```
numErr =

     0
```

Algorithms

The algorithms used to model the TGac channel are based on those used for the TGn channel and are described in wlanTGnChannel and [1]. The changes to support the TGac channel include:

- Increased bandwidth
- Higher-order MIMO
- Multi-user MIMO
- Reduced Doppler
- Dual-polarized antennas

Complete information on the changes required to support TGac channels can be found in [2].

Increased Bandwidth

TGac channels support bandwidths of up to 1.28 GHz, whereas TGn channels have a maximum bandwidth of 40 MHz. By increasing the sampling rate and decreasing the tap spacing of the power delay profile (PDP), the TGn model is used as the basis for TGac. The channel sampling rate is increased by a factor of $2^{\lceil \log_2(W/40) \rceil}$, where W is the bandwidth. The PDP tap spacing is reduced by the same factor.

Bandwidth, *W*	**Sampling Rate Expansion Factor**	**PDP Tap Spacing (ns)**
$W \leq 40$ MHz	1	10
40 MHz < $W \leq 80$ MHz	2	5
80 MHz < $W \leq 160$ MHz	4	2.5
160 MHz < $W \leq 320$ MHz	8	1.25
320 MHz < $W \leq 640$ MHz	16	0.625
640 MHz < $W \leq 1280$ MHz	32	0.3125

MIMO Enhancements

The TGn channel model supports no more than 4x4 MIMO, while the TGac model supports 8x8 MIMO.

The TGac model also includes support for multiple users as simultaneous communication takes place between access points and user stations. Accordingly, the TGac model extends the concept of cluster angles of arrival and departure to account for point-to-multipoint transmission. Further details are described in [2].

Reduced Doppler

Indoor channel measurements indicate that the magnitude of Doppler assumed in the TGn channel model is too high for stationary users. As such, the TGac channel model uses a reduced environment velocity of 0.089 km/hr. This model assumes a coherence time of 800 ms or, equivalently, an RMS Doppler spread of 0.4 Hz for a 5 GHz carrier frequency.

References

[1] Erceg, V., L. Schumacher, P. Kyritsi, et al. *TGn Channel Models*. Version 4. IEEE 802.11-03/940r4, May 2004.

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

[3] Kermoal, J. P., L. Schumacher, K. I. Pedersen, P. E. Mogensen, and F. Frederiksen, "A Stochastic MIMO Radio Channel Model with Experimental Validation". *IEEE Journal on Selected Areas in Communications*., Vol. 20, No. 6, August 2002, pp. 1211–1226.

See Also

wlanTGnChannel | comm.MIMOChannel

Introduced in R2015b

clone

System object: wlanTGacChannel

Create `wlanTGacChannel` object with same property values

Syntax

```
C = clone(H)
```

Description

`C = clone(H)` creates a `wlanTGacChannel` object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object with uninitialized states.

The clone method creates an instance of an object. The property values, but not internal states, are copied into the new instance of the object.

info

System object: wlanTGacChannel

Characteristic information about TGac Channel

Syntax

`S = info(OBJ)`

Description

`S = info(OBJ)` returns a structure, `S`, containing characteristic information about the wlanTGacChannel object, `OBJ`. The list summarizes the information contained in `S`.

- `ChannelFilterDelay`: Filter delay introduced by the implementation (samples)
- `PathDelays`: Delay of each of the discrete paths (seconds)
- `AveragePathGains`: Average gain of each of the discrete paths (dB)
- `Pathloss`: Path loss between the transmitter and receiver (dB).

isLocked

System object: wlanTGacChannel

Locked status for input attributes and nontunable properties

Syntax

```
TF = isLocked(H)
```

Description

`TF = isLocked(H)` returns the locked status, `TF` of the `wlanTGacChannel` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal initialization the first time the `step` method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. After locking, the `isLocked` method returns a `true` value.

release

System object: wlanTGacChannel

Allow property value and input characteristics changes

Syntax

`release(H)`

Description

`release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note: You can use the `release` method on a System object in code generated from MATLAB®, but once you release its resources, you cannot use that System object again.

reset

System object: wlanTGacChannel

Reset states of the `wlanTGacChannel` object

Syntax

`reset(H)`

Description

`reset(H)` resets the states of the `wlanTGacChannel` object, `H`.

If you set the `RandomStream` property of `H` to `Global stream`, the `reset` method only resets the filters. If you set `RandomStream` to `mt19937ar with seed`, the `reset` method not only resets the filters but also reinitializes the random number stream to the value of the `Seed` property.

step

System object: wlanTGacChannel

Filter signal through 802.11ac multipath fading channel

Syntax

```
Y = step(TGAC,X)
[Y,PATHGAINS] = step(TGAC,X)
```

Description

`Y = step(TGAC,X)` filters input signal `X` through 802.11ac (TGac) fading channel `TGAC` and returns the result in `Y`. The input `X` can be a double-precision data type scalar, vector, or 2-D matrix with real or complex values. `X` is of size N_s-by-N_t, where N_s represents the number of samples and N_t represents the number of transmit antennas. `Y` is the output signal of size N_s-by-N_r, where N_r represents the number of receive antennas. `Y` is of double-precision data type with complex values.

`[Y,PATHGAINS] = step(TGAC,X)` returns a complex N_s-by-N_p-by-N_t-by-N_r matrix `PATHGAINS` for the TGac channel System object, `TGAC`. N_p is the number of paths in the channel.

Note: `TGAC` specifies the System object on which to run this `step` method.

The object performs an initialization the first time the `step` method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Examples

Transmit VHT Waveform Through TGac Channel

Generate a VHT waveform and pass it through a TGac SISO channel. Display the spectrum of the resultant signal.

Set the channel bandwidth and the corresponding sample rate.

```
bw = 'CBW80';
fs = 80e6;
```

Generate a VHT waveform.

```
cfg = wlanVHTConfig;
wgc = wlanGeneratorConfig;
txSig = wlanWaveformGenerator(randi([0 1],1000,1),cfg,wgc);
```

Create a TGac SISO channel with path loss and shadowing enabled.

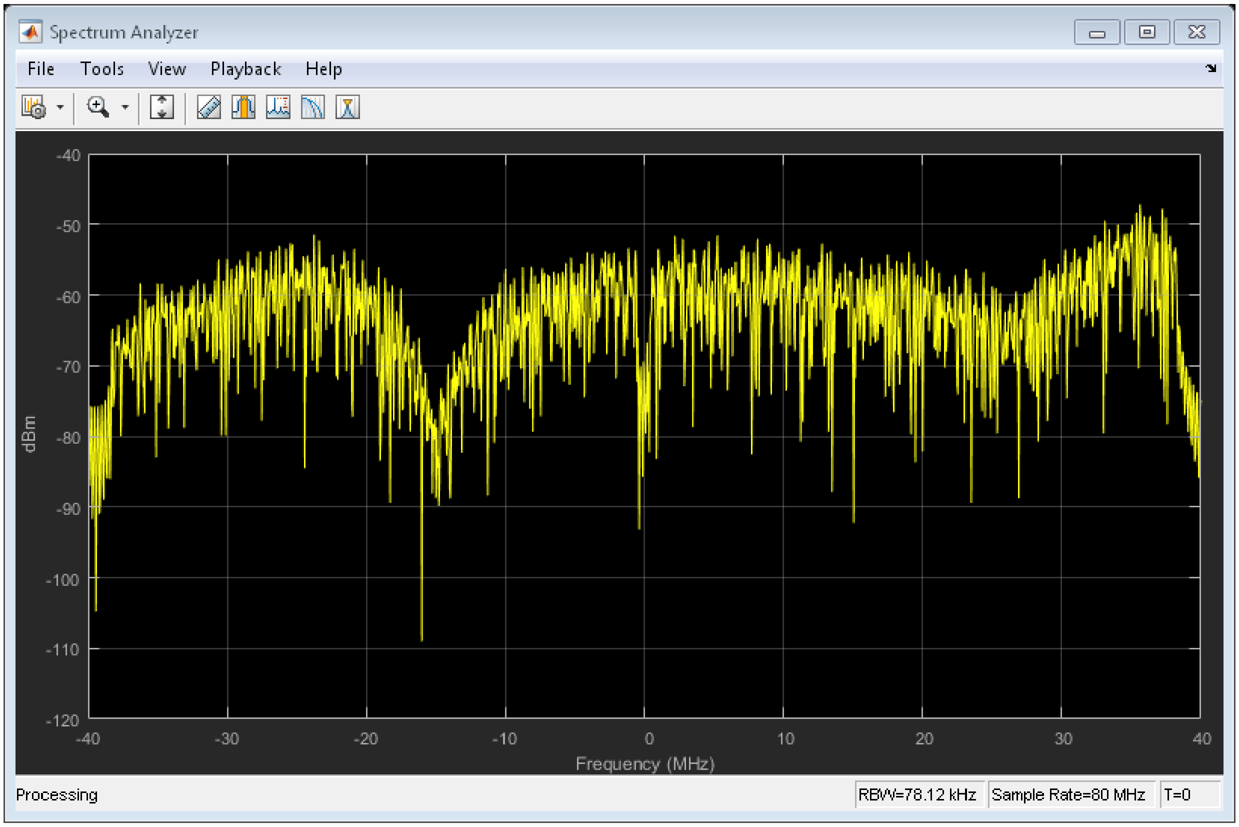
```
chan = wlanTGacChannel('SampleRate',fs,'ChannelBandwidth',bw, ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
```

Pass the VHT waveform through the channel.

```
rxSig = step(chan,txSig);
```

Plot the spectrum of the received waveform.

```
sa = dsp.SpectrumAnalyzer('SampleRate',fs,'YLimits',[-120 -40]);
step(sa,rxSig)
```



Because path loss and shadowing are enabled, the mean received power across the spectrum is approximately -60 dBm.

wlanTGnChannel System object

Filter signal through 802.11n multipath fading channel

Description

The `wlanTGnChannel` System object filters an input signal through an 802.11n (TGn) multipath fading channel.

The fading processing assumes the same parameters for all N_T-by-N_R links of the TGn channel. N_T is the number of transmit antennas and N_R is the number of receive antennas. Each link comprises all multipaths for that link.

To filter an input signal using a TGn multipath fading channel:

1. Define and set up your TGn channel object. See "Construction" on page 3-22.
2. Call step to filter the input signal through a TGn multipath fading channel according to the properties of `wlanTGnChannel`.

Construction

`tgn = wlanTGnChannel` creates a TGn fading channel System object, `tgn`. This object filters a real or complex input signal through the TGn channel to obtain the channel-impaired signal.

`tgn = wlanTGnChannel(Name,Value)` creates a TGn channel object, `tgn`, with the specified property `Name` set to the specified `Value`. You can specify additional name-value pair arguments in any order as `Name1`,`Value1`,...,`NameN`,`ValueN`.

Properties

`SampleRate`

Input signal sample rate (Hz)

Sample rate of the input signal in Hz, specified as a real positive scalar. The default is `20e6`.

`DelayProfile`

Delay profile model

Delay profile model, specified as a string. Permissible values are `'Model-A'`, `'Model-B'`, `'Model-C'`, `'Model-D'`, `'Model-E'`, and `'Model-F'`. The default is `'Model-B'`. To enable the `FluorescentEffect` property, select either `'Model-D'` or `'Model-E'` .

Parameter	Model					
	A	B	C	D	E	F
Breakpoint distance (m)	5	5	5	10	20	30
RMS delay spread (ns)	0	15	30	50	100	150
Maximum delay (ns)	0	80	200	390	730	1050
Rician K-factor (dB)	0	0	0	3	6	6
Number of clusters	1	2	2	3	4	6
Number of taps	1	9	14	18	18	18

`CarrierFrequency`

RF carrier frequency (Hz)

Carrier frequency of the channel in Hz, specified as a real positive scalar. The default is `5.25e9`.

`NormalizePathGains`

Normalize path gains

To normalize the fading processes such that the total power of the path gains, averaged over time, is 0 dB, set this property to `true` (default). When you set this property to `false`, the path gains are not normalized.

`NumTransmitAntennas`

Number of transmit antennas

Number of transmit antennas, specified as a positive integer scalar from `1` to `4`. The default is `1`.

`TransmitAntennaSpacing`

Distance between transmit antenna elements

Distance between transmit antenna elements, specified as a real positive scalar expressed in wavelengths. The default is `0.5`. This property is available when `NumTransmitAntennas` is greater than `1`.

`NumReceiveAntennas`

Number of receive antennas

Number of receive antennas, specified as a positive integer scalar from `1` to `4`. The default is `1`.

`ReceiveAntennaSpacing`

Distance between receive antenna elements

Distance between receive antenna elements, specified as a real positive scalar expressed in wavelengths. The default is `0.5`. This property is available when `NumReceiveAntennas` is greater than `1`.

`LargeScaleFadingEffect`

Large scale fading effects

Type of large-scale fading effects, specified as one of these strings: `'None'`, `'Pathloss'`, `'Shadowing'`, or `'Pathloss and shadowing'`. The default is `'None'`.

`TransmitReceiveDistance`

Distance between the transmitter and receiver (m)

Distance in meters between the transmitter and receiver, specified as a real positive scalar. The default is `3`.

`FluorescentEffect`

Enable fluorescent effect

To include Doppler effects due to fluorescent lighting, set this property to `true` (default). This property is available when `DelayProfile` is `'Model-D'` or `'Model-E'`.

`PowerLineFrequency`

Frequency of the power line (Hz)

Frequency of the power line in Hz, specified as either `'50Hz'` or `'60Hz'`. The default is `'60Hz'`. This property is available when `FluorescentEffect` is `true` and `DelayProfile` is `'Model-D'` or `'Model-E'`.

`RandomStream`

Source of random number stream

Source of random number stream, specified as `'Global stream'` or `'mt19937ar with seed'`. The default is `'Global stream'`.

If you set `RandomStream` to `'Global stream'`, the current global random number stream is used for normally distributed random number generation. In this case, the `reset` method resets the filters only.

If you set `RandomStream` to `'mt19937ar with seed'`, the mt19937ar algorithm is used for normally distributed random number generation. In this case, the `reset` method not only resets the filters but also reinitializes the random number stream to the value of the `Seed` property.

`Seed`

Initial seed of mt19937ar random number stream

Initial seed of an mt19937ar random number generator algorithm, specified as a real, nonnegative integer scalar. The default is `73`. This property applies when you set the `RandomStream` property to `'mt19937ar with seed'`. The `Seed` property reinitializes the mt19937ar random number stream in the `reset` method.

`NormalizeChannelOutputs`

Normalize channel outputs

To normalize the channel outputs by the number of receive antennas, set this property to `true` (default).

Methods

clone	Create `wlanTGnChannel` object with same property values
info	Characteristic information about TGn Channel
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
reset	Reset states of the `wlanTGnChannel` object
step	Filter signal through 802.11n multipath fading channel

Examples

Transmit HT Waveform Through TGn Channel

Generate an HT waveform and pass it through a TGn SISO channel. Display the spectrum of the resultant signal.

Set the channel bandwidth and the corresponding sample rate.

```
bw = 'CBW40';
fs = 40e6;
```

Generate an HT waveform for a 40 MHz channel.

```
cfg = wlanHTConfig('ChannelBandwidth',bw);
wgc = wlanGeneratorConfig;
txSig = wlanWaveformGenerator(randi([0 1],1000,1),cfg,wgc);
```

Create a TGn SISO channel with path loss and shadowing enabled.

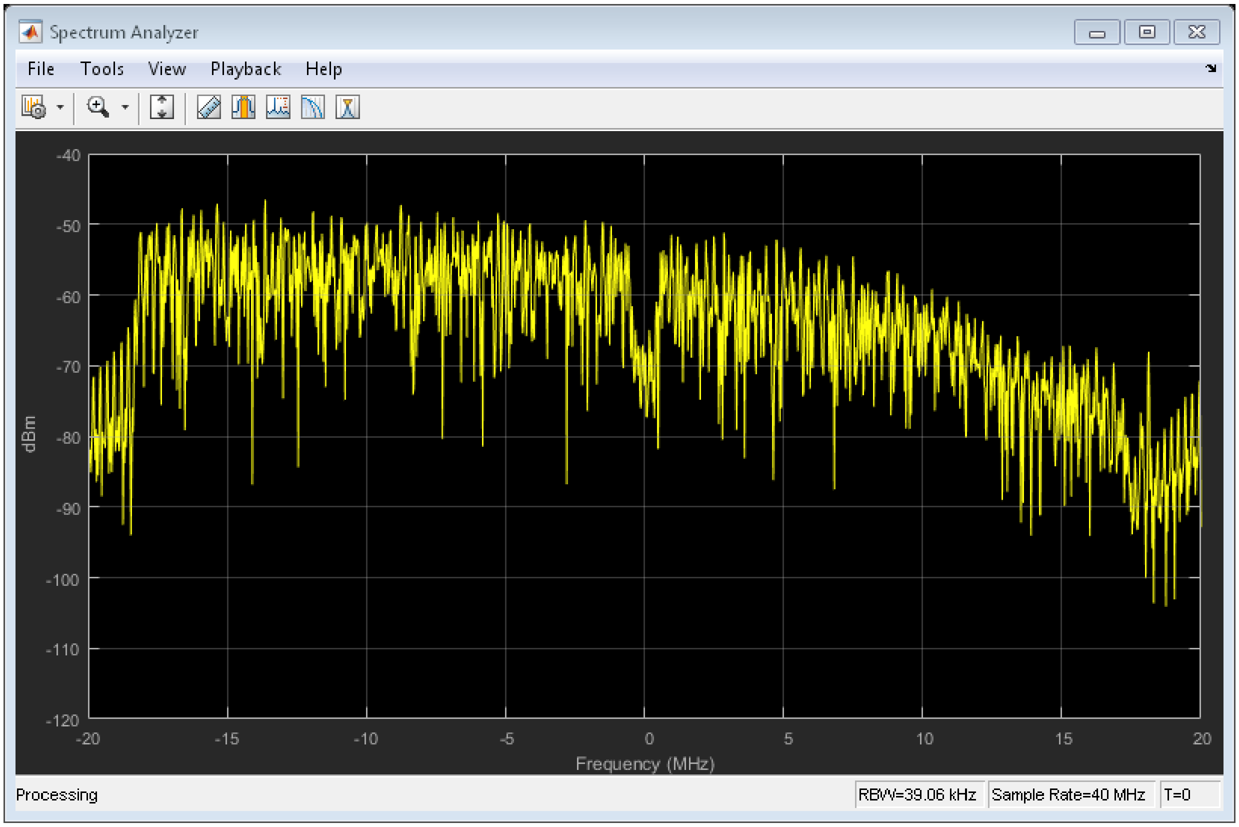
```
chan = wlanTGnChannel('SampleRate',fs, ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
```

Pass the HT waveform through the channel.

```
rxSig = step(chan,txSig);
```

Plot the spectrum of the received waveform.

```
sa = dsp.SpectrumAnalyzer('SampleRate',fs,'YLimits',[-120 -40]);
step(sa,rxSig)
```



Because path loss and shadowing are enabled, the mean received power across the spectrum is approximately -60 dBm.

Transmit HT Waveform Through 4x2 MIMO Channel

Create an HT waveform having four transmit antennas and two space-time streams.

```
cfg = wlanHTConfig('NumTransmitAntennas',4,'NumSpaceTimeStreams',2, ...
    'SpatialMapping','Fourier');
wgc = wlanGeneratorConfig;
txSig = wlanWaveformGenerator([1;0;0;1],cfg,wgc);
```

Create a 4x2 MIMO TGn channel and disable large-scale fading effects.

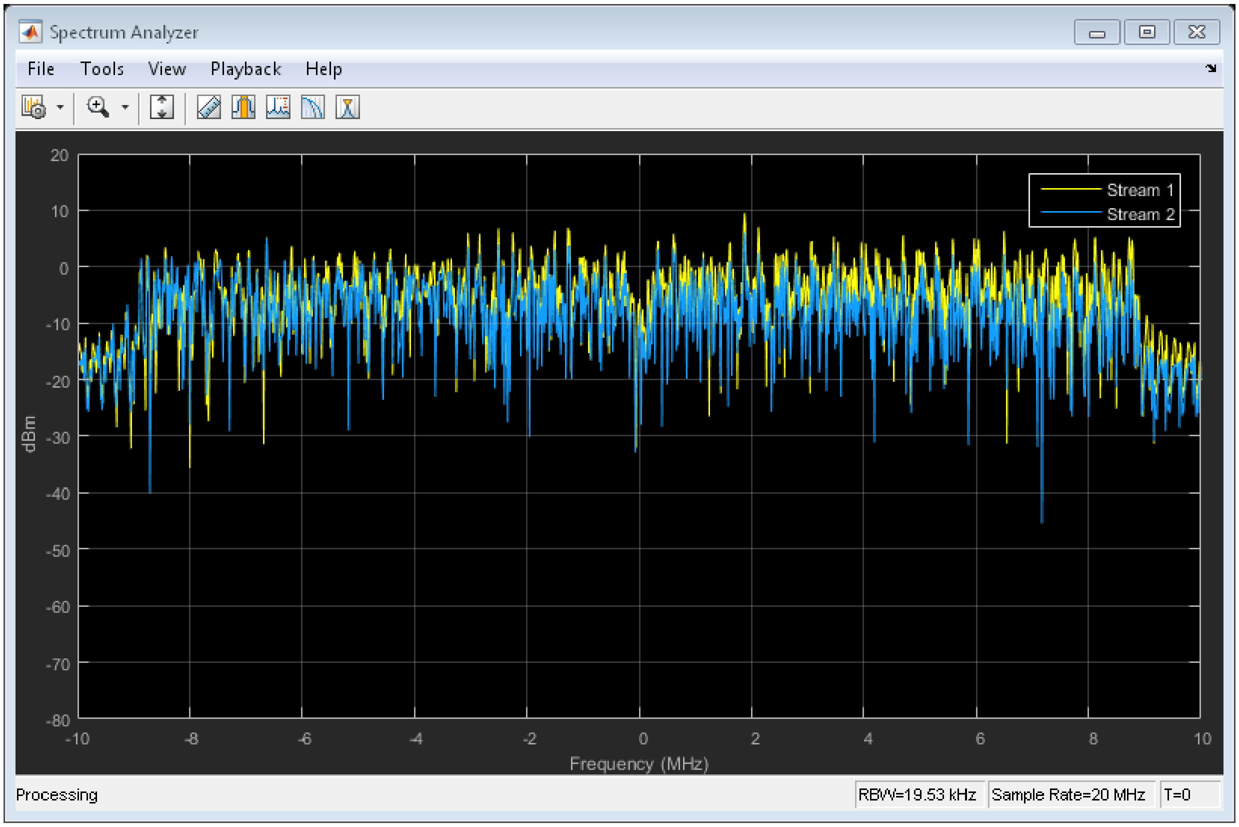
```
chan = wlanTGnChannel('SampleRate',20e6, ...
    'NumTransmitAntennas',4, ...
    'NumReceiveAntennas',2, ...
    'LargeScaleFadingEffect','None');
```

Pass the transmit waveform through the channel.

```
rxSig = step(chan,txSig);
```

Display the spectrum of the two received space-time streams.

```
sa = dsp.SpectrumAnalyzer('SampleRate',20e6, ...
    'ShowLegend',true, ...
    'ChannelNames',{'Stream 1','Stream 2'});
step(sa,rxSig)
```



Recover HT Data from 2x2 MIMO Channel

Transmit an HT-LTF and an HT data field through a noisy 2x2 MIMO channel. Demodulate the received HT-LTF to estimate the channel coefficients. Recover the HT data and determine the number of bit errors.

Set the channel bandwidth and corresponding sample rate.

```
bw = 'CBW40';
fs = 40e6;
```

Create HT-LTF and HT data fields having two transmit antennas and two space-time streams.

```
cfg = wlanHTConfig('ChannelBandwidth',bw, ...
    'NumTransmitAntennas',2,'NumSpaceTimeStreams',2);
txPSDU = randi([0 1],8*cfg.PSDULength,1);
txLTF = wlanHTLTF(cfg);
txDataSig = wlanHTData(txPSDU,cfg);
```

Create a 2x2 MIMO TGn channel with path loss and shadowing enabled.

```
tgn = wlanTGnChannel('SampleRate',fs, ...
    'NumTransmitAntennas',2,'NumReceiveAntennas',2, ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
```

Create an AWGN channel for a 40 MHz channel with a 9 dB noise figure. The noise variance, `nVar`, is equal to *kTBF*, where *k* is Boltzmann's constant, *T* is the ambient temperature of 290 K, *B* is the bandwidth (sample rate), and *F* is the receiver noise figure.

```
nVar = 10^((-228.6 + 10*log10(290) + 10*log10(fs) + 9)/10);
chAWGN = comm.AWGNChannel('NoiseMethod','Variance','Variance',nVar);
```

Pass the signals through the channel.

```
rxLTF = step(chAWGN,step(tgn,txLTF));
rxDataSig = step(chAWGN,step(tgn,txDataSig));
```

Demodulate the HT-LTF. Use the demodulated signal to estimate the channel coefficients.

```
dLTF = wlanHTLTFDemodulate(rxLTF,cfg);
chEst = wlanHTLTFChannelEstimate(dLTF,cfg);
```

Recover the data and determine the number of bit errors.

```
rxPSDU = wlanHTDataRecover(rxDataSig,chEst,nVar,cfg);
numErr = biterr(txPSDU,rxPSDU)
```

```
numErr =

     0
```

Algorithms

The 802.11n channel object uses a filtered Gaussian noise model in which the path delays, powers, angular spread, angles of arrival, and angles of departure are determined empirically. The specific modeling approach is described in [1].

Multipath Parameters

The channel is modeled as several clusters each of which represents an independent propagation path between the transmitter and the receiver. A cluster is composed of subpaths or taps which share angular spreads, angles of arrival, and angles of departure. Delay and power level vary from tap to tap. Within the TGn model, clusters comprise 1–7 taps. The cluster parameters for cluster 1 of Model B are shown in the table.

Parameter	Tap				
	1	2	3	4	5
Delay (ns)	0	10	20	30	40
Power (dB)	0	–5.4	–10.8	–16.2	–21.7
Angle of arrival (°)	4.3	4.3	4.3	4.3	4.3
Receiver angular spread (°)	14.4	14.4	14.4	14.4	14.4
Angle of departure (°)	225.1	225.1	225.1	225.1	225.1
Transmitter angular spread (°)	14.4	14.4	14.4	14.4	14.4

For each model, the first tap has a line-of-sight (LOS) between the transmitter and receiver, whereas all other taps are non-line-of-sight (NLOS). As a result, the first tap exhibits Rician behavior, while the others exhibit Rayleigh behavior. The Rician K-factor is the ratio between the power in the first tap and the power in the other taps. A large K-factor indicates a strong, LOS component.

The angles of arrival and departure for each cluster are randomly selected from a uniform distribution over $[0, 2\pi]$. These angles are independent of each other and are fixed for all channel realizations. By fixing the values, the transmit and receive correlation matrices are computed only once. Angular spread values were indirectly determined from empirical data and fall within the 20° to 40° range.

Path Loss and Shadowing

The path loss exponent and the standard deviation of the shadow fading loss characterize each model. The two parameters are depend on the presence of a line-of-sight between the transmitter and receiver. For paths with a transmitter-to-receiver distance, *d*, less that the breakpoint distance, d_{BP}, the LOS parameters apply. For $d > d_{BP}$, the NLOS parameters apply. The table summarizes the path loss and shadow fading parameters.

Parameter	**Model**					
	A	**B**	**C**	**D**	**E**	**F**
Breakpoint distance, d_{BP} (m)	5	5	5	10	20	30
Path loss exponent for $d \le d_{BP}$	2	2	2	2	2	2
Path loss exponent for $d > d_{BP}$	3.5	3.5	3.5	3.5	3.5	3.5
Shadow fading σ (dB) for $d \le d_{BP}$	3	3	3	3	3	3
Shadow fading σ (dB) for $d > d_{BP}$	4	4	5	5	6	6

Doppler Effects

In indoor environments, the transmitter and receiver are stationary, and Doppler effects arise from people moving between them. The TGn model employs a bell-shaped Doppler spectrum in which the environmental speed, v_0, is 1.2 km/hr. The Doppler spread, f_d, is calculated as $f_d = v_0/\lambda$, where λ is the carrier wavelength.

In addition to basic Doppler effects resulting from environmental motion, fluorescent lights introduce signal fading at twice the power line frequency. The effects show up as frequency-selective amplitude modulation. The effect is included in models D and E. To disable this effect, set the `FluorescentEffects` property to `false`.

References

[1] Erceg, V., L. Schumacher, P. Kyritsi, et al. *TGn Channel Models*. Version 4. IEEE 802.11-03/940r4, May 2004.

[2] Kermoal, J. P., L. Schumacher, K. I. Pedersen, P. E. Mogensen, and F. Frederiksen, “A Stochastic MIMO Radio Channel Model with Experimental Validation”. *IEEE Journal on Selected Areas in Communications.*, Vol. 20, No. 6, August 2002, pp. 1211–1226.

See Also

wlanTGacChannel | comm.MIMOChannel

Introduced in R2015b

clone

System object: wlanTGnChannel

Create `wlanTGnChannel` object with same property values

Syntax

```
C = clone(H)
```

Description

`C = clone(H)` creates a `wlanTGnChannel` object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object with uninitialized states.

The clone method creates an instance of an object. The property values, but not internal states, are copied into the new instance of the object.

info

System object: wlanTGnChannel

Characteristic information about TGn Channel

Syntax

```
S = info(OBJ)
```

Description

`S = info(OBJ)` returns a structure, `S`, containing characteristic information about the wlanTGnChannel object, `OBJ`. The list summarizes the information contained in `S`.

- `ChannelFilterDelay`: Filter delay introduced by the implementation (samples)
- `PathDelays`: Delay of each of the discrete paths (seconds)
- `AveragePathGains`: Average gain of each of the discrete paths (dB)
- `Pathloss`: Path loss between the transmitter and receiver (dB).

isLocked

System object: wlanTGnChannel

Locked status for input attributes and nontunable properties

Syntax

```
TF = isLocked(H)
```

Description

`TF = isLocked(H)` returns the locked status, `TF` of the `wlanTGnChannel` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal initialization the first time the `step` method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. After locking, the `isLocked` method returns a `true` value.

release

System object: wlanTGnChannel

Allow property value and input characteristics changes

Syntax

```
release(H)
```

Description

`release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note: You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

reset

System object: wlanTGnChannel

Reset states of the `wlanTGnChannel` object

Syntax

```
reset(H)
```

Description

`reset(H)` resets the states of the `wlanTGnChannel` object, `H`.

If you set the `RandomStream` property of `H` to `Global stream`, the `reset` method only resets the filters. If you set `RandomStream` to `mt19937ar with seed`, the `reset` method not only resets the filters but also reinitializes the random number stream to the value of the `Seed` property.

step

System object: wlanTGnChannel

Filter signal through 802.11n multipath fading channel

Syntax

```
Y = step(TGN,X)
[Y,PATHGAINS] = step(TGN,X)
```

Description

`Y = step(TGN,X)` filters input signal `X` through 802.11n (TGn) fading channel `TGN` and returns the result in `Y`. The input `X` can be a double-precision data type scalar, vector, or 2-D matrix with real or complex values. `X` is of size N_s-by-N_t, where N_s represents the number of samples and N_t represents the number of transmit antennas. `Y` is the output signal of size N_s-by-N_r, where N_r represents the number of receive antennas. `Y` is of double-precision data type with complex values.

`[Y,PATHGAINS] = step(TGN,X)` returns a complex N_s-by-N_p-by-N_t-by-N_r matrix `PATHGAINS` for the TGn channel System object, `TGN`. N_p is the number of paths in the channel.

Note: `TGN` specifies the System object on which to run this `step` method.

The object performs an initialization the first time the `step` method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Examples

Transmit HT Waveform Through TGn Channel

Generate an HT waveform and pass it through a TGn SISO channel. Display the spectrum of the resultant signal.

Set the channel bandwidth and the corresponding sample rate.

```
bw = 'CBW40';
fs = 40e6;
```

Generate an HT waveform for a 40 MHz channel.

```
cfg = wlanHTConfig('ChannelBandwidth',bw);
wgc = wlanGeneratorConfig;
txSig = wlanWaveformGenerator(randi([0 1],1000,1),cfg,wgc);
```

Create a TGn SISO channel with path loss and shadowing enabled.

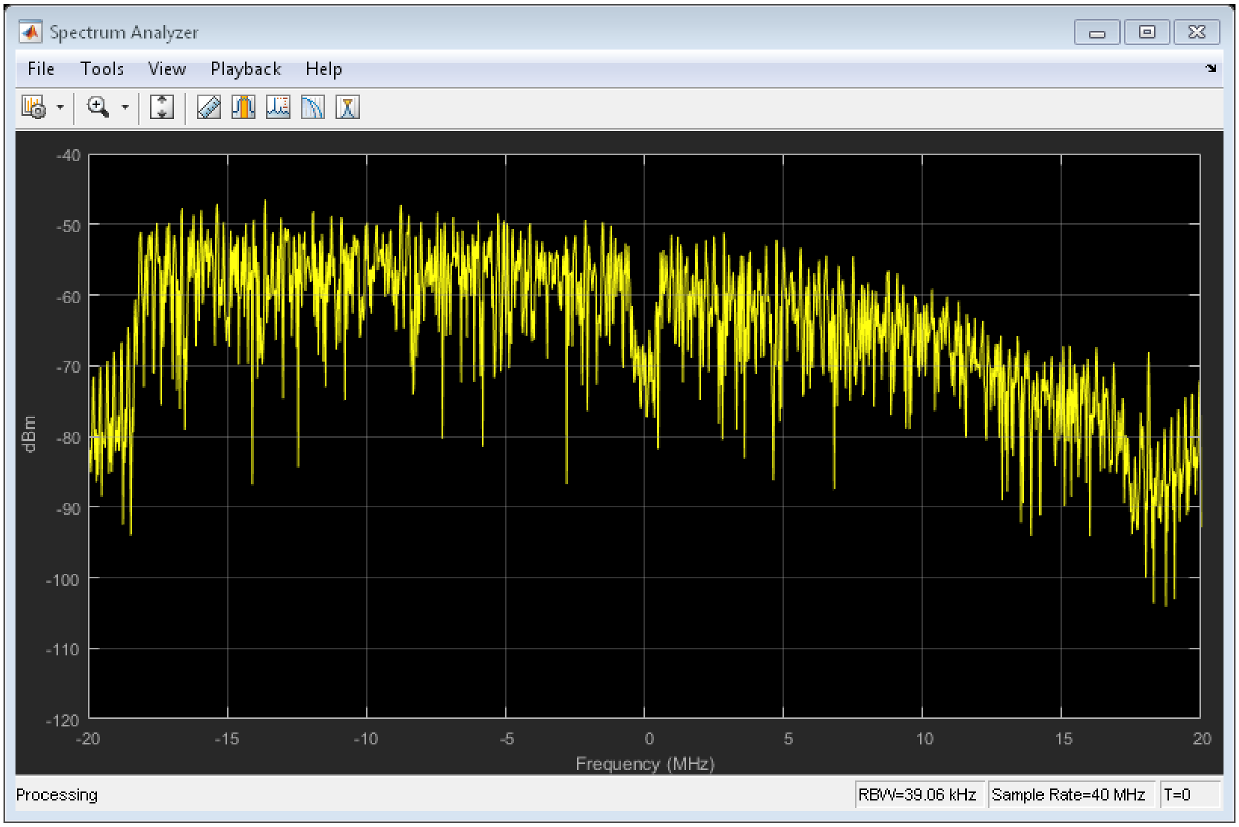
```
chan = wlanTGnChannel('SampleRate',fs, ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
```

Pass the HT waveform through the channel.

```
rxSig = step(chan,txSig);
```

Plot the spectrum of the received waveform.

```
sa = dsp.SpectrumAnalyzer('SampleRate',fs,'YLimits',[-120 -40]);
step(sa,rxSig)
```



Because path loss and shadowing are enabled, the mean received power across the spectrum is approximately -60 dBm.